1. Introducción

WayFinder es una innovadora aplicación móvil diseñada para revolucionar la forma en que las personas con discapacidad o movilidad reducida navegan por entornos urbanos. En un mundo donde la accesibilidad sigue siendo un desafío significativo, WayFinder se presenta como una solución tecnológica que ayuda a sus usuarios, proporcionándoles la información necesaria para moverse con confianza y autonomía en las ciudades.

1.1. Visión y Misión

La visión de WayFinder es crear ciudades más inclusivas y accesibles para todos. Nuestra misión es proporcionar una herramienta colaborativa que permita a las personas compartir y acceder a información en tiempo real sobre la accesibilidad de lugares, rutas y servicios en su entorno urbano.

1.2. Características Principales

WayFinder se distingue por sus características clave:

- **Mapa Interactivo:** Visualización geolocalizada de puntos de interés accesibles.
- **Sistema de Calificación Colaborativo:** Los usuarios pueden evaluar y comentar sobre la accesibilidad de diferentes lugares.
- **Alertas en Tiempo Real:** Información actualizada sobre obstáculos temporales que puedan afectar la movilidad.
- **Planificación de Rutas Accesibles:** Algoritmos inteligentes que sugieren las rutas más adecuadas según las necesidades específicas del usuario.

1.3. Impacto Social

WayFinder no es solo una aplicación; es un movimiento hacia la inclusión. Al fomentar la participación activa de la comunidad en la identificación y evaluación

de espacios accesibles, WayFinder contribuye a:

- Aumentar la conciencia sobre la importancia de la accesibilidad universal.
- Proporcionar datos valiosos a urbanistas y responsables de políticas públicas.
- Empoderar a las personas con discapacidad para que participen más plenamente en la vida urbana.
- Crear una comunidad comprometida con la construcción de ciudades más inclusivas.

1.4. Tecnología e Innovación

Basada en tecnologías de geolocalización de última generación y aprovechando el poder del crowdsourcing, WayFinder representa un salto cualitativo en las aplicaciones de movilidad urbana. La plataforma utiliza aprendizaje automático para mejorar continuamente la precisión de sus recomendaciones y se integra con otras tecnologías asistivas para ofrecer una experiencia de usuario fluida y accesible.

Con WayFinder, nos esforzamos por hacer que cada viaje sea una experiencia sin barreras, contribuyendo a un futuro donde la accesibilidad universal sea la norma, no la excepción.

2. Desarrollo

2.1. Requerimientos

El desarrollo exitoso de WayFinder depende de una clara definición y comprensión de sus requerimientos. Estos requerimientos abarcan tanto las funcionalidades específicas que la aplicación debe ofrecer como las características no funcionales que aseguran su calidad, rendimiento y usabilidad.

A continuación, se presenta una tabla detallada de los requerimientos funcionales y no funcionales de WayFinder:

Tabla 1: Requerimientos Funcionales de WayFinder

ID	Categoría	Descripción	Prioridad
RF1	Registro y Autenticación	Los usuarios deben poder registrarse e iniciar sesión en la aplicación	Alta
RF2	Geolocalización	La app debe mostrar un mapa interactivo con puntos de interés accesibles	Alta
RF3	Búsqueda	Los usuarios deben poder buscar lugares específicos y filtrar por tipo de accesibilidad	Alta
RF4	Evaluación	Los usuarios deben poder calificar y comentar sobre la accesibilidad de los lugares	Alta
RF5	Rutas	La app debe ofrecer planificación de rutas accesibles	Alta
RF6	Alertas	Los usuarios deben poder recibir y crear alertas sobre obstáculos temporales	Media
RF7	Perfil de Usuario	Los usuarios deben poder personalizar su perfil con sus necesidades específicas de accesibilidad	Media
RF8	Integración de Voz	La app debe permitir la navegación y el uso mediante comandos de voz	Media
RF9	Gamificación	Implementar un sistema de puntos o insignias para incentivar la participación	Baja
RF10	Recursos Educativos	Incluir una sección con información sobre accesibilidad y derechos	Baja

Tabla 2: Requerimientos No Funcionales de WayFinder

ID	Categoría	Descripción	Prioridad
RNF1	Rendimiento	La app debe cargar el mapa y los puntos de interés en menos de 3 segundos	Alta
RNF2	Usabilidad	La interfaz debe ser intuitiva y accesible, cumpliendo con las pautas WCAG 2.1	Alta
RNF3	Seguridad	Implementar cifrado de extremo a extremo para la protección de datos personales	Alta
RNF4	Disponibilidad	La app debe estar disponible el 99.9 % del tiempo	Alta
RNF5	Escalabilidad	La arquitectura debe soportar hasta 1 millón de usuarios concurrentes	Media
RNF6	Compatibilidad	La app debe funcionar en iOS 12+ y Android 8+	Alta
RNF7	Internacional	La app debe soportar múltiples idiomas y formatos regionales	Media
RNF8	Mantenibilidad	El código debe estar bien documentado y seguir principios SOLID	Media
RNF9	Backup	Realizar copias de seguridad automáticas diarias de todos los datos	Alta
RNF10	Actualizaciones	La app debe permitir actualizaciones automáticas sin interrupción del servicio	Media

2.2. Viabilidad

Considerando las funcionalidades propuestas para WayFinder, es fundamental evaluar si estas son técnica y operativamente viables, identificando los aspectos clave que contribuyen a su implementación exitosa. Este análisis permite determinar si el proyecto puede ejecutarse según lo planeado o si es necesario modificar o reestructurar algunas funcionalidades. Todo esto tiene como objetivo garantizar el éxito en las etapas posteriores del ciclo de vida del aplicativo y anticipar los retos que deberán ser abordados durante su desarrollo.

Funcionalidad	Mapa Interactivo de Accesibilidad
Descripción	Mapa en tiempo real con puntos de interés accesibles (rampas, baños adaptados, transporte público, etc.), con capacidad de filtrado por necesidades.
Viabilidad	Alta
Factores Técnicos	**Integración con la API de Google Maps**: Permite agregar marcadores personalizados y datos en tiempo real. Soporta filtros y visualización de datos en capas. **Desarrollo con React Native**: Facilita la implementación en dispositivos iOS y Android, utilizando bibliotecas de mapas como react-native-maps.
Retos	**Mantener actualizada la base de datos**: Requiere una fuente confiable inicial o depender de datos colaborativos desde el inicio. **Filtrado por necesidades**: Puede necesitar una estructura de datos compleja, pero es manejable con un diseño adecuado.
Recomendaciones	**Base de datos sólida**: Asegurar una base inicial mediante colaboración con organizaciones gubernamentales o privadas relacionadas con accesibilidad.

Funcionalidad	Planificación de Rutas Accesibles
Descripción	Creación de rutas en tiempo real considerando obstáculos (obras, bloqueos, etc.) y notificando afectaciones.
Viabilidad	Moderada
Factores Técnicos	**API de Google Maps**: Permite planificación de rutas y alertas en tiempo real (ej., tráfico, obras), pero no incluye detalles específicos para personas con discapacidad (ej., si una acera tiene una rampa). **Microservicios RESTful**: Permiten agregar datos adicionales a las rutas, como obstáculos y puntos accesibles, pero esto puede requerir integración con múltiples fuentes de datos.
Retos	**Obtener datos en tiempo real sobre obras y bloqueos**: Puede ser complejo dependiendo de la región y no siempre están disponibles en APIs públicas. **Personalización de rutas para accesibilidad**: Requiere adaptar las rutas para necesidades específicas de accesibilidad, lo que puede ser complejo sobre la API estándar de Google Maps.
Recomendaciones	Comenzar con rutas básicas accesibles basadas en puntos existentes (ej., rampas) y ampliar gradualmente con datos colaborativos y acuerdos con proveedores de datos municipales.

Funcionalidad	Comunidad Colaborativa
Descripción	Permitir que los usuarios añadan puntos de interés, evalúen la accesibilidad y contribuyan a actualizar los datos.
Viabilidad	Alta
Factores Técnicos	**Backend robusto (Node.js)**: Facilita la implementación de la funcionalidad de comunidad, gestionando los datos de usuarios y validaciones automáticas. **Base de datos (Firebase o MongoDB)**: Permite almacenar las contribuciones de los usuarios y gestionar datos de accesibilidad.
Retos	**Fiabilidad de los datos**: La calidad de los datos ingresados por los usuarios puede ser un problema sin mecanismos de validación. **Motivación de los usuarios**: Requiere incentivos para fomentar la participación activa, lo cual puede lograrse mediante recompensas o gamificación.
Recomendaciones	Establecer un sistema de moderación inicial y fomentar una comunidad activa a largo plazo a través de incentivos y participación continua.

2.3. Alcance

WayFinder es una aplicación móvil diseñada para promover la movilidad urbana y la accesibilidad de personas con discapacidad o movilidad reducida a través de mapas colaborativos que permiten identificar, valorar y compartir información sobre lugares, rutas y servicios accesibles. El proyecto incluye el desarrollo de una interfaz intuitiva y accesible, gestión de usuarios mediante registro e inicio de sesión, visualización de datos en tiempo real en mapas interactivos y almacenamiento centralizado de información confiable sobre puntos accesibles para promover la cooperación en la comunidad local. La aplicación también busca fomentar la colaboración entre personas en situaciones similares, permitiéndoles compartir experiencias, sugerencias y actualizaciones que fortalezcan la comunidad. A nivel tecnológico, se priorizará la comunicación fluida en tiempo real para garantizar que las actualizaciones relacionadas con la accesibilidad y movilidad estén siempre disponibles y reflejen el estado actual de los lugares y rutas. La aplicación se centrará en garantizar la privacidad y seguridad de los datos del usuario, inicialmente se admitirá como una aplicación nativa para Android y iOS. No incluye funciones avanzadas como la integración de IoT o la certificación oficial de accesibilidad.

2.3.1. MVP: Producto Mínimo Viable

El MVP de WayFinder proporcionará mapas interactivos que permitirán a los usuarios identificar puntos clave de accesibilidad, como rampas, ascensores y baños accesibles. Estos puntos estarán ubicados geográficamente y contendrán calificaciones cooperativas y descripciones proporcionadas por otros usuarios. Además, se habilitará una función para buscar rutas disponibles entre dos ubicaciones específicas. Los usuarios podrán utilizar filtros para personalizar los resultados, como priorizar rampas sobre ascensores o elegir la ruta más corta. Para fomentar la participación activa, se integrará un sistema básico de registro de usuarios, que permitirá a los usuarios verificar sus cuentas por correo electrónico o Google y contribuir a un mapa colaborativo. Otra característica importante son las actualizaciones instantáneas, que garantizan que las contribuciones de los usuarios aparezcan instantáneamente en el sistema, promoviendo una experiencia de colaboración dinámica. El MVP concluye con una fase de prueba interna para verificar el correcto funcionamiento de la herramienta y lanzar una versión beta en ciudades piloto, como Ciudad de México, adecuada a su diversidad y necesidades

específicas.

2.3.2. Objetivos Principales del MVP

El objetivo principal del MVP es validar la propuesta de valor de WayFinder y garantizar que los usuarios encuentren útil y útil el mapa de accesibilidad y el buscador de rutas. Otro objetivo clave es recopilar comentarios tempranos para identificar posibles problemas técnicos y de diseño y áreas de mejora. Esto permitirá realizar correcciones antes de que se extienda la solicitud. Finalmente, el MVP buscará construir una base tecnológica escalable, enfocándose en crear una arquitectura que permita la inclusión de funcionalidad avanzada en la próxima iteración.

2.3.3. Medidas de éxito del MVP

El éxito del MVP se medirá en función de tres métricas clave. En primer lugar, será importante la adopción activa entre los usuarios beta. Nuestro objetivo es que las funciones estén disponibles para la mayor cantidad de personas durante el período de prueba. En segundo lugar, los comentarios positivos de los usuarios sobre la usabilidad del mapa interactivo y la precisión de los datos nos permitirán medir el valor percibido del producto. Finalmente, identificar y documentar errores técnicos o problemas de diseño se considerará un éxito, ya que permitirá priorizar las mejoras en futuras iteraciones, asegurando que el proyecto continúe evolucionando.

2.4. Aplicación

El desarrollo de la aplicación móvil necesitará ciertas características propias de los requerimientos descritos anteriormente. Por ende, el esqueleto base data desde el desarrollo de Wireframes hasta el maquetado final, el cual se puede apreciar con mayor detalle desde el modelo de Figma: `https://www.figma.com/proto/53ZXxUR69UrIWW1WW7wzNY/Proyecto-Compu-M%C3%B3vil?node-id=2705-6442&t=ka0oUsRg9Qa5yngZ-1`.

2.4.1. Wireframes

Los wireframes de WayFinder están diseñados para proporcionar navegación intuitiva con un enfoque en la accesibilidad y una experiencia colaborativa que le permite identificar y compartir información instantáneamente sobre ubicaciones accesibles. Fueron diseñados teniendo en mente los principios funcionales de la aplicación para garantizar que ésta se desempeñe de manera óptima ne los dispositivos móviles.

Descripción del flujo de navegación

De acuerdo con el planteamiento original de WayFinder se espera contar con una navegación intuitiva amigable a través de las características de la pantalla sin saturar al usuario con demasiados elementos en pantalla. A continuación se detalla el flujo a través del nombre de cada pantalla en un orden jerárquico de aparición, acceso y despliegue de las mismas a través de un diagrama de flujo.

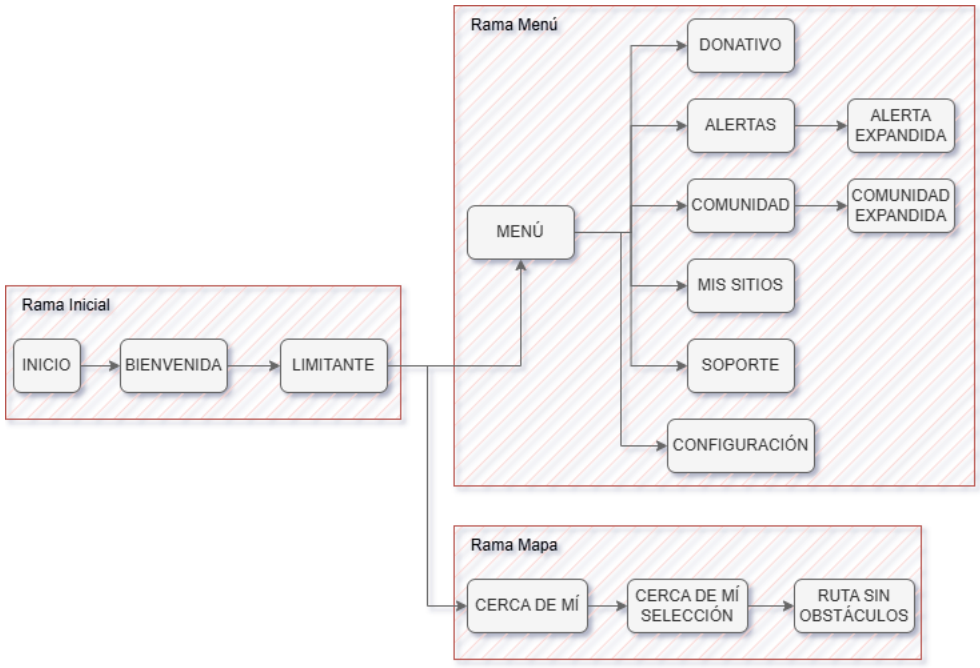

Figura 1: Flujo de despliegue de los WireFrames

La organización de los wireframes de WayFinder se encuentra dispuesta en tres apartados o **Ramas**, en donde se agrupan funcionalidades específicas de diferentes

WireFrames. Lo anterior, permite mejorar la fluidez y experiencia de usuario, pues las consultas a bases da datos o APIs, propias y de terceros se llevan a cabo una única ocasión en la para cada rama. Si bien, esta solución permite obtener un flujo mayor en cuanto a la navegación, también representa una mayor demanda de conectividad robusta, limitante asumida, dado que la aplicación tiene como objetivo ambientes urbanos donde la conectividad no es una limitante.

Las ramas implementadas son las siguientes:

- Inicial: Contiene los WireFrames: Inicio, Bienvenida y Limitante. Su función es acceder a la cuenta del usuario y desplegar las preferencias respecto a las limitantes.

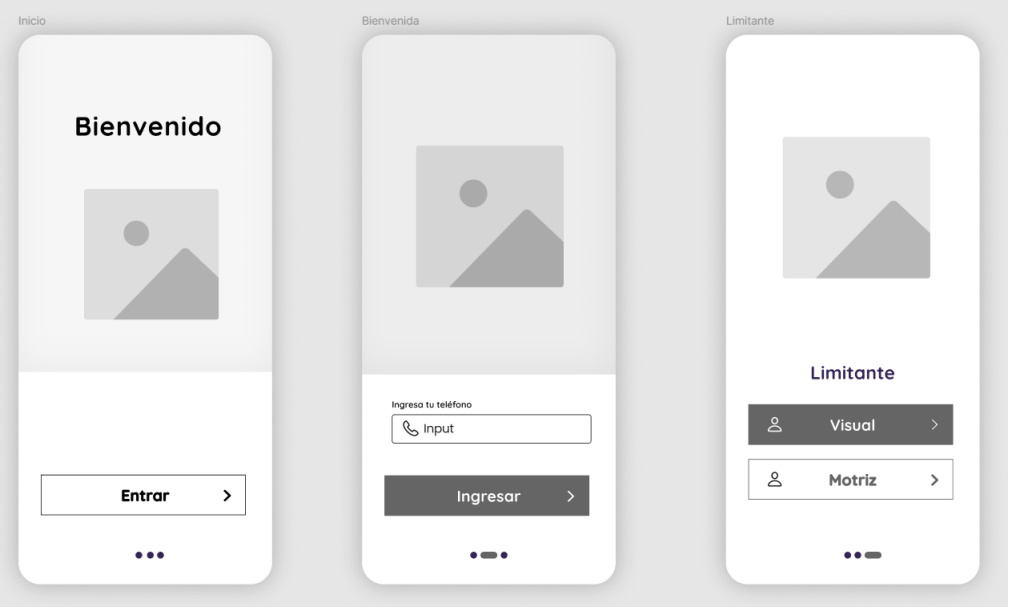

Figura 2: Wireframes: Inicio - Binvenida - Limitante.

- Menú: Contiene los WireFrames: Menú, Mis sitios, Donativo, Alertas, Alerta Expandido, Comunidad, Comunidad Expandida, Soporte y Configuración. Su funcionalidad es agrupar todas las características propias de la aplicación, además de proporcionar una ruta accesible a las dichas características.

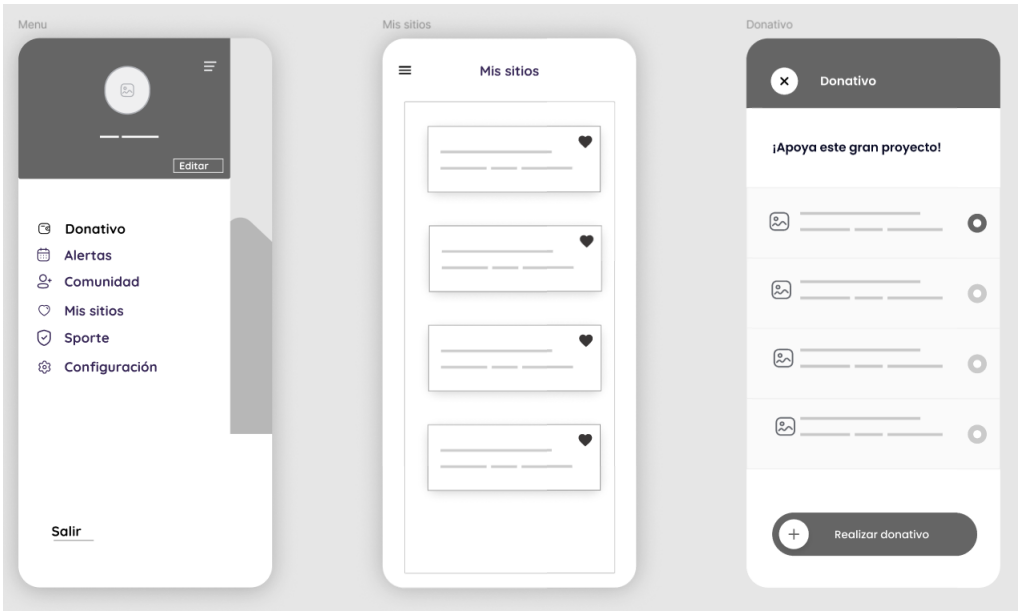

Figura 3: Wireframes: Menú - Mis Sitios - Donativo.

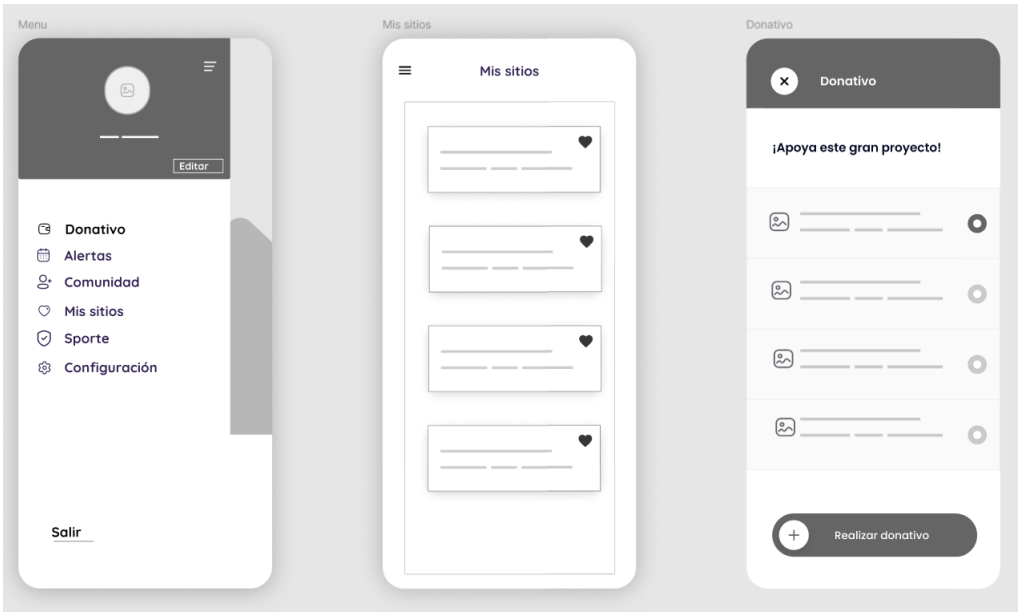

Figura 4: Wireframes: Menú - Mis Sitios - Donativo.

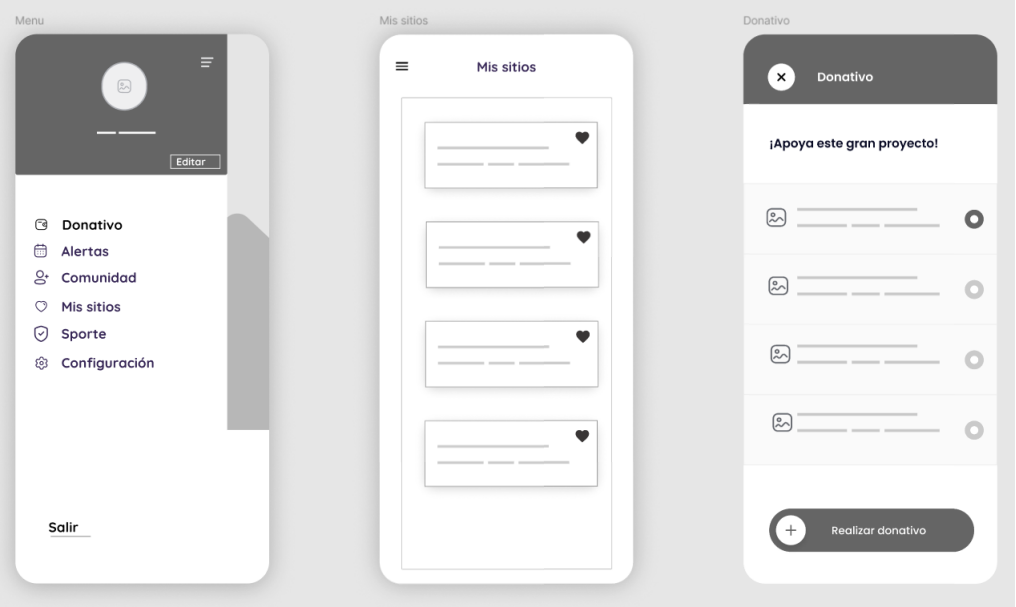

Figura 5: Wireframes: Menú - Mis Sitios - Donativo.

- Mapa: Contiene los WireFrames:

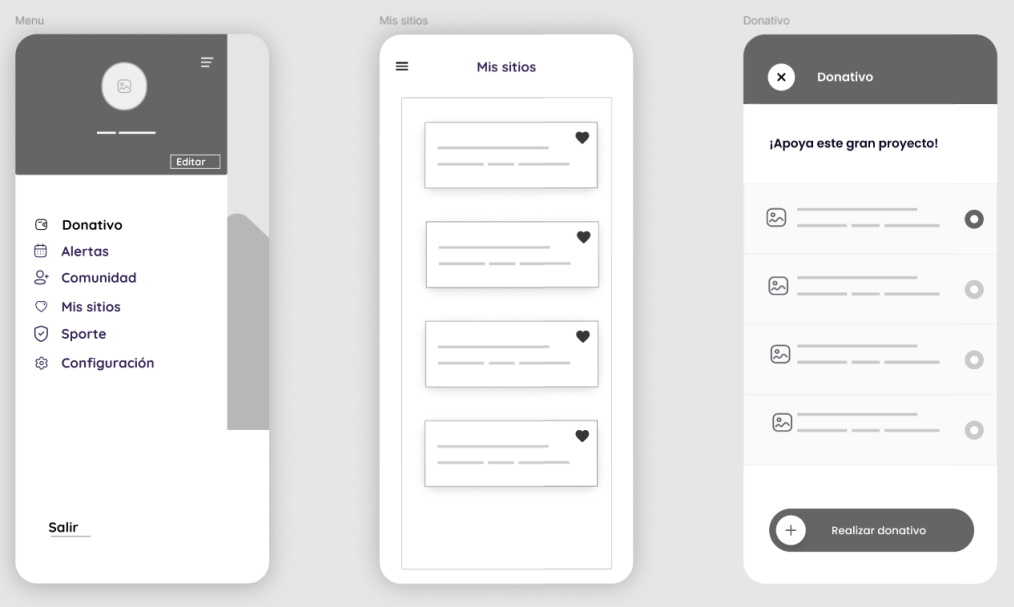

Figura 6: Wireframes: Inicio - Binvenida - Limitante.

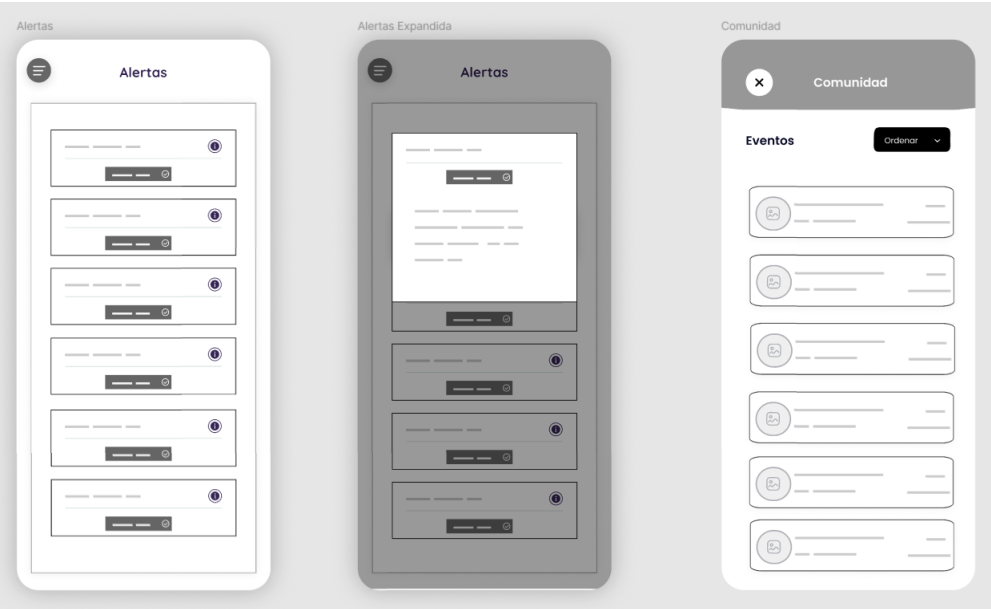

Figura 7: Wireframes: Inicio - Binvenida - Limitante.

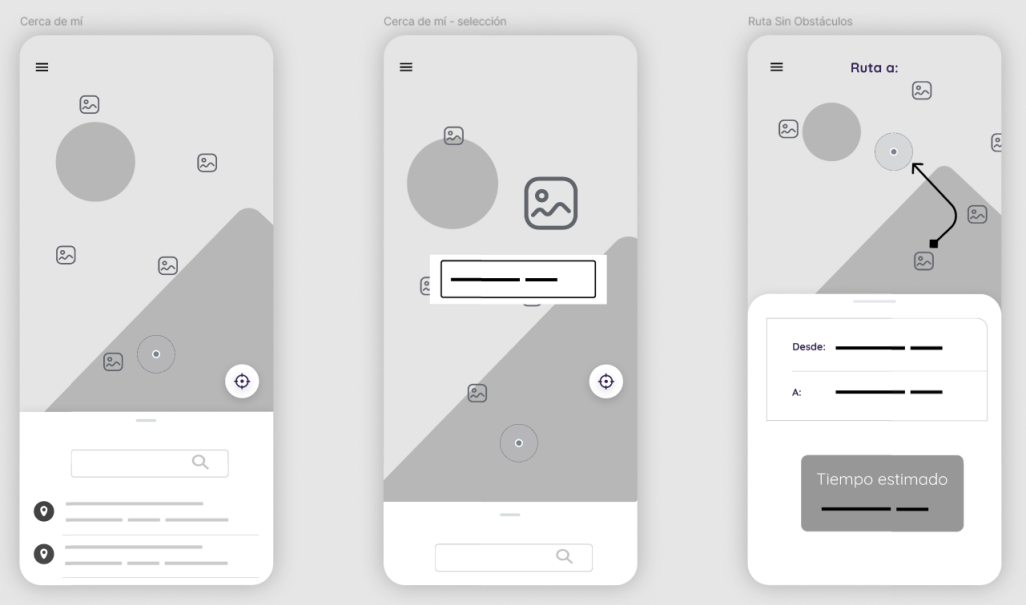

Figura 8: Wireframes: Inicio - Binvenida - Limitante.

La documentación de la pantalla Çerca de mí", misma que dirge a la pantalla Ruta Sin Obstáculos"se desglosa en 3 WireFrames con las siguientes caraterísitcas:

Nombre del WireFrame:Cerca de mí, Cerca de mí - selección Funcionalidad principal: Desplegar el mapa de la ubicación del usuario con indicadores de rutas de fácil acceso de acuerdo con la información proporcionada anteriormente sobre el tipo de limitante seleccionada. Mostrar detalles de las rutas desplegadas en el mapa y, tras la previa selección de la ruta adecuada para el usuario, desplegar indicaciones con diferentes funciones de accesibilidad para guiar al usuario a través de la ruta seleccionada. Información mostrada: Mapa de un radio de la ubicación del usuario, funciones de accesibilidad oficiales y registradas por la comunidad Datos: Tipo de datos: Vigencia del dato: Origen de los datos: Operaciones esperadas:

Tipos de datos	**Vigencia de los datos**	**Origen de los datos**	**Operaciones esperadas**
Datos de usuarios registrados	Permanente	Formulario de registro en la app	Creación, modificación y eliminación de cuentas
Puntos accesibles geolocalizados	Actualización diaria	Contribuciones de usuarios y bases de datos preexistentes	Inserción, actualización y consulta de puntos en el mapa
Rutas accesibles entre ubicaciones	Temporal (dinámico por consultas)	Algoritmos de geolocalización y contribuciones de usuarios	Generación dinámica, consulta y filtrado
Retroalimentación de usuarios	Permanente	Encuestas y reportes dentro de la app	Almacenamiento y análisis para mejoras futuras

Tabla 3: Clasificación de los datos y operaciones esperadas

Navegación e interacciones especiales:

Consideraciones de usabilidad:

Datos y servicios requeridos

El análisis detallado de los datos y servicios que necesitará la aplicación desde las primeras etapas de diseño es esencial para garantizar una implementación eficiente y coherente. Este proceso permite identificar los flujos de información y servicios requeridos, lo que facilita la integración de cada componente del sistema. Dado que cada wireframe está diseñado para manejar diferentes tipos de información, será necesario definir qué datos y servicios se utilizarán en cada caso. Esto incluye la especificación de si los datos son de consulta, registro, actualización o borrado, y qué tipo de servicio externo o interno será requerido para cada uno. Por ejemplo, algunos wireframes estarán enfocados en la interacción con servicios que devuelvan datos en formatos como JSON, mientras que otros podrían manejar objetos más complejos. Esta variedad en los tipos de manejo de datos hace que sea crucial detallar, desde el principio, los servicios necesarios, sus características, costos y las formas de autenticación para asegurar una integración adecuada y sin contratiempos con las APIs o servicios en la nube.

Pantalla: Alertas	**Pantalla: Alertas Expandida**	**Pantalla: Comunidad**	**Pantalla: Comunidad Expandida**
Tipos de datos:	**Tipos de datos:**	**Tipos de datos:**	**Tipos de datos:**
Datos requeridos: ID del usuario para identificar las alertas relacionadas	Datos requeridos: ID de la alerta.	Parámetros opcionales: Filtros u opciones de orden (query params como ?orden=fecha).	Datos requeridos: ID del evento.
Respuesta esperada: JSON con una lista de alertas.	Respuesta esperada: JSON con los detalles completos de la alerta.	Respuesta esperada: JSON con la lista de eventos, incluyendo metadatos para soporte de imágenes.	Respuesta esperada: JSON con los detalles completos del evento.
Servicios necesarios:	**Servicios necesarios:**	**Servicios necesarios:**	**Servicios necesarios:**
Consulta: Servicio para recuperar las alertas desde el backend (API GET /alertas).	Consulta: Servicio para obtener la información detallada de una alerta específica (API GET /alertas/id).	Consulta: Servicio para recuperar la lista de eventos desde el backend (API GET /eventos).	Consulta: Servicio para recuperar información detallada de un evento (API GET /eventos/id).

Pantalla: Limitante	**Pantalla: Mis sitios**	**Pantalla: Donativo**	**Pantalla: Cerca de mí**
Tipos de datos:	**Tipos de datos:**	**Tipos de datos:**	**Tipos de datos:**
Registro: La limitante seleccionada	Consulta: Recuperar sitios guardados en la base de datos del usuario.	Consulta: Recuperar opciones de donativo desde el servidor.	Consulta: Solicitar ubicaciones cercanas basadas en GPS o palabras clave.
Servicios necesarios:	**Servicios necesarios:**	**Servicios necesarios:**	**Servicios necesarios:**
Servicio de almacenamiento o actualización de datos: API para guardar o actualizar el estado del usuario con su limitante.	- Servicio de datos de sitios: API para listar sitios guardados	- Servicio de datos de donativos: API para listar opciones.	- Servicio de ubicación: API para devolver ubicaciones según coordenadas.
		- Servicio de pago: Integración con un procesador como Stripe o PayPal.	- Servicio de búsqueda: API para filtrar ubicaciones.

Almacenamiento

Como se ha observado, cada wireframe manejará diferentes tipos de datos, los cuales pueden ser consultados o adquiridos desde otros servicios. Sin embargo, no todos los datos serán almacenados localmente en el dispositivo. Es importante identificar si será necesario realizar almacenamiento local, qué datos se almacenarán y por qué se tomará esa decisión. Esta evaluación es crucial para justificar la necesidad de almacenamiento local en cada wireframe, ya que depende de factores como la naturaleza de los datos, la frecuencia de acceso y la importancia de mantener la persistencia de la información. Al entender estos aspectos, se puede asegurar que el almacenamiento local se implementará solo cuando sea verdaderamente necesario y adecuado para el flujo de trabajo del aplicativo.

Wireframe	**Tipo de almacenamiento**	**Datos consultados**	**Datos almacenados**	**Motivo**
Pantalla: Mis sitios	La pantalla Mis sitios solo consultará los datos desde una base de datos o API. No se almacenarán datos localmente en el dispositivo.	Los datos mostrados son sitios que el usuario ha guardado previamente en su perfil. Estos sitios se obtienen en tiempo real desde un servicio remoto (API).	El almacenamiento solo se realiza si el usuario agrega un nuevo sitio o modifica uno existente, lo cual implica la consulta y actualización de datos a través de una API.	Mantener los datos en consulta asegura que siempre se presenten datos actualizados y evita la redundancia en el almacenamiento local, a menos que se realicen cambios.
Pantalla: Donativo	En la pantalla Donativo, el flujo de información es de solo consulta y actualización de datos, sin necesidad de almacenamiento local. Los usuarios pueden realizar donativos en tiempo real a través de una conexión con el servidor.	La pantalla Donativo muestra una lista de los posibles proyectos o causas a las que el usuario puede hacer donativos. Esta información es consultada a través de una API que obtiene los datos actualizados del servidor.	Los datos que se almacenarán son las acciones de donativo que el usuario haya realizado. Cuando un usuario realice un donativo, los detalles como el monto y la causa del donativo se registrarán en el servidor a través de una API.	Al igual que en la pantalla Mis sitios, el almacenamiento local no es necesario porque los datos consultados (información sobre proyectos y causas) siempre deben estar actualizados.

Cerca de Mí	En la pantalla Cerca de Mí, se utiliza almacenamiento local temporal para almacenar los datos consultados. Cuando el usuario realiza una consulta, los resultados de la búsqueda se almacenan en un arreglo local dentro de la aplicación.	Los datos consultados son lugares cercanos basados en la ubicación del usuario, obtenidos a través de una consulta a una API o servicio de ubicación. Los resultados incluyen lugares como restaurantes, tiendas o servicios de accesibilidad cercanos.	Los datos almacenados son los resultados de la consulta, como nombres, direcciones, coordenadas y otros detalles de los lugares cercanos. Estos datos se guardan en un arreglo dentro de la aplicación.	El almacenamiento local en un arreglo permite mejorar la experiencia del usuario al evitar la necesidad de hacer repetidas consultas a la API. Además, facilita la navegación entre resultados y reduce el uso de la red, mejorando la velocidad y eficiencia de la aplicación.
Alertas	Esta pantalla realiza únicamente consultas a la base de datos o API para mostrar la lista de alertas. No se almacenan datos localmente en el dispositivo, ya que las alertas son dinámicas y pueden actualizarse frecuentemente desde el servidor.	Los datos mostrados incluyen el título, descripción breve, estado de lectura (leída/no leída), y otros posibles metadatos relacionados con las alertas. Estos datos se obtienen en tiempo real desde un servicio remoto (API).	No se almacenan datos localmente en esta pantalla. Si el usuario realiza una acción (por ejemplo, marcar una alerta como leída), dicha acción será registrada directamente en el servidor a través de la API.	Mantener los datos en consulta asegura que las alertas presentadas siempre estén actualizadas y evita redundancia o desactualización en el almacenamiento local.
Alertas Expandida	Esta pantalla consulta los datos de la alerta seleccionada desde una base de datos o API. No se almacenan datos localmente, ya que los detalles de la alerta son específicos y pueden variar.	Incluyen el título completo, descripción ampliada, estado de lectura, y cualquier otra información adicional sobre la alerta.	No se almacenan datos localmente. Si se requiere marcar como leída o realizar una acción específica, esta operación será registrada directamente en el servidor mediante una API.	Evitar redundancia y asegurar que la información presentada sea la más reciente proveniente del servidor.

Comunidad	La pantalla Comunidad solo consulta datos desde la base de datos o API. No se almacenan datos localmente, ya que la lista de eventos puede cambiar dependiendo de actualizaciones o filtros aplicados por el usuario.	Incluyen el título del evento, descripción breve, fecha, hora, y cualquier imagen asociada.	No se almacenan datos localmente. Si el usuario aplica filtros u opciones de ordenamiento, estas se manejan temporalmente en la memoria de la aplicación pero no se almacenan de forma persistente.	La consulta en tiempo real asegura que los eventos sean relevantes y estén actualizados. El almacenamiento local no es necesario porque los eventos son dinámicos y dependen de filtros que pueden cambiar.
Comunidad Expandida	La pantalla consulta los datos de un evento específico desde la base de datos o API. No se almacenan datos localmente.	Incluyen el título completo, descripción ampliada, fecha, hora, imagen asociada, y cualquier detalle adicional relacionado con el evento.	No se almacenan datos localmente en esta pantalla. La información se mantiene visible hasta que el usuario decida regresar o consultar otro evento.	Asegurar que los detalles mostrados sean los más actualizados disponibles en el servidor, evitando inconsistencias entre el almacenamiento local y remoto.

2.4.2. Usabilidad

En los wireframes descritos para la aplicación, se utilizarán varios sensores del teléfono para mejorar la experiencia del usuario y facilitar la navegación en función de sus limitaciones. Uno de los sensores clave que se empleará es el GPS, para obtener la ubicación en tiempo real del usuario. Este sensor es fundamental para la pantalla Çerca de mí", ya que permite mostrar un mapa interactivo con la localización precisa del usuario y detectar las rutas accesibles cercanas. El GPS proporcionará datos de latitud y longitud que, al ser procesados por la aplicación, se traducirán en una representación gráfica en el mapa y en las rutas sugeridas.

Además, en la pantalla Çerca de mí - Selecciónz Ruta sin obstáculos", el sensor de acelerómetro será utilizado para detectar el movimiento del usuario mientras recorre la ruta. Esto permitirá ofrecer instrucciones de navegación paso a paso y, en el caso de que el usuario se desvíe de la ruta, se activarán alertas o se actualizarán las direcciones, manteniendo la guía precisa en todo momento. El acelerómetro puede ayudar a detectar cambios de dirección, giros, o la velocidad del usuario para proporcionar una experiencia más adaptada a su progreso en la ruta.

En cuanto a la conexión de los sensores, la aplicación utilizará las APIs disponibles en el sistema operativo del teléfono, como Core Location para el GPS y Core Motion para el acelerómetro. Estos sensores estarán conectados de manera continua a la aplicación, permitiendo obtener y procesar la información en tiempo real, lo cual es crucial para actualizar la visualización del mapa, las rutas y las instrucciones de manera precisa y dinámica.

El uso de estos sensores no solo mejorará la experiencia de navegación, sino que también garantizará que los usuarios reciban orientación efectiva y oportuna según su ubicación y progreso, permitiendo una experiencia accesible y fluida en todo momento.

2.4.3. Maquetado

Durante el desarrollo de los wireframes, se observa la necesidad constante de ir un paso más allá de solo espacios para las pantallas presentes; es por eso que el maquetado final de WayFinder busca tener un diseño estético y fluido al mostrarse las diferentes pantallas.

El diseño final de las pantallas cumple una representación clara de la experiencia de usuario esperada, donde se ilustra como se organiza y se despliega la información para los casos que así se consideren. Dicho enfoque busca entender claramente la navegación final con sus respectivas funcionalidades que son cruciales para el desarrollo de la aplicación.

A continuación se muestran las pantallas realizadas a partir de los Wireframes creados. No obstante, en el presente documento solo se explicarán algunas de estas pantallas las cuales son indispensables en las funcionalidades de la aplicación. Así mismo, el resumen de las pantallas de manera más clara se puede observar en el siguiente link: Figma.

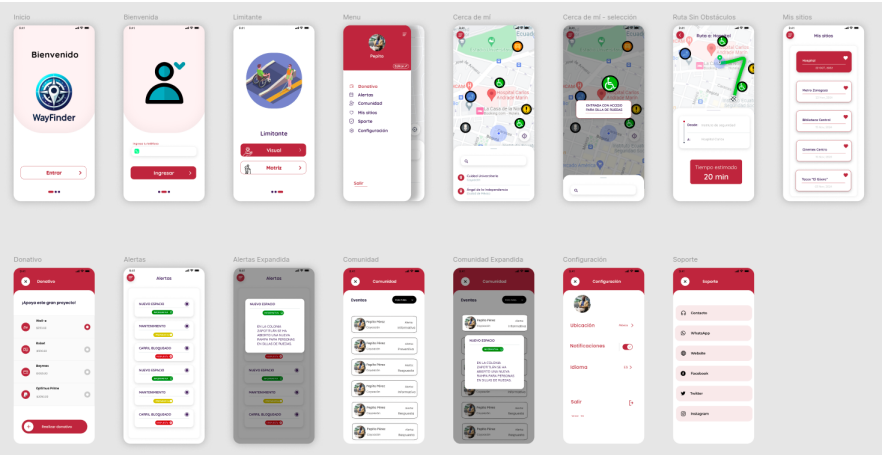

Figura 9: Vista general de las pantallas tras su maquetado.

Uno de los puntos clave de WayFinder y por la cual se desarrolló es la interacción con un mapa interactivo en tiempo real con zonas y eventos que requieran la atención. Tal como se elaboró y describió en el Wireframe, el acceso y búsqueda de este tipo de acontecimientos tiene como objetivo: ayudar a las personas.

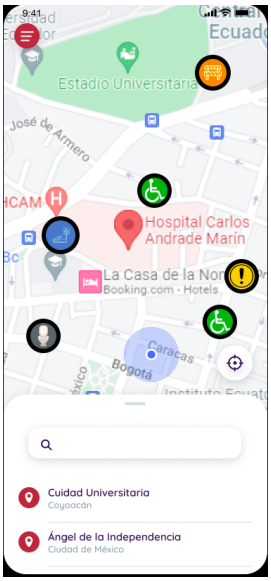

Figura 10: Pantalla *Cerca de mí.*

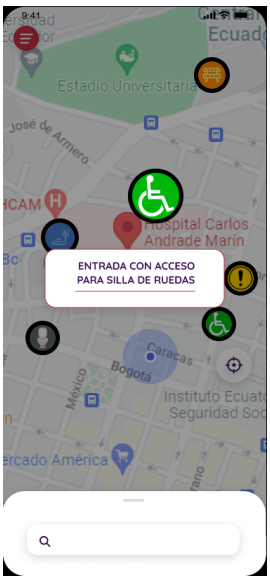

Figura 11: Selección de un evento.

De manera similar, se pueden tomar rutas con zonas de acceso universal al mismo tiempo de evitar obstáculos propios de la zona.

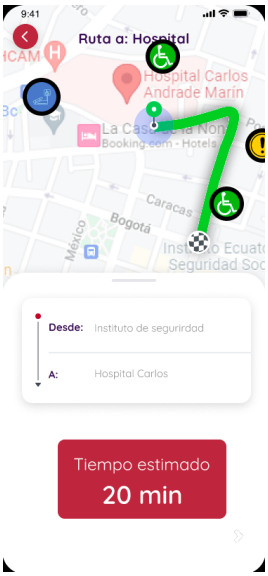

Figura 12: Ruta óptima generada.

Y como elemento adicional, la comunidad participante es clave para complementar aquellas alertas que WayFinder no detecte o sean recientes.

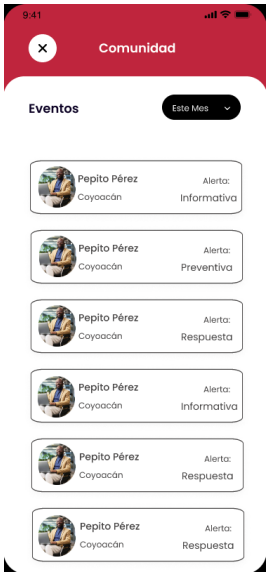

Figura 13: Pantalla *Comunidad*.

Figura 14: Selección de alerta.

Como se observa en las pantallas anteriores, los datos introducidos son a modo de ejemplo, simulando un caso real cuando se ha interactuado con elementos propios de la aplicación, buscando en todo momento la comodidad de los usuarios.

2.5. Programación

WayFinder será desarrollada utilizando un enfoque híbrido con el fin de garantizar el desarrollo multiplataforma buscando optimizar recursos. A continuación se detallan los lenguajes y tecnologías necesarias para cumplir dicho objetivo.

Lenguajes de programación

- Frontend: JavaScript con el framework React Native, ya que es excelente para el despliegue tanto en Android como en iOS.
- Backend: Node.js para el manejo de las APIs necesarias al momento de manipular la lógica de la aplicación.
- Base de datos: MongoDB por ser no relacional y lograr manipular datos pertenecientes a los mapas que se usarán.

En cuanto a las herramientas necesarias, se requiere en primer lugar el dominio de las mismas para lograr el cometido, pasando desde la idea hasta la culminación-mantenimiento de la aplicación. Por ello, se enlistan aquellas que servirán como complemento:

- Figma: diseño de interfaces UI/UX, además genera el prototipado de la apliación y plantea la línea base de la misma.
- Firebase: autenticación de usuarios por medio de Google y así notificar a los usuarios desde las notificaciones en sus dispositivos.
- Postman: para el manejo de APIs y su manipulación más sencilla; al mismo tiempo de lograr documentarlas.
- GitHub: controlador de versiones y a la vez de permitir la colaboración de desarrolladores.

Como se observa, no solo se requieren lenguajes de programación sino que también otros elementos propios del sistema. Adicionalmente, pese a desarrollar conscientemente la aplicación será necesario validar algunos permisos dentro de archivos de configuración para lograr la correcta publicación en tiendas oficiales.

Al hablar de Android (Google Play Store) se requiere solicitar permisos de ubicación en tiempo real para la manipulación del mapa. Así mismo, en *AndroidManifest.xml* es obligatorio agregar políticas de privacidad de los datos al manejar información personal.

Por su parte en iOS (App Store) la privacidad de los datos resalta como en el caso anterior. De la misma manera, dentro de *Info.plist* es necesario considerar permisos de ubicación para que logre ser publicada corectamente.

2.6. Tiempos de Desarrollo

Estimamos que el desarrollo de *WayFinder* puede llevarse a cabo en varias etapas para asegurar su correcta implementación y funcionalidad:

- **Primera versión (MVP):** El desarrollo del producto mínimo viable, que incluirá las funcionalidades básicas como la búsqueda de rutas accesibles y

el mapeo de puntos clave, está previsto para completarse en un periodo de 4 a 6 meses. Esta fase incluirá pruebas internas y un lanzamiento beta para un grupo reducido de usuarios en una ciudad piloto.

- **Versión mejorada:** Tras el feedback obtenido del lanzamiento inicial, se trabajará en una segunda versión que incluirá características adicionales como integración con otras aplicaciones de transporte y mejoras en la precisión de los datos de accesibilidad. Estimamos que esta segunda fase tomará aproximadamente otros 3 a 4 meses.
- **Expansión y escalabilidad:** Una vez consolidada la aplicación en su versión mejorada, se podrá planificar la expansión a nuevas ciudades, lo que implicará un periodo continuo de desarrollo y ajustes, en función de la retroalimentación de los usuarios y las necesidades locales. Este proceso de escalabilidad está previsto para ser iniciado a partir del primer año de su lanzamiento completo.