Universidad Politécnica
de Madrid

Escuela Técnica Superior de
Sistemas Informáticos

Máster Universitario En Desarrollo De Aplicaciones Y
Servicios Para Móviles

Trabajo Fin de Máster

Aplicación Móvil Geográfico: Arriving

Autor(a): Qi Ye

Tutor(a)s: Miguel Ángel Manso
Ramón Pablo Alcarria Garrido

Madrid, junio 2023

Este Trabajo Fin de Máster se ha depositado en la ETSI Sistemas Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Máster
Máster Universitario En Desarrollo De Aplicaciones Y Servicios Para Móviles
Título: Aplicación Móvil Geográfico: Arriving
Junio 2023

Autor(a): Qi Ye

Tutor(a)s:

Miguel Ángel Manso
Ingeniería Topográfica Y Cartografía
ETSI Topografía, Geodesia, Cartografía
Universidad Politécnica de Madrid

Ramón Pablo Alcarria Garrido
Ingeniería Topográfica Y Cartografía
ETSI Topografía, Geodesia, Cartografía
Universidad Politécnica de Madrid

Resumen

Hoy en día, con el ritmo acelerado de la sociedad, a menudo nos encontramos en la necesidad de anotar las tareas que debemos realizar durante el día haciendo uso de notas o utilizando aplicaciones de recordatorios.

En este proyecto, nos proponemos desarrollar una aplicación de recordatorios basada en geolocalización de forma que cuando el usuario se encuentre en el área que rodea una ubicación específica, recibirá una notificación sobre las tareas pendientes que debe realizar.

Para llevar a cabo este proyecto, aplicaremos los conocimientos adquiridos durante el máster, tales como Geoinformática, Desarrollo de Aplicaciones Android e Ingeniería de Software. Específicamente, utilizaremos técnicas de Geofencing y Geocodificación para trabajar con mapas, y aplicaremos técnicas de manejo de datos y desarrollo de aplicaciones móviles para la plataforma Android.

Abstract

Nowadays, with the fast pace of society, we often find ourselves needing to write down the tasks we need to accomplish during the day by making use of notes or using reminder applications.

In this project, we propose to develop a geolocation-based reminder application so that when the user is in the area surrounding a specific location, he/she will receive a notification about the pending tasks to be performed.

To carry out this project, we will apply the knowledge acquired during the master's program, such as Geoinformatics, Android Development, Software Engineering, and more. Specifically, we will utilize geofencing and geocoding techniques for mapping purposes, along with data management and mobile development techniques for the Android platform.

Tabla de contenidos

Tabla de ilustraciones

1 Introducción

El presente documento tiene como objetivo presentar los objetivos y el desarrollo de la aplicación, así como los trabajos futuros y las conclusiones del proyecto.

1.1 Objetivos y motivaciones

La mayoría de las aplicaciones de recordatorios requieren que se introduzca una fecha y hora para recordar una tarea, lo que limita la flexibilidad del usuario al tener que posponer la tarea si desea recibir una alarma nuevamente. En este proyecto, se busca evitar que la fecha sea una limitación para los recordatorios, brindando la libertad de realizar la tarea cuando el usuario se encuentre cerca de un lugar específico, recibiendo notificaciones sobre las tareas pendientes a medida que se acerque a dicho lugar.

La motivación principal del desarrollo de este proyecto es implementar una aplicación que pueda recordar las tareas al usuario al llegar a un lugar específico, evitando la necesidad de regresar al lugar por segunda vez.

Además, uno de los objetivos del proyecto es aplicar de manera práctica los conocimientos adquiridos durante el Máster, específicamente en el uso de tecnologías geográficas como el geofencing y el geocoding, así como en la codificación de una aplicación para dispositivos Android.

1.2 Requisitos y funcionalidades

Los requisitos y funcionalidades básicas de la aplicación son los siguientes:

1. **Creación de tareas y ubicaciones**: Permitir al usuario crear nuevas tareas y ubicaciones en la aplicación.
2. **Visualización del listado de tareas y ubicaciones**: Mostrar al usuario una lista de todas las tareas y ubicaciones guardadas en la aplicación.
3. **Visualización del mapa de ubicaciones**: Presentar al usuario un mapa donde pueda ver todas las ubicaciones guardadas.
4. **Notificaciones de tareas pendientes al llegar a una ubicación**: Emitir notificaciones al usuario cuando se acerque a una ubicación específica, recordándole las tareas pendientes asociadas a esa ubicación.
5. **Actualización de ubicaciones y envío de notificaciones en segundo plano**: Permitir la actualización de ubicaciones y el envío de notificaciones incluso cuando la aplicación se encuentra en segundo plano.

Además de estas funcionalidades básicas, se agregan las siguientes funcionalidades adicionales:

1. **Notificaciones a Telegram a través de un bot**: Proporcionar la opción de recibir y enviar mensajes definidos por el usuario en la aplicación, a través de *Telegram* mediante la integración de un bot.
2. **Creación y visualización de contactos de *Telegram***: Permitir al usuario crear y ver el listado de contactos agregados.

2 Estado de arte

En esta sección se realiza un análisis del mercado actual en relación con las aplicaciones de recordatorios.

2.1 Mercado

En la actualidad existen aplicaciones recordatorias que incluyen la opción de establecer recordatorios cuando el usuario entra o sale de una ubicación específica.

En nuestro proyecto, los recordatorios basados en la ubicación serán la función principal, mientras que los recordatorios basados en fechas podrían considerarse para futuros desarrollos.

Mediante la tecnología de geofencing, nuestra aplicación permitirá a los usuarios crear recordatorios que se activarán cuando entren o salgan de un área designada. Al eliminar la dependencia de fechas y horarios específicos, los usuarios pueden recibir recordatorios de sus tareas cuando se encuentren cerca de la ubicación relevante, aumentando las posibilidades de completar las tareas de manera eficiente.

2.1.1 Apple Reminders

La aplicación Reminders de Apple[8] incluye una funcionalidad que permite configurar alertas de tareas basadas en la ubicación. Esta aplicación, que viene preinstalada en los dispositivos iOS, ofrece a los usuarios la posibilidad de recibir notificaciones de tareas pendientes cuando alcanzan una ubicación específica o cuando abandonan esa ubicación.

La funcionalidad de recordatorios basados en ubicación en Apple Reminders se activa al establecer una ubicación como parte de la configuración de una tarea. Los usuarios pueden elegir entre dos opciones: entrada o salida. Cuando el usuario llega o sale de la zona definida, recibirá una notificación con la tarea asociada que debe realizar.

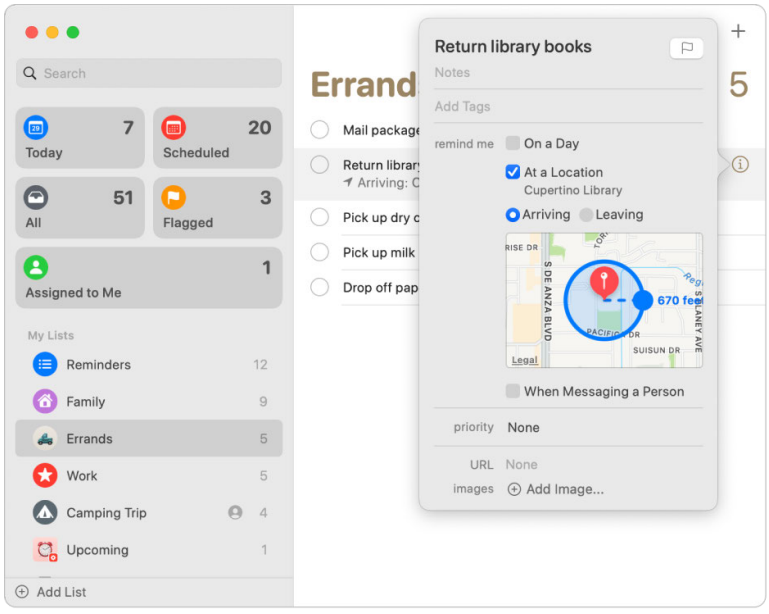


Figura 1: Apple Reminder – Ubicación

2.1.2 Todoist

Otra aplicación de recordatorios destacada es Todoist[9] . Se trata de una aplicación multiplataforma que permite a los usuarios gestionar y recordar sus tareas pendientes.

Si bien Todoist incluye la funcionalidad de agregar recordatorios basados en la ubicación, esta opción está disponible únicamente para usuarios con suscripción. Los usuarios que utilizan la versión gratuita de Todoist solo pueden configurar recordatorios basados en fecha y hora.

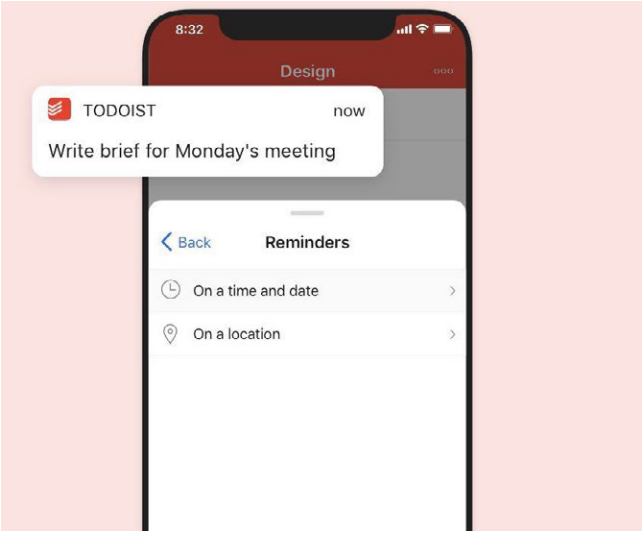


Figura 2: Todoist - Location

2.1.3 Google Keep

La aplicación Google Keep[10] se diferencia de las anteriores por no ser exclusivamente una aplicación de recordatorios, sino más bien una aplicación de notas y organización, permitiendo a los usuarios tomar notas rápidas, crear listas de tareas, capturar imágenes y grabar mensajes de voz.

Sin embargo, éste también ofrece la funcionalidad de establecer recordatorios, incluyendo tanto los basados en la fecha y hora como los basados en la ubicación.

Al ser una aplicación integrada en Google facilita la sincronización y el acceso a las notas y recordatorios en múltiples dispositivos.

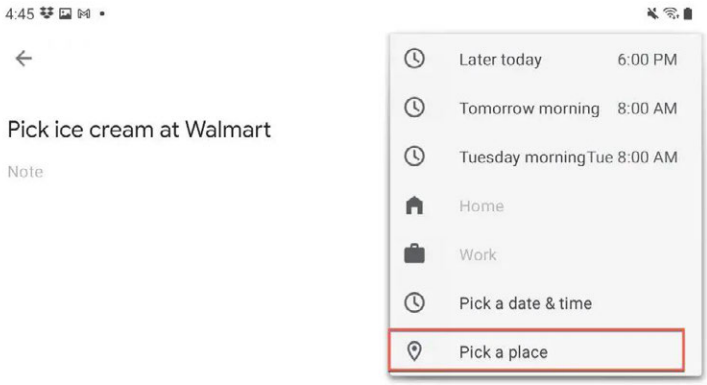


Figura 3: Google Keep - Location

3 Desarrollo

Este apartado se divide en dos secciones principales: diseño y maquetación, y código.

3.1 Entorno de trabajo

A continuación, se presenta una breve explicación de las tecnologías y técnicas utilizadas en el proyecto

3.1.1 Geofencing y geocoding

Este proyecto se centra en el aspecto clave de las ubicaciones, por lo que se utilizan las técnicas de geofencing y geocodificación para su implementación.

Geofencing es una técnica que permite establecer límites virtuales en el mapa, y en combinación con las ubicaciones proporcionadas por el GPS, permite capturar eventos de entrada y salida[1].

En el contexto de la aplicación de recordatorios basada en geolocalización, se utiliza el geofencing para delimitar las áreas de ubicaciones y activar las notificaciones correspondientes cuando el usuario se acerque o salga de una ubicación específica.

Figura 4: Geofences

Geocoding consiste en el proceso de convertir una dirección o descripción textual de un lugar en coordenadas geográficas (latitud y longitud)[2].

La técnica de geocodificación se emplea para traducir las direcciones introducidas por el usuario en la aplicación a ubicaciones geográficas precisas que se utilizarán para establecer las geovallas asociadas a las ubicaciones de las tareas.

Figura 5: Geocoding

Estas técnicas desempeñan un papel fundamental en el desarrollo de la aplicación de recordatorios basada en geolocalización ya que permiten la integración de la geolocalización y las notificaciones para la gestión de las tareas.

3.1.2 Telegram: bot

Un bot se trata de un programa automatizado diseñado para interactuar con usuarios a través de la plataforma de mensajería *Telegram*.

Los bots de *Telegram* pueden ser programados para realizar diversas tareas, como responder a comandos, enviar mensajes, proporcionar información, realizar acciones automatizadas, entre otras funciones[3].

En nuestro caso el bot se utiliza para obtener el identificador del chat y enviar / recibir los mensajes definidos por el usuario al llevar a una ubicación en concreto.

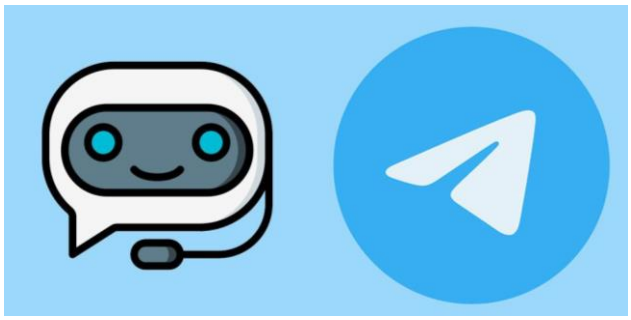


Figura 6: Bots Telegram

En *Telegram*, es posible crear bots de manera sencilla utilizando el bot de bots llamado *@BotFather*. Éste se trata de una herramienta proporcionada por *Telegram* que permite crear y administrar bots[4].

Para la creación del bot solo es necesario indicar un nombre que será el identificador único del bot, por lo que no puede ser igual que otro bot que ya exista.

En el momento que se haya creado correctamente el bot *@BotFather* devolverá el token de acceso necesario para interactuar con la API de Telegram y programar las funciones y respuestas deseadas para el bot.

Figura 7: BotFather

3.1.3 Node.js

Node.js es un entorno de ejecución para el servidor basado en Javascript ampliamente utilizado en el desarrollo backend de las aplicaciones debido a su simplicidad, facilidad de uso y eficiencia[5].

Es conocido por su capacidad de manejo de conexiones concurrentes y operación en tiempo real, permitiendo el manejo de múltiples solicitudes y eventos de manera eficiente y evitando bloqueos, lo que lo hace adecuado para aplicaciones que requieren una comunicación fluida y respuestas rápidas[6].

Figura 8: Node.js

Gracias a estas características, nos permite implementar un servidor que maneja los eventos de Telegram de manera eficiente (*véase apdo.3.2.5*).

3.1.4 Android Studio: Java

Android Studio es un entorno de desarrollo integrado (IDE) oficial de Google diseñado específicamente para el desarrollo de aplicaciones Android. Proporciona todas las herramientas y funcionalidades necesarias para crear, depurar y optimizar aplicaciones móviles compatibles con dispositivos Android[7].

Figura 9: Logo Android Studio

Este entorno ha facilitado el proceso de prueba de los requisitos y funcionalidades de la aplicación gracias a sus emuladores integrados. Estos emuladores permiten simular dispositivos Android en diferentes configuraciones de hardware y software, lo que facilita la verificación del correcto funcionamiento de la aplicación en diferentes entornos.

En cuanto al lenguaje de programación, en este proyecto se utiliza Java, que ha sido durante mucho tiempo el lenguaje de programación principal para el desarrollo de aplicaciones Android. Sin embargo, Kotlin se ha convertido en un lenguaje oficialmente compatible con Android y es altamente recomendado por Google, por lo que Android Studio ofrece herramientas de migración y

traducción automática para convertir el código Java existente a Kotlin, lo que facilita la transición entre ambos lenguajes.

En trabajos futuros, considerar la traducción del código a Kotlin podría aprovechar las ventajas que este lenguaje ofrece, como una sintaxis más concisa, nulabilidad segura y soporte mejorado para programación funcional.

3.2 Diseño y maquetación

Esta sección se centra en definir las diferentes pantallas de la aplicación, así como la disposición y funcionalidad de sus componentes.

3.2.1 Icono de la aplicación

El icono de la aplicación consiste en un tick dentro de un marcador de ubicación:

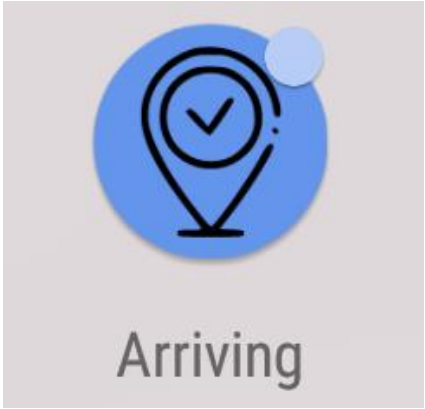


Figura 10: Icono de la aplicación

Este diseño se debe a que los objetivos de la aplicación son la finalización de las tareas (tick) y las notificaciones recordatorias basadas en la ubicación (marcador de ubicación).

3.2.2 Barra superior: Toolbar

La barra superior, también conocida como toolbar, es un componente comúnmente utilizado en las aplicaciones para dispositivos móviles y se encuentra ubicada en la parte superior de la pantalla.

Esta barra tiene como objetivo principal proporcionar acceso rápido a funcionalidades de la aplicación, así como la del mostrar un título que indica la sección o función actualmente activa.

3.2.2.1 Pantallas principales

La barra superior de las pantallas principales de la aplicación está diseñada de manera que incluye dos elementos principales: el nombre de la pantalla y un icono de información:

- **Nombre de la pantalla:** Proporciona una indicación visual al usuario sobre en qué parte de la aplicación se encuentra y qué puede esperar encontrar en esa sección específica
- **Icono de información:** Al hacer clic en este icono, el usuario puede acceder a un texto explicativo que proporciona detalles y descripciones sobre las funcionalidades específicas de la aplicación.

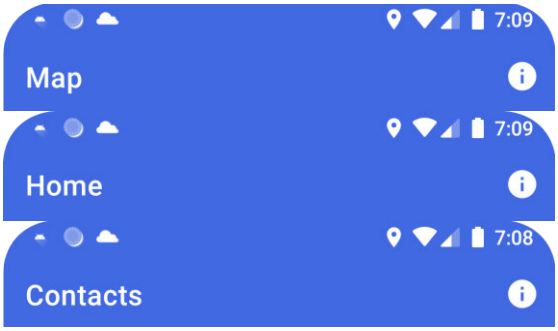


Figura 11: Toolbar Pantallas Principales

3.2.2.2 Pantallas auxiliares

En las pantallas auxiliares de la aplicación, la barra superior se compone de:

- **Icono de retroceso**: Representado por una flecha apuntando hacia la izquierda, permite al usuario regresar a la pantalla anteriormente visitada.
- **Título de la pantalla**: Proporciona una referencia rápida para que el usuario sepa en qué contexto se encuentra dentro de la aplicación.
- **Botón de acción**: El botón de acción es un elemento interactivo que se utiliza para realizar acciones específicas relacionadas con la pantalla actual. Dependiendo del contexto, este botón puede tener diferentes etiquetas, como "Guardar", "Añadir" y realizar acciones específicas de la pantalla en cuestión.

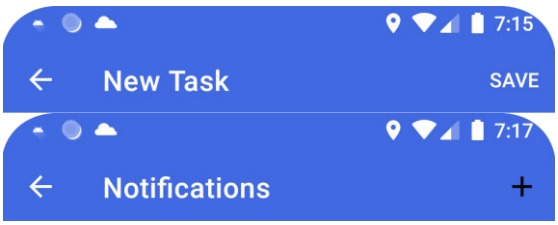


Figura 12: Toolbar Pantallas Auxiliares

3.2.3 Barra inferior: Navegación

La aplicación cuenta con una barra inferior de navegación que facilita el acceso y la transición entre las diferentes pantallas principales.

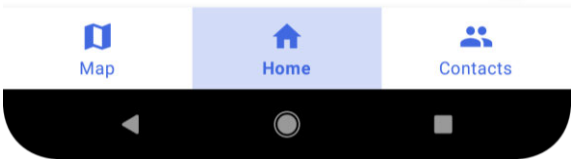


Figura 13: Barra de Navegación

3.2.3.1 Navegación

Desde los componentes de las pantallas principales se acceden a las pantallas secundarias diseñadas para realizar acciones tales como la creación y la edición.

En la siguiente figura se muestra un esquema sobre la navegación entre pantallas:

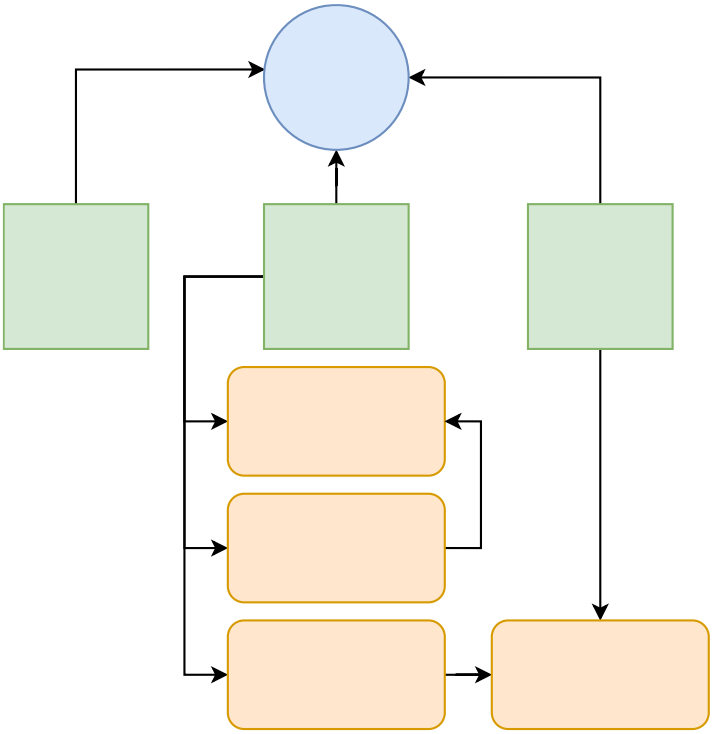


Figura 14: Esquema de Navegación

Como se muestra en la figura, la pantalla "Home" es el punto principal de acceso a todas las pantallas secundarias de la aplicación, proporcionando comodidad y agilidad al usuario, permitiéndole realizar sus tareas de manera más eficiente evitando la necesidad de navegar por diferentes secciones de la aplicación para acceder a las funcionalidades deseadas.

A modo resumen del esquema, las pantallas principales se tratan de:

- **Map**: Esta pantalla muestra de forma geográfica las ubicaciones guardadas en la aplicación.
- **Home**: Es la pantalla de inicio de la aplicación. Proporciona un listado de tareas y ubicaciones, donde el usuario puede ver de manera general sus ubicaciones y las tareas asociadas a ellas.
- **Contacts**: Esta pantalla muestra el listado de contactos agregados en la aplicación. Aquí el usuario puede gestionar los contactos y acceder a su información relacionada.

Y las pantallas auxiliares son:

- **Info**: Esta pantalla es accesible desde la barra superior de las pantallas principales. Proporciona una breve descripción de la aplicación y una explicación de la funcionalidad de Telegram. Su objetivo es brindar al usuario información adicional sobre la aplicación y su integración con Telegram
- **Nueva ubicación**: Permite al usuario crear y editar ubicaciones en la aplicación.
- **Nueva tarea**: Esta pantalla permite al usuario crear y editar tareas en la aplicación.

- **Nueva notificación**: Permite al usuario crear y editar notificaciones en la aplicación.
- **Nuevo contacto**: Esta pantalla permite al usuario crear y editar contactos en la aplicación

3.2.4 Pantalla Info

El objetivo de esta pantalla es describir los objetivos que persigue la aplicación, así como la de proporcionar las instrucciones sobre cómo se deben habilitar las notificaciones de Telegram.

Esto asegura que los usuarios comprendan la funcionalidad de la aplicación y puedan configurar las notificaciones correctamente para aprovechar al máximo las características ofrecidas.

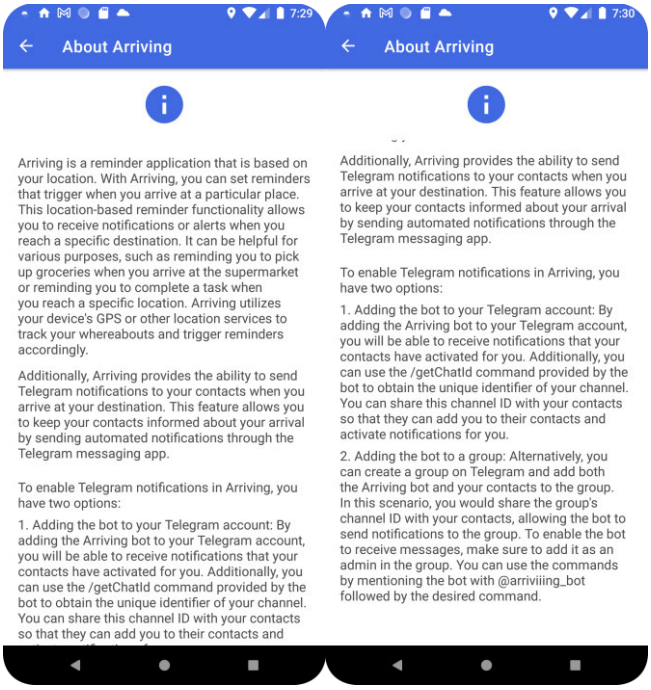


Figura 15: Pantalla de Información

3.2.5 Pantalla Home

El objetivo principal de esta pantalla es mostrar de manera clara y ordenada el listado de ubicaciones almacenadas, lo que permite al usuario acceder rápidamente a la información de cada ubicación y realizar acciones relacionadas, como editar o eliminar una ubicación existente, así como agregar nuevas ubicaciones y tareas.

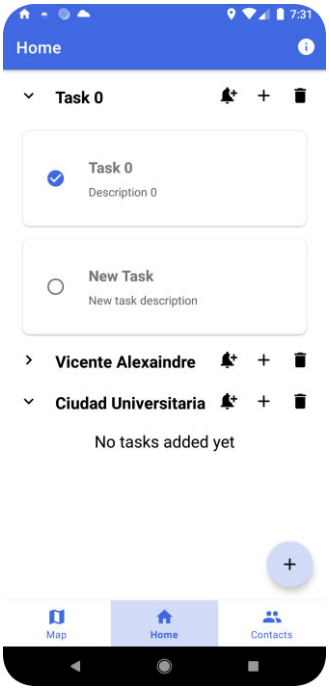


Figura 16: Pantalla Home

A continuación, se describen cada una de las partes que componen la pantalla.

3.2.5.1 Fragment Home

La pantalla Home se implementa utilizando un fragmento. Un fragmento es un módulo independiente y reutilizable que puede ser utilizado en diferentes actividades o en otras partes de la interfaz de usuario (*véase apdo. 3.2.1*).

A continuación, se describen los siguientes componentes que forman la vista.

3.2.5.1.1 Contenedor de ubicaciones

El contenedor de ubicaciones en la pantalla Home está compuesto por varios elementos, que se describen a continuación:

1. **Nombre de la ubicación**: Se trata del elemento identificador que permite la identificación de la ubicación. Al hacer clic en el nombre, se redirige a la pantalla de edición de la ubicación, donde se pueden realizar modificaciones (*véase apdo. 3.1.3.2.1*).

2. **Icono desplegable**: Este icono tiene la función de desplegar el contenedor y mostrar la lista de tareas pertenecientes a esta ubicación. Las tareas se presentan en forma de tarjetas, cada una con un checkbox que se puede marcar para indicar si la tarea ha sido completada.
   Al hacer clic en la tarjeta de una tarea, se redirige a la pantalla de tareas en modo edición, donde se pueden editar los detalles de la tarea.
   En caso de que no haya tareas añadidas para esta ubicación, se mostrará un mensaje indicando que no hay tareas disponibles.

3. **Icono añadir notificación**: Este icono tiene la función de abrir la pantalla de añadir mensajes a enviar o recibir en Telegram. Esta funcionalidad permite asociar notificaciones específicas a esta ubicación (*véase apdo. 3.1.3.1.3*)

4. **Icono "+"**: Este icono tiene la función de abrir la pantalla de creación de una nueva tarea para esta ubicación. Al hacer clic en este icono, se inicia el proceso de creación de una nueva tarea relacionada con esta ubicación (*véase apdo. 3.1.3.1.2*)

5. **Icono papelera**: Este icono tiene la función de eliminar la ubicación y todas las tareas que pertenecen a esta ubicación. Al hacer clic en este icono, se elimina la ubicación y sus tareas asociadas.

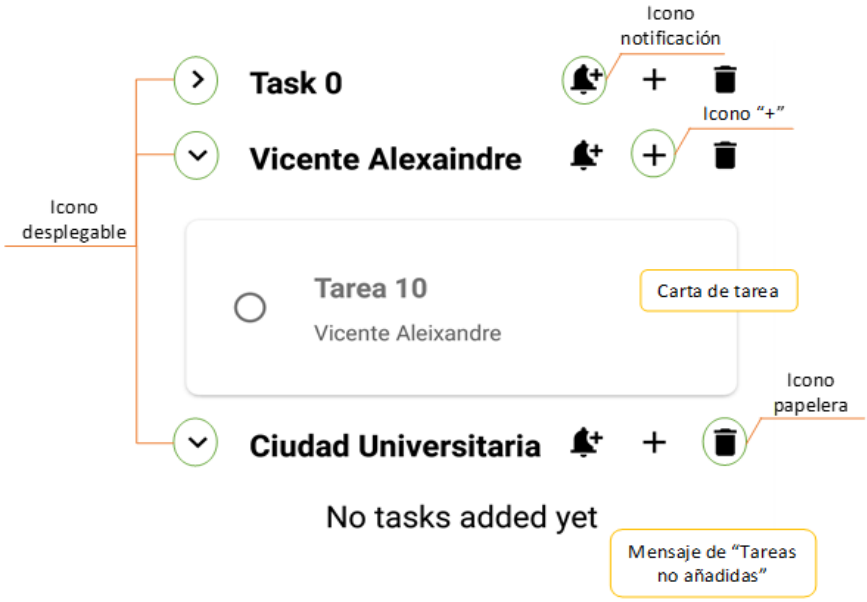


Figura 17: Contenedores de ubicaciones

3.2.5.1.2 Pantalla auxiliar: Tarea

La pantalla auxiliar de Tarea es fundamental en la aplicación, ya que permite al usuario crear y gestionar las tareas.

Para la creación de tareas, se solicita al usuario que ingrese un título y una descripción para identificar y describir la tarea respectivamente. Además, se le ofrece la opción de seleccionar la ubicación donde se llevará a cabo la tarea.

Al asociar una ubicación a la tarea, la aplicación puede emitir eventos cuando el usuario se acerque a esa ubicación específica. Esto se logra utilizando el GPS del dispositivo móvil. Cuando el usuario se encuentra cerca de la ubicación configurada para una tarea, la aplicación envía una notificación para recordarle que debe completar esa tarea.

Para facilitar la introducción de ubicaciones en la creación de tareas, la aplicación ofrece dos opciones:

1. **Nueva ubicación (pestaña NEW)**: Esta opción permite al usuario crear una nueva ubicación desde cero. Al seleccionar esta pestaña, se muestra una pantalla de ubicación en una vista minimizada (*véase apdo. 3.1.3.2.1*) con las mismas funcionalidades que ésta.
2. **Selección de ubicaciones (pestaña SELECT)**: Esta opción permite al usuario elegir una ubicación existente de una lista de ubicaciones previamente creadas y guardadas en la base de datos de la aplicación. Al seleccionar esta pestaña, se muestra un selector con el listado de ubicaciones disponibles.

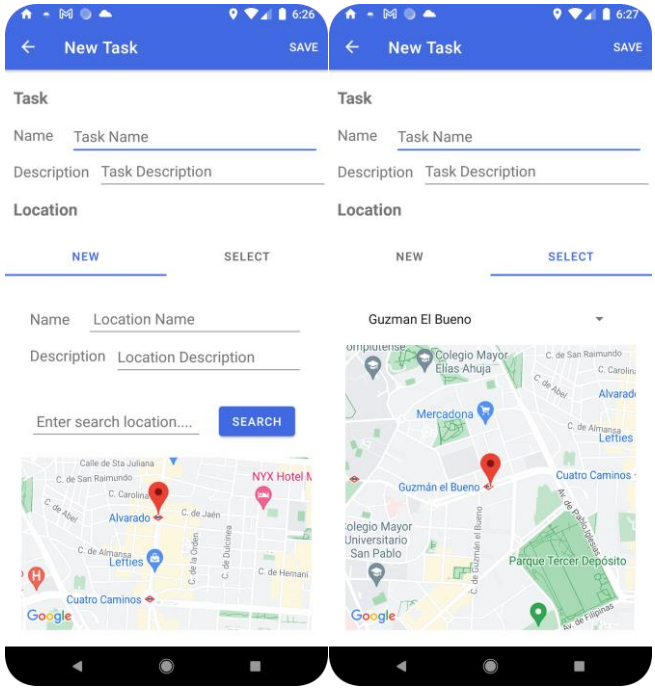


Figura 18: Pantalla auxiliar – Tareas

Esta pantalla es reutilizada para la edición de la tarea. Cuando se entra en modo edición se muestran los datos previamente introducidos de la tarea y el usuario puede realizar las modificaciones en el título y la descripción según sea necesario

Sin embargo, en el caso de la ubicación, la pantalla de edición solo permite modificar la ubicación seleccionando una de las ubicaciones existentes, no permitiendo al usuario crear una nueva ubicación.

3.2.5.1.3 Pantalla auxiliar: Notificaciones Telegram

Las notificaciones de Telegram es una funcionalidad adicional de la aplicación que permite el envío de mensajes a los canales de Telegram que el usuario ha definido y cuando éste llegue al destino indicado.

Las pantallas para añadir notificaciones en la aplicación muestran un listado de contactos guardados y se componen de los siguientes elementos:

- **Tarjetas de contactos**: Estas tarjetas contienen la información del contacto. En el caso de la pantalla de añadir notificaciones, estas tarjetas también incluyen un checkbox para seleccionar el contacto y agregar el mensaje que se enviará o recibirá a través de Telegram.
- **Botón Flotante**: Al hacer clic en él se redirige al usuario a la pantalla de contactos en modo creación, permitiendo agregar nuevos contactos a la lista (*véase apdo. 3.1.5.2*)
- **Diálogo**: Después de que el usuario haya seleccionado los contactos y hecho clic en guardar, se abrirá un diálogo que solicitará la introducción del mensaje que se enviará a Telegram a través del bot. El usuario puede escribir el mensaje en el cuadro de texto del diálogo y luego hacer clic en "Ok" para confirmar la configuración de la notificación.
  Además, si el usuario desea editar el mensaje introducido previamente para un contacto en particular, simplemente necesita hacer clic en la tarjeta de contacto correspondiente. Esto abrirá nuevamente el diálogo

de edición de mensaje, donde se mostrará el mensaje anteriormente introducido para que pueda ser modificado según sea necesario.

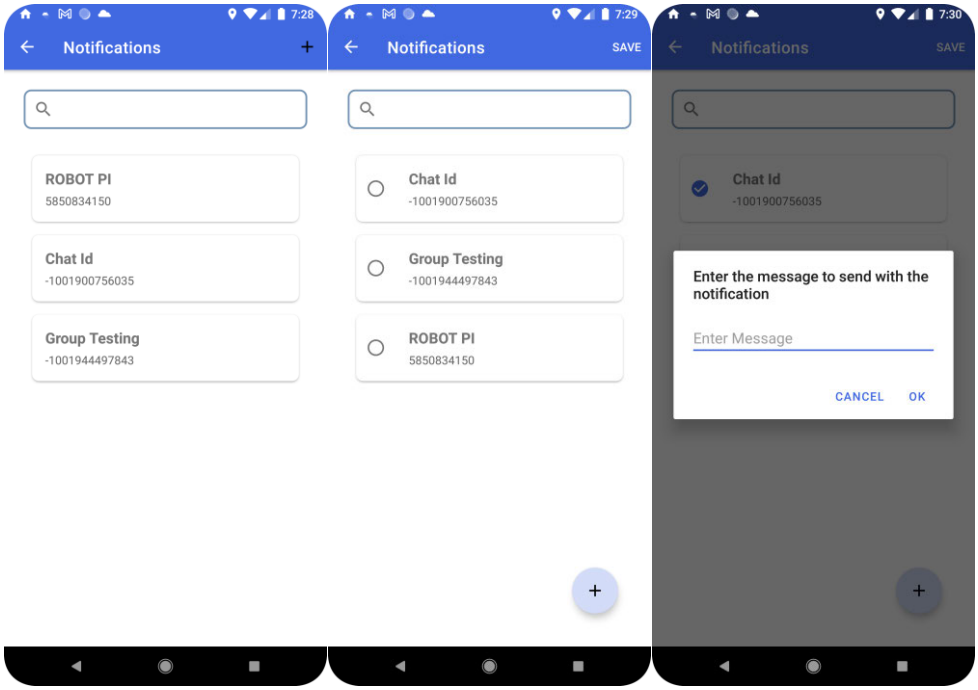


Figura 19: Pantalla auxiliar - Notificaciones Telegram

Esta pantalla de añadir notificaciones, reutiliza la misma pantalla de Contactos con la diferencia de permitir la introducción del mensaje a enviar o recibir a través de *Telegram* (*véase apdo. 3.1.5*)

3.2.5.2 Botón flotante

El botón flotante que se encuentra en el fragmento Home tiene la funcionalidad de redirigir al usuario a la pantalla auxiliar de ubicación. Esta pantalla permite tanto la creación como la edición de ubicaciones.

3.2.5.2.1 Pantalla auxiliar: Ubicación

La creación de ubicaciones es esencial para configurar geocercas para permitir generar eventos cuando el usuario accede a alguna de las ubicaciones agregadas

En la aplicación, se ofrecen dos formas de agregar ubicaciones geográficas:

1. **A través del mapa**: Esta opción permite al usuario seleccionar la ubicación deseada haciendo clic en el mapa. Al hacerlo, se capturan las coordenadas geográficas correspondientes y se guardan como coordenadas geográficas. Esta interacción visual con el mapa brinda una forma intuitiva de agregar ubicaciones.
2. **Búsqueda por dirección**: En esta opción, el usuario puede ingresar la dirección y utilizando la técnica de geocodificación, la aplicación traduce la dirección proporcionada en coordenadas geográficas. Esta opción facilita la introducción de ubicaciones sin necesidad de interactuar directamente con el mapa.

Estas dos opciones brindan flexibilidad al usuario para agregar ubicaciones a sus tareas, ya sea a través de la interacción visual en el mapa o ingresando la dirección directamente.

Además de la ubicación geográfica, se solicita al usuario que proporcione un nombre y una breve descripción de la ubicación. Estos datos ayudan a identificar y recordar la ubicación de manera más fácil para el usuario.

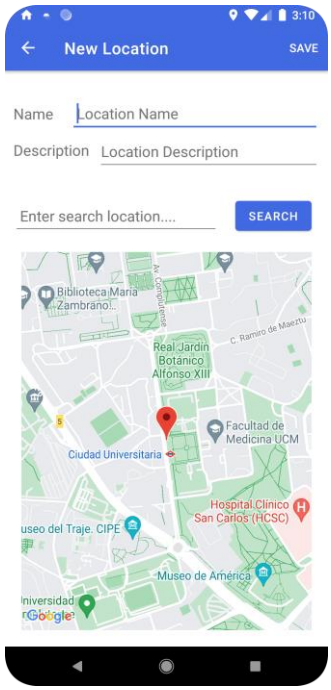


Figura 20: Pantalla auxiliar - Ubicación

A esta misma pantalla se accede a través del título de los contenedores de ubicaciones (*véase apdo. 3.1.3.1.1*) de la pantalla Home, en modo edición de la ubicación.

La edición se diferencia de la creación en que la información de la ubicación seleccionada se carga previamente con los datos guardados en la base de datos, lo que permite modificar los datos existentes.

3.2.6 Pantalla Map

La pantalla Map es una funcionalidad de la aplicación que permite la visualización gráfica de las ubicaciones guardadas. Al acceder a esta pantalla, se muestra un mapa en el que se representan las ubicaciones almacenadas.

Cada ubicación se muestra en forma de marcador en el mapa. Al hacer clic sobre una ubicación, se muestra el nombre de la misma en la etiqueta correspondiente. Esta representación visual facilita una visión global de la distribución geográfica de las mismas.

Figura 21: Pantalla Map

3.2.7 Pantalla Contactos

La pantalla Contactos es una sección de la aplicación que muestra un listado de todos los contactos que han sido agregados. Cada contacto se representa con su nombre y su identificador de canal.

3.2.7.1 Fragment Contacts

El fragmento Contacts se compone de varios elementos que facilitan la interacción del usuario con la lista de contactos. Estos elementos son los siguientes:

1. **Buscador**: Este componente permite al usuario buscar entre los contactos guardados en la aplicación. Al introducir la palabra de búsqueda, se realiza una búsqueda y se muestran aquellos contactos que coincidan con el término ingresado. Esto facilita la localización rápida de un contacto específico, especialmente cuando se tiene una lista extensa de contactos.
2. **Listado de contactos**: Este listado contiene la lista completa de los contactos guardados en la aplicación. Cada contacto se representa en forma de una tarjeta que contiene su nombre y el identificador del canal de chat de Telegram. Esta información permite al usuario identificar y seleccionar el contacto con el que desea interactuar.
3. **Botón flotante**: Este botón, ubicado en la esquina inferior derecha del fragmento, tiene la función de redirigir al usuario a la pantalla de creación de un nuevo contacto (véase apdo. 3.1.5.2).

Estos elementos en conjunto permiten al usuario gestionar los contactos de la aplicación.

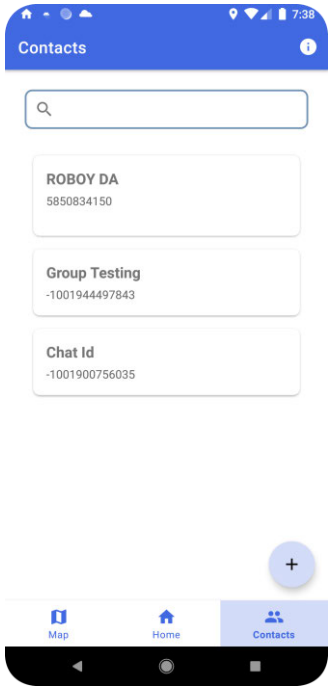


Figura 22: Pantalla Contacts

3.2.7.2 Pantalla auxiliar: Contacto

La pantalla auxiliar Contacto se utiliza para crear nuevos contactos en la aplicación. Para ello, se solicita al usuario ingresar el nombre del contacto y el identificador del chat de Telegram. Estos datos son importantes para asegurar que los mensajes enviados lleguen al destinatario correcto.

Además de los campos de nombre y chat, la pantalla Contacto también incluye una pequeña ayuda o guía para obtener el identificador del chat o canal. Esto permiten a los usuarios que no estén familiarizados obtener el identificador de un chat en Telegram.

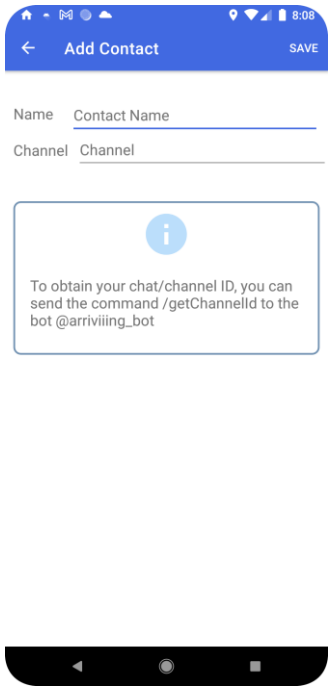


Figura 23: Pantalla auxiliar – Contactos

Al igual que otras pantallas de la aplicación, la pantalla auxiliar Contacto se reutiliza para editar contactos existentes.

Para acceder a esta pantalla solo es necesario clicar en las tarjetas de los contactos que se encuentran en el listado del fragmento Contacts. Al acceder a ésta, se cargarán los datos previamente introducidos del contacto seleccionado, permitiendo la modificación de estos.

3.2.8 Notificación

Cuando se reciben eventos de geovallas, se genera una notificación para informar al usuario sobre las tareas pendientes asociadas a esas ubicaciones. Estas tareas se agrupan en listados según si el usuario llega o sale de la ubicación geográfica establecida mostrando su título y descripción correspondientes.

El diseño de estas notificaciones sigue los estándares y pautas de diseño de Android.

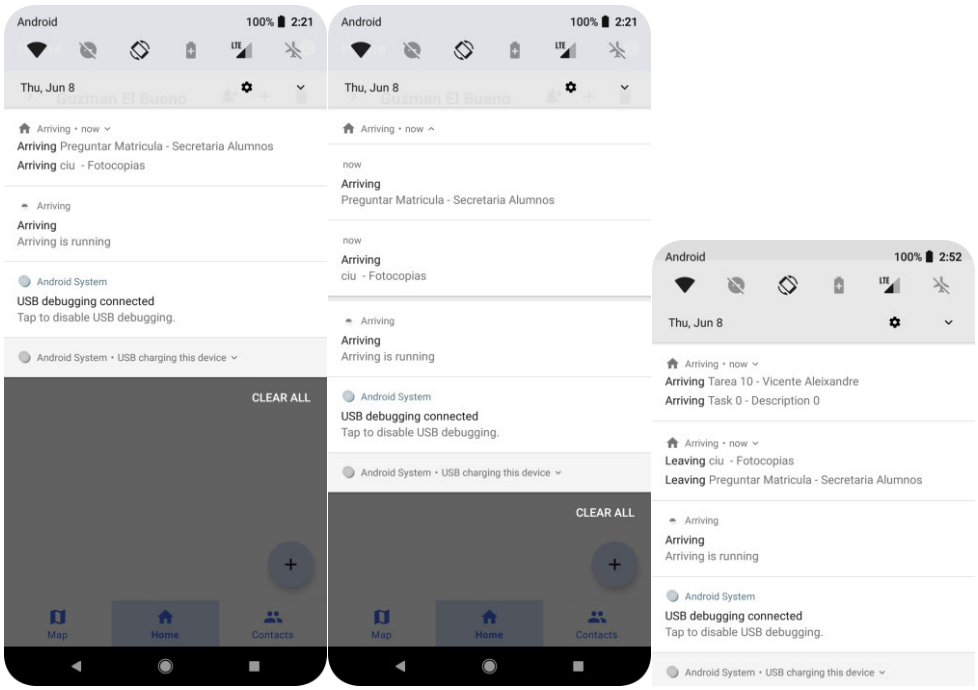


Figura 24: Notificaciones

Cuando el usuario hace clic en las notificaciones de tareas pendientes asociadas a ubicaciones, se abre la aplicación y se muestra la pantalla principal, que es la pantalla de inicio o Home.

3.3 Código

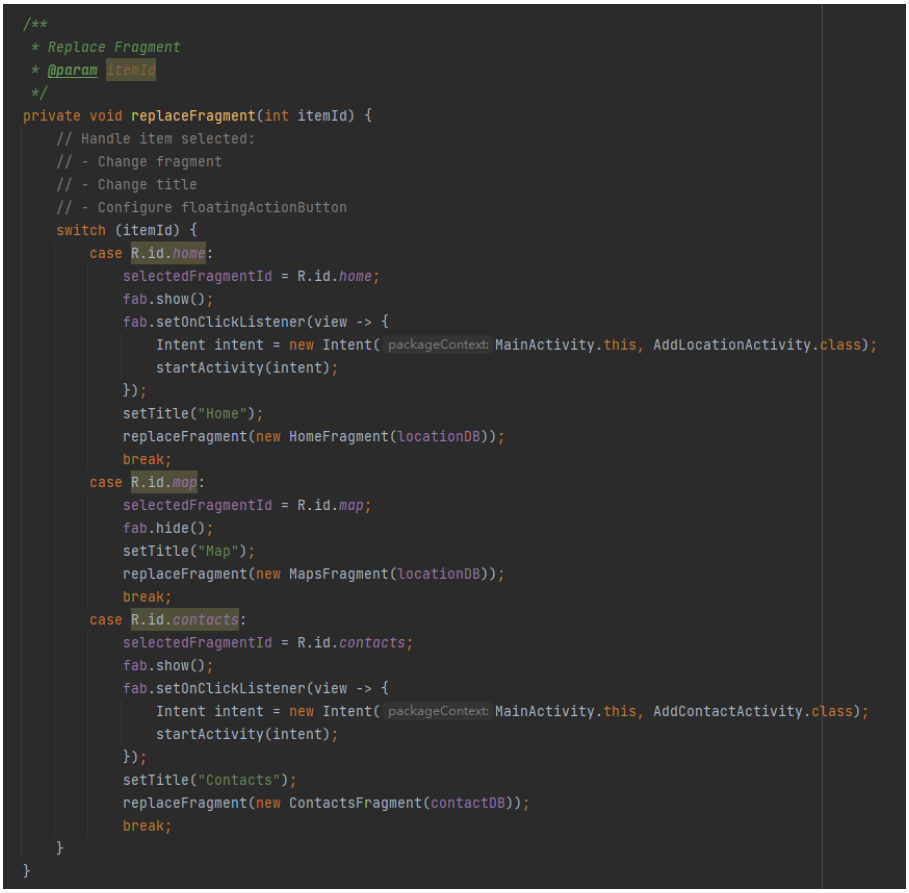
A continuación, se describe brevemente el código empleado en la implementación de la aplicación.

3.3.1 Fragmentos

Los fragmentos son componentes modulares en Android que se utilizan para construir interfaces de usuario flexibles y reutilizables en una actividad. Cada fragmento tiene su propio ciclo de vida y puede recibir eventos y gestionarlos de manera independiente, permitiendo una interacción fluida con el usuario[11] .

En el contexto de la aplicación, se utilizan fragmentos para representar las pantallas principales, Map, Home y Contacts. Éstos dependen de la actividad principal, MainActivity, que es responsable de administrar la transición entre ellos.

La navegación entre fragmentos se logra mediante el uso de una barra inferior que contiene iconos representativos de cada pantalla. Al hacer clic en uno de los iconos de la barra inferior, se realiza una transición al fragmento correspondiente, lo que permite al usuario acceder fácilmente a las diferentes funcionalidades de la aplicación.



```java
/**
 * Replace Fragment
 * @param itemId
 */
private void replaceFragment(int itemId) {
    // Handle item selected:
    // - Change fragment
    // - Change title
    // - Configure floatingActionButton
    switch (itemId) {
        case R.id.home:
            selectedFragmentId = R.id.home;
            fab.show();
            fab.setOnClickListener(view -> {
                Intent intent = new Intent( packageContext: MainActivity.this, AddLocationActivity.class);
                startActivity(intent);
            });
            setTitle("Home");
            replaceFragment(new HomeFragment(locationDB));
            break;
        case R.id.map:
            selectedFragmentId = R.id.map;
            fab.hide();
            setTitle("Map");
            replaceFragment(new MapsFragment(locationDB));
            break;
        case R.id.contacts:
            selectedFragmentId = R.id.contacts;
            fab.show();
            fab.setOnClickListener(view -> {
                Intent intent = new Intent( packageContext: MainActivity.this, AddContactActivity.class);
                startActivity(intent);
            });
            setTitle("Contacts");
            replaceFragment(new ContactsFragment(contactDB));
            break;
    }
}
```

Figura 25: Intercambio de pantallas

Esta figura muestra el código de intercambio de fragmentos en la pantalla principal de la aplicación mediante el método *`replaceFragment`*. Este método determina qué fragmento se muestra según el valor *`itemId`*, que indica el índice del menú de la barra inferior.

Además, se configura la acción del botón `FloatingActionButton` (`fab`) en función del fragmento actual, debido a que el comportamiento del botón varía según el fragmento en el que se encuentra el usuario (*véase en el apdo. 3.1*).

Esta estructura basada en fragmentos proporciona flexibilidad y facilidad en el intercambio de pantallas, lo que permite una navegación suave para el usuario. Al cambiar de un fragmento a otro, la interfaz de usuario se actualiza de manera fluida y se adapta a las necesidades y funcionalidades específicas de cada fragmento.

3.3.2 Actividad principal: MainActivity

Para iniciar la aplicación es necesario definir un punto de entrada en el archivo de manifiesto de la aplicación, que es un archivo XML que describe la estructura y las características de la aplicación[12] .

En nuestra aplicación, se ha definido como la primera actividad la actividad denominada MainActivity. Esta actividad se abrirá cuando el usuario lance inicialmente la aplicación desde el ícono de selección en el dispositivo.

Para ello, en el archivo de manifiesto, se debe declarar el objeto *Intent* que tiene información que el sistema Android usa para determinar qué componente debe iniciar, además de la información que el componente receptor usa para realizar la acción correctamente.

En este caso, las características que debemos incluir son:

- **Acción**: Añadiendo la acción *ACTION_MAIN* indicamos que este es el punto de entrada principal y que no se esperan datos para inicializar el componente.
- **Categoría**: Añadiendo la categoría *CATEGORY_LAUNCHER* indicamos que el ícono de esta actividad debe colocarse en el selector de la aplicación del sistema.

```
<activity
    android:name="com.example.MainActivity"
    android:exported="true"
    android:label="Arriving">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Figura 26: Declaración del MainActivity

3.3.2.1 Funciones

Al iniciar la aplicación, se llevan a cabo una serie de acciones relacionadas con los permisos, la actualización de la ubicación y los eventos de geofencing. Estas acciones permiten el buen funcionamiento de la aplicación:

- **Petición de permisos**: El usuario debe otorgar permisos para que la aplicación funcione correctamente, estos son el acceso a las ubicaciones y la emisión de notificaciones
- **Petición de datos**: Se solicitan al usuario una serie de datos que la aplicación requiere para una consistencia en la aplicación
- **Actualización de ubicación**: Para satisfacer la funcionalidad principal de la aplicación es necesario realizar un seguimiento constante de la ubicación del usuario
- **Eventos de geofencing**: Emitir eventos de geofencing cuando el usuario se acerque a la ubicación y emitir las notificaciones correspondientes.

Las acciones de actualización de ubicación y eventos de geofencing seguirán procesándose en segundo plano para satisfacer el objetivo principal de la aplicación: notificar al usuario de las tareas pendientes basadas en la ubicación.

3.3.2.1.1 Petición de permisos

La aplicación requiere permisos de acceso a la ubicación del dispositivo, por lo que, al iniciar la aplicación, se solicita al usuario que otorgue estos permisos.
Si el usuario rechaza la solicitud, la aplicación no podrá funcionar adecuadamente por lo que se cierra la aplicación.

```java
/**
 * On request permission result
 * If permission is granted, start with the app configuration
 * If not, show a denied permission dialog
 */
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if (requestCode == PERMISSIONS_REQUEST_LOCATION) {
        if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            configGeo();
        } else {
            Toast.makeText( context: this,  text: "Permission denied", Toast.LENGTH_SHORT).show();
            showPermissionDeniedDialog();
        }
    }
}
```

Figura 27: Petición de permisos

3.3.2.1.2 Petición de datos

Para poder utilizar las notificaciones de Telegram es necesario proporcionar el identificador único que identifica el chat donde el usuario interactúa con el bot.
Con este identificador el servidor puede guardar los datos necesarios del usuario, en este caso el nombre de usuario.

En caso de que el usuario no desee continuar con el proceso, el usuario puede no rellenar los datos. Pero al acceder a la pantalla de Contactos se pide de forma obligatoria, puesto que en este momento el usuario requiere del uso de Telegram, siendo el id del chat un requisito requerido para el correcto funcionamiento del servicio.

```java
/**
 * Request chatId
 */
private void requestChatId() {
    AlertDialog.Builder builder = new AlertDialog.Builder( context: this);
    builder.setTitle("Telegram chatId");
    builder.setMessage("For the bot to work, you need to enter your chatId. You can get it by sending the message /getChatId to @arriviiing bot.\n\n" +
            "If you don't want to use the bot, you can leave this field blank.");

    final EditText input = new EditText( context: this);
    input.setInputType(InputType.TYPE_CLASS_TEXT);
    builder.setView(input);
    builder.setPositiveButton( text: "Save", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            String chatId = input.getText().toString();
            if (chatId != null) {
                // Save chatId in preferences
                getSharedPreferences( name: "com.example.arriving", MODE_PRIVATE).edit().putString("chatId", chatId).apply();
            }
        }
    });
    builder.setNegativeButton( text: "Skip", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            // Skip
        }
    });
    builder.show();
}
```

Figura 28: Petición del chat id

3.3.2.1.3 Actualización de ubicación

Una vez que el usuario haya otorgado los permisos de ubicación, se inicia la actualización de la ubicación del dispositivo.
Esto permite que la aplicación pueda obtener la ubicación actual del usuario tanto en primer plano como en segundo plano, permitiendo emitir los eventos de geofencing y realizar las acciones correspondientes.

```
/**
 * Sets up the location request
 */
private void createLocationRequest() {
    mLocationRequest = new LocationRequest.Builder(UPDATE_INTERVAL_IN_MILLISECONDS)
            .setPriority(Priority.PRIORITY_HIGH_ACCURACY).build();
    mRequestingLocationUpdates = true;
}
```

Figura 29: Actualización de ubicación

3.3.2.1.4 Inicialización de eventos de geofencing

Para que se emitan los eventos de geofencing es necesario configurar el listado de ubicaciones que requieren la emisión de eventos. En este caso son las ubicaciones que el usuario ha guardado en la aplicación.

```
/**
 * Initialize geofences
 */
private void initializeGeofences() {
    // Create geofences from tasksLocations
    geofences = new ArrayList<>();
    for (TasksLocation tasksLocation : tasksLocations) {
        if (tasksLocation.getTasks().stream().anyMatch(Task::isDone)) {
            continue;
        }
        Geofence geofence = new Geofence.Builder()
                .setRequestId("geo-" + tasksLocation.getId()).setCircularRegion(tasksLocation.getLatitude(),
                        tasksLocation.getLongitude(),
                        tasksLocation.getRadius()).setExpirationDuration(Geofence.NEVER_EXPIRE)
                .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER | Geofence.GEOFENCE_TRANSITION_EXIT)
                .setNotificationResponsiveness(10000) // 10 seconds
                .setLoiteringDelay(60000) // 1 minute
                .build();
        geofences.add(geofence);
    }
}
```

Figura 30: Inicialización de las geovallas

3.3.2.2 Ciclo de vida

Aparte de habilitar los eventos de las geovallas, es necesario manejar el ciclo de vida de la aplicación.

En el ciclo de vida de la aplicación, es importante manejar adecuadamente las etapas en las que la aplicación pasa a segundo plano (*onPause*) y cuando se cierra (*onDestroy*):

1. **Segundo plano (*onPause*):** Cuando la aplicación pasa a segundo plano, es necesario asegurarse de que se sigan recibiendo los eventos de actualización de ubicación y realizar las acciones correspondientes. Para lograr esto, se habilita el servicio de ubicaciones para que las

actualizaciones de ubicación continúen y se emitan los eventos de geofencing cuando corresponda. Esto garantiza que la aplicación pueda seguir funcionando correctamente en segundo plano y proporcionar notificaciones oportunas al usuario.

```java
@Override
protected void onPause() {
    super.onPause();
    // Start background service
    startBackgroundService( mainActivity: this);
}

private void startBackgroundService(MainActivity mainActivity) {
    Intent serviceIntent = new Intent(mainActivity, LocationService.class);
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        ContextCompat.startForegroundService(mainActivity, serviceIntent);
    } else {
        startService(serviceIntent);
    }
}
```

Figura 31: Código onPause

2. **Cierre de la aplicación (*onDestroy*):** Al cerrar la aplicación, es crucial llevar a cabo algunas acciones para asegurar un cierre adecuado y evitar problemas de duplicación o recursos innecesarios. Esto incluye la destrucción de los escuchadores de eventos, la eliminación de las notificaciones y la limpieza de las geofences.

```java
@Override
public void onDestroy() {
    super.onDestroy();

    // Kill services
    Intent serviceIntent = new Intent( packageContext: this, LocationService.class);
    stopService(serviceIntent);

    // Kill notifications
    NotificationManager notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
    notificationManager.cancelAll();

    // Kill geofences
    killGeofencePendingIntent();
}
```

Figura 32: Código OnDestroy

3.3.3 GeofenceBroadcastReceiver

El componente GeofenceBroadcastReceiver es utilizado para captar y manejar los eventos emitidos por el sistema relacionados con las geovallas.

Su función principal es recibir estos eventos y realizar las acciones correspondientes en la aplicación.

El funcionamiento de los receptores es similar a un patrón de diseño de publicación y suscripción, donde el sistema publica los eventos y los receptores reciben y manejan los datos recibidos.

La implementación de un receptor se siguen estos pasos:

1. **Declarar el receptor en el archivo de manifiesto (AndroidManifest):** En este paso, se declara el receptor en el archivo de manifiesto de la aplicación. Se utiliza la acción *ACTION_GEOFENCE_TRANSITION* para indicar que el receptor escuchará los eventos de geovallas.

```
<receiver
    android:name="com.example.geo.GeofenceBroadcastReceiver"
    android:exported="false">
    <intent-filter>
        <action android:name="com.example.geofence.ACTION_GEOFENCE_TRANSITION"/>
    </intent-filter>
</receiver>
```

Figura 33: Manifiesto - Declaración del receptor

2. **Inicializar el receptor en la actividad**:
   En la actividad principal, que en este caso es MainActivity, se inicializa el receptor. Esto implica crear una solicitud de geovallas utilizando la clase *GeofencingRequest* y asociarla con el receptor GeofenceBroadcastReceiver.

```
/**
 * Config geofencing
 */
private void configGeo() {
    if (geofences == null || geofences.size() == 0) {
        return;
    }
    GeofencingRequest geofencingRequest = new GeofencingRequest.Builder()
            .setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER).addGeofences(geofences).build();

    // Create geofence pending intent
    Intent intent = new Intent( packageContext: this, GeofenceBroadcastReceiver.class);
    PendingIntent pendingIntent = PendingIntent.getBroadcast( context: this,  requestCode: 0, intent, PendingIntent.FLAG_UPDATE_CURRENT);
    geofencingClient.addGeofences(geofencingRequest, pendingIntent);
}
```

Figura 34: Código registro de geovallas

3. **Manejo de eventos**:
   Cuando el receptor recibe un evento de geovallas, se realiza el manejo correspondiente.
   En el código, se muestra el envío de notificaciones si existen tareas pendientes asociadas a la ubicación y se envían las notificaciones a través de Telegram si éstos han sido añadidos.

```
// Delete the id from the list of geofences to avoid duplicate notifications
if (ids.contains(id)){
    ids.remove(id);
    // Send notification
    ManageNotifications.sendNotification(id, context, title, message, GEOFENCE_CHANNEL_ID, GEOFENCE_CHANNEL_NAME);
    // If the user is arriving, send the Telegram notifications
    if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_ENTER) {
        ManageNotifications.sendMessageToServer(context, id);
    }
}
```

Figura 35: Código: emisión de notificaciones

4. **Dar de baja la "suscripción"**: Para evitar duplicados de receptores y garantizar el correcto funcionamiento, es importante dar de baja la suscripción a los eventos cuando la aplicación se cierra. Esto se

realiza en el momento de cierre de la aplicación, donde se elimina la asociación del receptor con las geovallas.

```
private void killGeofencePendingIntent() {
    // Create an intent for the geofence event (must match the one used to register geofences)
    Intent intent = new Intent( packageContext: this, GeofenceBroadcastReceiver.class);
    PendingIntent pendingIntent = PendingIntent.getBroadcast( context: this,  requestCode: 0, intent, PendingIntent.FLAG_UPDATE_CURRENT);

    // Obtain the GeofencingClient
    GeofencingClient geofencingClient = LocationServices.getGeofencingClient( activity: this);

    // Remove the geofences
    geofencingClient.removeGeofences(pendingIntent);
}
```

Figura 36: Código de baja de geovallas

3.3.4 Notificaciones

Las notificaciones enviadas al recibir los eventos de geofencing son posibles gracias a la clase *`NotificationCompat`* [13].

La clase *`NotificationCompat`* es una clase proporcionada por la biblioteca de soporte de Android que facilita la creación y la gestión de las notificaciones. Esta clase permite configurar las características y los comportamientos de las notificaciones.

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(context, channelId)
        .setSmallIcon(R.drawable.home)
        .setContentTitle(title)
        .setContentText(task.getName() + " - " + task.getDescription())
        .setGroup(id.toString())
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        .setAutoCancel(true)
        .setContentIntent(PendingIntent.getActivity(context,  requestCode: 0, intent, PendingIntent.FLAG_UPDATE_CURRENT));
```

Figura 37: Código Configuración de Notificación

Al recibir los eventos de geovallas, se crea una notificación que informa al usuario sobre las tareas pendientes asociadas a la ubicación en cuestión. Para ello, la notificación se compone del título que indica si el evento ha sido de entrada o salida (*`.setContentTitle`*) y una descripción que se trata de la concatenación del nombre y la descripción de la tarea (*`.setContentText`*).

Además, se le atribuye la función de redirigir al usuario a la pantalla principal de la aplicación (*`.setContentIntent`*) cuando éste pulse sobre ella.

3.3.5 Servidor

El servidor se implementa utilizando Node.js, gracias a su facilidad y simplicidad de desarrollo. En nuestro caso, el servidor estará formado por dos componentes: el bot de Telegram y el servidor REST.

El bot de Telegram se encarga de escuchar los mensajes que el usuario envía a través de Telegram, mientras que el servidor REST se encarga de recibir los eventos que la aplicación Android.

La capacidad de Node.js para manejar conexiones concurrentes y operar en tiempo real permite escuchar simultáneamente los eventos del bot de Telegram y atender las solicitudes REST de manera rápida y eficaz.

3.3.5.1 Bot Telegram

Como se ha dicho anteriormente, el bot de Telegram se usa para atender a las solicitudes que el usuario envía a través de Telegram

Para ello se usa la API que ofrece Telegram para Node.js, *node-telegram-bot-api*, que ofrece métodos sencillos que permiten la interacción con la plataforma de Telegram.

Para implementar éste, los pasos a seguir son los siguientes:

1. **Creación de la instancia**:
   Para utilizar la API de Telegram es necesario la importación mediante el método *require*, función de Node.js que permite la importación de módulos.
   Una vez importada la API, se crea una instancia del bot utilizando el token de acceso proporcionado por *@BotFather* al crear el bot.

```
const TelegramBot = require('node-telegram-bot-api');
token = 'your_bot_token'
// Create a new instance of the TelegramBot
const bot = new TelegramBot(token, { polling: true });
```

Figura 38: Instancia de TelegramBot

2. **Implementación del manejador de eventos**:
   Utilizando el método *on* de la API de Telegram se puede escuchar los mensajes que el bot recibe y en función del contenido del mensaje se realizan las acciones específicas como son el envío de mensajes o la realización de alguna operación.

```
// Listen for incoming messages
bot.on('message', (msg) => {
    const chatId = msg.chat.id;
    const type = msg.chat.type;
    let messageText = msg.text;

    // Check if the message is from a group
    if (type === 'supergroup') {
        messageText = getCommandFromGroupMsg(messageText);
    }
    if (![undefined, null].includes(messageText)) {
        // Check if the message contains the command /setUsername
        if (messageText.includes("/setUsername")) {
            setUsernameCommand(chatId, messageText);

        }
        // Check if the message contains the command /editUsername
        else if (messageText.includes("/editUsername")) {
            editUsernameCommand(chatId, messageText);
        }
        else {
            otherCommand(chatId, messageText);
        }
    }
});
```

Figura 39: Código de Manejo de Comandos

3. **Envío de mensajes**:
   Para aumentar la interacción entre el bot y el usuario es necesario el envío de mensajes por parte del bot en respuesta a los mensajes que recibe del usuario. Para ello, la API proporciona el método *sendMessage* que permite el envío de mensajes a través del bot, especificando el chat al cual se envía el mensaje y el contenido del mensaje a enviar.

Con estos pasos conseguimos un bot en funcionamiento:

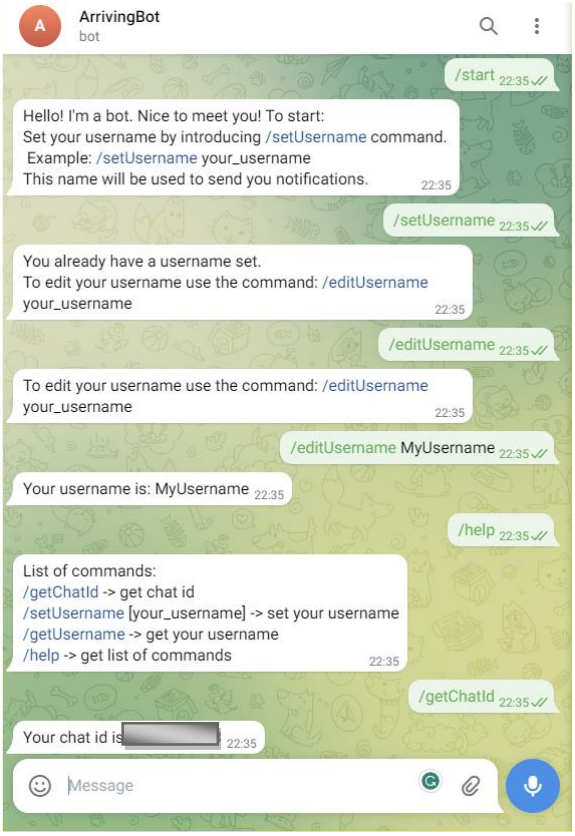


Figura 40: Mensajes con el Bot

3.3.5.1.1 Procesamiento de mensajes enviados al bot: comandos

Como se muestra en la figura 29, los mensajes tienen un tratamiento diferente según tipo y comando.

A. Según tipo

Según de donde provienen los mensajes, los mensajes requieren de un procesado diferente:

- Chat privado: En el chat privado, el usuario únicamente puede interactuar con el bot, por lo que todos los mensajes deben ser procesados. En este caso, se realizan las acciones correspondientes al comando indicado o se envía una ayuda que muestra el listado de comandos disponibles.
- Grupo o canal: En caso de ser un grupo o canal, el usuario puede interactuar por otro usuario del grupo. Por ello es necesario distinguir entre mensajes entre usuarios y los que se mencionan al bot.
  Una vez obtenido el mensaje dirigido al bot es necesario separar la mención del mensaje obteniendo así el comando a procesar.

```
// Function to get the command from a group message
function getCommandFromGroupMsg(messageText) {
    // Check if the message contains a mention of the bot
    if (messageText.includes('@' + botInformation.username)) {
        // separate the command from the arguments
        messageText = messageText.replace('@' + botInformation.username, '').trim();
    } else {
        // If the message doesn't contain a mention of the bot, it's not a command
        messageText = undefined;
    }
    return messageText;
}
```

Figura 41: Procesamiento de mensajes recibidos por un grupo

B. Según comando

Una vez que el mensaje ha sido diferenciado y separado, obteniendo el comando en cuestión, se realiza la acción necesaria. Estas acciones se dividen en dos categorías:

- Comandos relacionados con el nombre de usuario (username):
  Estos comandos requieren guardar el nombre de usuario junto con su identificador único de chat. En la implementación actual, se realiza el almacenamiento en un archivo de texto utilizando los métodos proporcionados por Node.js.

  Sin embargo, como parte de los trabajos futuros, se sugiere evolucionar este método de almacenamiento hacia una base de datos o almacenamiento en la nube para una gestión más eficiente y escalable.

- Resto de comandos:
  Estos comandos tienen como función principal proporcionar información solicitada por el usuario. No requieren una atención especial por parte del bot y solo deben enviar mensajes con la información requerida.

3.3.5.2 Servidor REST: notificaciones

Para el envío de notificaciones a través de Telegram cuando el usuario se encuentra en una ubicación específica, se ha implementado un servidor REST que escucha eventos que recibe por parte del cliente (aplicación Android) y se comunica con el bot de Telegram para emitir los mensajes correspondientes.

Los pasos para implementar el servidor y el cliente REST son los siguientes:

- **Construcción del servidor**:
  El servidor ha sido implementado utilizando el módulo *`express`*, que se trata de un framework de backend ofrecido en Node.js. Éste proporciona un sistema de enrutamiento que facilita la conexión de la aplicación Android con el servidor.

  Al igual que el bot de Telegram, el módulo de *`express`* ofrece la función *`listen`* para aceptar las solitudes recibidas.

```
const express = require('express');
const app = express();
const port = 3000;

// Start the server
app.listen(port, () => {
    console.log(`Server is running on port ${port}`);
});
```

Figura 42: Servidor: Creación del servidor

Como se muestra en la siguiente figura 32, se configura la ruta */send-message* para recibir operaciones POST.

Al recibir solicitudes en esta ruta, el servidor se comunica con el bot de Telegram para enviar el mensaje definido por el usuario, junto con un mensaje predefinido que indica la procedencia de éste y el nombre del usuario que lo envía, al chat correspondiente.

```
public void sendMessage(String from, String to, String message) {
    // Create an instance of Uri.Builder
    Uri.Builder builder = Uri.parse(baseUrl + "/send-message").buildUpon();

    // Add body
    JSONObject jsonBody = new JSONObject();
    try {
        jsonBody.put( name: "from", from);
        jsonBody.put( name: "to", to);
        jsonBody.put( name: "message", message);
    } catch (Exception e) {
        Log.e(TAG,  msg: "Error: " + e.toString());
    }

    // Create request queue
    RequestQueue queue = Volley.newRequestQueue(context);

    JsonObjectRequest request = new JsonObjectRequest(Request.Method.POST, builder.toString(), jsonBody,
            response -> {
                Log.i(TAG,  msg: "Response is: " + response);
            }, error -> Log.e(TAG,  msg: "Error: " + error.toString()));

    queue.add(request);
}
```

Figura 43: Servidor: Envío de mensajes

- **Construcción del cliente**:
  El cliente, en este caso la aplicación Android, debe conectarse a la API REST del servidor anterior. Para ello, se utiliza la biblioteca HTTP Volley que permite realizar solicitudes HTTP de manera sencilla y manejar las respuestas de manera eficiente.
  El cliente Android debe construir y enviar la solicitud HTTP POST al servidor, proporcionando los datos necesarios, que son los identificadores del chat del emisor y receptor y el mensaje a enviar.

```java
public void sendMessage(String from, String to, String message) {
    // Create an instance of Uri.Builder
    Uri.Builder builder = Uri.parse(baseUrl + "/send-message").buildUpon();

    // Add query parameters
    builder.appendQueryParameter("from", from);
    builder.appendQueryParameter("to", to);
    builder.appendQueryParameter("message", message);

    // Create a String request
    String url = builder.build().toString();

    RequestQueue queue = Volley.newRequestQueue(context);
    StringRequest stringRequest = new StringRequest(Request.Method.POST, url,
            response -> {
                Log.i(TAG,  msg: "Response is: " + response);
            }, error -> Log.e(TAG,  msg: "Error: " + error.toString()));
    queue.add(stringRequest);
}
```

Figura 44: Cliente: configuración REST

Con estas configuraciones se establece la comunicación entre la aplicación y el servidor permitiendo el envío de las notificaciones a través de Telegram.

3.4 Privacidad y seguridad de datos

Los datos manejados en la aplicación no requieren una protección extrema debido a su naturaleza no altamente sensibles. Sin embargo, se toman medidas para garantizar la protección y seguridad del usuario.

3.4.1 Servidor: Telegram Bot

El servidor se implementa el algoritmo de encriptación AES-256-CBC utilizando el módulo *crypto* proporcionado por Node.js. Este algoritmo[14] de encriptación se utiliza para proteger los datos almacenados en la base de datos local.

```javascript
const algorithm = 'aes-256-cbc';
const secretKey = 'secret_key'; // Replace with your own secret key

// Convert secretKey to Buffer
const secretKeyBuffer = Buffer.from(secretKey, 'hex');

const crypto = require('crypto');
```

Figura 45: Módulo crypto

Para encriptar los datos, se utiliza una clave de encriptación (secretKey) y un vector de inicialización (iv). Estos valores se utilizan en el proceso de encriptación para garantizar que los datos estén protegidos. El uso de AES-256-CBC proporciona una capa adicional de seguridad al proteger los datos almacenados en la base de datos de posibles accesos no autorizados.

```
function encryptData(data, key, iv) {
    if ([undefined, null, ''].includes(data)) {
        return data;
    }
    const cipher = crypto.createCipheriv(algorithm, key, iv);
    let encrypted = cipher.update(data, 'utf8', 'hex');
    encrypted += cipher.final('hex');
    return encrypted + iv.toString('hex');
}
```

Figura 46: Encriptación de datos

En cuanto a los mensajes enviados a través del chat de Telegram, el servidor no almacena los mensajes ni ningún otro dato, excepto el identificador del chat o grupo y el nombre del usuario. Esto se hace con el fin de garantizar la privacidad de las conversaciones y minimizar el riesgo de filtración de información. Al no almacenar los mensajes, se reduce la posibilidad de que terceros accedan a información confidencial o privada intercambiada a través de la aplicación.

3.4.2 Cliente: Aplicación Android

En el caso del cliente, la aplicación Android, la seguridad de los datos almacenados es proporcionada por el framework SQLite que se utiliza en Android[15]. Dado que los datos manejados en la aplicación no son altamente sensibles, no se ha implementado ningún método de protección adicional.

Los datos que se guardan en la base de datos local de la aplicación incluyen información como ubicaciones, tareas y contactos. Estos datos son necesarios para el correcto funcionamiento de la aplicación y se almacenan utilizando las capacidades de seguridad y control de acceso proporcionadas por el framework SQLite de Android.

Es importante destacar que los datos a los que se accede durante el uso de la aplicación, como la ubicación actual, no se guardarán en ningún caso. Estos datos se utilizan de forma inmediata para brindar las funcionalidades requeridas por la aplicación en tiempo real. Esto garantiza que la información sensible, como la ubicación del usuario, no se almacene innecesariamente, lo que contribuye a la protección de la privacidad del usuario y la minimización de los riesgos de seguridad asociados.

4 Conclusiones y trabajos futuros

4.1 Conclusión

En conclusión, el desarrollo de este proyecto ha resaltado la importancia de cada etapa del ciclo de desarrollo de software. En concreto:

- **Planificación**: una buena planificación del proyecto permite que el desarrollo sea de forma constante y continua, evitando los bloqueos debido a la mala definición de requisitos y metas
- **Diseño**: con el diseño de la aplicación se ha podido identificar y separar cada una de las pantallas y funcionalidad, facilitando el desarrollo posterior del código de la aplicación.

Estas etapas proporcionan una base sólida que facilita el desarrollo del código y la implementación de las funcionalidades requeridas.

Además, el desarrollo de este proyecto ha permitido la aplicación directa de los conocimientos adquiridos durante el proceso de aprendizaje como son el desarrollo software, las técnicas de geofencing y geocoding y el desarrollo de aplicaciones para dispositivos Android.

4.2 Trabajos futuros

Los trabajos futuros de la aplicación se pueden dividir en dos secciones principales: mejoras y actualizaciones de funcionalidades.

A continuación, se detallan algunas áreas de mejora y actualización:

- **Mejoras de diseño y usabilidad**:
  - Mejorar el diseño estético de la aplicación para ofrecer una interfaz más atractiva y moderna.
  - Optimizar la usabilidad de la aplicación, simplificando la navegación y haciéndola más intuitiva para los usuarios.
  - Realizar pruebas de usuario y recopilar comentarios para identificar áreas de mejora y realizar ajustes en base a la retroalimentación recibida.
- **Actualizaciones de funcionalidades**:
  - Implementar la opción de repetición para las tareas, lo que permitirá configurar recordatorios recurrentes automáticamente.
  - Mejorar el rendimiento de las diferentes pantallas, optimizando el tiempo de carga y respuesta de la aplicación.
  - Ampliar la funcionalidad de la pantalla Map, permitiendo la creación de nuevas ubicaciones directamente en el mapa y mostrar información detallada sobre cada ubicación, como tareas asociadas y otros datos relevantes.
- **Funcionalidades adicionales**:
  - Implementar mejoras en las notificaciones para que sean más informativas, incluyendo detalles específicos de las tareas y permitiendo la personalización de las notificaciones por parte del usuario.
  - Permitir el acceso rápido a las tareas asociadas a una ubicación específica, brindando una vista rápida de las tareas pendientes en cada ubicación.
  - Incorporar la posibilidad de personalizar la apariencia y configuración de la aplicación, como temas de color, fuentes y preferencias de visualización.

    - Explorar la integración con otras plataformas y servicios, como calendarios externos, para una mejor sincronización de tareas y recordatorios.
- **Almacenamiento**:
    - Evolución del almacenamiento local al almacenamiento en línea para una gestión más efectiva de los datos en la aplicación.

5 Bibliografía

[1] Definición de Geofencing, Online, https://www.useit.es/blog/geofencing-que-es-y-como-funciona

[2] Api Geocoding, Online, https://developers.google.com/maps/documentation/javascript/geocoding?hl=es#:~:text=La%20geocodificaci%C3%B3n%20es%20el%20proceso,marcadores%20o%20posicionar%20el%20mapa.

[3] Bots Telegram, Online, https://www.xataka.com/basics/bots-telegram-que-como-funcionan-recomendados-para-empezar

[4] Bots Api, Online, https://core.telegram.org/bots

[5] Node.js, Online, https://nodejs.org/en

[6] ¿Por qué usar Nodejs?, Online, https://mortegac.medium.com/porque-usar-nodejs-en-el-2020-38ea971c6425

[7] Android Studio, Online, https://developer.android.com/studio/features?hl=es-419

[8] Apple Reminders, Online, https://support.apple.com/en-us/HT205890

[9] Todoist, Online, https://todoist.com/es/help/articles/introduction-to-reminders

[10] Google Keep, Online, https://support.google.com/keep/answer/3187168?hl=en&co=GENIE.Platform%3DDesktop

[11] Fragmentos, Online, https://developer.android.com/guide/components/fragments?hl=es-419#:~:text=Un%20Fragment%20representa%20un%20comportamiento,un%20fragmento%20en%20diferentes%20actividades.

[12] Intents y filtros de intents, Online, https://developer.android.com/guide/components/intents-filters?hl=es-419

[13] NotificationCompat, Online, https://developer.android.com/training/notify-user/build-notification?hl=es-419

[14] Encriptación AES, Online, https://medium.com/swlh/an-introduction-to-the-advanced-encryption-standard-aes-d7b72cc8de97

[15] Seguridad SQLite Android, Online, https://developer.android.com/training/data-storage/sqlite