UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Diseño de una aplicación móvil para gestionar recordatorios de medicación

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Alfaro Martínez, Javier

Tutor/a: Vidal Oriola, Germán Francisco

CURSO ACADÉMICO: 2023/2024

Resum

Els tractaments farmacològics són un pilar fonamental de la medicina occidental actual. Quan un pacient ha d'iniciar un tractament nou poden sorgir dubtes i problemes respecte al mateix. Una bona adherència terapèutica repercutix beneficiosament en la salut de la població. Molts pacients, especialment els majors de seixanta-cinc anys, es troben polimedicats, la qual cosa en moltes ocasions porta a errors en la medicació o l'oblit de preses.

L'objectiu principal d'este treball és dissenyar i desenvolupar una aplicació mòbil que permeta registrar medicaments i associar un recordatoris a estos, permetent la seua gestió en qualsevol moment i lloc. Cada medicament tindrà informació i instruccions. Se seguirà l'activitat de l'usuari mitjançant informes, els quals poden ser enviats a persones de suport. L'aplicació està orientada a persones majors, té una interfície senzilla i no necessita connexió a internet. L'aplicació s'ha desenvolupat per a dispositius mòbils *Android*. Per a dur a terme este projecte, s'ha emprat una metodologia àgil adaptada al context del desenvolupament.

Paraules clau: aplicació, Android, Kotlin, mòbil, Android Studio, Room, MVVM, recordatori, medicament

Resumen

Los tratamientos farmacológicos son un pilar fundamental de la medicina occidental actual. Cuando un paciente tiene que iniciar un tratamiento nuevo pueden surgir dudas y problemas respecto al mismo. Una buena adherencia terapéutica repercute beneficiosamente en la salud de la población. Muchos pacientes, especialmente los mayores de sesenta y cinco años, se encuentran polimedicados, lo que en muchas ocasiones lleva a errores en la medicación o el olvido de tomas.

El objetivo principal de este trabajo es diseñar y desarrollar una aplicación móvil que permita registrar medicamentos y asociar un recordatorios a estos, permitiendo su gestión en cualquier momento y lugar. Cada medicamento tendrá información e instrucciones para facilitar la toma. Se seguirá la actividad del usuario mediante informes, los cuales pueden ser enviados a personas de apoyo. La aplicación está orientada a personas mayores, tiene una interfaz sencilla y no necesita conexión a internet. La aplicación se ha desarrollado para dispositivos móviles *Android*. Para llevar a cabo este proyecto, se ha empleado una metodología ágil adaptada al contexto del desarrollo.

Palabras clave: aplicación, Android, Kotlin, móvil, Android Studio, Room, MVVM, recordatorio, medicamento

Abstract

Pharmacological treatments are the cornerstone of today's Western medicine. When a patient has to start a new treatment, doubts and problems may arise. Good adherence to treatment has a beneficial effect on the health of the population. Many patients, especially those over sixty-five, are polymedicated, which often leads to medication errors.

The main objective of this work is to design and develop a mobile application that allows to register medications and associate a reminder to these. Furthermore, it allows its management at any place. Each medication have information and instructions to facilitate the take. The user's activity is tracked through reports which can be sent by email.

The application is oriented to elderly people so it has a simple interface and do not require internet connection. The application was developed for Android mobile devices. To carry out this project, an agile methodology adapted to the development context has been used.

Key words: app, Android, Kotlin, mobile, Android Studio, Room, MVVM, reminder, medicine

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

1.1 Motivación

Los tratamientos farmacológicos son un pilar fundamental de la medicina occidental actual. Cuando un paciente tiene que iniciar un tratamiento nuevo pueden surgir dudas y problemas respecto al mismo.

Una buena adherencia terapéutica repercute beneficiosamente en la salud de la población. La salud no es algo estático, puede ir cambiando a lo largo de diferentes épocas, así como lo hacen los fármacos que precisa cada paciente en un momento determinado.

Muchos pacientes, especialmente los mayores de sesenta y cinco años, se encuentran polimedicados (más de cinco fármacos) lo cual en muchas ocasiones lleva a errores en la medicación, olvido de tomas, sobredosificación, efectos adversos no deseados, etc.

Una aplicación para el control de la medicación podría ayudar a evitar todos los problemas mencionados anteriormente, ya que mejoraría la adherencia al tratamiento, pudiendo ajustar los recordatorios a las necesidades de cada paciente facilitando así que tomen su medicación de forma adecuada.

Una aplicación de estas características puede proporcionar datos sobre los medicamentos, incluyendo descripciones, dosis recomendadas, efectos secundarios, interacciones con otros fármacos, etc, así como proporcionar avisos y recordatorios de la toma de los medicamentos.

En resumen, se persigue el desarrollo de una aplicación móvil que facilite el acceso a esta información desde cualquier lugar, permitiendo además que los familiares más cercanos tengan acceso a toda esta información.

1.2 Objetivos

El presente proyecto propone una solución que permita al usuario i) seguir uno o varios tratamientos mejorando la adherencia mediante recordatorios ii) configurar y personalizar los recordatorios iii) gestionar el recuento de medicamentos restantes por tratamiento y obtener informes periódicos. Dicho objetivo se alcanzará mediante el diseño y desarrollo de una aplicación móvil nativa.

Para el correcto desarrollo y funcionamiento de la aplicación propuesta, se deberán cumplir los siguientes requerimientos:

- Estudiar y desarrollar la funcionalidad que permita al usuario crear y recibir recordatorios para la toma de medicamentos mediante un dispositivo móvil, permitiendo el uso de funcionalidades integradas como notificaciones o cámara.
- Implementación de un *backend* y una base de datos local que permita el almacenamiento y la persistencia de los datos del usuario aun sin conexión a internet.
- Perseverar en la aplicación de los principios de *Clean Code* y *Clean Arquitecture* durante la etapa de desarrollo, implementando patrones de diseño y empleando conjuntos de *test* con el propósito de simplificar el procedimiento de mantenimiento.
- Empleo de una metodología ágil que nos posibilite y facilite la gestión y seguimiento del desarrollo del proyecto.
- Configurar y llevar a cabo un *Mininum Viable Product* (MVP) que posibilite al usuario el uso de las funciones esenciales en la versión inicial.

A su vez se establecen también algunos objetivos personales que se darán por completados durante el desarrollo de este proyecto:

- Adentrarse en el desarrollo de aplicaciones móvil, desde su inicio hasta su conclusión, haciendo uso de las mejores prácticas y consejos ofrecidos por artículos, documentos y comunidades de expertos.
- Absorber el conocimiento de un lenguaje de programación novedoso, comprendiendo su funcionamiento interno y su repercusión en el sector de las aplicaciones móviles.

1.3 Estructura de la memoria

La memoria se organiza en múltiples secciones donde se detallará todo el proceso de desarrollo de este proyecto. Se presenta a continuación el contenido de los restantes capítulos;

2. **Metodología de trabajo**: se detalla el uso de herramientas y metodologías empleadas a lo largo del desarrollo del proyecto.
3. **Análisis estratégico**: en esta sección se lleva a cabo un análisis de las soluciones vinculadas a aplicaciones de recordatorio de medicamentos en el ámbito digital, con el propósito de facilitar la identificación y extracción de requisitos.
4. **Análisis del problema**: esta sección detalla el procedimiento de obtención de requisitos y luego se obtiene una lista de los requisitos acordados.
5. **Diseño de la solución**: se exponen las decisiones adoptadas en cuanto a la arquitectura, lenguaje de programación, patrones de diseño y las tecnologías utilizadas.
6. **Implementación de la solución**: se describen las distintas secciones que integran la aplicación, explorando su comportamiento y el funcionamiento interno de cada una, profundizando en los aspectos técnicos.
7. **Despliegue y mantenimiento**: se describe el procedimiento implementado para llevar a cabo el mantenimiento y hacer que el aplicativo sea accesible públicamente.
8. **Conclusión y trabajo futuro**: evaluamos los resultados obtenidos durante la realización del trabajo y las mejoras a implementar en futuras versiones del proyecto.

CAPÍTULO 2
Metodología de trabajo

La solución de software resultante, al igual que cualquier otra aplicación, debe cumplir con unos requisitos específicos para abordar problemas concretos. A su vez, antes de considerar completado el desarrollo, este debe someterse a diversos procesos que, en conjunto, son conocidos como Ciclo de Vida del Desarrollo del Software (SDLC, por sus siglas en inglés).

Algunas de sus fases más comunes son las siguientes:

- **Planificación**: se definen los requisitos esenciales para la aplicación con el propósito de establecer un objetivo principal sin abordar detalles técnicos. Esta etapa también involucra tareas como el análisis de costos y beneficios, así como la estimación y asignación de recursos.
- **Diseño**: se lleva a cabo un análisis de los requisitos y se identifican las soluciones óptimas para la creación del software, y se elabora un diseño técnico detallado que establece las funciones tecnológicas necesarias para el desarrollo.
- **Implementación**: se estudian con detalle los requisitos establecidos y se procede a la creación del código en el lenguaje de programación seleccionado.
- **Pruebas**: se utiliza una mezcla de pruebas automáticas y manuales por parte del equipo de desarrollo para asegurar la calidad del software y detectar posibles errores.
- **Despliegue**: consiste en ejecutar la puesta en marcha del producto software, permitiendo que el sistema completo esté disponible para el usuario final.
- **Mantenimiento**: durante esta fase, se corrigen errores, se solucionan problemas informados por los clientes y se gestiona cualquier cambio implementado en el software.

Existen diversos modelos que organizan estas fases que podemos dividir entre tradicionales y ágiles. Las prácticas tradicionales siguen, de manera secuencial, el ciclo mencionado anteriormente, algunos ejemplos de este enfoque incluyen: Modelo en Cascada [1] y *V-Model* [2]. Estas metodologías se caracterizan porque realizan el trabajo de cada etapa una vez y en orden, por lo que resultarían demasiado rígidas en nuestro caso. Esto se debe a que tanto los requisitos como el tiempo de desarrollo están establecidos, lo que limitaría la capacidad de adaptación a los cambios que puedan surgir durante el desarrollo. Dado que las etapas se realizan en secuencia, los resultados del desarrollo solo se manifiestan en las últimas fases del ciclo de vida del proyecto y cualquier modificación

en el producto en este punto podría generar un aumento considerable en los costos y tiempos del proyecto.

Ante las desventajas señaladas, se ha decidido adoptar una metodología ágil, lo que posibilita un mayor control sobre la calidad del producto, así como la reducción de fallos y la implementación de mejoras continuas. Concretamente, optaremos por utilizar *SCRUM* [3] para el desarrollo de este proyecto. El desarrollo se llevará a cabo de manera iterativa, lo que implica la división del producto en múltiples iteraciones, obteniendo así un *MVP* (*Minimum Viable Product*) en cada una de ellas. En las primeras iteraciones, nos centraremos en desarrollar las capacidades mínimas que debe tener la aplicación. Al tratarse de un proyecto individual, no hay roles específicos para *product owner* y *scrum master*, por lo que ambos serán desempeñados por el desarrollador. El desarrollo se dividirá en *sprints* de dos semanas.

Al principio de cada *sprint*, se realizará una reunión (*sprint planning*) en la que se examinarán los requisitos del *backlog* y se tomarán decisiones sobre cuáles serán desarrollados en el siguiente *sprint*, y no podrán ser modificados durante esas semanas. Puesto que el desarrollo es individual, se realizará una revisión diaria, similar a una *daily*, en la cual se analizará el progreso del *sprint* y se dará especial atención a las tareas que lo necesiten.

Para mantener un trabajo bien organizado y documentado, optaremos por utilizar la técnica Kanban. La implementación de esta metodología se lleva a cabo a través de tableros Kanban, un enfoque visual de los flujos y carga de trabajo, lo que facilita una gestión más eficiente y transparente de las tareas. Cada columna en el tablero representa una fase del trabajo. En este proyecto, hemos optado por incluir: *To do*, *In progress*, *Testing* y *Done* (ver Figura 2.1). En el tablero, las tareas están representadas por tarjetas visuales que avanzan a través de todas las columnas, desde *To do* hasta *Done*, marcando su progreso hasta su finalización.

En internet, existen diversas herramientas que nos brindan la capacidad de crear tableros Kanban para organizar de manera efectiva nuestro trabajo. Hemos decidido utilizar Trello [4], ya que nos proporciona una excelente experiencia de usuario, es fácil de aprender, de uso gratuito y accesible tanto en ordenadores como en dispositivos móviles.

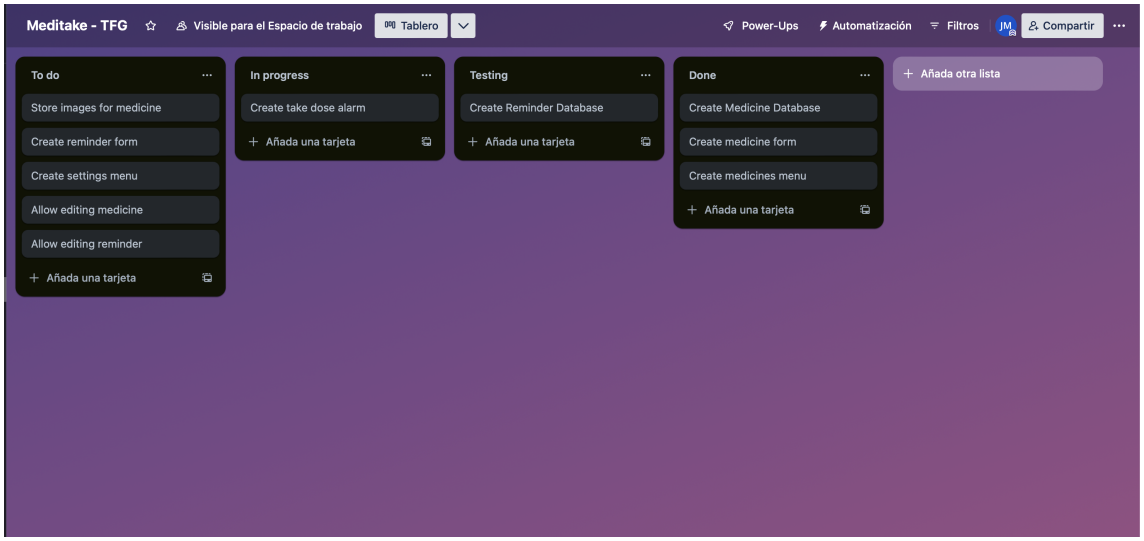

Figura 2.1: Tablero Kanban del proyecto en un *sprint* intermedio

CAPÍTULO 3
Análisis estratégico

3.1 Aplicaciones

Se han explorado diversas aplicaciones relacionadas con los recordatorios y seguimiento de medicamentos, eligiendo las aplicaciones principales y mejor valoradas de las tiendas más relevantes: *Google Play* y *App Store*.

3.1.1. Mytherapy

Mytherapy es una aplicación multiplataforma gratuita dirigida a todos los públicos. Actualmente está en su versión 3.147.1. Destaca por su diseño y animaciones, incluyendo toda la funcionalidad necesaria para llevar el seguimiento de la medicación. Podemos observar un ejemplo de esto en la Figura 3.1.

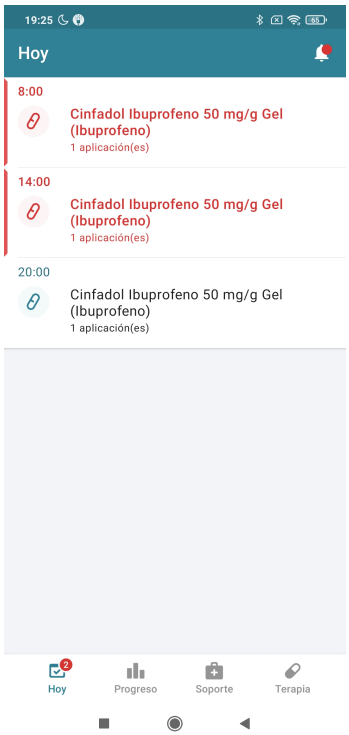

Figura 3.1: Pantalla de tomas pendientes *Mytherapy*

Esta aplicación establece conexión a internet y requiere la creación de un usuario en el que se guardarán todos los datos. También posibilita que se añada una persona de apoyo para supervisar las tomas. Además, el aplicativo ofrece funcionalidades para gestionar citas médicas como observamos en la Figura 3.2.

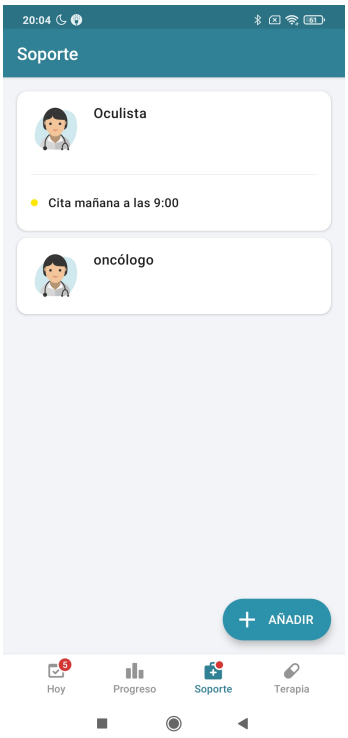

Figura 3.2: Pantalla de citas pendientes *Mytherapy*

Es importante resaltar la cantidad de notificaciones recibidas a lo largo del día, lo cual podría resultar incómodo y causar confusión en el momento de la toma del medicamento.

3.1.2. Medicamentos

Medicamentos es una aplicación gratuita orientada a un público mayor. Como observamos en la Figura 3.3 su interfaz es más simple y carece de detalles. Actualmente se encuentra en su versión 2.3.

La aplicación ofrece la opción de registrar diferentes médicos, los cuales puedes asignar a los medicamentos para recordar quién o qué especialidad los ha prescrito (ver Figura 3.4).

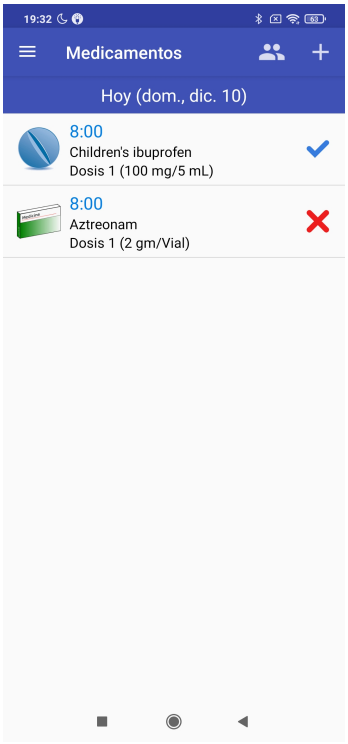

Figura 3.3: Pantalla de recordatorios *Medicamentos*

La aplicación permite generar informes con la información de las tomas recientes del usuario, con la opción de enviarlos por correo electrónico y exportarlos en formato *Excel*. También se pueden crear diferentes perfiles en un mismo dispositivo, facilitando así la gestión de tratamientos diferentes de manera clara y sin confusión.

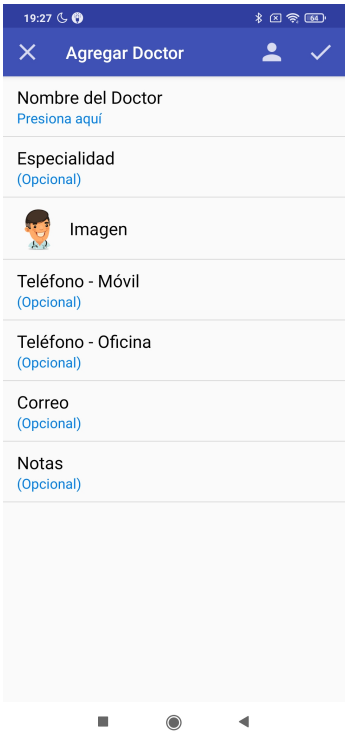

Figura 3.4: Pantalla de registro de médico *Medicamentos*

El aplicativo opera sin conexión a internet y almacena localmente los datos del usuario. Además, ofrece la opción de exportar estos datos en un archivo que puede ser enviado e importado en otro dispositivo.

3.1.3. Medisafe

Medisafe, aplicación parcialmente gratuita, se encuentra actualmente en su versión 9.37. El aplicativo tiene una interfaz de usuario cuidada (ver Figura 3.5) y diversas animaciones. El público objetivo de esta aplicación es más joven y la configuración de la misma se percibe como más compleja.

En sus funciones gratuitas, la aplicación incluye un calendario que muestra de manera útil y clara los medicamentos pendientes de toma para cada día, proporcionando una forma efectiva de verificar el estado de estas.

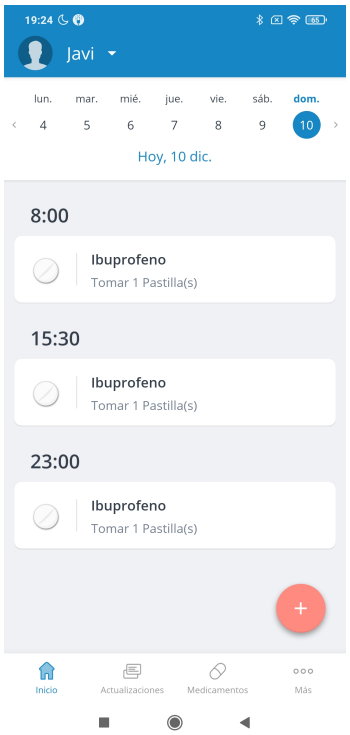

Figura 3.5: Pantalla de tomas pendientes *Medisafe*

Esta aplicación también permite el envío de informes a familiares o médicos; en estos informes también se incluyen cálculos basados en las tomas del usuario, como el porcentaje de adherencia. La aplicación incluye recordatorios para reabastecer los medicamentos que están a punto de agotarse.

El aspecto sobresaliente de esta aplicación es la capacidad de agregar otras mediciones de salud, como el nivel de dolor experimentado en un momento específico. Los datos del usuario se almacenan en línea y están asociados a una cuenta.

Como característica de pago destaca la integración de esta aplicación con dispositivos *Android Wear*, el sistema operativo de relojes inteligentes, permitiendo recibir y confirmar las tomas.

3.2 Análisis de las aplicaciones

Este análisis nos permite identificar los puntos fuertes de nuestra propuesta y priorizar ciertas funcionalidades. Además, brinda nuevas ideas para incorporar a nuestro aplicativo. La Tabla 3.1 muestra el resumen de este análisis.

Tabla 3.1: Comparativa de características entre aplicaciones del mismo sector

Caracteristica	MyTherapy	Medicamentos	Medisafe	Propuesta
Aplicación multiplataforma	SI	NO	SI	NO
Añadir cualquier tipo de medicamento[1]	NO	NO	NO	SI
Establecer existencias del medicamento	SI	NO	SI	SI
Calendario de tomas	SI	NO	SI	NO
Cuenta asociada	SI	NO	SI	NO
Establecer número de dosis a tomar	SI	SI	NO	SI
Establecer instrucciones de toma	NO	NO	NO	SI
Establecer imagen de medicamento	NO	SI	NO	SI
Recibir notificación de toma	SI	SI	SI	SI
Añadir fecha fin de tratamiento	NO	SI	SI	SI
Seguimiento de adherencia	SI	NO	SI	SI
Envío de informes a persona de apoyo mediante correo electrónico	SI	NO	SI	SI
Añadir contacto médico	SI	SI	SI	NO
Posponer toma	SI	SI	SI	NO
Perfil múltiple	NO	SI	SI	NO

En el análisis de las aplicaciones, ninguna sobresale por su simplicidad al confirmar la toma de un medicamento, ya que las pantallas carecen de claridad y están llenas de información innecesaria.

Nuestra propuesta se centrará en obtener los requisitos principales para la gestión de los medicamentos y recordatorios, por ello esta primera versión descartará funcionalidades como perfil múltiple o calendario de tomas. No se tendrá una cuenta asociada ya

[1] En las aplicaciones mencionadas el medicamento es seleccionado a partir de una lista predeterminada mientras que en nuestra propuesta se personaliza el nombre e imagen.

que la primera versión de este aplicativo no tendrá conexión a internet. No se creará una sección específica para añadir un contacto médico ya que se podrá añadir una persona de apoyo mediante correo electrónico.

3.3 Propuesta

Después de este análisis, podemos concluir que la primera versión de nuestra aplicación incluirá las siguientes características:

- Se podrá registrar cualquier tipo de medicamento incluyendo nombre, dosis, imagen y descripción del mismo.
- Se podrá incorporar el número de comprimidos que contiene un envase y recibir un aviso visual cuando este se acerque a su fin.
- Será posible crear un recordatorio para cada medicamento pudiendo elegir varios momentos del día.
- La frecuencia de los recordatorios podrá ser fija o se podrá seleccionar los días de la semana para recibir los avisos de toma.
- El aplicativo permitirá editar los medicamentos y recordatorios en cualquier momento, así como eliminarlos.
- Se generarán informes recurrentes que incluirán el resumen de tomas del usuario.
- Se incluirá la opción de añadir un correo electrónico de contacto que recibirá los informes generados.
- Se enviará una notificación de toma, dando la opción al usuario de indicar si ha sido efectiva o no.

CAPÍTULO 4
Análisis del problema

4.1 Especificación de requisitos

En este apartado, abordaremos la especificación de requisitos. Este proceso consiste en registrar todos los requisitos del sistema y del usuario en un documento. Es fundamental que estos requisitos sean claros, completos y coherentes y verificables. Es necesario detallar todos los requisitos que formarán parte del producto al finalizar este proyecto.

Optaremos por una técnica informal a la hora de especificar, la cual implica el uso de un lenguaje natural y preciso con la finalidad de que sea fácilmente comprendida por el lector.

Como se ha señalado previamente, el enfoque de este proyecto se centra en las aplicaciones móviles disponible en dispositivos *Android*. La aplicación en su versión inicial *MVP* llevará el nombre de "*Meditake*", una combinación de las palabras "*Medicine*"(medicamento) y "*take*"(toma), ambas provenientes del inglés.

4.1.1. Terminología

En la Tabla 4.1 se describen los términos vinculados con la ontología del sistema:

Tabla 4.1: Términos ontológicos de la aplicación

Témino	**Descripción**
User (Usuario)	Individuo que utilizará las funcionalidades completas de la aplicación
Medicine (Medicamento)	Fármaco o tratamiento a seguir
Reminder (Recordatorio)	Aviso recurrente para seguir el tratamiento indicado

Por otro lado, los términos técnicos que forman parte del sistema son especificados en la Tabla 4.2.

Tabla 4.2: Términos técnicos de la aplicación

Término	Descripción
Dose Type (tipo de dosis)	Formato de presentación del fármaco
Email	Dirección de correo electrónico que el usuario utilizará para recibir informes
Report (Informe)	Resumen de los medicamentos tomados por el usuario

4.1.2. Modelo Dominio

Tras la introducción de la terminología específica del sistema, se presenta un diagrama de clases (ver Figura 4.1) conforme a las convenciones clásicas de UML [5] para facilitar la comprensión del contexto del sistema.

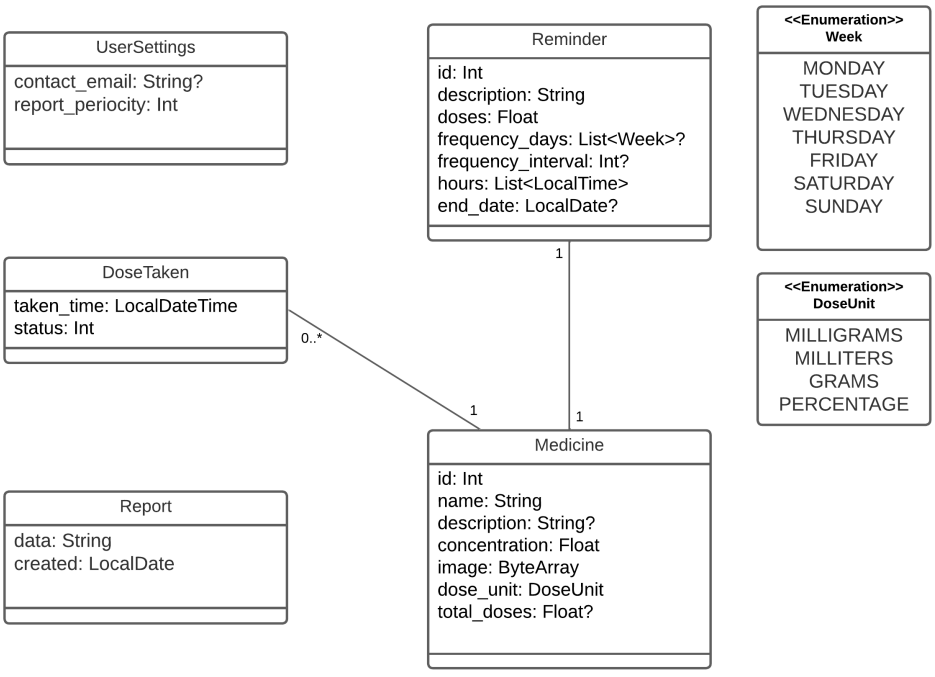

Figura 4.1: Diagrama del modelo de dominio

A continuación se explica la función de cada clase y se definen los atributos que las componen. La anotación ? indica la posibilidad de que el atributo sea nulo en el sistema.

- **Clase *User***: Datos de configuración del usuario (ver Tabla 4.3).

Tabla 4.3: Atributos de la clase *UserSetting*

Atributos de *User*	**Descripción**
contact_email *String?*	Correo electrónico utilizado para el envío de informes a una persona auxiliar
report_periocity *Int*	Utilizado para indicar el tipo de periodicidad de los informes

- **Enumerado *DoseType***: Enumerado que contiene los diferentes formatos y medidas de dosis disponibles para las tomas; utilizado para crear un medicamento.
- **Enumerado *WeekDays***: Enumerado que contiene las todos los días de una semana; utilizado para crear un recordatorio.
- **Clase *Reminder***: Clase que contiene los datos de un recordatorio (ver Tabla 4.4).

Tabla 4.4: Atributos de la clase *Reminder*

Atributos de *Reminder*	**Descripción**
id *Int*	Número entero único que identifica al recordatorio
description *String*	Utilizado para añadir información extra al recordatorio que será mostrada al usuario en la toma
dose *Float*	Cantidad de medicamento que el usuario debe ingerir en la toma
frequency_days *List<Week>?*	Días de la semana en el se realiza el aviso de la toma de medicamento
frequency_interval *Int?*	Periodo de días que establece la frecuencia para la notificación de la toma de medicamentos.
hours *List<LocalTime>*	Momentos del día en el se mostrará el aviso de toma
end_date *DateDate?*	Fecha de fin del tratamiento relacionado con el recordatorio

- **Clase *Medicine***: Clase que contiene los datos de un medicamento utilizado en un tratamiento (ver Tabla 4.5).

Tabla 4.5: Atributos de la clase *Medicine*

Atributos de *Medicine*	**Descripción**
id *Int*	Número entero único que identifica el medicamento
name *String*	Nombre del medicamento
description *String?*	Descripción utilizada para añadir información adicional sobre el medicamento registrado
concentration *Float*	Cantidad de medicamento que contiene cada dosis de este
image *ByteArray*	Imagen del medicamento
dose_unit *DoseUnit*	Indica la medida o formato de presentación del medicamento
total_doses *Float?*	Cantidad de medicamento total disponible para las tomas

- **Clase *DoseTaken***: Clase de persistencia que contiene la información de cada toma, utilizada para la creación de reportes (ver Tabla 4.6).

Tabla 4.6: Atributos de la clase *DoseTaken*

Atributos de *DoseTaken*	**Descripción**
taken_time *LocalDateTime*	Fecha y hora en la que se ha realizado la toma del medicamento
status *Int*	Entero utilizado para indicar el estado de la toma, asignando el valor 0 para indicar que no ha sido tomada y el valor 1 indicando su toma correcta

- **Clase *Report***: Clase de persistencia que contiene la los reportes generados (ver Tabla 4.7).

Tabla 4.7: Atributos de la clase *Report*

Atributos de *Report*	**Descripción**
data *String*	Compilación de todas las tomas con un formato que mejora la legibilidad del informe.
created *LocalDate*	Fecha de generación del informe.

4.1.3. Restricciones del Sistema

En la Tabla 4.8 se presentan algunas de las restricciones que podrían limitar ciertas funcionalidades ofrecidas por la aplicación.

Tabla 4.8: Restricciones del sistema

Restricción	Descripción
Almacenamiento del dispositivo	Existe la posibilidad de que el dispositivo no cuente con suficiente memoria disponible para almacenar la información de los medicamentos y recordatorios.
Notificaciones activas	Existe la posibilidad de que el usuario tenga desactivadas las notificaciones del sistema o de la aplicación, lo que podría resultar en la no recepción de los avisos de toma.
Permisos de acceso a cámara y alarma	La negación por parte del usuario de los permisos de acceso a la cámara y las alarmas del dispositivo podría tener un impacto negativo en la creación de medicamentos y recordatorios.

4.1.4. Límites del sistema

La representación de los límites de la aplicación se realizará mediante un diagrama de contexto, donde se presenta la jerarquía de los actores que interactuarán con el producto.

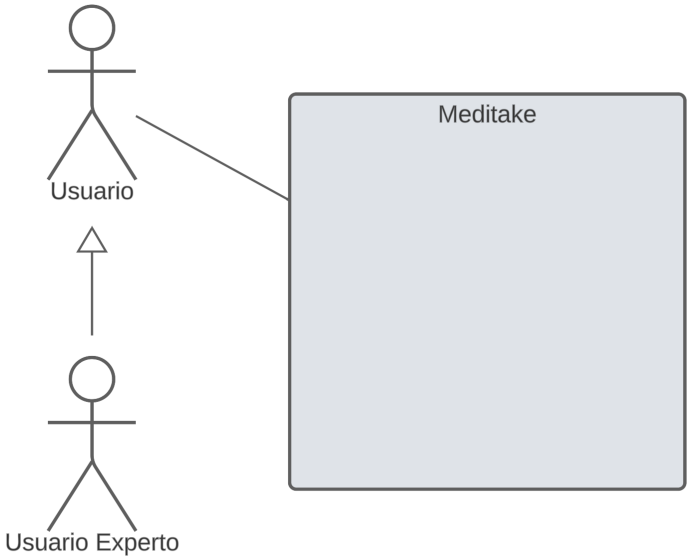

Figura 4.2: Diagrama de contexto

El diagrama de la Figura 4.2 muestra claramente que únicamente dos actores interactúan con el sistema:

1. **Usuario**: usuario inexperto que usará las funciones básicas de la aplicación, es decir, la toma del medicamento.
2. **Usuario experto**: que empleará tanto las funciones del usuario inexperto como la configuración de los recordatorios y la visualización de informes.

4.1.5. Características del sistema

A continuación, se detallan las características finales que debe poseer el aplicativo, presentadas a través de *casos de uso*.

Cada uno de los requisitos funcionales ha sido marcado con un identificador RFXX, el cual será utilizado para hacer referencia a ellos en sus respectivas especificaciones.

4.1.5.1. Casos de uso principales

En la Figura 4.3 se muestran los casos de uso estarán presentes en la *pantalla principal* y *notificación de toma*.

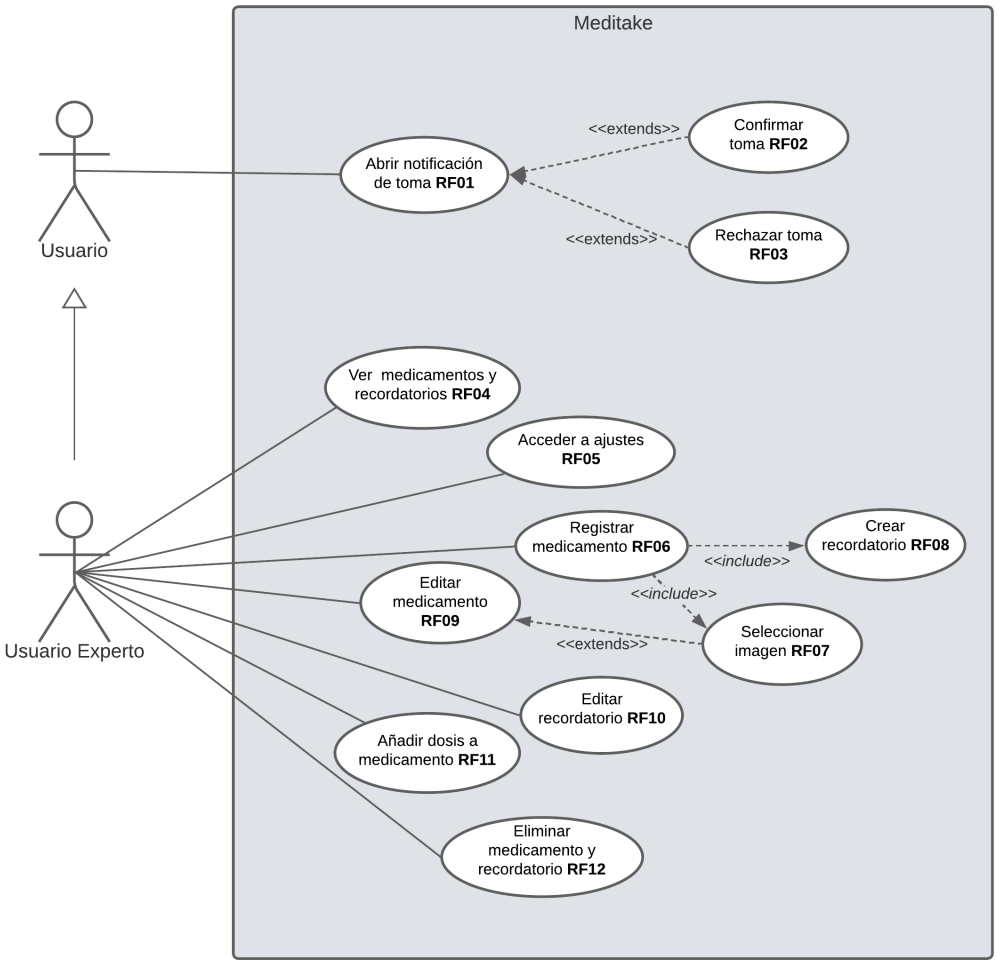

Figura 4.3: Diagrama de casos de uso principales

4.1.5.2. Casos de uso de ajustes

En la Figura 4.4 se presentan aquellos visibles en la *pantalla de ajustes*.

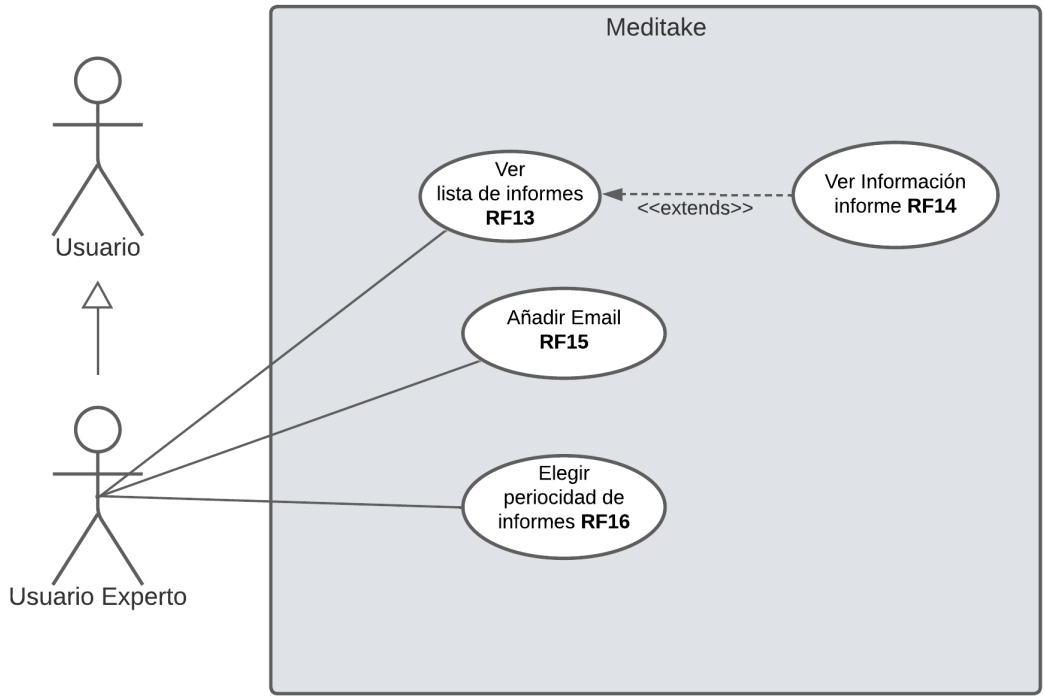

Figura 4.4: Diagrama de casos de uso ajustes

4.1.6. Requisitos funcionales

Actor: **Usuario**	**RF01 - Abrir notificación de toma**
Descripción	Se despliega un aviso en pantalla a través de una notificación del sistema, señalando la presencia de un medicamento o tratamiento que aún no ha sido tomado, y el usuario debe pulsar sobre ella
Precondiciones	Debe haber al menos un recordatorio creado y activo
Entradas	Ninguna
Salidas	Pantalla de toma de medicamento
Postcondiciones	Se muestra pantalla de toma de medicamento
Flujo principal	1. Pulsar sobre la notificación flotante.

Actor: **Usuario**	**RF02 - Confirmar toma**
Descripción	Después de leer las instrucciones presentadas en la pantalla, el usuario confirma la toma correcta del medicamento
Precondiciones	Debe haber al menos un recordatorio activo y el usuario debe haber pulsado sobre la notificación
Entradas	Ninguna
Salidas	Ninguna
Postcondiciones	Se registra la toma correcta del medicamento
Flujo principal	1. Pulsar sobre botón de confirmación de toma

Actor: **Usuario**	**RF03 - Rechazar toma**
Descripción	Después de leer las instrucciones presentadas en la pantalla, el usuario confirma que no ha tomado el medicamento
Precondiciones	Debe haber al menos un recordatorio activo y el usuario debe haber pulsado sobre la notificación
Entradas	Ninguna
Salidas	Ninguna
Postcondiciones	Se registra la omisión de la toma del medicamento
Flujo principal	1. Pulsar sobre botón de omisión de toma

Actor: **Usuario Experto**	**RF04 - Ver medicamentos y recordatorios**
Descripción	Se presenta un menú de navegación que permite acceder a información detallada de los medicamentos, como imágenes o dosis restantes, y simultáneamente ofrece información del recordatorio asignado, como las horas de toma
Precondiciones	Ninguna
Entradas	Ninguna
Salidas	Listado de medicamento y recordatorios
Postcondiciones	Se muestran los medicamentos y recordatorios registrados
Flujo principal	1. Abrir la aplicación.
Flujo secundario	1. Finalizar el flujo de registrar medicamento.

Actor: **Usuario Experto**	**RF05 - Acceder a ajustes**
Descripción	Se presenta el conjunto de configuraciones y opciones de personalización para los informes
Precondiciones	Ninguna
Entradas	Ninguna
Salidas	Listado de configuraciones e informes
Postcondiciones	Se muestra el listado de configuraciones e informes
Flujo principal	1. Presionar sobre los 3 puntos en la parte superior. 2. Pulsar sobre el botón de *Ajustes*.

Actor: **Usuario Experto**	**RF06 - Registrar medicamento**
Descripción	Permite registrar un medicamento que va a ser usado en un tratamiento
Precondiciones	Ninguna
Entradas	Nombre, descripción, concentración, medida o formato, imagen, dosis totales disponibles
Salidas	Ninguna
Postcondiciones	Medicamento registrado, disponible en el listado de medicamentos
Flujo principal	1. Acceder al listado de medicamentos 2. Presionar añadir nuevo medicamento 3. Introducir datos necesarios 4. Pulsar *Continuar*

Actor: **Usuario Experto**	**RF07 - Seleccionar imagen**
Descripción	Permite añadir una imagen a un medicamento
Precondiciones	Estar en la pantalla de registro de medicamento
Entradas	Ninguna
Salidas	Imagen
Postcondiciones	Una imagen queda seleccionada para el medicamento
Flujo principal	1. Acceder a crear o editar medicamento 2. Presionar sobre *añadir imagen* 3. Seleccionar imagen desde la galería del dispositivo

Actor: **Usuario Experto**	**RF08 - Crear recordatorio**
Descripción	Permite crear un recordatorio para la toma del medicamento
Precondiciones	Ninguna
Entradas	Instrucciones de toma, dosis a tomar, días de la semana o frecuencia, hora de la toma, fecha fin del tratamiento
Salidas	Ninguna
Postcondiciones	Recordatorio creado, activo y disponible en el listado de recordatorios
Flujo principal	1. Acceder a añadir recordatorio 2. Pulsar *continuar* 3. Introducir los datos necesarios 4. Pulsar *guardar*

Actor: **Usuario Experto**	**RF09 - Editar medicamento**
Descripción	Permite editar la información de un medicamento ya creado
Precondiciones	Medicamento registrado y disponible en el listado de medicamentos
Entradas	Nombre, descripción, concentración, medida o formato, imagen, dosis totales disponible
Salidas	Ninguna
Postcondiciones	Medicamento actualizado y disponible en el listado de medicamentos
Flujo principal	1. Acceder al listado de medicamentos 2. Pulsar sobre los tres puntos de un medicamento ya registrado 3. Presionar sobre *editar medicamento* 4. Introducir o actualizar datos necesarios 5. Pulsar *guardar*

Actor: **Usuario Experto**	**RF10 - Editar recordatorio**
Descripción	Permite editar un recordatorio ya creado para la toma de medicina
Precondiciones	Recordatorio creado y disponible en la lista de recordatorios
Entradas	Instrucciones de toma, dosis a tomar, días de la semana o frecuencia, hora de la toma, fecha fin del tratamiento
Salidas	Ninguna
Postcondiciones	Recordatorio editado, activo y disponible en el listado de recordatorios
Flujo principal	1. Acceder al listado de Recordatorios 2. Pulsar sobre los tres puntos de un medicamento ya registrado 3. Presionar sobre *editar recordatorio* 4. Introducir o editar los datos necesarios 5. Pulsar *guardar*

Actor: **Usuario Experto**	**RF11 - Añadir dosis a medicamento**
Descripción	Permite añadir rápidamente una nueva cantidad de dosis disponible a un medicamento que ya está registrado
Precondiciones	Medicamento registrado y disponible en la lista de medicamentos
Entradas	Número de dosis
Salidas	Ninguna
Postcondiciones	Medicamento actualizado y disponible en el listado de medicamentos
Flujo principal	1. Acceder al listado de medicamentos 2. Pulsar sobre el botón de añadir dosis 3. Introducir número de dosis 4. Pulsar *guardar*

Actor: **Usuario Experto**	**RF12 - Eliminar medicamento y recordatorio**
Descripción	Permite eliminar un medicamento y todos su recordatorios
Precondiciones	Medicamento y recordatorio registrados y disponibles en la lista de medicamentos
Entradas	Ninguna
Salidas	Ninguna
Postcondiciones	Medicamento y recordatorio eliminado
Flujo principal	1. Acceder al listado de medicamentos 2. Pulsar sobre los tres puntos de un medicamento ya registrado 3. Presionar sobre *eliminar*

Actor: **Usuario Experto**	**RF13 - Ver lista de informes**
Descripción	Posibilita la visualización de todos los informes, presentando la fecha en que fueron generados
Precondiciones	Ninguna
Entradas	Ninguna
Salidas	Listado de informes
Postcondiciones	Se muestra la pantalla de informes
Flujo principal	1. Acceder a ajustes 2. Presionar sobre la sección de *informes*

Actor: **Usuario Experto**	**RF14 - Ver información de reporte**
Descripción	Facilita la visualización detallada del informe, dividiendo la información por tomas e indicando el medicamento, así como si fue o no tomado
Precondiciones	Es obligatorio que exista al menos un informe generado
Entradas	Ninguna
Salidas	Información del informe
Postcondiciones	Lista detallada de las tomas
Flujo principal	1. Acceder a ajustes 2. Presionar sobre la sección de *informes* 3. Pulsar sobre un informe de la lista

Actor: **Usuario Experto**	**RF15 - Añadir Email**
Descripción	Permite añadir un email de contacto para el envío de informes periódicos
Precondiciones	Ninguna
Entradas	Email de contacto
Salidas	Ninguna
Postcondiciones	Configuración actualizada
Flujo principal	1. Acceder a ajustes 2. Introducir datos 3. Pulsar *guardar*

Actor: **Usuario Experto**	**RF16 - Elegir periocidad de informes**
Descripción	Permite cambiar la periocidad para el envío de reportes
Precondiciones	Ninguna
Entradas	Tipo de periocidad
Salidas	Ninguna
Postcondiciones	Configuración actualizada
Flujo principal	1. Acceder a ajustes 2. Introducir datos 3. Pulsar *guardar*

4.2 Prototipo de pantallas

Antes de iniciar el proceso de desarrollo, se tomó la decisión de confeccionar un conjunto de *MockUps* que servirán como orientación para el diseño de las pantallas. En ellos se señalan cada uno de los casos de uso mencionados anteriormente, numerados del 1 al 16.

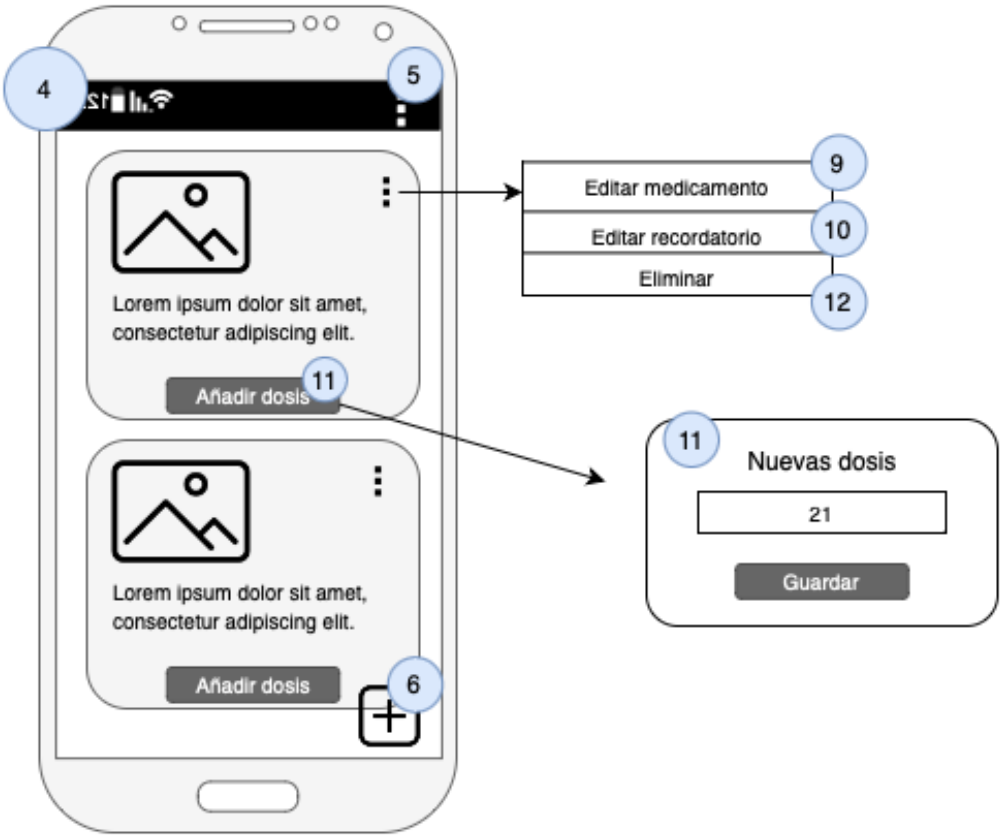

Figura 4.5: *Mockup* Pantalla listado de medicamentos

La pantalla principal o pantalla de inicio, se presenta como un listado de medicamentos y recordatorios creados (ver Figura 4.5). Desde esta pantalla, se habilita la gestión de dichos elementos, permitiendo acciones como la edición o eliminación. Asimismo, desde esta pantalla, se brinda acceso a las configuraciones y a la creación de nuevos medicamentos y recordatorios.

Figura 4.6: *Mockup* Pantalla de creación y edición de medicamentos y recordatorios

La Figura 4.6 exhibe las pantallas dedicadas a la creación de nuevos medicamentos y recordatorios, respectivamente. En ambas, se muestran todos los campos que deben completarse. Cabe destacar que el diseño de estas pantallas es idéntico para la edición de medicamentos y recordatorios.

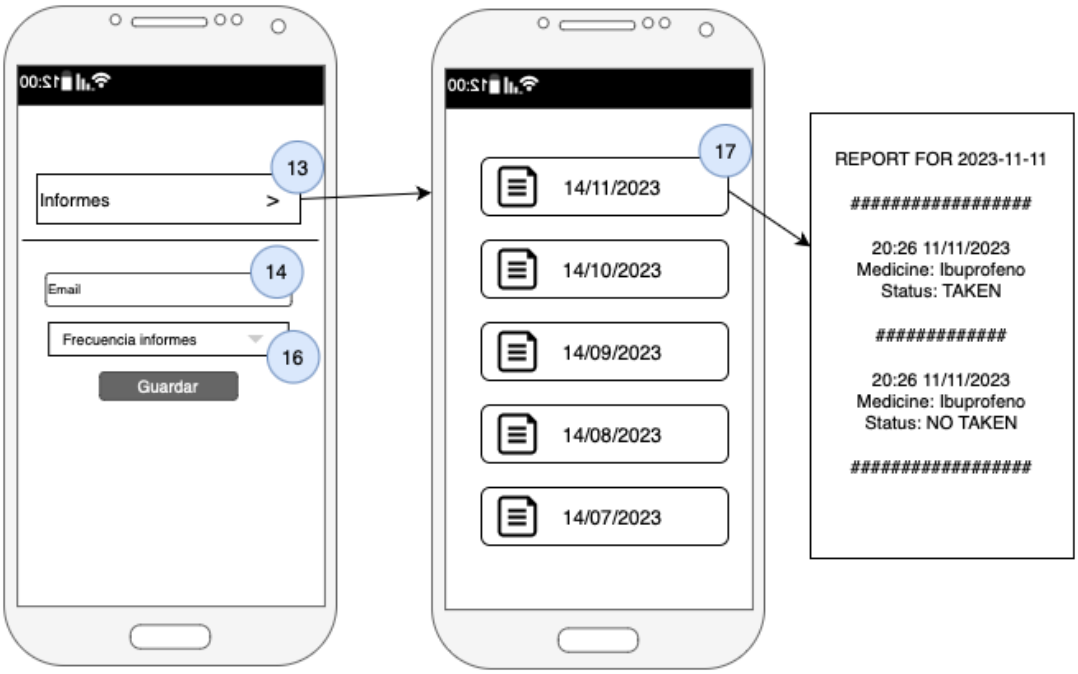

Figura 4.7: *Mockup* Pantallas de ajustes y reportes

La Figura 4.7 muestra la pantalla destinada a la configuración de los informes, donde se visualizan los informes generados y se permite ajustar la dirección de correo para el envío, así como la periodicidad de dicho envío.

Figura 4.8: *Mockup* Pantalla de toma de medicamento

En el *mockup* representado en la Figura 4.8, se puede visualizar la notificación recibida al momento de tomar un medicamento, junto con la información asociada a este. También se presentan las diversas opciones disponibles para marcar.

CAPÍTULO 5
Diseño de la solución

5.1 Tecnología empleada

Como se ha explicado en secciones previas, el desarrollo de este *MVP* se materializará a través de una aplicación nativa destinada a la plataforma *Android* [19]. Algunas de las ventajas que nos ofrecen las aplicaciones nativas frente a multiplataforma son:

- **Rendimiento**: las aplicaciones nativas están optimizadas para el sistema operativo específico y pueden aprovechar al máximo los recursos del dispositivo, lo que resulta en un mayor rendimiento.
- **Características del dispositivo**: estas aplicaciones tienen acceso completo a todas las características y funcionalidades del dispositivo, como la cámara, el GPS, alarma, etc.
- **Seguridad**: el desarrollo móvil nativo posibilita la utilización de mecanismos y algoritmos del sistema para cifrar y almacenar datos, implementar certificados de seguridad y limitar el acceso a la información.
- **Interfaces de usuario**: al aprovechar los componentes UI/UX nativos del sistema, los desarrolladores evitan posibles problemas de compatibilidad entre diferentes dispositivos.

5.1.1. Kotlin

Kotlin es un lenguaje de programación estático orientado a objetos [6]. Este lenguaje proporciona una sintaxis y conceptos afines a los de otros lenguajes, como *C#*, *Java* y *Scala*. Para el desarrollo de aplicaciones en *Android*, los lenguajes de programación más comúnmente utilizados son *Kotlin* y *Java*. Las principales ventajas del uso de *Kotlin* frente a *Java* son:

- **Curva de aprendizaje**: *Kotlin* facilita el aprendizaje y la revisión del código, ya que su sintaxis es clara y accesible.
- **Seguridad contra referencias nulas**: este lenguaje aborda el problema de las referencias nulas (null) de manera más segura. El sistema de tipos de *Kotlin* ayuda a prevenir los errores relacionados con referencias nulas durante la compilación.
- **Interoperabilidad con Java**: la capacidad de usar bibliotecas de *Java* en proyectos *Kotlin* y viceversa simplifica el proceso de migración de proyectos *Java* a *Kotlin*.

El uso de *Corrutinas* (ver Figura 5.1) es una de las ventajas más destacadas de este lenguaje de programación, ya que optimizan la programación asíncrona y posibilitan el manejo de tareas de larga duración que podrían afectar el rendimiento del proceso principal de la aplicación.

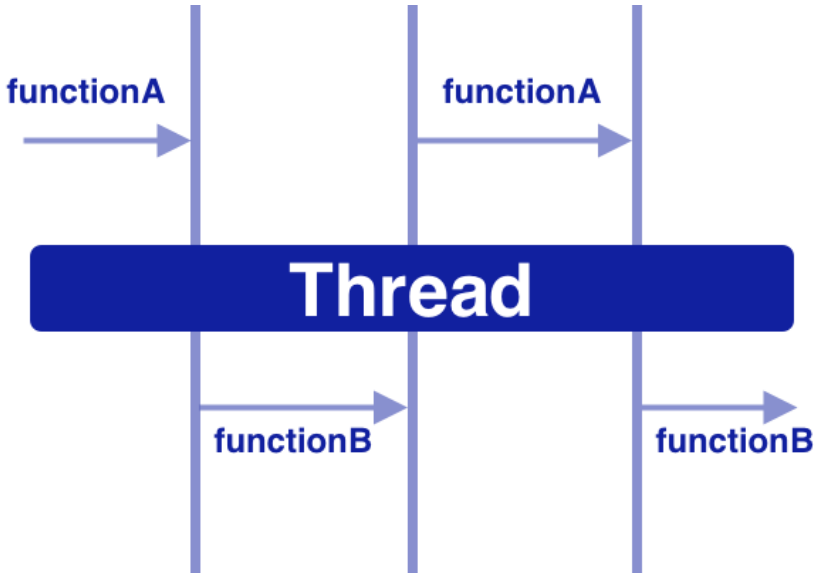

Figura 5.1: Flujo de Corrutinas [7]

Kotlin ha sido seleccionado por *Google* como el lenguaje de programación principal para el desarrollo *Android*, garantizando su respaldo. Dado que todos los nuevos desarrollos están mayormente orientados a *Kotlin*, se recomienda iniciar nuevos proyectos utilizando este lenguaje [9]. De esta manera, y teniendo en cuenta las ventajas previamente mencionadas, hemos decidido emplear *Kotlin* como lenguaje de programación en este proyecto.

5.1.2. Room

Room es la biblioteca de persistencia que proporciona una capa de abstracción sobre SQLite en Android, nos posibilita conservar datos localmente y la emplearemos para preservar toda la información del usuario, desde los medicamentos y recordatorios registrados hasta la configuración de la aplicación.

Room nos permite llevar a cabo operaciones en la base de datos sin la necesidad de escribir consultas SQL directamente e incluye características como el soporte para consultas LiveData [11]. También nos permite una manejo sencillo de las entidades de base de datos y un mayor control sobre las versiones y migraciones de base de datos.

Room se divide en tres componentes principales:

- **Clase de la base de datos**: aquí se encuentran las clases encargadas de la base de datos, funcionando como el punto de acceso central para conectar con los datos persistentes de la aplicación.
- Las **entidades de datos**: representan las tablas de la base de datos de la aplicación, definiendo la estructura y relaciones fundamentales entre los datos.
- Los **objetos de acceso a datos** (*DAOs*): ofrecen métodos para consultar, actualizar, insertar y borrar datos en la base de datos.

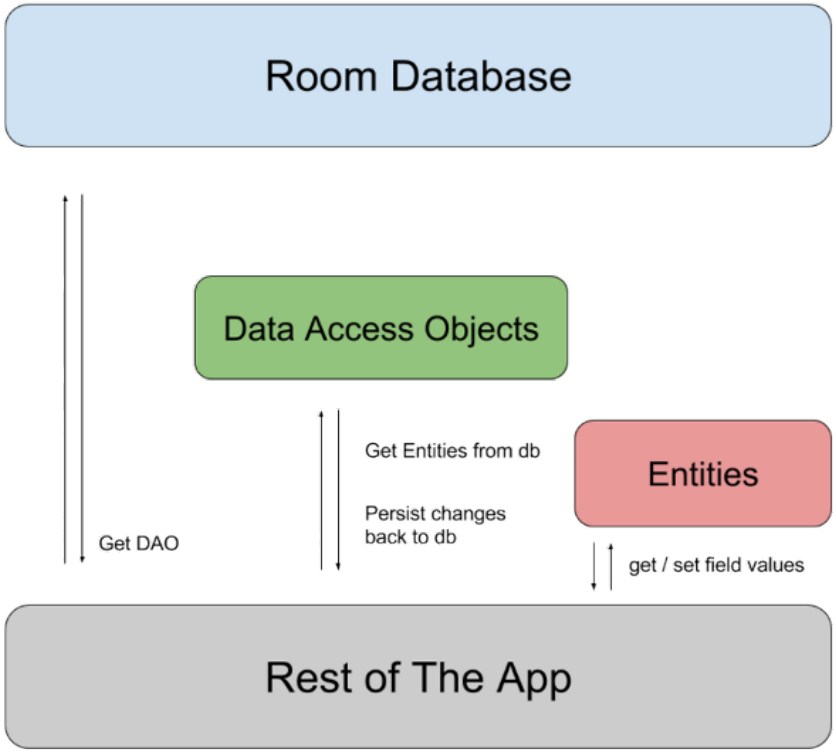

Figura 5.2: Diagrama de los componentes de *Room*

Como podemos observar en la Figura 5.2 la clase de base de datos, dentro de sus funciones, nos proporciona instancias de los DAOs. Estos, a su vez, nos permiten recuperar los datos en forma de instancias de entidad de datos, las cuales podemos utilizar para actualizar filas de las tablas de la base de datos.

Dado que la aplicación se utilizará principalmente sin conexión a internet, en nuestro proyecto definiremos los tres componentes esenciales de *Room* definidos anteriormente.

5.1.3. Android Studio

La herramienta seleccionada como entorno de desarrollo integrado (*IDE*) es *Android Studio* [8], desarrollada por JetBrains. Este *IDE* está especialmente concebido para el desarrollo de aplicaciones *Android* y cuenta con numerosos atajos, herramientas de depuración de archivos y la capacidad de integrar simuladores de *Android* para ejecutar y probar aplicaciones.

5.1.4. Git

Para el control de versiones, se empleará *Git* [10] y el proyecto se almacenará en un repositorio en *GitHub*. Además, adoptaremos la metodología *GitHub Flow* (ver Figura 5.3), que consta de:

- La rama principal, también llamada *master*, que contendrá las versiones finales del aplicativo.
- Ramas *feature* para cada nueva funcionalidad a desarrollar, dividiéndolas según sea necesario para incluir una parte de la arquitectura.

- Ramas "bugfix"para la solución rápida de errores encontrados en las versiones finales.

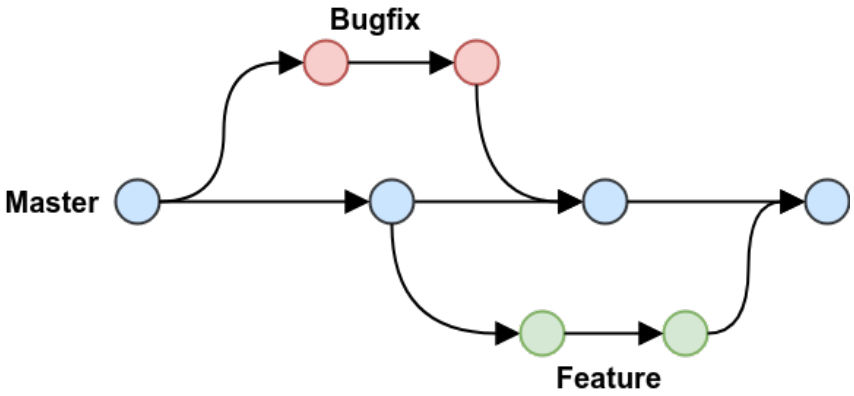

Figura 5.3: Diagrama del funcionamiento de *GitHub Flow*

Esta metodología posibilita entregas de valor constantes del aplicativo y es compatible con herramientas de integración continua (*CI/CD*) [17].

5.2 Arquitectura del sistema

Una vez establecidas las tecnologías que se emplearán en el desarrollo de la solución, es crucial definir una arquitectura que se adapte a los requisitos establecidos.

Se necesita una arquitectura de software sencilla, dado que el desarrollo se realizará por una persona. Además, debe estar bien establecida para facilitar la escalabilidad del sistema, así como las fases de prueba y mantenimiento.

Teniendo en cuenta lo mencionado anteriormente, la elección recae en una arquitectura de estilo *Clean Architecture* [12], cuyo objetivo es abstraer al máximo los diversos bloques o componentes que integrarán el sistema.

5.2.1. MVVM

En línea con la arquitectura mencionada previamente, hemos elegido implementar la arquitectura Model-View-ViewModel (MVVM), la cual se adapta de manera más adecuada a los *frameworks* de *Android*. Esta arquitectura separa la interfaz de usuario de la lógica de negocio y los modelos, podemos observar esta división en la Figura 5.4. Gracias a esta arquitectura podremos obtener:

- Componentes que cumplen con una única funcionalidad claramente definida.
- Flujo de datos unidireccional en todas las capas de la aplicación.
- Eliminación de código repetido.
- Código más claro y fácil de entender.

Complementaremos esta arquitectura con *Casos de Uso*, que definen las operaciones que el usuario puede llevar a cabo con nuestra aplicación, y *repositorios*, que aíslan las fuentes de datos.

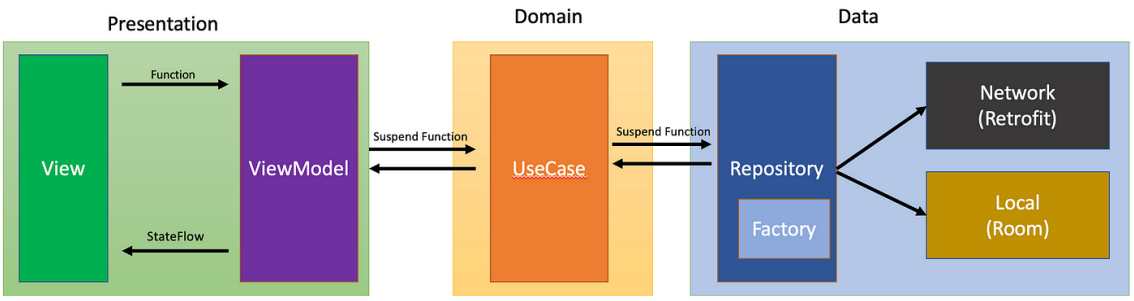

Figura 5.4: Diagrama de *Model-View-ViewModel* con *Casos de Uso* y *repositorios*

A continuación se definen las diferentes capas:

- **Capa de datos** (*data*): encargada de gestionar el flujo de datos y recuperar información de distintas fuentes, como bases de datos o archivos. Actúa como una capa de abstracción y protege a los datos de la interacción directa.
- **Capa de dominio** (*domain*): busca transformar los datos brutos recibidos de la capa de datos en un formato procesado que sea cómodo y manejable para la capa de presentación.
- **Capa de presentación** (*presentation*): actúa como interfaz entre el usuario y la aplicación. La capa de presentación gestiona las entradas del usuario y presenta los resultados correspondientes. En esta arquitectura la capa de presentación se divide en:
  - *vista* (*view*), la cual debe de permanecer simple y sin lógica de negocio
  - *modelo de vista* (*view model*), que se encarga de la comunicación y gestión de los datos recibidos por la capa de dominio.

CAPÍTULO 6
Implementación de la solución

6.1 Estructura del proyecto

En el proceso de desarrollo de software, es fundamental contar con una estructura de carpetas bien definida que se ajuste a la arquitectura seleccionada. Por lo tanto, siguiendo la arquitectura y el patrón *MVVM*, se ha establecido el árbol de directorios mostrado en la Figura 6.1.

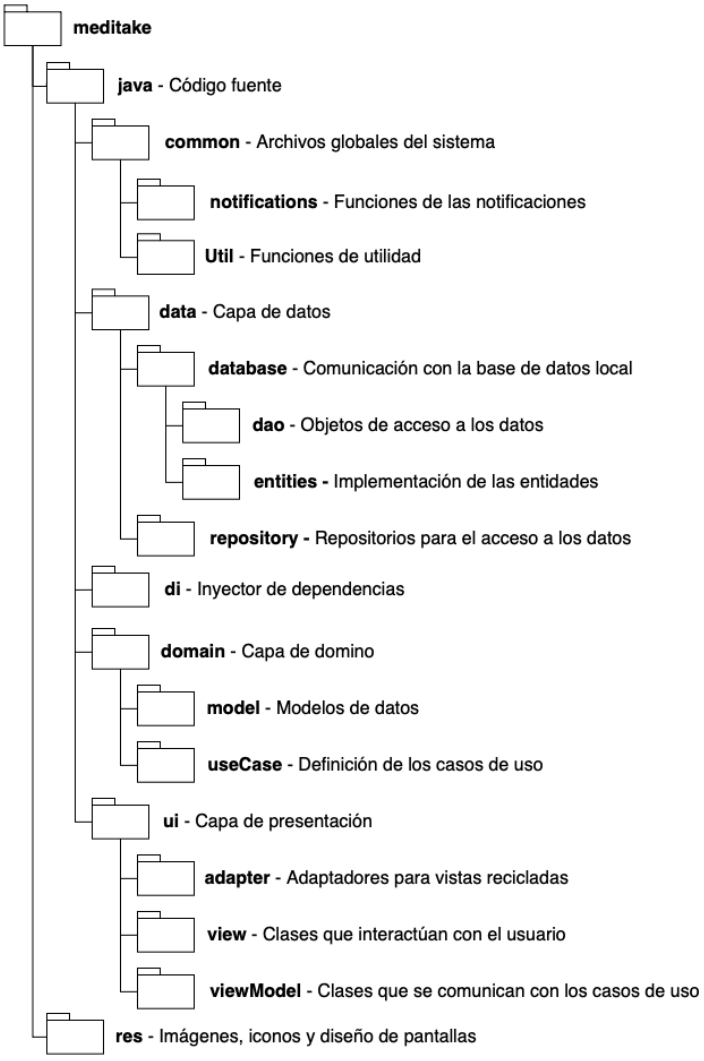

Figura 6.1: Estructura de directorios de *Meditake*

6.2 Implementación de la aplicación

En esta sección, se describirá de manera concisa pero detallada el proceso que hemos seguido para el desarrollo de las diversas partes del proyecto. Este apartado se completa con fragmentos de código que podemos encontrar en el Apéndice A

6.2.1. Implementación de la capa de datos

Comenzaremos con el desarrollo de la *capa de datos* de la funcionalidad. En esta fase, definiremos los modelos y los objetos de acceso a los datos utilizando clases de datos *Entity* e interfaces *Dao*, respectivamente. Además, procederemos con la implementación de los *repositorios*.

Las clases *Entity* definirán todos los atributos de las tablas SQL respectivamente.

Las interfaces *Dao* se encargarán de la comunicación directa con la base de datos local *Room* e implementará todas las consultas SQL necesarias para gestionar los datos.

Los *repositorios* se apoyarán sobre las interfaces *Dao* para ofrecer métodos limpios y claros para el acceso de datos; estos recibirán los objetos desde base de datos y los transformarán a los modelos del dominio para ser tratados.

6.2.2. Implementación de la capa de dominio

Comenzamos el desarrollo de la *capa de dominio* con la definición de los modelos que serán empleados por la *capa de presentación*. Utilizaremos *clases de datos* en este proceso.

Dentro de esta sección, se incorpora la definición de los enumerados *DoseUnit* y *Week*, los cuales se emplearán para la creación de medicamentos y recordatorios, respectivamente.

A continuación, se establecen los *casos de uso* que serán empleados por los *viewModels* para llevar a cabo la funcionalidad definida. Estos *casos de uso* pueden recibir parámetros, como en el caso de guardar un medicamento, o devolver datos en forma de modelo, como al recuperar los recordatorios creados.

6.2.3. Implementación de la capa de presentación

Como se ha mencionado en capítulos anteriores, en esta capa definiremos las *views* también conocidas como *Activity* [13] en *Android*. Se ocuparán de la operatividad de las pantallas, desde poblarlas de datos hasta observar las acciones del usuario. Desarrollaremos una instancia para cada pantalla presente en la aplicación.

Por cada *view* debemos implementar su correspondiente *viewModel* que se encargará de recibir las acciones del usuario y transformarlas en peticiones a la base de datos a través de los casos de uso.

Para la correcta comunicación entre estas dos clases las *views* se subscribirán a los objetos de *viewModel* y, cuando estos cambien, actuarán en consecuencia.

Asimismo, es esencial determinar los diseños y componentes que las pantallas del aplicativo incorporarán. Para ello se definen archivos *.xml* que contienen la estructura de componentes de cada pantalla, podemos observar esta estructura en la Figura 6.2.

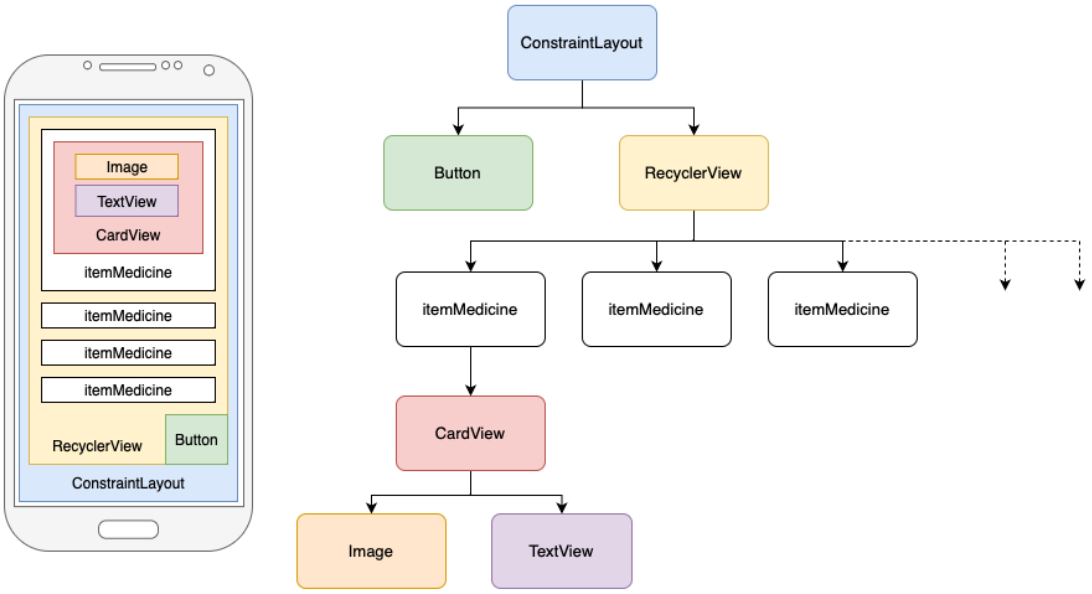

Figura 6.2: Estructura de los componentes de la Pantalla de listado de medicinas

A través de estos archivos, las *views* atienden a los distintos componentes y establecen las acciones que ejecuta el usuario.

6.2.4. Inyección de dependencias

Con el objetivo de asegurar la accesibilidad en cada instancia del aplicativo a todas las capas y clases mencionadas, se utilizará un Inyector de Dependencias.

Para ello utilizaremos la anotación *@Inject* propia de *Android Studio* para la inyección de dependencias. Esta anotación facilita la incorporación de las clases y métodos que mencionemos en nuestro *constructor*. Además creamos la clase *RoomModule* en el directorio *di* para permitir la inyección de dependencias de las interfaces *Dao* sobre los *repositorios* y el acceso a la base de datos local. Esta práctica es muy recomendable, ya que posibilita la reutilización del código, simplifica el proceso de refactorización y optimiza las pruebas.

6.2.5. Tests

Los *test* son esenciales en todo desarrollo de software, ya que posibilitan la identificación de errores en el software y la verificación de su cumplimiento con los requisitos especificados.

En consecuencia, se llevaron a cabo múltiples *tests* unitarios destinados a certificar funciones individuales, los cuales fueron implementados sobre los métodos fundamentales del aplicativo, específicamente en aquellos orientados al tratamiento de datos.

Se han desarrollado pruebas unitarias sobre los métodos de los casos de uso. En estos *test* se realizan llamadas a los métodos con diferentes entradas de datos, se simulan la respuestas obtenidas de base de datos y se comprueba que la respuesta final de los métodos sea la esperada.

Al finalizar de cada funcionalidad se realizaron *tests end-to-end* (e2e) manuales, en los que se seguía el flujo de un usuario real para así poder determinar el correcto funcionamiento de estas nuevas funcionalidades.

6.3 Sprints

En este apartado se expondrá el trabajo realizado en cada *sprint*, especificando los tiempos invertidos, los resultado obtenidos, las pruebas realizadas y los problemas afrontados, si los hubo.

Como se mencionó en el Capítulo 2, los *sprints* tuvieron una duración de dos semanas cada uno, invirtiendo aproximadamente 65 horas por *sprint*.

6.3.1. Sprint 1

En este *sprint* se invirtió un total de 63 horas, de las cuales 25 fueron dedicadas a aprender los principios del lenguaje de programación *Kotlin* y *Room*. Se utilizaron 10 horas para la creación de la base de datos y las inyecciones de dependencias necesarias para el correcto funcionamiento de *Room*. Las horas restantes fueron invertidas en la creación de las entidades de medicamentos y recordatorios; también se elaboró las pantallas necesarias para su creación.

Se realizaron los primeros test unitarios sobre los métodos de los casos de uso. Se siguió el flujo del usuario y se crearon medicamentos y recordatorios. De esta forma se comprobó el correcto funcionamiento y almacenamiento de los datos.

El principal problema de este *sprint* fue el desconocimiento de estas nuevas tecnologías y fue solventado mediante el estudio en artículos y foros especializados.

6.3.2. Sprint 2

Este *sprint* tomó un total de 64 horas, durante las primeras 26 horas se elaboró la pantalla que muestra el resumen de todos los medicamentos y recordatorios creados. Se invirtieron otras 24 horas en la creación de la pantalla sobre la que se indica la correcta administración del medicamento y se programó la entidad de datos para guardar esta información. El resto de horas se invirtieron en la programación de la notificación recibida en cada recordatorio.

Se crearon medicamentos y recordatorios para comprobar el correcto funcionamiento de la pantalla de resumen. Se configuró un recordatorio y se esperó a recibir la notificación de este; posteriormente, se entró en la pantalla de toma y se confirmó la toma. Se comprobó que todos los cambios fueron reflejados en la base de datos del dispositivo.

Durante este *sprint* se encontró una limitación con las alarmas de Android, ya que no permite configurarlas para un periodo específico o para días de la semana concretos. Finalmente, se optó por la creación de varias alarmas para un mismo recordatorio.

6.3.3. Sprint 3

La mitad de las horas del *sprint* fueron invertidas en la creación de la funcionalidad que permite editar o eliminar medicamentos y recordatorios. Durante la otra mitad del *sprint* se creó la pantalla de ajustes y la entidad correspondiente para guardar la configuración del usuario. Finalmente, en este *sprint* se invirtieron 62 horas

Se siguió el flujo del usuario, se crearon, editaron y eliminaron varios medicamentos y recordatorios comprobando que estos cambios se reflejaban en la base de datos. También se instaló la aplicación desde cero y se comprobó que la configuración inicial de ajustes se creaba correctamente.

6.3.4. Sprint 4

En este último *sprint* se invirtieron un total de 65 horas. Las primeras 35 horas se centraron en la creación de la entidad necesaria para guardar en base de datos los informes. También se creó la funcionalidad encargada de generar los informes. El resto del tiempo se invirtió en la creación de la pantalla encargada de mostrar los informes generados. También se implementó el servicio encargado de enviar los informes al correo electrónico proporcionado en ajustes.

Para probar estas funcionalidades se realizaron varias tomas de medicamentos y se estableció una frecuencia de informes y un correo electrónico. Se comprobó que los informes se generaran y almacenaran en base de datos y que el correo hubiera sido enviado.

Varios servicios para el envío de correos fueron probados ya que la mayoría de ellos no eran compatibles con la aplicación. Finalmente, se creó una cuenta de *Google* fue creada para el envío de correos.

6.4 Resultados de la implementación

Para el desarrollo de las interfaces se han utilizado las recomendaciones y los componentes de *Material Design* [20] y de esta forma se ha conseguido una consistencia a través de todas las ventanas del aplicativo e incluso con otra aplicaciones nativas *Android* del mercado. A continuación, se presentarán imágenes finales del aplicativo.

En las Figuras 6.4 y 6.3 observamos la primera pantalla con la que interactúa el usuario: el listado de medicamentos (vacío, si no se ha registrado ninguno). Desde esta pantalla se puede eliminar medicamentos registrados y añadir nuevas dosis, navegar hacia la creación de nuevos medicamentos o a la edición de estos.

Figura 6.3: Pantalla de inicio vacía

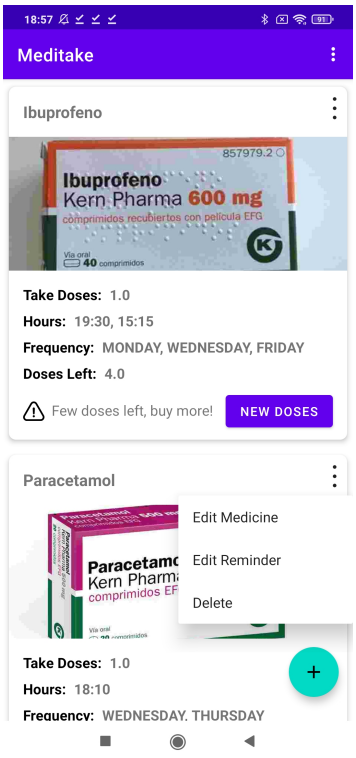

Figura 6.4: Pantalla de inicio

En la Figura 6.5 se muestra la pantalla en la cual se pueden registrar o editar medicamentos, mientras que la Figura 6.6 muestra la pantalla encargada de la edición y registro de recordatorios.

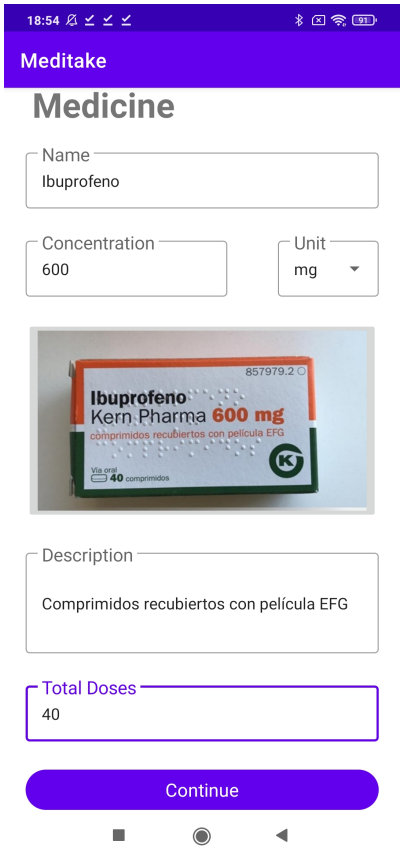

Figura 6.5: Pantalla de registro y edición de medicamentos

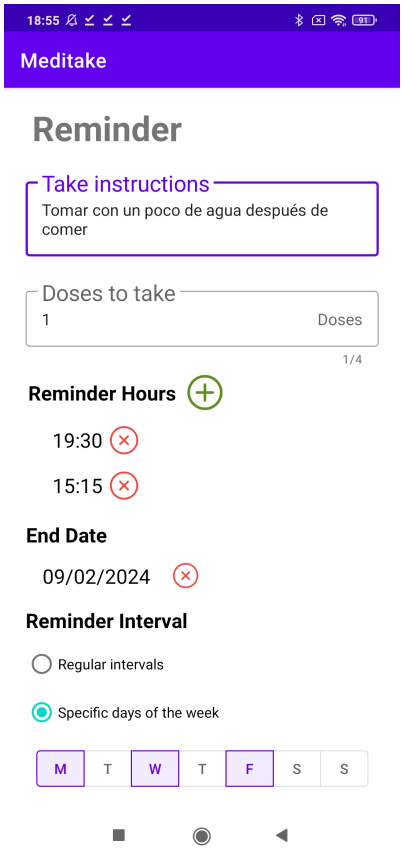

Figura 6.6: Pantalla de registro y edición de recordatorios

En la Figura 6.7 se observa la pantalla de ajustes del usuario en la que se puede actualizar el correo electrónico para el envío de informes y la periocidad de estos. Desde esta pantalla se puede navegar a la visualización de los reportes generados.

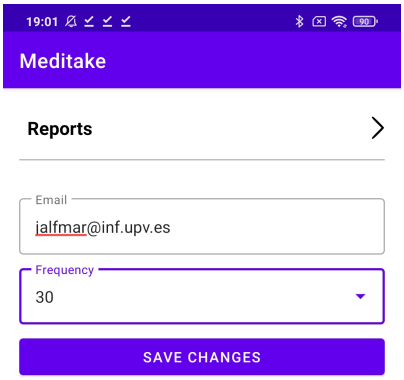

Figura 6.7: Pantalla de ajustes

La Figura 6.8 muestra la notificación recibida para indicar al usuario que debe tomar un medicamento. La pantalla asociada a la toma de medicamento se muestra en la Figura 6.9, donde se observan las instrucciones necesarias para la toma y se permite indicar si el medicamento ha sido tomado o no.

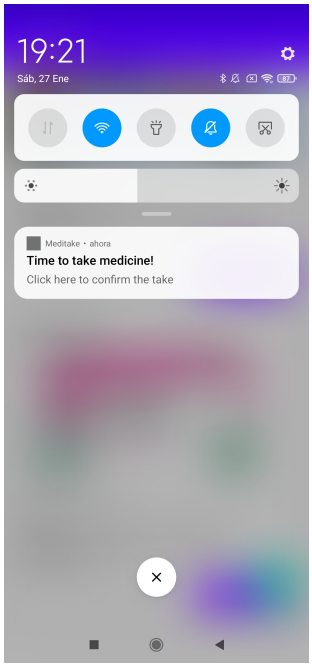

Figura 6.8: Notificación de toma

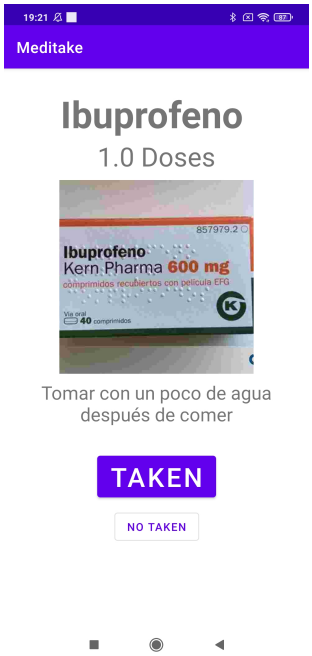

Figura 6.9: Pantalla de toma

La Figura 6.10 muestra el listado de todos los informes generados, desde donde podemos navegar a la pantalla que muestra en detalle el informe (ver Figura 6.11)

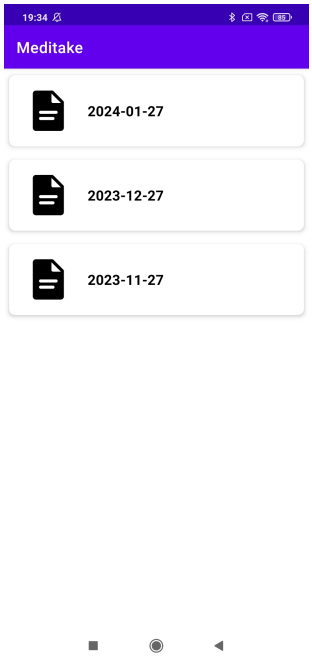

Figura 6.10: Pantalla listado de informes

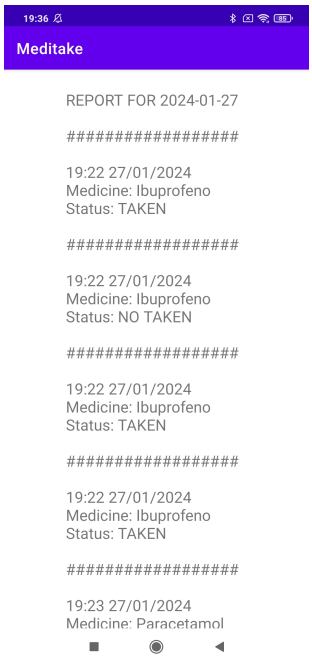

Figura 6.11: Pantalla de informe

CAPÍTULO 7
Despliegue y mantenimiento

7.1 Integración continua

Tal como se explicó en capítulos anteriores, hemos optado por utilizar la metodología *ágil SCRUM*, la cual nos facilita la implementación de la *Integración Continua*, puesto que al concluir cada interacción de las tareas, estaremos en condiciones de realizar un despliegue. Además, nos adherimos a la metodología *GitHub Flow* y, gracias a sus entregas de valor constantes, podemos implementar la *Integración Continua* (*CI/CD*).

Junto con estas metodologías, podemos hacer uso de herramientas como *Jenkins* [14] para llevar a cabo el despliegue de *CI/CD*. Esta herramienta nos permite:

- Realizar la compilación de los proyectos de Android y ejecutar las pruebas de manera automática.
- Implementar la automatización del despliegue de manera automática tras realizar un *merge* en la rama *master*, en cualquier entorno seleccionado.
- Hacer *rollback* en cualquier momento, es decir volver a una versión anterior de la aplicación.

Gracias a esta herramienta, en caso de fallos durante la compilación o las pruebas, se bloqueará el despliegue de los cambios realizados y se enviará una notificación al desarrollador informando de los errores detectados.

7.2 Mantenimiento

Es crucial resaltar la importancia de continuar adaptando el proyecto incluso después de su publicación. El mantenimiento [16] es esencial por los siguientes motivos:

- **Compatibilidad**: la actualización de la aplicación asegura su compatibilidad con las versiones más recientes de *Android*, ya que se incorporan cambios y mejoras en el sistema operativo. Esto previene posibles problemas de funcionamiento en dispositivos actualizados.
- **Nuevas características**: la actualización constante ayuda a mantener la aplicación relevante y competitiva, al mismo tiempo que se pueden atender las nuevas solicitudes de los usuarios.

- **Corrección de errores**: dado que los errores y problemas técnicos pueden presentarse en cualquier momento, las actualizaciones nos ofrecen la oportunidad de corregir estos problemas y mejorar la estabilidad del aplicativo.

De acuerdo con estos motivos, el aplicativo se actualizará de manera regular para corregir errores y agregar nuevas funcionalidades.

7.3 Publicación

En la fase final, una vez alcanzado el MVP, la última etapa implica hacer que el aplicativo sea accesible a los usuarios. Para lograrlo, llevaremos a cabo la publicación de la aplicación en la tienda principal para *Android*, *Google Play Store*. A continuación, describiremos los pasos a seguir:

1. Creación de una cuenta como desarrollador en Google
2. Etiquetar el contenido de la aplicación como apto para todos los públicos.
3. Establecer versión inicial del aplicativo (1.0.0)
4. Generar y subir el *APK* [15] de la aplicación
5. Enviar el aplicativo a revisión, que una vez aprobado será publicado automáticamente por *Google*.

En cada actualización del aplicativo, se debe incrementar la versión y seguir los pasos a partir del número 4.

CAPÍTULO 8
Conclusiones y trabajos futuros

8.1 Relación con los estudios cursados

Es crucial resaltar la conexión existente entre los conocimientos adquiridos durante los estudios y aquellos aplicados en este proyecto.

Para la selección de la metodología de software en este proyecto, fue necesario realizar un estudio profundo de las diversas opciones disponibles. Las asignaturas de *Proceso del software* y *Proyecto de Ingeniería del software* jugaron un papel fundamental al facilitar el proceso de selección de la metodología, ya que introdujeron muchas de las metodologías relevantes. La elección final se inclinó hacia *SCRUM*, que ya había sido implementada en la asignatura de *Proyecto de Ingeniería del Software (PIN)*.

Puesto que se trata de un proyecto individual, los conocimientos adquiridos en la asignatura *PIN* han resultado fundamentales para la organización y el estudio de los recursos necesarios para llevar a cabo este proyecto. Asimismo, gracias al estudio y desarrollo de una aplicación móvil en esta asignatura, hemos encontrado más fácilmente el *framework* y las tecnologías ideales para llevar a cabo este proyecto.

La asignatura de *Análisis y Elicitación de Requisitos* desempeñó un papel fundamental al identificar y analizar los requisitos que conforman el aplicativo, siguiendo el estándar *UML* visto en la asignatura.

La aplicación de patrones de diseño y la comprensión de los principios de código limpio, estudiados en la asignatura *Diseño del Software*, han posibilitado la elección de una arquitectura óptima y la obtención de un código más legible y funcional.

8.2 Conclusiones

Como resumen final, al mirar hacia atrás y evaluar los objetivos establecidos al inicio del proyecto, se puede afirmar que se ha cumplido el objetivo principal. Se alcanzó con éxito el objetivo de desarrollar un *MVP* de una aplicación móvil que gestiona recordatorios para la toma de medicamentos y controla el recuento restante de estos de manera efectiva.

Se ha alcanzado con éxito la implementación de una base de datos local para la persistencia de los datos del usuario utilizando *Room*, aplicando las mejores prácticas de *Android* estudiadas a lo largo del desarrollo del proyecto.

Además, se logró la implementación exitosa de una arquitectura basada en *Clean Architecture*, específicamente aplicando el patrón *MVVM*. Todo ello se logró mediante el estudio, análisis y aplicación de una metodología ágil; *SCRUM*.

Por último, es importante resaltar el logro de los objetivos personales, ya que durante el desarrollo de este proyecto se llevó a cabo un análisis exhaustivo de numerosos artículos, medios digitales y libros académicos. Se aplicaron buenas prácticas, patrones de diseño y el enfoque de *Clean Code*.

Además, se ha dedicado tiempo al estudio, aprendizaje y aplicación de un nuevo lenguaje de programación, *Kotlin*, siguiendo sus recomendaciones de uso por parte de sus desarrolladores y las comunidades de expertos.

8.3 Trabajos futuros

Aunque el desarrollo de este proyecto se ha centrado en alcanzar los objetivos y requisitos previamente mencionados, a lo largo de su evolución han surgido algunos puntos e ideas que podrían ser incorporados en versiones futuras.

En la actualidad, el uso de esta aplicación está limitado al entorno local, lo cual inicialmente no representa un problema, ya que su objetivo es la población de edad avanzada. Sin embargo, para adaptarla a todos los usuarios, es necesario gestionar la información en la red, posibilitando el uso de diferentes dispositivos desde cualquier lugar del mundo.

Para lograrlo, podríamos considerar la introducción de una base de datos en la red mediante *Firebase* [18]. Al integrarla con los servicios de *Google*, los usuarios podrían iniciar sesión con sus cuentas y así mantener todos sus datos desde cualquier dispositivo móvil.

También existen múltiples oportunidades para implementar mejoras estéticas en las interfaces, tales como la inclusión de animaciones o iconos personalizados. Mejorar la estética del aplicativo se presenta como un requisito fundamental para atraer nuevos usuarios.

Otra opción a considerar en el futuro sería la reconstrucción de la aplicación para dispositivos iOS mediante *XCode* y *Swift* [21].

Como última consideración, se podría explorar la integración del aplicativo con servicios externos como *Amazon Pharmacy* [22]. En este escenario, cuando el usuario tenga pocas dosis restantes de un medicamento, recibiría automáticamente una notificación y se realizaría un pedido del medicamento a través de esta plataforma, eliminando la necesidad de estar pendiente de adquirir los medicamentos. Actualmente este servicio no está disponible en España pero esta función podría estar incluida en aquellos países que lo permitan.

Bibliografía

[1] S. Roger Pressman. *Ingeniería del software: Un enfoque práctico*. Editorial McGraw-Hill, 3.ª Edición, Pag. 26-30.

[2] Ruwanthi Ranasinghe. *V — Model in Software Testing*. Medium, 2021. URL: https://medium.com/@ruwanthiranasinghe1996/v-model-in-software-testing-9b196bba5a45.

[3] Manuel Trigas Gallego. *Metodología Scrum*. Universitat Oberta de Catalunya, 2012, Pag. 32-41. URL: https://openaccess.uoc.edu/bitstream/10609/17885/1/mtrigasTFC0612memoria.pdf.

[4] Trello Inc. *What is Trello?*, consultado el 22 octubre de 2023. URL: https://support.atlassian.com/trello/docs/what-is-trello/.

[5] Object Management Group. *OMG Unified Modeling Language, Superstructure*, agosto, 2011. Consultar a https://www.omg.org/spec/UML/2.4.1/Superstructure/PDF.

[6] Developer Android. *Descripción general de Kotlin*, consultado el 30 de octubre de 2023. URL: https://developer.android.com/kotlin/overview.

[7] Amit Shekhar. *Mastering Kotlin Coroutines*, 2022. URL: https://amitshekhar.me/blog/kotlin-coroutines.

[8] Developer Android. *Introducción a Android Studio*, consultado el 15 de diciembre de 2023. URL: https://developer.android.com/studio/intro.

[9] TechCrunch. *Kotlin is now Google's preferred language for Android app development*, consultado el 10 de diciembre de 2023. URL: http://social.techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/.

[10] Patrick Porto. *4 branching workflows for Git*. Medium, consultado el 27 de enero de 2024. URL: https://medium.com/@patrickporto/4-branching-workflows-for-git-30d0aaee7bf.

[11] Developer Android. *Descripción general de LiveData*, consultado el 15 de diciembre de 2023. URL: https://developer.android.com/topic/libraries/architecture/livedata.

[12] Bharath. *The Clean Architecture*. Medium, 2022. URL: https://betterprogramming.pub/the-clean-architecture-beginners-guide-e4b7058c1165.

[13] Developer Android. *Introducción a las actividades*, consultado el 20 de diciembre de 2023. URL: https://developer.android.com/guide/components/activities/intro-activities.

[14] Sentrio. *Introducción a Jenkins: ¿qué es, para qué sirve y cómo funciona?*, 2022 URL: https://sentrio.io/blog/que-es-jenkins/.

[15] Actualizatec. *¿Qué es un APK?*, consultado el 20 de diciembre de 2023. URL: https://actualizatec.com/blog/que-es-un-apk/.

[16] CSE2305 Object-Oriented Software Engineering. *Software Maintenance and Re-engineering*, consultado el 24 de diciembre de 2023. URL: https://users.monash.edu/~jonmc/CSE2305/Topics/13.25.SWEng4/html/text.html.

[17] Atlassian. *Continuous integration vs. delivery vs. deployment*, consultado el 28 de diciembre de 2023. URL: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment.

[18] Yanina Muradas. *Qué es Firebase: Conoce la plataforma de Google*, 2021. URL: https://openwebinars.net/blog/que-es-firebase-de-google/.

[19] James Chen. *Android Operating System (OS): Definition and How It Works*, junio, 2022. URL: https://www.investopedia.com/terms/a/android-operating-system.asp.

[20] Google. *What's Material?*, consultado el 12 de enero de 2024. URL: https://m3.material.io/get-started.

[21] Apple Inc. *Xcode 15*, consultado el 28 de diciembre de 2023. URL: https://developer.apple.com/xcode/.

[22] Amazon. *Getting started with Amazon Pharmacy*, consultado el 28 de diciembre de 2023. URL: https://pharmacy.amazon.com/how-it-works.

APÉNDICE A
Código

En este anexo se muestran los fragmentos de código expuestos de ejemplo y extraídos de la fase de implementación.

A.1 Capa de datos

Este apartado contiene el código correspondiente a la entidad, objeto de acceso a datos y repositorio para *Medicine*.

```
@Entity(tableName = "medicine_table")
data class MedicineEntity(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "id") val id: Int = 0,
    @ColumnInfo(name = "name") val name: String,
    @ColumnInfo(name = "description") val description: String?,
    @ColumnInfo(name = "concentration") val concentration: Float,
    @ColumnInfo(name = "image") val image: ByteArray,
    @ColumnInfo(name = "dose_unit") val dose_unit: DoseUnit,
    @ColumnInfo(name = "total_doses") val total_doses: Float?
)
```

```
@Dao
interface MedicineDao {
    @Query("SELECT * FROM medicine_table")
    suspend fun getAllMedicines(): List<MedicineEntity>

    @Query("SELECT * FROM medicine_table WHERE id = :medicineId")
    suspend fun getMedicine(medicineId: Int): MedicineEntity

    @Update
    suspend fun update(medicine: MedicineEntity)

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(medicine: MedicineEntity): Long

    @Query("DELETE FROM medicine_table WHERE id = :medicineId")
    suspend fun deleteMedicine(medicineId: Int)
}
```

```
class MedicineRepository @Inject constructor(
    private val medicineDao: MedicineDao
) {
    suspend fun getMedicines(): List<Medicine> {
        val response: List<MedicineEntity> = medicineDao.getAllMedicines()
        return response.map { it.toDomain() }
```

```
    }

    suspend fun getMedicine(medicineId: Int): Medicine {
        val response: MedicineEntity = medicineDao.getMedicine(medicineId)
        return response.toDomain()
    }

    suspend fun insertMedicine(medicine: MedicineEntity): Long {
        return medicineDao.insert(medicine)
    }

    suspend fun updateMedicine(medicine: MedicineEntity) {
        return medicineDao.update(medicine)
    }

    suspend fun deleteMedicine(medicineId: Int) {
        medicineDao.deleteMedicine(medicineId)
    }
}
```

A.2 Capa de domino

Este apartado contiene el código correspondiente al modelo *Reminder*, caso de uso *getReminder* y enumerados *DoseUnit* y *Week*.

```
data class Reminder(
    var id: Int = 0,
    val medicine_id: Int,
    var description: String,
    var doses: Float,
    var frequency_days: List<Week>?,
    var frequency_interval: Int?,
    var hours: List<LocalTime>,
    var end_date: LocalDate?,
)
```

```
enum class DoseUnit(val value: String) {
    MILLIGRAMS("mg"),
    MILLILITERS("ml"),
    GRAMS("g"),
    PERCENTAGE("%");
}
```

```
enum class Week(val value: String) {
    MONDAY("Mon"),
    TUESDAY("Tue"),
    WEDNESDAY("Wed"),
    THURSDAY("Thu"),
    FRIDAY("Fri"),
    SATURDAY("Sat"),
    SUNDAY("Sun");
}
```

```
class GetReminderUseCase @Inject constructor(
    private val repository: ReminderRepository
){
    suspend operator fun invoke(medicineId: Int): Reminder {
        return repository.getReminderByMedicineId(medicineId)
    }
}
```

A.3 Capa de presentación

Obsevamos como la clase *ListMedicineActivity* se subscribe al objeto *medicines* e *isLoading* para actualizar el adaptador de la lista una vez obtenidas todas la medicinas desde la clase *ListMedicineViewModel*

```
class ListMedicineActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        listMedicineViewModel.medicines.observe(
            this,
            Observer {
                medicinesList = it.toMutableList()
                adapter.updateMedicines(it.toMutableList())
            })
        listMedicineViewModel.isLoading.observe(this, Observer {
            binding.loading.isVisible = it
        })
    }
}
```

```
class ListMedicineViewModel @Inject constructor(
    private val getMedicines: GetMedicines,
) : ViewModel() {
    val medicines = MutableLiveData<List<Medicine>>()
    val isLoading = MutableLiveData<Boolean>()

    init {
        viewModelScope.launch {
            isLoading.postValue(true)
            val result = getMedicines()

            if (!result.isNullOrEmpty()) {
                medicines.postValue(result)
                isLoading.postValue(false)
            }
        }
    }
}
```

A.4 Inyección de dependencias

Observamos la clase *RoomModule* que nos permite el acceso a la base de datos local *Room* y los diferentes objetos de acceso de datos (*Daos*)

```
object RoomModule {

    private const val DATABASE_NAME = "meditake_database"

    @Singleton
    @Provides
    fun provideRoom(@ApplicationContext context: Context) =
        Room.databaseBuilder(context, Database::class.java, DATABASE_NAME).
            build()

    @Singleton
    @Provides
    fun provideUserDao(db: Database) = db.getUserDao()
```

```
13
14      @Singleton
15      @Provides
16      fun provideMedicineDao(db: Database) = db.getMedicineDao()
17
18      @Singleton
19      @Provides
20      fun provideReminderDao(db: Database) = db.getReminderDao()
21
22      @Singleton
23      @Provides
24      fun provideDoseTakenDao(db: Database) = db.getDoseTakenDao()
25
26      @Singleton
27      @Provides
28      fun provideReportDao(db: Database) = db.getReportDao()
29  }
```

A.5 Tests

Observamos la clase de *test* para el caso de uso de guardar la configuración del usuario. En esta clase, se pueden observar dos pruebas que verifican que la configuración se guarda de manera diferente dependiendo de si existía previamente o no.

```
1   class SaveSettingsUseCaseTest {
2       @Mock
3       private lateinit var mockRepository: SettingsRepository
4
5       @InjectMocks
6       private lateinit var saveSettingsUseCase: SaveSettingsUseCase
7
8       @Before
9       fun setUp() {
10          MockitoAnnotations.initMocks(this)
11      }
12
13      @Test
14      fun save_existing_settings() {
15          val settings = Settings(id = 1, email = "test@example.com",
                reportFrequency = 30)
16          saveSettingsUseCase(settings)
17          verify(mockRepository).updateSettings(settings.toDatabase())
18      }
19
20      @Test
21      fun save_non_existing_settings() {
22          val settings = Settings(id = 0, email = "test@example.com",
                reportFrequency = 30)
23          saveSettingsUseCase(settings)
24          verify(mockRepository).insertSettings(settings.toDatabase())
25      }
26  }
```

APÉNDICE B
Objetivos de Desarrollo Sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

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

La aplicación desarrollada se enfoca directamente en la gestión de múltiples tratamientos médicos simultáneos, con la finalidad de evitar confusiones entre ellos. Está principalmente orientada a personas mayores. Esta aplicación tiene una gran relación con el objetivo de desarrollo sostenible *3. Salud y bienestar* ya que contribuye directamente a mejorar la salud de las personas mayores al garantizar que tomen sus medicamentos de manera oportuna. La aplicación no solo recuerda la toma de medicamentos, sino que también contribuye a la prevención de problemas de salud. Al asegurar que los usuarios tomen sus medicamentos de manera regular, se reducen los riesgos de complicaciones y hospitalizaciones, lo que mejora la salud general de los usuarios.

La aplicación tiene relación con la reducción de las desigualdades ya que esta proporciona una herramienta de gestión de medicamentos que es fácilmente accesible para

personas mayores, independientemente de su ubicación geográfica o nivel socioeconómico. Esta aplicación proporciona recordatorios personalizados, lo que asegura que todos, independientemente de sus limitaciones físicas o cognitivas, puedan beneficiarse de la gestión efectiva de la medicación.

Con respecto a la ODS Ciudades y Comunidades Sostenibles la aplicación facilita la gestión de la medicación en el hogar, y ayuda a reducir la carga sobre los servicios de salud locales y sistemas hospitalarios. También al permitir que las personas mayores gestionen su salud de manera independiente, se contribuye a la creación de comunidades más sostenibles, donde estas personas pueden continuar viviendo en sus hogares durante más tiempo sin depender de instalaciones de atención a largo plazo.

En parte esta aplicación contribuye al ODS Alianzas para lograr objetivos ya que la posibilidad de enviar informes a personas externas de apoyo fomenta la colaboración entre los usuarios y los profesionales de la salud o familiares. Esta colaboración también permite adaptar los planes de atención médica según las circunstancias específicas de cada usuario. Esta capacidad de compartir informes crea una red de apoyo para las personas mayores y también alivia la carga de los cuidadores al tener acceso a toda esta información.