DSM 04L

UNIVERSIDAD DON BOSCO

TRABAJO DE INVESTIGACION: MVVM

Profesor: Alexander Alberto Siguenza Campos

Alumno: Ruben Oswaldo Escalante Amaya EA132101

29/04/2023

índice

Introducción

En los últimos años, la arquitectura de software se ha vuelto cada vez más importante en el desarrollo de aplicaciones móviles y web. MVVM es un patrón de arquitectura de software que se ha vuelto muy popular en el desarrollo de aplicaciones móviles y web, debido a su capacidad para separar la lógica de presentación de la lógica de negocio, lo que permite una mejor organización y mantenimiento del código.

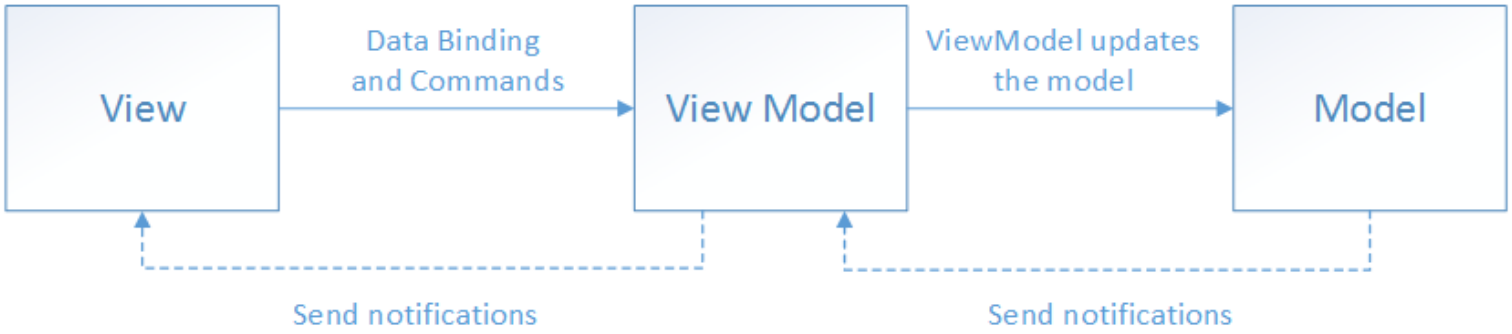
En este patrón, el Modelo representa los datos y la lógica de negocio de la aplicación, la Vista es la parte de la aplicación que se encarga de mostrar la información al usuario, y el ViewModel es el intermediario entre el Modelo y la Vista. El ViewModel se encarga de procesar y transformar los datos del Modelo para que la Vista los pueda mostrar al usuario de manera adecuada, y maneja las interacciones del usuario con la Vista.

¿Qué es MVVM?

MVVM es un patrón arquitectónico de software utilizado en el desarrollo de aplicaciones de software. Las siglas MVVM significan Modelo-Vista-VistaModelo.

En MVVM, el Modelo representa los datos y la lógica de negocios de la aplicación, la Vista representa la interfaz de usuario (UI) que muestra los datos al usuario, y el ViewModel es un intermediario entre el Modelo y la Vista. El ViewModel se encarga de actualizar la Vista con los datos del Modelo y de manejar la lógica de la interfaz de usuario.

El patrón MVVM se utiliza comúnmente en aplicaciones de software que utilizan el paradigma de programación orientado a objetos y también se utiliza con frecuencia en el desarrollo de aplicaciones móviles y aplicaciones web. MVVM permite una separación clara de la lógica de negocio y la presentación de la aplicación, lo que facilita la reutilización del código y el mantenimiento de la aplicación.



Características del patrón de arquitectura MVVM

Dentro de las característica y propiedades relevantes del patrón de arquitectura MVVM, se encuentra su capacidad para separar de forma limpia la presentación de una aplicación determinada y la lógica del negocio de su interfaz de usuario. Cabe resaltar que esta separación entre interfaz de usuario y lógica de la app contribuye a abordar múltiples tipos de inconvenientes de desarrollo, lo que facilita los procesos de prueba, mantenimiento y evolución del sistema.

Uno de los beneficios de usar este patrón de arquitectura es que permite que los desarrolladores creen pruebas unitarias para el Model View y el modelo, sin que sea necesario el uso de la vista.

Además de esto, con el patrón de arquitectura mvvm, los encargados del diseño y desarrollo de aplicaciones pueden ser capaces de trabajar de manera simultánea e independiente,

cada uno en sus componentes durante los procesos de la app. Así pues, mientras los diseñadores tienen la posibilidad de enfocarse en la vista, los desarrolladores pueden hacerse cargo de los componentes de la vista y del modelo de vista.

Elementos MVVM

Hay tres componentes principales en el patrón MVVM: el modelo, la vista y el modelo de vista. Cada uno sirve para un propósito distinto.

Además de comprender las responsabilidades de cada componente, también es importante comprender cómo interactúan. En un nivel alto, la vista "conoce" el modelo de vista y el modelo de vista "conoce" el modelo, pero el modelo no es consciente del modelo de vista y el modelo de vista no es consciente de la vista. Por lo tanto, el modelo de vista aísla la vista del modelo y permite que el modelo evolucione independientemente de la vista.

Ventajas

Las ventajas de usar el patrón MVVM son las siguientes:

- Si una implementación de modelo existente encapsula la lógica de negocios existente, puede ser difícil o arriesgada cambiarla. En este escenario, el modelo de vista actúa como adaptador para las clases de modelo y evita que realice cambios importantes en el código del modelo.
- Los desarrolladores pueden crear pruebas unitarias para el modelo de vista y el modelo, sin usar la vista. Las pruebas unitarias para el modelo de vista pueden ejercer exactamente la misma funcionalidad que la vista.
- La interfaz de usuario de la aplicación se puede rediseñar sin tocar el modelo de vista y el código del modelo, siempre que la vista se implemente completamente en XAML o C#. Por lo tanto, una nueva versión de la vista debe funcionar con el modelo de vista existente.
- Los diseñadores y desarrolladores pueden trabajar de forma independiente y simultánea en sus componentes durante el desarrollo. Los diseñadores pueden centrarse en la vista, mientras que los desarrolladores pueden trabajar en el modelo de vista y los componentes del modelo.

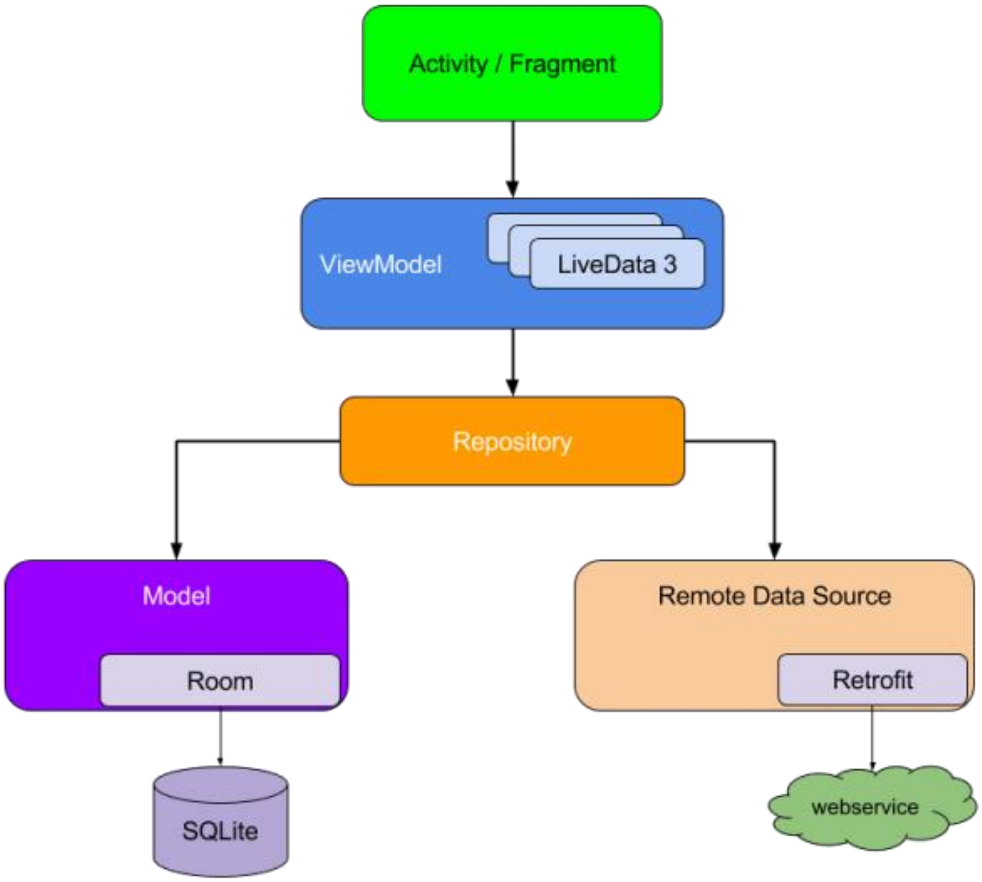
Conceptos claves

La clave para usar MVVM de forma eficaz radica en comprender cómo factorizar el código de la aplicación en las clases correctas y cómo interactúan las clases.

View: La vista es responsable de definir la estructura, el diseño y la apariencia de lo que ve el usuario en la pantalla.

Viewmodel: El modelo de vista implementa propiedades y comandos a los que la vista puede enlazar datos y notifica a la vista los cambios de estado a través de eventos de notificación de cambios. Las propiedades y comandos que proporciona el modelo de vista definen la funcionalidad que ofrece la interfaz de usuario, pero la vista determina cómo se va a mostrar esa funcionalidad.

Modelo: Las clases de modelo son clases no visuales que encapsulan los datos de la aplicación. Por lo tanto, el modelo se puede considerar como que representa el modelo de dominio de la aplicación, que normalmente incluye un modelo de datos junto con la lógica de validación y negocios. Algunos ejemplos de objetos de modelo incluyen objetos de transferencia de datos (DTO), Objetos CLR antiguos sin formato (POCOs) y objetos de entidad y proxy generados.



Aplicación creada:

La aplicación para demostrar MVVM se ha creado para presentarla como proyecto de catedra para esta materia en la teoría, así que aún falta trabajo por hacerle, pero una funcionalidad CRUD usando MVVM si tiene.

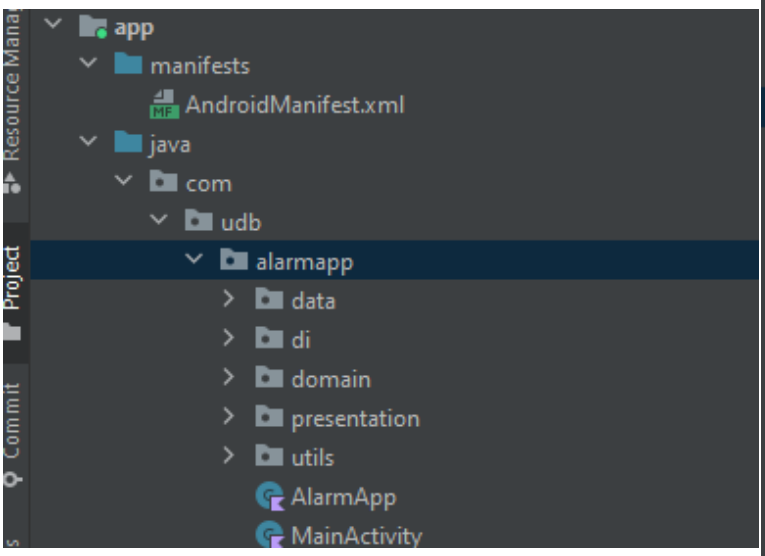
La aplicación trata sobre poder recordar medicamentos a personas, estas pueden elegir crear una alarma y vincularla a un grupo de medicamentos para que se les notifique en la fecha estipulada que deben tomar su medicamento. La aplicación hace uso de room, dagger hilt, mvvm y jetpack compose.

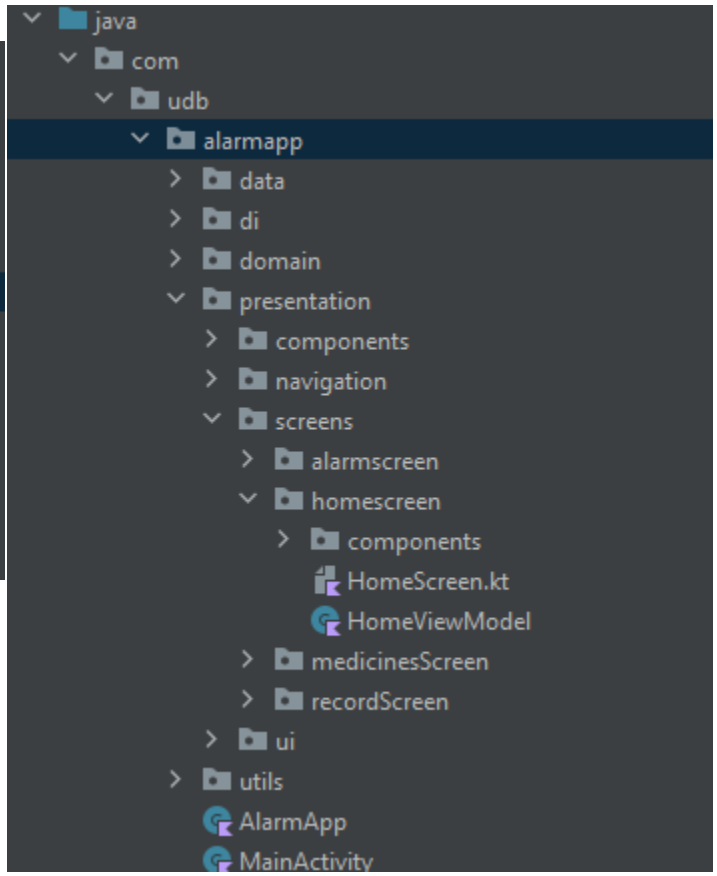
Arquitectura desarrollada

Como arquitectura, se cuenta con un manejo de carpetas que incluye: "data" para representar nuestro modelo, "di" para la configuración de los módulos para inyección de dependencias con Dagger Hilt, "domain" para la lógica de negocio. En nuestro caso, se espera que acá se maneje la lógica del manejo de fechas y horas. Si bien es cierto que en MVVM el dominio no se contempla, se recomienda usarlo cuando la lógica de negocio es grande, ya que así agregamos otra capa de abstracción para mantener un código más estructurado y limpio.

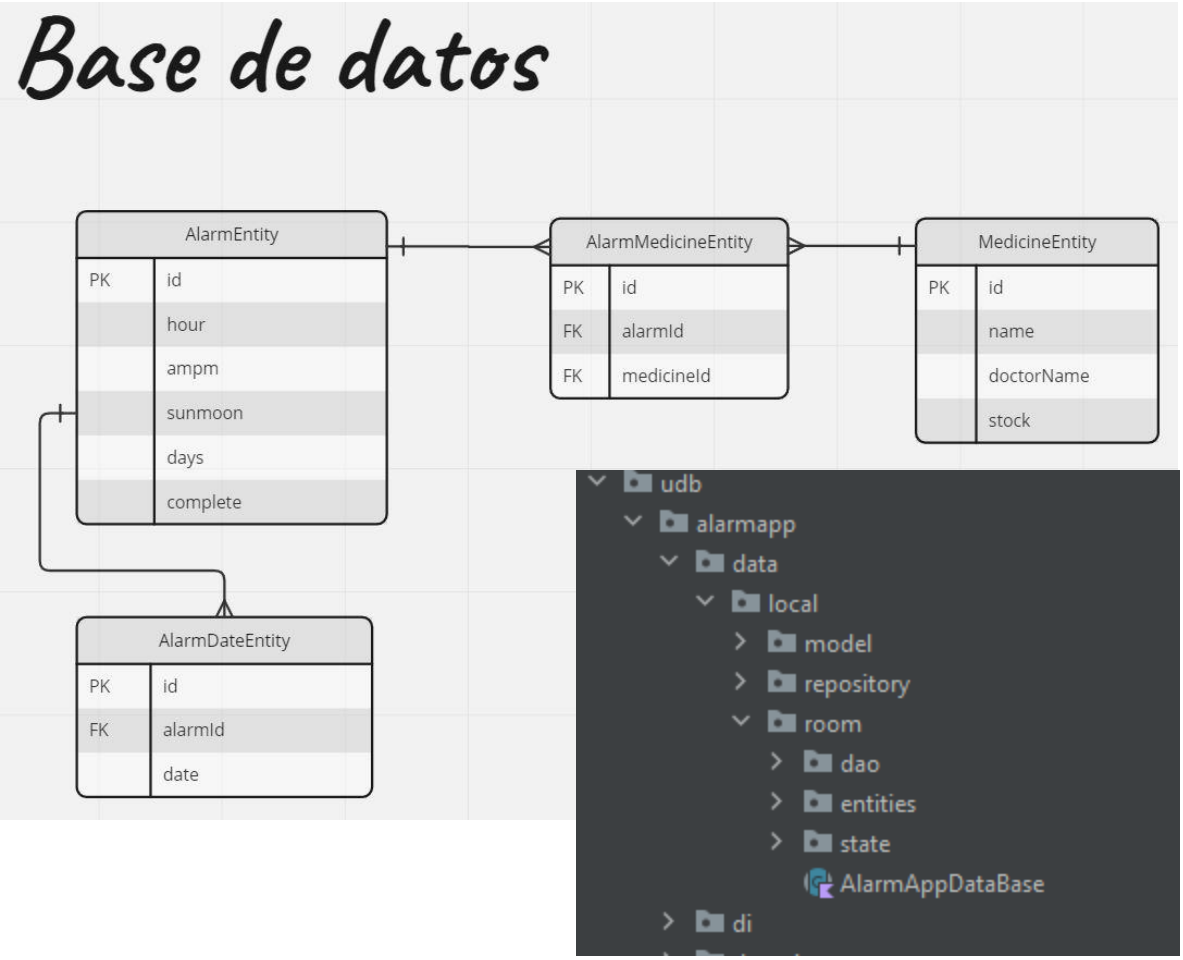
Por último, contamos con la capa de vista, llamada en este caso presentacion, donde manejamos todo lo relacionado con la vista, componentes, navegación y screens. Para esta aplicación, se usa la arquitectura de diseño de una sola actividad como punto de entrada, con múltiples screens, que se asemeja a cómo se trabaja en otras plataformas basadas en componentes como Angular, Flutter o React. Como preferencia personal, se agrega el ViewModel en el directorio donde pertenece su screen, así toda la lógica de una screen está en un solo lugar.

Por recomendación de la arquitectura, se sugiere agregar una carpeta extra de ViewModels para manejar ViewModels que se compartirán con otros ViewModels, esto para el manejo de los datos que se obtienen de la base de datos. Por ejemplo, las medicinas se necesitan en múltiples screens, por lo que no sería recomendable mantener esa lógica en el ViewModel de MedicineScreen. Es mejor mantenerlo en un ViewModel compartido para que los demás ViewModels consulten a este, esta parte espero agregarla en el futuro antes de la entrega final del proyecto de catedra para manejar correctamente los viewmodel compartidos.

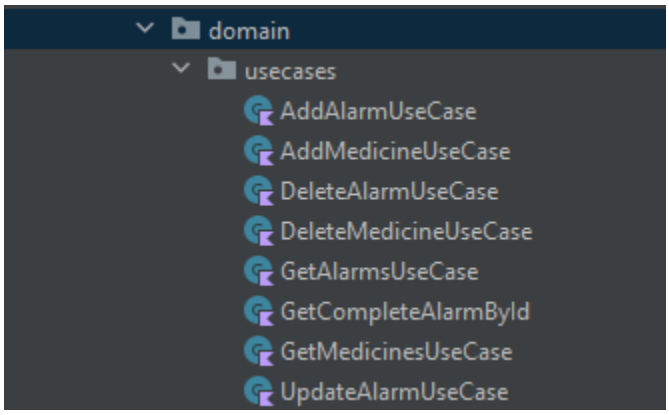




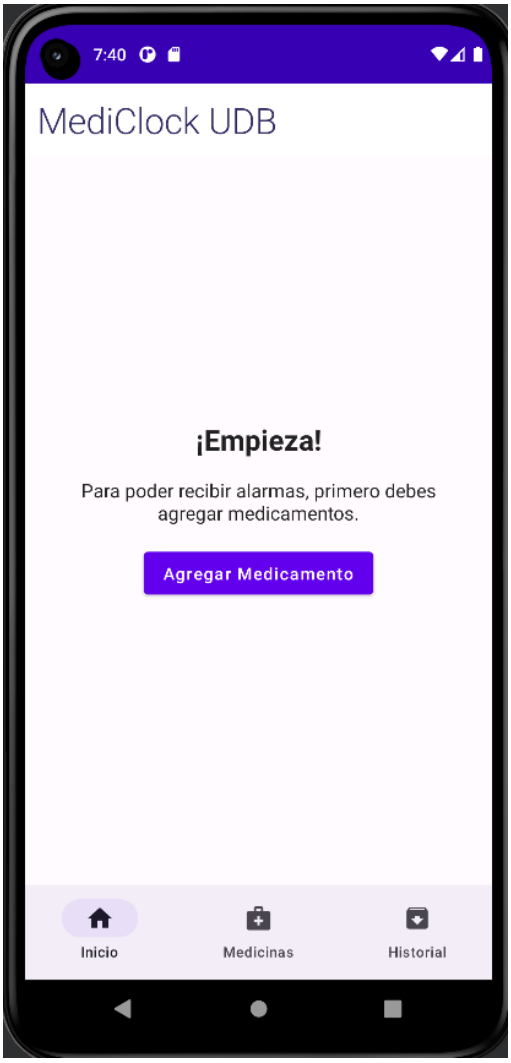
La aplicación cuenta con una base de datos relacional sqlite usando room, toda la configuración de room se encuentra en la capa de modelo, que la hemos llamado data, aquí tenemos modelos para el manejo de los datos, entidades para el manejo de la comunicación de los datos entre la base de datos y nuestra aplicación, y repositorios, estos sirven como una capa de abstracción que comunica la capa de modelo con las demás capas.

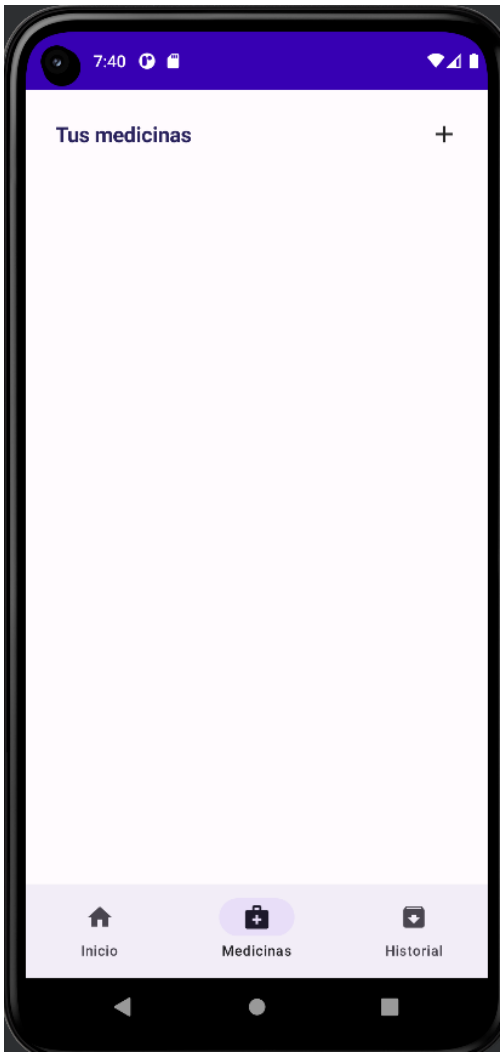


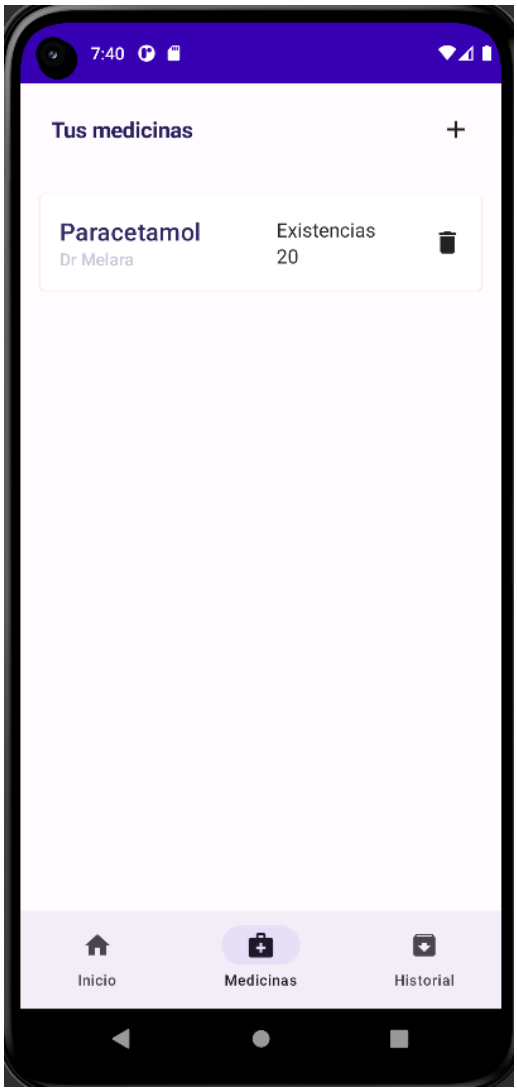
La capa de dominio actualmente solo cuenta con casos de uso con muy poca responsabilidad. Se espera agregar en esta capa toda la lógica de negocio, que en el caso de la aplicación sería el manejo de fechas, cálculo de días, milisegundos, etc. Todo lo relacionado con la manipulación de fechas y horas. Actualmente, esta lógica está en los ViewModels, lo que hace que los ViewModels sean muy grandes y cuenten con más responsabilidad de la que deberían, haciéndolos muy complejos y difíciles de mantener. Además, debido a mi inexperiencia en crear buen código, hay muchas malas prácticas. Sin embargo, con lo nuevo que he aprendido, confío en mejorar el código haciendo uso de esta capa de dominio.

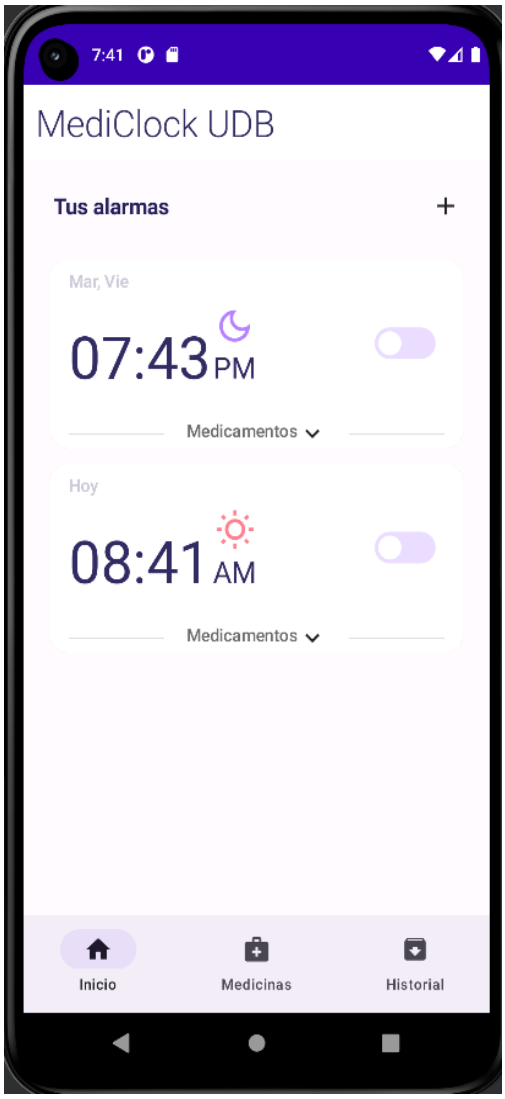


La aplicación actualmente cuenta con los crud de medicinas y alarmas, su funcionamiento es como se ve en las siguientes imágenes:









Aún hay ciertos comportamientos extraños en AlarmScreen, mas específicamente en el manejo del estado en el viewmodel, a veces el estado no se destruye al cambiar de screen, pero en un 80% de los casos muestra correctamente la informacion, ya he investigado sobre el manejo de efectos secundarios y la correcta manipulación del viewmodel así que espero solucionar estos problemas prontos.

Para editar una alarma se debe hacer doble tap en ella y para borrarla se debe mantener un long press, esto ahorita por facilidad en el desarrollo, se espera mejorar esto.

Bibliografía

https://inmediatum.com/blog/ingenieria/mvvm-que-es-y-como-funciona/

https://openwebinars.net/blog/la-arquitectura-mvvm-y-sus-componentes/

https://keepcoding.io/blog/que-es-el-patron-de-arquitectura-mvvm/

https://learn.microsoft.com/es-es/dotnet/architecture/maui/mvvm