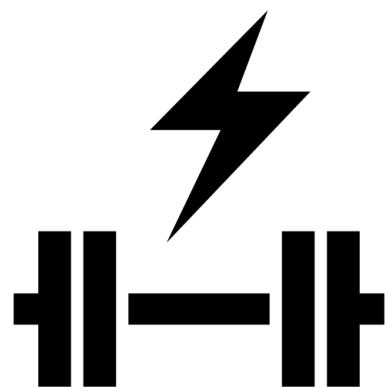


# **UNIVERSIDAD COMPLUTENSE DE MADRID**

FACULTAD DE INFORMÁTICA

GRUPO 9



Programación de aplicaciones para dispositivos móviles

## **Fit Journal**

Realizado por:  
**Jaime Pablo Vázquez Martín**

Profesora:  
**María Cruz Valiente Blázquez**

Curso: 4º  
**2023/2024**

# Índice

<b>1. Descripción general de la aplicación.....</b>	<b>1</b>
<b>2. Requisitos funcionales.....</b>	<b>2</b>
<b>3. Justificación del diseño y consideraciones técnicas fundamentales.....</b>	<b>3</b>
3.1 Arquitectura y pseudo-arquitectura del proyecto.....	3
3.1.1 Principios Clean.....	3
3.1.2 Descripción genérica patrón MVI.....	3
3.1.3 Aplicación práctica de MVI en Fit Journal.....	5
3.2 Tecnologías utilizadas.....	5
3.2.1 Dagger hilt.....	5
3.2.2 Calendar.....	6
3.2.3 Compose Destinations.....	6
3.2.4 Coil.....	6
<b>4. Capturas de pantallas de la aplicación.....</b>	<b>7</b>
4.1 Splash.....	8
4.2 Login.....	8
4.2.1 Error.....	9
4.3 Sign Up.....	9
4.3.1 Error.....	10
4.4 Home.....	10
4.5 Workouts.....	11
4.5.1 Edit workouts / Confirm deletion.....	11
4.6 Add workout.....	12
4.6.1 Add exercise.....	12
4.6.2 Create workout.....	13
4.7 Exercise list.....	14
4.8 Exercise.....	14
4.8.1 Rest.....	15
4.8.2 Exercise (apaisado).....	15
4.9 Workout complete.....	16
4.10 Profile Screen.....	16
4.10.1 Upload photo.....	17
4.10.2 View progress photo.....	17
4.12 App settings.....	18
4.13 Edit profile.....	18
4.13.1 Upload photo.....	19
<b>5. Descripción de almacenamiento persistente utilizado en la app.....</b>	<b>20</b>
5.1 Data store.....	20
5.2 Room.....	20
5.3 Firebase authentication.....	21
5.4 Firebase Firestore.....	21
5.5 Firebase Storage.....	21

6. Conclusiones personales.....	22
7. Bibliografía.....	22

**Link al repositorio del proyecto en github: [repositorio FitJournal](#)**

# 1. Descripción general de la aplicación

Fit Journal es una aplicación que sirve como diario digital de gimnasio, en ella sus usuarios pueden crear sus propias rutinas de entrenamiento, buscando ejercicios a través de un servicio remoto y pudiendo añadirlos a rutinas con nombres personalizados. Los usuarios también podrán seguir su progreso en el gimnasio a través de la app subiendo imágenes que se guardarán en almacenamiento en la nube usando Firebase Storage. En la pestaña Home los usuarios podrán ver un calendario con los días en los que han entrenado. La gestión de usuarios (tanto autenticación como datos de los mismos) se hace a través del servicio google Firebase.

La aplicación se ha construido de principio a fin intentando seguir principios de diseño, buenas prácticas y arquitecturas que permitan un proyecto entendible y escalable a futuro. El potencial de la aplicación es bastante grande pero debido al poco tiempo que se ha tenido para desarrollarlo no ha dado tiempo a optimizar y ampliar funcionalidades, así que se podría considerar como una primera versión que será ampliada en el futuro con posible potencial para ser publicada en la Playstore de Google.

## 2. Requisitos funcionales

- Crear perfil: En la aplicación se podrán crear perfiles que se guardarán en una base de datos en la nube y se podrá iniciar sesión en cualquier dispositivo que tenga la aplicación.
- Modificar perfil: Se podrán añadir y modificar distintos datos del perfil del usuario que ayudarán a tener una experiencia más personalizada en la aplicación.
- Foto de perfil: Permite subir una foto de perfil que se guardará en la nube.
- Crear una rutina: Permite crear una rutina en la que después se añadirán los ejercicios.
- Seleccionar parámetros de cada ejercicio: Permite ajustar cada ejercicio de la rutina para seleccionar el número de series, pesos, etc. Estos datos se podrán actualizar cada vez que se requiera.
- Buscar un ejercicio: Permite buscar un ejercicio determinado escribiendo su palabra clave. La aplicación consumirá datos de una API externa para devolver los resultados de la búsqueda.
- Añadir ejercicio a rutina: Una vez buscado el ejercicio se podrá añadir a una rutina para utilizarlos en próximas sesiones.
- Calendario: Permite ver el calendario, con los días en los que has entrenado, también permite seleccionar un día y ver los detalles del entrenamiento de ese día (en caso de que se haya entrenado).
- Progreso: Permite a los usuarios subir una foto suya para que le sirva de referencia en el futuro.
- Idiomas: La aplicación se podrá usar tanto en inglés como en español, a excepción de el calendario y las llamadas a servicio remoto, que al ser un servicio de tercero solo devuelve los datos en inglés por lo que los detalles de cada ejercicio y los días de la semana y mes del calendario sólo se podrán ver en inglés.

### 3. Justificación del diseño y consideraciones técnicas fundamentales

#### 3.1 Arquitectura y pseudo-arquitectura del proyecto

##### 3.1.1 Principios Clean

Para la realización de un código robusto, escalable y encapsulado se han seguido los principios de diseño de arquitectura [Clean](#), asimismo, el código se ha dividido en cuatro capas:

- core: en esta capa van los recursos comunes que pueden ser accedidos por todas las capas, en este proyecto en particular se ha utilizado para guardar todo el boilerplate necesario para la aplicación del [MVI](#) explicado en detalle más adelante.
- data: en esta capa es donde se gestionan todos los servicios y se preparan para ser enviados a la capa domain (explicada en el siguiente punto). Esta capa tiene toda la lógica de la aplicación y es la encargada de decidir de dónde sacar los datos (por ejemplo si acceder a base de datos local o remota).
- domain: En esta capa se encuentra la lógica de negocio, esta capa sería común a todas las versiones de la aplicación si se hicieran para otros sistemas operativos (IOS, Windows... a excepción de la adecuación con el lenguaje de programación específico de cada sistema operativo). Es la encargada de solicitar información a la capa de data sin 'importarle' de qué servicio venga esta.
- ui: Esta es toda la capa de interfaz de usuario de la aplicación, realizada utilizando Jetpack compose y siguiendo la pseudo-arquitectura [MVI](#) (Model-View-Intent)

Las capas de ui y data se comunican a través de la lógica de negocio (capa domain) que actúa como puente, en esta capa se han añadido también los denominados casos de uso, clases que han servido para encapsular aún más determinadas acciones (ver ejemplos en el código)

##### 3.1.2 Descripción genérica patrón MVI

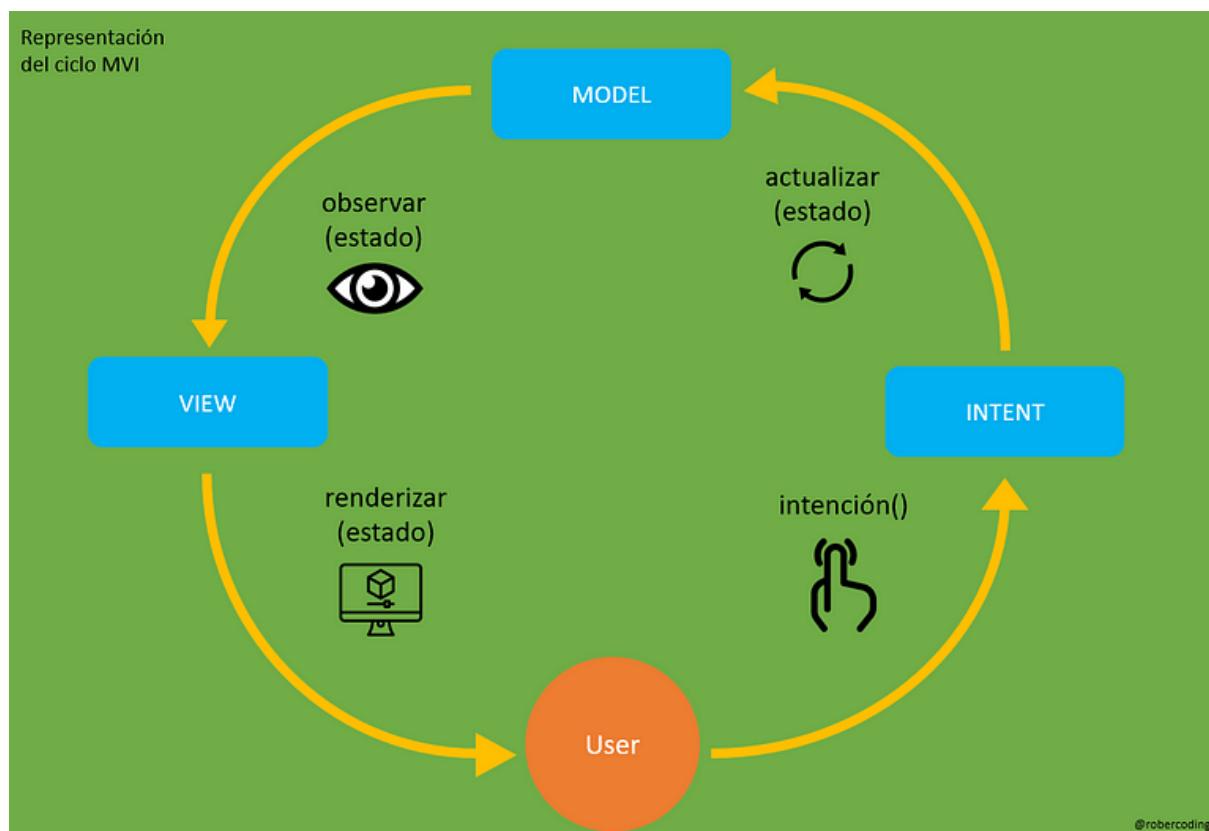
Debido a que en una aplicación móvil la mayor parte del trabajo se realiza en la interfaz de usuario, es habitual seguir patrones de diseño (mejor denominados: pseudo-arquitecturas). En el caso de Fit Journal había dos posibilidades de patrones a seguir: [MVVM](#) (Model-View-Viewmodel) o [MVI](#) (Model-View-Intent). Se decidió seguir el patrón MVI por ser más organizado y viable para aplicaciones de tamaño grande, a su vez es un patrón que se adapta muy bien al paradigma de programación en kotlin (programación funcional), su única desventaja es que

introduce código de relleno para la estructuración (denominado boilerplate), pero esto a su vez ayuda a que sea más entendible por terceros a medida que aumenta el tamaño del proyecto. A continuación se explica en qué consiste esta pseudo-arquitectura y como se ha aplicado en el proyecto. MVI es una de las nuevas pseudo-arquitecturas reactivas que han surgido recientemente para Android. Los roles de cada componente son los siguientes:

**Model:** Model es un término que puede variar en distintas pseudo-arquitecturas, en MVI es el que representa un estado de los datos, que puede ir cambiando. Por ejemplo podemos tener estados para la carga de datos (una variable *isLoading*) y otra para el usuario (empieza siendo nulo y cuando se han cargado los datos se reescribe la variable)

**View:** View se encarga de observar los estados y renderizar en la vista. Por ejemplo, en el caso de observar el estado *isLoading* renderizaríamos un ProgressBar, y cuando observe *Success* renderizaríamos en pantalla los datos.

**Intent:** Intent se define como una acción (No confundir con Intent de Android) que se lanza cuando el usuario realiza una acción (o el sistema solicita un cambio de estado). Estas acciones se procesan en un pipe del viewmodel y van sobreescritiendo el estado.



Representación gráfica del patrón MVI ([fuente](#))

Esta estructura de flujo unidireccional de datos tiene muchas ventajas que ayudan a crear un código entendible, organizado y escalable.

### 3.1.3 Aplicación práctica de MVI en Fit Journal

Dentro de la capa de UI se creará una carpeta por cada pantalla (o feature). Cada feature tendrá tres componentes (para facilitar la descripción y el entendimiento de la estructura de una feature se utilizará como ejemplo la pantalla de login, para otras features se cambiaría login por el nombre de la nueva pantalla), a continuación se hace una breve descripción de cada componente aunque es posible que haya detalles que se pasen por alto:

- *LoginScreen*: este archivo contiene todas la funciones componibles necesarias para representar la vista, se le inyecta el viewmodel que contiene el estado y pinta sus componentes en función de este. Cuando el usuario o sistema realizan una acción (como pulsar un botón o que se acabe un temporizador), se llama al método correspondiente del viewmodel que añade el intent al pipe de esa feature.
- *LoginViewModel*: Cada viewmodel hereda de una clase llamada *BaseViewModel* en la que está declarado el pipe y un método abstracto *reduce*, encargado de procesar cada tipo de intent declarado (sobreescribir el estado)
- *LoginContract*: Este archivo declara el estado (una data class), con todos sus atributos y todos los posibles intents que puedan lanzarse (una clase sellada)

## 3.2 Tecnologías utilizadas

En este apartado se describirán tecnologías utilizadas para mejorar la estructura, rendimiento y utilidad del proyecto.

### 3.2.1 Dagger hilt

Todo el proyecto ha sido implementado utilizando inyección de dependencias mediante Dagger Hilt.

Dagger Hilt es un framework que realiza operaciones en tiempo de compilación (generando ficheros) para Android, el cual se encarga de crear y administrar la creación de objetos en toda la aplicación, esto ayuda facilita la realización de pruebas y la codificación inmensamente ya que evita el tener que saber donde se crea cada objeto.

Se han creado módulos para inyectar todos los servicios remotos y de persistencia de datos: Retrofit, Firebase o Room, creando automáticamente instancias Singleton que serán utilizadas por todas las interfaces que las requieren.

### 3.2.2 Calendar

Para el calendario que se muestra en la pantalla de Home se ha utilizado una biblioteca creada por la comunidad ([repositorio biblioteca](#)) y se ha aprovechado la gran versatilidad que ofrece la biblioteca para ajustarla a la funcionalidad deseada.

### 3.2.3 Compose Destinations

La navegación nativa que ofrece jetpack compose no es demasiado cómoda por el momento ya que se navega con rutas en formato de string y puede llegar a ser confuso y poco seguro. Para solucionar este problema se ha encontrado otra biblioteca creada por la comunidad ([repositorio biblioteca](#)) que parte con la base de la navegación nativa pero añade una capa por encima para hacer una navegación más segura y sencilla. Este ha sido el recurso encontrado más útil para la realización del proyecto con mucha diferencia, convirtiendo la navegación entre pantallas en algo muy sencillo.

### 3.2.4 Coil

Una biblioteca ([Documentación Coil](#)) creada para Jetpack compose (equivalente a [glide](#) en Android views). Sirve para cargar imágenes en las vistas, proporciona el componente *AsyncImage* totalmente personalizable que proporciona máximo rendimiento. Se ha optado por Coil en lugar de bibliotecas como [Picasso](#) ya que ofrece un rendimiento e integración con Jetpack Compose mucho mejor.

## 4. Diagrama de navegación y Capturas de pantallas de la aplicación

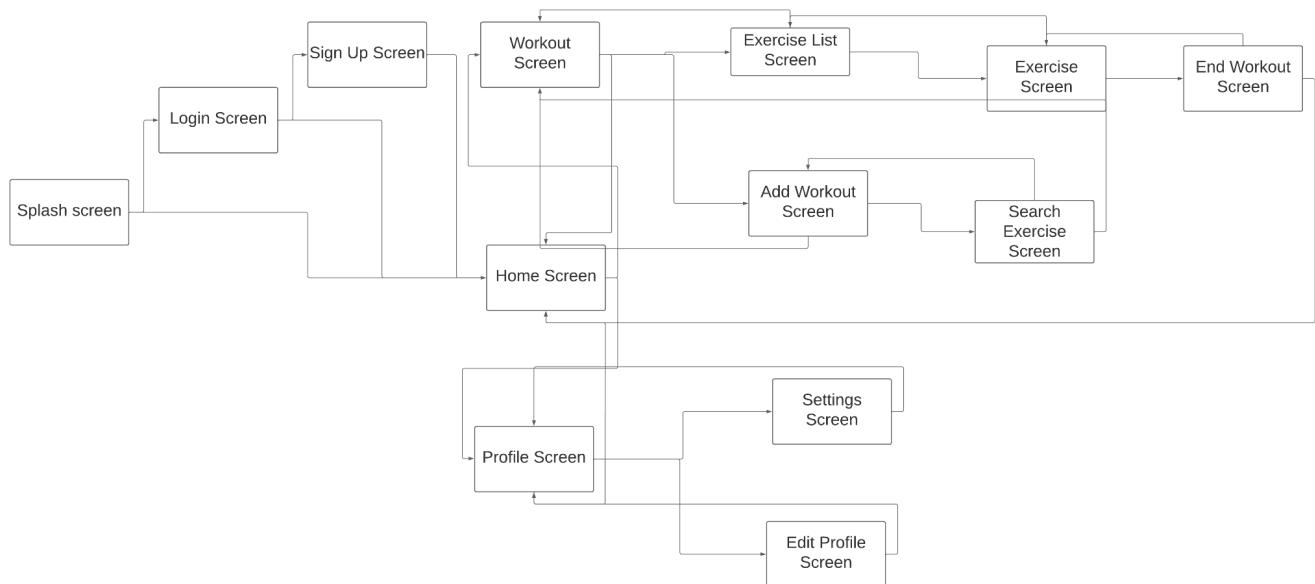
A continuación se ha incluido el diagrama de navegación de la aplicación y capturas de pantalla de todas las pantallas.

En el diagrama de navegación, cada bloque representa una pantalla y cada flecha un posible destino de la misma.

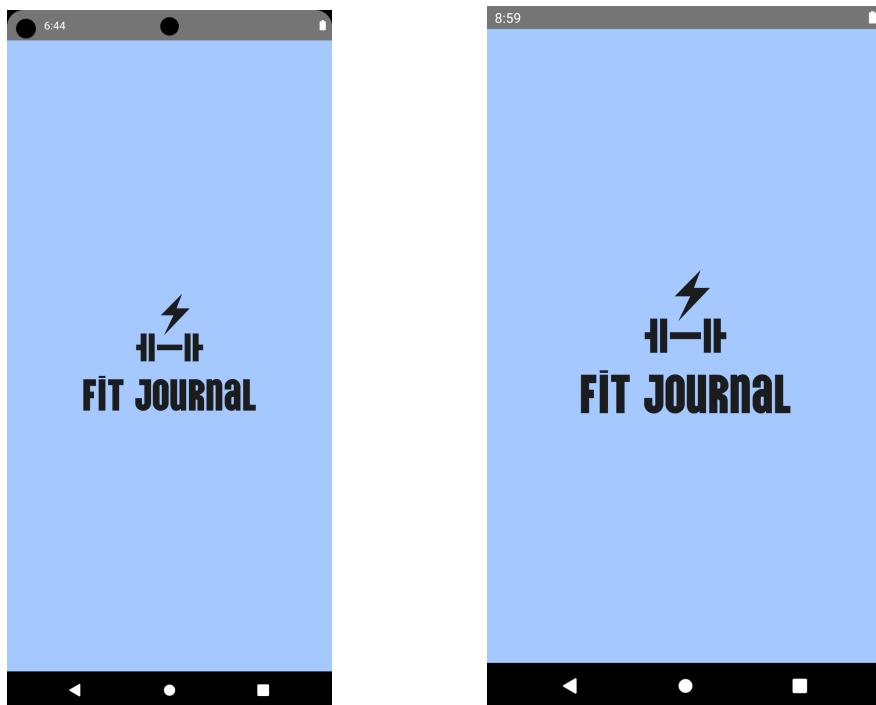
En cuanto a las capturas de pantalla, las de la izquierda han sido realizadas en un emulador del *Google Pixel 7 Pro* con la aplicación en modo oscuro. Las de la derecha han sido realizadas en un emulador del *Nexus 5X* con la aplicación en modo claro. Al ser un teléfono corto en algunas pantallas no cabe todo el contenido, pero en todos los casos se puede hacer scroll para verlo completo.

En el caso de la pantalla *Exercise* (4.8) Se ha desarrollado también una versión de la interfaz en apaisado, se mostrará en el apartado 4.8.2.

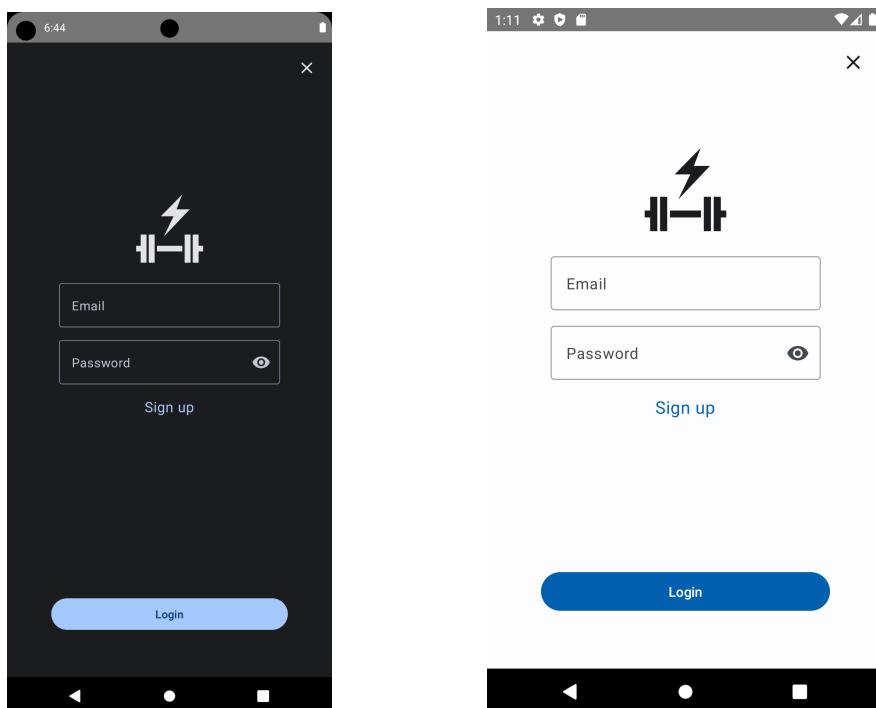
### 4.1 Diagrama de Navegación



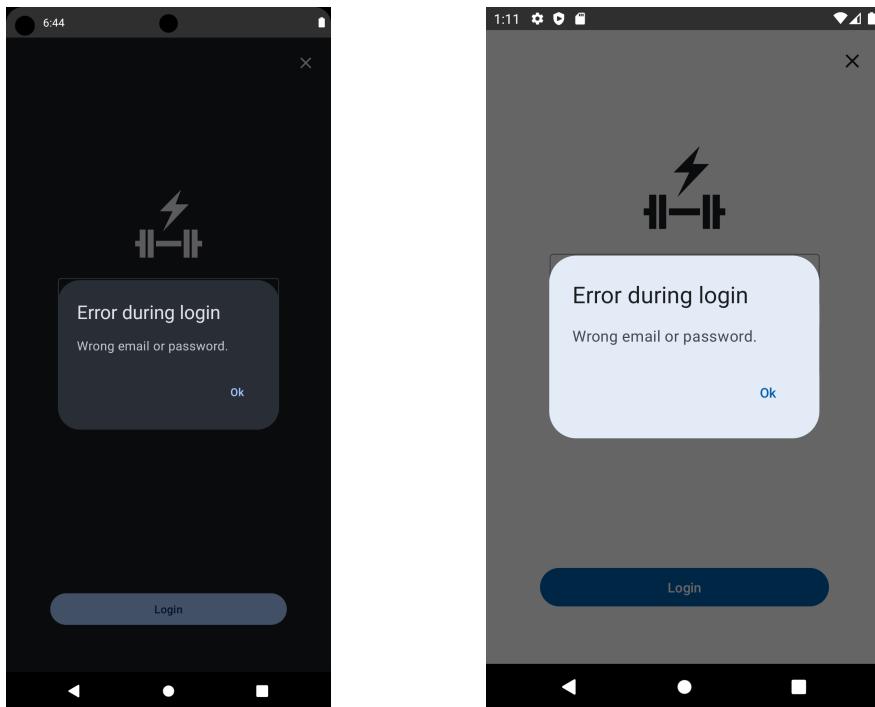
## 4.2 Splash Screen



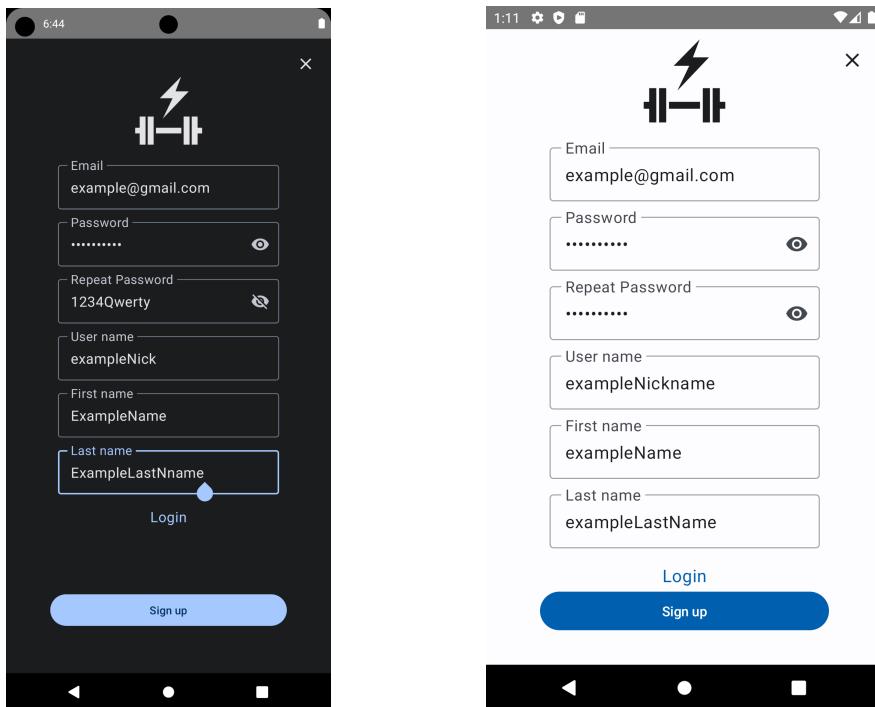
## 4.3 Login Screen



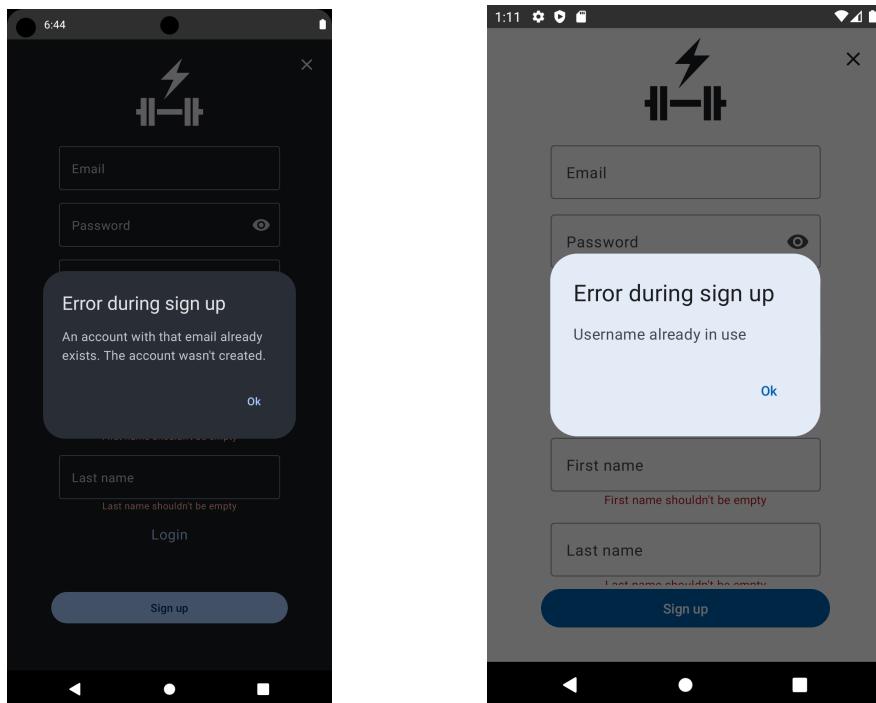
#### 4.3.1 Error



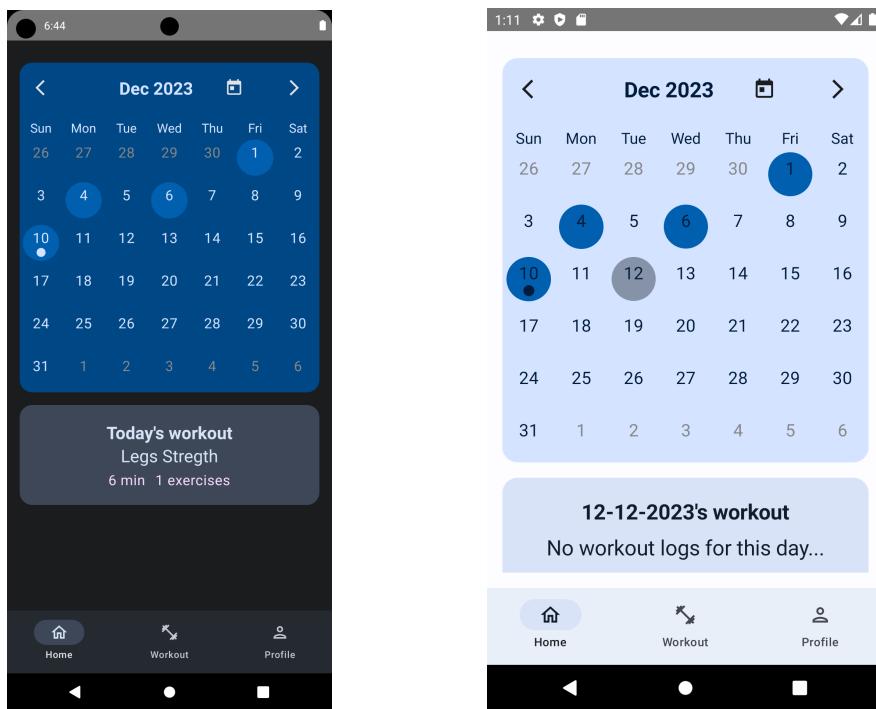
#### 4.4 Sign Up Screen



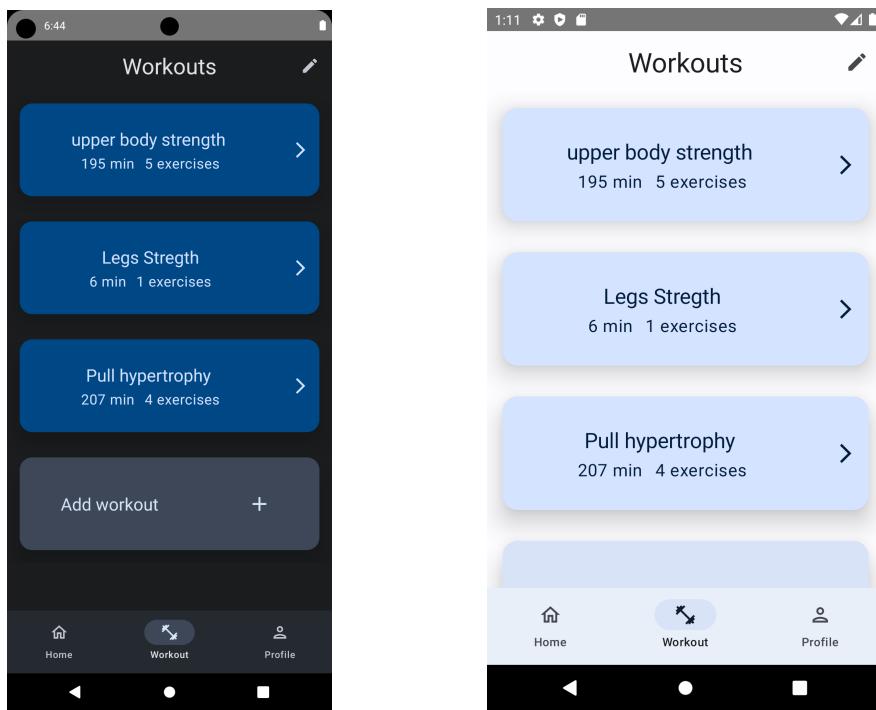
#### 4.4.1 Error



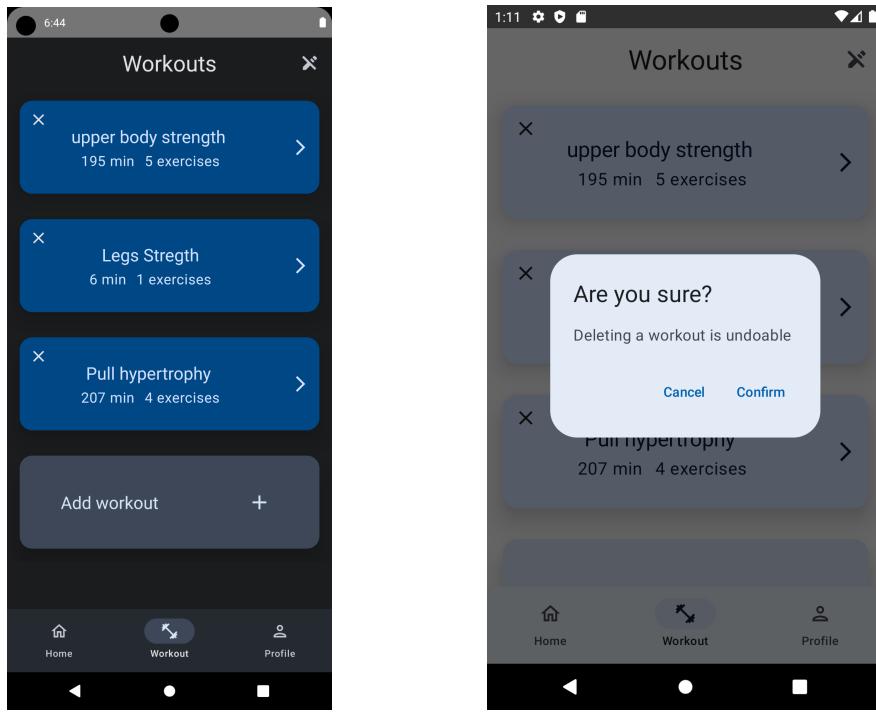
#### 4.5 Home Screen



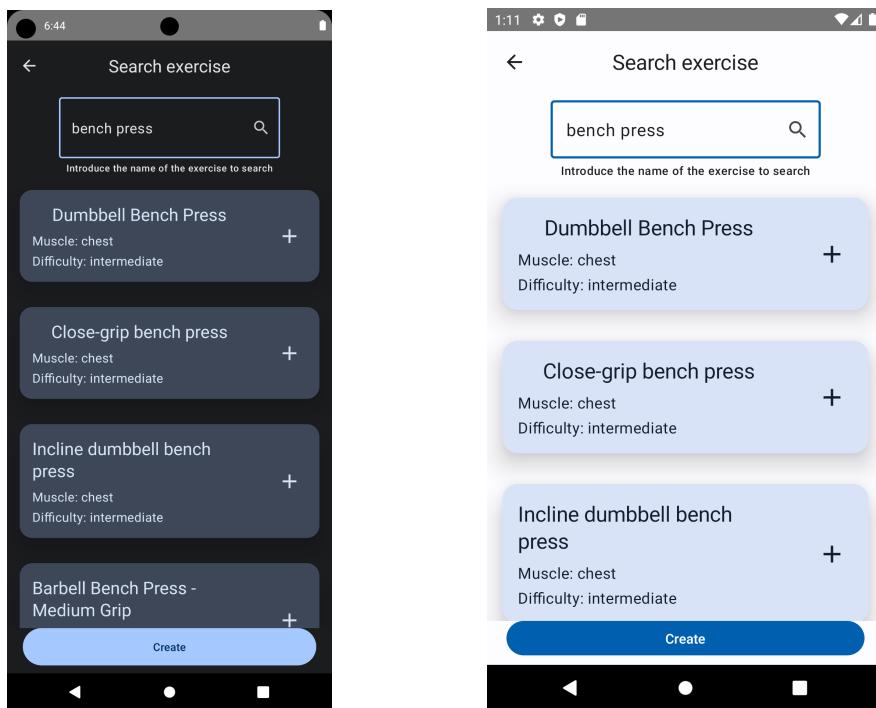
## 4.6 Workouts Screen



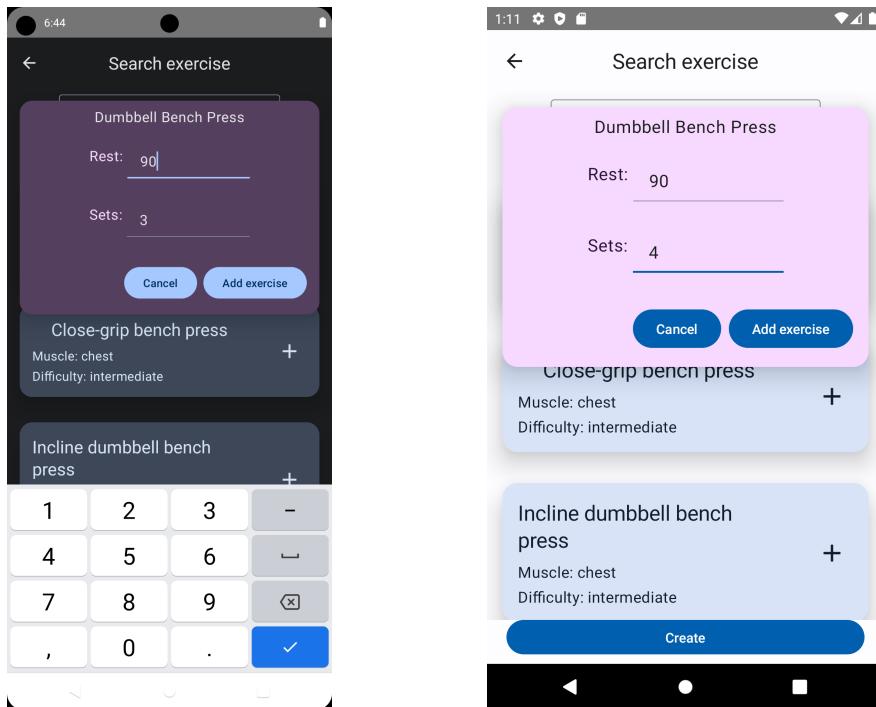
### 4.6.1 Edit workouts / Confirm deletion



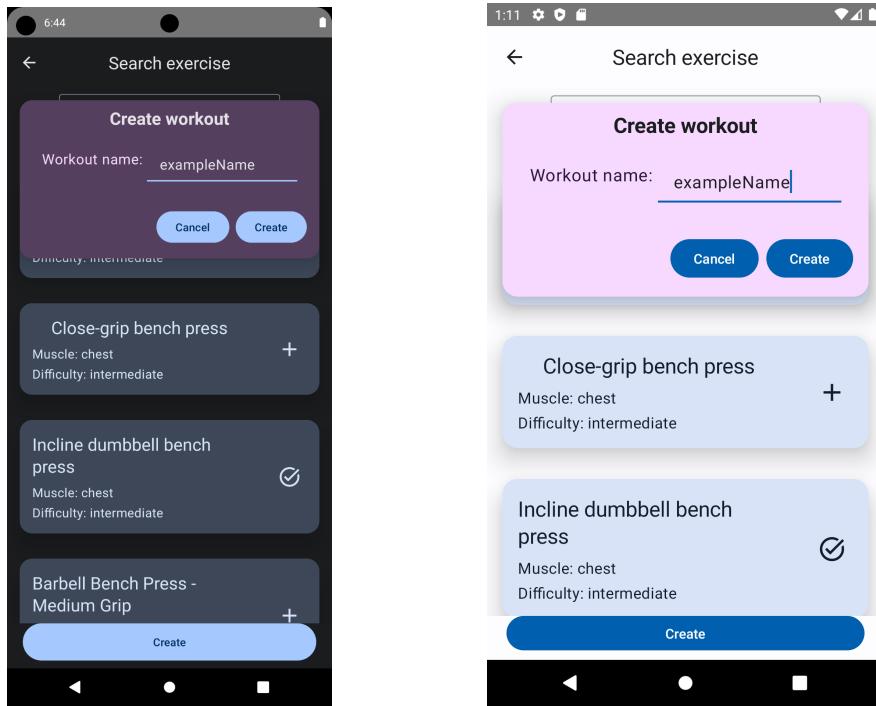
## 4.7 Add workout Screen



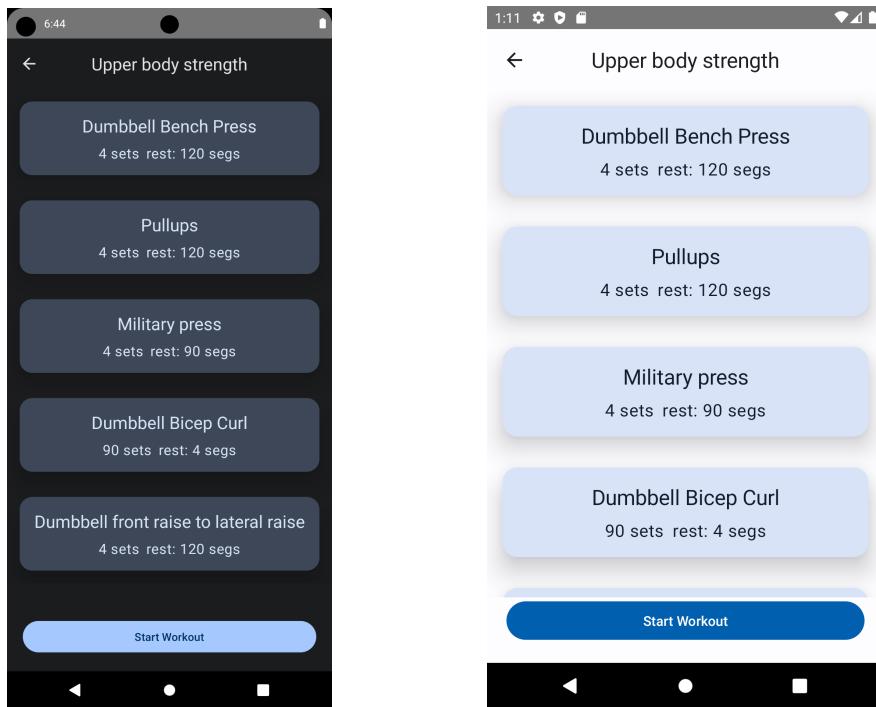
### 4.7.1 Add exercise



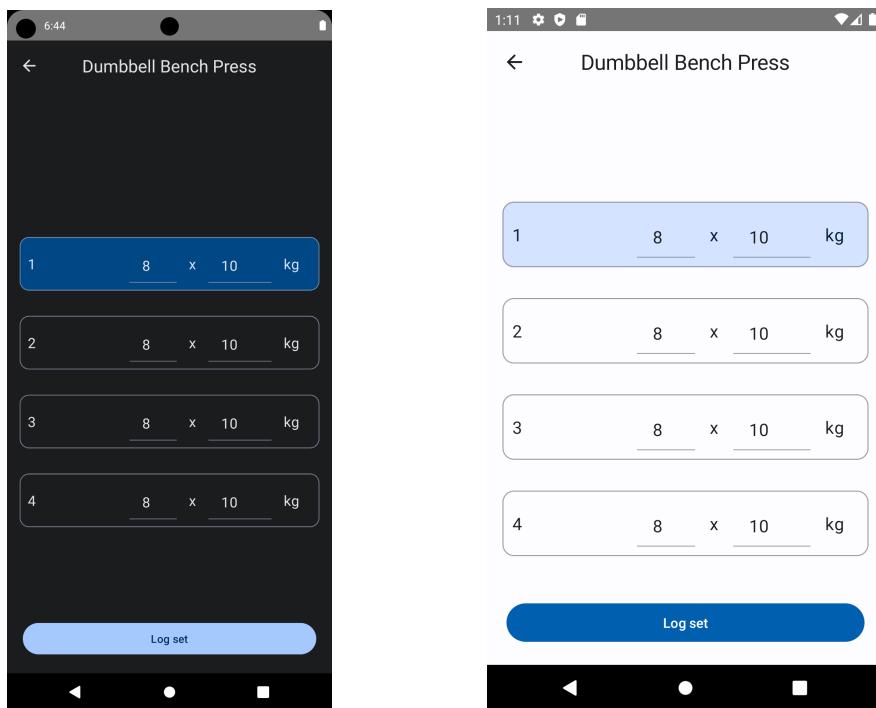
## 4.7.2 Create workout



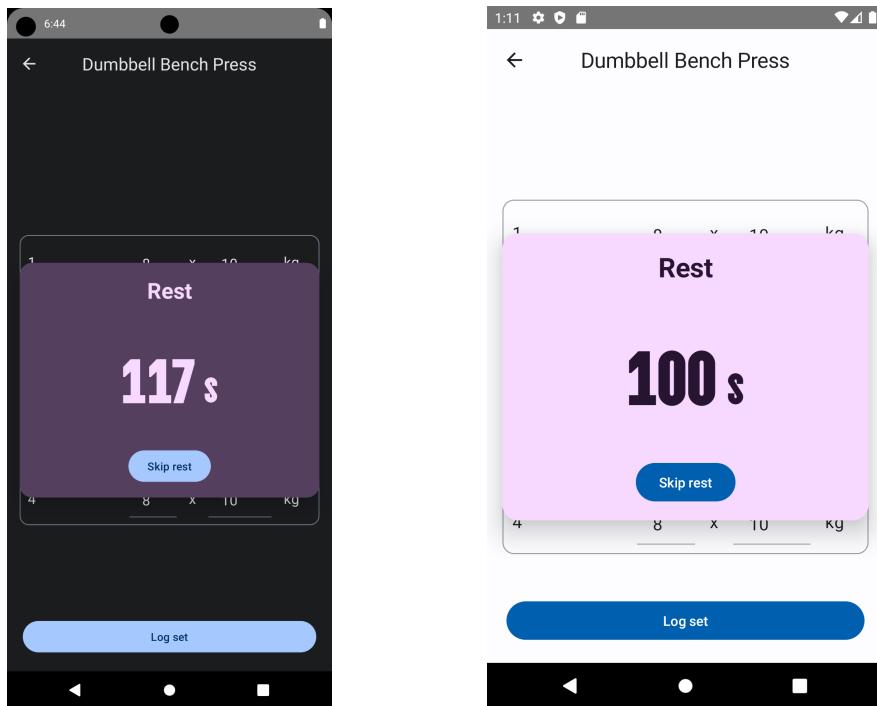
## 4.8 Exercise list Screen



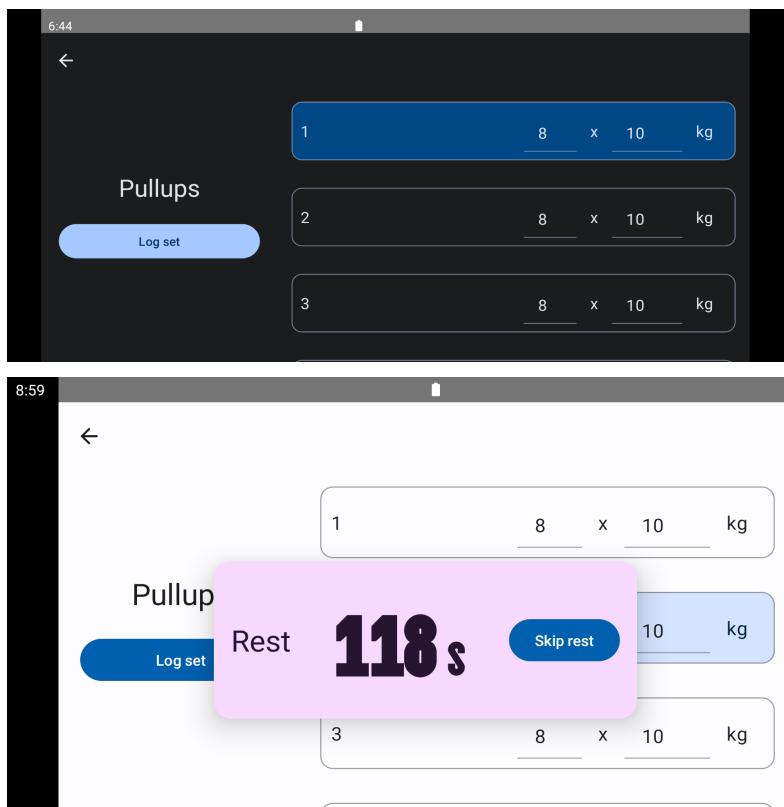
## 4.9 Exercise Screen



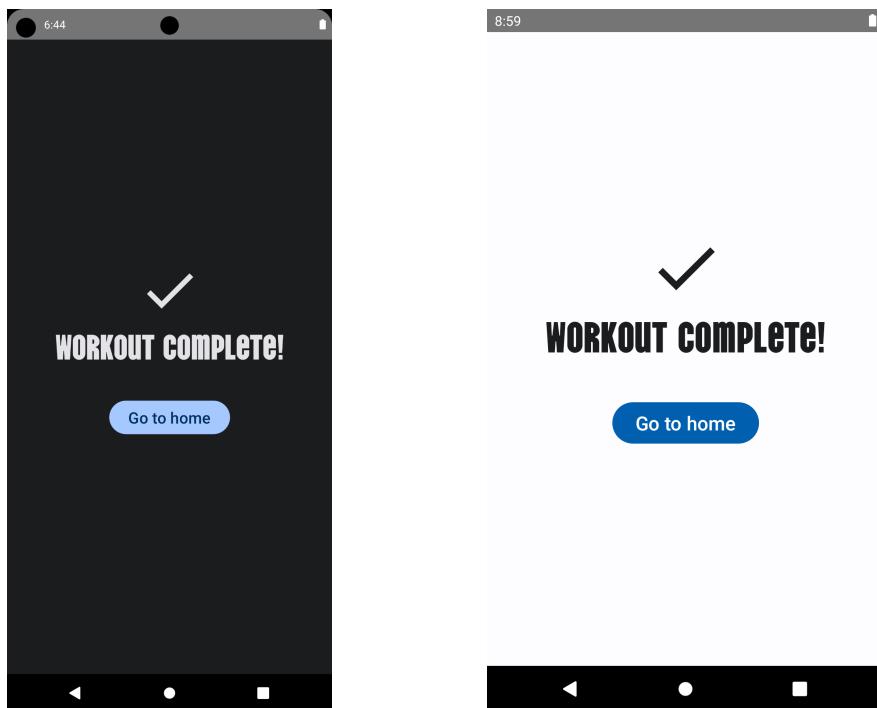
#### 4.9.1 Rest



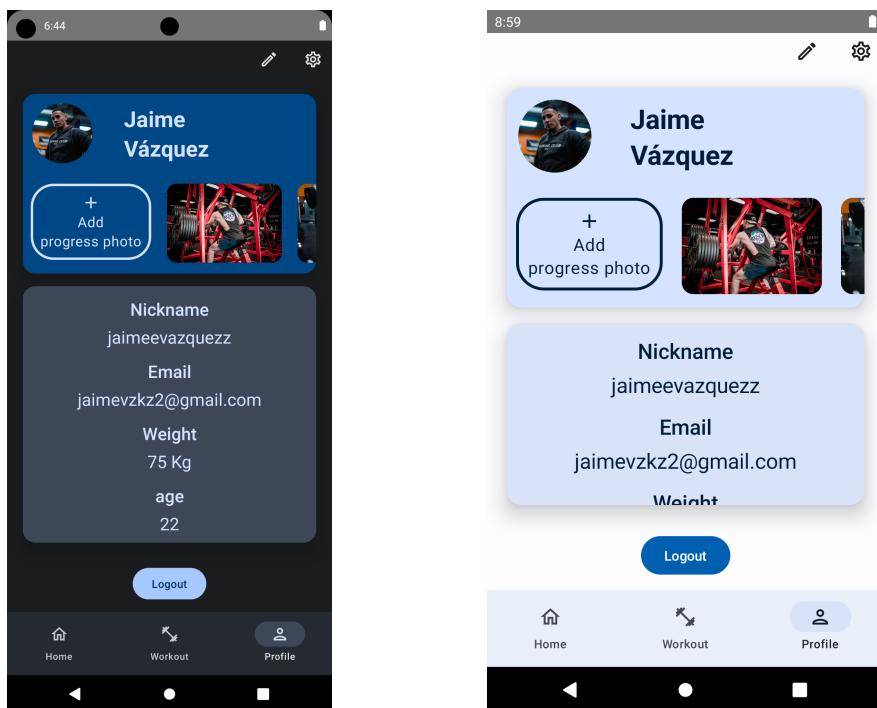
#### 4.9.2 Exercise (apaisado)



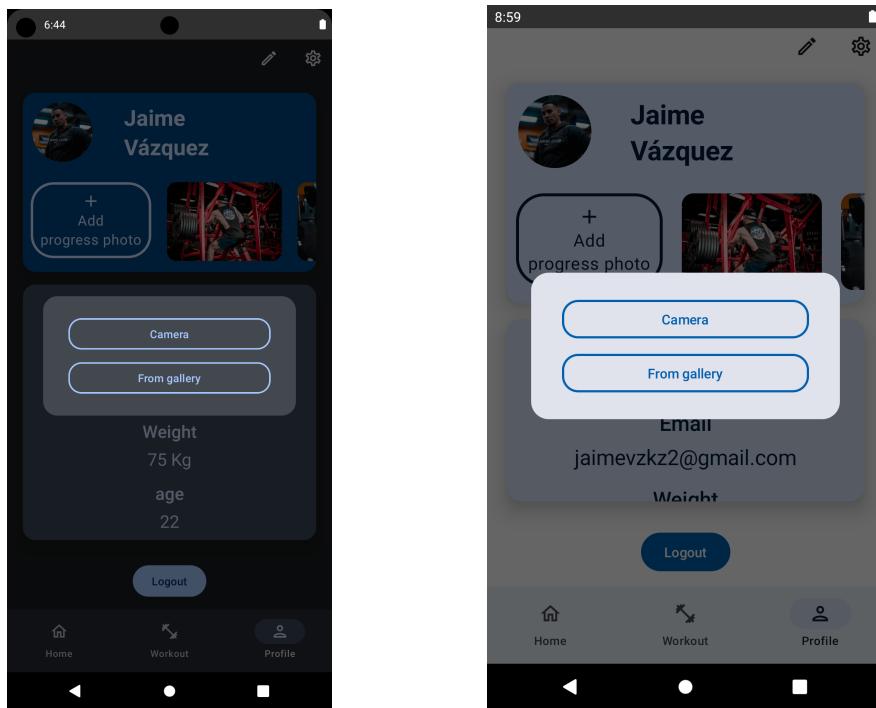
## 4.10 Workout complete Screen



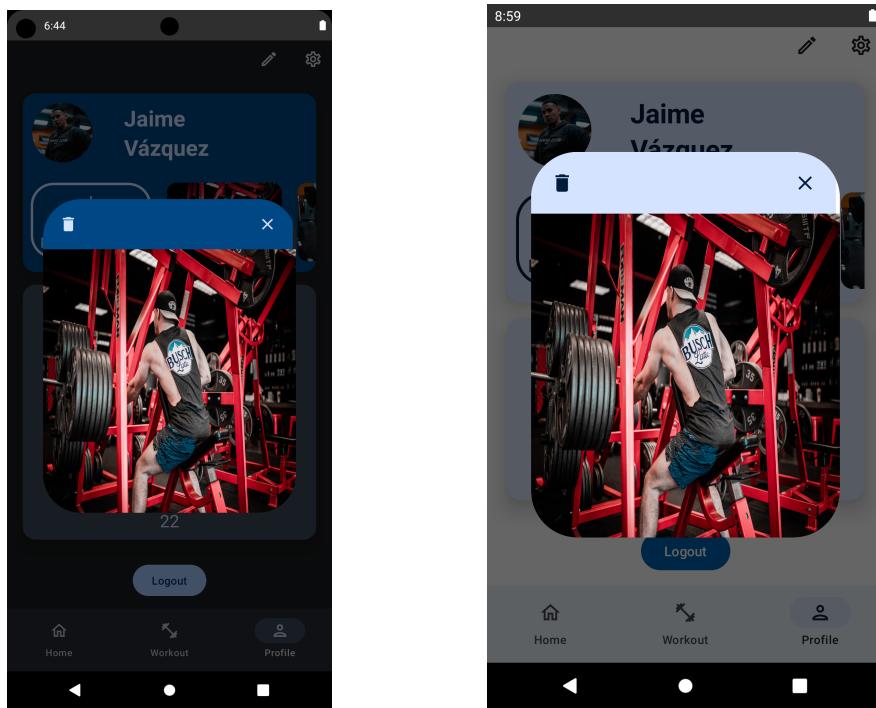
## 4.11 Profile Screen Screen



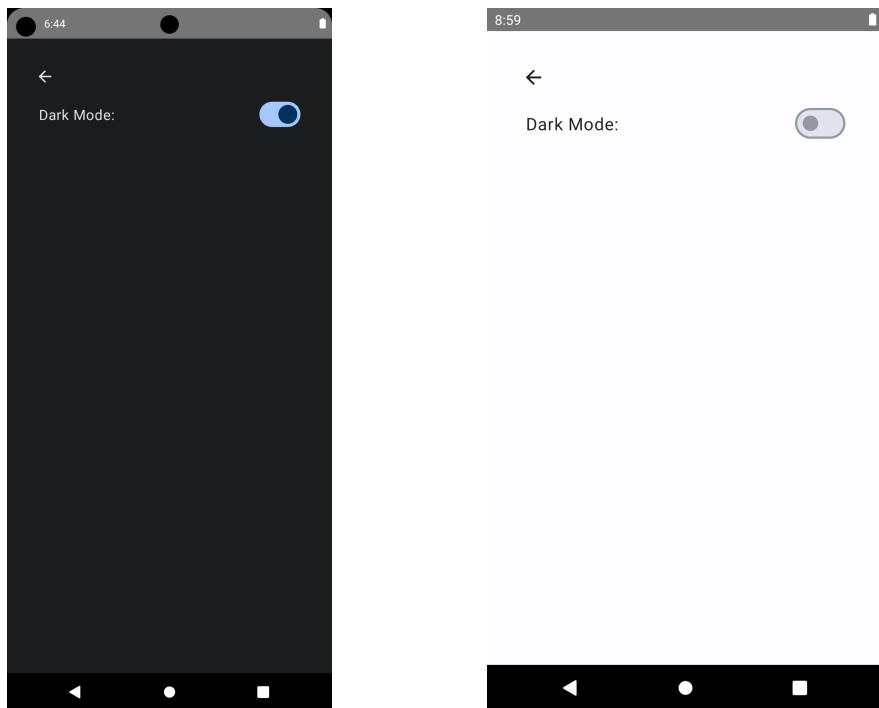
#### 4.11.1 Upload photo



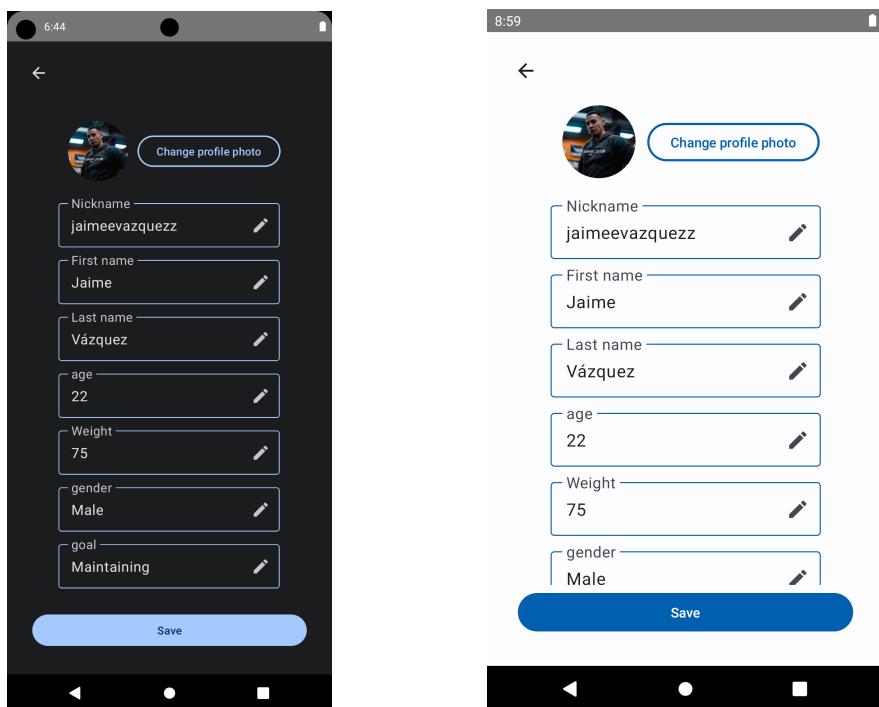
#### 4.11.2 View progress photo



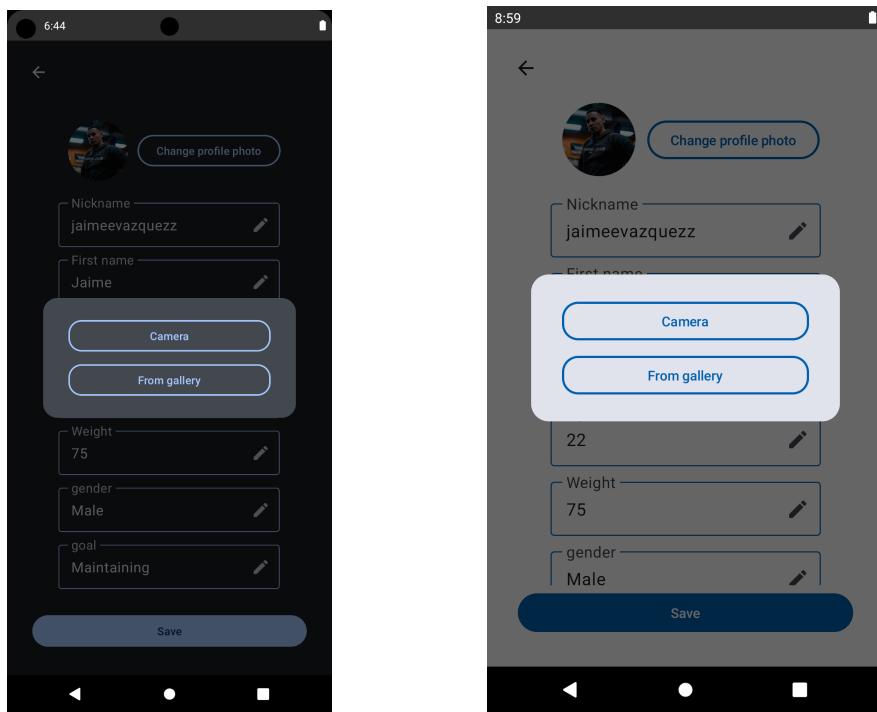
## 4.12 App settings Screen



## 4.13 Edit profile Screen



#### 4.13.1 Upload photo



## 5. Descripción de almacenamiento persistente utilizado en la app

Para el desarrollo de la aplicación se han utilizado dos formas de almacenamiento persistente: [Data store](#) para almacenar datos de acceso más rápido y [Room](#) como base de datos local para guardar los datos de usuario.

### 5.1 Data store

Data store es un sistema de almacenamiento persistente que permite guardar pares clave valor en la aplicación para un acceso rápido a través de corrutinas.

En el caso de Fit Journal se ha utilizado para guardar 2 cosas:

- Tema de la aplicación: de tal forma que cuando se cambia su valor, devuelve un flow que indica a la Main Activity el tema que debe utilizar. Este tema se mantiene en cache por lo que se podría cerrar la aplicación y al abrirla se recordaría el tema establecido.
- Nickname y uid de usuario iniciado: De tal forma que cuando se necesiten datos del usuario desde cualquier parte de la app se puede conseguir el nickname o el uid (que son únicos en base de datos) según necesidad, con el nickname se podrá acceder a room para conseguir todos los datos del usuario. El uid sirve para un acceso más rápido en Firestore debido a que es una base de datos que se indexa con este campo.

### 5.2 Room

Room es un ORM (Object-Relational mapping) que permite trabajar de una forma más sencilla con bases de datos SQL. Para interactuar con Room se ha creado una base datos devolverá a la app los Data Access Objects (DAO) encargados de persistir la información en la base de datos y de devolver las entities.

Room se ha utilizado para guardar el UserModel (modelo de datos de un usuario) indexado por el campo nickname, debido a que el modelo de datos contenía objetos de tipos sin conversión automática a Room, se han guardado los datos en formato Json, y para ello se ha utilizado un *Typeconverter* personalizado que se encarga de pasar el modelo a una cadena Json. A su vez, ha sido necesario crear un *Gson Converter* personalizado para poder convertir en json algunos datos (las fechas de tipo *LocalDate* y las *Uri*). Todos estos converters personalizados se han guardado en el fichero 'Converters.kt'.

# 6. Descripción de servicios remotos utilizados en la app

En este apartado se explicarán los servicios remotos utilizados en la aplicación. Para la búsqueda de ejercicios se ha utilizado Retrofit. Por otro lado se han utilizado múltiples servicios proporcionados por [Firebase](#) para almacenamiento de datos en la nube, permitiendo así la autenticación de usuarios ([Firebase Authentication](#)), el uso de bases de datos ([Firebase Firestore](#)) y el almacenamiento de imágenes ([Firebase Storage](#)). En los siguientes subapartados se explica cómo se ha utilizado cada servicio.

## 6.1 Retrofit

Retrofit es una biblioteca de Android desarrollada por [Square](#) que facilita la implementación de peticiones HTTP desde cualquier aplicación móvil. Para conseguir los datos se ha utilizado una API gratuita encontrada en la página web [API ninjas](#) que ofrece gran variedad de APIs para la creación de aplicaciones de todo tipo. La API utilizada ha sido [Exercises Api](#), que dado el nombre de un ejercicio devuelve una lista de hasta diez ejercicios con distintos atributos. La API permite también realizar búsquedas filtrando por músculo, tipo o dificultad, pero debido al tiempo del proyecto solo se ha implementado la búsqueda por nombre de ejercicio.

Para poder realizar las llamadas al servicio la web requería una cabecera HTTP con una clave de API asignada a cada usuario, para ello se ha añadido en el proyecto al objeto retrofit un *Interceptor* que ha servido para añadir las cabeceras necesarias (véase el archivo 'AuthInterceptor.kt' en el código para más detalles).

## 6.2 Firebase authentication

Firebase Authentication proporciona servicios de backend, SDK fáciles de usar y bibliotecas de IU ya elaboradas para autenticar a los usuarios en la aplicación. Se ha configurado la autenticación mediante email y contraseña.

## 6.3 Firebase Firestore

Firestore es una base de datos NoSQL flexible, escalable y en la nube, creada en la infraestructura de Google Cloud, a fin de almacenar y sincronizar datos para el desarrollo tanto del lado del cliente como del servidor.

El diseño de la base de datos externa ha sido uno de los mayores desafíos en el desarrollo del proyecto ya que se ha buscado la eficiencia en las consultas con el objetivo de consumir la menor cantidad de consultas posibles.

## 6.4 Firebase Storage

Cloud Storage para Firebase es un servicio de almacenamiento de objetos potente, simple y rentable construido para el escalamiento de Google. Los SDK de Firebase para Cloud Storage agregan la seguridad de Google a las operaciones de carga y descarga de archivos.

Se ha utilizado para la carga y descarga de imágenes, tanto de las fotos de progreso como la foto de perfil de los usuarios.

## 7. Conclusiones personales

En este apartado se explicará la experiencia personal del desarrollo de Fit Journal, se hará en primera persona ya que el proyecto ha sido desarrollado de principio a fin por Jaime Vázquez, alumno de cuarto en el grado de ingeniería informática de la Universidad Complutense de Madrid.

El desarrollo de Fit Journal ha supuesto un gran desafío para el desarrollo de mis aptitudes académicas, ya que el reto de construir una aplicación nativa entera desde cero, partiendo de conocimientos básicos de Android views y kotlin (han servido de base para aprender a programar usando Jetpack Compose) y solo parecía en un principio un reto demasiado grande para compaginar con el resto de asignaturas de la carrera. Sin embargo, el hecho de ir descubriendo y aprendiendo nuevas tecnologías y trucos cada semana, junto con el hecho de poder comprobar que eran cosas que se aplicaban en muchas de las aplicaciones que uso en mi día a día me ha motivado para dedicarle todas las horas que han sido necesarias.

Cómo puntos negativos en lo respectivo al desarrollo del proyecto, me gustaría recalcar algunas cosas:

- **Android studio:** Pese a que el IDE de desarrollo nativo de android se actualiza constantemente y cada vez va un poco mejor; y que gracias al uso de kotlin se facilitan muchos procedimientos con respecto a java, los tiempos de compilación a veces llegan a ser muy lento (he llegado a estar esperando 20 minutos a que compilara el proyecto, aunque he de reconocer que en contadas ocasiones) y el IDE da errores internos que la mayoría de veces se solucionan haciendo un ‘Clean and rebuild’ del proyecto. Pese a todo lo anterior y aunque sin duda la experiencia es mejorable también tiene cosas muy buenas como la claridad de la documentación que ayuda a solucionar casi cualquier error.
- **Trabajo en solitario:** Pese a que trabajar en solitario tiene algunas cosas buenas en lo respectivo a la autoorganización, la carga de trabajo en este proyecto ha sido bastante elevada y sin duda me habría ayudado mucho contar con un equipo de compañeros que me ayudaran tanto en el desarrollo, perfeccionamiento de detalles y documentación. A pesar de lo anterior, el hecho de haber tenido que realizar yo todo el trabajo me ha ayudado a maximizar el aprendizaje ya que he tenido que entender cada cosa que he hecho y conozco cada detalle en la implementación del proyecto.

En conclusión, el desarrollo de esta aplicación me ha ayudado a introducirme en el mundo del desarrollo de aplicaciones para móviles y sin duda seguiré indagando en este mundo para, alomejor, dedicarme a ello algún día.

## 8. Bibliografía

- [Explicación pseudo-arquitectura MVI](#)
- [Biblioteca Calendar creada por la comunidad](#)
- [Biblioteca para compose destinations creada por la comunidad](#)
- [Documentación Coil](#)
- [Documentación glide](#)
- [Documentación Data store](#)
- [Documentación Room](#)
- [Documentación de retrofit](#)
- [Web de API ninjas](#)
- [Api utilizada en el proyecto](#)
- [Documentación Firebase](#)
  - [Documentación Firebase Authentication](#)
  - [Documentación Firebase Firestore](#)
  - [Documentación Firebase Storage](#)