Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Memoria de seguimiento

Creación de una Aplicación móvil y los Servicios REST Asociados para la Recogida de Datos Nutricionales

Autor: Andrés Yanguas Cariñena
Tutor(a): Raúl Alonso Calvo

Madrid, Junio 2024

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

Trabajo Fin de Grado
Grado en Ingeniería Informática

Título: Creación de una Aplicación móvil y los Servicios REST Asociados para la Recogida de Datos Nutricionales

Junio 2024

Autor: Andrés Yanguas Cariñena
Tutor: Raúl Alonso Calvo
Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Resumen

El proyecto “Creación de una Aplicación móvil y los Servicios REST Asociados para la Recogida de Datos Nutricionales“ tiene como objetivo proporcionar a los usuarios una herramienta completa para el seguimiento y gestión de su ingesta nutricional diaria. A través de la implementación de tecnologías punteras y efectivas, se ha desarrollado una aplicación móvil Android y un backend eficiente que trabajan en conjunto para ofrecer una experiencia de usuario óptima y personalizada.

Se han utilizado diversas tecnologías para permitir que los usuarios mantengan un registro detallado de sus ingestas nutricionales diarias. La aplicación móvil desarrollada en Kotlin para Android y el backend implementado con FastAPI garantizan una integración y un rendimiento óptimo. El backend, alojado en Render y utilizando MongoDB Atlas como base de datos, proporciona una infraestructura rápida y escalable para la gestión de datos. Los usuarios pueden registrar sus comidas y alimentos consumidos, gestionar estos datos y sincronizarlos con la base de datos en la nube.

Una de las características clave de la aplicación es la capacidad de mostrar estadísticas detalladas sobre las ingestas nutricionales del usuario. Utilizando la fórmula de Mifflin-St Jeor, la aplicación calcula las recomendaciones diarias de calorías y macronutrientes (carbohidratos, proteínas y grasas). Los usuarios pueden comparar sus ingestas reales con estas recomendaciones a través de gráficos interactivos y visualizaciones claras.

Finalmente, para asegurar que las recomendaciones basadas en la fórmula de Mifflin-St Jeor se mantengan actualizadas, los usuarios pueden adaptar sus datos personales dentro de la aplicación. Esto incluye la posibilidad de actualizar su género, nivel de actividad física, peso, altura y edad. La pantalla de perfil permite la edición de estos datos personales, asegurando que las recomendaciones nutricionales reflejen siempre la situación actual del usuario.

Abstract

The project “Creation of a Mobile Application and Associated REST Services for the Collection of Nutritional Data“ aims to provide users with a complete tool for the monitoring and management of their daily nutritional intake. Through the implementation of cutting-edge and effective technologies, an efficient Android mobile application and backend have been developed that work together to offer an optimal and personalized user experience.

Various technologies have been used to allow users to keep a detailed record of their daily nutritional intakes. The mobile app developed in Kotlin for Android and the backend implemented with FastAPI ensure optimal integration and performance. The backend, hosted on Render and using MongoDB Atlas as a database, provides a fast and scalable infrastructure for data management. Users can record their meals and food consumed, manage this data, and synchronize it with the cloud database.

One of the key features of the app is the ability to display detailed statistics on the user’s nutritional intakes. Using the Mifflin-St Jeor formula, the app calculates daily calorie and macronutrient recommendations (carbohydrates, protein, and fat). Users can compare their actual intakes to these recommendations through interactive charts and clear visualizations.

Finally, to ensure that recommendations based on the Mifflin-St Jeor formula are kept up to date, users can tailor their personal data within the app. This includes the ability to update your gender, physical activity level, weight, height, and age. The profile screen allows the editing of this personal data, ensuring that nutritional recommendations always reflect the user’s current situation.

Tabla de contenidos

Capítulo 1

Introducción

A menudo, la información nutricional es inaccesible o confusa y las elecciones alimenticias se hacen cada vez más complejas debido a la abundancia de opciones, ésto emerge en la necesidad de herramientas tecnológicas que faciliten decisiones alimentarias conscientes y equilibradas. El proyecto que abordamos en este Trabajo de Fin de Grado (TFG) surge precisamente para responder a esta necesidad. Con el objetivo de promover estilos de vida saludables, nos proponemos diseñar e implementar una aplicación móvil que no solo sirva como una herramienta de apoyo en la optimización de hábitos nutricionales, sino que también promueva una comprensión más profunda y accesible de la nutrición personal. Según el informe de junio de 2020 del Comité Permanente de Nutrición del Sistema de las Naciones Unidas [1], "las tecnologías digitales son herramientas importantes que pueden ayudar a transformar los sistemas alimentarios y asistir en el diseño y la entrega de medidas de alimentación y nutrición".

Este proyecto se centra en la creación de una aplicación Android intuitiva y fácil de usar, que permita a los usuarios registrar y analizar su ingesta diaria de alimentos y micronutrientes. A través de esta aplicación, buscamos proporcionar una plataforma que facilite el seguimiento de los hábitos alimentarios, ofreciendo a los usuarios una forma clara y sencilla de comprender su nutrición y hacer ajustes donde sea necesario. La aplicación contará con un sistema de recordatorios y notificaciones, motivando a los usuarios a mantener un equilibrio nutricional y apoyándolos en el seguimiento constante de sus hábitos alimentarios.

Además, el proyecto incluye el desarrollo de un servicio web REST, que jugará un papel crucial en el procesamientode los datos nutricionales recogidos. Este servicio permitirá codificar la información nutricional utilizando nomenclaturas estándares y tablas de alimentos, asegurando así una interpretación precisa de los datos.

Los objetivos específicos del proyecto son:

1. Implementar una aplicación Android para la recogida de hábitos alimenticios, incluyendo la creación de la interfaz de usuario y la lógica de negocio.

2. Interconectar la aplicación con un servicio web de micronutrientes a través de un API REST.
3. Desarrollar métodos y servicios para almacenar en el servidor los datos dietéticos de los usuarios.
4. Realizar el despliegue y las pruebas del sistema para garantizar su funcionalidad y eficacia.

Capítulo 2

Estado del arte

El campo del desarrollo de aplicaciones móviles ha experimentado un crecimiento y evolución notables. La emergencia de nuevas tecnologías, frameworks y metodologías ha transformado la manera en que se crean y se despliegan estas aplicaciones. Este capítulo se centra en explorar el estado actual del arte en el desarrollo de aplicaciones móviles principalmente, haciendo un énfasis en tecnologías punteras de backend y de bases de datos.

La relevancia de este análisis radica proporcionar una comprensión profunda de las tendencias actuales, las mejores prácticas y los desafíos que enfrentan los desarrolladores en el campo del desarrollo móvil. Al explorar estas áreas, no solo se busca entender las herramientas y técnicas disponibles, sino también anticipar cómo podrían evolucionar en el futuro.

El capítulo aborda los siguientes apartados:

- Tendencias en el desarrollo de aplicaciones móviles
- Análisis de las tecnologías de backend y bases de datos
- Interacciones entre el frontend y el backend
- Consideraciones de UI/UX

2.1. Tendencias en el desarrollo de aplicaciones móviles

2.1.1. Kotlin-First

En 2019, desde Google, se anunció que la programación enfocada a desarrollo móvil estaría cada vez más orientado al lenguaje Kotlin [2]. Kotlin es un lenguaje de programación estático que permite programación funcional y orientada a objetos, estáticamente tipado, de alto nivel y propósito general con inferencia de tipos.

Ventajas de Kotlin

Principalmente, la ventaja más significativa es la prioridad de Kotlin en el desarrollo Android, es decir, Google crea nuevas herramientas y contenido de desarrollo de Android teniendo en cuenta a los usarios de Kotlin. Además, otras ventajas [3] que ofrece este lenguaje son:

- Menos código combinado con mayor legibilidad
- Menos errores comunes
- Soporte de Kotlin en bibliotecas de Jetpack
- Soporte para desarrollo multiplataforma
- Lenguaje y entorno maduros
- Interoperabilidad con JavaHaz
- Aprendizaje fácil
- Gran comunidad

Gradle Kotlin DSL

Además de la programación lógica de la aplicación, Google está impulsando la configuarción de compilación de Gradle con el mismo lenguaje Kotlin como reemplazo de Groovy. Google argumenta que, Kotlin es más legible y ofrece una mejor verificación de tiempo de compilación y compatibilidad con IDE ante Groovy [4].

2.1.2. Patrón de arquitectura MVVM y Clean Architecture

El patrón Model-View-ViewModel (MVVM) y la Clean Architecture se han establecido como enfoques estándar en el diseño de aplicaciones móviles.

Model-View-ViewModel

MVVM es un patrón arquitectónico que separa la lógica de la interfaz de usuario (UI) de la lógica de negocios y los datos de la aplicación. Esta separación facilita la modularidad, la escalabilidad y la mantenibilidad del código, al mismo tiempo que mejora la capacidad de realizar pruebas unitarias y automatizadas [5].

Las principales características de MVVM incluyen:

- **Modelo**: Representa los datos y la lógica de negocio.
- **Vista**: Encargada de la presentación y de reaccionar a las interacciones del usuario.
- **ViewModel**: Actúa como intermediario entre la Vista y el Modelo, manejando la lógica de presentación.

Clean Architecture

Clean Architecture, propuesta por Robert C. Martin, no es una arquitectura, sino una guía direccionada a seguir y aplicar los principios SOLID. Se centra en la separación de preocupaciones, donde cada capa de la arquitectura tiene responsabilidades claras y definidas. Esto no solo facilita la prueba y el mantenimiento del código, sino que también promueve un diseño más robusto y menos propenso a errores [6].

Las capas típicas de una Clean Architecture, desde la más interior a la más exterior, incluyen:

- **Entidades:** Representan los objetos de dominio del negocio.
- **Casos de uso:** Contienen la lógica de negocio específica.
- **Controladores y Presentadores:** Encargados de la lógica de presentación.
- **Frameworks y Drivers:** Incluyen herramientas como bases de datos, UI, y dispositivos externos.

Estas capas no están ordenadas aleatoriamente, hay dos consideraciones a la hora de trabajar con esta guía.

Regla de Dependencia Esta regla denota que toda capa exterior dependerá de una capa interior, siendo la capa de la lógica de dominio la última sin dependencia alguna.

Flujo de datos Existe un flujo de datos, éste hace un recorrido por todas las capas. Lo hace en ambos sentidos, comenzando con la entrada de datos por parte del usuario, atraviesa la capa de la lógica de negocio, por último se almacena o se envían a un servicio remoto. En este último caso, el flujo continúa con el retorno del resultado de la operación y finalmente se presenta este resultado al usuario.

Por lo tanto, podemos concluir los tres módulos priincipales para un correcto desarrollo siguiendo la guía de Robert C. Martin:

- **Data:** Contiene los recursos necesarios para acceder a los datos necesarios. Tanto los remotos como las instancias locales.
- **Domain:** Incluye los elementos descriptivos de la lógica de negocio de la aplicación. Este módulo es el único que no depende del Android framework ni de dependencias de terceros, es decir, si la aplicación va a abarcar más de un sistema operativo, la lógica de negocio se mantendrá siempre equivalente.
- **Presentation:** Módulo encargado de mostrar los elementos visuales al usuario, alberga todos los componentes y sistema de UI. Si se emplea MVVM, se deberán incluir los ViewModels en esta sección.

La combinación de MVVM con Clean Architecture en el desarrollo de aplicaciones móviles no solo mejora la calidad y la escalabilidad del código, sino que

también facilita el desarrollo en equipos grandes y la adaptación a cambios en los requerimientos o tecnologías.

2.1.3. Herramientas y tecnologías punteras

En el ámbito del desarrollo de aplicaciones Android, varias herramientas y tecnologías se han establecido como punteras por su eficiencia, facilidad de uso y capacidad para mejorar la calidad del código y la experiencia del usuario.

Retrofit [7]

Retrofit es una biblioteca tipo cliente HTTP, que facilita la comunicación con APIs web mediante anotaciones en las declaraciones de métodos. Su uso simplifica el proceso de interacción con servicios web, permitiendo una integración y manejo de datos de red más eficiente.

Navigation Component [8]

El Navigation Component ofrece una forma simplificada de implementar navegación en una aplicación Android. Proporciona una manera coherente y predecible de gestionar la navegación entre los distintos fragmentos y actividades en la aplicación. Además gestiona la pila que almacena el historial de navegación (back stack [9]), esto significa que en aplicaciones complejas con múltiples niveles de navegación Navigation Component proporciona una forma más estructurada y fácil de manejar el back stack, permite a los usuarios navegar hacia atrás a través de la aplicación de una manera intuitiva y gestionada.

Dagger-Hilt [10]

Dagger-Hilt es una biblioteca de inyección de dependencias que reduce la complejidad de configurar Dagger en aplicaciones Android. Hace que la gestión de dependencias sea más sencilla y menos propensa a errores, permitiendo un código más limpio y testeable.

Room [11]

Room es una capa de abstracción sobre SQLite que ofrece una forma más fluida de acceso a la base de datos en aplicaciones Android. Proporciona comprobación en tiempo de compilación de consultas SQL y una integración más natural con el código de la aplicación.

Coroutines [12]

Las Coroutines en Kotlin son una forma de simplificar el trabajo asíncrono en aplicaciones Android. Permiten escribir código asincrónico de una manera más secuencial y legible, ayudando a evitar los problemas comunes relacionados con las operaciones asíncronas. Estas corrutinas se deben ejecutar en tres subprocesos de Android, para ello Kotlin proporciona tres despachadores (Dispatchers [13]):

- **Dispatchers.Main:** Para interactuar con la IU y realizar trabajos rápidos.
- **Dispatchers.IO:** Para Entrada/Salida de disco o de red.
- **Dispatchers.Default:** Para a realizar trabajo que usa la CPU de manera intensiva.

Flows [14]

Flows en Kotlin son una herramienta para trabajar con secuencias de datos que se calculan de manera asíncrona. Son especialmente útiles para manejar flujos de datos que cambian con el tiempo.

2.2. Análisis de las tecnologías de backend y bases de datos

2.2.1. FastAPI [15]

FastAPI es un moderno framework de backend para Python que ha ganado popularidad por su rapidez, facilidad de uso y escalabilidad. Ofrece características notables como:

- **Rendimiento Alto**: Gracias a su uso de Starlette para las operaciones asincrónicas, FastAPI ofrece un rendimiento comparable a NodeJS y Go.
- **Facilidad de Uso**: Con una sintaxis sencilla y clara, facilita la implementación de APIs robustas y eficientes.
- **Validación de Datos Automática**: Utiliza Pydantic para la validación de datos, lo que minimiza los errores comunes en la entrada de datos.
- **Generación Automática de Documentación**: Proporciona documentación interactiva generada automáticamente.

2.2.2. MongoDB [16]

MongoDB es una base de datos NoSQL que se destaca por su flexibilidad y rendimiento en aplicaciones que requieren un manejo de grandes volúmenes de datos y una estructura de datos no relacional. Sus características incluyen:

- **Esquemas Dinámicos**: Permite cambiar la estructura de los datos sobre la marcha, lo que es ideal para aplicaciones que evolucionan rápidamente.
- **Escalabilidad**: Soporta una escalabilidad horizontal, lo que permite manejar eficientemente grandes volúmenes de datos.
- **Consultas Potentes**: Ofrece un sistema de consulta rico y flexible.
- **Alto Rendimiento**: Optimizado para operaciones de lectura y escritura rápidas.

2.3. Interacciones entre el frontend y el backend

Las interacciones entre el frontend y el backend son fundamentales en el desarrollo de aplicaciones móviles, ya que definen cómo las diferentes partes de la aplicación se comunican y trabajan juntas para ofrecer una experiencia de usuario fluida y funcional.

2.3.1. Comunicación API

Una de las interacciones más críticas entre el frontend y el backend es la comunicación a través de APIs (Interfaces de Programación de Aplicaciones). Las APIs proporcionan una manera estandarizada para que el frontend (la aplicación móvil) solicite y reciba datos del backend (el servidor), y viceversa.

2.3.2. REST: Principios y Prácticas

Arquitectura REST

REST es un estilo arquitectónico para sistemas distribuidos y es ampliamente utilizado para crear APIs que permiten la comunicación entre el frontend y el backend. Se basa en principios de arquitectura web estandarizada y utiliza métodos HTTP para realizar operaciones.

Principios de REST

- **Comunicación sin Estado**: Cada solicitud de cliente a servidor debe contener toda la información necesaria para entender y procesar la solicitud. El servidor no almacena el estado del cliente entre solicitudes.
- **Interfaz Uniforme**: Una interfaz uniforme entre los componentes permite que el sistema sea modular y escalable.
- **Sistema de Capas**: La arquitectura puede estar compuesta por varias capas, cada una con su funcionalidad, mejorando la escalabilidad y seguridad.
- **Recursos Identificables**: En REST, los recursos (datos o servicios) están identificados de manera única a través de URIs.
- **Representaciones**: Los recursos se manipulan a través de sus representaciones (como JSON o XML) enviadas al cliente.

2.3.3. Autenticación y Seguridad

La autenticación y la seguridad son vitales en las interacciones entre el frontend y el backend. Esto incluye asegurar la comunicación API, autenticar usuarios y proteger los datos sensibles.

OAuth [17]

OAuth es un estándar de autorización que permite a las aplicaciones obtener acceso limitado a cuentas de usuario en un servicio HTTP. Se utiliza ampliamente para permitir que los usuarios compartan su información de forma segura entre sitios web sin exponer sus credenciales.

El flujo de OAuth se ordena:

- El usuario solicita acceso a recursos protegidos.
- El proveedor de OAuth presenta un formulario de autenticación al usuario.
- El usuario autoriza la solicitud y recibe un token de autorización.
- La aplicación intercambia el token de autorización por un token de acceso.
- El token de acceso se utiliza para acceder a los recursos protegidos en nombre del usuario.

Bearer Token [18]

Bearer Token es un esquema de autenticación HTTP que implica la seguridad del proceso de acceso. En este esquema, el portador del token de acceso tiene el derecho de acceso a un recurso determinado. El token es una cadena opaca que el cliente debe enviar en el encabezado HTTP de Autorización cuando realiza solicitudes a recursos protegidos.

El uso de Bearer Token sigue la siguiente forma:

- Se obtiene un token de acceso a través de un proceso de autenticación, como OAuth o una solicitud de inicio de sesión.
- El cliente incluye este token en el encabezado de Autorización en sus solicitudes HTTP al backend.
- El backend verifica la validez del token antes de permitir el acceso a los recursos protegidos.

JSON Web Tokens (JWT) [19]

JWT es un método compacto y seguro de transmitir información entre dos partes en forma de objeto JSON. Esta información puede ser verificada y confiable porque está firmada digitalmente.

Características de JWT:

- **Compacto**: Puede ser enviado a través de una URL, parámetro POST, o en un encabezado HTTP.
- **Autocontenido**: La carga útil contiene toda la información necesaria sobre el usuario, evitando la necesidad de consultar la base de datos más de una vez.

- **Seguridad**: La firma del token asegura que solo el emisor del token puede alterar los datos.

2.4. Consideraciones de UI/UX

La interfaz de usuario (UI) y la experiencia del usuario (UX) son aspectos cruciales en el desarrollo de aplicaciones móviles. La elección de las herramientas y técnicas adecuadas para diseñar la UI influye significativamente en la experiencia general del usuario.

2.4.1. XML o Jetpack Compose para el diseño de UI

El lenguaje de marcado XML ha sido durante mucho tiempo el estándar en el desarrollo de interfaces de usuario en Android. Ofrece varias ventajas. Jetpack Compose, en cambio, es un moderno toolkit de UI para Android orientado a programación declarativa que permite una construcción más concisa de interfaces de usuario.

Ventajas de XML frente a Jetpack Compose

A pesar del revolucionario impacto y la intención de Google de avanzar la programación Android utilizando Jetpack Compose, la elección de XML puede ser preferible por varias razones:

- **Familiaridad y Recursos Existentes:** Para los desarrolladores con experiencia previa en Android, XML es una tecnología familiar con una amplia gama de recursos y documentación disponibles.
- **Control Detallado de Diseños:** XML proporciona un control más fino sobre los diseños, especialmente en casos de UI complejas o personalizadas.
- **Integración con Herramientas Existentes:** XML presenta más herramientas existentes en comparación al reciente Jetpack Compose.

2.4.2. Consideraciones de UX

Independientemente de la tecnología de UI elegida, aspectos de UX como la facilidad de uso, la accesibilidad, el rendimiento y la coherencia visual son fundamentales. Es esencial que la aplicación sea intuitiva, reactiva y agradable desde el punto de vista del usuario.

Diseño Centrado en el Usuario

Un enfoque en las necesidades y expectativas del usuario es clave para el diseño de una experiencia de usuario efectiva.

Testing [20]

Las pruebas regulares de código son importantes para garantizar que la lógica y la interfaz de usuario cumpla las necesidades del usuario. Para ello, Android ofrece una guía de cómo la aplicación debe incluir las siguientes tres categorias de pruebas:

- **Tests unitarios:** Pruebas que validan una única unidad de código.
- **Tests de integración:** Pruebas que validan las interacciones entre módulos relacionados.
- **Tests de UI:** Pruebas que validan la trayectoria de los usuarios en varios módulos de la app.

Capítulo 3

Desarrollo

3.1. Configuración del entorno de desarrollo

En el contexto de este proyecto, se ha optado por la utilización de una máquina virtual ejecutando Ubuntu 22.04. Esta decisión se sustenta en diversas ventajas que ofrece tanto el uso de máquinas virtuales, como la elección de un sistema operativo basado en Linux para el desarrollo de aplicaciones.

Ventajas del uso de máquinas virtuales

El empleo de máquinas virtuales presenta diferentes beneficios en el ámbito del desarrollo de software. Primero, una máquina virtual proporciona un entorno completamente aislado del sistema operativo principal, siendo una característica crucial para garantizar que las configuraciones específicas del proyecto no afecten ni sean afectadas por otras aplicaciones o configuraciones de la máquina host. Además, facilita la gestión de dependencias y la configuración del entorno sin riesgos de alterar el sistema subyacente. En segundo lugar, las máquinas virtuales permiten una portabilidad y reproducibilidad del entorno de desarrollo. Esto significa que el entorno configurado puede ser fácilmente replicado y desplegado en otras máquinas sin necesidad de realizar ajustes adicionales. Finalmente, el uso de máquinas virtuales ofrece una flexibilidad significativa en la gestión de recursos. Es posible asignar y modificar los recursos de hardware disponibles para la VM, como memoria RAM y capacidad de CPU, según las necesidades del proyecto. Esta adaptabilidad es crucial para optimizar el rendimiento del entorno de desarrollo en función de la carga de trabajo específica.

Ventajas de utilizar un sistema operativo Linux

La elección de Ubuntu 22.04, una distribución de Linux, como sistema operativo para la máquina virtual también es importante en el desarrollo eficiente de este proyecto. Una ventaja significativa de Linux es su naturaleza de código abierto, que facilita una amplia personalización del entorno operativo según las necesidades específicas del proyecto. Además, Linux ofrece soporte para una multitud

de herramientas de desarrollo de software, lo que es indispensable para entornos de programación. En el contexto de este proyecto, el uso de Ubuntu 22.04 ha permitido la implementación eficiente de tecnologías como Redis. Redis, que se beneficia de la facilidad de instalación y configuración en sistemas basados en Linux, proporciona una solución de almacenamiento en memoria de alta velocidad, ideal para gestionar las sesiones de usuario. Además, trabajar en Linux incentiva el uso intensivo de la terminal, una práctica que puede aumentar la eficiencia y la productividad en el desarrollo de software. La terminal de Linux es una herramienta poderosa para la automatización de tareas, la administración de sistemas y el control de versiones, facilitando así un entorno de desarrollo más controlado y eficiente.

3.1.1. Android Studio

Android Studio es el IDE oficial para el desarrollo de aplicaciones Android, desarrollado por Google y basado en IntelliJ IDEA de JetBrains. Ofrece un entorno cohesivo y especializado para el desarrollo de aplicaciones Android, integrando todas las herramientas necesarias para diseñar, desarrollar y depurar aplicaciones en Kotlin. Android Studio es recomendado por Google por su integración directa con el SDK de Android, por su amplia gama de funcionalidades y el soporte continuo para las últimas tecnologías Android.

Build gradle

```
plugins {
    id("com.android.application") version "8.4.1" apply false
    id("org.jetbrains.kotlin.android") version "1.9.24" apply false
    id("com.google.dagger.hilt.android") version "2.48" apply false
    id("androidx.navigation.safeargs.kotlin") version "2.7.5" apply false
    id("com.google.devtools.ksp") version "1.9.24-1.0.20" apply false
}
```

Figura 3.1: build.gradle de nivel de proyecto

Los plugins en la configuración de Gradle habilitan funcionalidades específicas dentro del IDE y el proceso de compilación:

- **Android Application:** Habilita la funcionalidad de aplicación Android en el proyecto.
- **Kotlin Android:** Permite la integración de Kotlin en el proyecto Android.
- **Kotlin Kapt:** Proporciona soporte para la anotación de procesadores en Kotlin, necesario para herramientas como Dagger Hilt.

- **Google Dagger Hilt Android:** Facilita la inyección de dependencias en el proyecto usando Hilt.
- **AndroidX Navigation Safe Args Kotlin:** Genera clases de ayuda para pasar datos de forma segura entre componentes de navegación.
- **Google Devtools KSP (Kotlin Symbol Processing):** Proporciona un API para el procesamiento de anotaciones de Kotlin, utilizado junto con Room para procesar anotaciones de entidades y DAOs.

```
dependencies {

    val navVersion = "2.7.7"
    val hiltVersion = "2.48"
    val retrofitVersion = "2.9.0"
    val roomVersion = "2.6.1"

    implementation("androidx.activity:activity-ktx:1.9.0")
    implementation("androidx.core:core-ktx:1.13.1")
    implementation("androidx.appcompat:appcompat:1.7.0")
    implementation("com.google.android.material:material:1.12.0")
    implementation("androidx.constraintlayout:constraintlayout:2.1.4")
    testImplementation("junit:junit:4.13.2")
    androidTestImplementation("androidx.test.ext:junit:1.1.5")
    androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1")

    // Navigation Component
    implementation("androidx.navigation:navigation-fragment-ktx:$navVersion")
    implementation("androidx.navigation:navigation-ui-ktx:$navVersion")

    // DaggerHilt
    implementation("com.google.dagger:hilt-android:$hiltVersion")
    kapt("com.google.dagger:hilt-compiler:$hiltVersion")

    // Retrofit
    implementation ("com.squareup.retrofit2:retrofit:$retrofitVersion")
    implementation ("com.squareup.retrofit2:converter-gson:$retrofitVersion")
    implementation ("com.squareup.okhttp3:logging-interceptor:4.3.1")

    // Coroutines
    implementation ("org.jetbrains.kotlinx:kotlinx-coroutines-core:1.8.0")

    // MPAndroidChart
    implementation("com.github.PhilJay:MPAndroidChart:v3.1.0")

    // Shimmer
    implementation("com.facebook.shimmer:shimmer:0.5.0")

    // Security Crypto
    implementation("androidx.security:security-crypto:1.1.0-alpha06")

    // Room
    implementation("androidx.room:room-runtime:$roomVersion")
    ksp("androidx.room:room-compiler:$roomVersion")
    implementation("androidx.room:room-ktx:$roomVersion")

    // Paging
    implementation("androidx.paging:paging-runtime-ktx:3.3.0")

    // Splash
    implementation("androidx.core:core-splashscreen:1.0.1")
}
```


Figura 3.2: build.gradle de nivel de aplicación

Dentro del build gradle del módulo de aplicación, son definidas las dependencias. Estas dependencias son esenciales para incluir bibliotecas externas y otros módulos que proporcionan funcionalidades requeridas por la aplicación.

- **AndroidX Libraries:**
  - **activity-ktx, core-ktx, appcompat:** Estas bibliotecas de Kotlin extensions proporcionan métodos de extensión que permiten un código más idiomático en Kotlin. Facilitan el manejo de componentes de la aplicación como activities y fragments, y mejoran la interoperabilidad entre Kotlin y las APIs estándar de Android.
  - **constraintlayout:** Permite el uso de ConstraintLayout, una herramienta de diseño versátil y potente que soporta interfaces complejas con una estructura de vistas plana, lo que mejora el rendimiento de renderizado.
- **Navigation Component:**
  - **navigation-fragment-ktx, navigation-ui-ktx:** Facilitan la implementación de la navegación declarativa entre los componentes de la aplicación, permitiendo una gestión más limpia y segura del flujo de navegación mediante el uso de archivos XML. Esta integración es importante para las aplicaciones que utilizan múltiples fragments.
- **Dagger Hilt:**
  - **hilt-android:** Proporciona una infraestructura completa para la inyección de dependencias en Android, permitiendo una integración más fácil de los componentes de la aplicación de manera escalable y eficiente. Reduce el boilerplate necesario para la inyección de dependencias.
  - **hilt-compiler:** Procesa las anotaciones de Hilt en tiempo de compilación, generando el código necesario para la inyección de dependencias.
- **Retrofit:**
  - **retrofit:** Es una biblioteca tipo-safe HTTP client para Android que facilita la integración con APIs, transformando la API REST en una interfaz de, en el caso de este proyecto, Kotlin.
  - **converter-gson:** Proporciona un convertidor que utiliza Gson para la serialización y deserialización de objetos, integrando JSON de manera efectiva dentro de las peticiones y respuestas HTTP.
  - **logging-interceptor:** Ofrece una herramienta de OkHttp que registra las peticiones y respuestas HTTP, útil para depurar y verificar la integridad de las comunicaciones de red.
- **Coroutines de Kotlin:**
  - **kotlin-coroutines-core:** Gestiona la asincronía y el manejo de operaciones que bloquean hilos, como las consultas de red o de bases de

datos, a través de un modelo de programación más simple y menos propenso a errores que los callbacks tradicionales.

- **MPAndroidChart:**
  - **MPAndroidChart:** Es una biblioteca para renderizar gráficos en Android, soportando una amplia variedad de tipos de gráficos con personalización extensa.
- **Shimmer:**
  - **shimmer:** Ofrece un efecto de carga para las vistas que mejora la percepción del usuario sobre las operaciones en curso, típicamente usadas en cargas de red o procesos largos en el fondo.
- **Security Crypto:**
  - **security-crypto:** Proporciona implementaciones seguras de criptografía para el almacenamiento de datos y la comunicación.
- **Room:**
  - **room-runtime, room-ktx:** Ofrece una capa de abstracción sobre SQLite para permitir un manejo más robusto y menos propenso a errores de bases de datos locales.
  - **room-compiler:** Procesa las anotaciones de las entidades y DAOs para generar código en tiempo de compilación, mejorando el rendimiento y reduciendo el potencial de errores en tiempo de ejecución.
- **Paging:**
  - **paging-runtime-ktx:** Maneja eficientemente cargas de datos segmentadas para su visualización en UI, permitiendo la gestión de grandes sets de datos de manera eficiente.
- **SplashScreen:**
  - **core-splashscreen:** Facilita la implementación de una pantalla de inicio moderna de acuerdo con las directrices más recientes de Android, mejorando la experiencia de arranque de la aplicación.

3.1.2. Visual Studio Code

Visual Studio Code es un entorno de desarrollo integrado, creado por Microsoft. Es ampliamente utilizado por desarrolladores de todo el mundo debido a su versatilidad, eficiencia y amplio soporte para diferentes lenguajes de programación y tecnologías. Visual Studio Code es particularmente apreciado en la comunidad de Python por su capacidad de adaptarse a diversas necesidades de desarrollo, desde scripts simples hasta proyectos complejos como, en el caso de este proyecto, aplicaciones web con FastAPI.

Plugings

Para optimizar Visual Studio Code para el desarrollo del proyecto, he requerido de ciertos plugins que facilitan la escritura de código, la depuración y la gestión del proyecto. A continuación, se describen los plugins seleccionados para el desarrollo del proyecto:

- **Pyhton:**
  - **Python:** Plugin oficial proporcionado por Microsoft para el soporte de Python en Visual Studio Code. Facilita características como el resaltado de sintaxis, la navegación por el código, el autocompletado inteligente y el soporte para entornos virtuales. Es fundamental para cualquier proyecto Python, ya que sienta las bases para integrar otros plugins y herramientas.
  - **Pylance:** Extensión que mejora las capacidades de Python en Visual Studio Code. Proporciona un servicio de lenguaje rápido y rico en características, basado en el servidor de lenguaje Pyright de Microsoft. Ofrece un autocompletado superior, análisis estático de código, refactorización, y soporte de tipado mejorado, lo cual es crucial para mantener un código limpio y eficiente en proyectos complejos.
  - **Python Debugger:** Permite la depuración integrada en VS Code. Es vital para el desarrollo y la depuración efectiva de aplicaciones, ya que permite a los desarrolladores ejecutar su código paso a paso, inspeccionar variables y evaluar expresiones en tiempo real, facilitando la identificación y corrección de errores.
  - **Autopep8:** Herramienta que formatea automáticamente el código Python según las recomendaciones del PEP 8, la guía de estilo para el código Python. Este plugin asegura que el código sea no solo funcional sino también limpio y consistente, lo cual es esencial para la mantenibilidad y legibilidad del mismo.
- **Bases de datos:**
  - **MongoDB:** Este plugin facilita la integración y el manejo de bases de datos MongoDB dentro de Visual Studio Code. Permite conectar, explorar y manipular bases de datos MongoDB directamente desde el IDE, simplificando el desarrollo de aplicaciones que dependen de este tipo de bases de datos no SQL.
  - **Database Client JDBC y Redis:** Database Client JDBC permite conectar y manipular bases de datos que soportan JDBC, facilitando la integración con bases de datos SQL directamente desde el IDE. Por otro lado, Redis ofrece soporte para trabajar con un almacén de datos en memoria, permitiendo a los desarrolladores interactuar con Redis para operaciones de caché y almacenamiento de clave-valor.
- **Thunder Client:**

    - **Thunder Client:** Cliente REST integrado en Visual Studio Code que permite a los desarrolladores probar sus APIs directamente desde el IDE. Es una herramienta útil para desarrollar y probar endpoints sin necesidad de recurrir a software externo como Postman.

- **GitLens:**
    - **GitLens:** GitLens mejora la funcionalidad de Git en VS Code, ofreciendo visualizaciones avanzadas, navegación mejorada y capacidades de seguimiento de cambios en el código. Esto es especialmente útil en entornos colaborativos, donde el seguimiento de contribuciones individuales y la comprensión de la historia del código son cruciales. A su vez, es útil para entornos individuales de forma de salvaguarda de código en la nube.
- **Docker:**
    - **Docker:** Proporciona soporte para trabajar con contenedores Docker, permitiendo a los desarrolladores construir, administrar y desplegar contenedores directamente desde el IDE. Esto es particularmente útil para proyectos que utilizan microservicios o que necesitan un entorno de desarrollo replicable..

Requirements

Figura 3.3: Archivo requirements.txt

En el contexto del proyecto, se requiere un conjunto específico de librerías para aprovechar al máximo sus capacidades. A continuación, se detallan cada una de las librerías requeridas para el backend del proyecto, explicando su propósito y funcionalidad.

- **aioredis:** Biblioteca de cliente Python asincrónica para Redis, que aprovecha asyncio (la biblioteca estándar de Python para escribir código concurrente). Se utiliza para interactuar de manera asincrónica con bases de datos Redis, permitiendo operaciones no bloqueantes que son esenciales para el rendimiento en entornos de aplicaciones web que manejan múltiples usuarios y conexiones simultáneas.
- **app:** Dependencia típica que se refiere al paquete de la aplicación actual.
- **fastapi:** Proporciona herramientas para crear APIs robustas y eficientes, con soporte automático de documentación y validación de datos, facilitando el desarrollo rápido y la escritura de código menos propenso a errores.
- **motor:** Permite realizar operaciones en la base de datos MongoDB de manera asincrónica, mejorando la capacidad de manejo de concurrencia y la eficiencia de las aplicaciones I/O-bound.
- **passlib:** Biblioteca integral de Python para la gestión de contraseñas. Ofrece herramientas para la creación y verificación de contraseñas utilizando varios algoritmos de hash, lo cual es crucial para la seguridad en las aplicaciones web.
- **pydantic:** Biblioteca de validación y manejo de datos basada en anotaciones de tipo Python. Se utiliza para definir cómo los datos deben ser parseados y validados en un formato estructurado, asegurando que los datos entrantes a la aplicación cumplan con los formatos esperados, lo cual es vital para la integridad y seguridad de la aplicación.
- **pydantic_settings:** Extensión de Pydantic diseñada para la gestión de configuraciones de aplicaciones. Facilita la carga y validación de configuraciones de aplicaciones desde múltiples fuentes, asegurando una configuración robusta y fácil de mantener.
- **pymongo:** Biblioteca oficial de MongoDB para Python. Proporciona herramientas para trabajar con MongoDB, facilitando operaciones como la inserción, actualización, y recuperación de documentos de bases de datos MongoDB.
- **python-dotenv:** biblioteca que permite cargar variables de entorno desde un archivo `.env`. Esencial para manejar configuraciones sensibles, como credenciales de base de datos o claves secretas, fuera del código fuente, mejorando la seguridad y la flexibilidad en diferentes entornos de desarrollo y producción.
- **python-multipart:** Permite manejar datos multipart/form-data en Python, que es un formato usado para la carga de archivos.
- **python-jose:** JOSE (JavaScript Object Signing and Encryption) es una biblioteca para firmar y verificar JSON Web Tokens. Importante para la autenticación y la transmisión segura de información entre clientes y servidores en aplicaciones web.

3.2. Diseño y Modelado

En este apartado se detallarán los aspectos fundamentales del diseño y modelado del sistema desarrollado para la aplicación móvil y los servicios REST asociados, cuyo objetivo principal es la recogida de datos nutricionales. El diseño y modelado constituyen una fase crítica en el desarrollo de cualquier aplicación, ya que permiten establecer una base sólida sobre la cual se construirá todo el sistema. Esta sección se desglosa en tres subapartados esenciales: Modelado de Entidades, Arquitectura del Sistema y Seguridad y Autenticación. Cada uno de estos componentes desempeña un papel vital en la definición de la estructura y funcionamiento del sistema.

3.2.1. Modelado de Entidades

El modelado de entidades es un paso indispensable en el diseño de sistemas de bases de datos, ya que define la estructura y las relaciones de los datos que serán almacenados y gestionados por el sistema. En el contexto del proyecto para la recogida de datos nutricionales, el modelado de entidades se centra en identificar los objetos del mundo real que deben ser representados en la base de datos y en establecer las relaciones entre ellos. Este proceso se realiza a través de la definición de clases y atributos que reflejan la información relevante para la aplicación.

Identificación de entidades

Cada entidad representa un objeto o concepto significativo dentro del dominio de la aplicación. En el contexto de este proyecto, las principales entidades identificadas son:

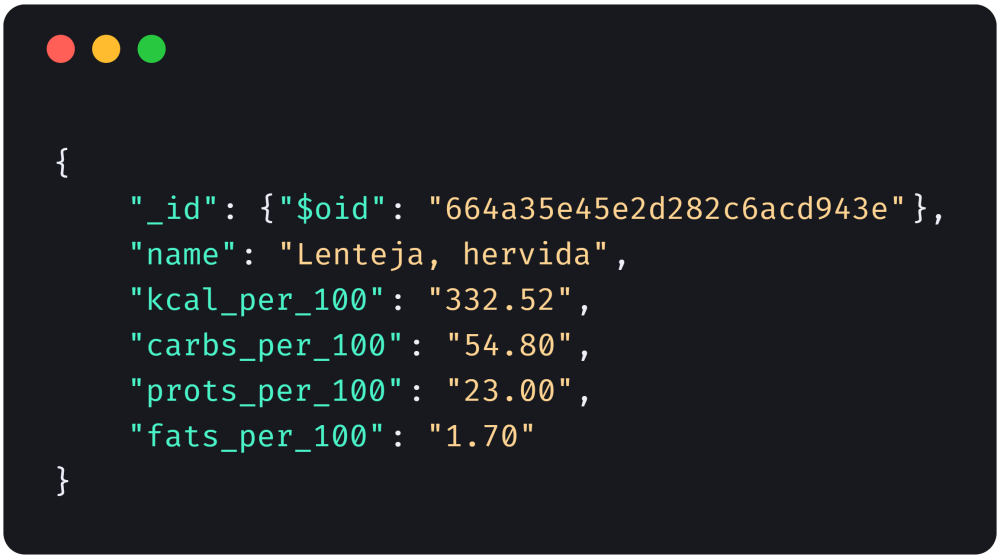

```
{
    "_id": {"$oid": "666a34f94f7abddf996854a0"},
    "username": "johndoe",
    "email": "johndoe@example.com",
    "hashed_password": "$2b$12$YmhIbK7f9F3rq2raJ5omqOIy1cPvjrFyfo/ivJjFQvR3XVFJ3aXW6",
    "order_meal": ["Desayuno", "Almuerzo", "Comida", "Merienda"],
    "profile": {
        "age": "23",
        "gender": "0",
        "height": "183",
        "physical_activity_level": "2",
        "weight": "91"
    }
}
```


Figura 3.4: Ejemplo entidad User

- **User:** Representa a cada persona que utiliza la aplicación.
  - **`_id`:** Identificador único del usuario.
  - **`username`:** Nombre de usuario.
  - **`email`:** Dirección de correo electrónico del usuario.
  - **`hashed_password`:** Contraseña encriptada para la autenticación del usuario.
  - **`order_meal`:** Lista de nombres de comidas ordenadas, guardadas por el usuario.
  - **`profile`:** Diccionario con información adicional del perfil del usuario, incluyendo edad, género, altura, nivel de actividad física y peso.

Figura 3.5: Ejemplo entidad Meal

- **Meal:** Representa una comida que contiene varios alimentos.
  - **`_id`:** Identificador único de la comida.
  - **`user_id`:** Identificador del usuario que registró la comida.
  - **`date`:** Fecha en que se registró la comida.
  - **`name`:** Nombre de la comida.
  - **`index`:** Índice de la comida en el orden diario.
  - **`items`:** Lista de ítems que componen la comida, cada uno con una referencia a un alimento y su cantidad en gramos.



```
{
    "_id": {"$oid": "664a35e45e2d282c6acd943e"},
    "name": "Lenteja, hervida",
    "kcal_per_100": "332.52",
    "carbs_per_100": "54.80",
    "prots_per_100": "23.00",
    "fats_per_100": "1.70"
}
```

Figura 3.6: Ejemplo entidad Food

- **Food:** Incluye todos los alimentos registrados en la base de datos.
  - **`_id`:** Identificador único del alimento.
  - **`name`:** Nombre del alimento.
  - **`kcal_per_100`:** Kilocalorías por cada 100 gramos del alimento.
  - **`carbs_per_100`:** Carbohidratos por cada 100 gramos del alimento.
  - **`prots_per_100`:** Proteínas por cada 100 gramos del alimento.
  - **`fats_per_100`:** Grasas por cada 100 gramos del alimento.

En este proyecto, se utiliza una base de datos NoSQL, específicamente MongoDB, para almacenar los datos. MongoDB es una base de datos orientada a documentos que permite almacenar datos en estructuras flexibles similares a formato JSON. Esta flexibilidad es ideal para aplicaciones que requieren esquemas de datos dinámicos y escalabilidad horizontal.

Cada entidad en nuestro modelo de datos se almacena en una colección separada dentro de la base de datos MongoDB:

Usuarios: Se almacenan en la colección users. Comidas: Se almacenan en la colección meals. Alimentos: Se almacenan en la colección foods.

MongoDB utiliza identificadores únicos (_id) para cada documento en una colección, lo que permite acceder y gestionar los datos de manera eficiente. Además, las relaciones entre documentos se manejan a través de referencias, almacenando identificadores de documentos relacionados dentro de los mismos.

Por ejemplo, en la entidad Meal, el campo user_id es una referencia al identificador del documento User que registró la comida, y cada ítem en el campo items contiene un identificador de un Food y la cantidad consumida.

Esta estructura permite que la base de datos sea flexible y escalable, adaptándose a las necesidades cambiantes de la aplicación sin requerir esquemas rígidos. Además, la orientación a documentos de MongoDB facilita la consulta y manipulación de datos complejos, lo que es esencial para una aplicación que maneja información detallada sobre nutrición y consumo de alimentos.

Base de Datos SQLite de la Aplicación Móvil

En el contexto del desarrollo de aplicaciones móviles, la persistencia de datos es un aspecto fundamental para garantizar que la información se mantenga accesible y coherente incluso cuando la aplicación no tiene conexión a Internet. En este proyecto, para la recogida de datos nutricionales, se ha implementado una base de datos SQLite para gestionar el almacenamiento local de la información. SQLite es una biblioteca de software que proporciona una base de datos relacional, liviana y autoconfigurada, ideal para aplicaciones móviles debido a su simplicidad y eficiencia.

La base de datos SQLite en la aplicación se organiza mediante el uso de la biblioteca Room de Android, que actúa como una capa de abstracción sobre SQLite, facilitando el manejo de la base de datos mediante objetos Kotlin. Room proporciona una API robusta y sencilla para realizar operaciones de base de datos y asegura la verificación del esquema en tiempo de compilación.

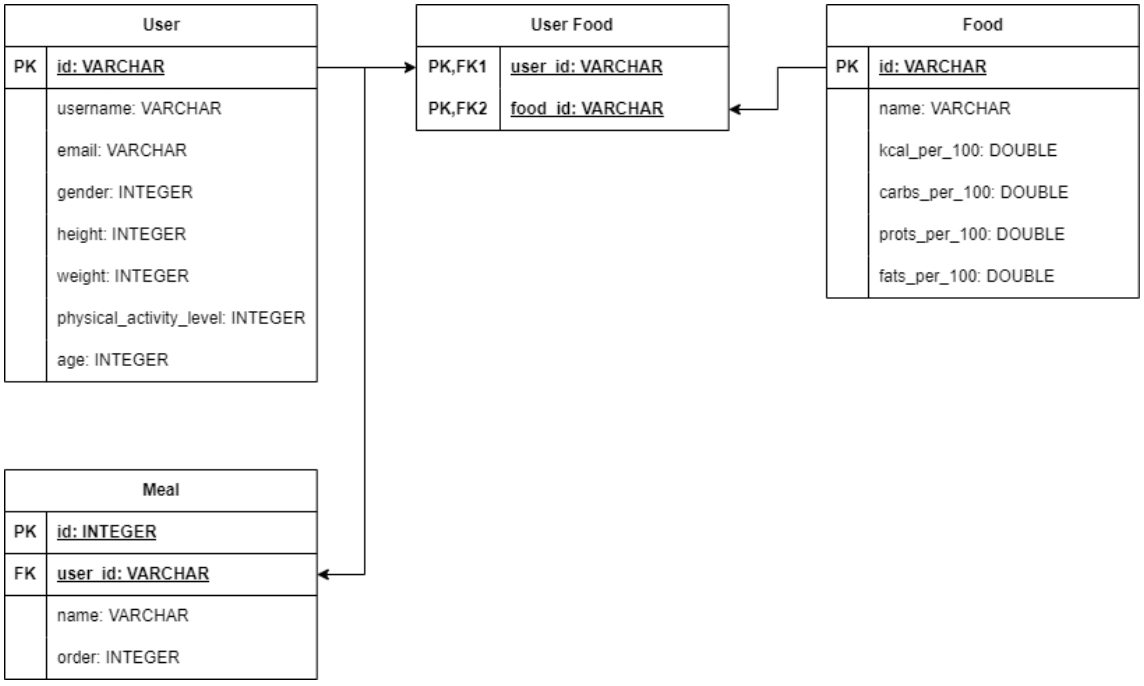


Figura 3.7: Esquema de la base de datos SQLite

Las principales entidades de la base de datos se definen como clases de datos en Kotlin, utilizando anotaciones de Room para mapearlas a tablas de la base de datos. A continuación, se describen las principales entidades y sus atributos:

	id	username	email	gender	height	weight	physical_activity_level	age
	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Fi...
1	666a34f94f7abddf996854a0	johndoe	johndoe@example.com	0	183	91	2	23

Figura 3.8: Tabla de usuarios de la base de datos de la aplicación

- **User Entity:** Representa a cada persona que utiliza la aplicación.
  - **`id`:** Identificador único del usuario.
  - **`username`:** Nombre de usuario.
  - **`email`:** Dirección de correo electrónico del usuario.
  - **`gender`:** Género del usuario.
  - **`height`:** Altura del usuario.
  - **`weight`:** Peso del usuario.
  - **`physical_activity_level`:** Nivel de actividad física ejercida por el usuario.
  - **`age`:** Edad del usuario.

	id	user_id	order	name
	Filtro	Filtro	Filtro	Filtro
1	77	666a34f94f7abddf996854a0	0	Desayuno
2	78	666a34f94f7abddf996854a0	1	Almuerzo
3	79	666a34f94f7abddf996854a0	2	Comida
4	80	666a34f94f7abddf996854a0	3	Merienda

Figura 3.9: Tabla de comidas de la base de datos de la aplicación

- **Meal Entity:** Representa cada comida guardada por el usuario.
  - **`id`:** Identificador único de la comida.
  - **`user_id`:** Identificador único del usuario que añade la comida.
  - **`name`:** Nombre de la comida.
  - **`order`:** Posición que ocupa la comida respecto a las demás.

	id	name	kcal_per_100	carbs_per_100	prots_per_100	fats_per_100
	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro
1	664a35e45e2d282c6acd9373	Refresco, sabor cola, bajo en calorías	0.41	0.1	0.0	0.0
2	664a35e45e2d282c6acd9370	"Refresco tipo ""tónica"""	37.54	9.2	0.0	0.0
3	664a35e45e2d282c6acd928b	Kebab	304.81	16.7	8.8	7.9
4	664a35e45e2d282c6acd943e	Lenteja, hervida	332.52	54.8	23.0	1.7
5	664a35e45e2d282c6acd9129	Queso de tiétar	398.44	0.0	21.48	35.0

Figura 3.10: Tabla de alimentos de la base de datos de la aplicación

- **Food Entity:** Representa cada alimento que ha sido añadido alguna vez como favorito.
  - **`id`:** Identificador único del alimento.
  - **`name`:** Nombre del alimento.
  - **`kcal_per_100`:** Valor energético del alimento por cada 100 gramos.
  - **`carbs_per_100`:** Hidratos de carbono del alimento por cada 100 gramos.
  - **`prots_per_100`:** Proteínas del alimento por cada 100 gramos.
  - **`fats_per_100`:** Grasas del alimento por cada 100 gramos.

	user_id	food_id
	Filtro	Filtro
1	666a34f94f7abddf996854a0	664a35e45e2d282c6acd9373
2	666a34f94f7abddf996854a0	664a35e45e2d282c6acd9370
3	666a34f94f7abddf996854a0	664a35e45e2d282c6acd928b
4	666a34f94f7abddf996854a0	664a35e45e2d282c6acd943e
5	666a34f94f7abddf996854a0	664a35e45e2d282c6acd9129

Figura 3.11: Tabla de relación usuario - alimentos de la base de datos de la aplicación

- **User Food Entity:** Representa la relación de cada alimento favorito de cada usuario.
  - **`user_id`:** Identificador único del usuario.
  - **`food_id`:** Identificador único del alimento.

3.2.2. Arquitectura del Sistema

El proyecto está diseñado con una arquitectura cliente-servidor, donde el frontend es una aplicación móvil que se comunica con un backend para gestionar los datos nutricionales de los usuarios. La arquitectura del proyecto se puede dividir en varios componentes clave: el frontend, el backend, la base de datos y

el sistema de cache. A continuación, se describe cada uno de estos componentes y su papel en la arquitectura general del sistema.

Frontend

El frontend del proyecto es una aplicación móvil que sigue los principios de Clean Architecture y el patrón Modelo, Vista, Vista-Modelo. A continuación, se describe el papel de cada directorio dentro del subdirectorio `main`, destacando su importancia y función en la arquitectura general del proyecto.

- **Directorio `core`:** El directorio core contiene componentes esenciales y reutilizables de la aplicación que son fundamentales para su funcionamiento. Estos componentes incluyen la inyección de dependencias, excepciones personalizadas y extensiones de clases.

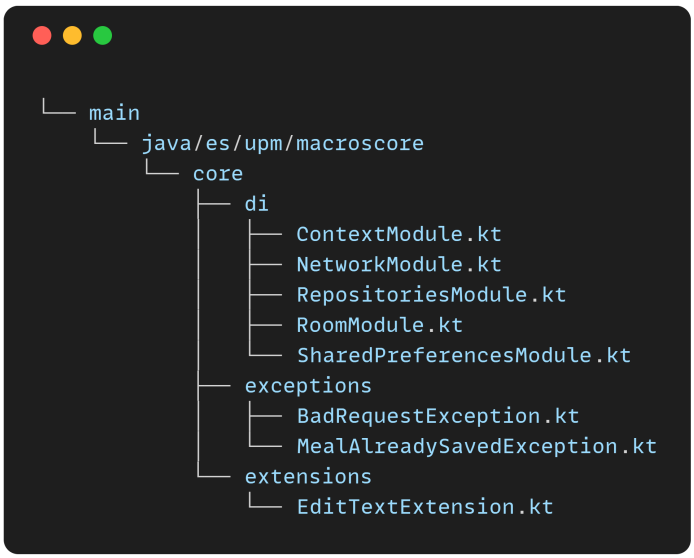


Figura 3.12: Directorio core

- **`di` (Dependency Injection):** Este subdirectorio se encarga de configurar la inyección de dependencias utilizando Dagger-Hilt. Aquí se definen los módulos que proporcionan instancias de las dependencias necesarias en toda la aplicación, como el contexto de la aplicación, servicios de red, repositorios y configuraciones de la base de datos.
- **`exceptions`:** Almacena las excepciones personalizadas que se utilizan en la aplicación para manejar errores específicos de manera estructurada. Estas excepciones permiten un manejo de errores claro y predecible, mejorando la calidad del código.
- **`extensions`:** Contiene extensiones de clases estándar de Kotlin. Las extensiones mejoran la funcionalidad de estas clases, agregando fun-

ciones, y simplifican su uso en toda la aplicación, promoviendo el código limpio y reutilizable.

- **Directorio `data`:** El directorio data es responsable de la gestión de datos en la aplicación. Este directorio implementa la lógica de acceso a datos y es crucial para separar la lógica de negocio del almacenamiento de datos. Se subdivide en varios subdirectorios que organizan el acceso a la red, la base de datos local y los mapeos de datos.

Figura 3.13: Directorio data I: implementation & local

- **`implementation`:** Contiene las implementaciones de los repositorios que interactúan con las fuentes de datos. Estos repositorios implementan las interfaces definidas en el dominio y proporcionan una capa de abstracción sobre el acceso a los datos.
- **`local`:** Gestiona el acceso a la base de datos local utilizando Room. Este subdirectorio incluye definiciones de la base de datos, DAOs (Data Access Objects) y entidades que representan las tablas de la base de datos.

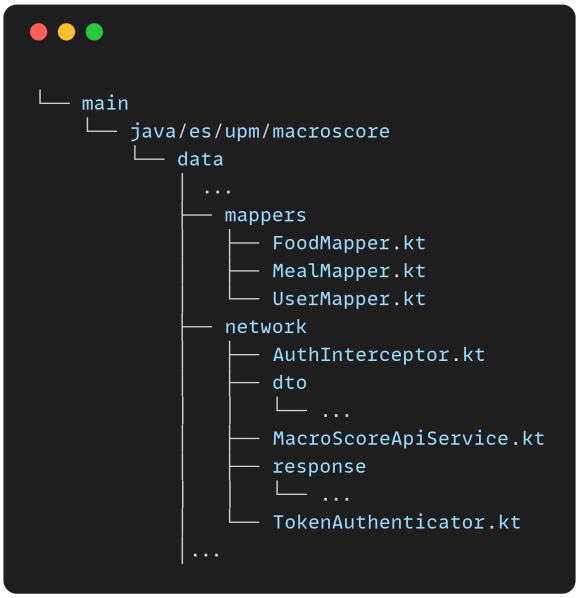


Figura 3.14: Directorio data II: mappers & network

- **`mappers`:** Define los mapeos entre diferentes capas de datos, como las entidades de la base de datos, los DTOs de red, las respuestas del consumo de APIs y los modelos de dominio. Estos mapeos aseguran que los datos se transformen correctamente entre la capa de data y la de dominio.
- **`network`:** El directorio network es una parte principal del directorio data, ya que maneja todas las interacciones de la aplicación con servicios externos a través de la red. Este directorio es responsable de configurar la comunicación HTTP, definir las interfaces de la API, consumir las APIs, manejar la autenticación y transformar los datos de la red en modelos utilizables por la aplicación.

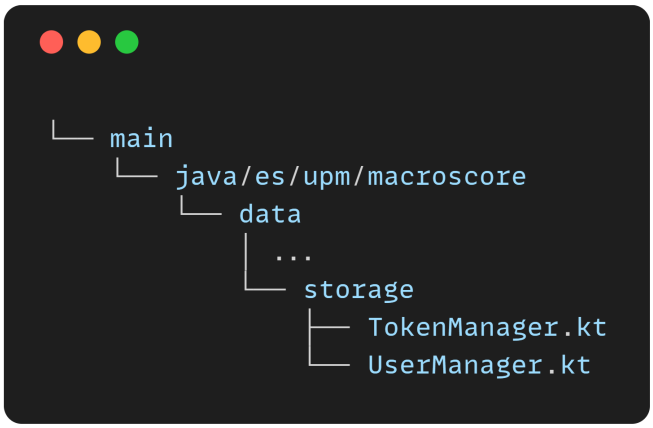


Figura 3.15: Directorio data III: storage

- **`storage`:** Gestiona el almacenamiento persistente de datos simples como tokens y preferencias del usuario. Este almacenamiento se utiliza para datos que necesitan ser accesibles rápidamente y no requieren una estructura compleja de base de datos.

- **Directorio `domain`:** El directorio domain es el corazón de la lógica de negocio de la aplicación. Este directorio contiene los modelos de dominio, los repositorios de dominio y los casos de uso. La independencia de este directorio de otras capas asegura que la lógica de negocio se mantenga pura y fácil de probar.

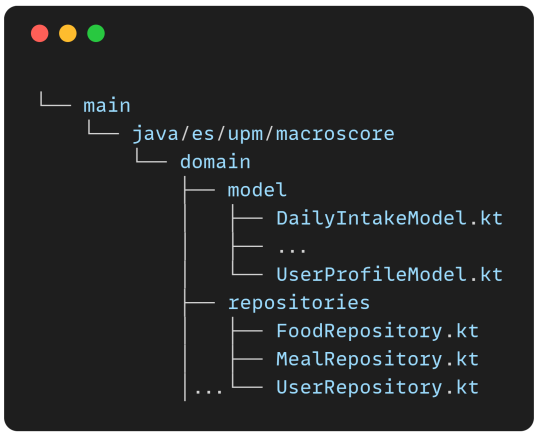


Figura 3.16: Directorio domain I: model & repositories

- **`model`:** Contiene los modelos de dominio que representan las entidades de negocio. Estos modelos son independientes de cualquier framework específico y encapsulan la lógica y las reglas de negocio.

- **`repositories`:** Define las interfaces de los repositorios que el dominio

utiliza para acceder a los datos. Estas interfaces son implementadas en la capa de datos y permiten la inyección de dependencias, asegurando que la lógica de negocio no dependa directamente de los detalles de implementación del almacenamiento de datos.

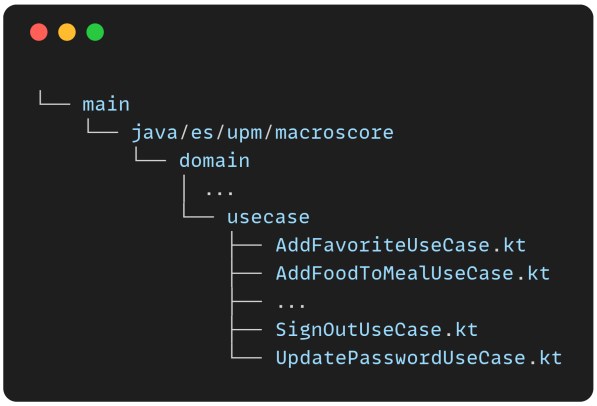


Figura 3.17: Directorio domain II: usecase

- **`usecase`:** Define los casos de uso que encapsulan la lógica de negocio de la aplicación. Cada caso de uso representa una acción específica que el usuario puede realizar, y coordina las interacciones entre los repositorios y los modelos de dominio.

- **Directorio `ui`:** El directorio ui maneja la interfaz de usuario de la aplicación. Implementa las actividades, fragmentos y viewmodels siguiendo el patrón MVVM. Este directorio organiza el código de la UI en diferentes secciones según la funcionalidad, facilitando la mantenibilidad y la escalabilidad del código de la interfaz de usuario.

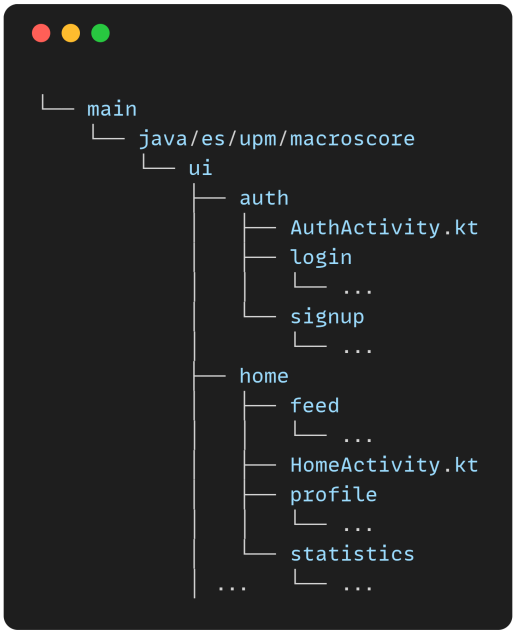


Figura 3.18: Directorio ui I: auth & home

- **`auth`:** Gestiona la autenticación de usuarios, incluyendo el inicio de sesión y el registro. Este subdirectorio contiene los viewmodels y las vistas relacionadas con la autenticación, asegurando una separación clara de responsabilidades.
- **`home`:** Contiene las pantallas principales de la aplicación, como el feed de alimentos y comidas. Este subdirectorio organiza los viewmodels y las vistas que gestionan la pantalla de inicio, la pantalla de estadísticas y el perfil de usuario, además de la navegación principal de la aplicación entres las pantallas dichas.

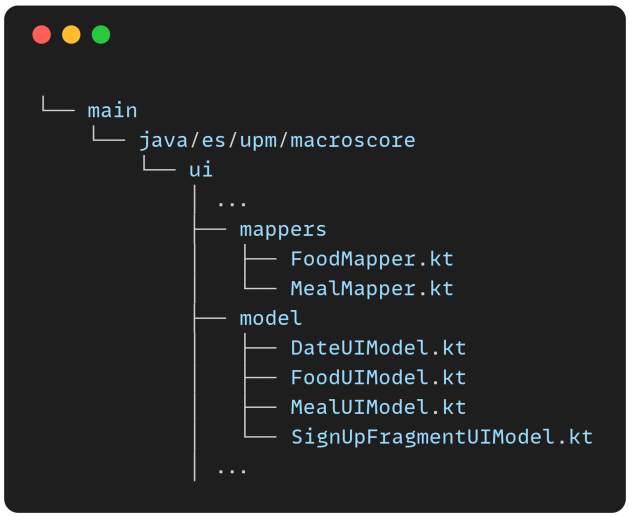


Figura 3.19: Directorio ui II: mappers & model

- **`mappers`:** Define los mapeos entre la capa de presentación y los modelos de dominio. Estos mapeos aseguran que los datos se transformen correctamente entre las diferentes capas de la aplicación.
- **`model`:** Define los modelos de la capa de presentación. Los modelos de vista contienen los datos que la UI necesita para mostrar la información al usuario. Esto incluye no solo los datos de negocio, sino también estados específicos de la UI, como indicadores de carga, mensajes de error y datos formateados.

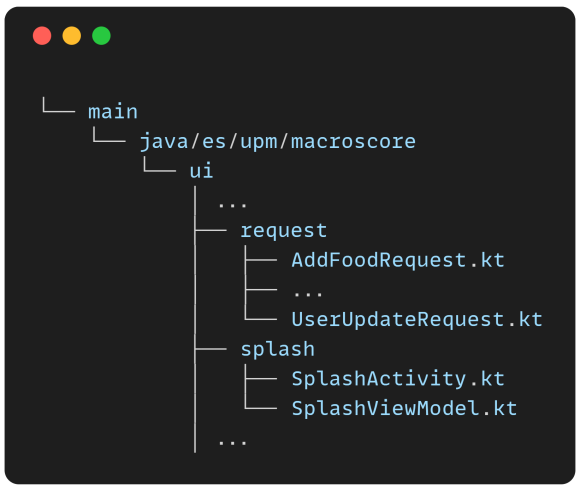


Figura 3.20: Directorio ui III: request & splash

- **`request`:** Contiene clases que representan las solicitudes que se envían desde la UI a otras capas de la aplicación, como la capa de dominio o la capa de red. Estas solicitudes encapsulan los datos necesarios para realizar operaciones específicas, como crear, actualizar o eliminar recursos.
- **`splash`:** Define la pantalla de carga inicial de la aplicación. Aquí se implementa la lógica que se ejecuta cuando la aplicación se inicia, asegurando que todos los recursos necesarios estén listos antes de que el usuario interactúe con la aplicación.

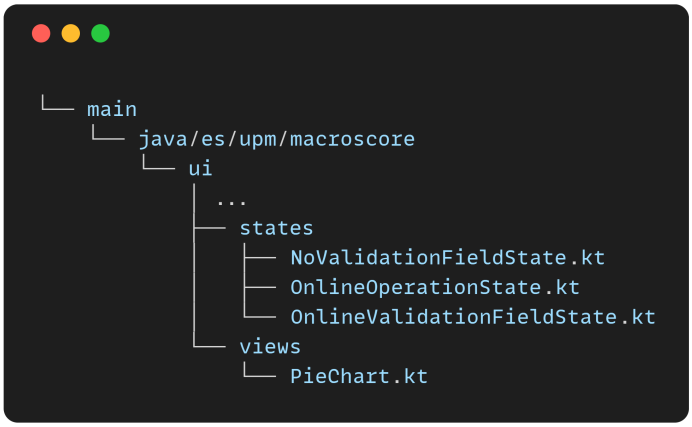


Figura 3.21: Directorio ui IV: states & views

- **`states`:** Contiene las clases que representan los diferentes estados de las vistas, facilitando el manejo de diferentes situaciones de la UI, como la validación de campos y las operaciones online.
- **`views`:** Define las vistas personalizadas, en el contexto de este proyecto, gráficos de pastel.

Backend

La estructura de directorios del backend está diseñada para proporcionar una separación clara de responsabilidades, lo que facilita el desarrollo, mantenimiento y escalabilidad del sistema. Cada directorio desempeña un papel importante en la organización del código y en la implementación de la lógica de la aplicación:

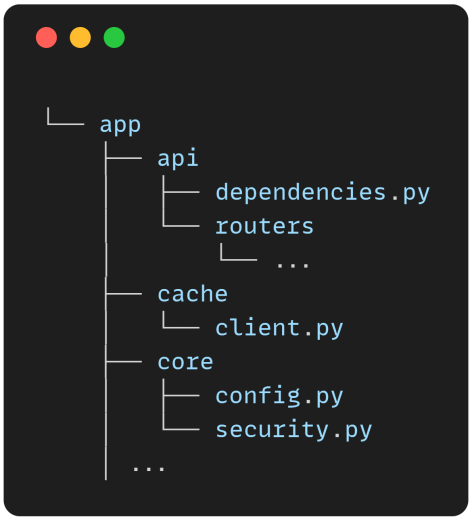


Figura 3.22: Directorio app I: api, cache & core

- **`api`:** El directorio api es responsable de definir las interfaces de programación de aplicaciones (API) que la aplicación expone. Estas interfaces permiten la comunicación entre el cliente (la aplicación móvil) y el servidor (backend). El directorio api está dividido en dos partes principales:
  - **`dependencies`:** Gestiona las dependencias necesarias para los routers de la API, asegurando que los componentes necesarios estén disponibles.
  - **`routers`:** Contiene los archivos de enrutamiento que definen los diferentes endpoints de la API. Cada archivo maneja una parte específica de la funcionalidad de la aplicación, como la gestión de alimentos, usuarios, comidas, autenticación, etc.
- **`cache`:** El directorio cache se encarga de la implementación de la lógica de almacenamiento en caché en Redis. El almacenamiento en caché mejora el rendimiento de la aplicación al guardar temporalmente datos que se acceden con frecuencia, reduciendo así el tiempo de respuesta y la carga en la base de datos.
- **`core`:** El directorio core contiene la configuración central de la aplicación y los componentes de seguridad esenciales para el funcionamiento seguro del sistema. Aquí se definen los parámetros de configuración global y las estrategias de seguridad como la autenticación y la autorización.

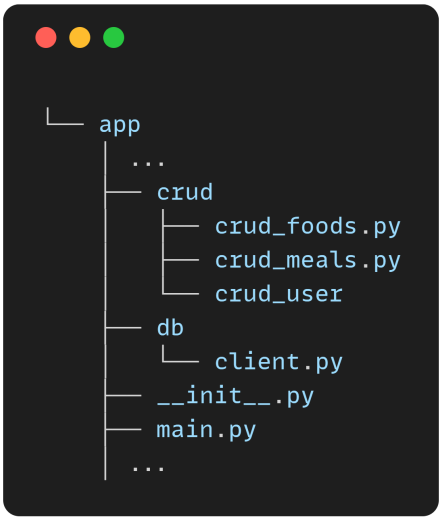


Figura 3.23: Directorio app II: crud, db, __init__ & main

- **crud:** El directorio crud (Create, Read, Update, Delete) implementa las operaciones básicas de gestión de datos para las diferentes entidades de la aplicación. Este directorio organiza el código que interactúa directamente con la base de datos de MongoDB para realizar estas operaciones de manera eficiente.
- **db:** El directorio db se encarga de la configuración y gestión de la conexión a la base de datos. Este directorio asegura que la aplicación pueda conectarse y comunicarse eficazmente con la base de datos, manejando la inicialización y el mantenimiento de la conexión.
- **__init__.py:** El archivo __init__.py es un componente fundamental en cualquier proyecto Python, especialmente en aquellos que están estructurados en múltiples módulos y paquetes. Su propósito principal es convertir un directorio en un paquete de Python, permitiendo que sus subdirectorios y archivos sean importados como módulos.
- **main.py:** Configura y lanza la aplicación FastAPI, gestiona las conexiones a las bases de datos y define los endpoints de la API mediante routers. Este archivo es crucial para iniciar y gestionar el ciclo de vida de la aplicación, asegurando que todas las dependencias y conexiones estén correctamente configuradas y gestionadas.

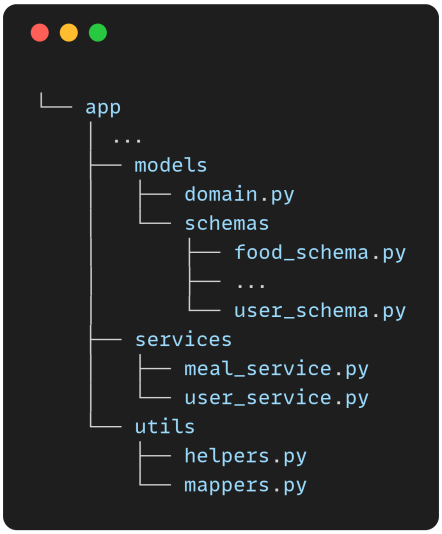


Figura 3.24: Directorio app III: models, services & utils

- **`models`:** El directorio models contiene las definiciones de los modelos de datos y los esquemas que representan las estructuras de datos utilizadas en la aplicación. Este directorio se divide en:
  - **`domain`:** Define los modelos de dominio que representan las entidades de negocio de la aplicación.
  - **`schemas`:** Contiene los esquemas Pydantic para la validación y serialización de datos, asegurando que los datos entrantes y salientes cumplan con las expectativas definidas.
- **`services`:** El directorio services implementa la lógica de negocio de la aplicación. Aquí se definen los servicios que encapsulan las operaciones complejas y las reglas de negocio que no se manejan directamente en los endpoints de la API, facilitando la separación de preocupaciones.
- **`utils`:** El directorio utils contiene funciones auxiliares y utilitarias que son reutilizables en diferentes partes de la aplicación. Estas utilidades ayudan a simplificar y estandarizar tareas comunes a lo largo del código, mejorando la eficiencia y la mantenibilidad.

3.3. Implementación

3.3.1. Aplicación Cliente

La aplicación cliente es la interfaz con la que interactúan los usuarios para gestionar sus datos nutricionales. Está desarrollada en Kotlin para Android y sigue el patrón de arquitectura Model-View-ViewModel para asegurar una separación clara de responsabilidades y facilitar la mantenibilidad. La aplicación consta de

varias pantallas clave, cada una enfocada en proporcionar una experiencia de usuario intuitiva y eficiente.

Pantalla Splash

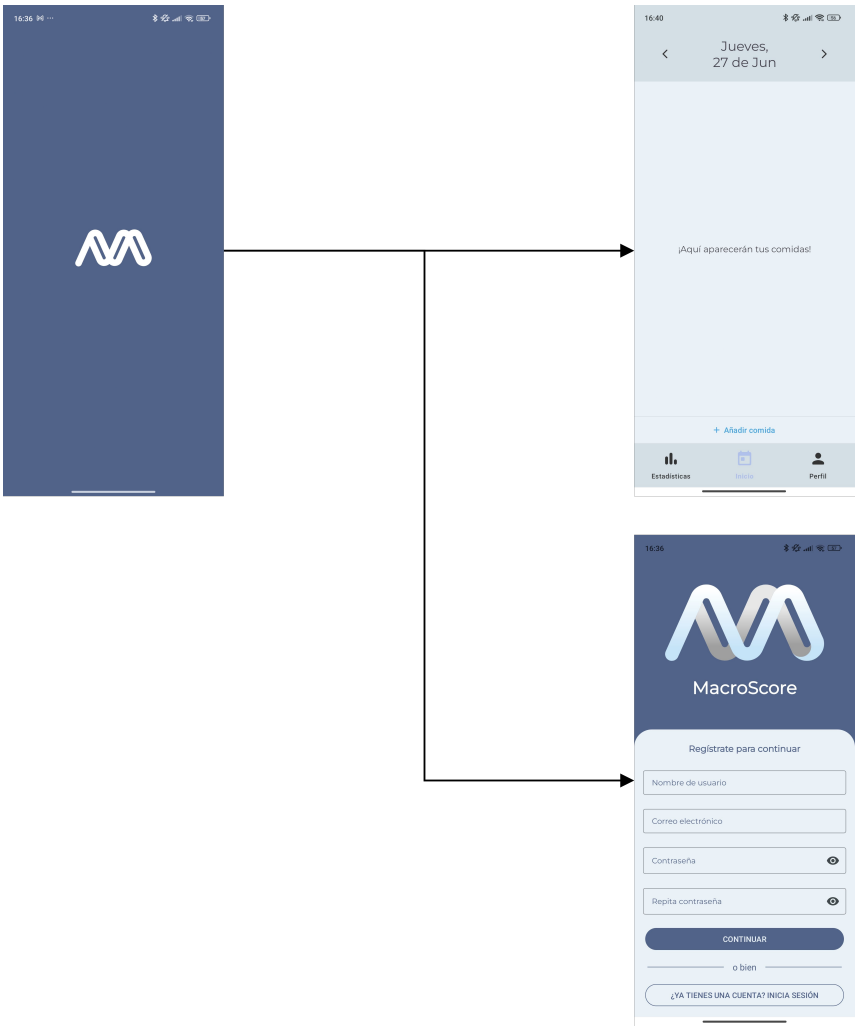


Figura 3.25: Flujo de navegación de la pantalla Splash

La pantalla Splash Screen es la primera que se muestra al abrir la aplicación. Su función principal es verificar el estado de autenticación del usuario.

- **Búsqueda de tokens de sesión:** Al iniciarse, la pantalla busca los tokens de sesión almacenados en el dispositivo. Estos tokens sirven para autenticar al usuario sin necesidad de ingresar nuevamente sus credenciales.
- **Petición al backend:** Con los tokens de sesión, la aplicación realiza una petición al backend para verificar si los tokens siguen siendo válidos y activos.
- **Navegación condicional:**

- Si los tokens son válidos, la aplicación navega automáticamente a la pantalla principal (Home), permitiendo un acceso rápido y sin interrupciones a las funcionalidades de la aplicación.
- Si los tokens no son válidos o han expirado, la aplicación redirige al usuario a la pantalla de autenticación, donde deberá ingresar sus credenciales nuevamente.

Pantalla de Autenticación

La pantalla de autenticación permite a los usuarios iniciar sesión en la aplicación o registrarse si no tienen una cuenta. Esta pantalla utiliza un mapa de navegación para moverse entre diferentes fragmentos que componen el flujo de autenticación.

Figura 3.26: Flujo de navegación de la pantalla de Autenticación

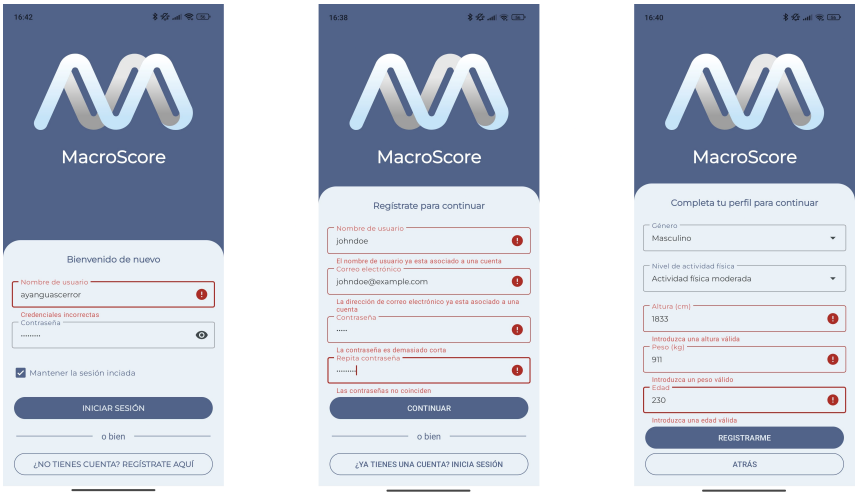


Figura 3.27: Ejemplo de errores de los datos de entrada

- **Fragment de Login:**
  - **Validación de credenciales:** Al ingresar sus credenciales, la aplicación verifica su validez a través de una petición al backend. Esta petición guarda los tokens de sesión recibidos y a continuación, manda otra petición para recoger los datos del usuario y almacenarlos en la base de datos de la aplicación.
  - **Mantener la sesión iniciada:** Si el usuario desea no hacer login cuando le caduque el access token, puede marcar la casilla de Mantener la sesión iniciada, lo que indica al backend que envíe un token de refresco.
  - **Navegación a Home:** Si las credenciales son correctas, el usuario es redirigido a la pantalla Home. Si no lo son, se muestra un error en el campo del nombre de usuario indicando que las credenciales son incorrectas.
- **Primer Fragment de Signup:**
  - **Campos de texto:** Incluye campos para email, nombre de usuario, contraseña y confirmación de contraseña.
  - **Principales comidas:** Se incluye una lista con los principales nombres de las comidas, que en español son: Desayuno, Almuerzo, Comida, Merienda y Cena. Pulsando en cualquiera de ellos, se establece como nombre en el text field.
  - **Validación:**
    - **Email:** Se valida que no esté ya en uso y que tenga formato de email. Si no cumple estas premisas, se muestra un error en el campo correspondiente.
    - **Nombre de usuario:** Se valida que no esté ya en uso, que no use carácteres especiales y no supere un número determinado de caracteres. Si no cumple estas premisas, se muestra un error en el campo correspondiente.
    - **Contraseña y repetición de contraseña**: Se verifica que tenga al menos 8 caracteres y que la confirmación coincida con la contraseña ingresada.
  - **Navegación al Segundo Fragmento:** Si todos los campos son válidos, el usuario puede navegar al siguiente fragmento para completar el registro.
- **Segundo Fragment de Signup:**
  - **Campos de texto:** Incluye campos para género, nivel de actividad física, altura, peso y edad.
  - **Validación:**

- **Género y nivel de actividad física:** Se validan mediante menús desplegables que aseguran que la opción seleccionada esté dentro de un conjunto predefinido.
- **Altura, peso y edad:** Se verifica que las entradas sean realistas y estén dentro de un rango aceptable.

- **Navegación a la pantalla Home:** Si todos los campos son válidos, se hace una petición al servidor para almacenar el nuevo usuario, tras ello se hace automáticamente la petición de login y sigue el mismo proceso que en la pantalla de inicio de sesión.

Pantalla Home

La pantalla Home es la pantalla principal de la aplicación, donde los usuarios pueden ver y gestionar sus comidas diarias.

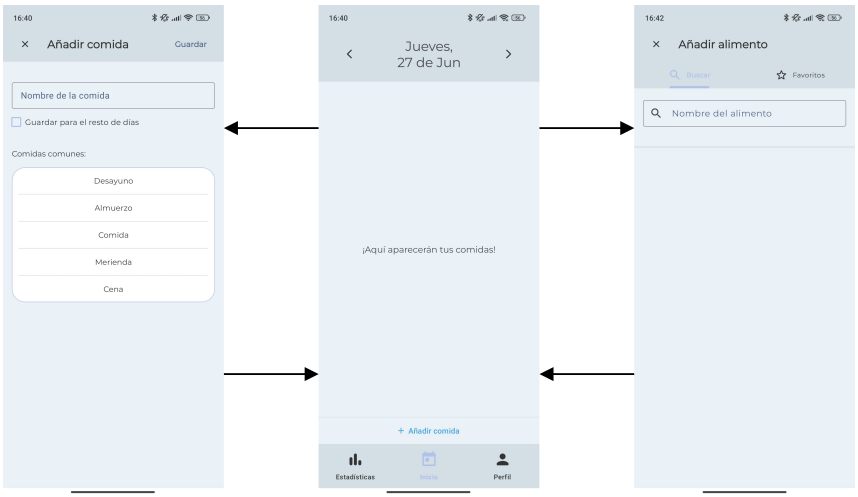


Figura 3.28: Flujo de navegación de la pantalla Home y los Dialogs

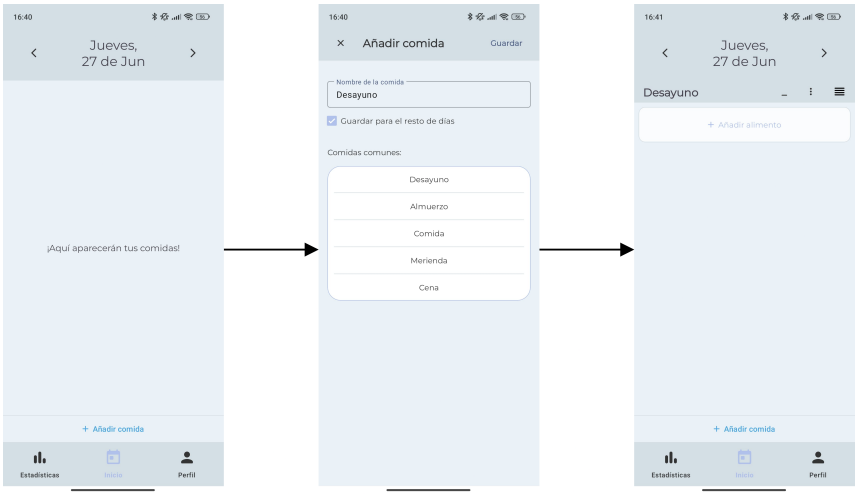


Figura 3.29: Ejemplo de añadir una comida

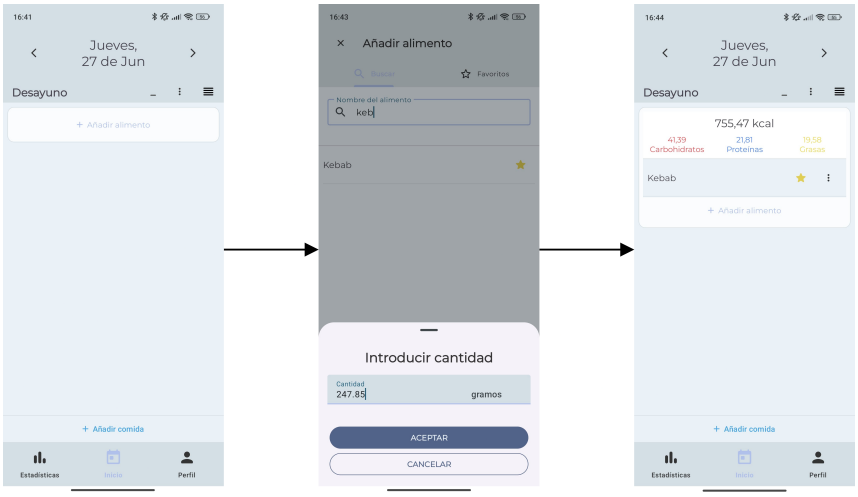


Figura 3.30: Ejemplo de añadir un alimento

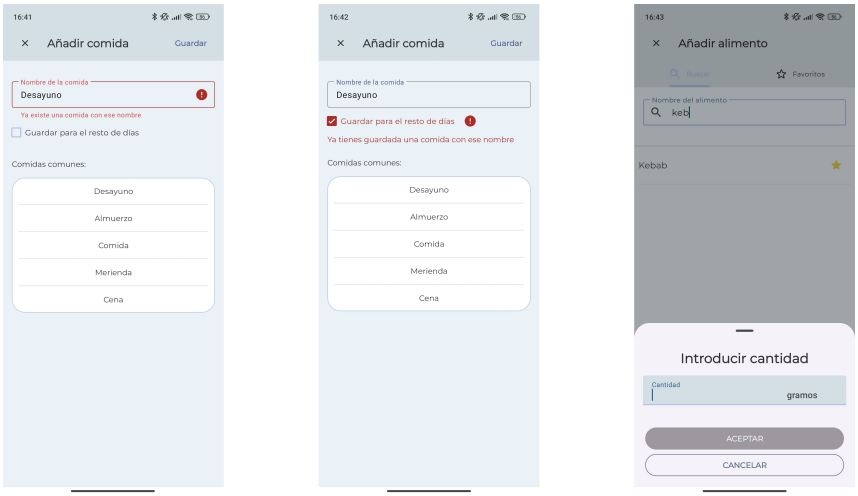


Figura 3.31: Errores de campos de entrada en dialogs

- **Inicialización:** Al iniciar esta pantalla, la aplicación realiza una petición al backend para recoger los datos guardados del día actual y los muestra en un RecyclerView.
- **Navegación entre días:** Los usuarios pueden navegar entre distintos días y refrescar los datos con los correspondientes al día seleccionado.
- **Añadir comida:** El usuario puede pulsar el botón .Añadir comida", que abre un diálogo de pantalla completa donde puede ingresar los datos de la nueva comida.
- **Editar o eliminar comida:** Pulsando el botón "más"(tres puntos verticales) en una comida, el usuario puede elegir editar o eliminar la comida:
  - **Editar comida:** Abre un bottom sheet donde se puede editar el nombre de la comida, teniendo dos botones para aceptar o cancelar la acción.
  - **Eliminar comida:** Muestra un diálogo de confirmación, y si el usuario confirma, la comida es eliminada.
- **Reordenar comidas:** El usuario puede mantener pulsado el ImageView de reordenar comidas y reorganizarlas según su preferencia, al soltar el item, se envía una petición PATCH al backend para editar los datos según el nuevo orden.
- **Estado de las comidas:** Cada comida tiene cuatro estados distintos que el usuario puede manipular: MINIMIZED, COLLAPSED, EMPTY y EXPANDED.
  - **MINIMIZED:** Es el estado que menos ocupa en la pantalla, el usuario puede establecer y revertir este estado pulsando el botón de minimizar.
  - **COLLAPSED:** Este estado muestra el resumen de todas calorías y macronutrientes que tienen los alimentos de la comida. El usuario puede establecer y revertir este estado pulsando el mismo resumen.

  - **EMPTY:** Este estado no es manipulable por el usuario, se establece cuando no existen entradas de alimentos en la comida.
  - **EXPANDED:** Este estado muestra el resumen de todas calorías y macronutrientes que tienen los alimentos de la comida, además de cada alimento añadido. El usuario puede establecer y revertir este estado pulsando el resumen.

- **Añadir alimento a una comida:** Dentro de cada comida, el usuario puede pulsar el botón .Añadir alimento", que abre un diálogo de pantalla completa para seleccionar el alimento.
- **Editar o eliminar alimento:** Pulsando el botón "más"(tres puntos verticales) en un alimento, el usuario puede elegir editar o eliminar el alimento:
  - **Editar alimento:** Abre un bottom sheet donde se puede editar el peso del alimento, teniendo dos botones para aceptar o cancelar la acción.
  - **Eliminar Comida:** Muestra un diálogo de confirmación, y si el usuario confirma, el alimento es eliminado.
- **Guardar favorito:** Al pulsar el botón "favoritoçon el ícono de una estrella, el usuario guarda o elimina ese alimento como favorito.
- **Dialog de comidas:** Este dialog permite al usuario agregar una nueva comida.
  - **Campos del diálogo:** Incluye un campo de texto para el nombre de la comida y un checkbox para elegir si se debe crear en todos los días.
  - **Validaciones:**
    - **Nombre de la comida:** Valida si la comida ya existe en ese día.
    - **Checkbox:** Valida si la comida se quiere guardar para todos los días y ya existe una comida guardada con el mismo nombre.
  - **Navegación en el dialog:** El usuario puede cancelar y salir del dialog o guardar la comida haciendo una petición POST al backend. Al completar la petición, se sale del dialog.
- **Dialog de comidas:** Este diálog permite al usuario seleccionar y agregar alimentos a una comida.
  - **Estructura del dialog:** Contiene un ViewPager2 con dos fragmentos:
    - **Buscar alimentos:** El primer fragmento permite buscar alimentos en la base de datos a través de un patrón ingresado por el usuario.
    - **Alimentos favoritos:** El segundo fragmento muestra los alimentos guardados como favoritos por el usuario en la base de datos local de la aplicación.
  - **Añadir alimento:** Al pulsar en un alimento, se abre un bottom sheet para establecer la cantidad consumida. Al guardar, se navega de vuelta

a la pantalla Home con el nuevo alimento añadido a la comida correspondiente.

- **Guardar favorito:** Al pulsar el botón "favoritoçon el ícono de una estrella, el usuario guarda o elimina ese alimento como favorito.

Pantalla de Estadísticas

La pantalla Statistics proporciona a los usuarios una visión detallada de su consumo nutricional a lo largo de la semana, comparando los datos reales con los valores recomendados según la fórmula de Mifflin-St. Jeor.

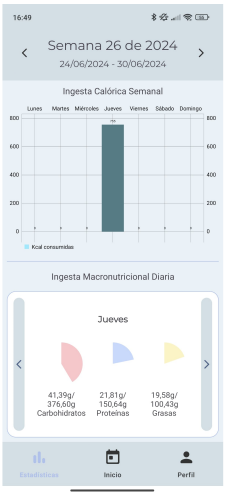


Figura 3.32: Pantalla de estadísticas

- **Inicialización:** Al iniciar esta pantalla, la aplicación realiza una petición al backend para recoger los datos nutricionales de la semana actual. Estos datos incluyen las kilocalorías consumidas diariamente y se presentan en un gráfico de barras utilizando la librería MPAndroidChart y los macronutrientes principales (carbohidratos, proteínas y grasas) que son pintados en un "PieChart".
- **Gráfico de barras:** El gráfico de barras muestra las kilocalorías consumidas día a día por cada día de la semana, junto con las kilocalorías recomendadas calculadas mediante la fórmula de Mifflin-St. Jeor basada en el perfil del usuario.
- **Gráficos circulares de macronutrientes:** A continuación del gráfico de barras, un ViewPager2 permite navegar entre los días de la semana, mostrando tres gráficos de pastel por cada día. Estos gráficos representan el consumo diario de carbohidratos, proteínas y grasas, comparados con los valores recomendados por la fórmula de Mifflin-St. Jeor.
- **Navegación entre semanas:** Los usuarios pueden navegar entre diferentes semanas y actualizar los datos de las gráficas con la información correspondiente a cada semana.

- **Interacción entre Gráficos:**
  - **Resaltar en el gráfico de barras:** Al pulsar en una barra del gráfico de barras, esta se resalta y el ViewPager2 navega automáticamente hasta el día correspondiente, sincronizando la vista entre ambos componentes.
  - **Navegación en ViewPager2:** Al navegar por los días en el ViewPager2, la barra correspondiente en el gráfico de barras se resalta, proporcionando una retroalimentación visual consistente y facilitando la navegación y comparación de datos.

Pantalla de Perfil

La pantalla de perfil permite a los usuarios ver y editar su información personal, así como gestionar otros aspectos de su cuenta. La estructura de esta pantalla se divide en varias secciones principales para facilitar la navegación y la edición de datos.

Figura 3.33: Pantalla de perfil con las subcategorías visibles

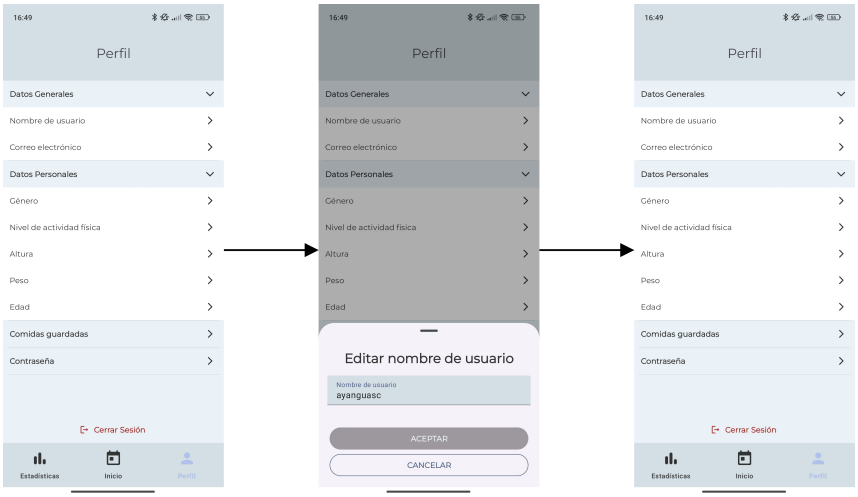


Figura 3.34: Flujo de editar una subcategoría

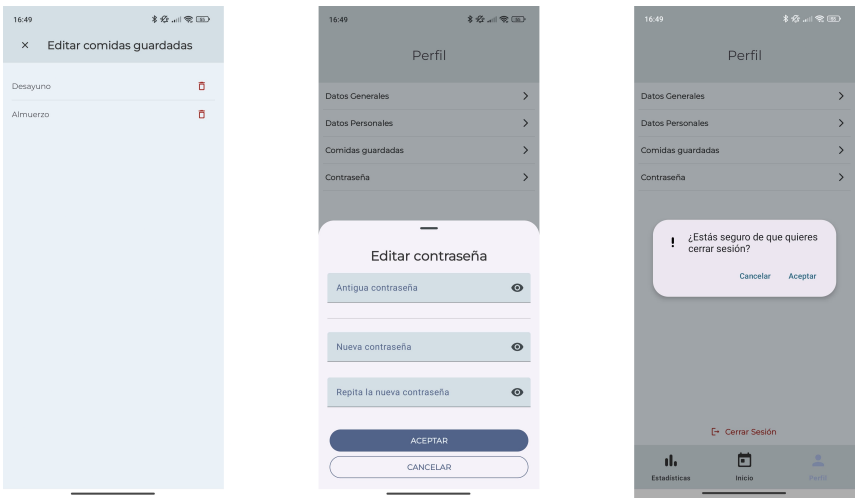


Figura 3.35: Editar comidas guardas, editar contraseña & cerrar sesión

- **Inicialización:** Al iniciar esta pantalla, la aplicación realiza una petición a la base de datos local SQLite para recuperar los datos del usuario. Estos datos se guardan en el momento del login o del signup.
- **Categorías de Perfil:** La pantalla se organiza en cuatro categorías principales: Datos generales, Datos personales, Comidas guardadas y Contraseña.
  - **Datos generales:** Incluye el nombre de usuario y el correo electrónico.
  - **Datos personales:** Contiene información como el género, nivel de actividad física, altura, peso y edad.
  - **Comidas guardadas:** Permite al usuario gestionar las comidas que ha guardado.
  - **Contraseña:** Facilita el cambio de la contraseña del usuario.

- **Interactividad por categorías:**
  - Al pulsar sobre **Datos generales** o **Datos personales**, las subcategorías se expanden o colapsan según su estado actual.
  - Al seleccionar una subcategoría, se abre un bottom sheet mostrando el valor actual de la misma. En este bottom sheet, el usuario puede editar la subcategoría con las mismas restricciones de entrada que se aplican durante el registro. Si los datos no son válidos, se muestran los mismos errores que en el registro.
  - **Modificación especial del nombre de usuario:** Cambiar el nombre de usuario requiere cerrar sesión, ya que el token de sesión queda inválido. Antes de proceder con el cambio, se muestra un mensaje de advertencia.
- **Gestión de las comidas guardadas:** Al pulsar el botón de **Comidas guardadas**, aparece un dialog de pantalla completa donde el usuario puede borrar comidas guardadas. Cada acción de borrado requiere una confirmación a través de otro dialog.
- **Cambio de contraseña:** Al pulsar el botón de **Contraseña**, se abre un bottom sheet con tres campos: contraseña actual, nueva contraseña y repetición de la nueva contraseña. La contraseña actual debe coincidir con la registrada en el sistema, y la nueva contraseña debe tener al menos 8 caracteres y coincidir con su repetición.
- **Cerrar sesión:** La pantalla incluye un botón de **Cerrar Sesión**. Al confirmar la acción, los tokens de sesión almacenados en la aplicación se borran y el usuario es redirigido a la pantalla de autenticación, evitando el retorno mediante el uso de flags.

3.3.2. Servicio REST

El Servicio REST es un componente crucial en la arquitectura del backend de la aplicación, ya que define cómo los datos nutricionales y las interacciones del usuario son gestionados y accesibles desde el frontend, que en este caso es una aplicación móvil desarrollada en Kotlin para Android. Este servicio permite que la aplicación móvil se comunique con el servidor de manera eficiente y segura, facilitando la creación, lectura, actualización y eliminación (CRUD) de datos a través de la red.

Manejo de Peticiones

Los routers en FastAPI permiten organizar los endpoints en módulos separados, facilitando la gestión y escalabilidad del código. Cada router está dedicado a una entidad o funcionalidad específica y se define en un archivo separado.

```python
router: APIRouter = APIRouter(prefix='/foods')


@router.get('', status_code=status.HTTP_200_OK)
async def search_foods(
    request: Request,
    pattern: str = Query(None, min_length=3),
    skip: int = Query(0),
    limit: int = Query(10),
    db: AsyncIOMotorDatabase = Depends(get_db)
)
```

Figura 3.36: Router Foods

- **`Router de Alimentos (foods.py):`** Define los endpoints relacionados con los alimentos, incluyendo únicamente la búsqueda de alimentos.
  - **`@GET /foods`** Recupera una lista de todos los alimentos disponibles en la base de datos que coinciden con el patrón.
    - **Parámetros:**
      - **`pattern`:** Un término de búsqueda para filtrar los alimentos (mínimo 3 caracteres).
      - **`skip` (opcional):** Número de resultados a omitir (por defecto 0).
      - **`limit` (opcional):** Número de resultados por página (por defecto 10).
    - **Respuesta:**
      - **200 OK:** Lista paginada de alimentos.
      - **422 Unprocessable Entity:** Error de validación.

```python
router: APIRouter = APIRouter(prefix='/login')


@router.post('')
async def login_for_access_token(
    cache: Redis = Depends(get_cache),
    db: AsyncIOMotorDatabase = Depends(get_db),
    form_data: OAuth2PasswordRequestForm = Depends()
)
```

Figura 3.37: Router Login

- **`Router de Login (login.py):`** Define el endpoint relacionado con el inicio de sesión del usuario.
  - **`@POST /login`** Permite iniciar sesión y obtener un token de acceso.
    - **Parámetros:**
      - **`grant_type`**: Tipo de grant (debe ser "password").
      - **`username`**: Nombre de usuario.
      - **`password`**: Contraseña del usuario.
      - **`scope` (opcional)**: Alcance de la solicitud.
      - **`client_id` (opcional)**: ID del cliente.
      - **`client_secret` (opcional)**: Secreto del cliente.
    - **Respuesta:**
      - **200 OK:** Token de acceso y/o de refresco.
      - **422 Unprocessable Entity:** Error de validación.

```python
router: APIRouter = APIRouter(prefix='/signup')


@router.post('', response_model=UserOut)
async def signup(
    user_in: UserIn,
    db: AsyncIOMotorDatabase = Depends(get_db)
)
```

Figura 3.38: Router Signup

- **`Router de Signup (signup.py):`** Define el endpoint relacionado con el nuevo registro de usuario.
  - **`@POST /signup`** Permite registrar un nuevo usuario.
    - **Parámetros:**
      - **`username`**: Nombre de usuario.
      - **`email`**: Correo electrónico.
      - **`profile`**: Perfil del usuario.
      - **`password`**: Contraseña del usuario.
    - **Respuesta:**
      - **200 OK:** Datos del usuario registrado.
      - **422 Unprocessable Entity:** Error de validación.

```python
router: APIRouter = APIRouter(prefix='/refresh')


@router.post('')
async def refresh_token(
    cache: Redis = Depends(get_cache),
    db: AsyncIOMotorDatabase = Depends(get_db),
    refresh_token: str = Body( ... )
)
```

Figura 3.39: Refresh Router

- **`Router de Refresh (refresh.py):`** Define el endpoint relacionado con la obtención de nuevos tokens en caso de que el usuario tenga refresh token.
  - **`@POST /refresh`** Permite refrescar los tokens de acceso y de refresco.
    - **Parámetros:**
      - **`refresh_token`**: Token de refresco.
    - **Respuesta:**
      - **200 OK:** Nuevos token de acceso y refresco.
      - **422 Unprocessable Entity:** Error de validación.

```python
router: APIRouter = APIRouter(prefix='/meals')


@router.post('', response_model=Meal)
async def create_meal(
    meal_in: MealIn,
    user: User = Depends(get_current_user),
    db: AsyncIOMotorDatabase = Depends(get_db)
)

@router.patch('/{meal_id}/rename', status_code=status.HTTP_200_OK)
async def rename_meal(
    meal_id: str,
    new_name: str = Body( ... ),
    user: User = Depends(get_current_user),
    db: AsyncIOMotorDatabase = Depends(get_db)
)

@router.delete('/{meal_id}', status_code=status.HTTP_204_NO_CONTENT)
async def delete_meal(
    meal_id: str,
    user: User = Depends(get_current_user),
    db: AsyncIOMotorDatabase = Depends(get_db)
)
```

Figura 3.40: Router Meals I

```python
router: APIRouter = APIRouter(prefix='/meals')


@router.get('/week')
async def get_weekly_meals(
    start_week_date: str = Query(None),
    end_week_date: str = Query(None),
    user: User = Depends(get_current_user),
    db: AsyncIOMotorDatabase = Depends(get_db)
)

@router.get('', response_model=List[Meal])
async def get_meals(
    target_date: str = Query(None),
    user: User = Depends(get_current_user),
    db: AsyncIOMotorDatabase = Depends(get_db)
)

@router.patch('/reorder')
async def reorder_meals(
    meals: List[MealOrder],
    user: User = Depends(get_current_user),
    db: AsyncIOMotorDatabase = Depends(get_db)
)
```

Figura 3.41: Router Meals II

```python
router: APIRouter = APIRouter(prefix='/meals')


@router.post('/{meal_id}/foods', response_model=FoodItem)
async def add_food(
    meal_id: str,
    food_in: FoodIn,
    user: User = Depends(get_current_user),
    db: AsyncIOMotorDatabase = Depends(get_db)
)


@router.patch('/{meal_id}/foods/{food_id}/weight')
async def patch_new_weight(
    meal_id: str,
    food_id: str,
    weight: float = Body( ... ),
    user: User = Depends(get_current_user),
    db: AsyncIOMotorDatabase = Depends(get_db)
)

@router.delete('/{meal_id}/foods/{food_id}', status_code=status.HTTP_204_NO_CONTENT)
async def delete_food(
    meal_id: str,
    food_id: str,
    user: User = Depends(get_current_user),
    db: AsyncIOMotorDatabase = Depends(get_db)
)
```

Figura 3.42: Router Meals III

- **`Router de Comidas (meals.py):`** Gestiona las operaciones relacionadas con las comidas del usuario, permitiendo crear nuevas comidas, editar comidas existentes y eliminar comidas.
  - **`@POST /meals`** Permite crear una nueva comida.
    - **Parámetros:**
      - **`name`**: Nombre de la comida.
      - **`datetime`**: Fecha de la comida.
      - **`save_meal`**: Booleano indicando si se debe guardar la comida.
    - **Respuesta:**
      - **200 OK:** Comida creada.
      - **422 Unprocessable Entity:** Error de validación.
  - **`@GET /meals`** Recupera una lista de comidas según una fecha.
    - **Parámetros:**
      - **`target_date`:** Fecha objetivo para filtrar las comidas.
    - **Respuesta:**
      - **200 OK:** Lista de comidas.
      - **422 Unprocessable Entity:** Error de validación.
  - **`@PATCH /meals/{meal_id}/rename`** Permite renombrar una comida.
    - **Parámetros:**

        - ◇ **`meal_id`**: ID de la comida.
        - ◇ **`new_name`**: Nuevo nombre de la comida.
    - **Respuesta:**
        - ◇ **200 OK:** Comida renombrada.
        - ◇ **422 Unprocessable Entity:** Error de validación.
- **`@DELETE /meals/{meal_id}`** Permite eliminar una comida.
    - **Parámetros:**
        - ◇ **`meal_id`**: ID de la comida.
    - **Respuesta:**
        - ◇ **204 No Content:** Comida eliminada.
        - ◇ **422 Unprocessable Entity:** Error de validación.
- **`@GET /meals/week`** Recupera las comidas de la semana.
    - **Parámetros:**
        - ◇ **`start_week_date`:** Fecha de inicio de la semana.
        - ◇ **`end_week_date`:** Fecha de fin de la semana.
    - **Respuesta:**
        - ◇ **200 OK:** Lista de comidas semanales.
        - ◇ **422 Unprocessable Entity:** Error de validación.
- **`@PATCH /meals/reorder`** Permite reordenar las comidas.
    - **Parámetros:**
        - ◇ **`meals`**: Array de objetos con el nuevo orden de las comidas.
    - **Respuesta:**
        - ◇ **200 OK:** Comidas reordenadas.
        - ◇ **422 Unprocessable Entity:** Error de validación.
- **`@POST /meals/{meal_id}/foods`** Permite añadir un alimento a una comida.
    - **Parámetros:**
        - ◇ **`meal_id`**: ID de la comida.
        - ◇ **`food`**: Objeto con el detalle del alimento, siendo el ID y el peso.
    - **Respuesta:**
        - ◇ **200 OK:** Alimento añadido.
        - ◇ **422 Unprocessable Entity:** Error de validación.

- **`@PATCH /meals/{meal_id}/foods/{food_id}/weight`** Permite actualizar el peso de un alimento en una comida.
  - **Parámetros:**
    - **`meal_id`**: ID de la comida.
    - **`food_id`**: ID del alimento.
    - **`weight`**: Nuevo peso del alimento.
  - **Respuesta:**
    - **200 OK:** Peso actualizado.
    - **422 Unprocessable Entity:** Error de validación.
- **`@DELETE /meals/{meal_id}/foods/{food_id}`** Permite eliminar un alimento de una comida.
  - **Parámetros:**
    - **`meal_id`**: ID de la comida.
    - **`food_id`**: ID del alimento.
  - **Respuesta:**
    - **204 No Content:** Alimento eliminado.
    - **422 Unprocessable Entity:** Error de validación.

```python
router: APIRouter = APIRouter(prefix='/users')


@router.get('/check_email')
async def check_user_by_email(
    email: str = Query( ... ),
    db: AsyncIOMotorDatabase = Depends(get_db)
)

@router.get('/check_username')
async def check_user_by_username(
    username: str = Query( ... ),
    db: AsyncIOMotorDatabase = Depends(get_db)
)

@router.get('/me', response_model=UserOut, status_code=status.HTTP_200_OK)
async def get_user_data(user: User = Depends(get_current_user))
```

Figura 3.43: Router Users I

```python
router: APIRouter = APIRouter(prefix='/users')


@router.delete('/me/{meal_name}', status_code=status.HTTP_204_NO_CONTENT)
async def remove_meal_in_template(
    meal_name: str,
    user: User = Depends(get_current_user),
    db: AsyncIOMotorDatabase = Depends(get_db)
)

@router.patch('/me/update', status_code=status.HTTP_200_OK)
async def update_user(
    update_fields: UserUpdateRequest = Body( ... ),
    user: User = Depends(get_current_user),
    db: AsyncIOMotorDatabase = Depends(get_db)
)

@router.patch('/me/update_password', status_code=status.HTTP_200_OK)
async def update_password(
    password_request: UpdatePassword = Body( ... ),
    user: User = Depends(get_current_user),
    db: AsyncIOMotorDatabase = Depends(get_db)
)
```

Figura 3.44: Router Users II

- **`Router de Usuarios (users.py):`** Maneja la gestión de usuarios, incluyendo el registro de nuevos usuarios y la actualización de información del perfil.
  - **`@GET /users/check_email`** Verifica si un correo electrónico ya está registrado.
    - **Parámetros:**
      - **`email`**: Correo electrónico a verificar.
    - **Respuesta:**
      - **200 OK:** Correo verificado.
      - **422 Unprocessable Entity:** Error de validación.
  - **`@GET /users/check_username`** Verifica si un nombre de usuario ya está registrado.
    - **Parámetros:**

        - ◇ **`username`**: Nombre de usuario a verificar.
    - **Respuesta:**
        - ◇ **200 OK:** Nombre de usuario verificado.
        - ◇ **422 Unprocessable Entity:** Error de validación.
- **`@GET /users/me`** Recupera los datos del usuario autenticado.
    - **Respuesta:**
        - ◇ **200 OK:** Datos del usuario.
- **`@DELETE /users/me/{meal_name}`** Elimina una comida de la plantilla del usuario.
    - **Parámetros:**
        - ◇ **`meal_name`**: Nombre de la comida a eliminar.
    - **Respuesta:**
        - ◇ **204 No Content:** Comida eliminada.
        - ◇ **422 Unprocessable Entity:** Error de validación.
- **`@PATCH /users/me/update`** Permite actualizar los datos del usuario.
    - **Parámetros:**
        - ◇ **`username`**: Nuevo nombre de usuario (opcional).
        - ◇ **`email`**: Nuevo correo electrónico (opcional).
        - ◇ **`gender`**: Nuevo género (opcional).
        - ◇ **`height`**: Nueva altura (opcional).
        - ◇ **`weight`**: Nuevo peso (opcional).
        - ◇ **`age`**: Nueva edad (opcional).
        - ◇ **`physical_activity_level`**: Nuevo nivel de actividad física (opcional).
    - **Respuesta:**
        - ◇ **200 OK:** Datos del usuario actualizados.
        - ◇ **422 Unprocessable Entity:** Error de validación.
- **`@PATCH /users/me/update_password`** Permite actualizar la contraseña del usuario.
    - **Parámetros:**
        - ◇ **`old_password`**: Contraseña actual.
        - ◇ **`new_password`**: Nueva contraseña.
    - **Respuesta:**

- ◇ **200 OK:** Contraseña actualizada.
- ◇ **422 Unprocessable Entity:** Error de validación.

3.3.3. Pruebas y Validación

El proceso de pruebas y validación es el proceso para asegurar la calidad y funcionalidad de la aplicación. A lo largo del desarrollo del proyecto, se realizaron pruebas exhaustivas tanto en el backend como en el frontend para garantizar que cada componente funcionara según lo esperado y cumpliera con los requisitos establecidos.

Pruebas del Frontend

La aplicación se probó continuamente a medida que se desarrollaba, utilizando un enfoque iterativo para asegurar que cada componente y pantalla funcionara correctamente.

- **Pruebas de componentes:** Cada componente del frontend se probó individualmente para asegurar su funcionalidad antes de integrarse con otros componentes. Esto incluyó la validación de la lógica de los ViewModels y la correcta actualización de la interfaz de usuario en respuesta a los cambios en los datos.
  - **Interacción del usuario:** Se simularon interacciones del usuario, como clics, entradas de texto y navegación entre pantallas, para verificar que la aplicación respondiera correctamente y proporcionara una experiencia de usuario fluida.
- **Pruebas de pantallas completas:** Las pruebas de pantallas completas se realizaron para verificar la integración de los componentes y asegurar que la navegación entre pantallas funcionara sin problemas. Se prestó especial atención a la validación de los datos de entrada y la correcta visualización de los datos recuperados del backend.
  - **Validación de navegación:** Se verificó que la navegación entre pantallas funcionara correctamente, incluyendo la transición entre pantallas de autenticación, home, estadísticas y perfil.
- **Pruebas de interfaz de usuario:** Se aseguraron de que todos los elementos de la UI estuvieran alineados y funcionaran como se esperaba en diferentes dispositivos y tamaños de pantalla.
  - **Consistencia visual:** Se verificó que la apariencia de la aplicación fuera consistente con el diseño previsto, y que los elementos de la interfaz estuvieran correctamente alineados y fueran fácilmente accesibles.

Pruebas del Backend

El backend de la aplicación fue probado utilizando Thunder Client, una herramienta de extensión para Visual Studio Code que facilita la realización de prue-

bas HTTP. El enfoque de las pruebas incluyó la validación de las operaciones CRUD (Crear, Leer, Actualizar y Eliminar) para cada uno de los endpoints del Servicio REST.

- **Datos de entrada correctos:** Se enviaron solicitudes con datos válidos a cada uno de los endpoints para verificar que el backend procesara las solicitudes correctamente y devolviera las respuestas esperadas. Estas pruebas aseguraron que las operaciones de la API funcionaran como se diseñó.
  - **Validación de las respuestas:** Se compararon las respuestas del servidor con los resultados esperados para asegurar que los datos se procesaran y devolvieran correctamente.
- **Datos de entrada erróneos:** Se probaron escenarios con datos de entrada incorrectos o faltantes para verificar que el backend manejara adecuadamente los errores y devolviera mensajes de error útiles y comprensibles.
  - **Manejo de errores:** Se verificó que el backend respondiera con los códigos de estado HTTP adecuados (por ejemplo, 400 Bad Request, 404 Not Found) y mensajes de error descriptivos que ayudaran a identificar y corregir el problema.

Capítulo 4

Análisis de impacto

En este capítulo se realiza un análisis del impacto potencial de los resultados obtenidos durante la realización del proyecto “Creación de una Aplicación móvil y los Servicios REST Asociados para la Recogida de Datos Nutricionales“ en diversos contextos. Se destacan los beneficios esperados, los posibles efectos adversos y las decisiones tomadas basadas en la consideración del impacto. Además, se analiza el potencial impacto respecto a los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030.

Impacto Personal Los usuarios pueden mantener un registro preciso de su ingesta nutricional, lo que les ayuda a seguir dietas más saludables y equilibradas. La aplicación proporciona información sobre las necesidades nutricionales basadas en la fórmula de Mifflin-St Jeor, aumentando el conocimiento del usuario sobre su dieta y necesidades calóricas. Por otro lado, los usuarios podrían volverse dependientes de la aplicación para la gestión de su dieta, perdiendo la capacidad de hacerlo manualmente. Se tomaron decisiones de diseño centradas en la usabilidad para asegurar que la aplicación sea fácil de usar por una amplia gama de usuarios.

Impacto Social La aplicación puede influir positivamente y promover los hábitos alimenticios de la sociedad, promoviendo una dieta más equilibrada y saludable. Facilita el acceso a información nutricional a un público amplio, incluyendo a personas con diferentes niveles de conocimiento sobre nutrición. Como efecto adverso, existe el riesgo de que los datos personales y nutricionales de los usuarios puedan ser comprometidos si no se manejan adecuadamente. Para mitigar este riesgo, se implementaron medidas de seguridad robustas para proteger la información privada de los usuarios.

Impacto Medioambental El desarrollo y la operación de la aplicación implican el uso de recursos tecnológicos que tienen una huella de carbono.

Impacto en los Objetivos de Desarrollo Sostenible (ODS) El proyecto contribuye a varios ODS de la Agenda 2030:

- **ODS 3: Salud y Bienestar:** Promueve una vida saludable y el bienestar para todos, en todas las edades.
- **ODS 12: Producción y Consumo Responsables:** Fomenta patrones de consumo sostenibles, especialmente en lo que respecta a la alimentación.

Capítulo 5

Conclusiones y trabajo futuro

El proyecto ha cumplido con éxito sus objetivos principales, proporcionando una herramienta efectiva para el seguimiento y gestión de la nutrición personal. El impacto potencial de esta aplicación es significativo. Puede contribuir a que los usuarios puedan mantener un registro detallado de sus ingestas nutricionales, visualizar estadísticas comparativas y adaptar sus datos personales para mantener la precisión de las recomendaciones.

En lo personal, la realización de este proyecto ha sido una experiencia sumamente enriquecedora. A lo largo del desarrollo del proyecto, he adquirido un profundo conocimiento en el uso de tecnologías modernas como FastAPI y he potenciado e incrementado las que ya tenía de Kotlin, así como en el diseño e implementación de arquitecturas de software. Este proyecto no solo ha permitido crear una herramienta práctica y funcional que puede ayudar a las personas a gestionar mejor su nutrición, sino que también ha reforzado mis habilidades en desarrollo de software y gestión de proyectos.

Líneas futuras Para seguir mejorando la aplicación y ofrecer una experiencia aún más completa, se han identificado varias áreas de trabajo futuro:

- **Modo oscuro:** Implementar un modo oscuro para mejorar la experiencia del usuario en condiciones de poca luz y reducir la fatiga visual.
- **Mejor Host Cloud:** Migrar el backend a un host cloud más confiable y robusto que no se inhabilite si deja de recibir peticiones, garantizando una mayor disponibilidad.
- **Adaptación a modo hotizontal:** Adaptar la aplicación para que se pueda usar con el móvil en modo horizontal, mejorando la flexibilidad y usabilidad en diferentes orientaciones de pantalla.
- **Añadir raciones preestablecidas de alimentos:** Incluir opciones de raciones preestablecidas para alimentos (por ejemplo, una cucharadita de sal o piezas pequñas/medianas/grandes de frutas), facilitando la entrada de datos y mejorando la experiencia de usuario.

- **Mejoras en la interfaz de usuario:** Continuar mejorando la interfaz de usuario basada en feedback de los usuarios, incluyendo optimizaciones de usabilidad y accesibilidad.
- **Funcionalidades de comunidad:** Implementar funcionalidades de comunidad donde los usuarios puedan compartir recetas, consejos nutricionales y motivarse mutuamente, creando un entorno más interactivo y de apoyo.

Bibliografía

[1] U. N. S. S. C. on Nutrition, «Nutrition in a Digital World», 2020. dirección: www.unscn.org.

[2] Google. «Enfoque de prioridad de Kotlin en Android». (2023), dirección: https://developer.android.com/kotlin/first?hl=es-419 (visitado 11-10-2023).

[3] JetBrains. «Android Overview - Kotlin Programming Language». (2023), dirección: https://kotlinlang.org/docs/android-overview.html (visitado 11-10-2023).

[4] Google. «Migrar a Kotlin DSL». (2023), dirección: https://developer.android.com/build/migrate-to-kotlin-dsl?hl=es-419 (visitado 11-10-2023).

[5] Google. «Guía de arquitectura de aplicaciones Android: MVVM». (2023), dirección: https://developer.android.com/jetpack/guide?hl=es-419 (visitado 11-10-2023).

[6] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2017, ISBN: 978-0134494166.

[7] Square. «Retrofit». (2023), dirección: https://square.github.io/retrofit/ (visitado 11-10-2023).

[8] Google. «Comenzar con Navigation Component». (2023), dirección: https://developer.android.com/guide/navigation/get-started?hl=es-419 (visitado 11-10-2023).

[9] Google. «Navigation Component y el manejo del Back Stack». (2023), dirección: https://developer.android.com/guide/navigation/backstack?hl=es-419 (visitado 11-10-2023).

[10] Dagger. «Dagger-Hilt». (2023), dirección: https://dagger.dev/hilt/ (visitado 11-10-2023).

[11] Google. «Room: Persistencia de datos». (2023), dirección: https://developer.android.com/training/data-storage/room?hl=es-419 (visitado 11-10-2023).

[12] A. Developers. «Kotlin Coroutines». (2023), dirección: https://developer.android.com/kotlin/coroutines?hl=es-419 (visitado 11-10-2023).

[13] A. Developers. «Coroutines avanzadas en Kotlin». (2023), dirección: https://developer.android.com/kotlin/coroutines/coroutines-adv?hl=es-419 (visitado 11-10-2023).

[14] A. Developers. «Kotlin Flow». (2023), dirección: https://developer.android.com/kotlin/flow?hl=es-419 (visitado 11-10-2023).

[15] S. Ramírez. «FastAPI Documentation». (2023), dirección: https://fastapi.tiangolo.com/ (visitado 11-10-2023).

[16] MongoDB. «¿Qué es MongoDB?» (2023), dirección: https://www.mongodb.com/es/what-is-mongodb (visitado 11-10-2023).

[17] Auth0. «Introducción a IAM: ¿Qué es OAuth 2?» (2023), dirección: https://auth0.com/es/intro-to-iam/what-is-oauth-2 (visitado 11-10-2023).

[18] OAuthLib. «Bearer Tokens». (2023), dirección: https://oauthlib.readthedocs.io/en/latest/oauth2/tokens/bearer.html (visitado 11-10-2023).

[19] Auth0. «Autenticación y flujo de autorización: Autenticar con JWT de clave privada». (2023), dirección: https://auth0.com/docs/get-started/authentication-and-authorization-flow/authenticate-with-private-key-jwt (visitado 11-10-2023).

[20] Google. «Aspectos básicos de las pruebas». (2023), dirección: https://developer.android.com/training/testing/fundamentals?hl=es-419 (visitado 11-10-2023).

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Sun Jun 30 20:48:54 CEST 2024
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)