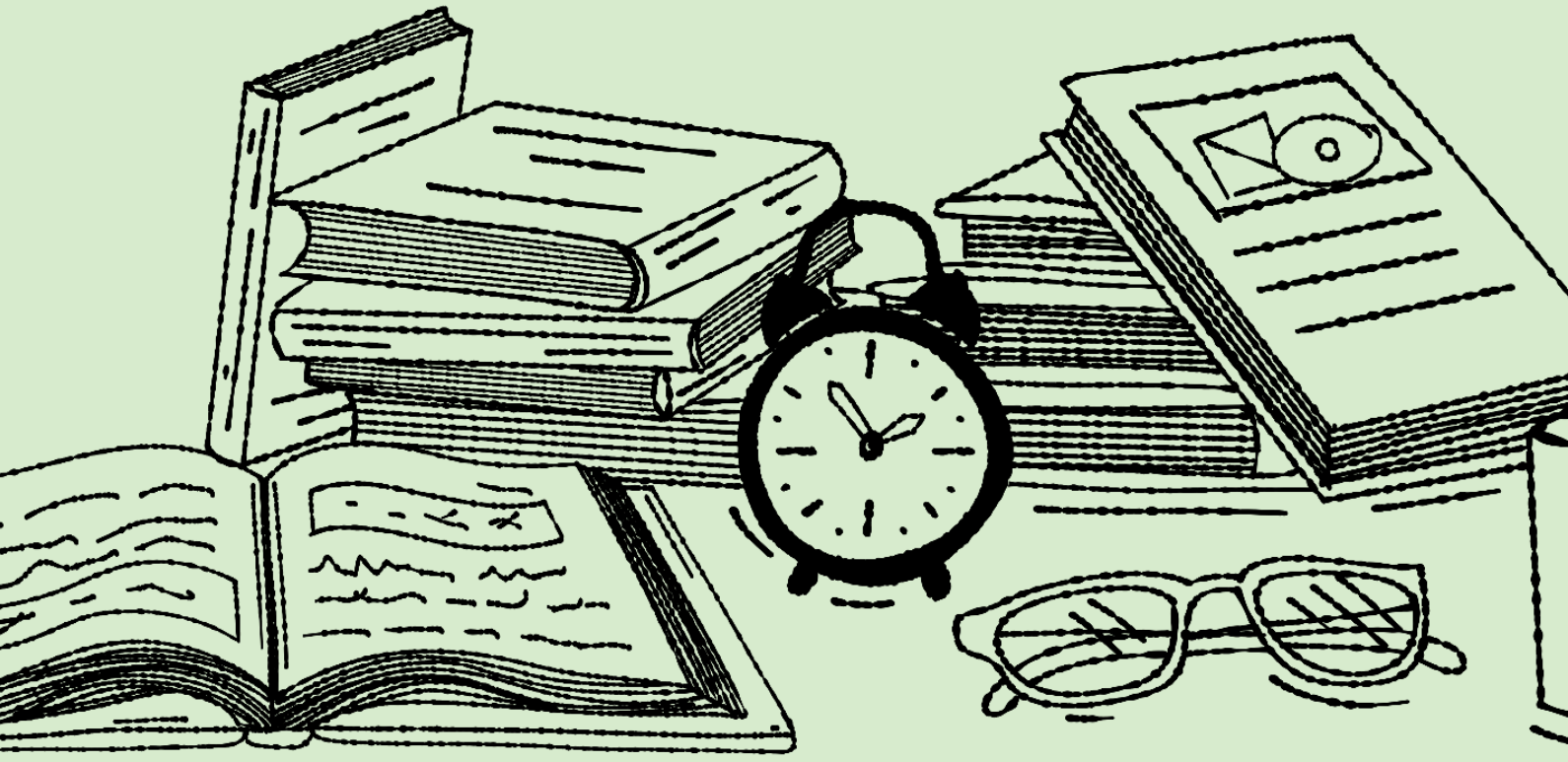
Proyecto Android - PMDM

Laura Blanco Gutiérrez

Óscar García Delgado

Samuel Leyton Rojas



Índice:

1. Introducción.

FruiTask es una aplicación que sirve como herramienta de apoyo al estudio para estudiantes que necesitan organizar mejor sus tareas, exámenes y proyectos, pero que al mismo tiempo quieren algo más motivador que una simple lista de pendientes.

La idea central es combinar gestión académica y gamificación mediante una mascota virtual con forma de fruta (kiwi, manzana o sandía). Esta mascota acompaña al usuario en su día a día y va subiendo o bajando de nivel según se vayan cumpliendo o no las actividades registradas.

La aplicación se estructura en varios apartados clave:

- Una pantalla principal donde se muestran todas las actividades.
- Un calendario que resalta en distintos colores las actividades según su tipo y fecha.
- Un cronómetro que el usuario puede utilizar para controlar sus sesiones de estudio, como se hace en el método Pomodoro.
- Una pantalla inicial de selección de mascota, que aparece únicamente la primera vez que se abre la aplicación.

Con este enfoque, la app no solo ayuda a recordar qué hay que hacer y cuándo, sino que también introduce un componente de juego: el estudiante siente que cuida de su mascota cumpliendo sus obligaciones académicas.

El objetivo principal del proyecto es desarrollar una aplicación Android funcional que permita:

- Registrar y gestionar actividades académicas, clasificadas en: tareas, exámenes o proyectos.
- Visualizar estas actividades de forma intuitiva, tanto en una lista principal como en un calendario.
- Motivar al usuario mediante una mascota virtual que sube de nivel cuando el usuario marca las actividades como realizadas o pierde nivel cuando las actividades no llegan a completarse.
- Ofrecer herramientas complementarias como un cronómetro de estudio, que invite a trabajar por bloques de tiempo.

Se pretende que la aplicación sea visualmente clara, ligera e intuitiva, es decir, fácil de entender sin necesidad de manual.

2. Descripción del proyecto.

La aplicación está pensada como un asistente de estudio gamificado.

El flujo típico de uso es el siguiente:

1. El usuario abre la app por primera vez y se le pide que elija una mascota: kiwi, manzana o sandía. Esta mascota será la que vea en la pantalla principal, representando su progreso.
2. En la pantalla principal, el usuario puede:
    - Ver todas sus actividades.
    - Crear nuevas actividades indicando su título, fecha, y tipo de actividad.
    - Marcar una actividad como realizada.
3. Desde la barra de navegación, el usuario puede acceder al:
    - Calendario: donde se ven los días con actividades, resaltados en distintos colores según el tipo.
    - Cronómetro: útil para sesiones tipo “pomodoro” o bloques de estudio.
4. La mascota evoluciona según el comportamiento del usuario:
    - Si el usuario completa las actividades a tiempo, la mascota gana niveles.
    - Si las actividades cumplen la fecha y no han sido marcadas como realizadas, la mascota pierde niveles.

De esta forma, la app no solo se limita a listar tareas, sino que insiste en la idea de responsabilidad y constancia, representada por el estado de la mascota.

3. Diagrama de pantallas.

3.1. Flujo de navegación.

La primera ejecución se realiza en la pantalla para seleccionar la mascota, y, tras elegir la mascota, te redirige a la pantalla principal. En esta pantalla se tiene acceso a la mascota, la lista de actividades, y también, a crear una nueva actividad.

Todas las pantallas tienen una barra de navegación para poder desplazarse a los diferentes componentes: pantalla principal, calendario o cronómetro.

Desde el calendario, al pulsar en un día resaltado, muestra la actividad que es.

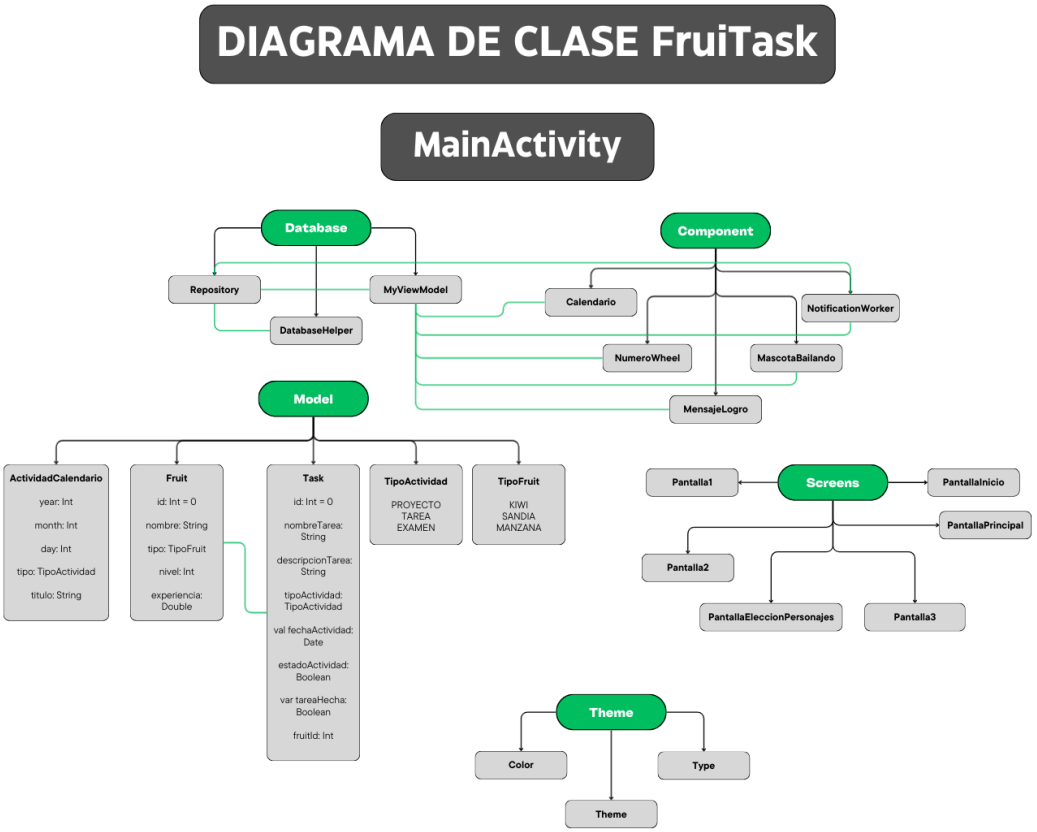
Por otro lado, en cronómetro se puede iniciar el tiempo que el usuario, y una vez iniciado, pararlo o cancelarlo.

3.2. Pantallas principales.

Para diseñar las pantallas principales, se crearon los bocetos utilizando Canva con el fin de definir la distribución visual, la paleta de colores y los elementos principales de la interfaz antes de implementarlo en Android Studio. Las capturas que se muestran a continuación muestran la estructura propuesta, y sirven de guía para facilitar el desarrollo en la plataforma.

4. Diagrama de clases.

La organización sigue la arquitectura recomendada en Android (MVVM: Model, ViewModel, Repository), junto con componentes de interfaz y lógicas auxiliares.



4.1. Estructura de clases.

La siguiente es una descripción detallada de las clases que componen el proyecto, organizadas por carpetas.

Antes que nada, el *MainActivity*: clase que actúa como la vista principal. También se gestionan notificaciones y se pide el permiso para ello (se puede pedir al seleccionar cierta pantalla por ejemplo, pero en este caso es aquí, para que los pida nada más iniciar la APP).

- Carpeta database:

DatabaseHelper: Creación, modificación y actualización de las tablas.

MyViewModel: Las funciones necesarias (insertar frutas, completar tareas, etc.) Utilizando el Repository.

Repository: Donde se programa las funciones que serán utilizadas por MyViewModel

- Carpeta model:

TipoActividad: Tipo de actividades que hay.

TipoFruit: Tipos de fruta que hay.

Fruit: almacena id, nivel, nombre, tipo de fruta y experiencia.

Task: almacena id, nombre de la tarea, descripción, tipo, fecha, estado y el id de la fruta.

ActividadCalendario: almacena año, mes, día, tipo de actividad y título de la actividad.

- Carpeta component:

NúmeroWheel: clase encargada de generar la rueda de números.

Calendario: clase encargada de gestionar el calendario.

MascotaBailando: Cuando sube de nivel la mascota inserta el video y el texto.

NotificationWorker: Se encarga de crear las notificaciones.

MensajeLogro: Establecer el mensaje personalizado para cada fruta.

- Carpeta screens:

Pantalla1: pantalla de la app con la mascota y tareas.

Pantalla2: pantalla con el calendario.

Pantalla3: pantalla con el temporizador.

PantallaEleccionPersonajes: pantalla con el calendario.

PantallaInicio: gestiona cual pantalla mostrar al inicio, si es la primera vez en la app, te pone selección de personaje, si ya tienes uno, te redirige a la pantalla principal.

PantallaPrincipal: pantalla que añade el navegador abajo en todas las pantallas y va cambiando entre las distintas pantallas.

- Carpeta theme:

Color: para definir los colores que queramos usar en la app con un nombre.

Theme: cual es el "tema" de la app, los colores que queremos que siempre se usen, el tipo de letra, etc.

Type: para definir distintas tipografías, como queremos que sea la letra.

4.2. Relaciones entre componentes.

Relaciones del modelo y la base de datos:

- Task y Fruit: muchas tareas pertenecen a una fruta. Tipo: 1 a N, por lo que se implementa fruitId en Task.
- Repository y DatabaseHelper. El Repository utiliza internamente la base de datos.
- MyViewModel y Repository. Todas las llamadas a datos pasan por el Repository.

Relaciones entre la lógica y los componentes visuales

- MyViewModel y Calendario. El ViewModel proporciona la lista de tareas al calendario, y este último actualiza su vista según los datos del ViewModel.
- MyViewModel y NumeroWheel. El ViewModel puede observar valores del cronómetro, pero la rueda es un componente independiente.
- MyViewModel y MascotaBailando. Cuando el ViewModel detecta que la fruta sube de nivel, activa la animación correspondiente.
- MyViewModel y MensajeLogro. Se solicita un mensaje motivacional cuando ocurre un evento (tarea completada, subida de nivel).

Relaciones con sistema de notificaciones

- NotificationWorker y Repository. Consulta las tareas próximas a vencer.
- MyViewModel y NotificationWorker. Puede programar notificaciones al registrar una actividad.

5. Persistencia empleada.

5.1. Tecnologías utilizadas.

Para la persistencia de datos en FruiTask se ha utilizado una base de datos local mediante SQLite, la cual permite almacenar información de manera estructurada dentro del propio dispositivo del usuario. Esta solución garantiza que las tareas, el progreso de la mascota y cualquier otro dato relevante permanezcan disponibles incluso cuando la aplicación se cierre o el dispositivo no tenga conexión a Internet.

El lenguaje principal que se ha utilizado para el desarrollo de la aplicación ha sido Kotlin, que además facilita la creación de data classes y enums, recursos clave para modelar entidades como Fruit, Task, TipoFruit y TipoActividad.

Por otro lado, GitHub fue la plataforma empleada para la gestión del código fuente del proyecto, puesto que permite mantener el proyecto organizado, seguro y accesible, garantizando un desarrollo ordenado y profesional.

Canva fue la herramienta utilizada para realizar los diseños preliminares y prototipos visuales de la aplicación FruiTask. Permitió crear una visión estética clara antes de empezar el desarrollo técnico.

Por último, Trello se utilizó para la gestión del proyecto y la administración de tareas. Permitió mantener una metodología ágil, evitando desorganización y facilitando la comunicación dentro del equipo.

5.2. Estructura de datos.

La estructura de datos de FruiTask se centra en dos tablas principales dentro de la base de datos SQLite:

- Fruit: aquí es donde almacenamos las frutas que tiene el usuario. Cada fruta viene con varios atributos, como id, nombre, tipo, nivel y experiencia.

- Task: esta tabla representa las tareas que están asociadas a cada fruta. Cada tarea incluye atributos como id, nombre de la tarea, descripción de la tarea, tipo de actividad, fecha de actividad, estado de la actividad, si la tarea está hecha y fruitId (que es una clave foránea que apunta a la tabla Fruit).

Los modelos que utilizamos son: Fruit y Task. Ambos se corresponden con las tablas antes mencionadas y se representan mediante data classes de Kotlin.

En cuanto a los enums, tenemos TipoFruit (que incluye KIWI, SANDÍA, MANZANA) y TipoActividad (que abarca PROYECTO, TAREA, EXAMEN).

Por otro lado, hay una relación de uno a muchos: cada fruta puede tener varias tareas asociadas. Si se elimina una fruta, todas sus tareas se borrarán automáticamente gracias a la restricción de clave foránea (ON DELETE CASCADE).

Por último, el acceso a la base de datos se realiza a través de DatabaseHelper y FruitTaskRepository. MyViewModel utiliza LiveData y coroutines para mantener la información actualizada en la interfaz sin bloquear la UI.

5.3. Justificación de la elección.

La elección del uso de la base de datos SQLite es debido a que está integrado en Android y por lo tanto no hace falta dependencias externas, es ligero y permite manejar relaciones simples entre frutas y tareas.

Los modelos y enums en Kotlin nos permiten mapear los datos de forma clara y segura, mientras que LiveData y coroutines se encargan de que la interfaz de usuario se mantenga reactiva sin bloquear el hilo principal. La relación uno a muchos y el uso de clave foránea con ON DELETE CASCADE aseguran la integridad y consistencia de los datos.

Finalmente, la estructura del proyecto siguiendo MVVM, junto a repositorios y ViewModels, permite organizar el código de manera limpia, escalable y fácil de mantener, lo cual justifica plenamente las decisiones tecnológicas adoptadas.

6. Permisos y justificación

La aplicación requiere una serie de permisos para poder ofrecer correctamente sus funcionalidades principales. Estos permisos se solicitan siguiendo las directrices de Android y sólo cuando son necesarios para el funcionamiento de la aplicación. En este caso, el único permiso que ha hecho falta para FruiTask ha sido el permiso de notificaciones.

Este permiso permite que la aplicación envíe notificaciones al usuario. Los motivos de pedir este permiso son que:

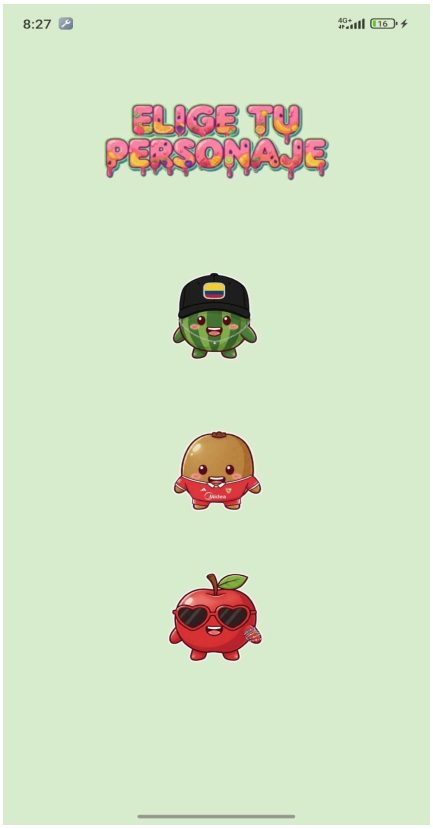
- Permite enviar recordatorios periódicos sobre tareas próximas a vencer o actividades del día.
- La app utiliza un NotificationWorker que ejecuta comprobaciones cada cierto tiempo (por ejemplo, cada 15 minutos) y avisa al usuario si tiene tareas pendientes.
- Facilita mantener la motivación del usuario mediante mensajes de progreso, como la subida de nivel de la mascota.

Sin este permiso, la aplicación no podría avisar al estudiante sobre sus obligaciones académicas, perdiendo una funcionalidad clave.

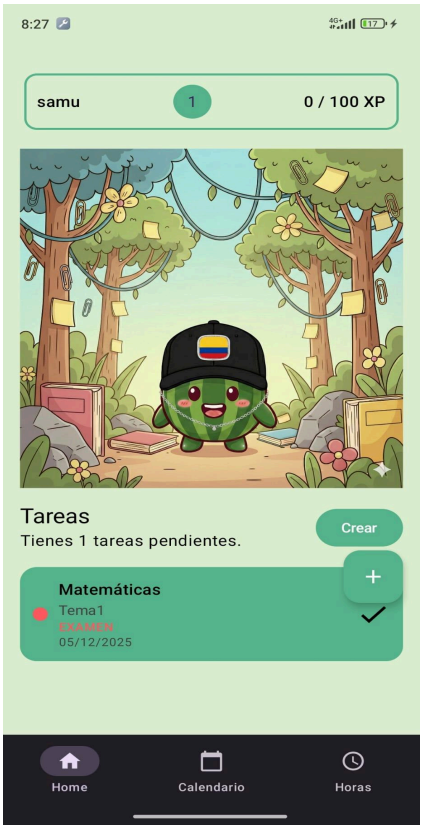
7. Capturas de pantallas y emuladores.

7.1. Capturas de la aplicación.

Pantalla Selección:

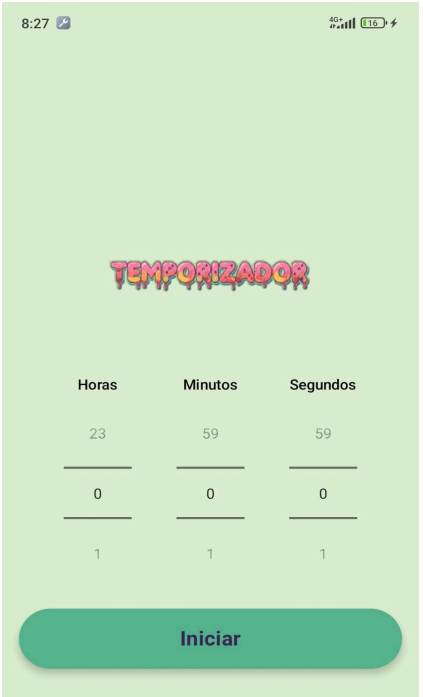


Pantalla 1:

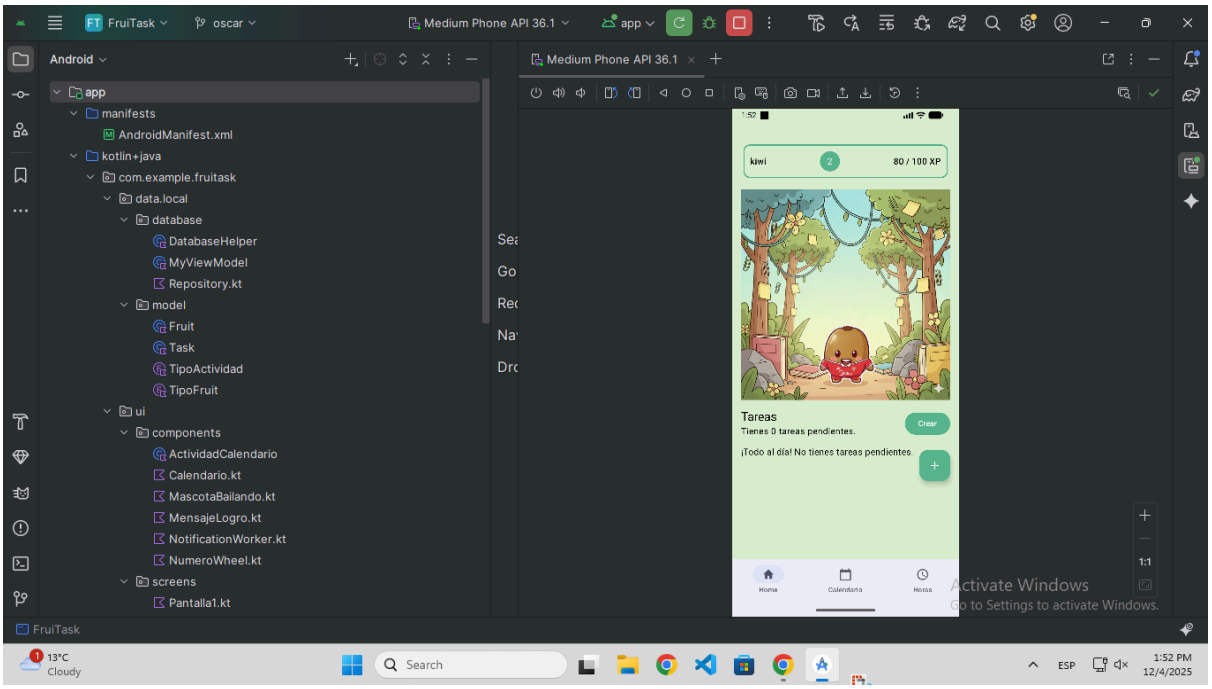


Pantalla 2:

Pantalla 3:



7.2. Evidencias en emuladores.



8. Dificultades encontradas.

8.1. Problemas técnicos.

Durante el desarrollo de la aplicación han ido surgiendo diversos problemas técnicos, como los que se detallan a continuación:

- Dificultades para conectar la aplicación con la base de datos. Al inicio del proyecto, se presentaron problemas para decidir qué método de conexión utilizar con SQLite, ya que existen varias formas posibles: uso directo de SQLiteOpenHelper, implementación de DAOs, o emplear librerías avanzadas como Room.

- Incompatibilidades con versiones de dependencias en Gradle. La aplicación presentó múltiples advertencias y errores debido a incompatibilidades entre versiones de librerías declaradas en module:app. Algunas dependencias no eran compatibles con la versión del SDK o del plugin de Android, lo que provocaba fallos durante la compilación.

- Lentitud del IDE Android Studio. Android Studio tarda mucho tiempo en la ejecución de la aplicación en el emulador, lo que afectó al ritmo de trabajo.

- Conflicto a la hora de usar la base de datos con cada consulta , cerrando la conexión y con eso creando fallos que hemos solucionado.

- El uso de Jetpack Compose, especialmente en la gestión de tareas y navegación, generó dificultades al principio.
- Problemas para mostrar las tareas en el calendario del móvil. Se intentó utilizar el componente nativo CalendarView para resaltar los días con actividades, pero este componente no permite marcar fechas específicas. Como resultado, no era posible mostrar visualmente qué días tenían tareas registradas.

8.2. Soluciones aplicadas.

Las soluciones que se aplicaron a los problemas descritos anteriormente fueron:

- Utilizar la forma explicada por el profesor. Para evitar confusiones y errores, se adoptó el enfoque explicado por el profesor para implementar la base de datos con SQLiteOpenHelper, Repository y ViewModel. Esto permitió definir una estructura clara y estable.
- Utilizar las recomendaciones del IDE de IntelliJ para modificar a la versión más reciente y usar libs.
- Para solventar la lentitud con Android Studio se decidió conectar un dispositivo Android real vía USB para compilar la app con más rapidez.
- Corrección en el manejo de la base de datos. Se solucionó el fallo dejando de cerrar la conexión tras cada operación. La conexión ahora se mantiene abierta mientras sea necesaria y solo se cierra correctamente al finalizar procesos globales
- Para el control de actividades con JetPack Compose, se solucionó gestionando correctamente los estados (state) y la navegación entre pantallas mediante NavController.
- Al descubrir que CalendarView no soporta fechas resaltadas, se optó por crear un calendario personalizado desde cero, definir manualmente la cuadrícula de días, y resaltar los días con tareas mediante colores o indicadores visuales. Este enfoque permitió mantener la funcionalidad prevista por el diseño de la app.

9. Mejoras futuras.

Aunque FruiTask cumple con los objetivos funcionales planteados para esta versión inicial, hay una gran lista de mejoras que hacer en un futuro. Estas mejoras permitirían mejorar la experiencia del usuario, aumentar la personalización y hacer la aplicación más atractiva, interactiva y completa.

1. Incorporar un sistema de cambio del idioma de la aplicación (por ejemplo, entre español, inglés o francés).
2. Permitir crear y gestionar varios perfiles dentro de la misma aplicación. Esta función sería útil para que se animen entre compañeros.
3. Añadir nuevas mascotas y estilos visuales, aumentando la variedad y permitiendo una mayor personalización de la experiencia, haciendo que el usuario se sienta más identificado con su avatar.

4. Incorporar una función de comunicación en línea mediante chat de texto. Esto permitiría que los usuarios interactúen, compartan avances, se motiven mutuamente o incluso formen grupos de estudio.

5. Permitir que el usuario añada tareas mediante reconocimiento de voz, hablando con la fruta.

10. Conclusiones.

El desarrollo de FruiTask ha significado poner en práctica los conocimientos adquiridos anteriormente en clase y enfrentar retos reales del desarrollo de aplicaciones Android, logrando un producto funcional consistente con los objetivos iniciales.

En primer lugar, el proyecto nos ha permitido comprender cómo integrar la gestión de tareas académicas con elementos de gamificación, utilizando una mascota virtual que evoluciona según el rendimiento del usuario. Esta combinación ha demostrado ser una estrategia eficaz para aumentar la motivación y el compromiso, aportando un valor añadido respecto a las clásicas aplicaciones de listas de tareas.

Otro aspecto importante del proyecto ha sido el trabajo con persistencia de datos y actualización automática de la interfaz. El uso de LiveData y coroutines ha permitido que los cambios en la base de datos se reflejen de inmediato en la UI, mejorando la experiencia del usuario y asegurando un funcionamiento fluido de la aplicación.

Durante la creación de la aplicación hubo varios problemas, y resolverlos nos ayudó a mejorar nuestras habilidades para identificar los errores, analizarlos y solucionarlos, además de comprender más el funcionamiento de Android Studio.

Finalmente, el proyecto también ha ayudado a mejorar nuestras competencias en trabajo colaborativo, organización y documentación, utilizando plataformas como GitHub, Trello y Canva para coordinar tareas, diseñar interfaces y registrar los avances del equipo. Esto nos ha permitido llevar a cabo un proyecto completo, desde el diseño conceptual hasta la implementación funcional.