ILERNA

CFGS DESARROLLO DE APLICACIONES MULTIPLATAFORMA
Proyecto Final de Ciclo

TIDYTASK

APLICACIÓN ANDROID PARA LA GESTIÓN DE TAREAS PERSONALES.

Autor: Andrea Catalán Menacho

Tutor: Javier Martin Rivero

Fecha de entrega: 28/05/2025

Convocatoria: 2024 2025

Índice

Introducción.....**2**

Motivación.....**3**

Abstract.....**3**

Objetivo general..... **4**

Objetivos específicos..... **4**

Metodología utilizada…….. **4**

Tecnologías y herramientas utilizadas……. **5**

Estimación de recursos y planificación…… **7**

Requisitos funcionales..... **11**

Requisitos no funcionales…….. **11**

Modelo Entidad-Relación……... **12**

Casos de uso y especificaciones..... **13**

Diseño (mockups y diseño técnico).....**18**

Prototipos de código……**25**

Despliegue y pruebas……...**27**

Conclusiones..... **30**

Opinión personal.....**31**

Vías futuras..... **31**

Glosario..... .....**32**

Bibliografía / Webgrafía.....**34**

Introducción

Actualmente, la organización personal es un aspecto fundamental para el bienestar y la productividad, tanto en el ámbito académico como en el profesional. A lo largo del ciclo formativo de Desarrollo de Aplicaciones Multiplataforma he tenido la oportunidad de adquirir conocimientos en programación, diseño de interfaces, bases de datos y gestión de proyectos, los cuales me han permitido abordar el desarrollo de una aplicación funcional y útil en el día a día.

El proyecto que presento, titulado **TidyTask**, consiste en el desarrollo de una aplicación Android destinada a la gestión de tareas personales. Su objetivo es ayudar a los usuarios a planificar sus actividades, recibir recordatorios y priorizar responsabilidades mediante una interfaz sencilla y accesible.

Esta aplicación ha sido concebida desde una necesidad real de mejorar mi organización diaria, y representa además una forma de integrar y aplicar de forma práctica los conocimientos adquiridos durante los dos años de formación. A lo largo de este documento se detallarán los objetivos planteados, la metodología empleada, las herramientas utilizadas, así como el análisis, diseño, desarrollo y pruebas del proyecto.

Motivación

La idea de este proyecto surgió por una necesidad personal: a lo largo de estos dos años de ciclo he tenido dificultades para organizarme, recordar fechas importantes o tareas pendientes, tanto en el ámbito académico como en el personal. Aunque existen muchas aplicaciones de listas de tareas, la mayoría me resultaban complicadas, poco visuales o demasiado cargadas de funciones que no necesitaba.

Por eso decidí crear una aplicación sencilla, intuitiva y que se adaptara a lo que yo, como estudiante y persona ocupada, realmente necesitaba: apuntar tareas rápidamente, organizarlas por categorías o prioridad, y recibir recordatorios en el momento justo.

Además, este proyecto me ha permitido aplicar de forma práctica todo lo que he aprendido durante el ciclo: desde el diseño de la base de datos hasta el uso de notificaciones y diseño de interfaces en Android. Ha sido una forma de cerrar el ciclo con un proyecto útil, funcional y con el que me siento identificada.

Abstract

TidyTask is an Android application developed as a final project for the Multiplatform Application Development course. The app allows users to create, organize, and manage their daily tasks through a simple and intuitive interface.

The project was built using Kotlin, XML, and SQLite for local data storage. All the logic runs locally, with no internet connection required. While some features like notification reminders are planned for future versions, the current version provides a lightweight and practical tool for personal organization, especially useful for students and professionals with tight schedules.

Objetivo general

Desarrollar una aplicación Android que permita gestionar tareas personales de forma sencilla, rápida y sin necesidad de conexión a internet, con organización por categorías y funciones planificadas de recordatorio para futuras versiones.

Objetivos específicos

- Diseñar una interfaz clara e intuitiva en XML, adaptada a dispositivos móviles.
- Implementar un sistema de creación, edición y eliminación de tareas.
- Almacenar los datos de forma local utilizando SQLite.
- Categorizar las tareas por prioridad.
- Estudiar la incorporación de recordatorios mediante AlarmManager como mejora futura.
- Garantizar que la aplicación funcione de forma completamente offline.
- Aplicar buenas prácticas de desarrollo y usabilidad móvil.

Metodología

Para llevar a cabo este proyecto he seguido una metodología ágil basada en Scrum, adaptada a un trabajo individual. Esta metodología me ha permitido organizar las tareas en fases semanales, priorizar funcionalidades y adaptarme a los cambios o dificultades que han surgido durante el desarrollo.

En primer lugar, realicé una fase de análisis para definir los objetivos del proyecto, los requisitos principales y las tecnologías necesarias. A partir de ahí, dividí el trabajo en bloques semanales donde cada uno se centraba en una parte del desarrollo: diseño de la interfaz, creación de la base de datos, implementación de la lógica principal, pruebas, etc.

Para organizarme utilicé Trello, una herramienta que permite gestionar tareas mediante tableros visuales. En él dividí las tareas en columnas como “Por hacer”,

“En proceso” y “Finalizado”, lo que me ayudó a llevar un control visual del progreso y saber en todo momento en qué punto me encontraba.

Durante el desarrollo también hice pequeñas validaciones funcionales para asegurar que lo que iba implementando cumplía los objetivos previstos, realizando ajustes cuando fue necesario. Esta flexibilidad ha sido clave para poder avanzar de forma progresiva y ordenada.

Tecnologías utilizadas

A continuación, se detallan las principales tecnologías y herramientas utilizadas durante el desarrollo del proyecto:

- **Kotlin** 

Lenguaje principal de programación con el que se ha desarrollado toda la lógica de la aplicación. Kotlin permite escribir código claro, conciso y está plenamente integrado en el desarrollo Android actual.

- **Android Studio** 

Entorno de desarrollo oficial de Android, utilizado para diseñar las interfaces, programar el comportamiento de la app, ejecutar pruebas y generar los APK de instalación.

- **XML** 

Lenguaje de marcado utilizado para el diseño de las interfaces gráficas (layouts). Permite estructurar y definir visualmente cada pantalla de la aplicación.

- **SQLite** 

Sistema de base de datos local integrado en Android, utilizado para almacenar y gestionar las tareas del usuario sin necesidad de conexión a internet. La estructura se ha definido a través de sentencias SQL y clases auxiliares para su gestión.

- **Figma**

Herramienta de diseño utilizada para crear los mockups de la aplicación y planificar visualmente la estructura de las pantallas antes de implementarlas.

- **Trello**

Herramienta de gestión de tareas utilizada para organizar el proyecto siguiendo una metodología ágil. Se han definido las fases del desarrollo y se ha hecho seguimiento semanal del progreso.

- **Canva**

Herramienta de diseño gráfico que ha servido como inspiración para algunos elementos visuales de la interfaz, especialmente para seleccionar ideas de fondos, colores y estilos suaves.

- **Draw.io**

Aplicación online utilizada para realizar el modelo entidad-relación y los diagramas de casos de uso, ya que permite generar esquemas visuales de forma rápida y organizada.

- **GitHub**

Plataforma utilizada para almacenar el código fuente del proyecto, mantener un control de versiones y disponer de una copia de respaldo del trabajo realizado.

- **Google y YouTube**

Fuentes principales de documentación, resolución de dudas y aprendizaje de conceptos técnicos. Han sido clave para avanzar en el desarrollo y solucionar errores.

Estimación de recursos y planificación

A lo largo del desarrollo del proyecto se elaboraron dos versiones del cronograma de trabajo: una estimación inicial y una planificación real. Esta doble planificación permitió contrastar las expectativas con la ejecución efectiva del proyecto, adaptándose así a los tiempos reales de desarrollo y facilitando una mejor gestión del trabajo.

Planificación estimada (inicial)

La planificación inicial del proyecto se diseñó para organizar el trabajo por fases, distribuyendo las tareas a lo largo de once semanas. El objetivo era establecer una hoja de ruta clara que contemplara el análisis, el diseño, la implementación y la entrega final del proyecto.

A continuación, se detalla la tabla con la planificación estimada:

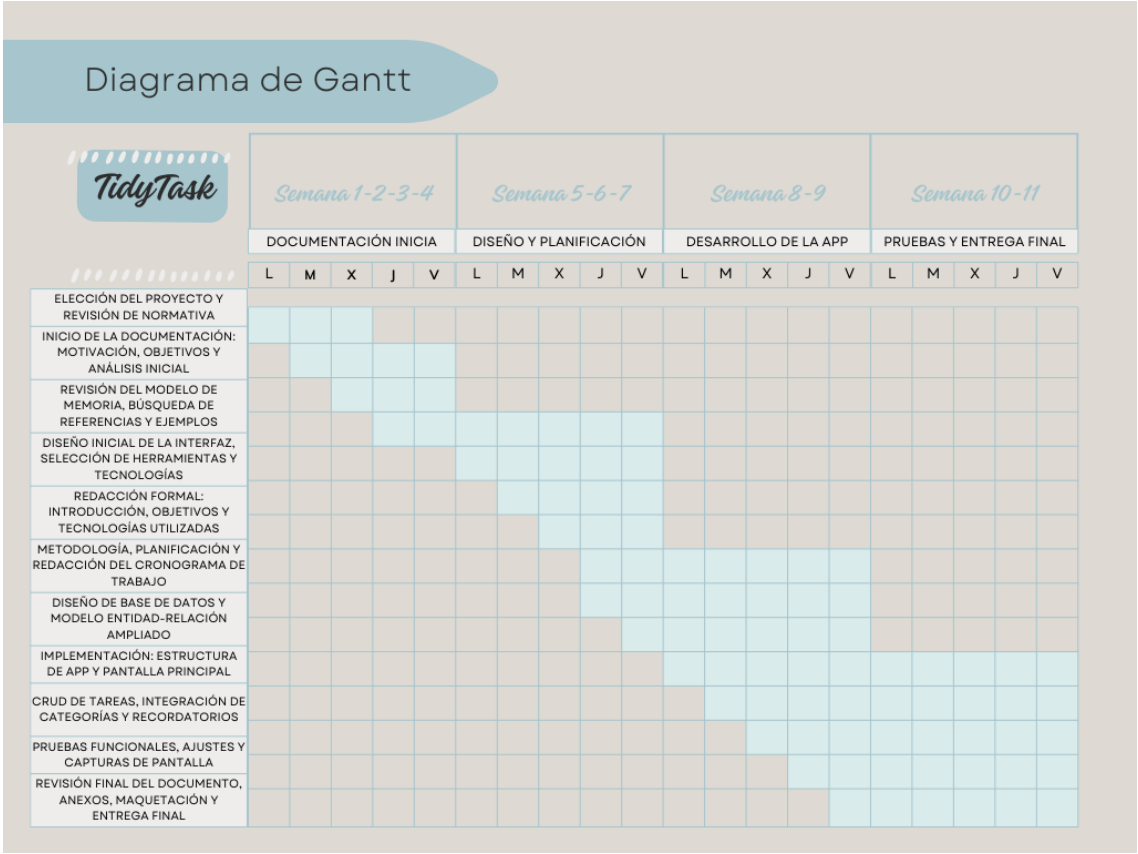
Semana	Fechas	Tareas principales
1	**11 – 17 de marzo**	Elección del proyecto, primera reunión con el tutor y revisión de normativa
2	**18 – 24 de marzo**	Inicio de la documentación: motivación, objetivos y análisis inicial
3	**25 – 31 de marzo**	Revisión del modelo de memoria, búsqueda de referencias y ejemplos
4	**1 – 7 de abril**	Diseño inicial de la interfaz, selección de herramientas y tecnologías
5	**8 – 14 de abril**	Redacción formal: introducción, objetivos y tecnologías utilizadas
6	**15 – 21 de abril**	Metodología, planificación y redacción del cronograma de trabajo
7	**22 – 28 de abril**	Diseño de base de datos y modelo entidad-relación ampliado
8	**29 abril – 5 mayo**	Implementación: estructura de app y pantalla principal

9 **6 – 12 de mayo** CRUD de tareas, integración de categorías y recordatorios

10 **13 – 17 de mayo** Pruebas funcionales, ajustes y capturas de pantalla

11 **18 – 20 de mayo** Revisión final del documento, anexos, maquetación y entrega final

En base a esta tabla se elaboró el siguiente diagrama de Gantt estimado que sirvió como guía visual del cronograma previsto.



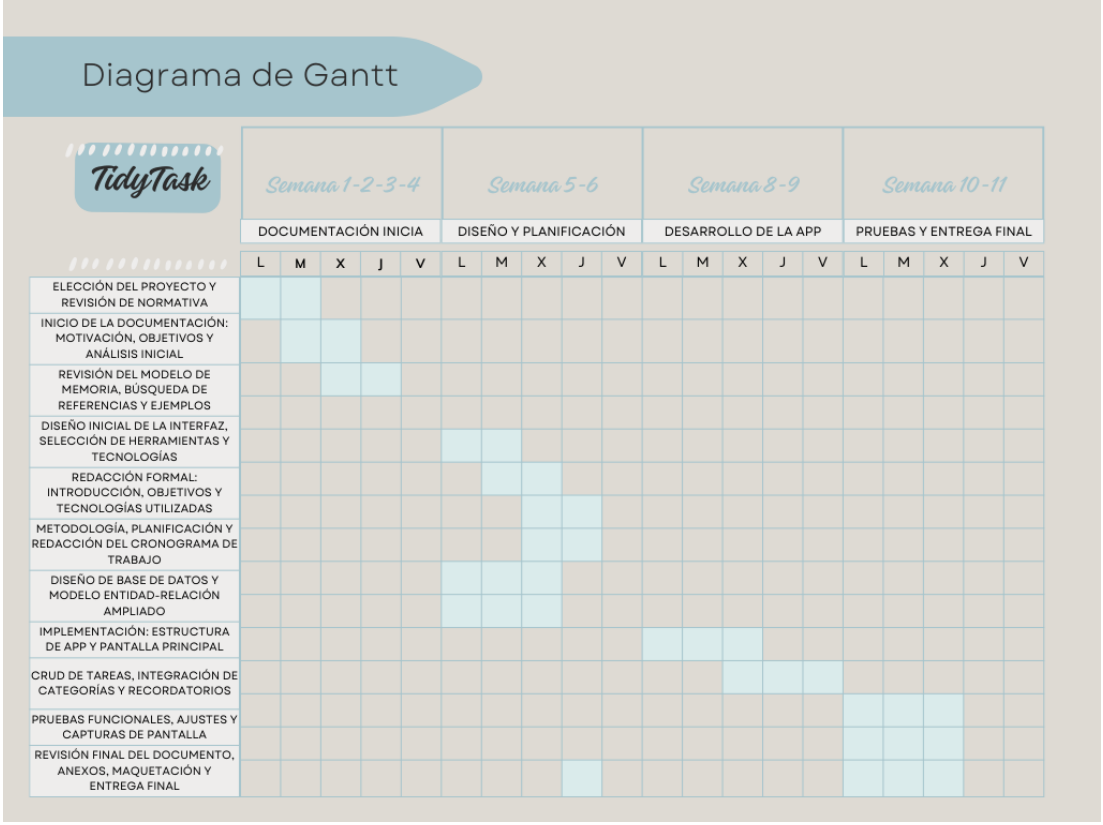
Estimación Real

A medida que avanzaba el proyecto, se fue realizando un seguimiento de los avances reales utilizando el repositorio de GitHub, donde cada commit sirvió como evidencia del trabajo realizado en cada fase.

La siguiente tabla recoge la planificación real en función de los tiempos efectivamente dedicados:

Semana	Fecha aprox.	**Actividades**
Semana 1–4	Hasta 7 abril	Subida de documentación inicial + creación del proyecto
Semana 5–6	8–21 abril	Clases básicas, DB, pruebas JUnit, documentación
Semana 7	22 – 28 abril	Semana dedicada a la revisión general del trabajo, organización de las tareas pendientes y planificación de las siguientes funcionalidades antes del desarrollo intensivo.
Semana 8–9	29 abril – 12 mayo	Funcionalidad: CRUD, selección de fecha, hora, etc.
Semana 10–11	13–21 mayo	Implementación visual, pruebas finales, ajustes, interfaz, mejoras, README, entrega final

A partir de esta tabla se creó el diagrama de Gantt real, reflejando fielmente el tiempo real de ejecución según los registros y la dinámica del proyecto.



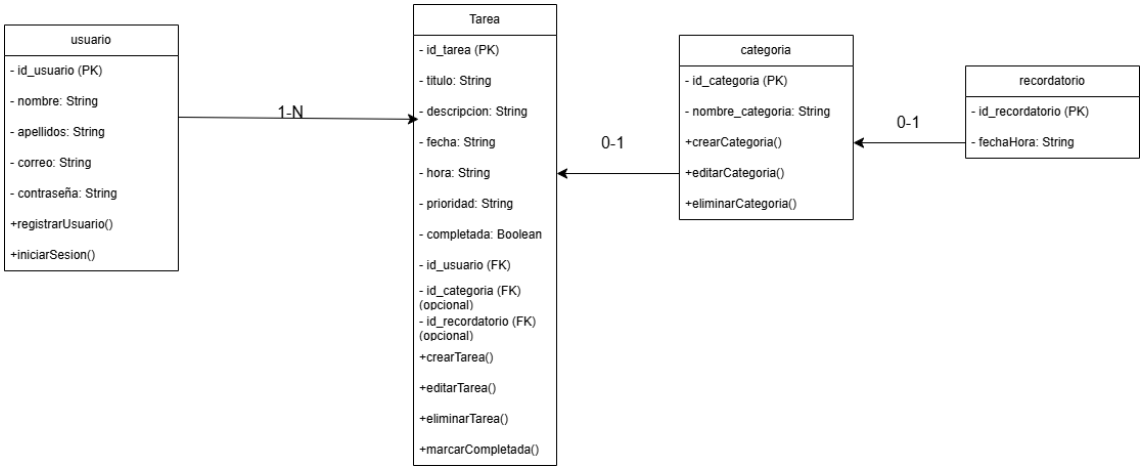
Requisitos funcionales

- RF1: La aplicación debe permitir al usuario registrarse con nombre, correo y contraseña.
- RF2: La aplicación debe permitir iniciar sesión con credenciales válidas.
- RF3: El usuario debe poder crear tareas con título, descripción, fecha, hora y prioridad.
- RF4: El usuario debe poder editar los datos de una tarea existente.
- RF5: El usuario debe poder eliminar tareas.
- RF6: La aplicación debe permitir visualizar una lista de tareas del usuario.
- RF7: El usuario debe poder marcar tareas como completadas.
- RF8: La aplicación debe guardar los datos de forma persistente usando SQLite.
- RF9: El usuario debe poder consultar las tareas con fecha y hora asignadas.
- RF10: El usuario debe poder filtrar tareas por fecha desde la lista de tareas.
- RF11: Las tareas pueden clasificarse por prioridad o categoría.

Requisitos no funcionales

- La aplicación debe funcionar sin conexión a internet.
- La interfaz debe ser clara, intuitiva y fácil de usar.
- El tiempo de respuesta de la aplicación debe ser inferior a 2 segundos por acción.
- La aplicación debe estar diseñada exclusivamente para dispositivos Android.
- Los datos deben guardarse de forma segura en la base de datos local del dispositivo.
- La aplicación debe tener un consumo de batería y recursos optimizado.
- El diseño debe adaptarse a distintos tamaños de pantalla.

Modelo Entidad-Relación



El modelo E-R de la aplicación **TidyTask** se compone de cuatro entidades principales: **Usuario**, **Tarea**, **Categoría** y **Recordatorio**.

La entidad **Usuario** almacena los datos necesarios para gestionar el acceso y las tareas de forma personalizada.

La entidad **Tarea** contiene toda la información relativa a cada actividad: título, descripción, fecha, hora, prioridad, estado de completado, y los identificadores de categoría y recordatorio asociados.

La entidad **Categoría** fue diseñada para representar posibles clasificaciones de tareas (por ejemplo: tipo, ámbito o prioridad). Sin embargo, su funcionalidad completa (crear, editar y gestionar categoría) no ha sido implementada, ya que se consideró más práctico incorporar directamente un campo de texto prioridad en la entidad Tarea.

A pesar de no estar activa en el flujo funcional actual, la entidad **Categoría** se mantiene en el modelo como diseño previsto, abriendo la posibilidad a futuras versiones que permitan una clasificación más avanzada o editable por el usuario.

La entidad **Recordatorio** está prevista para almacenar la fecha y hora asociadas a una tarea. Aunque su funcionalidad completa (lanzar una notificación al usuario) no ha sido implementada, se deja diseñada su estructura como mejora futura.

Cada usuario puede tener varias tareas asociadas (relación 1:N). Cada tarea pertenece a una única categoría y puede tener asociado un recordatorio.

Con esta estructura se garantiza una gestión clara de las tareas, se facilita su organización, y se mantiene la información de forma persistente mediante una base de datos local SQLite.

Casos de Uso

A continuación, se detallan los casos de uso principales del sistema, representados en formato tabla. Cada uno describe el nombre del caso, el actor implicado, los requisitos relacionados, su función dentro de la aplicación y observaciones relevantes para su ejecución.

Caso de uso Nº	1
Nombre	Registrarse
Alias	Crear cuenta
Actores	Usuario
Requisito Funcional	El sistema debe permitir registrarse mediante nombre, correo y contraseña.
Descripción	Permite al usuario crear una cuenta en la app rellenando un formulario.
Referencias	RF1
Comentarios	Se validan datos antes de registrar.

Caso de uso Nº	2
Nombre	Iniciar sesión
Alias	Autenticarse
Actores	Usuario
Requisito Funcional	El sistema debe permitir acceder a usuarios registrados.
Descripción	Permite al usuario acceder a su área privada introduciendo correo y contraseña.
Referencias	RF2
Comentarios	Si las credenciales no son válidas, se muestra un mensaje de error.

Caso de uso Nº	3
Nombre	Crear tarea
Alias	Añadir tarea
Actores	Usuario
Requisito Funcional	El usuario puede añadir una nueva tarea con título, descripción, fecha, hora y prioridad.
Descripción	El sistema permite crear una tarea si los datos son válidos.
Referencias	RF3, RF11
Comentarios	Se incluye validación de campos obligatorios.

Caso de uso Nº	4
Nombre	Editar tarea
Alias	Modificar tarea
Actores	Usuario
Requisito Funcional	El usuario puede modificar tareas ya creadas.
Descripción	Permite al usuario modificar título, fecha, hora o prioridad de una tarea existente
Referencias	RF4, RF11
Comentarios	Solo se permite si la tarea existe y pertenece al usuario.

Caso de uso Nº	5
Nombre	Eliminar tarea
Alias	Borrar tarea
Actores	Usuario
Requisito Funcional	El usuario puede eliminar tareas que ya no necesita.
Descripción	El sistema borra la tarea seleccionada tras confirmar la acción.
Referencias	RF5
Comentarios	Se solicita confirmación para evitar borrado accidental.

Caso de uso Nº	6
Nombre	Visualizar tareas
Alias	Ver tareas
Actores	Usuario
Requisito Funcional	El sistema debe mostrar la lista de tareas del usuario.
Descripción	Muestra todas las tareas del usuario en pantalla tras iniciar sesión.
Referencias	RF6, RF10
Comentarios	La vista está ordenada cronológicamente.

Caso de uso Nº	7
Nombre	Programar recordatorio
Alias	Asignar alarma
Actores	Usuario
Requisito Funcional	El usuario puede asignar una notificación a una tarea.
Descripción	Permite definir una fecha y hora para recibir una alerta relacionada con la tarea.
Referencias	Funcionalidad planificada, no implementada
Comentarios	Se puede añadir desde “crear” o “editar” tarea

Caso de uso Nº	8
Nombre	Consultar recordatorios
Alias	Ver tareas con fecha/hora
Actores	Usuario
Requisito Funcional	El sistema debe permitir visualizar las tareas que tienen recordatorios asignados.
Descripción	Permite al usuario consultar las tareas con fecha y/o hora programada desde el icono de la campana.
Referencias	RF9
Comentarios	Esta funcionalidad no lanza notificaciones automáticas; solo

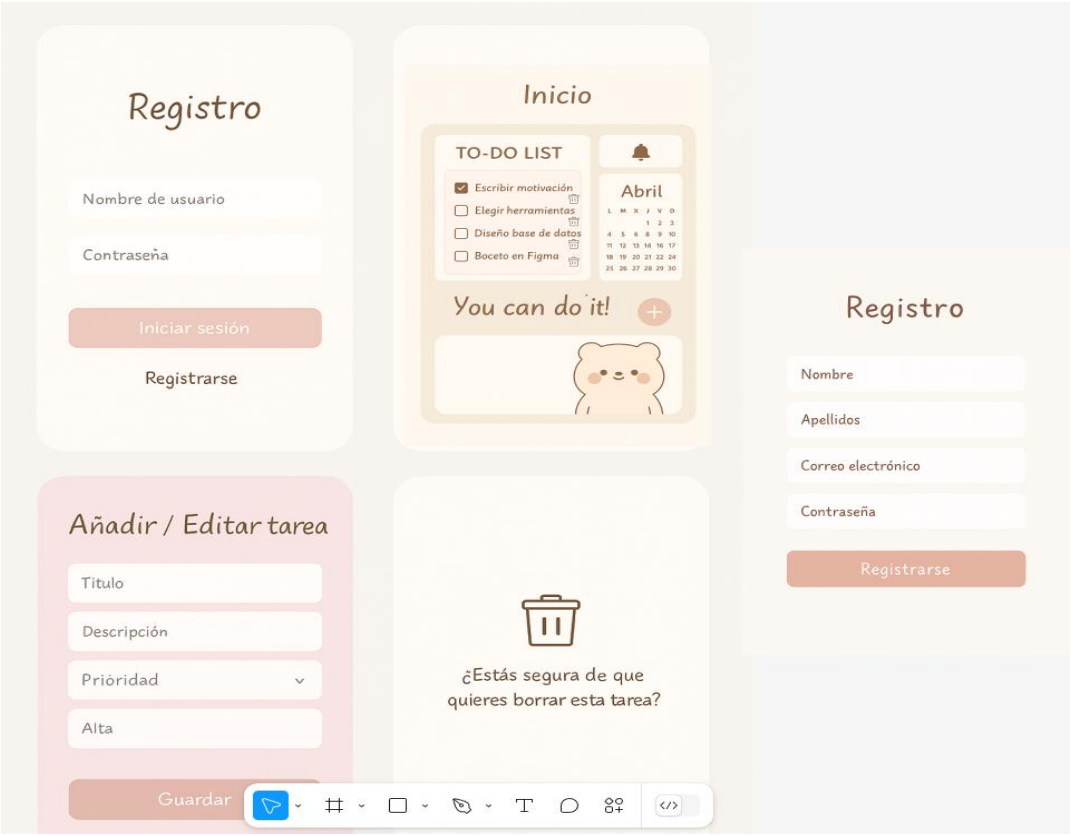
	muestra las tareas que tienen fecha/hora asignada.

Algunos requisitos funcionales como RF7 (marcar tarea como completada) y RF8 (guardar con SQLite) están cubiertos de forma interna o como parte de otros casos de uso y no se describen de forma independiente.

DISEÑO

Para el diseño previo de la aplicación, se elaboró el siguiente mockup en Figma, donde se representan las pantallas clave del sistema: registro, inicio, creación de tareas y confirmación de borrado. Esta maqueta sirvió como referencia visual para mantener la coherencia estética durante la implementación.

Se utilizaron colores pastel suaves (rosado, beige, marrón, salmón) para transmitir calidez y motivación al usuario.



Diseño técnico de la aplicación

El proyecto está estructurado en distintas clases organizadas por funcionalidad. Se ha seguido un enfoque modular, separando claramente la lógica de base de datos, las actividades principales y los adaptadores.

Clases de modelo:

Tarea: contiene los atributos principales de una tarea (título, descripción, fecha, hora, prioridad, estado de completado, etc.).

Usuario: representa los datos del usuario (nombre, correo, contraseña).

Recordatorio: almacena la hora asociada a una tarea para consultas visuales.

Base de datos (SQLite)

TareaDBHelper: gestiona todas las operaciones con la base de datos local: insertar, editar, eliminar y consultar tareas. Incluye funcionalidades de filtrado por fecha y tareas con hora asignada.

UsuarioDBHelper: permite registrar nuevos usuarios y validar el inicio de sesión

Actividades principales

InicioActivity: Es la pantalla inicial desde la que el usuario puede elegir entre registrarse o iniciar sesión. Contiene dos botones que redirigen a **RegistroActivity o LoginActivity.**

RegistroActivity: Pantalla que permite al usuario registrarse completando un formulario con su nombre, apellidos, correo y contraseña. Se valida que todos los campos estén rellenados y que el correo no esté duplicado en la base de datos.

La operación de registro se gestiona desde la clase **UsuarioDBHelper.** En caso de éxito, se muestra un mensaje y se retorna a la pantalla anterior.

Incluye también un botón de retroceso para volver a la pantalla de inicio.

```
if (nombre.isNotEmpty() && apellidos.isNotEmpty() && correo.isNotEmpty() && contrasena.isNotEmpty()) {
    val usuario = Usuario( idUsuario: 0, nombre, apellidos, correo, contrasena)
    val registrado = dbHelper.registrarUsuario(usuario)

    if (registrado) {
        Toast.makeText( context: this,  text: "Usuario registrado con éxito", Toast.LENGTH_SHORT).show()
        finish()
    } else {
        Toast.makeText( context: this,  text: "Ese correo ya está registrado", Toast.LENGTH_SHORT).show()
    }
```

LoginActivity.kt: Esta pantalla permite al usuario acceder a su cuenta introduciendo el correo y la contraseña previamente registrados.

Se valida que los campos no estén vacíos y se comprueban las credenciales mediante el método **validarUsuario()** de la clase **UsuarioDBHelper**.

Si el acceso es correcto, se muestra un mensaje y se redirige a la pantalla principal **(TareasActivity)**. En caso contrario, se informa del error.

Incluye además un botón para volver atrás.

```
if (correo.isNotEmpty() && contrasena.isNotEmpty()) {
    val valido = dbHelper.validarUsuario(correo, contrasena)

    if (valido) {
        Toast.makeText( context: this,  text: "Inicio de sesión exitoso", Toast.LENGTH_SHORT).show()
        startActivity(Intent( packageContext: this, TareasActivity::class.java))
        finish()
    } else {
        Toast.makeText( context: this,  text: "Credenciales incorrectas", Toast.LENGTH_SHORT).show()
    }
```

TareasActivity.kt: es el núcleo funcional de la aplicación, donde el usuario visualiza, filtra, edita y elimina sus tareas.

Utiliza un **RecyclerView** con un **TareaAdapter** personalizado que permite mostrar el listado, desplegar detalles, marcar como completadas y eliminar tareas mediante iconos.

Entre sus funcionalidades destacadas se incluyen:

Filtrado por fecha: permite seleccionar tareas de un día concreto mediante un DatePickerDialog.

Filtrado por prioridad: mediante un Spinner, el usuario puede ver solo las tareas "Alta", "Media" o "Baja".

Visualización de recordatorios: mediante un icono de campana, se muestran las tareas con hora asignada.

Botón de cerrar sesión: redirige al usuario a la pantalla de inicio

- **Método filtrarPorPrioridad():**

```
private fun filtrarPorPrioridad(prioridadSeleccionada: String) {
    val dbHelper = TareaDBHelper( context: this)
    val todasLasTareas = dbHelper.obtenerTodasLasTareas() // o el método que tú tengas

    val tareasFiltradas = when (prioridadSeleccionada) {
        "Alta" -> todasLasTareas.filter { it.prioridad.equals( other: "Alta", ignoreCase = true) }
        "Media" -> todasLasTareas.filter { it.prioridad.equals( other: "Media", ignoreCase = true) }
        "Baja" -> todasLasTareas.filter { it.prioridad.equals( other: "Baja", ignoreCase = true) }
        else -> todasLasTareas
    }

    tareaAdapter.actualizarLista(tareasFiltradas)
}
```

- **Método mostrarTareas() (refresca el listado y lanza EditarTareaActivity):**

```
private fun mostrarTareas() {
    val listaTareas = dbHelper.obtenerTodasLasTareas()
    tareaAdapter = TareaAdapter(
        listaTareas,
        onTareaClick = { tarea ->
            val intent = Intent( packageContext: this, EditarTareaActivity::class.java).apply {
                putExtra( name: "idTarea", tarea.idTarea)
                putExtra( name: "titulo", tarea.titulo)
                putExtra( name: "descripcion", tarea.descripcion)
                putExtra( name: "prioridad", tarea.prioridad)
                putExtra( name: "fecha", tarea.fecha)
                putExtra( name: "hora", tarea.hora)
            }
            startActivity(intent)
        },
        onTareaLongClick = { tarea ->
            mostrarDialogoEliminar(tarea)
        }
    )

    recyclerView.layoutManager = LinearLayoutManager( context: this)
    recyclerView.adapter = tareaAdapter
}
```

AgregarTareaActivity.kt: Esta pantalla permite al usuario crear una nueva tarea completando un formulario con los siguientes campos: título, descripción, prioridad (mediante **Spinner**), fecha **(DatePicker)** y hora **(TimePicker).**

Al pulsar el botón Guardar tarea, se valida que los campos obligatorios estén completos y, si es así, se inserta la nueva tarea en la base de datos mediante **TareaDBHelper**.

```
btnGuardar.setOnClickListener {
    val titulo = etTitulo.text.toString()
    val descripcion = etDescripcion.text.toString()
    val prioridad = spinnerPrioridad.selectedItem.toString()

    if (titulo.isNotEmpty() && descripcion.isNotEmpty()) {
        val tarea = Tarea(
            idTarea = 0,
            titulo = titulo,
            descripcion = descripcion,
            fecha = fechaSeleccionada ?: "",
            hora = horaSeleccionada ?: "",
            prioridad = prioridad,
            completada = false,
            idUsuario = 1
        )

        val insertada = dbHelper.insertarTarea(tarea)

        if (insertada) {
            Toast.makeText( context: this,  text: "Tarea guardada con éxito", Toast.LENGTH_SHORT).show()
            finish()
        } else {
            Toast.makeText( context: this,  text: "Error al guardar la tarea", Toast.LENGTH_SHORT).show()
        }

    } else {
        Toast.makeText( context: this,  text: "Por favor completa todos los campos", Toast.LENGTH_SHORT).show()
    }
}
```

EditarTareaActivity.kt : Esta pantalla permite al usuario modificar una tarea existente. Al acceder desde la lista, se cargan automáticamente los datos actuales (título, descripción, prioridad, fecha y hora) mediante los Intent extras.

El usuario puede cambiar cualquier campo y guardarlo, lo que actualiza la tarea en la base de datos local **(TareaDBHelper).** También puede cancelar los cambios y volver atrás.

```
btnGuardar.setOnClickListener {
    val nuevoTitulo = etTitulo.text.toString()
    val nuevaDescripcion = etDescripcion.text.toString()
    val nuevaPrioridad = spinnerPrioridad.selectedItem.toString()
    val fechaFinal = fechaSeleccionada ?: intent.getStringExtra( name: "fecha")
    val horaFinal = horaSeleccionada ?: intent.getStringExtra( name: "hora")

    if (nuevoTitulo.isBlank() || nuevaDescripcion.isBlank()) {
        Toast.makeText( context: this,  text: "Completa todos los campos", Toast.LENGTH_SHORT).show()
        return@setOnClickListener
    }

    val actualizada = dbHelper.actualizarTarea(tareaId, nuevoTitulo, nuevaDescripcion, nuevaPrioridad, fechaFinal, horaFinal)
    if (actualizada) {
        Toast.makeText( context: this,  text: "Tarea actualizada", Toast.LENGTH_SHORT).show()
        finish()
    } else {
        Toast.makeText( context: this,  text: "Error al actualizar", Toast.LENGTH_SHORT).show()
    }
}
```

TareaAdapter.kt : es el adaptador personalizado del **RecyclerView** de tareas. Se encarga de inflar cada ítem de la lista **(item_tarea.xml)** y de gestionar las siguientes acciones:

- Mostrar los datos principales de cada tarea.
- Permitir marcarla como completada (con estilo tachado).
- Eliminar una tarea mediante un botón de papelera.
- Expandir o contraer detalles (descripción, fecha y hora) con una flecha.

La expansión se gestiona usando un set interno de tareas expandidas, y el tachado visual del título se aplica usando **Paint.STRIKE_THRU_TEXT_FLAG.**

```
// Expandir/colapsar detalles
val estaExpandida = tareasExpandidas.contains(tarea.idTarea)
holder.contenedorDetalles.visibility = if (estaExpandida) View.VISIBLE else View.GONE
holder.btnExpandir.setBackgroundResource(
    if (estaExpandida) android.R.drawable.arrow_up_float
    else android.R.drawable.arrow_down_float
)

holder.btnExpandir.setOnClickListener {
    if (tareasExpandidas.contains(tarea.idTarea)) {
        tareasExpandidas.remove(tarea.idTarea)
    } else {
        tareasExpandidas.add(tarea.idTarea)
    }
    notifyItemChanged(position)
}
```

TareaDBHelper.kt: es la clase que gestiona el acceso a la base de datos SQLite. Contiene todos los métodos CRUD (crear, leer, actualizar, eliminar) para las tareas.

También incluye funciones auxiliares como:

obtenerTodasLasTareas()

insertarTarea(tarea: Tarea)

actualizarTarea(...)

eliminarTareaPorId(...)

obtenerTareasConHora()

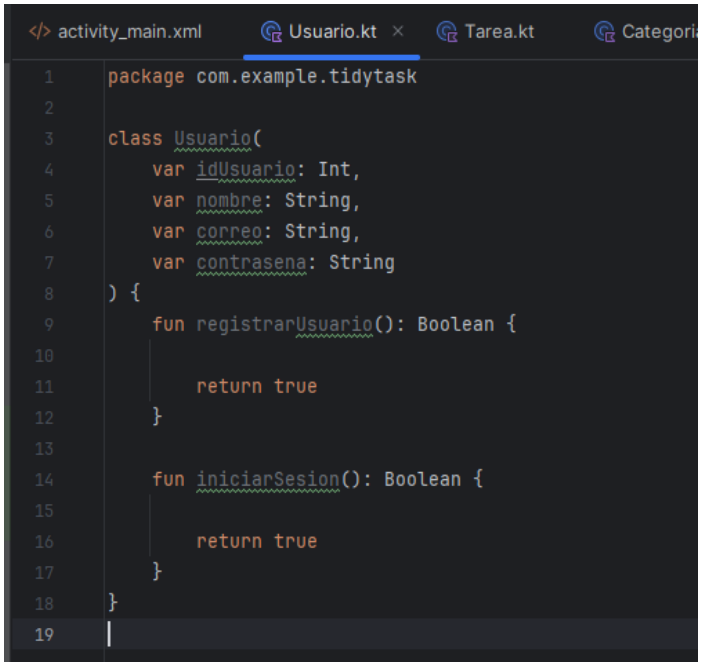
actualizarEstadoCompletada(...)

Gracias a esta clase, los datos de las tareas se mantienen de forma persistente en el dispositivo.

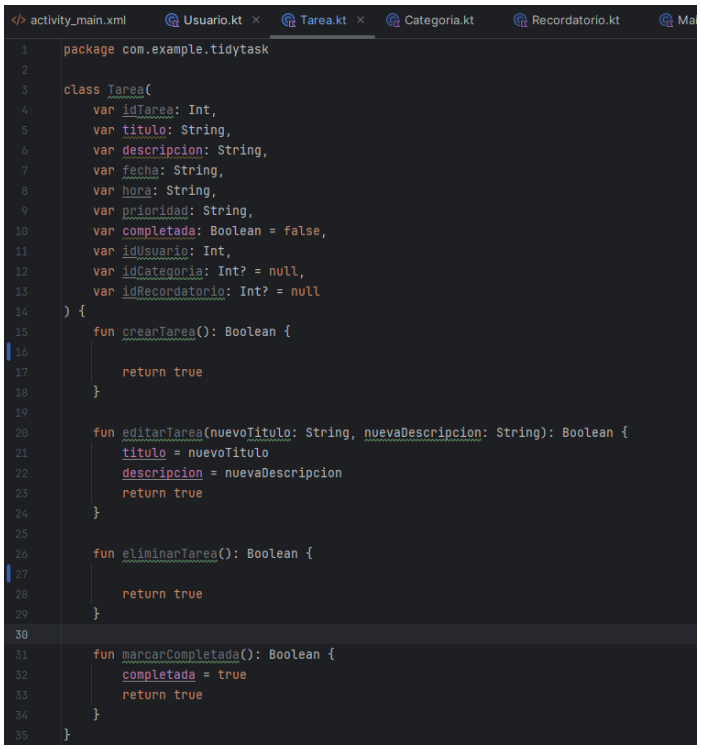
```
fun insertarTarea(tarea: Tarea): Boolean {
    val db = writableDatabase
    val values = ContentValues().apply {
        put("titulo", tarea.titulo)
        put("descripcion", tarea.descripcion)
        put("fecha", tarea.fecha)
        put("hora", tarea.hora)
        put("prioridad", tarea.prioridad)
        put("completada", if (tarea.completada) 1 else 0)
        put("idUsuario", tarea.idUsuario)
    }

    val resultado = db.insert( table: "tareas",  nullColumnHack: null, values)
    db.close()
    return resultado != -1L
}
```

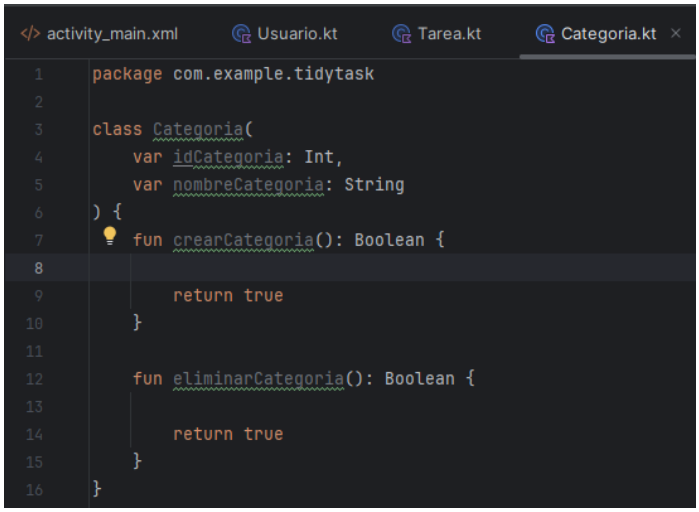
PROTOTIPOS DE CÓDIGO



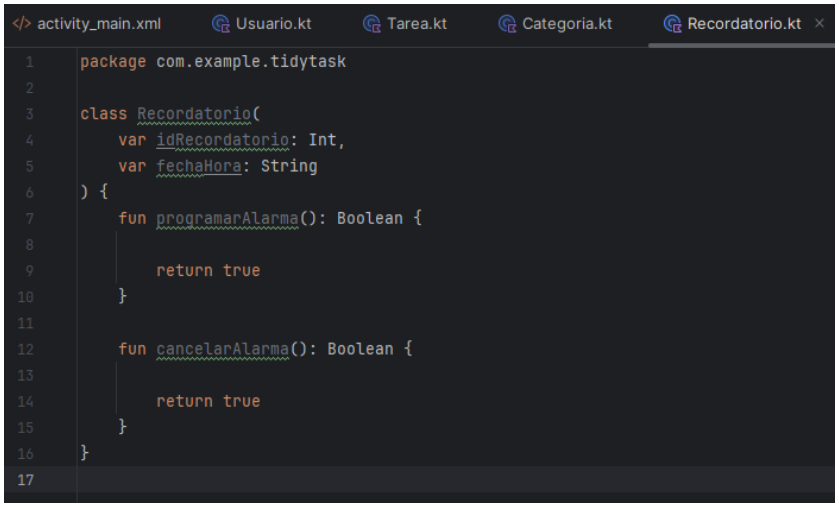
La clase Usuario define los atributos principales de un usuario registrado, y proporciona funciones básicas para registrar nuevos usuarios e iniciar sesión en la aplicación.



La clase Tarea contiene la estructura básica de una tarea, incluyendo atributos como título, descripción, fecha y prioridad. La tarea, podrá ser categorizada o recordada si el usuario así lo desea pero no es obligatorio. Además, implementa funciones para crear, editar, eliminar y marcar tareas como completadas.



La clase Categoría permitirá agrupar tareas bajo diferentes categorías creadas por el usuario, facilitando su organización. Incluye funciones para crear y eliminar categorías.



La clase Recordatorio permitirá asociar una fecha y hora a una tarea para programar una notificación futura. Simula las funciones de programar y cancelar alarmas.

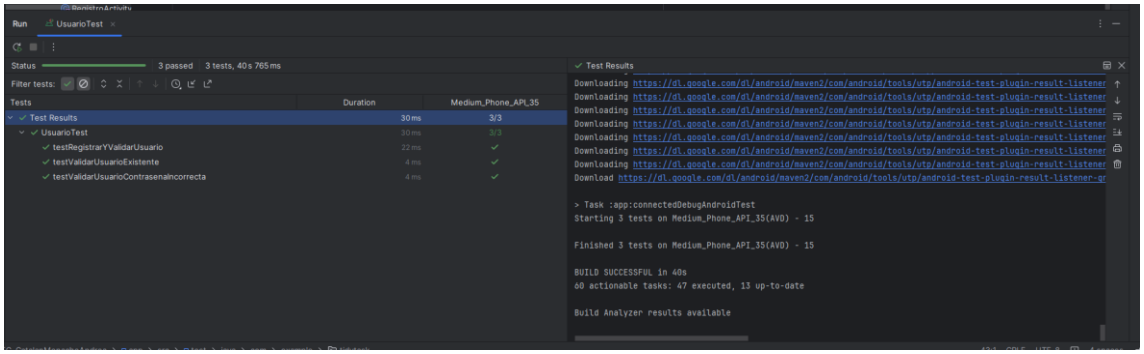
Despliegue y pruebas

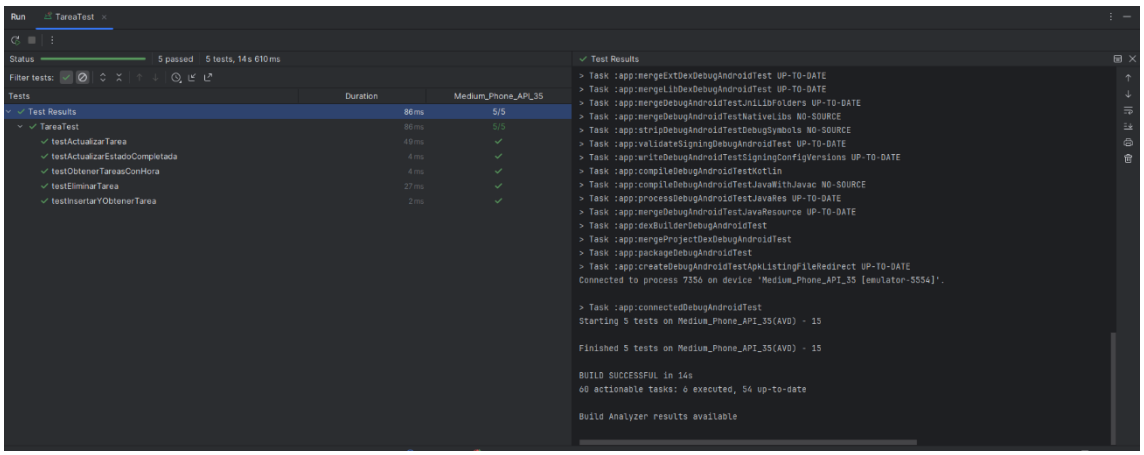
A lo largo del desarrollo se han realizado pruebas funcionales de caja negra, centradas en verificar que las acciones visibles por el usuario funcionen correctamente según los requisitos definidos. A continuación, se muestran algunas de las pruebas realizadas:

Nº	Objetivo probado	Requisitos probados	Pruebas que realizar
1	Registro de usuario	RF1	Introducir datos válidos en el formulario de registro y pulsar “Registrarse”.
2	Inicio de sesión	RF2	Introducir credenciales válidas y acceder a la app.
3	Creación de tarea	RF3, RF8	Crear una tarea con título, descripción, fecha y prioridad, y verificar que aparece.
4	Eliminación de tarea	RF5	Pulsar el icono de papelera, aceptar el diálogo de confirmación y comprobar que desaparece.
5	Marcar tarea como completada	RF7	Marcar el checkbox de una tarea y comprobar que se aplica el tachado.
6		RF11	Seleccionar “Alta” en el Spinner y verificar que se

	Filtrar tareas por prioridad		muestran solo esas tareas.
7	Filtrar tareas por fecha	RF10	Seleccionar una fecha desde el botón “Filtrar por fecha” y comprobar la lista.
8	Ver tareas con hora asignada (campana)	RF9	Pulsar el icono de campana y verificar que se muestran solo las tareas con hora.

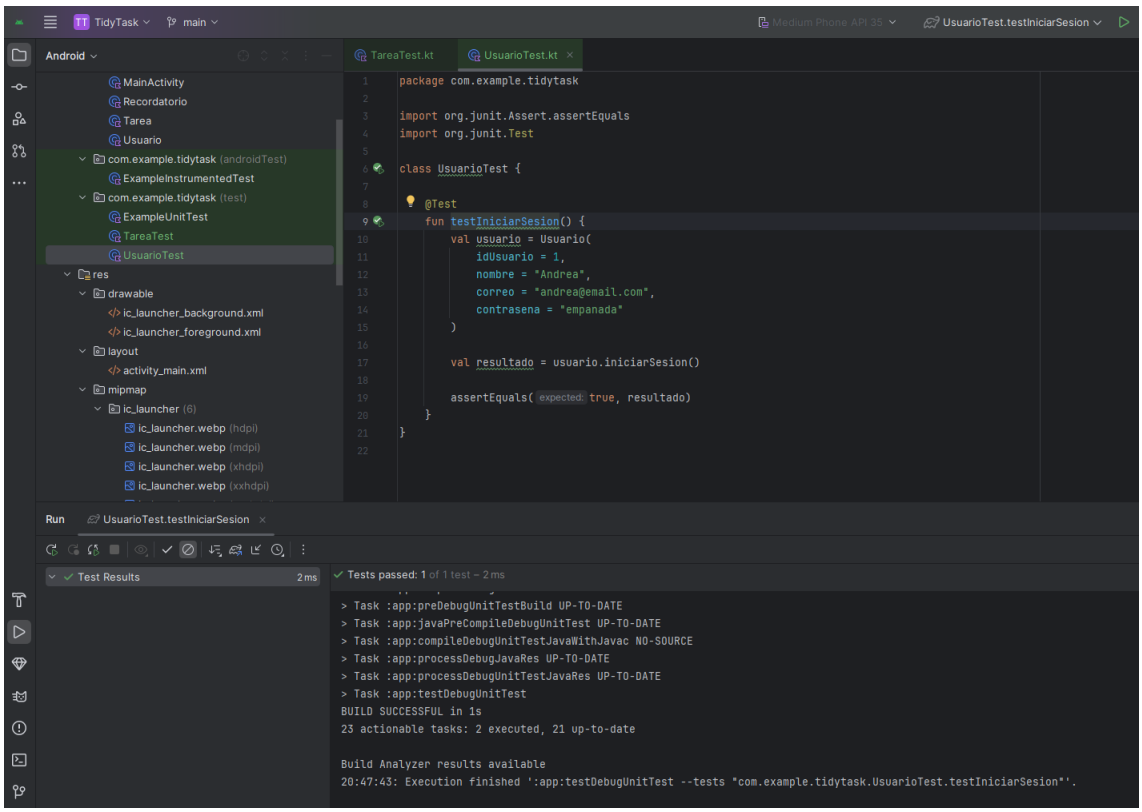
Además, se han complementado estas pruebas funcionales con pruebas unitarias en las clases **TareaTest y UsuarioTest**, usando JUnit, para asegurar que las operaciones de base de datos funcionan de forma interna correctamente.





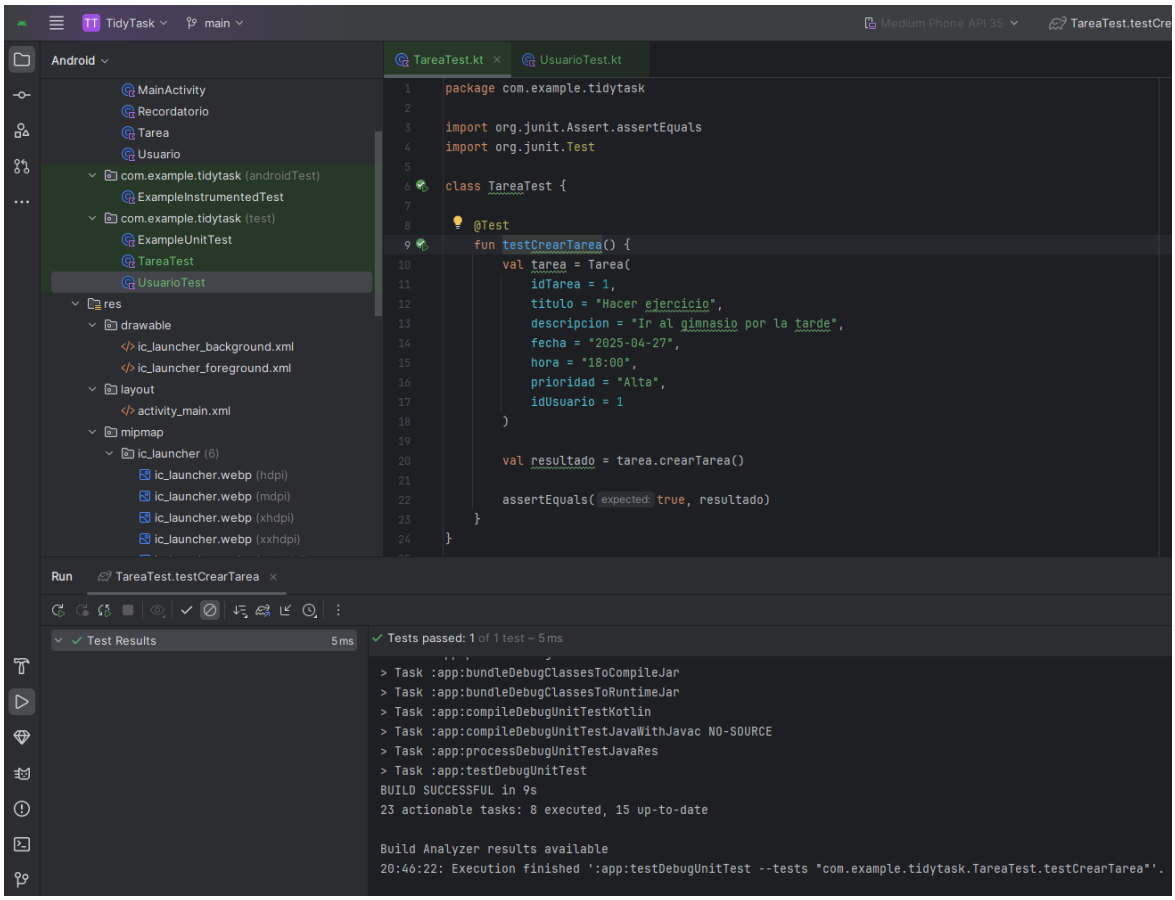
TEST PROTOTIPOS

Prueba unitaria de inicio de sesión de usuario en TidyTask



Se comprueba que el método iniciarSesion() permite acceder correctamente cuando el usuario tiene los datos completos.

Prueba unitaria de creación de tarea en TidyTask.



Se comprueba que el método crearTarea() funciona correctamente, devolviendo true al crear una tarea con todos sus datos completos.

Conclusiones

Tras varias semanas de trabajo, el desarrollo de la aplicación TidyTask ha supuesto un gran reto y, a la vez, una oportunidad para aplicar de forma práctica muchos de los conocimientos que he ido adquiriendo a lo largo del ciclo. El proyecto ha evolucionado desde una idea inicial sencilla hasta convertirse en una aplicación funcional, con una interfaz cuidada y una estructura técnica coherente.

A lo largo del proceso he conseguido cumplir casi todos los objetivos propuestos: el usuario puede registrarse, iniciar sesión, crear, editar, eliminar y visualizar tareas. También se ha implementado el filtrado por fecha y prioridad, la gestión local de datos con SQLite y un pequeño sistema de recordatorio visual mediante hora asignada.

Aunque no llegué a implementar el sistema de notificaciones automáticas con AlarmManager, la estructura está preparada para poder integrarlo en un futuro.

Uno de los puntos más positivos del proyecto ha sido mantener una coherencia estética desde el diseño del mockup hasta la app final. Me siento especialmente satisfecha con el diseño visual, el uso de colores suaves y una interfaz que transmite orden, simplicidad y calidez.

Opinión personal

Este proyecto ha sido, sin duda, la parte más desafiante de mi formación. No solo por la carga técnica, sino también por la necesidad de organizarme, resolver errores por mi cuenta y mantener la motivación incluso cuando algo no salía como esperaba. He aprendido muchísimo, sobre todo a trabajar de manera más autónoma y a confiar un poco más en mis capacidades, aunque a veces haya dudado.

Me llevo una experiencia muy valiosa. He descubierto que soy capaz de desarrollar algo útil, tomar decisiones de diseño, encontrar soluciones y, sobre todo, terminar lo que empiezo. Sé que aún me queda mucho por aprender, pero haber completado este proyecto me ha demostrado que estoy en el camino correcto.

Vías futuras

Aunque he conseguido que la aplicación TidyTask funcione correctamente y cumpla con los objetivos principales, hay algunas partes que me gustaría mejorar o ampliar en un futuro si tuviera más tiempo.

Una de ellas es la parte de los recordatorios. Aunque ahora se puede asignar una hora a cada tarea y se muestran desde el icono de la campana, no llegué a implementar las notificaciones reales con AlarmManager, algo que me hubiera gustado conseguir. Dejo todo preparado por si más adelante quiero retomarlo.

También pensé en hacer algo más con la parte de categorías, como permitir al usuario crearlas, editarlas o clasificarlas mejor. Por ahora lo solucioné con el campo de

prioridad, que cumple una función parecida, pero la entidad Categoría sigue ahí en el diseño por si decido desarrollarla más adelante.

Otra mejora sería permitir que más de un usuario pueda iniciar sesión y tener sus propias tareas, ya que ahora mismo todo se gestiona de forma local y hay un único usuario en uso, aunque la base de datos sí permite tener varios.

Y por último, también me gustaría en el futuro hacer que los datos se pudieran guardar online, por ejemplo usando Firebase, para no perder las tareas si se borra la app, y tal vez incluso compartir listas con otros usuarios. Además, podría seguir mejorando el diseño y añadir un modo oscuro, o hacer que la app se adapte aún mejor a distintos tamaños de pantalla.

En resumen, el proyecto está completo y funcional, pero sé que hay muchas posibilidades de seguir aprendiendo y mejorando si más adelante quiero retomarlo con calma.

Glosario

Activity: Pantalla o interfaz principal de una app Android. Cada vez que se abre una nueva sección (como registrar una tarea o editarla), se lanza una nueva actividad.

Intent: Mecanismo que permite pasar de una activity a otra. Se usa, por ejemplo, para ir del login a la lista de tareas o para abrir la pantalla de edición.

RecyclerView: Componente que permite mostrar listas de elementos (en este caso, las tareas) de forma eficiente, incluso si hay muchas. Se usa junto con un adaptador.

Adapter (TareaAdapter): Clase que se encarga de "dibujar" cada tarea en la lista del RecyclerView, y también de controlar qué pasa al marcarla como completada, eliminarla o desplegar detalles.

SQLite: Base de datos local que permite guardar los datos aunque se cierre la app. Toda la información de tareas y usuarios se almacena ahí.

Spinner: Desplegable que permite al usuario elegir entre varias opciones, como la prioridad de la tarea (Alta, Media o Baja).

DatePicker / TimePicker: Cuadros de diálogo que permiten seleccionar una fecha o una hora de forma visual. Se usan para asignar recordatorios a las tareas.

Toast: Mensaje temporal que aparece en pantalla para avisar al usuario de que algo se ha hecho correctamente o ha fallado.

AlarmManager: Herramienta de Android que permite programar alertas o notificaciones en el futuro. Aunque no se ha implementado, la estructura está pensada para integrarlo.

JUnit: Framework usado para crear pruebas unitarias que permiten comprobar si ciertas partes del código funcionan correctamente (como insertar o eliminar tareas).

notifyDataSetChanged / notifyItemRemoved: Funciones del adaptador que actualizan la vista del RecyclerView cuando cambian los datos (por ejemplo, si se borra una tarea).

View.GONE / View.VISIBLE: Constantes que indican si un elemento debe estar oculto o visible en pantalla. Se usan para mostrar u ocultar botones como el de volver a la lista completa.

Bibliografía

Documentación oficial:

Android Developers. (s.f.). *Develop apps for Android*. Google. https://developer.android.com/

Google. (s.f.). *AlarmManager | Android Developers*. https://developer.android.com/reference/android/app/AlarmManager

SQLite. (s.f.). *SQLite Documentation*. https://www.sqlite.org/docs.html

Tutoriales en vídeo:

Programmer Android. (2022, 17 noviembre). *Build A To-Do List App in Android Studio | Beginner's Guide* [Video]. YouTube. https://www.youtube.com/watch?v=i9mkAoZ8FNk

Coding with Dev. (2022, 3 diciembre). *To Do List App using RecyclerView Android Studio Kotlin Example* [Video]. YouTube. https://www.youtube.com/watch?v=RfIR4oaSVfQ

HellSoft. (2023, 10 abril). *Crear una Lista de Tareas (ToDoList) en Android Studio* [Video]. YouTube. https://www.youtube.com/watch?v=LP7PQsdWIqw

SoyProgramador. (2023, 1 mayo). *LISTA DE TAREAS EN KOTLIN - ANDROID STUDIO* [Video]. YouTube. https://www.youtube.com/watch?v=hzYy_1O7L7U

TheCrazyProgrammer. (2023, 14 julio). *Crear Listas dinámicas con RecyclerView en Android Studio y Kotlin* [Video]. YouTube. https://www.youtube.com/watch?v=aLVIqRJziPY

Repositorios en GitHub:

Coding-Meet. (2022). *Todo-App*. GitHub. https://github.com/Coding-Meet/Todo-App

Jesper. (2020). *todo-kotlin*. GitHub. https://github.com/jesper/todo-kotlin

Diseños de interfaz (UI):

Freepik. (s.f.). *Kit interfaz usuario aplicación móvil ToDo List*. https://www.freepik.es/psd-premium/kit-interfaz-usuario-aplicacion-movil-todo-list_33005639.htm

Pinterest. (s.f.). *To Do List | Diseño de aplicación móvil*. https://www.pinterest.com/pin/453315518744284193/

Otros recursos de apoyo:

Stack Overflow. (s.f.). *Preguntas y respuestas técnicas sobre desarrollo Android*. https://stackoverflow.com/

Medium. (s.f.). *Artículos y guías sobre desarrollo Android y Kotlin*. https://medium.com/

GitHub. (s.f.). *Ejemplos de apps Android Kotlin*. https://github.com/search?q=android+kotlin+todo+list&type=repositories

OpenAI. (2024). *Asistencia técnica y generación de contenido mediante inteligencia artificial*.