Instituto Puig Castellar
Ciclo formativo: DAM 2 B
Grado: CFGM / CFGS
Crédito de Síntesis / Proyecto intermodular

Memento

“Recuerda Todo”

Autor/a/es: Arcides Manuel Mateo Terrero
Tutor: Nombre y Apellidos
Curso: 2025-2026
Fecha de entrega: 17/05/2026

Resumen del proyecto
Este proyecto se centra en la temática de la organización personal y la gestión del tiempo mediante herramientas digitales. Parte de la problemática actual en la que muchas personas tienen dificultades para planificar sus tareas, recordar eventos importantes y mantener un equilibrio entre estudios, trabajo y vida personal, lo que a menudo provoca estrés y baja productividad.

El objetivo principal del proyecto es desarrollar una aplicación de gestión de tareas y recordatorios, dirigida a cualquier persona que quiera mejorar su organización diaria de una manera sencilla y eficiente.

La metodología seguida incluye el análisis de necesidades de los usuarios, el diseño de una interfaz intuitiva y accesible, y el desarrollo de una aplicación nativa para Android. La solución integra funciones como la creación de tareas, calendario interactivo, notificaciones automáticas, sincronización en la nube.

Como conclusiones, el proyecto demuestra que una herramienta digital bien diseñada puede mejorar significativamente la organización personal, reducir la carga mental y aumentar la productividad. Además, se confirma la importancia de un diseño claro y adaptable para garantizar una buena experiencia de usuario.

Palabras clave: Gestión de tareas, organización personal, sincronización en la nube, recordatorios, productividad digital, Android, Kotlin.

Abstract:
This project focuses on personal organization and time management through digital tools. It stems from the current problem that many people struggle to plan their tasks, remember important events, and maintain a balance between studies, work, and personal life, which often leads to stress and low productivity.

The main objective of the project is to develop a task and reminder management application aimed at students, professionals, and any user who wants to improve their daily organization in a simple and efficient way.

The methodology includes user needs analysis, design of an intuitive and accessible interface, and the development of a native Android application. The solution integrates features such as task creation, interactive calendar, automatic notifications, and cloud synchronization.

In conclusion, the project demonstrates that a well-designed digital tool can significantly improve personal organization, reduce mental load, and increase productivity. It also confirms the importance of a clear and adaptable design to ensure a good user experience.

Keywords: Task management, personal organization, cloud synchronization, reminders, digital productivity, Android, Kotlin.

Índice

1. Presentación del proyecto

1.1 Introducción

Memento es una aplicación de gestión de tareas y recordatorios, diseñada para ayudar a estudiantes, trabajadores y cualquier persona que necesite organizar su tiempo de manera eficiente. El objetivo principal es desarrollar una herramienta moderna, intuitiva y accesible que permita a los usuarios planificar su día a día, establecer recordatorios automáticos, recibir notificaciones y visualizar sus actividades mediante un calendario integrado.

El proyecto se aborda con una visión práctica, combinando funcionalidades clásicas de los gestores de tareas (creación, edición, eliminación, categorización) con características avanzadas como la sincronización en la nube, y las estadísticas de productividad.

La solución se ha materializado en una aplicación para dispositivos Android, desarrollada con Kotlin y XML en Android Studio. El backend y la persistencia de datos se apoyan en **Room** para el almacenamiento local y **Supabase** para la infraestructura en la nube, aprovechando su base de datos **PostgreSQL**, su sistema de autenticación y la sincronización en tiempo real. Esta memoria recoge todo el proceso de análisis, diseño, planificación y desarrollo, con el objetivo de convertir la idea inicial en un producto funcional y de calidad.

1.2 Contexto

En la sociedad actual, las personas están sometidas a un ritmo de vida acelerado, lleno de estímulos y responsabilidades constantes. La gestión del tiempo se ha convertido en uno de los retos más importantes, tanto en el ámbito académico como en el profesional y personal. Como estudiante del ciclo de Desarrollo de Aplicaciones Multiplataforma (**DAM**), he sido testigo directo de esta realidad: compaginar clases, proyectos, exámenes y la vida personal no siempre es fácil, y a menudo me he sentido desbordado por la cantidad de tareas pendientes.

Existen muchas aplicaciones de gestión de tareas en el mercado, pero al probarlas siempre encontraba algo que no terminaba de encajar: interfaces demasiado complejas, falta de sincronización entre dispositivos, ausencia de recordatorios efectivos o, simplemente, una sensación de que no estaban hechas a mi medida. Por eso decidí desarrollar mi propia herramienta.

Quería algo sencillo pero potente, que me ayudara a organizar mi día a día y que, al mismo tiempo, me permitiera aplicar todos los conocimientos adquiridos durante el ciclo. Así nace **Memento**: una aplicación pensada para que cualquier usuario pueda darle su toque personal —con categorías de colores, prioridades, filtros y temas— sin renunciar a funcionalidades modernas como la sincronización en la nube, las notificaciones automáticas o el calendario interactivo.

Este proyecto no solo cumple con los requisitos del proyecto final, sino que representa un reto personal: construir desde cero una herramienta funcional, hecha a mi medida, y que también pueda ayudar a otras personas a mejorar su organización diaria.

1.3 Justificación

La realización del proyecto **Memento** se justifica por varias razones. En primer lugar, desde una perspectiva social y personal, cada vez más personas —y yo mismo como estudiante— necesitan herramientas digitales que les ayuden a organizarse y a reducir la carga mental asociada a la gestión de múltiples obligaciones. La aplicación pretende dar respuesta a esta necesidad ofreciendo una interfaz limpia, accesible y adaptable a diferentes perfiles de usuario.

En segundo lugar, el proyecto tiene un valor formativo muy importante para mí. Me permite poner en práctica los conocimientos adquiridos durante el ciclo de **DAM**, especialmente en programación con Kotlin, diseño de bases de datos, integración con servicios como Supabase y uso de herramientas de control de versiones. También supone un reto personal aprender a

gestionar un proyecto completo, desde la idea inicial hasta la entrega final, asumiendo limitaciones de tiempo y conocimientos.

Finalmente, desde un punto de vista técnico y estratégico, **Memento** aborda un nicho de aplicaciones que pueden evolucionar hacia soluciones colaborativas e inteligentes. Su arquitectura modular y escalable permitirá futuras ampliaciones, como estadísticas de productividad, integraciones opcionales con Google Calendar o reconocimiento de voz, con el potencial de convertirse en una herramienta realmente útil para la comunidad.

1.4 Objetivos

Objetivo general

Desarrollar una aplicación móvil funcional, Memento, capaz de ayudar a las personas en su día a día facilitando la gestión del tiempo mediante una herramienta intuitiva de tareas y recordatorios.

Objetivos específicos

Los objetivos específicos se organizan en tres fases según su prioridad y el grado de finalización, destacando los hitos alcanzados y los que quedaron pendientes de implementar.

Fase 1 – Funcionalidades básicas (MVP)

Se ha conseguido implementar el núcleo esencial de la aplicación: autenticación de usuarios (registro, inicio y cierre de sesión), gestión completa de tareas (crear, editar, eliminar y cambiar estado), asignación de prioridades, fechas y horas, así como la creación de categorías personalizables con colores. También se incluyen filtros, búsqueda y edición del perfil de usuario. El único objetivo no alcanzado en esta fase fue el **cambio de tema claro/oscuro**.

Fase 2 – Mejoras (requisitos opcionales)

Se completaron con éxito varios objetivos avanzados: recuperación de contraseña por correo, sincronización en la nube entre dispositivos, notificaciones automáticas y calendario interactivo. Quedaron pendientes por falta de tiempo o complejidad las tareas recurrentes, la papelera (restauración) y la exportación a CSV.

Fase 3 – Funcionalidades avanzadas

Se logró implementar el objetivo de estadísticas de productividad. Los demás objetivos previstos (compartir listas con permisos, asignar tareas a otros usuarios, integración con Google Calendar y comandos de voz) no se completaron en esta versión del proyecto.

2. Estrategia y planificación

2.1 Estrategia de desarrollo y viabilidad

Para garantizar que **Memento** responde con precisión a las necesidades de las personas con dificultades para organizar su tiempo, el desarrollo se planteó bajo una arquitectura modular y personalizada. Esta decisión me permite aplicar los conocimientos adquiridos durante el ciclo formativo, mantener el control total sobre la arquitectura y el diseño, y adaptar la aplicación a las necesidades específicas que fui identificando sobre la marcha.

Cabe señalar que, desde el inicio, fui muy optimista con el alcance del proyecto. No todos los requisitos definidos se han completado, especialmente los de las fases 2 y 3. Sin embargo, la aplicación se ha diseñado para que sea perfectamente usable con solo las funcionalidades básicas. Las características no añadidas son extras que aportan valor, pero por falta de tiempo y conocimientos no se han podido completar. Durante el desarrollo hubo varios reinicios y cambios de planteamiento hasta llegar a la versión actual.

Viabilidad del proyecto

- **Viabilidad técnica:** Dispongo de las herramientas necesarias (Android Studio, Kotlin, Supabase, GitHub) y de los conocimientos básicos para abordar el desarrollo. Supabase proporciona autenticación, sincronización en la nube y persistencia sin necesidad de implementar un backend propio. Los reinicios iniciales sirvieron para aprender y afinar la solución.
- **Viabilidad económica:** Todas las herramientas utilizadas son gratuitas o tienen planes gratuitos suficientes. No hay costes de licencias ni de infraestructura.
- **Viabilidad temporal:** El proyecto se planificó en tres fases (MVP, mejoras y avanzadas) ajustándose al calendario del curso. Las funcionalidades no implementadas se han pospuesto por falta de tiempo o complejidad técnica, pero la planificación priorizó un producto funcional y usable.

2.2 Metodología de trabajo

Para el desarrollo de **Memento** he optado por una metodología ágil adaptada al trabajo individual, basada en los principios de Scrum pero simplificada a mis necesidades como desarrollador en solitario. En lugar de seguir ceremonias formales (daily meetings, retrospectivas, etc.), me centré en organizar el trabajo mediante retos y funcionalidades concretas que iba completando poco a poco, con un enfoque iterativo y orientado a resultados.

Esta metodología me ha permitido:

- Organizar el trabajo en pequeños ciclos, donde cada ciclo podría abordaba varias funcionalidades específicas.
- Adaptarme rápidamente a imprevistos y cambios, algo especialmente necesario dado que hubo varios reinicios y cambios de planteamiento hasta llegar a la versión actual de la app.
- Priorizar sin piedad: primero el producto mínimo viable (MVP), luego mejoras, y al final las funcionalidades avanzadas (muchas de las cuales quedaron fuera por tiempo o complejidad).

Herramientas de seguimiento

Para gestionar el día a día utilicé **Trello** como tablero Kanban, con cuatro estados que reflejaban el flujo de trabajo de cada tarea o reto:

Estado	Significado
Pendientes	Tareas o funcionalidades identificadas pero aún no iniciadas. Aquí acumulaba todas las ideas y requisitos.
En progreso	Tarea en la que estaba trabajando activamente. Intentaba tener solo una o dos a la vez para no dispersarme.
Revisión	Funcionalidad implementada pero pendiente de pruebas, depuración o validación. A menudo volvía a *En progreso* si encontraba errores.
Completado	Tarea finalizada, probada y funcionando correctamente. Un pequeño logro en el camino.

Cada tarjeta de Trello incluía una descripción de la funcionalidad, etiquetas de prioridad (alta, media, baja) y la fase del proyecto a la que pertenecía (MVP, mejora o avanzada). Este sistema visual me ayudó a mantener el rumbo y a no olvidar nada, aunque a veces la columna *Pendientes* creciera más de lo deseable.

Además de Trello, utilicé **GitHub** para el control de versiones. Trabajé con una rama principal (**main**). Los commits los realicé de forma regular, aunque al principio eran algo caóticos; con el tiempo fui mejorando. No utilicé un diagrama de Gantt rígido, pero sí mantuve una planificación general en una hoja de cálculo para tener una visión global de los plazos.

2.3 Planificación

El desarrollo de **Memento** se organizó en varias fases, con una estimación para todo el curso. Reconozco que fui demasiado optimista al planificar, pero la estructura por fases me ayudó a mantener el rumbo. A continuación se detalla el cronograma previsto y cómo se desarrolló realmente, teniendo en cuenta los reinicios, los cambios de planteamiento y el optimismo inicial.

Fase	Descripción	Duración estimada
Planificación y análisis	Estudio de requisitos, definición de alcance, análisis de riesgos, planificación inicial.	2 semanas
Desarrollo MVP (Fase 1)	Implementación de funcionalidades básicas: registro, login, gestión de tareas, categorías, filtros, búsqueda, edición de perfil.	6 semanas
Desarrollo mejoras (Fase 2)	Implementación de funcionalidades opcionales: recuperación de contraseña, sincronización en la nube, calendario interactivo. (Otras mejoras como papelera, exportación CSV no se completaron).	6 semanas
Desarrollo avanzado (Fase 3)	Implementación de funcionalidades avanzadas: como estadísticas se lograron y otras opcionales no se completaron como compartir listas, permisos, reconocimiento de voz, IA, integración con Google Calendar.	4 semanas
Pruebas y correcciones	Pruebas funcionales, de usabilidad, de rendimiento, corrección de errores.	2 semanas
Documentación y memoria	Redacción de la memoria final, manual de usuario, preparación de anexos.	2 semanas

Nota final: A pesar de los reinicios y de no haber completado todas las funcionalidades previstas, la planificación por fases me permitió priorizar lo esencial. El resultado es una aplicación usable que cumple con las funcionalidades básicas y algunas mejoras, dejando las características más avanzadas como posibles ampliaciones futuras.

3. Análisis

3.1 Casos de uso

A continuación se describen las principales interacciones que un usuario puede realizar con **Memento**. Los casos de uso reflejan las funcionalidades realmente implementadas en la versión actual de la aplicación.

Código	Nombre del caso de uso	Actor principal	Descripción	Resultado esperado
CU1	Registrar usuario	Usuario	El usuario introduce su correo electrónico y contraseña para crear una cuenta en Memento.	Cuenta creada correctamente y datos guardados en la nube.
CU2	Iniciar sesión	Usuario	El usuario introduce su correo y contraseña para acceder a la aplicación.	Acceso concedido y sincronización de datos con la nube realizada.
CU3	Crear tarea	Usuario	El usuario añade una nueva tarea, asignando título, descripción, fecha, hora, prioridad y categoría.	Tarea registrada correctamente, visible en la lista de tareas y en el calendario interactivo.
CU4	Editar o eliminar tarea	Usuario	El usuario selecciona una tarea existente y la modifica o elimina según sea necesario.	Cambios reflejados en la lista de tareas, en el calendario y sincronizados en la nube.
CU5	Recibir recordatorio	Usuario	El usuario recibe una notificación automática en la fecha y hora programada para una tarea.	Usuario notificado correctamente y recordatorio cumplido.
CU6	Consultar historial y progreso	Usuario	El usuario revisa tareas completadas, pendientes y estadísticas de productividad.	Visualización clara del historial y métricas de productividad actualizadas.
CU7	Recuperar contraseña	Usuario	El usuario solicita un enlace de recuperación enviado a su correo electrónico.	El usuario recibe un enlace para restablecer su contraseña de forma segura.
CU8	Sincronizar datos	Usuario	El sistema sincroniza automáticamente las tareas y el	Los datos están actualizados y

			perfil entre todos los dispositivos del usuario.	accesibles desde cualquier dispositivo.

3.2 Requisitos funcionales

A continuación se detallan las funcionalidades que Memento ofrece a sus usuarios, las marcadas con **“!”** no estan implementadas.

Código	Descripción del requisito funcional
RF1	El sistema permitirá registrar nuevos usuarios con correo electrónico y contraseña segura.
RF2	Los usuarios podrán iniciar sesión con su correo y contraseña registrados.
RF3	El sistema permitirá cerrar sesión en cualquier dispositivo.
RF4	Los usuarios podrán editar su perfil básico (nombre y foto de perfil).
RF5	El sistema permitirá crear nuevas tareas con: título, descripción, fecha límite, hora y prioridad.
RF6	Los usuarios podrán editar cualquier información de las tareas creadas.
RF7	Los usuarios podrán eliminar tareas (con confirmación simple).
RF8	Las tareas tendrán 3 estados posibles: "Pendiente", "En Progreso" y "Completada".
RF9	El sistema mostrará las tareas en vista de lista ordenable por orden de creación.
RF10	Los usuarios podrán filtrar tareas por categoría, prioridad o estado.

RF11	Los usuarios podrán ordenar tareas por 3 niveles de prioridad.
RF12	Los usuarios podrán buscar tareas por palabras clave en título o descripción.
RF13	Los usuarios podrán crear, editar y eliminar categorías personalizadas.
RF14	Las categorías tendrán colores asignables para identificación visual.
RF15	Los usuarios podrán asignar categorías a las tareas.
RF16!	Los usuarios podrán cambiar entre tema claro y oscuro.
REQUISITOS OPCIONALES (Fase 2)	
RF17	Los usuarios podrán recuperar su contraseña mediante enlace enviado al correo electrónico.
RF18!	El sistema permitirá crear tareas recurrentes (diarias, semanales, mensuales).
RF19	Los usuarios tendrán acceso a un calendario interactivo con vista mensual.
RF20!	El sistema permitirá restaurar tareas eliminadas desde la papelera (hasta 30 días).
RF21!	Los usuarios podrán exportar sus tareas a formato CSV.
RF22	El sistema enviará notificaciones automáticas según la fecha/hora de la tarea.
RF23	Las tareas se sincronizarán entre todos los dispositivos del usuario.
RF24	El sistema realizará sincronización en la nube.
FUNCIONALIDADES AVANZADAS (FASE 3)	

RF25!	Los usuarios podrán compartir listas de tareas con otros usuarios
RF26!	El sistema permitirá permisos diferenciados: "Solo lectura" o "Lectura y escritura".
RF27!	Los usuarios podrán asignar tareas a otros usuarios en proyectos compartidos.
RF28!	El sistema se integrará con Google Calendar para importar eventos.
RF29!	Los usuarios podrán crear tareas mediante comandos de voz.
RF30	El sistema mostrará estadísticas de productividad.

3.3 Requisitos no funcionales

Los requisitos no funcionales definen cómo debe comportarse el sistema en aspectos como rendimiento, seguridad, usabilidad y fiabilidad.

Código	**Descripción del requisito no funcional**
RNF1	La App será accesible desde dispositivos móviles Android.
RNF2	Las páginas deberán cargarse en menos de tres segundos.
RNF3	Los datos de usuarios y tareas estarán cifrados y seguros (tanto en tránsito como en reposo).
RNF4	La aplicación debe ser intuitiva y fácil de usar, con navegación clara.
RNF5	Las notificaciones deberán entregarse de forma fiable y oportuna según el usuario lo haya programado.

RNF6	El backend (Supabase) no debe sufrir degradación del rendimiento con hasta 100 usuarios simultáneos.
RNF7	La App funcionará correctamente sin conexión a internet y sincronizará los cambios al recuperar la señal.

Nota: Al igual que en los requisitos funcionales, se entiende que estos requisitos no funcionales son los esperados. En la implementación real, algunos aspectos (como la velocidad de carga en redes muy lentas o la fiabilidad extrema de las notificaciones) pueden verse afectados por las limitaciones del entorno de desarrollo y las herramientas gratuitas utilizadas.

3.4 Análisis de alternativas tecnológicas

(Explica qué opciones tecnológicas has considerado y por qué has elegido la solución final.)

Lenguaje de programación

Alternativa	Ventajas	Inconvenientes
Java	Lenguaje maduro, amplia comunidad, conocido en el ciclo DAM.	Código más verboso, menos seguro (null safety), menor productividad en ciertos aspectos.
Kotlin	Lenguaje oficial para Android, moderno, conciso, seguro (null safety), interoperable con Java.	Curva de aprendizaje inicial si no se conoce Java.

Justificación: Se eligió **Kotlin** porque es el lenguaje recomendado por Google para el desarrollo Android actual. Permite escribir código más limpio, seguro y con menos errores. Además, facilita la integración con las últimas librerías y herramientas.

Backend y base de datos

Alternativa	Ventajas	Inconvenientes
Firebase	Popular, autenticación sencilla,	Base de datos NoSQL

	base de datos en tiempo real, notificaciones push integradas.	(Firestore) que no se ajusta al modelo relacional del proyecto; costes elevados al escalar.
Backend propio (Spring Boot + PostgreSQL)	Control total sobre la API y la base de datos, arquitectura escalable.	Requiere desarrollar y mantener el backend completo, mayor complejidad y tiempo de desarrollo.
Supabase	Backend como servicio (BaaS) con PostgreSQL relacional, autenticación, almacenamiento, sincronización en tiempo real y plan gratuito generoso.	Dependencia de un servicio externo; curva de aprendizaje para las políticas de seguridad (RLS).

Justificación: Se eligió **Supabase** porque ofrece una base de datos **PostgreSQL relacional**, ideal para el modelo de datos de Memento (usuarios, tareas, categorías, comparticiones). Además, incluye autenticación y sincronización en tiempo real sin necesidad de implementar un backend propio, lo que ahorra tiempo y complejidad. Su plan gratuito cubre las necesidades del proyecto.

Entorno de desarrollo (IDE)

Alternativa	Ventajas	Inconvenientes
IntelliJ IDEA	Potente para Kotlin, buena refactorización.	La versión gratuita (Community) no tiene soporte completo para desarrollo Android.
Visual Studio Code	Ligero, extensible mediante plugins.	Requiere configurar muchos complementos; no ofrece todas las herramientas integradas para Android.
Android Studio	IDE oficial para Android, integración completa con Gradle, emuladores, depuración, soporte nativo para Kotlin y XML.	Consume muchos recursos; puede ser lento en equipos modestos.

Justificación: Android Studio es el IDE estándar para el desarrollo Android. Ofrece herramientas específicas (diseño de layouts, depuración, emuladores, perfilado de rendimiento) que facilitan el trabajo. Al ser el oficial, cuenta con amplia documentación y soporte de la comunidad.

Control de versiones y seguimiento

Alternativa	Ventajas	Inconvenientes
GitLab	Pipelines CI/CD integradas, repositorios privados gratuitos.	Menos popular en entornos educativos.
Trello	Tablero Kanban visual, fácil de usar, gratuito.	No integra directamente con el código.
GitHub	Control de versiones estándar, repositorios privados gratuitos, integración con proyectos, amplia comunidad.	Curva de aprendizaje para comandos avanzados.

Justificación: Se ha utilizado **GitHub** para el control de versiones por ser el estándar en la industria, permitir trabajar con ramas y facilitar la colaboración (aunque el proyecto sea individual). Para el seguimiento de tareas se eligió **Trello** por su sencillez y su sistema de columnas (Pendientes, En progreso, Revisión, Completado), que se adapta perfectamente al flujo de trabajo ágil individual.

4. Diseño

4.1 Arquitectura del sistema

MementoApp está construida bajo los principios de **Clean Architecture** y el patrón **MVVM (Model-View-ViewModel)**, lo que garantiza un código modular, testeable y fácil de mantener. La organización es la siguiente:

- **Capa de presentación (UI):**
    - Se utiliza una arquitectura de **Single Activity** con múltiples fragmentos coordinados por el **Jetpack Navigation Component**. Esto permite una navegación fluida y consistente.
    - **ViewBinding** sustituye al antiguo **findViewById**, generando clases que vinculan directamente el código Kotlin con el XML, evitando errores de puntero nulo.
    - La lógica de creación de tareas está centralizada en **TaskDialogHelper.kt**, que utiliza **Material 3 Dialogs**. Esto permite invocar la creación de tareas desde cualquier pantalla (Home, Calendario, Ajustes) sin duplicar código.
    - Los fragmentos implementan una interfaz común (GlobalActionProvider) para reaccionar al clic del botón de acción flotante (FAB) central de la MainActivity.
- **Capa de negocio (ViewModel):**
    - Los ViewModels gestionan el estado de la UI mediante **StateFlow** (Kotlin Flow), asegurando que los datos se mantengan sincronizados entre las vistas y la base de datos.
    - **Dagger Hilt** se utiliza para la inyección de dependencias, facilitando la provisión de repositorios y casos de uso de forma limpia y escalable.
- **Capa de datos (Repository Pattern):**
    - **Supabase** actúa como backend principal, proporcionando autenticación y una base de datos **PostgreSQL** en tiempo real.
    - **Room** se emplea para la persistencia local, permitiendo que la aplicación funcione en **modo offline** y sincronice los cambios al recuperar la conexión.

4.2 Modelo de datos

El sistema se basa en un esquema relacional diseñado para la eficiencia y la seguridad. Las principales entidades son:

- **Entidad Task (Tareas):**
    - **id:** Identificador único (UUID).
    - **title / description:** Información textual de la tarea.

    - **priority:** Tipo enumerado (LOW, MEDIUM, HIGH) para clasificación visual.
    - **status:** Estado de progreso (PENDING, IN_PROGRESS, COMPLETED).
    - **dueDate:** Fecha de vencimiento almacenada como Timestamp.
    - **categoryId:** Relación (clave foránea) con la tabla de categorías.
    - **userId:** Vinculación con el sistema de autenticación de Supabase.
- **Entidad Category (Categorías):**
    - **id:** Identificador único.
    - **name:** Nombre de la categoría.
    - **color / icon:** Atributos visuales para la personalización de la interfaz.
- **Seguridad:**
    - Se aplican políticas de **RLS (Row Level Security)** en Supabase, asegurando que cada usuario solo pueda acceder a sus propios registros.
    - Además, en la capa local (Room) se filtran los datos por userId para mantener la coherencia.

4.3 Diseño de interfaz

El diseño sigue las líneas de **Material Design 3 (Material You)**, priorizando una experiencia de usuario limpia, directa y moderna.

- **Navegación intuitiva:**
    - Una **BottomNavigationView** centraliza el acceso a las funciones principales (Home, Calendario, Estadísticas, Ajustes).
    - Se utiliza **Z-axis layering**: componentes como la barra de navegación y el FAB tienen elevaciones específicas (elevation) para indicar su jerarquía sobre el contenido en scroll.
- **Experiencia de usuario (UX):**
    - **Scroll inmersivo:** El uso de clipToPadding="false" en los listados permite que las tareas fluyan bajo la barra de navegación, eliminando barreras visuales y aprovechando el 100% de la pantalla.

    - **Feedback inmediato:** Se utiliza **SwipeRefreshLayout** para actualizaciones manuales y **Material Pickers (Date & Time)** para una entrada de datos amigable y moderna.
    - **Accesibilidad:** Los estados de carga se gestionan con **ProgressBars**, y se muestran **estados vacíos (EmptyStates)** que guían al usuario cuando no hay tareas registradas.
    - **Notificaciones:** Se emplea **AlarmManager** para que los recordatorios suenen exactamente cuando deben, incluso si la app está cerrada.
- **Componentes destacados:**
    - **MaterialCardView** y **Chips** para filtros visuales.
    - **FloatingActionButton (FAB)** central para la creación rápida de tareas.
    - **ViewBinding y Navigation Component** como base técnica de la interfaz.

5. Desarrollo

5.1 Estructura del proyecto

El código está organizado siguiendo los principios de Arquitectura Limpia (**Clean Architecture**) y modularización por capas, lo que facilita la escalabilidad del sistema.

A continuación se describen las carpetas principales:

- **presentation/:** Capa de interfaz de usuario. Está subdividida por módulos funcionales (**home**, **calendar**, **stats**, **settings**, **auth**). Cada módulo contiene sus Fragments (UI) y sus respectivos **ViewModels**. Se utiliza Jetpack Navigation Component para la navegación entre pantallas y ViewBinding para enlazar el XML con el código Kotlin de forma segura y eficiente.
- **domain/:** Es el corazón de la aplicación. Contiene las entidades de negocio puras (**Task**, **Category**) y las interfaces de los repositorios (**TaskRepository**,

CategoryRepository). Además, incluye casos de uso (Use Cases) como **GetTasksUseCase**, que encapsulan la lógica de negocio reutilizable (por ejemplo, filtrar tareas por estado o categoría). Esta capa es independiente de cualquier framework externo.

- **data/:** Implementa la lógica de acceso a datos. Aquí se encuentran las implementaciones concretas de los repositorios (ej. **CategoryRepositoryImpl**), que deciden si obtener la información de Supabase (remoto) o de Room (base de datos local). También se incluyen los DAOs de Room y los DTOs para la serialización.
- **utils/:** Clases de soporte transversales. Destacan **TaskDialogHelper** (encapsula la creación de tareas con **MaterialDatePicker** y **MaterialTimePicker**), **NotificationHelper** y **NotificationReceiver** (gestionan las notificaciones con **AlarmManager**).
- **res/:** Recursos organizados sistemáticamente. Los **layouts** siguen el estándar de nombres (**fragment_**, **item_**, **dialog_***). Los menús definen la navegación inferior (**BottomNavigationView**) y las opciones de la barra de herramientas.

5.2 Implementación de funcionalidades

A continuación se explica cómo se han implementado los requisitos funcionales (RF) realmente completados, agrupados por áreas. Se ha prestado especial atención a la **sincronización offline-first**, la **reactividad** y la **experiencia de usuario**

Autenticación y perfil (RF1 a RF4, RF17)

- **Registro e inicio de sesión**: Se utiliza **Supabase Auth** con correo electrónico y contraseña. Las llamadas se realizan desde un repositorio que expone el estado de autenticación mediante **StateFlow**.
- **Cierre de sesión**: Se invoca **signOut()** y se limpia el estado local.
- **Edición de perfil**: Permite cambiar nombre y foto (almacenada en Supabase Storage).
- **Recuperación de contraseña**: Se usa **resetPasswordForEmail()** (Supabase) para enviar un enlace mágico al correo.

Gestión de tareas (RF5 a RF12)

- **CRUD completo**: Las tareas se representan con la entidad **Task**. El **TaskRepository** combina Room (fuente local) y Supabase (remota).

- **Estados y prioridades**: Los campos **status** (Pendiente, En Progreso, Completada) y **priority** (Baja, Media, Alta) son enumerados. La interfaz permite cambiarlos desde un menú contextual en cada elemento de la lista.
- **Vista de lista ordenable**: El **RecyclerView** muestra las tareas ordenadas por fecha de creación. También se puede ordenar por prioridad.
- **Filtros y búsqueda**: En **HomeFragment** se combina un **SearchView** con un **ChipGroup**. Al escribir o seleccionar un chip, el ViewModel aplica operadores de filtrado sobre el **StateFlow** de tareas, actualizando la UI instantáneamente (reactividad pura).

Categorías personalizables (RF13 a RF15)

- **Sincronización híbrida**: **CategoryRepositoryImpl** expone un **Flow<List<Category>>** (Room como fuente de verdad). Los cambios en la base de datos local se reflejan en la UI sin recarga manual.
- **Sincronización bidireccional**: **fetchCategoriesFromRemote** descarga las categorías de Supabase y las guarda en Room. **createCategory / updateCategory** primero modifican Room y luego intentan replicar el cambio en Supabase.
- **Contador de tareas (optimización)**: Cada categoría tiene un campo **taskCount**. Al crear o eliminar una tarea, se incrementa o decrementa automáticamente, evitando consultas **COUNT(*)** costosas.
- **Categorías por defecto**: En **createDefaultCategoriesIfNeeded**, si un usuario es nuevo, la app crea automáticamente un conjunto inicial (Trabajo, Personal, Compras, etc.) para que no empiece con la pantalla vacía.
- **Colores**: Se asigna un color a cada categoría (selector predefinido).

Calendario interactivo (RF19)

- Pantalla con un **CalendarView** adaptado a Material Design 3. Cada día muestra indicadores circulares si hay tareas pendientes. Al hacer clic en una fecha, se abre el listado de tareas de ese día. La navegación es fluida gracias al **Navigation Component**.

Notificaciones automáticas (RF22)

- **AlarmManager + BroadcastReceiver**: **NotificationScheduler** agenda la notificación en la fecha/hora exacta. Cuando se dispara, **NotificationReceiver** construye y muestra la notificación usando **NotificationManagerCompat**. Funciona incluso con la app en segundo plano o cerrada.
- **Canales de notificación**: Se crean los canales obligatorios para Android 8+ (NotificationHelper).

Sincronización en la nube y modo offline (RF23, RF24)

- **Estrategia offline-first**: El repositorio persiste primero en Room y, si hay conexión, envía los cambios a Supabase mediante PostgREST.
- **Flujos reactivos**: **getCategoriesStream** y **getTasksStream** emiten **Flow**; la UI se suscribe a ellos y se actualiza automáticamente.
- **Aislamiento multiusuario**: Cada usuario solo ve sus propios datos gracias a las políticas **RLS (Row Level Security)** configuradas en Supabase (filtrado por **userId**).
- **Sincronización al recuperar conexión**: Un worker (WorkManager) se encarga de sincronizar las operaciones pendientes cuando se restablece la red.

Estadísticas de productividad (RF30)

- Pantalla **StatsFragment** que muestra un resumen estadístico actual del usuario: número total de tareas, desglose por estado (Pendiente, En progreso, Completado) y distribución por prioridad (Alta, Media, Baja). Los datos se calculan a partir del flujo de tareas del repositorio en tiempo real.

Estrategias transversales de implementación

- **Inyección de dependencias con Hilt**: Se usan anotaciones **@Singleton**, **@Inject** y módulos para proporcionar una única instancia de repositorios, clientes de Supabase y DAOs de Room. Esto ahorra memoria y facilita las pruebas.
- **Centralización de la creación de tareas**: **TaskDialogHelper** encapsula los pickers de fecha/hora y la lógica de validación. Puede invocarse desde cualquier fragmento (home, calendario, estadísticas) sin duplicar código.

- **Delegación de acción global**: Se definió la interfaz **GlobalActionProvider**. La **MainActivity** no sabe qué hace el FAB; simplemente notifica al fragmento activo, que ejecuta la acción correspondiente (por ejemplo, mostrar el diálogo de nueva tarea).
- **Corrutinas y Flow**: Toda la asincronía (red, base de datos) se maneja con corrutinas de Kotlin, garantizando que el hilo principal no se bloquee.

5.3 Pruebas

Para garantizar el correcto funcionamiento de **Memento** y ofrecer una experiencia fiable al usuario, se llevaron a cabo diferentes tipos de pruebas durante el desarrollo. Dado que se trata de un proyecto académico, las pruebas se enfocaron en validar las funcionalidades clave desde el punto de vista del usuario final

Pruebas funcionales manuales

Se verificó una por una todas las funcionalidades implementadas, asegurando que cumplen con los requisitos (RF):

- **Autenticación (RF1, RF2, RF3, RF17)**: Registro, inicio de sesión, cierre y recuperación de contraseña. Todo funciona correctamente.
- **Gestión de tareas (RF5 a RF12)**: Creación, edición, eliminación y cambio de estado. Las operaciones se reflejan en la UI y en Room.
- **Categorías (RF13 a RF15)**: Creación, edición, asignación de colores, categorías por defecto y filtrado.
- **Filtros y búsqueda (RF10, RF12)**: Probados exhaustivamente, devuelven los resultados esperados en tiempo real.
- **Calendario (RF19)**: Las tareas aparecen en las fechas correctas, se puede navegar entre meses.
- **Notificaciones (RF22)**: Se programaron tareas con fecha/hora y se confirmó que la notificación se muestra incluso con la app en segundo plano.
- **Sincronización en la nube y offline (RF23, RF24)**: Cambios realizados sin conexión: se guardan localmente y, al recuperar la red, se sincronizan automáticamente. También

se probó que los datos se reflejan en otro dispositivo al iniciar sesión con la misma cuenta.

- **Estadísticas (RF30)**: Los gráficos se actualizan correctamente según las tareas completadas/pendientes.

Pruebas de navegación y usabilidad

Se evaluó la fluidez y la intuición de la interfaz:

- La navegación entre secciones (Home, Calendario, Estadísticas, Ajustes) es rápida y sin errores.
- Se corrigió un problema de superposición visual (espacio fantasma) en el listado de tareas y se ajustó la elevación del FAB para cumplir con Material Design 3.
- Se verificó el comportamiento en “lista vacía” (mensajes claros).
- Se probó la app en el **emulador Pixel 9 Pro** y en **emulador Pixel 9 XL** confirmando que la apariencia y el rendimiento son correctos en diferentes tamaños de pantalla.

Pruebas de casos extremos

- **Sin conexión a internet**: La app permite crear/editar tareas; al recuperar la señal, los datos se sincronizan sin intervención manual.
- **Fechas de vencimiento pasadas**: Se muestran correctamente sin errores.
- **Uso prolongado**: Ciclos intensivos de creación/eliminación de tareas para verificar estabilidad y consumo de recursos.

Resultados generales

El uso de **Hilt**, **Room**, **Supabase** y las **corrutinas** ha permitido construir una arquitectura robusta y preparada para futuras ampliaciones. Las funcionalidades no implementadas (tema claro/oscuro, tareas recurrentes, papelera, exportación CSV, compartición avanzada,

etc.) quedan identificadas como trabajo futuro.

6. Conclusiones

6.1 Conclusiones generales

El desarrollo de **Memento** ha sido un proyecto ambicioso que partió de la necesidad real de disponer de una herramienta sencilla pero potente para la gestión de tareas y recordatorios. A lo largo del proceso se ha construido una aplicación Android funcional, estable y utilizable, que permite a los usuarios organizar su día a día mediante tareas, categorías personalizables, filtros, sincronización en la nube.

A pesar de que el alcance inicial era muy optimista y se planificaron numerosas funcionalidades (calendario interactivo, notificaciones automáticas, papelera, exportación CSV, compartición avanzada, etc.), el proyecto ha sabido priorizar y entregar un producto mínimo viable de calidad. Las funcionalidades no implementadas quedan identificadas como posibles ampliaciones futuras, sin que ello afecte al cumplimiento del objetivo principal: ofrecer una aplicación de gestión de tareas estable, sincronizada y fácil de usar.

Desde el punto de vista técnico, **Memento** ha seguido los estándares profesionales de Android: Kotlin, Clean Architecture, MVVM, Hilt para inyección de dependencias, Jetpack Navigation, Room para persistencia local y Supabase como backend. El resultado es una aplicación moderna, mantenible y escalable.

6.2 Consecución de objetivos

Tal como se detalló en el apartado 1.4, los objetivos se organizaron en tres fases. A continuación se analiza el grado de cumplimiento:

- **Fase 1 – Funcionalidades básicas:**
  Completada casi en su totalidad. Todas las funciones esenciales (registro, login,

gestión completa de tareas, categorías, filtrado, búsqueda, edición de perfil) están implementadas. Únicamente el cambio de tema claro/oscuro quedó pendiente.

- **Fase 2 – Mejoras (requisitos opcionales):**
  Se lograron implementar la recuperación de contraseña por correo, la sincronización en la nube, las notificaciones automáticas y el calendario interactivo. Quedaron pendientes las tareas recurrentes, la papelera y la exportación a CSV.
- **Fase 3 – Funcionalidades avanzadas:**
  Cumplida mínimamente. Se implementaron las estadísticas de productividad, pero no se lograron completar la compartición de listas, los permisos diferenciados, la asignación de tareas a otros usuarios, la integración con Google Calendar ni los comandos de voz.

Por tanto, el objetivo general (desarrollar una aplicación móvil funcional para ayudar a las personas en la gestión del tiempo) se ha cumplido satisfactoriamente. Se ha entregado una herramienta usable y sincronizada que cumple las expectativas básicas y algunas mejoras adicionales. Las funcionalidades no alcanzadas eran precisamente las más complejas o las que requerían más tiempo, pero no impiden el uso diario de la aplicación.

6.3 Valoración de la metodología y planificación

La metodología ágil adaptada al trabajo individual ha demostrado ser adecuada para el proyecto. El uso de **Trello** con los cuatro estados (Pendientes, En progreso, Revisión, Completado) permitió visualizar el avance de forma clara y priorizar tareas en cada sprint. **GitHub** proporcionó el control de versiones necesario para experimentar sin miedo a perder trabajo.

Sin embargo, la planificación inicial fue excesivamente optimista y se asumió que se podrían completar las tres fases. La realidad mostró que los reinicios, los cambios de planteamiento, la curva de aprendizaje de algunas herramientas (especialmente **Supabase** y sus políticas de seguridad RLS) y la complejidad de ciertas funcionalidades (notificaciones, sincronización *offline-first*) consumieron más tiempo del previsto. Aun así, la planificación por fases

permitió priorizar el **MVP** y entregar un producto funcional, dejando las características más avanzadas para después.

En retrospectiva, la planificación debería haber sido más conservadora y haber reservado un margen mayor para imprevistos. Pero la flexibilidad de la metodología ágil permitió reajustar el rumbo sin frustraciones excesivas.

6.4 Visión de futuro

A pesar de que no se implementaron todas las funcionalidades planificadas, **Memento** tiene un amplio margen de mejora. Estas son las líneas de desarrollo futuro que se podrían abordar en próximas versiones:

- **Completar la Fase 2:** Añadir tareas recurrentes, papelera con restauración y exportación a CSV.
- **Implementar la Fase 3:** Compartir listas con permisos (lectura/escritura), asignar tareas a otros usuarios, integración con Google Calendar y comandos de voz.
- **Mejorar la experiencia offline:** Optimizar la sincronización en entornos de red inestable y reducir el consumo de batería.
- **Añadir widget para la pantalla de inicio:** Permitir ver y marcar tareas sin abrir la aplicación.
- **Soporte para temas dinámicos (Material You):** Adaptar la interfaz a los colores del sistema en Android 12+.
- **Mejorar las estadísticas:** Incluir gráficos más detallados y predicciones de productividad utilizando inteligencia artificial (uno de los objetivos iniciales no alcanzados).
- **Publicar en Google Play:** Una vez madurada la aplicación, prepararla para su distribución comercial.

En definitiva, **Memento** es un proyecto vivo que puede seguir creciendo. Lo aprendido durante este desarrollo (técnicamente y en gestión) sienta una base sólida para futuras iteraciones, ya sea como trabajo personal o como posible producto real.

7. Glosario

API (Application Programming Interface): Conjunto de reglas que permite la comunicación entre aplicaciones.

BaaS (Backend as a Service): Servicio en la nube que proporciona backend (base de datos, autenticación) sin gestionar servidores.

Clean Architecture: Arquitectura de software que separa el código en capas independientes (presentación, dominio, datos).

CSV (Comma-Separated Values): Formato de archivo de texto plano para datos tabulares.

DAO (Data Access Object): Objeto que encapsula el acceso a la base de datos.

DTO (Data Transfer Object): Objeto para transportar datos entre capas.

FAB (Floating Action Button): Botón circular flotante de Material Design para acciones principales.

Hilt: Biblioteca de inyección de dependencias para Android (basada en Dagger).

Jetpack Navigation Component: Biblioteca para gestionar la navegación entre pantallas.

Kotlin: Lenguaje de programación oficial para Android.

Material Design 3 (Material You): Sistema de diseño de Google.

MockK: Biblioteca de mocking para pruebas unitarias en Kotlin.

MVVM (Model-View-ViewModel): Patrón arquitectónico que separa interfaz, lógica y datos.

PostgreSQL: Sistema de base de datos relacional utilizado en Supabase.

RLS (Row Level Security): Política de seguridad que filtra filas según el usuario autenticado.

Room: Biblioteca de persistencia local en Android (abstracción sobre SQLite).

StateFlow: Flujo reactivo de Kotlin para emitir estados observables.

Supabase: Plataforma BaaS de código abierto basada en PostgreSQL.

ViewBinding: Generación de enlaces seguros entre XML y código Kotlin.

8. Bibliografía

Documentación oficial:

1. Google Developers (2025). *Documentación de Android*. developer.android.com
2. JetBrains (2025). *Documentación de Kotlin*. kotlinlang.org
3. Supabase (2025). *Documentación oficial*. supabase.io/docs
4. Material Design (2025). *Material Design 3*. m3.material.io

Librerías y herramientas:

5. Dagger Hilt (2025). *Guía de inyección de dependencias*. dagger.dev/hilt
6. Room Persistence Library (2025). *Guía de persistencia*. developer.android.com/training/data-storage/room
7. Jetpack Navigation (2025). *Guía de navegación*. developer.android.com/guide/navigation

Recursos académicos:

8. Apuntes del ciclo DAM (2024-2025). Instituto Puig Castellar. Módulos: Programación (Kotlin), Bases de Datos, Desarrollo de Interfaces.
9. Google Codelabs (2025). *Android Basics in Kotlin*. developer.android.com/codelabs

Herramientas de desarrollo:

10. GitHub (2025). *Documentación de Git*. docs.github.com
11. Trello (2025). *Guía de Kanban*. trello.com/guide

9. Anexos

(Incluye material adicional como capturas, código, diagramas, etc.)

Anexo A: Estado de implementación de requisitos

A continuación se muestra una visión general del grado de cumplimiento de los requisitos funcionales previstos para **Memento**. La mayoría de las funcionalidades esenciales están operativas, y aquellas no incluidas en la versión actual quedan como posibles mejoras futuras.

Área	Estado
Gestión de usuarios (registro, login, perfiles)	✅ Completado
Gestión completa de tareas (CRUD, estados, prioridades)	✅ Completado
Categorías personalizables con colores	✅ Completado
Filtrado, ordenación y búsqueda de tareas	✅ Completado
Calendario interactivo y notificaciones automáticas	✅ Completado
Sincronización en la nube	✅ Completado
Estadísticas de productividad	✅ Completado
Recuperación de contraseña por correo	✅ Completado
Cambio de tema claro/oscuro	Pendiente (no implementado)
Tareas recurrentes, papelera, exportación CSV	Pendientes (no implementados)
Funcionalidades colaborativas e integraciones externas	Pendientes (futuras ampliaciones)

Nota: El proyecto cumple con su objetivo principal: ofrecer una herramienta usable y sincronizada para la gestión diaria de tareas. Las funcionalidades pendientes no afectan al uso básico de la aplicación.

Anexo B: Uso de inteligencia artificial y ayudas externas

Durante el desarrollo de **Memento** y la redacción de esta memoria, se han utilizado diversas fuentes de apoyo:

- **Inteligencia artificial (ChatGPT, modelo GPT-4):** Se ha empleado como herramienta de ayuda para:
    - Generar borradores iniciales de algunos apartados de la memoria.
    - Mejorar la redacción, coherencia y estructura del documento.
    - Resolver **errores puntuales en el código** (por ejemplo, problemas con la configuración de Supabase, políticas RLS, gestión de StateFlow, o la correcta implementación de notificaciones con AlarmManager). En cada caso, se ha entendido y adaptado la solución antes de integrarla.
    - Obtener sugerencias sobre arquitectura (Clean Architecture + MVVM) y buenas prácticas en Kotlin.
- **Foros y sitios web especializados:** Se han consultado recursos como **Stack Overflow**, **GitHub Issues**, la documentación oficial de **Supabase**, **Android Developers** y tutoriales de **Material Design 3**. Estas referencias han sido fundamentales para resolver problemas técnicos concretos.

  Todo el código final, la arquitectura de la aplicación y las decisiones de diseño han sido **comprendidas, validadas y asumidas**. El uso de estas herramientas se ha limitado a un apoyo en la resolución de problemas y en la mejora de la expresión escrita, sin delegar en ningún caso el trabajo de análisis, desarrollo o depuración.

Anexo C: Herramientas de desarrollo y recursos técnicos

- **IDE:** Android Studio (última versión estable)
- **Lenguaje:** Kotlin
- **Backend:** Supabase (PostgreSQL, Auth, Realtime, RLS)
- **Persistencia local:** Room
- **Inyección de dependencias:** Dagger Hilt

- **Navegación:** Jetpack Navigation Component
- **Pruebas manuales:** Emulador Pixel 9 Pro (API 33) y dispositivo físico Xiaomi Mi 11 (API 31)
- **Control de versiones:** Git + GitHub
- **Gestión de tareas:** Trello (tablero Kanban)

Anexo D: Licencia