

MEMORIA ESCRITA DEL PROYECTO

CFGS Desarrollo de Aplicaciones Multiplataforma

PrioList: Gestor de tareas eficaz

Autor: Alberto Jesús Gómez Aranda

Tutor: Gotzon Valcárcel

Fecha de entrega: 09/12/2024

Convocatoria: 1s2425

Documentos del proyecto: https://drive.google.com/drive/folders/1j-YtOHnITwONtPrW_8qdFFM1pZ5I50Hh?usp=drive_link



Índice

1.	Introducción.....	2
1.1.	Motivación	2
1.2.	Abstract	3
1.3.	Objetivos propuestos (generales y específicos).....	3
2.	Estado del Arte.....	5
3.	Metodología usada.....	8
4.	Tecnologías y herramientas utilizadas en el proyecto	10
5.	Planificación, Diagnóstico y Contexto Laboral	11
6.	Análisis del proyecto	16
7.	Diseño del proyecto.....	25
8.	Despliegue y pruebas.....	31
9.	Conclusiones.....	34
10.	Vías futuras	35
11.	Bibliografía/Webgrafía	37
12.	Anexos	39
12.1.	Manual de Instalación	39
12.2.	Manual de Usuario.....	40

1. Introducción

El presente documento es la memoria del proyecto fin de ciclo de Alberto Jesús Gómez Aranda perteneciente al ciclo de Desarrollo de Aplicaciones Multiplataforma. *PrioList* es una aplicación orientada a la gestión de tareas personales de manera eficaz y sencilla, tratando de mantener las funcionalidades precisas para que el uso de la aplicación sea intuitivo, permitiendo la creación y actualización de tareas pudiendo asignarle recordatorios y categorías. De la misma manera se podrán organizar todas las tareas en una vista general para visualizar de manera clara la prioridad de las mismas.

1.1. Motivación

El desarrollo de *PrioList* surge de la necesidad de gestionar de manera eficiente las tareas personales en un mundo cada vez más dinámico y lleno de compromisos. La falta de organización puede generar estrés, afectar la productividad y llevar a olvidar responsabilidades importantes. Aunque existen muchas aplicaciones de gestión de tareas, muchas de ellas son complejas, requieren aprendizaje previo o incluyen funciones que los usuarios no siempre necesitan. Además, es común el requerimiento de acceder mediante registros a muchas de estas aplicaciones, pese a que no ofrecen servicio en distintas plataformas.

PrioList se propone como una solución simple pero eficaz, enfocada en priorizar tareas de forma clara y accesible. La motivación principal para desarrollar esta aplicación radica en mi experiencia personal cuando se me propuso usar distintas aplicaciones de este tipo que requerían una gran inversión inicial de esfuerzo para entender la totalidad de herramientas ofrecidas y un esfuerzo continuado por mantener organizado dicho espacio de trabajo que acababa ocasionando el abandono de la aplicación, de la misma manera es igual de motivante la oportunidad de proporcionar una herramienta intuitiva y ligera, que permita a los usuarios concentrarse en lo realmente importante: cumplir sus objetivos diarios y mantener el equilibrio entre sus responsabilidades y su tiempo libre.

Además, este proyecto es una excelente oportunidad para aplicar conocimientos adquiridos en el grado de Desarrollo de Aplicaciones Multiplataforma, como el diseño de interfaces

amigables, la implementación de bases de datos locales y la utilización de Kotlin como lenguaje principal. La experiencia de crear *PrioList* no solo responde a una necesidad personal y profesional, sino que también ofrece la posibilidad de aportar una solución útil para mi día a día como al de otros posibles usuarios.

1.2. Abstract

Task management application for keeping personal reminders up to day in one place.

The user will be able to create a task, assign it with a description, a category, a priority and a reminder that will pop up at a specified date. User will be able also to update tasks in case it's needed. User can delete task when completed or in case it's not needed anymore.

The application allows the user to sort out tasks by different parameters to make it more accessible and easier to keep track. There is also the option to switch between different grids adding comfort in the visualization of tasks.

1.3. Objetivos propuestos (generales y específicos)

El objetivo general de desarrollar una aplicación móvil como *PrioList* es permitir a los usuarios gestionar de manera eficiente sus tareas personales a través de una solución funcional e intuitiva. La aplicación incluirá funcionalidades esenciales como la creación, edición, eliminación y organización de tareas, integrando recordatorios personalizados y almacenamiento local mediante SQLite.

- i. Implementar funcionalidades clave:
 - a. Permitir a los usuarios agregar tareas con detalles como:
 - 1. Nombre de la tarea
 - 2. Descripción o notas asociadas
 - 3. Fecha y hora de vencimiento
 - 4. Categorías y prioridades asignadas
 - b. Incorporar recordatorios configurables para tareas específicas.

- ii. Utilizar SQLite y Room para garantizar que los datos se almacenen de forma local, preservando la información en todo momento.
- iii. Optimizar la gestión de tareas: Incluir funcionalidades para organizar las tareas por categorías y prioridad, ofreciendo una visión clara de las responsabilidades.
- iv. Implementar la arquitectura MVVM (Model-View-ViewModel) en el desarrollo de *PrioList* para garantizar una separación clara de responsabilidades entre los componentes de la aplicación. Este enfoque permitirá:
 - i. Mejorar la mantenibilidad del código: Al separar la lógica de negocio (Model), la lógica de presentación (ViewModel) y la interfaz de usuario (View), será más sencillo realizar modificaciones, agregar nuevas funcionalidades o depurar errores sin afectar otras capas.
 - ii. Facilitar la reutilización de componentes: El uso de ViewModels permitirá compartir la lógica de presentación entre diferentes vistas, promoviendo un desarrollo modular y escalable.
 - iii. Optimizar las pruebas unitarias: Gracias a la independencia de la lógica de negocio y la lógica de presentación respecto de la interfaz de usuario, será más fácil realizar pruebas unitarias para validar el comportamiento de la aplicación.
 - iv. Mantener una interfaz reactiva y actualizada: Al aprovechar la vinculación de datos y la reactividad, la vista se actualizará automáticamente en respuesta a los cambios en los datos, proporcionando una experiencia de usuario fluida y coherente.
- V. Fomentar la experiencia del usuario: Garantizar que todas las interacciones sean rápidas, sencillas y adaptadas a las necesidades reales de los usuarios, con especial atención a la claridad y facilidad de uso.

2. Estado del Arte

Introducción

Las aplicaciones de gestión de tareas y productividad han adquirido un papel esencial en la vida diaria de individuos y organizaciones. Estas herramientas permiten organizar, priorizar y realizar un seguimiento de actividades, mejorando la eficiencia y la planificación. En este estado del arte, se abordan las principales características, beneficios y retos de estas aplicaciones, así como su evolución en el mercado y las tendencias tecnológicas que han influido en su desarrollo.

Evolución de las Aplicaciones de Gestión de Tareas

El concepto de gestionar tareas de manera estructurada no es nuevo y ha estado presente desde los sistemas analógicos como las agendas físicas y las listas en papel. Sin embargo, la llegada de las computadoras personales en la década de 1980 marcó un punto de inflexión. Durante esta etapa inicial, surgieron las primeras aplicaciones informáticas que ofrecían funcionalidades limitadas, como calendarios y listas de tareas básicas. Estas herramientas estaban diseñadas principalmente para entornos corporativos y carecían de acceso remoto o capacidades de colaboración.

En los años 90, con el auge de sistemas operativos gráficos como Windows, las aplicaciones de gestión de tareas comenzaron a integrarse en suites de oficina. Microsoft Outlook, por ejemplo, ofrecía una herramienta integrada que combinaba correo electrónico, calendario y listas de tareas. Este enfoque empezó a popularizarse entre profesionales que requerían una gestión más centralizada de sus actividades.

La verdadera revolución llegó en la década de 2000 con la inclusión de los smartphones. Aplicaciones como Remember The Milk o Wunderlist aprovecharon la conectividad móvil y la sincronización en la nube, permitiendo a los usuarios acceder a sus listas desde cualquier lugar. Este período marcó el inicio de la interoperabilidad entre dispositivos, una característica que se convertiría en estándar en los años posteriores.

En la actualidad, las aplicaciones de gestión de tareas han evolucionado hacia soluciones más integrales y personalizables. Herramientas como Trello, Todoist y Notion ofrecen capacidades avanzadas como gestión de proyectos, colaboración en equipo y análisis de productividad. Además, la incorporación de inteligencia artificial ha comenzado a transformar

la experiencia del usuario, facilitando la automatización de tareas y la personalización de recomendaciones.

Características de las Aplicaciones de Gestión de Tareas

Las aplicaciones modernas de gestión de tareas destacan por ofrecer una variedad de funcionalidades que buscan adaptarse a diferentes perfiles de usuarios. Entre las características más comunes se encuentran:

- **Creación de tareas:** Permite a los usuarios definir actividades con detalles como nombres, descripciones, fechas de vencimiento y niveles de prioridad.
- **Recordatorios:** Notificaciones que alertan sobre tareas pendientes o próximas a vencer.
- **Organización por categorías o etiquetas:** Facilita la clasificación y búsqueda de tareas relacionadas.
- **Integración multiplataforma:** La mayoría de las herramientas actuales son accesibles desde dispositivos móviles, tablets, ordenadores y navegadores web, garantizando una experiencia continua entre plataformas.
- **Colaboración en equipo:** Muchas aplicaciones, como Trello o Asana, permiten la asignación de tareas a múltiples usuarios y la interacción en tiempo real mediante comentarios o actualizaciones.
- **Integración con otros servicios:** Herramientas como Todoist o Notion se integran con calendarios, correos electrónicos y aplicaciones de comunicación, ofreciendo una experiencia más holística.

Análisis Comparativo de Aplicaciones Representativas

Algunas de las aplicaciones más destacadas en el mercado incluyen:

1. **Trello:** Basada en el método Kanban, Trello organiza tareas en tableros y columnas. Su enfoque visual facilita la gestión de proyectos en equipo. Sin embargo, su flexibilidad puede resultar abrumadora para usuarios que buscan una solución más simple.

2. **Todoist:** Popular por su simplicidad y versatilidad, permite gestionar tareas de manera personal o en equipo. Ofrece funciones como objetivos diarios y un sistema de puntuación para fomentar la productividad.
3. **Microsoft To Do:** Esta herramienta se integra con el ecosistema de Microsoft, proporcionando una experiencia conectada para usuarios que ya utilizan Office 365.
4. **Notion:** Aunque no es exclusivamente una aplicación de gestión de tareas, su versatilidad permite organizar tareas, notas y proyectos en un mismo espacio, atrayendo a usuarios que buscan una solución integral.
5. **Obsidian:** Orientada principalmente a la toma de notas interconectadas, Obsidian incluye funcionalidades que permiten gestionar tareas dentro de un flujo de trabajo basado en mapas mentales. Su enfoque en la organización del conocimiento la hace única, especialmente para usuarios que combinan la gestión de tareas con la creación de contenido estructurado.
6. **Asana:** Orientada al trabajo colaborativo, es ampliamente utilizada por equipos en empresas. Su robustez la hace ideal para proyectos grandes, aunque puede resultar compleja para tareas personales.

Retos y Limitaciones

A pesar de su utilidad, estas aplicaciones enfrentan desafíos. Uno de los principales es el exceso de funcionalidades, que puede saturar al usuario promedio. Además, la dependencia de la conectividad a internet en muchas aplicaciones dificulta su uso en zonas con acceso limitado a redes. Otro reto importante es la gestión de la privacidad, ya que muchos usuarios son reticentes a compartir información sensible en plataformas digitales.

Tendencias Actuales

El desarrollo de estas aplicaciones sigue tendencias marcadas por avances tecnológicos y cambios en las necesidades del usuario:

1. **Inteligencia Artificial (IA):** La incorporación de IA permite ofrecer recomendaciones personalizadas, priorización automática de tareas y predicciones sobre tiempos de finalización.

2. **Integración multiplataforma:** Las aplicaciones buscan garantizar una experiencia uniforme en dispositivos móviles, de escritorio y web.
3. **Gestión emocional:** Algunas herramientas, como Serene, se enfocan en la productividad consciente, combinando tareas con ejercicios de relajación y pausas programadas.
4. **Simplicidad en el diseño:** Frente a la saturación de funciones, hay una tendencia hacia interfaces minimalistas y personalizables que se adaptan a las necesidades individuales.

Conclusiones

Las aplicaciones de gestión de tareas han evolucionado significativamente, adaptándose a los cambios tecnológicos y a las demandas de los usuarios. Desde herramientas básicas de listas de tareas hasta plataformas integrales con IA, estas aplicaciones desempeñan un papel fundamental en la mejora de la productividad. Sin embargo, el equilibrio entre funcionalidad, simplicidad y privacidad sigue siendo un desafío clave para los desarrolladores.

En este contexto, *PrioList* se presenta como una propuesta que busca abordar estas problemáticas mediante una solución sencilla, eficiente y centrada en las necesidades reales del usuario.

3. Metodología usada

Dado que *PrioList* contaba con unos requisitos claros desde su concepción y no contaba con la necesidad de trabajar en equipo, con lo que ello conlleva, me decidí por aplicar la metodología Kanban.

Las razones por las que decidí aplicar esta metodología son las siguientes:

- Tratarse de un proyecto de poca duración en el tiempo, alrededor de mes y medio.
- Visualización clara y sencilla sobre las tareas a llevar a cabo
- Sencillez en la planificación, uso y seguimiento.
- Los requerimientos de *PrioList* estaban cerrados con poca posibilidad de variar

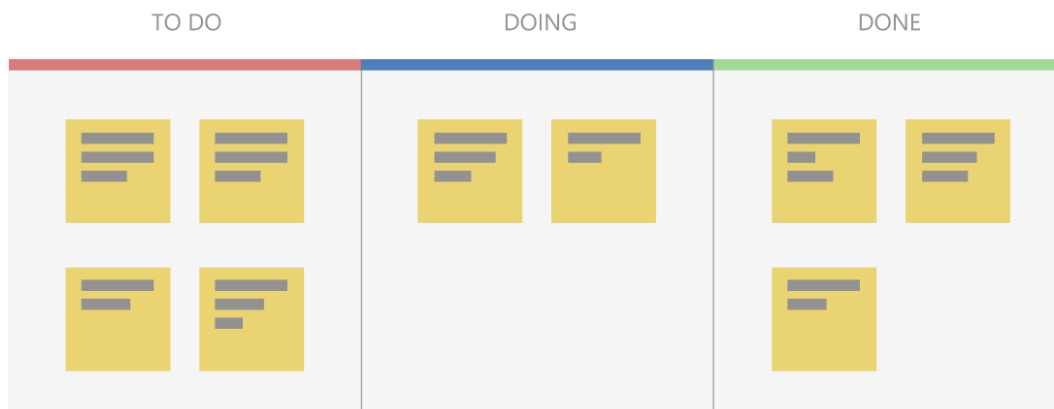


Ilustración 2. Concepto de tablero Kanban

Otros métodos posibles

Al evaluar qué metodología utilizar para el desarrollo de mi proyecto, consideré otras opciones que acabaron siendo descartadas como el modelo cascado, el modelo V y la programación extrema (XP).

El modelo cascada sigue una estructura lineal y rígida en la que cada fase debe completarse antes de pasar a la siguiente. Aunque es útil para proyectos con requisitos bien definidos desde el principio, no ofrece la flexibilidad necesaria para adaptarse a cambios inesperados.





Por otro lado, el modelo V se caracteriza por ser un enfoque secuencial donde cada fase de desarrollo está estrechamente vinculada con su correspondiente fase de pruebas, lo que permite una validación continua del producto. Este modelo proporciona un marco claro y estructurado, lo que facilita el control de calidad y asegura que los requisitos sean verificados a medida que se avanza. Sin embargo, al igual que el modelo cascada, este enfoque presenta desafíos al ser menos flexible para incorporar cambios durante el proceso de desarrollo, lo que puede ser un inconveniente si surgen nuevas necesidades a medida que avanza el proyecto.

Por último, la Programación Extrema (XP) también fue considerada. Este enfoque se centra en la retroalimentación constante del cliente y en la realización de ciclos incrementales donde se va refinando el producto con cada entrega. Sin embargo, debido a que en mi caso no cuento con ninguna figura que revise y aporte comentarios frecuentes en cada ciclo, esta metodología podría resultar demasiado exigente en cuanto al tiempo y los recursos.


requeridos. Además, el enfoque de entregas incrementales que aporta valor funcional con cada ciclo hubiera alargado innecesariamente el proceso.


4. Tecnologías y herramientas utilizadas en el proyecto

Herramientas de desarrollo

	Herramienta	Descripción y uso
	Android Studio Ladybug 2024.2.1	IDE para el desarrollo nativo de aplicaciones Android y diseño de layouts XML. Especialmente útil dada la integración de su emulador de dispositivos. Soporta Kotlin.
	Kotlin 1.9.20	Lenguaje de programación en que se desarrollará el proyecto. No he utilizado una versión superior debido a la facilidad que hay para encontrar documentación y resolución de problemas para las librerías usadas previo al lanzamiento de Kotlin 2.0.0
	Github	Alojamiento para el repositorio donde se podrá mantener el código fuente ordenado y añadir nuevas funcionalidades sin riesgo de romper otras mediante ramas.
	Notion	Web para la creación y organización de tareas, anotaciones y recopilación de webgrafía y recursos de interés para el proyecto

Base de datos

	Herramienta	Descripción y uso
	SQLite	Motor de base de datos local que nos permitirá almacenar y realizar consultas de manera ligera y eficiente en nuestra base de datos de tamaño pequeño

	RoomDB 2.6.1	Librería que añadirá una capa de abstracción a SQLite y nos permitirá usar interfaces DAO, Entidades y hacer migraciones automáticas.
---	-----------------	---

Otras herramientas utilizadas:

- www.diagrams.net → Realización de esquemas, diagramas E-R, casos de uso y diagramas de entidad.
- www.teamgantt.com → Representación del diagrama de Gantt para controlar el estado de las tareas del proyecto.
- www.pictogrammers.com → Descarga de iconos del tema Material Design.
- www.visily.ai → Realización de mockups para las vistas de la aplicación.
- Microsoft Word → Realización de la memoria del proyecto.
- Microsoft PowerPoint → Realización de las diapositivas de cara a la presentación en video.
- OBS Studio → Grabación de la defensa del proyecto para permitir la creación de escenas donde se pueda ver la webcam, la presentación y posteriormente el emulador de Android para la demo.
- Poco X6 Pro → Smartphone con Android 14 para pruebas.
- Motorola Moto g42 → Smartphone con Android 12 para pruebas.

5. Planificación, Diagnóstico y Contexto Laboral

En el proceso de planificación del proyecto, las tareas fundamentales se han definido considerando tres elementos clave: requisitos funcionales, análisis preliminar y metodología de desarrollo. Además, se ha incorporado una fase complementaria no convencional en el desarrollo de software, destinada específicamente a la elaboración de la documentación de la memoria y la preparación del material audiovisual de presentación.

La siguiente tabla detalla el resultado final de este proceso:

Fase	ID	Actividad	Estimación(h)
Investigación	R1	Definir el alcance del del proyecto, las tecnologías y metodologías a usar y realizar una estimación del trabajo necesario.	5

Investigación	R2	Definir los requisitos que debería cumplir la aplicación para ser un producto viable.	10
Investigación	R3	Llegar al acuerdo de no añadir más funcionalidad a no ser que sea imprescindible.	2
Diseño	D1	Creación de diagramas de entidad-relación, diagramas de casos de uso y diagramas de clase.	8
Diseño	D2	Producción de mockups que se aplicarán en las vistas de la aplicación.	10
Diseño	D3	Decidir entre cuál de los diseños creados es el más funcional.	2
Entorno	E1	Instalación de Android Studio e investigación sobre la adecuación de las versiones de las tecnologías que se van a usar en el proyecto.	3
Implementación	I1	Creación de la vista principal de la aplicación, así como el tema que predominará en la aplicación.	30
Implementación	I2	Creación de las vistas y formularios que se utilizarán para la interacción del usuario con los datos de la aplicación.	20
Implementación	I3	Creación de las vistas que estarán disponibles que el usuario vea las tareas de distintas formas.	15
Implementación	I4	Creación del modelo de datos y la adaptación necesaria para almacenarlos en la base de datos.	10
Implementación	I5	Implementar la librería RoomDB para tener persistencia local de datos en la aplicación.	10
Implementación	I6	Creación de distintos componentes necesarios para la lógica de la aplicación.	20
Implementación	I7	Creación de utilidades necesarias para el correcto manejo de los datos.	10
Implementación	I8	Implementación del servicio de notificaciones de Android.	10
Implementación	I9	Realización de pruebas y revisión del cumplimiento de los casos de uso.	5
Documentación	P1	Escritura y revisión continua de la memoria del proyecto.	16
Documentación	P2	Creación de diapositivas de cara a la defensa.	2
Documentación	P3	Defensa del proyecto en video.	2

El proyecto será ejecutado íntegramente por un único recurso humano, lo que implica la asunción total de responsabilidades y tareas. Dadas las restricciones temporales, no se han contemplado periodos vacacionales ni festivos, optimizando la disponibilidad para el desarrollo del proyecto.

El organigrama temporal propuesto:

- Duración total: Del 27 de octubre al 7 de diciembre.
- Dedicación semanal: 33 horas.

Alberto Jesús Gómez Aranda

- Distribución aproximada: 4,5 horas diarias.

La planificación original presentaba una infravaloración en la asignación de horas de trabajo. Inicialmente se programaron 190 horas, mientras que la ejecución real se aproximó a las 205 horas. Las principales causas de este desajuste fueron:

- I. Inexactitud en la asignación inicial de horas diarias en las primeras fases.
- II. Necesidad de compensar mediante horas extraordinarias en la fase de implementación.
- III. Ajustes operativos para mantener la programación dentro de unos márgenes razonables.

Estrategias para revertir el desajuste:

- I. Compensación de la asignación de tiempo en las primeras fases.
- II. Aplicación de horarios ampliados (hasta 8 horas diarias)
- III. Gestión eficaz para evitar desplazamientos de tareas a días posteriores.

Conclusión operativa:

Fue posible mantener la estructura de trabajo establecida inicialmente, sin sobrepasar los límites de una jornada laboral convencional. La flexibilidad y la adaptabilidad fueron esenciales para gestionar las desviaciones detectadas y garantizar el cumplimiento de los objetivos y plazos del proyecto.

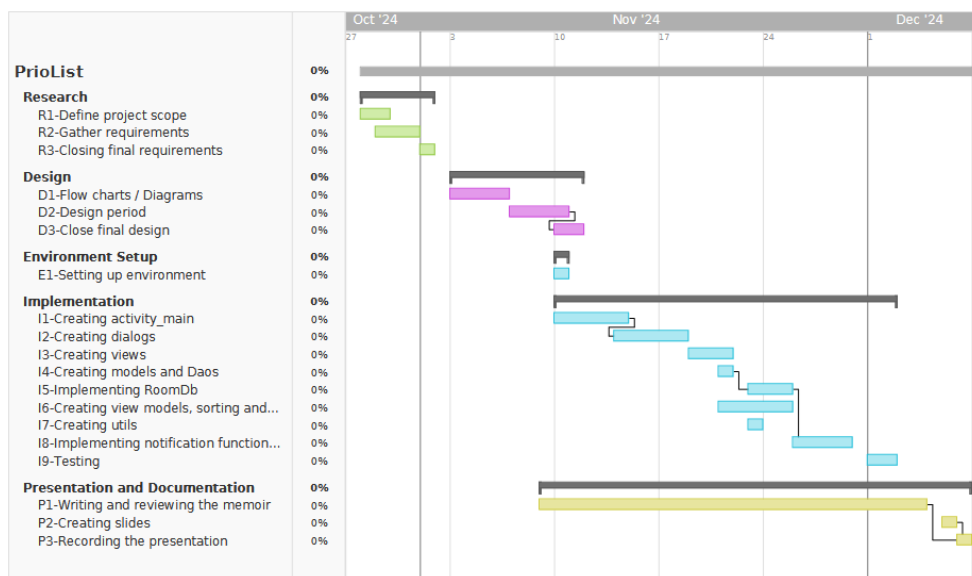


Ilustración 3. Diagrama de Gantt una vez finalizado el proyecto

Esta metodología permite un enfoque sistemático y riguroso de la planificación, contemplando tanto los aspectos técnicos como los operativos del desarrollo.

Contexto Laboral

El desarrollo de este proyecto surge como respuesta directa a una necesidad organizativa actual: optimizar la gestión, coordinación y control de las peticiones y consultas relacionadas con nuestro software. Más allá de su finalidad operativa, esta iniciativa representa una importante oportunidad de desarrollo profesional y ampliación de competencias técnicas.

Desde una perspectiva personal, el proyecto ha supuesto para mí una inmersión profunda en dos tecnologías muy demandadas, como son Android y Kotlin, que sin duda enriquecerán mi perfil profesional:

El proceso de desarrollo de este proyecto ha supuesto que haya tenido que ampliar mis conocimientos sobre las herramientas de desarrollo empleadas, provocando una inmersión de lleno en todo el ecosistema de desarrollo Android y permitiéndome aprender en cada decisión por pequeña que fuese durante todo el proceso de elaboración del proyecto.

Alberto Jesús Gómez Aranda

Más allá de las líneas de código, este proyecto representa un punto de inflexión en mi carrera profesional. Ha significado:

- La capacidad de transformar un reto técnico en una solución práctica.
- Una oportunidad de crecimiento profesional.
- La demostración de adaptabilidad tecnológica.

Este proceso de desarrollo ha confirmado mi convicción sobre la importancia de:

- La formación continua
- La adaptación a las nuevas tecnologías
- La búsqueda constante de soluciones innovadoras

Cada reto técnico se ha convertido en una oportunidad de aprendizaje y en un ejercicio de proveer soluciones prácticas con las herramientas disponibles, consolidando habilidades que serán fundamentales en mi futuro desarrollo profesional.

En el dinámico ecosistema empresarial actual, marcado por constantes transformaciones y escenarios de gran complejidad, desarrollar una rigurosa capacidad analítica se ha convertido en un diferenciador estratégico para cualquier iniciativa organizativa. La herramienta DAFO se posiciona como un instrumento de diagnóstico esencial, ofreciendo una metodología estructurada que permite una comprensión holística y sistemática de la dinámica interna y externa de un proyecto o entidad. Debemos aplicar esta herramienta a nuestro proyecto para orientarnos dentro del mercado.



Ilustración 4. Matriz Dafo de PrioList

6. Análisis del proyecto

En la etapa inicial de diagnóstico, se procede a la identificación y caracterización exhaustiva de los elementos constitutivos del sistema, determinando los parámetros de interacción y delimitando el alcance y restricciones operativas.

Siguiendo el orden lógico este apartado empezará detallando los requisitos funcionales y no funcionales para proveer de contexto en la siguiente parte dedicada a la exposición de los diagramas requeridos a la hora de empezar el desarrollo del proyecto.

Requisitos funcionales:

1. Gestión dinámica de tareas

La funcionalidad de creación de tareas se concibe como una experiencia fluida y simple.

- Implementación de un sistema de ingreso de tareas que requiera únicamente información esencial.
- Campos mínimos predeterminados.

2. Sistema de categorización personalizada

Generación de categorías mediante entrada manual para tener una máxima personalización.

3. Mecanismo de priorización.

- Modelo de priorización múltiple:

Clasificación por:

- Prioridad
- Categoría
- Nombre
- Notificaciones push up individuales para cada tarea.

Requisitos no funcionales:

1. Rendimiento: Eficiencia Técnica

- Velocidad de Carga:
 - Tiempo máximo de inicio: 2 segundos.
 - Animación en mientras carga.
- Optimización Energética:
 - Gestión inteligente de recursos del dispositivo

2. Usabilidad: Diseño centrado en el usuario

- Interfaz minimalista:
 - Diseño siguiendo principios de diseño flat
 - Jerarquía visual clara
 - Transiciones suaves entre estados
- Curva de aprendizaje ultrarrápida:
 - Tiempo de familiarización: menos de 3 minutos
 - Interfaz intuitiva y clara.

3. Compatibilidad: Accesibilidad multiplataforma

- Soporte Android integral:
 - Compatibilidad desde Android 5.0, Api 21
- Adaptabilidad visual:
 - Diseño responsive

4. Arquitectura MVVM: Diseño modular y mantenible

- Separación de responsabilidades:
 - Modelo: Gestión de datos y lógica de negocio
 - Vista: Representación visual de la interfaz de usuario
 - ViewModel: Intermediario entre Modelo y Vista
- Beneficios de arquitectura:
 - Desacoplamiento de componentes
 - Facilidad de testing
 - Mantenibilidad del código
- Características clave:
 - Binding directo entre Vista y ViewModel
 - Independencia de la interfaz de usuario
 - Reutilización de componentes

Diagrama Entidad-Relación

Seguidamente se representa el modelo entidad-relación, usado para representar datos, entidades y las relaciones entre ellas para diseñar la base de datos.

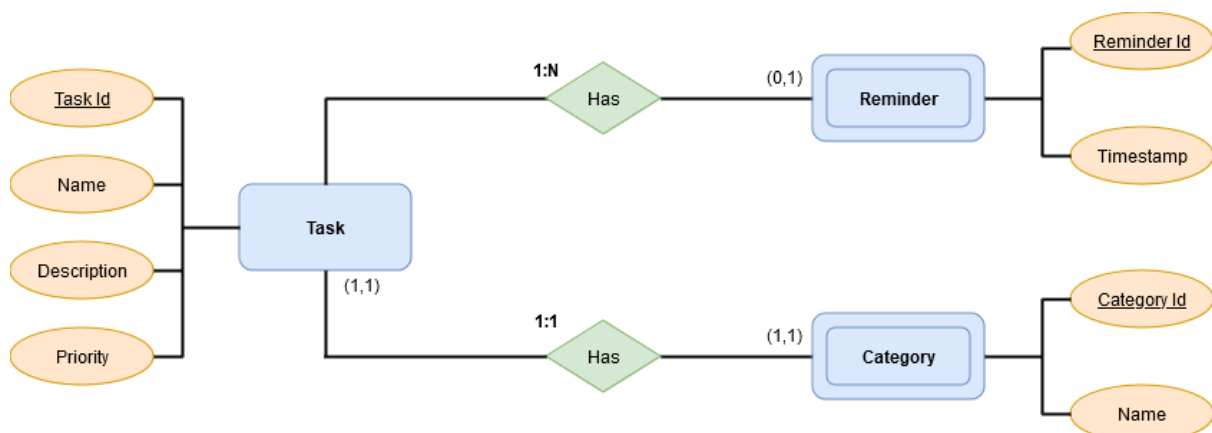


Ilustración 5. Diagrama Entidad-Relación de PrioList

Entidades:

- **Task:** Entidad central de la aplicación, tareas que el usuario crea, actualiza o elimina.
- **Reminder:** Recordatorio referente a una tarea.

- **Category:** Categoría asignada a cada tarea.

Casos de uso

Los casos de uso nos permiten planificar el comportamiento que debe cumplir la aplicación para satisfacer las necesidades del usuario. Para ello es necesaria la creación de diagramas sencillos que permitan visualizar correctamente y mejoren la comprensión del sistema.

Aquí se muestra el diagrama de casos de uso general para toda la aplicación:

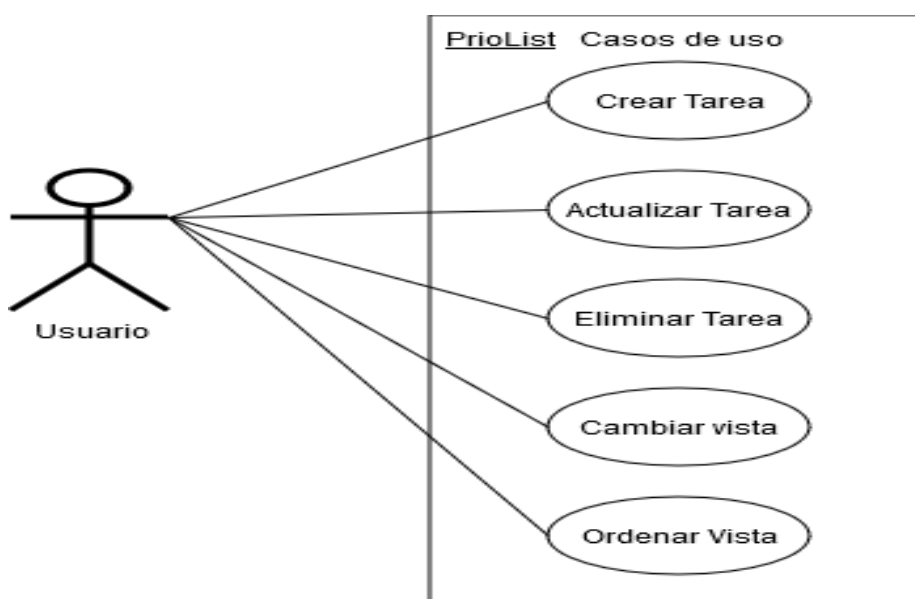


Ilustración 6. Casos de uso de PrioList

Como podemos ver en la ilustración, la aplicación ha de cumplir cinco casos de uso para que se considere funcional y provea al usuario del servicio planificado. A continuación, se expondrán los distintos casos de uso.

CU-1 CREAR TAREA

Identificador de Caso Uso	CU-1
Nombre	Crear tarea
Descripción	Proceso de crear una tarea

Secuencia normal	
Usuario	Software
1. Pulsa sobre el botón "Add Task"	
	2. Se muestra el diálogo de creación de tarea
3. Se introducen los datos de la tarea	
	4. Se registra la tarea
	5. Se vuelve a la vista general
	6. Se lanza una notificación en la hora establecida
Excepciones	Software
Usuario no guarda la tarea	Vuelve a la pantalla principal
Usuario no incluye hora de notificación	Notificación aparece automáticamente
Usuario no incluye título	Muestra advertencia de campo requerido
Usuario no incluye descripción	Muestra advertencia de campo requerido
CU relacionados	
Precondición	Debe rellenar los campos título y descripción
Post condición	El usuario puede visualizar la tarea

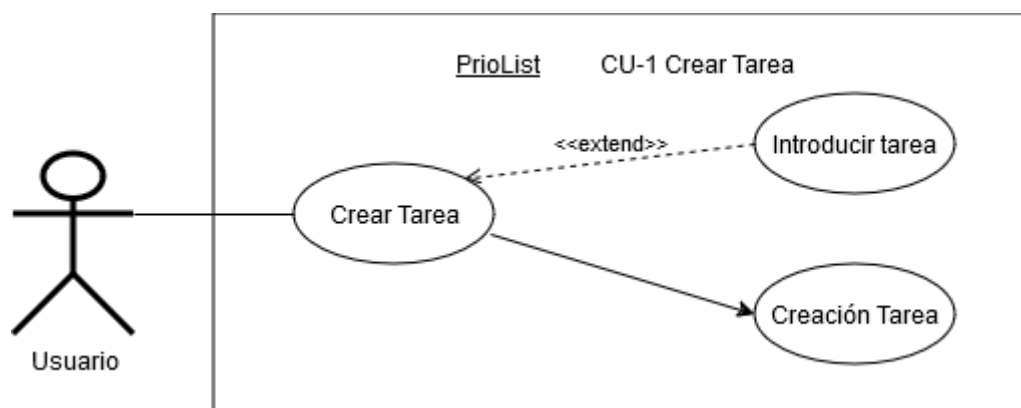


Ilustración 7. Caso de uso "Crear Tarea"

CU-2 ACTUALIZAR TAREA

Identificador de Caso Uso	CU-2
Nombre	Actualizar tarea

Descripción	Proceso de actualizar una tarea
Secuencia normal	
Usuario	Software
1. Pulsa sobre el botón "Modify Task"	
	2. Se muestra el diálogo de modificación de tarea
3. Se introducen los datos de la tarea	
	4. Se actualiza la tarea
	5. Se vuelve a la vista general
	6. Se lanza una notificación en la hora establecida
Excepciones	Software
Usuario no guarda la actualización	Vuelve a la pantalla principal con la tarea en su estado anterior
Usuario no incluye hora de notificación	Notificación aparece automáticamente
Usuario borra el título	Muestra advertencia de campo requerido
Usuario borra la descripción	Muestra advertencia de campo requerido
CU relacionados	CU-1
Precondición	1. Debe rellenar los campos título y descripción 2. Debe existir la tarea
Post condición	El usuario puede visualizar la tarea actualizada

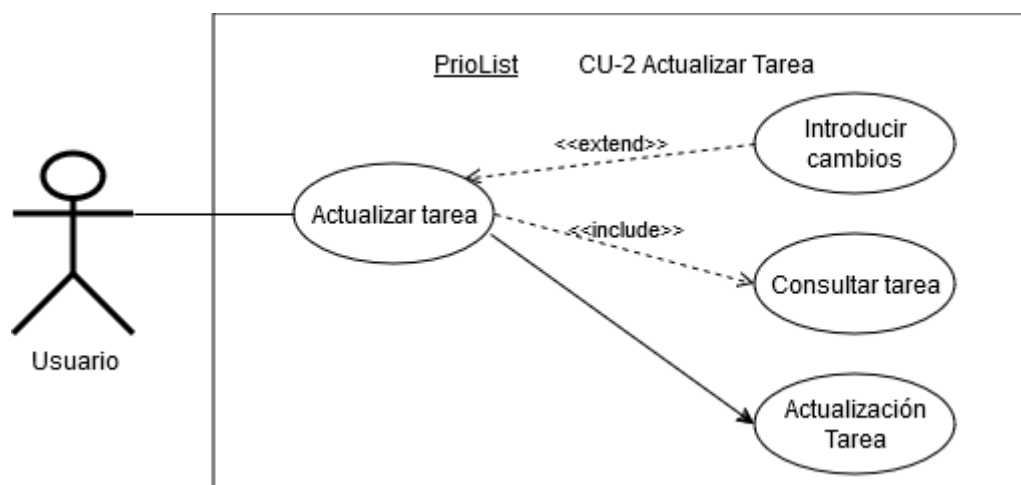


Ilustración 8. Caso de uso "Actualizar Tarea"

CU-3 ELIMINAR TAREA

Identificador de Caso Uso	CU-3
Nombre	Eliminar tarea
Descripción	Proceso de eliminar una tarea
Secuencia normal	
Usuario	Software
1. Pulsa sobre el botón "Delete Task"	
	4. Se elimina la tarea
	5. Se vuelve a la vista general
Excepciones	Software
CU relacionados	CU-1
Precondición	Debe existir la tarea
Post condición	El usuario deja de visualizar la tarea

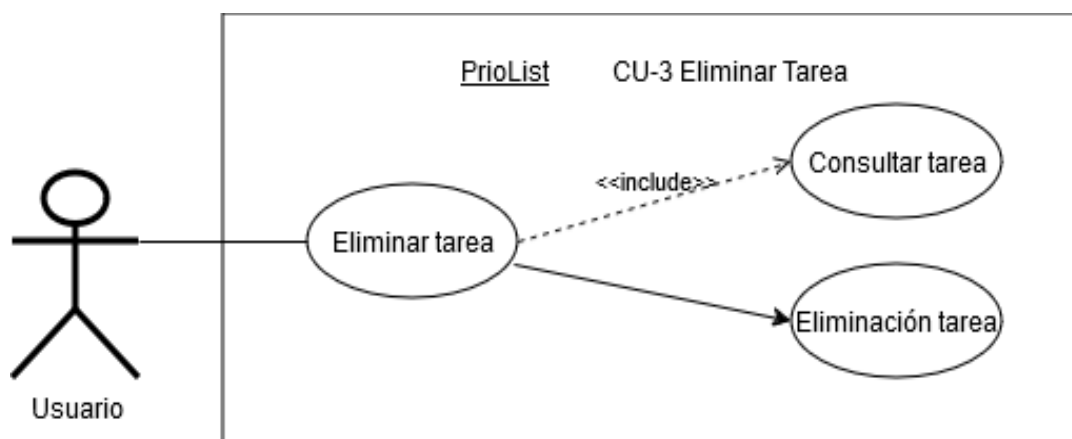


Ilustración 9. Caso de uso "Eliminar Tarea"

CU-4 CAMBIAR VISTA

Identificador de Caso Uso	CU-4
Nombre	Cambiar vista
Descripción	Proceso de cambiar el diseño de la vista general
Secuencia normal	
Usuario	Software
1. Pulsa sobre el botón "Change View"	
	2. Acceso a las tareas existentes

	3. Acceso al orden previo de tareas
	4.Cambio de la vista general
Excepciones	Software
Usuario no tiene tareas	No hay cambio aparente de la vista general
CU relacionados	CU-1
Precondición	Debe existir la tarea
Post condición	El usuario puede visualizar las tareas con la organización en pantalla seleccionada

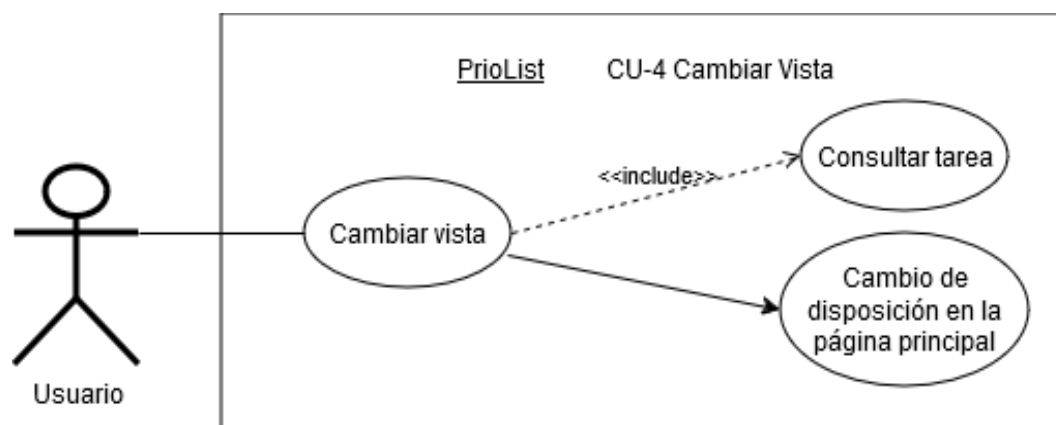


Ilustración 10. Caso de uso "Cambiar Vista"

CU-5 ORDENAR VISTA

Identificador de Caso Uso	CU-5
Nombre	Ordenar vista
Descripción	Proceso de organización de tareas según varios parámetros
Secuencia normal	
Usuario	Software
1. Pulsa sobre el botón "Sort Task"	
	2. Se muestra el diálogo de selección de parámetros que ordenar
3. Se introducen el parámetro deseado	
	4.Acceso a las tareas existentes

	5.Reordenación según el parámetro introducido
	6.Consulta la vista actual
	7.Modifica la visualización de tareas en la vista general
Excepciones	Software
Usuario no selecciona el parámetro deseado	Vuelta a la vista general sin modificaciones
Usuario selecciona el parámetro ya especificado	Vuelta a la vista general sin modificaciones
Existen tareas sin ese parámetro	Dichas tareas se muestran al final de la vista general
CU relacionados	CU-1, CU-4
Precondición	1.Debe existir la tarea 2.Debe existir parámetros para organizar
Post condición	El usuario puede visualizar la lista de tareas reordenada según el parámetro elegido

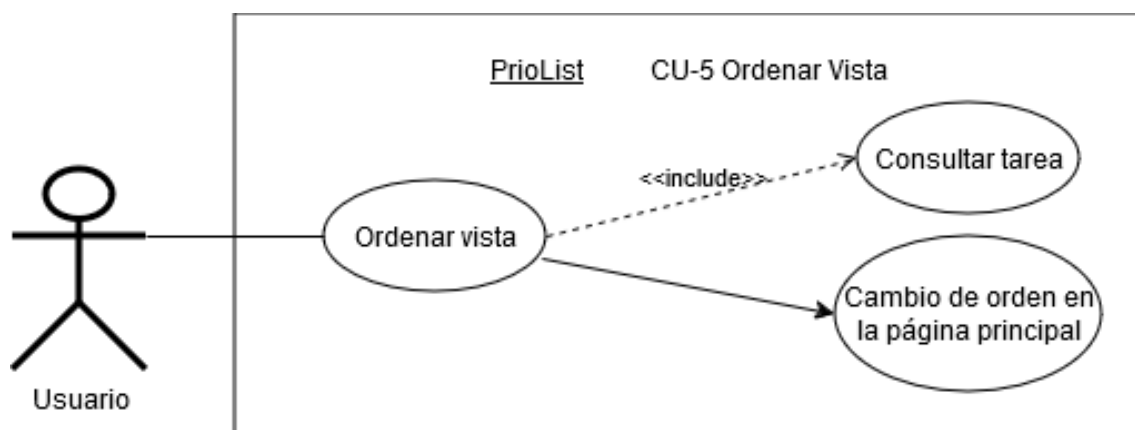


Ilustración 11. Caso de uso "Ordenar Vista"

Diagrama de clases

Durante la fase de análisis también es de vital importancia generar un diagrama de clases para esquematizar la estructura de datos y aproximar el alcance que va a conllevar implementarla. En el caso de *PrioList* éste diagrama refleja fidedignamente la intencionalidad de mantener la aplicación con funcionalidades minimalistas para simplificar la el uso de la misma.

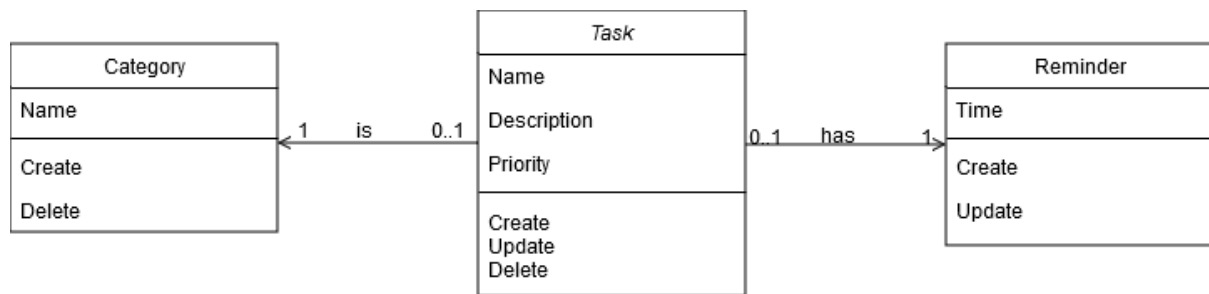


Ilustración 12. Diagrama de clases de PrioList

7. Diseño del proyecto

7.1. Prototipos

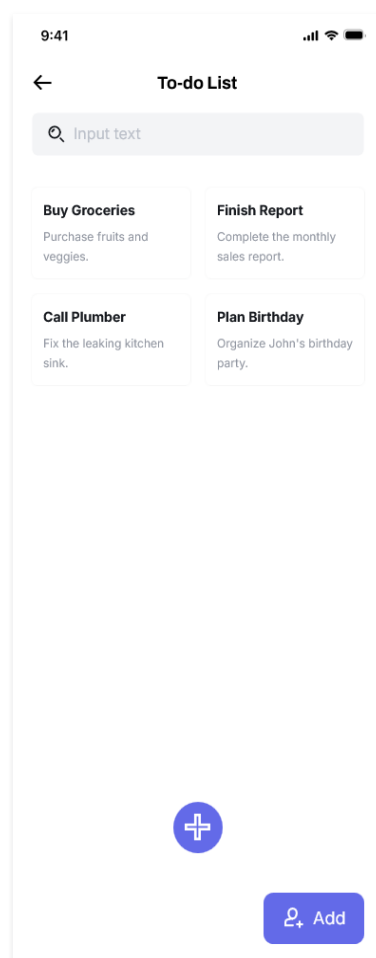


Ilustración 13. Mockup de la vista cuadriculada

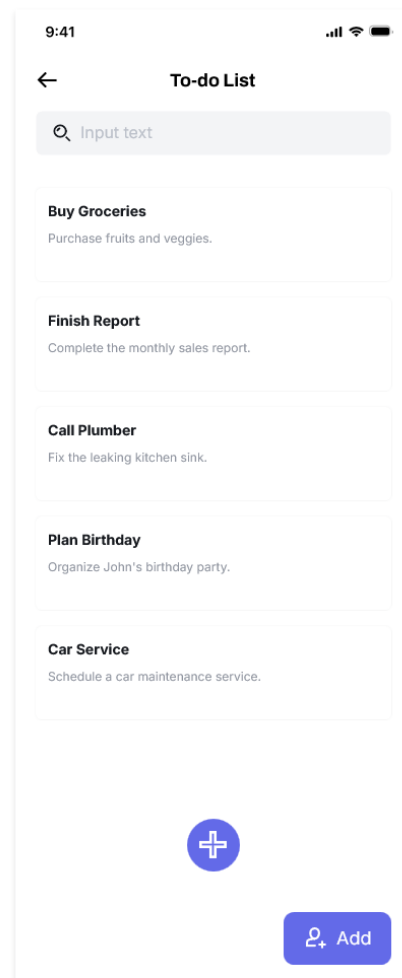


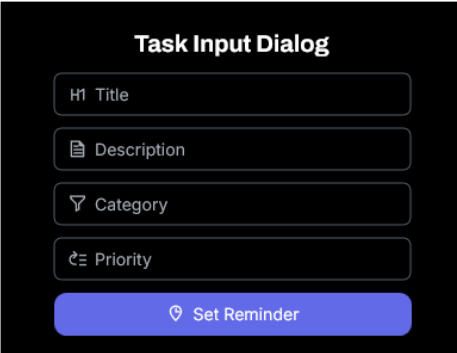
Ilustración 14. Mockup de la vista en forma de lista

7.1.1. Vista principal

Como se puede observar en la página anterior se crearon dos ilustraciones para las dos vistas que soporta actualmente *PrioList* de manera que el usuario pudiese elegir la que más cómoda le resultase. Se incluyeron dos versiones del botón “Add Task” en cada lista para probar con ambas la experiencia de usuario. Durante el desarrollo se acabó decidiendo en la opción cuadrangular con texto dado que provoca mayor armonía con las formas de las tareas.

De la misma manera se incluye en la barra superior un botón de retroceso que finalmente se desestimó por falta de necesidad.

7.1.2. Diálogo de creación/actualización de tareas

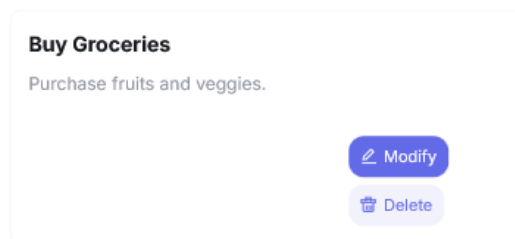
A screenshot of a 'Task Input Dialog' form. The form is titled 'Task Input Dialog' in bold. It contains four input fields: 'H1 Title', 'Description' (with a document icon), 'Category' (with a funnel icon), and 'Priority' (with a flag icon). Below these fields is a blue button labeled 'Set Reminder' with a location pin icon.

Made with Visily

Ilustración 15. Ventana de diálogo de creación/actualización de tareas

Posteriormente se creó el diálogo que aparecería a la hora de crear tareas tras pulsar “Add Task”, dado que los campos a modificar son los mismos se reutilizó para la funcionalidad de actualizar. Consta de cuatro campos para insertar texto y un botón para establecer una hora determinada.

7.1.3. Vista de tarea

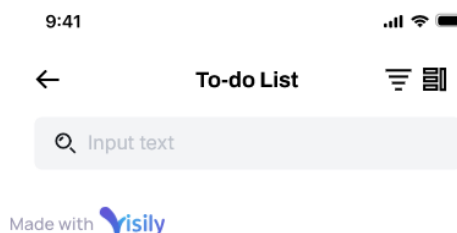


Made with Visily

Ilustración 16. Vista individual de tarea

Cada tarea crearía su propia carta en la vista general para su lectura, además se le añadiría dos botones, para modificar y eliminar. En la versión final de la aplicación se eliminó el texto de los botones para una mayor optimización del espacio en la vista cuadrículada.

7.1.4. Barra superior



Made with Visily

Ilustración 17. Barra superior con botones de organizar y cambiar vista añadidos.

Este prototipo refleja la idea de simplicidad que se pretende alcanzar durante el desarrollo del proyecto mediante el uso mínimo de texto en la comunicación con el usuario representando las funcionalidades de manera iconográfica en la medida de lo posible.

7.2. Arquitectura

Se ha elegido el modelo MVVM (Model-View-ViewModel) por su contribución clave para el desarrollo ágil y mantenible de aplicaciones como *PrioList*, al proporcionar una arquitectura

Alberto Jesús Gómez Aranda

que divide claramente las responsabilidades. Este enfoque organiza la complejidad en tres componentes principales:

- **Modelo:** gestiona los datos de forma pura, como tareas y prioridades, aislados de la lógica de presentación. Esto mejora la claridad y facilita las pruebas.
- **ViewModel:** Actúa como intermediario, transformando los datos del modelo en formatos listos para la vista, lo que permite manipular y filtrar la información sin complicaciones.
- **Vista:** Refleja el estado del ViewModel, siendo simple y estable. Esto reduce errores y asegura interfaces predecibles.

MVVM facilita las pruebas mediante componentes desacoplados, mejora la escalabilidad al permitir la integración de nuevas funciones y optimiza el trabajo en equipo al evitar interferencias entre capas. Además, la vinculación bidireccional de datos automatiza la sincronización entre los datos y la interfaz, reduciendo el código innecesario.

Más que una arquitectura, MVVM representa una estrategia de desarrollo centrada en la claridad, la mantenibilidad y la flexibilidad, ideal para aplicaciones como *PrioList* que priorizan la productividad personal.

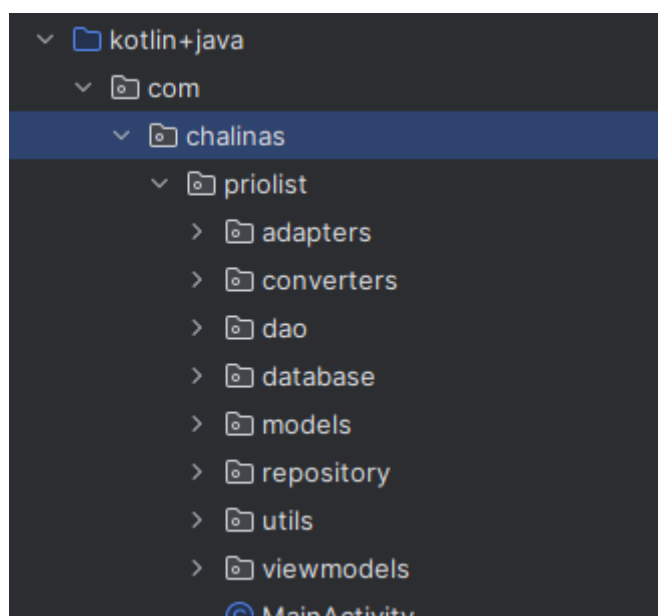


Ilustración 18. Árbol del proyecto

En el árbol del proyecto se puede ver claramente la delimitación de cada directorio y representar de manera clara la estructura del modelo MVVM y el funcionamiento general de *PrioList*.

Los modelos son el núcleo de los datos de una aplicación y representan las principales entidades. Estas estructuras, diseñadas específicamente para organizar la información, no incluyen lógica de procesamiento. Su función es reflejar objetos adaptando conceptos del mundo real a formatos comprensibles para la aplicación.

Los objetos de acceso a datos (DAO) sirven de intermediarios entre la aplicación y la base de datos. Estas interfaces especializadas definen las operaciones de consulta, inserción, actualización y eliminación, simplificando la interacción con el almacenamiento persistente. Los DAO ocultan los detalles internos de la aplicación y permiten un acceso directo y eficaz a los datos.

La carpeta Database centraliza la gestión del almacenamiento de datos en la aplicación. Este módulo configura todos los aspectos relacionados con la persistencia, incluyendo la creación, configuración y mantenimiento de la base de datos. Además, define las versiones y entidades de la base de datos, proporcionando métodos para interactuar con las DAOs.

El Repository actúa como una capa de abstracción que separa las fuentes de datos de los componentes que las consumen. Este componente unifica diferentes fuentes de información y decide dinámicamente de dónde obtener los datos. Proporciona una interfaz coherente y accesible para los ViewModels.

Los ViewModels gestionan la lógica de presentación, transformando los datos para que puedan representarse de forma óptima en la interfaz de usuario. Funcionan como intermediarios entre los datos y la visualización, garantizando que el estado de la interfaz se mantiene actualizado y obteniendo información del repositorio cuando es necesario.

Los adaptadores se encargan de representar los datos en componentes de lista, convirtiendo los modelos de datos en elementos visuales interactivos. Son esenciales para mostrar colecciones de información de forma eficaz, gestionando cada elemento de forma independiente y optimizando el rendimiento de las listas.

Los conversores se encargan de transformar tipos de datos complejos en formatos compatibles con Room Database. Esto incluye la conversión de estructuras personalizadas, como fechas o listas, en formatos que puedan almacenarse y manipularse correctamente en la base de datos.

Por último, las Utilidades agrupan funciones y herramientas generales que no encajan en categorías específicas.

7.3. Clases

Hay dos clases especialmente interesantes en este tipo de proyectos que merecen una profundización especial dada su vital importancia en el desarrollo de *PrioList*.

- **TaskViewModel**

La clase TaskViewModel es un modelo de vista en Android diseñado para gestionar los datos y la lógica de negocio asociada a las tareas. Su constructor recibe un objeto Application, que proporciona acceso a los recursos de la app, y un TaskRepository, encargado de realizar operaciones con la base de datos.

Entre sus propiedades principales, TaskViewModel utiliza objetos MutableLiveData para manejar la tarea actual, la lista de tareas, posibles errores y el estado de carga de los datos. A través de métodos como insertTask, updateTask y deleteTask, la clase permite realizar diversas operaciones sobre las tareas, desde consultar la base de datos hasta insertar, actualizar o eliminar elementos.

En cuanto a la lógica de negocio, cuenta con funciones que validan y procesan los datos de las tareas, como isValidTask, que verifica si los datos de una tarea son válidos, o

isTaskCompleted, que determina su estado de finalización. Todo esto se complementa con la interacción directa con el TaskRepository, que facilita las operaciones CRUD sobre la base de datos.

- **TaskRVVBListAdapter**

La clase TaskRVVBListAdapter es un adaptador diseñado específicamente para un RecyclerView en Android, cuya función principal es mostrar una lista de tareas en formatos de vista tipo lista o cuadrícula. Este adaptador utiliza un constructor que recibe un objeto MutableLiveData para definir el formato de visualización y una función lambda que gestiona interacciones del usuario, como la eliminación o actualización de tareas.

Cuenta con dos clases internas: ListTaskViewHolder y GridTaskViewHolder, que manejan las vistas para los formatos de lista y cuadrícula, respectivamente. Ambas se basan en bindings generados automáticamente por Android Studio, lo que simplifica la configuración de las vistas con los datos de cada tarea. Además, implementa métodos fundamentales como onCreateViewHolder y onBindViewHolder para gestionar la creación y configuración de las vistas, así como getItemViewType para determinar el tipo de visualización según la posición. También incluye una clase interna, DiffCallback, que compara tareas para identificar cambios basados en sus identificadores y contenido.

Los diseños de las vistas están definidos en archivos de layout, como ViewTaskListLayoutBinding y ViewTaskGridLayoutBinding, lo que facilita la gestión del diseño de las interfaces.

8. Despliegue y pruebas

Para la compilación de fuentes de *PrioList* se puede clonar el siguiente repositorio o desde la carpeta drive proporcionada en la portada:

<https://github.com/chalinas/priolist>

Alberto Jesús Gómez Aranda

Una vez clonado hemos de abrir el proyecto con Android Studio y lanzar un emulador Android con una versión mínima de API 21 para ejecutar correctamente la aplicación. Esta versión se ha escogido principalmente para que el 95% de los dispositivos Android en el mercado sean capaces de soportar la ejecución de *PrioList*, así como dar soporte completo a Material Design.

Ya abierto el proyecto seleccionamos “Run” en la barra de tareas superior, el proyecto compilará y se lanzará automáticamente el emulador. Acto seguido se instalará *PrioList* y estará disponible para su uso.

En caso de no querer compilar el código fuente existe la opción de descargar de los mismos enlaces el archivo .apk e instalarlo en un dispositivo Android o arrastrar el archivo al emulador lo que instalará la aplicación automáticamente.

A continuación, se expondrán las pruebas de caja negra que se han ido realizando a cabo durante el desarrollo de *PrioList*.

CU-1 CREAR TAREA		
Objetivo probado	Requisitos probados	Pruebas a realizar
Creación de tareas correcta.	Se comprueba capacidad de crear tareas de manera correcta.	Arrancar la aplicación, introducir una tarea con los campos requeridos correctamente y comprobar que se crea.
Error “Required field”.	Task no admite tareas con título vacío.	Arrancar la aplicación, introducir una tarea, pero no rellenar el campo “Title”.
Error “Required field”.	Task no admite tareas con descripción vacía.	Arrancar la aplicación, introducir una tarea, pero no rellenar el campo “Description”.

CU-2 ACTUALIZAR TAREA		
Objetivo probado	Requisitos probados	Pruebas a realizar

Actualización de tareas correcta.	Se comprueba capacidad de actualizar tareas de manera correcta.	Pulsar en "Modify Task", introducir una tarea con los campos requeridos correctamente y comprobar que se actualiza.
Error "Required field".	Task no admite tareas con título vacío.	Arrancar la aplicación, introducir una tarea, pero no rellenar el campo "Title".
Error "Required field".	Task no admite tareas con descripción vacía.	Arrancar la aplicación, introducir una tarea, pero no rellenar el campo "Description".

CU-3 ELIMINAR TAREA

Objetivo probado	Requisitos probados	Pruebas a realizar
Eliminación de tareas correcta.	Se comprueba capacidad de eliminar tareas de manera correcta.	Pulsar en "Delete Task" en una tarea que este previamente creada.

CU-4 CAMBIAR VISTA

Objetivo probado	Requisitos probados	Pruebas a realizar
Se cambia la disposición de tareas completamente.	Se comprueba capacidad de cambiar la distribución de tareas en la vista general correctamente.	Pulsar en "Change View".
No se modifica la vista general si no existen tareas.	La vista general necesita de tareas creadas para modificar su distribución.	Pulsar en "Change View" sin ninguna tarea en la base de datos.

CU-2 ORDENAR TAREA

Objetivo probado	Requisitos probados	Pruebas a realizar
Ordenación de tareas correcta.	Se comprueba capacidad de ordenar las tareas de manera correcta según diferentes parámetros.	Pulsar en "Sort Task", introducir una tarea con los campos requeridos correctamente y comprobar que se actualiza.
Se actualiza automáticamente la vista general.	Los datos son siempre accesibles.	Cambiar la categoría de una tarea para ver si cambia en la vista general.
Se actualiza automáticamente la vista general.	Los datos son siempre accesibles.	Cambiar la prioridad de una tarea para ver si cambia en la vista general.
Se actualiza automáticamente la vista general.	Los datos son siempre accesibles.	Cambiar la hora de notificación de una tarea para ver si cambia en la vista general.

9. Conclusiones

El desarrollo de *PrioList* ha representado un punto de inflexión crucial en mi trayectoria profesional como desarrollador de software. Más allá de ser un proyecto meramente técnico, esta experiencia ha sido un viaje de crecimiento personal y profesional que ha redefinido mi enfoque hacia el desarrollo de aplicaciones multiplataforma. La inmersión simultánea en los roles de desarrollador back-end y front-end me ha proporcionado una perspectiva holística del proceso de construcción de software, consolidando mi metodología de resolución de problemas de manera integral y contextualizada.

La arquitectura del proyecto me ha obligado a adoptar un paradigma de pensamiento sistémico, donde cada decisión técnica se evalúa no solo por su implementación inmediata, sino por su potencial impacto en la escalabilidad y mantenibilidad del sistema. Esta aproximación me ha permitido desarrollar una capacidad analítica más sofisticada, anticipando posibles fricciones y diseñando soluciones proactivas antes de que se materialicen como limitaciones reales.

El lenguaje Kotlin ha sido sin duda el protagonista fundamental de este proyecto. Su filosofía de diseño parte de una premisa revolucionaria: simplificar la complejidad inherente al desarrollo de software sin comprometer la potencia y flexibilidad del código. Esta característica ha sido particularmente reveladora durante mi proceso de implementación, donde he podido constatar cómo Kotlin reduce la verbosidad típica de otros lenguajes manteniendo una sintaxis limpia y expresiva.

La curva de aprendizaje de Kotlin es notablemente suave. Aunque en *PrioList* no he podido explorar exhaustivamente todos los frameworks disponibles, la investigación preliminar ha despertado un interés significativo por profundizar en tecnologías como Ktor para desarrollo backend y Compose para interfaces multiplataforma.

El desarrollo de *PrioList* no ha estado exento de desafíos técnicos complejos. Uno de los aspectos más destacables ha sido la gestión de dependencias, un área que requiere un

Alberto Jesús Gómez Aranda

dominio preciso y una comprensión profunda de los mecanismos de procesamiento de anotaciones en Kotlin. La dicotomía entre KSP (Kotlin Symbol Processing) y KAPT (Kotlin Annotation Processing Tool) presentó inicialmente un escenario de considerable complejidad. Esta fase del proyecto se convirtió en una oportunidad para profundizar mi comprensión de los procesos de metaprogramación y generación de código.

Otro reto considerable fue la transformación de mockups iniciales en interfaces de usuario (XML) estéticamente coherentes y funcionalmente robustas. Como primera experiencia integral en diseño de interfaces para Android, esta etapa reveló mis áreas de mejora en diseño de experiencia de usuario. La traducción precisa de conceptos de diseño a implementaciones técnicas requirió un proceso que no puedo dar por finalizado dado que considero que es la mayor carencia que presenta a día de hoy el proyecto de cara al futuro.

PrioList no ha sido simplemente un proyecto de software, sino un viaje de transformación profesional. Ha consolidado mi convicción de que el verdadero valor de un desarrollador radica no solo en su capacidad técnica, sino en su habilidad para adaptar, aprender y evolucionar constantemente. Kotlin, con su filosofía de diseño centrada en la claridad y la eficiencia, ha sido un compañero ideal en este proceso de crecimiento.

Las lecciones aprendidas trascienden el ámbito puramente técnico, abarcando dimensiones de gestión de proyectos, diseño de arquitecturas de software y comunicación efectiva entre componentes tecnológicos. *PrioList* representa más que un proyecto culminado; es un camino que acaba de empezar y evolucionará constantemente como, al menos, campo de pruebas en mi formación continua.

10. Vías futuras

10.1. Objetivos que se plantearon en la propuesta, pero no se alcanzaron

Durante el periodo previo a la propuesta *PrioList* incluía conceptualmente la capacidad de soportar distintas vías de notificación al usuario, no solamente notificaciones push-up, si bien

estas son las más útiles queda a futuro añadir otras variaciones como la creación de alarmas para cada tarea.

10.2. Futuras mejoras de la aplicación

El aspecto más deficiente en la actualidad de *PrioList* como ya se expone anteriormente es su estética visual. Incluir un tema claro sería beneficioso para abarcar un mayor abanico de gustos de usuario, asimismo incluir paletas de colores para representar intuitivamente de manera visual las categorías a las que pertenecen las tareas.

Por otro lado, la aplicación se beneficiaría de insertar vistas y órdenes concretos que son aplicados de manera general en distintos sistemas de productividad. Esto supondría una inclusión de *PrioList* como herramienta complementaria a personas que hayan adoptado estos tipos de métodos. Los más interesante se incluirán en los próximos meses siendo:

- Bullet Journal: Asignando rangos de tiempo a cada tarea, ya sea día, semana, mes o año.
- Getting Things Done: Asignando tiempos de ejecución y mostrando las tareas que requieran menos de 3 minutos llevarlas a cabo en un lugar prioritario de la vista.
- Matriz Eisenhower: Asignando además de prioridad un rango de importancia y mostrando dichas tareas en una vista matricial.

Todas estas vistas han de ser activables y desactivables a petición del usuario para evitar desconcierto durante el uso de *PrioList*.

De la misma manera pivotar de una aplicación local a un sistema multidispositivo sería beneficioso a la hora de proveer al usuario de accesibilidad. Esto conlleva pivotar el back-end desde RoomDB a otro sistema de gestión de base de datos como Firebase y seguidamente implementar una pagina web o aplicación de escritorio para su uso multiplataforma.

Por último, se implementará un sistema de periodicidades personalizado para que ciertas tareas recurrentes se autogeneren en referencia a una tarea completada, ejemplos de esto puede ser hacer la compra o limpiar el arenero del gato. Esta utilidad pese a ser simple mejora

la experiencia de usuario ahorrando tiempo en tareas mecánicas como la introducción de datos en la aplicación.

11. Bibliografía/Webgrafía

1. *Cómo guardar contenido en una base de datos local con Room*. (s. f.). Android Developers. <https://developer.android.com/training/data-storage/room?hl=es-419>
2. *Migra de kapt a KSP*. (s. f.). Android Developers. <https://developer.android.com/build/migrate-to-ksp?hl=es-419#add-ksp>
3. google. (s. f.). *Releases · google/ksp*. GitHub. <https://github.com/google/ksp/releases>
4. *KSP quickstart / Kotlin*. (s. f.). Kotlin Help. <https://kotlinlang.org/docs/ksp-quickstart.html#pass-options-to-processors>
5. «cannot resolve symbol R» in Android Studio. (s. f.). Stack Overflow. <https://stackoverflow.com/questions/17054000/cannot-resolve-symbol-r-in-android-studio/30676933>
6. *Room cannot verify the data integrity. Looks like you've changed schema but forgot to update the version number*. (s. f.). Stack Overflow. <https://stackoverflow.com/questions/60649376/room-cannot-verify-the-data-integrity-looks-like-youve-changed-schema-but-forg>
7. *Why am I getting this warning and how to resolve it «This version only understands SDK XML versions up to 2 but an SDK XML. . .»* (s. f.). Stack Overflow. <https://stackoverflow.com/questions/70368175/why-am-i-getting-this-warning-and-how-to-resolve-it-this-version-only-understan>

8. Philipp Lackner. (2023, 15 marzo). *The FULL Beginner Guide for Room in Android / Local Database Tutorial for Android* [Vídeo]. YouTube.
<https://www.youtube.com/watch?v=bOd3wO0uFr8>
9. Ramos, J. (2021, 22 enero). *Android: ¿Qué es MVC, MVP y MVVM?* Programación y Más. <https://programacionymas.com/blog/android-mvc-mvp-mvvm>
10. *Cómo crear un diseño basado en tarjetas*. (s. f.). Android Developers.
<https://developer.android.com/develop/ui/views/layout/cardview?hl=es-419>
11. Universitat Politècnica de València - UPV. (2022, 24 febrero). *Casos de uso y diagramas de casos de uso / / UPV* [Vídeo]. YouTube.
<https://www.youtube.com/watch?v=iFcDoP6jEeE>

12. Anexos

12.1. Manual de Instalación

Requisitos del Sistema

Especificaciones de compatibilidad de sistema operativo:

- Plataforma Android
- Versión mínima soportada: Android 5.0 (API level 21)

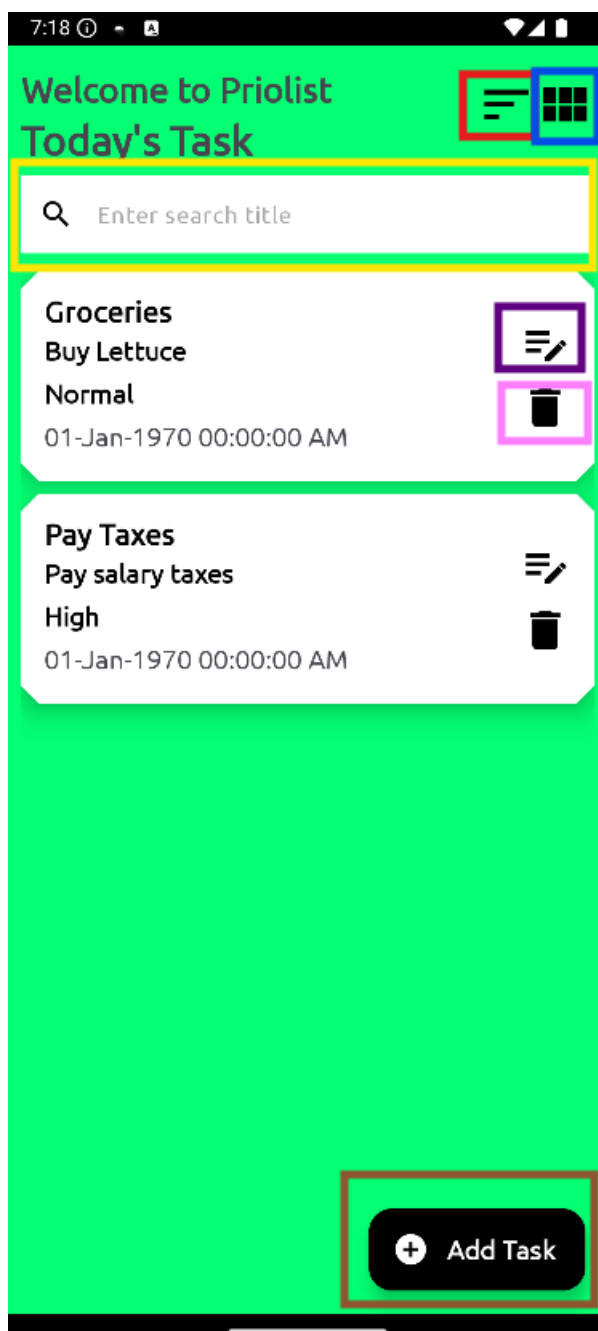
Especificaciones técnicas para dispositivos móviles:

- Memoria RAM: 2GB
 - Garantiza rendimiento fluido de la aplicación
 - Permite ejecución simultánea de procesos en segundo plano
 - Suficiente para cargar y gestionar múltiples tareas
- Almacenamiento Interno: 50MB
 - Espacio compacto para la instalación de la aplicación
 - Diseño eficiente que minimiza el uso de recursos
 - Permite almacenamiento local de tareas y configuraciones

Si su dispositivo cumple con las características mencionadas puede instalar *PrioList* mediante la descarga del archivo .apk en el enlace <https://www.github.com/chalinas/priolist>. Posteriormente diríjase a la carpeta de descargas de su dispositivo móvil y seleccione el archivo. Asegúrese de seleccionar “Permitir instalación desde fuentes desconocidas” en su sección de ajustes.

12.2. Manual de Usuario

Vista General



Podemos observar los distintos botones y campos de acción en esta ilustración.

ROJO → Botón “Sort Task” utilizado para introducir el parámetro por el que ordenar la vista.

AZUL → Botón “Change View” utilizado para cambiar la disposición de tareas en la vista.

AMARILLO → Barra de texto para buscar tarea por título.

VIOLETA → Botón “Modify Task” para cambiar las propiedades de una tarea

ROSA → Botón “Delete Task” para eliminar una tarea.

MARRÓN → Botón “Add Task” para agregar una tarea.

Ilustración 19. Colorización de acciones

Diálogo de creación/modificación de tareas

The image shows a 'New Task' dialog box with several color-coded annotations:

- PÚRPURA**: A box around the close button (X) in the top right corner.
- ROJO**: A box around the 'Enter the task title' text input field.
- AZUL**: A box around the 'Enter the task description' text input field.
- VIOLETA**: A box around the priority dropdown menu showing 'High'.
- AMARILLO**: A box around the category dropdown menu showing 'Personal'.
- MARRÓN**: A box around the 'Set Notification Time' button.
- ROSA**: A box around the 'Save task' button.

Ilustración 20. Colorización de acciones

PÚRPURA → Botón “Cancel Changes” para volver a la vista general sin ninguna modificación.

ROJO → Barra de texto para introducir el título de la tarea.

AZUL → Barra de texto para introducir la descripción de la tarea.

VIOLETA → Desplegable para seleccionar la prioridad de la tarea.

AMARILLO → Desplegable para seleccionar la categoría deseada y dado el caso introducir una nueva.

MARRÓN → Botón “Set Notification Time” para agregar una notificación a una hora determinada a la tarea.

ROSA → Botón “Save Task” para guardar una tarea.

PrioList: Gestor de tareas eficaz

Alberto Jesús Gómez Aranda

ILERNA.

