

12 DE NOVIEMBRE DE 2025

# Lanzador De Apps PSP

2ºDAM

David Márquez López  
IES ARMANDO COTARELO VALLEDOR  
Tutor: Daniel Resúa Melón

## Contenido

1. Introducción.....	2
1.1 Descripción del Proyecto .....	2
1.2 Objetivos Específicos.....	2
2. Descripción de tecnologías usadas y diseño de la arquitectura del sistema ..	3
2.1 Tecnologías Utilizadas .....	3
2.2. Diseño de la Arquitectura del Sistema .....	4
3. Descripción de las funcionalidades con capturas del proceso .....	6
3.1. Interfaz Principal y Detección del Sistema .....	6
3.2. Escaneo del Sistema .....	7
3.3. Visualización y Listado de Aplicaciones.....	8
3.4. Búsqueda, Filtrado y Ordenación.....	9
3.5. Lanzamiento de Aplicaciones (ProcessBuilder) .....	11
3.6. Funcionalidades Adicionales: Añadir Manualmente y Abrir Ubicación...	11
4. Manual de usuario.....	12
4.1. Requisitos Previos e Instalación .....	12
4.2. Primeros Pasos: Escanear el Sistema .....	13
4.3. Interactuar con la Lista de Aplicaciones.....	13
4.4. Búsqueda y Filtrado .....	13
4.5. Añadir una Aplicación Manualmente .....	14
5. Pruebas .....	14
5.1. Pruebas Funcionales .....	15
5.2. Pruebas de Interfaz de Usuario (UI) y Filtrado.....	16
5.3. Resumen de Pruebas .....	17
6. Conclusiones y dificultades encontradas .....	17
6.1. Conclusiones.....	17
6.2. Dificultades Encontradas .....	18
7. Referencias .....	18
8. Anexos .....	19
Anexo A: Repositorio de Código Fuente .....	19
Anexo B: Uso de Herramientas de Inteligencia Artificial en el Proyecto .....	19

# 1. Introducción.

## 1.1 Descripción del Proyecto

El presente proyecto aborda el desarrollo de una aplicación de escritorio denominada "Lanzador de Aplicaciones". El propósito fundamental de esta herramienta es proporcionar al usuario una interfaz centralizada y eficiente desde la cual puede escanear, visualizar y ejecutar las distintas aplicaciones y juegos instalados en su sistema operativo.

La solución se ha implementado íntegramente en el lenguaje de programación Kotlin, aprovechando el *framework* declarativo Compose for Desktop para la construcción de una interfaz de usuario moderna y reactiva. Esta herramienta resuelve la problemática habitual de la dispersión de ejecutables y accesos directos, unificando el acceso en un único catálogo filtrable y ordenable que detecta automáticamente el software instalado.

## 1.2 Objetivos Específicos

Para guiar el desarrollo y asegurar que el producto final cumpla con las expectativas, se establecieron los siguientes requisitos y objetivos específicos:

- Implementar un mecanismo de escaneo del sistema capaz de detectar automáticamente las aplicaciones y juegos instalados, con lógicas diferenciadas para Windows y Linux.
- Proporcionar la funcionalidad de lanzar cualquier aplicación seleccionada desde la interfaz con un solo clic, utilizando para ello la clase `java.lang.ProcessBuilder`.
- Mostrar información básica y relevante de cada aplicación, incluyendo su nombre, el ícono nativo del sistema y la ruta completa del ejecutable.
- Incluir una opción para que el usuario pueda agregar manualmente aplicaciones que no hayan sido detectadas por el escaneo automático, mediante un selector de archivos (`java.awt.FileDialog`).

- Implementar un manejo de errores robusto (mediante bloques try-catch) para gestionar situaciones anómalas, como rutas de ejecutables inválidas o fallos durante el inicio del proceso externo.
- Desarrollar una interfaz de usuario clara que permita al usuario buscar, filtrar (Todas, Sistema, Usuario) y ordenar (A-Z, Z-A) la lista de aplicaciones.

## 2. Descripción de tecnologías usadas y diseño de la arquitectura del sistema

### 2.1 Tecnologías Utilizadas

El desarrollo del proyecto se ha fundamentado en un conjunto de tecnologías modernas y robustas, seleccionadas para cumplir con los requisitos de multiplataforma, rendimiento y acceso nativo al sistema.

- **Kotlin:** Es el lenguaje de programación principal utilizado para la totalidad del proyecto. Su sintaxis concisa, seguridad frente a nulos (null-safety) y excelente interoperabilidad con Java lo convierten en la base sobre la que se construye la aplicación.
- **Compose for Desktop:** Se ha empleado este *framework* de UI declarativo y moderno para construir toda la interfaz gráfica de usuario. Permite crear interfaces reactivas y estéticamente consistentes con un código más limpio y mantenable en comparación con *frameworks* tradicionales. Su uso es central en el archivo App.kt y en el punto de entrada main.kt.
- **Corrutinas de Kotlin:** Se han utilizado para gestionar las tareas de larga duración que no deben bloquear el hilo principal de la interfaz de usuario (UI Thread). Específicamente, el escaneo del sistema (Scanner.escanearJuegos) se invoca dentro de un scope.launch y se ejecuta en el despachador de Entrada/Salida (Dispatchers.IO), evitando que la aplicación se congele durante la búsqueda de archivos.
- **java.lang.ProcessBuilder:** Tal como exigen los requisitos, esta clase de Java se utiliza para el lanzamiento de los procesos externos. Es el

mecanismo que ejecuta el archivo .exe o el comando de Linux correspondiente a la aplicación seleccionada por el usuario.

- **java.io.File:** Esta API clásica de Java es fundamental para la lógica de escaneo. Se utiliza para instanciar, recorrer y validar los directorios del sistema (como C:\Program Files o /usr/share/applications) y para verificar la existencia y propiedades de los archivos ejecutables.
- **AWT (java.awt) y Swing (javax.swing):** Aunque Compose for Desktop gestiona la UI principal, se ha recurrido a componentes de AWT y Swing para funcionalidades específicas de interacción con el sistema operativo nativo que Compose aún no provee:
  - java.awt.FileDialog se utiliza para mostrar el diálogo nativo de "Abrir archivo", permitiendo al usuario añadir una aplicación manualmente.
  - javax.swing.filechooser.FileSystemView y javax.swing.ImageIcon se emplean en la lógica de IconExtractor.kt para obtener el ícono asociado a un tipo de archivo (especialmente .exe en Windows) directamente del sistema operativo.
  - java.awt.image.BufferedImage y java.awt.Graphics2D se usan para el procesamiento y escalado de alta calidad de los iconos extraídos.

## 2.2. Diseño de la Arquitectura del Sistema

El proyecto se ha estructurado siguiendo un patrón claro de separación de responsabilidades (Separation of Concerns), donde cada archivo o grupo de archivos tiene un propósito definido, facilitando la mantenibilidad y la escalabilidad.

- **Punto de Entrada (main.kt)** La ejecución de la aplicación comienza en main.kt, cuya única función es configurar y lanzar la ventana principal de la aplicación (androidx.compose.ui.window.Window) y albergar el componente raíz de la interfaz, App.

- **Núcleo de la Aplicación (UI y Estado - App.kt)** Este es el archivo central de la aplicación. Contiene el composable @Composable fun App, que define toda la estructura de la interfaz de usuario: el encabezado, los botones, el campo de búsqueda, los menús desplegables de filtro y la lista de aplicaciones. Además, gestiona el estado de la aplicación (la lista de juegos, el texto de búsqueda, el filtro actual, etc.) utilizando delegados de estado de Compose (mutableStateOf y remember). También gestiona todos los eventos de interacción del usuario, como los clics en los botones, que disparan la lógica de escaneo o el lanzamiento de procesos.
- **Lógica de Detección (Servicios - DetectorSO.kt y Scanner.kt)** Estos archivos contienen la lógica de negocio principal:
  - DetectorSO.kt: Es un *singleton* (object) que detecta el sistema operativo actual (Windows o Linux) de forma diferida (by lazy) al arrancar la aplicación, basándose en la propiedad del sistema "os.name".
  - Scanner.kt: Es el servicio de escaneo. Su método principal, escanearJuegos(), consulta primero a DetectorSO.actual y, en función del resultado, delega en escanearWindows() o escanearLinux(). Estos métodos contienen la lógica específica de cada plataforma para encontrar las aplicaciones (búsqueda recursiva de .exe en Windows o parseo de archivos .desktop en Linux).
- **Gestión de Iconos (Utilidades - IconExtractor.kt e IconUtils.kt)** Este par de archivos se especializa en la compleja tarea de obtener los iconos nativos de las aplicaciones:
  - IconExtractor.kt: Contiene la lógica de bajo nivel para interactuar con las APIs de Swing (FileSystemView) en Windows y para buscar en las rutas estándar de iconos (/usr/share/icons) en Linux. También incluye funciones avanzadas para escalar las imágenes con alta calidad.

- `IconUtils.kt`: Actúa como un puente entre `IconExtractor` y la UI de Compose. Proporciona funciones suspend que ejecutan la extracción de iconos en un hilo de `Dispatchers.IO` y convierten la imagen resultante (ya sea un `BufferedImage` de Java o un `ByteArray`) en un `ImageBitmap` compatible con Compose, implementando además un sistema de caché simple (`iconCache`) para mejorar el rendimiento.
- **Archivos Auxiliares (`Greeting.kt` y `Platform.kt`)** Estos archivos, generados habitualmente por la plantilla de Kotlin Multiplatform, definen clases (`Greeting`, `JVMPlatform`) que identifican la plataforma actual, aunque no son críticos para la lógica principal del lanzador de aplicaciones.

### 3. Descripción de las funcionalidades con capturas del proceso

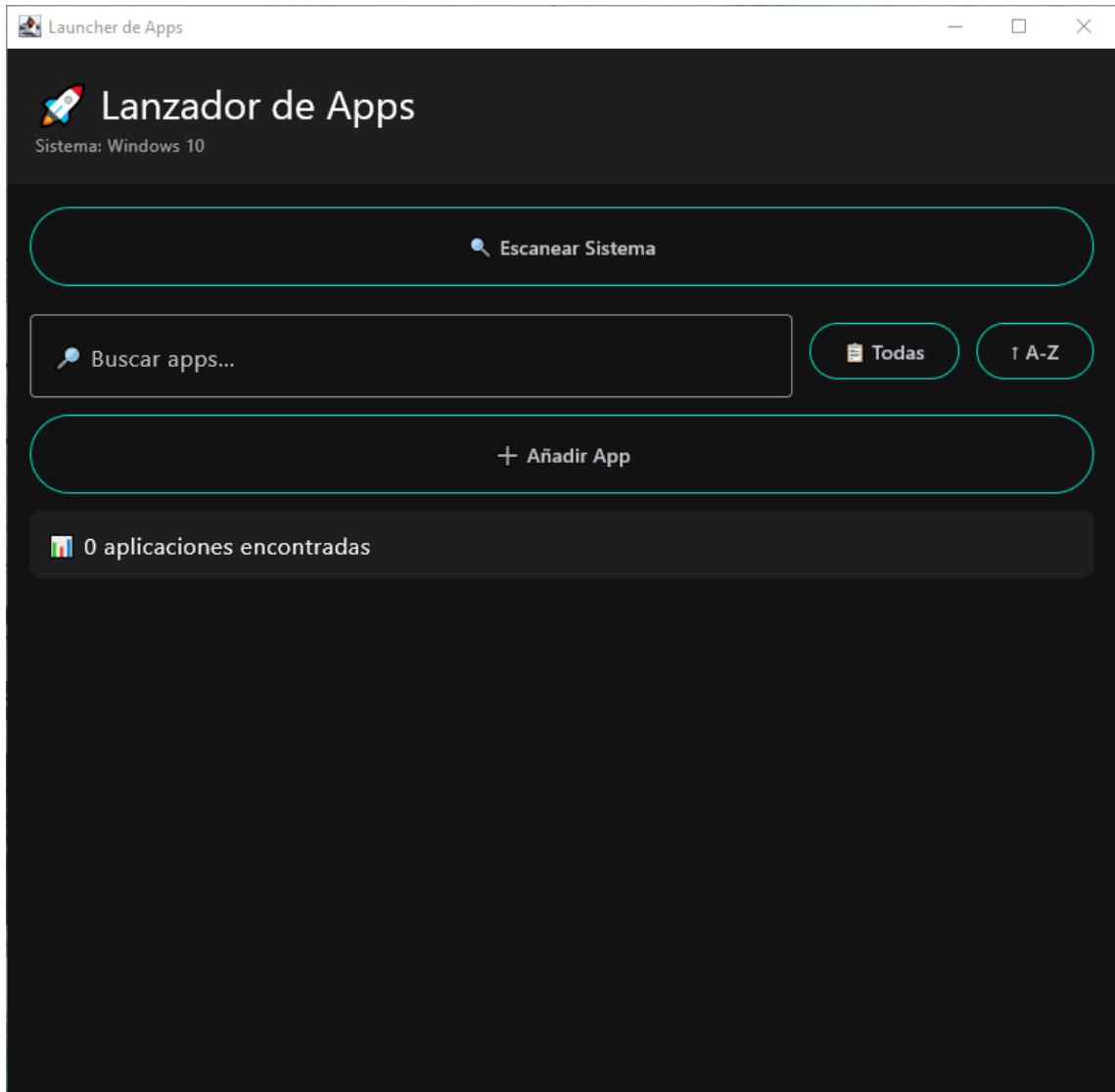
A continuación, se detallan las funcionalidades principales de la aplicación, ilustrando cada una con capturas de pantalla de su funcionamiento.

#### 3.1. Interfaz Principal y Detección del Sistema

Al iniciar la aplicación, el usuario es recibido por una interfaz limpia y organizada. En la parte superior, un encabezado prominente muestra el título " Lanzador de Apps" y, justo debajo, detecta y muestra automáticamente el sistema operativo anfitrión (en este caso, detectado a través de `DetectorSO.osName`), proporcionando una confirmación visual inmediata de la plataforma.

La parte central de la aplicación contiene los controles principales: el botón de escaneo, la barra de búsqueda y los filtros, que se describen en los siguientes apartados.

*Imagen 1: Vista principal de la aplicación al iniciarse.*

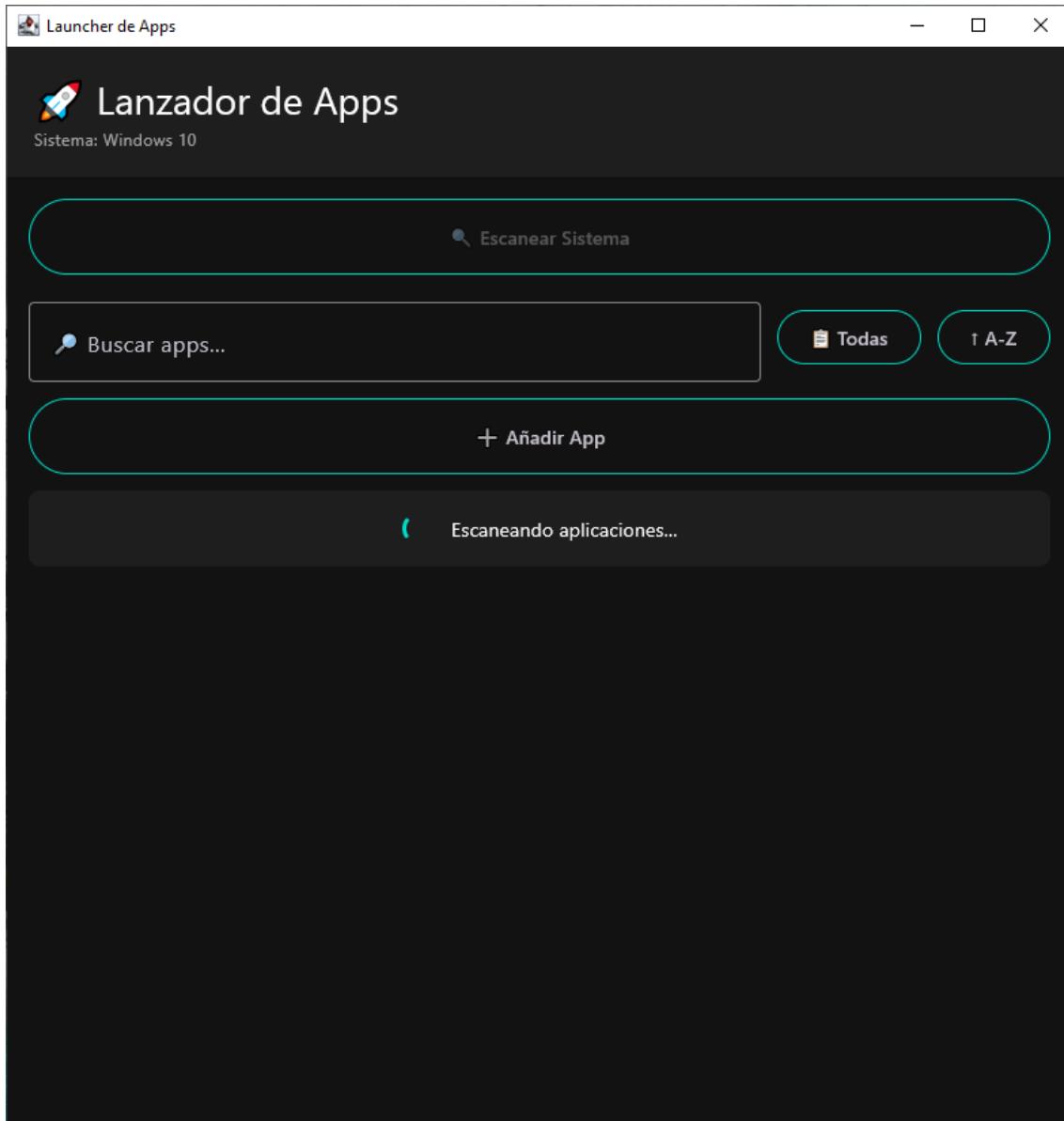


### 3.2. Escaneo del Sistema

La funcionalidad central de la aplicación se inicia pulsando el botón "🔍 Escanear Sistema". Al hacerlo, se lanza una corutina en un hilo de Dispatchers.IO que invoca al Scanner.kt. Este módulo aplica la lógica de detección correspondiente (búsqueda recursiva en Windows o parseo de .desktop en Linux).

Mientras el escaneo está en progreso, el botón se deshabilita y aparece un indicador de carga (CircularProgressIndicator) con el texto "Escaneando aplicaciones...", proporcionando retroalimentación visual al usuario y evitando acciones concurrentes. Una vez completado, el indicador desaparece y la lista se puebla con los resultados, mostrando un contador del total de aplicaciones encontradas.

Imagen 2: Indicador de carga y estado "Escaneando aplicaciones..." activo.



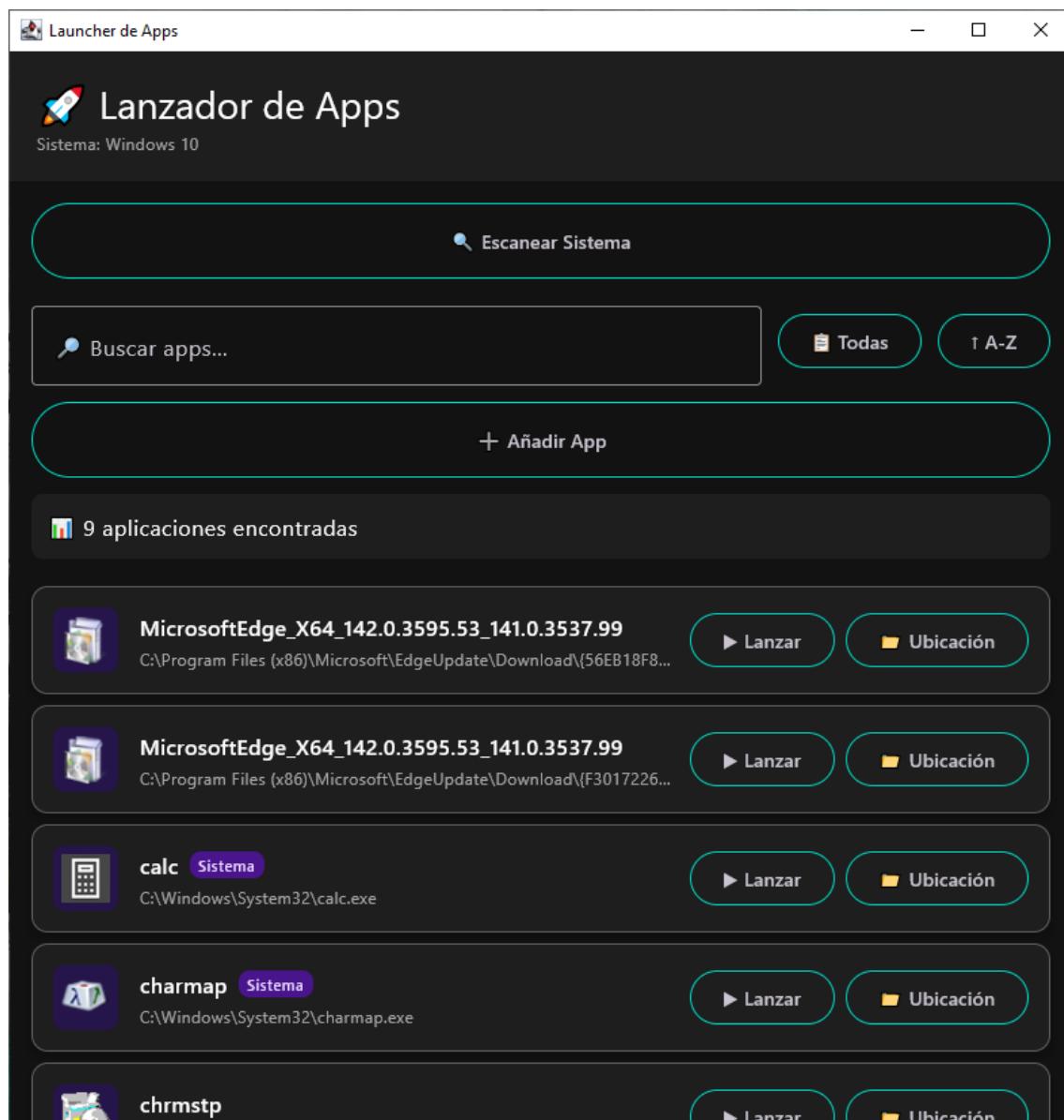
### 3.3. Visualización y Listado de Aplicaciones

Las aplicaciones detectadas se muestran en una lista vertical (LazyColumn), que garantiza un rendimiento óptimo incluso con cientos de elementos. Cada fila de la lista está diseñada para mostrar la información clave de la aplicación:

- **Icono:** Se extrae el ícono nativo del sistema gracias a IconUtils.kt, proporcionando un identificador visual claro.
- **Nombre:** El nombre del ejecutable o el nombre definido en el archivo .desktop.

- **Etiqueta "Sistema":** Si la aplicación se detecta como parte del sistema (isSystemApp = true), se muestra una pequeña etiqueta distintiva.
- **Ruta:** La ruta completa del ejecutable, con un sistema de elipsis (...) para evitar desbordamientos de texto.

*Imagen 3: Lista de aplicaciones detectadas, mostrando iconos, nombres, rutas y etiquetas.*

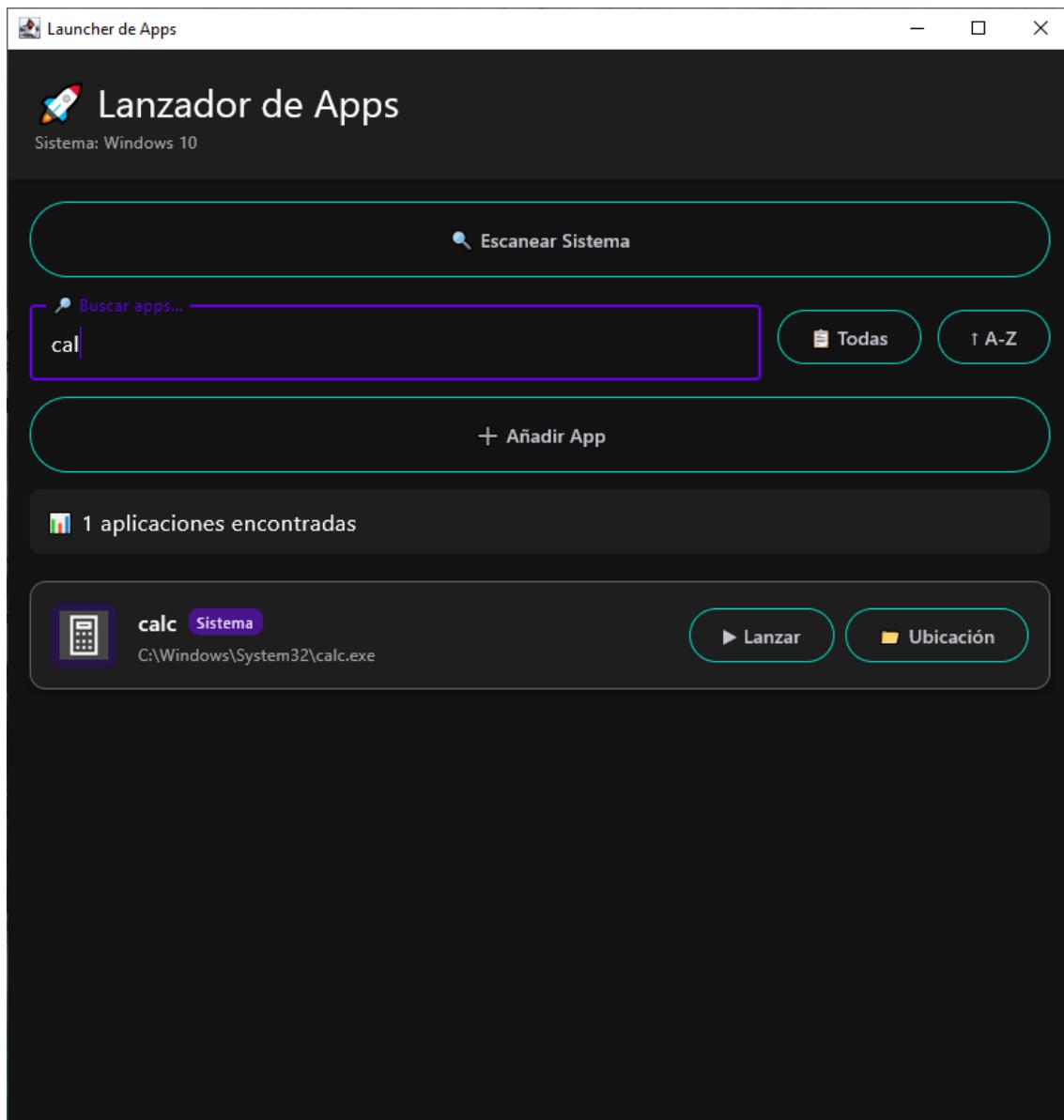


### 3.4. Búsqueda, Filtrado y Ordenación

Para gestionar listas extensas, la interfaz incluye potentes herramientas de refinamiento:

- **Búsqueda (OutlinedTextField):** Un campo de texto permite al usuario filtrar la lista en tiempo real. La lógica de filtrado es *case-insensitive* (no distingue mayúsculas de minúsculas).
- **Filtro de Tipo (DropdownMenu):** Permite mostrar "Todos" las aplicaciones, solo las de "Sistema" o solo las de "Usuario".
- **Ordenación (DropdownMenu):** Permite ordenar la lista alfabéticamente de forma ascendente ("↑ A-Z") o descendente ("↓ Z-A").

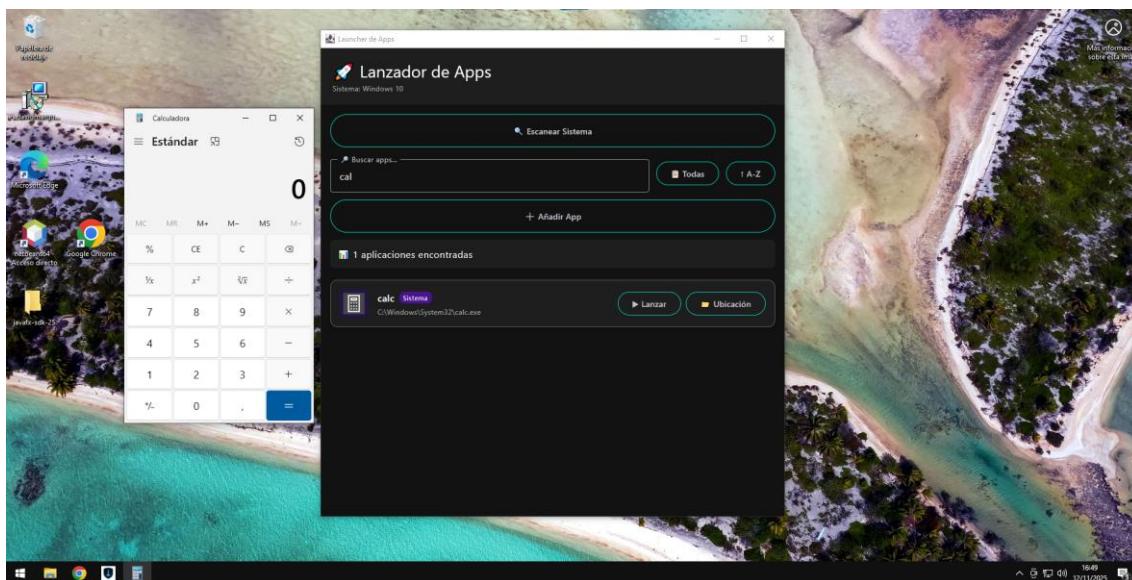
Estos estados se combinan para actualizar la lista mostrada (listaFiltrada) de forma reactiva cada vez que cambia un parámetro. *Ejemplo de filtrado por búsqueda (p.ej., "calc") y el menú de ordenación desplegado.*



### 3.5. Lanzamiento de Aplicaciones (ProcessBuilder)

El objetivo principal de la aplicación se cumple con el botón "▶ Lanzar". Al pulsarlo, se ejecuta el siguiente fragmento de código, que instancia ProcessBuilder con la ruta de la aplicación (juego.ruta) y llama a .start() para iniciar el proceso externo. La operación está envuelta en un bloque try-catch (IOException) para manejar cualquier error que pueda ocurrir si el archivo no existe o falla la ejecución.

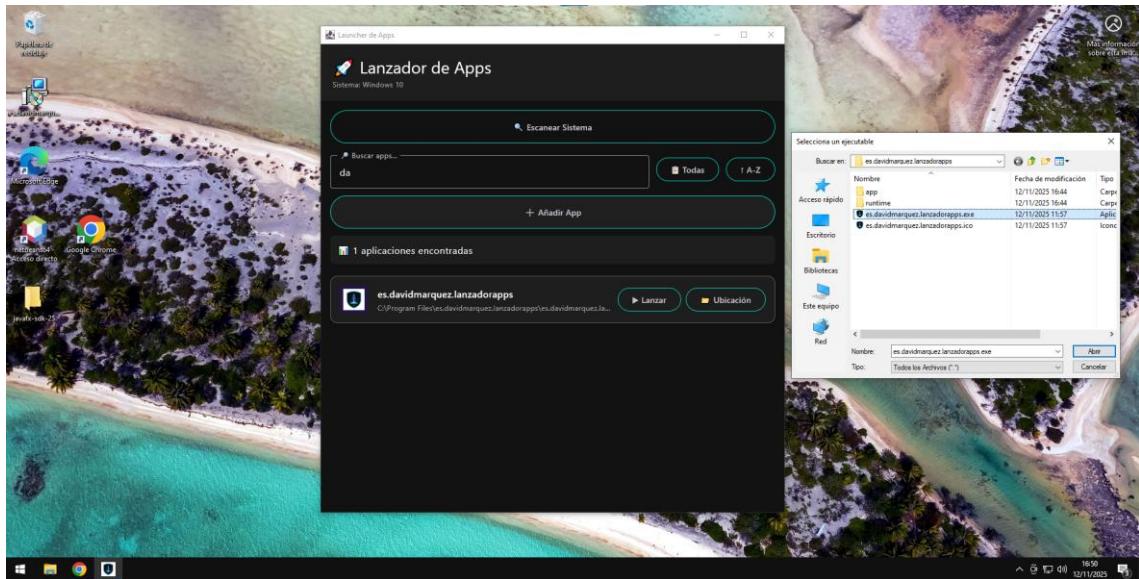
*Imagen 5: Ejecución de 'calc.exe' tras pulsar el botón "Lanzar".*



### 3.6. Funcionalidades Adicionales: Añadir Manualmente y Abrir Ubicación

- **Añadir App:** El botón "✚ Añadir App" abre un diálogo nativo del sistema (java.awt.FileDialog). El usuario puede navegar por sus directorios y seleccionar un archivo ejecutable. Una vez seleccionado, la aplicación se añade a la lista principal como una aplicación de usuario (isSystemApp = false).
- **Abrir Ubicación:** El botón "📁 Ubicación" utiliza la clase java.awt.Desktop para abrir el explorador de archivos del sistema directamente en la carpeta que contiene el ejecutable (file.parentFile), facilitando el acceso al directorio de instalación.

*Imagen 6: Proceso de añadir una aplicación manualmente mediante el selector de archivos.*



## 4. Manual de usuario

Este manual describe los pasos básicos para utilizar la aplicación "Lanzador de Apps".

### 4.1. Requisitos Previos e Instalación

La aplicación se distribuye como un paquete de instalación nativo que incluye todas las dependencias necesarias. **No se requiere tener Java instalado en el sistema.**

1. Visite la sección "Releases" del repositorio del proyecto en GitHub:  
<https://github.com/davsark/LanzadorApps>
2. Descargue el archivo de instalación adecuado para su sistema operativo.
  - **En Windows:** Descargue el archivo lanzador-de-apps.msi (o similar). Ejecute el instalador y siga las instrucciones en pantalla. Una vez finalizado, la aplicación estará lista para usarse.
  - **En Linux (basado en Debian/Ubuntu):**
    1. Descargue el archivo lanzador-de-apps-linux.deb.
    2. Haga doble clic en el archivo .deb para abrirlo con el instalador de software de su sistema (como "Software" en Ubuntu o "Gdebi").

3. Pulse el botón "Instalar" y, si se le solicita, introduzca su contraseña de administrador.
4. (Alternativa por terminal): Navegue a la carpeta de descargas y ejecute: sudo dpkg -i lanzador-de-apps-linux.deb
3. Una vez instalado, busque "Lanzador de Apps" en el menú de aplicaciones de su sistema.

## 4.2. Primeros Pasos: Escanear el Sistema

La primera vez que abra la aplicación, la lista de aplicaciones estará vacía.

1. Haga clic en el botón " Escanear Sistema".
2. La aplicación mostrará un indicador de carga mientras busca aplicaciones en los directorios estándar de su sistema operativo.
3. Una vez finalizado el escaneo, la lista se llenará automáticamente con todas las aplicaciones detectadas. El contador sobre la lista (p.ej., "50 aplicaciones encontradas") le indicará el total.

## 4.3. Interactuar con la Lista de Aplicaciones

Tras el escaneo, verá una lista con todas las aplicaciones.

- **Lanzar una aplicación:** Para ejecutar un programa, búsqelo en la lista y haga clic en el botón " Lanzar". La aplicación se iniciará como un proceso separado.
- **Abrir la ubicación:** Si desea ver dónde está instalada una aplicación, haga clic en el botón " Ubicación". Se abrirá el explorador de archivos de su sistema en la carpeta que contiene el ejecutable.

## 4.4. Búsqueda y Filtrado

Si tiene muchas aplicaciones, puede usar las herramientas de organización:

- **Buscar:** Escriba el nombre de una aplicación en el campo de texto " Buscar apps..." para filtrar la lista en tiempo real.

- **Filtrar por tipo:** Use el menú desplegable (que por defecto dice "  Todas") para mostrar únicamente aplicaciones de "  Sistema" o de "  Usuario".
- **Ordenar:** Use el menú desplegable (por defecto " $\uparrow$  A-Z") para cambiar el orden alfabético de ascendente a descendente (" $\downarrow$  Z-A").

#### 4.5. Añadir una Aplicación Manualmente

Si el escáner no detectó una aplicación (por ejemplo, una aplicación *portable* o instalada en una ruta no estándar), puede añadirla manualmente:

1. Haga clic en el botón "  Añadir App".
2. Se abrirá una ventana de diálogo de su sistema operativo ("Selecciona un ejecutable").
3. Navegue por sus carpetas, seleccione el archivo ejecutable (p.ej., juego.exe o un binario en Linux) y pulse "Abrir".
4. La aplicación se añadirá inmediatamente a la lista y será clasificada como aplicación de "Usuario".

### 5. Pruebas

#### 5. Pruebas

Para asegurar la calidad y el correcto funcionamiento de la aplicación, se ha llevado a cabo un conjunto de pruebas funcionales y de usabilidad en los sistemas operativos objetivo: Windows 10, Windows 11 y una distribución de Linux (Ubuntu 22.04).

A continuación, se detallan las pruebas realizadas, los resultados esperados y los resultados obtenidos.

## 5.1. Pruebas Funcionales

Prueba	Funcionalidad	Método de Prueba	Resultado Esperado	Resultado Obtenido
P-01	Detección de SO (DetectorSO.kt)	Iniciar la aplicación en W10, W11 y Linux.	La UI debe mostrar "Sistema: Windows..." o "Sistema: Linux..." correctamente.	<b>Superada</b>
P-02	Escaneo (Windows) (Scanner.kt)	Pulsar "Escanear" en W10 y W11.	La lista se puebla con .exe de Program Files y System32 (ej: calc.exe, notepad.exe).	<b>Superada</b>
P-03	Escaneo (Linux) (Scanner.kt)	Pulsar "Escanear" en Linux.	La lista se puebla con apps de .desktop (ej: Firefox, Terminal, Editor de Texto).	<b>Superada</b>
P-04	Lanzamiento de App (ProcessBuilder)	Pulsar "► Lanzar" en varias apps (ej: mspaint.exe en W10/W11, gedit en Linux).	La aplicación seleccionada debe iniciarse correctamente en el sistema operativo.	<b>Superada</b>
P-05	Añadir App Manual (FileDialog)	Pulsar "+ Añadir App", seleccionar un ejecutable no listado.	La nueva app aparece en la lista, clasificada como "Usuario" y es lanzable.	<b>Superada</b>

Prueba	Funcionalidad	Método de Prueba	Resultado Esperado	Resultado Obtenido
P-06	Abrir Ubicación (Desktop.open)	Pulsar "Ubicación" en una app.	Se abre el explorador de archivos del sistema en la carpeta que contiene el ejecutable.	<b>Superada</b>

## 5.2. Pruebas de Interfaz de Usuario (UI) y Filtrado

Estas pruebas validan la reactividad y la usabilidad de la interfaz desarrollada.

Prueba	Funcionalidad	Método de Prueba	Resultado Esperado	Resultado Obtenido
P-07	Búsqueda en tiempo real (App.kt)	Escribir un término (ej: "note") en la barra "Buscar apps...".	La lista se filtra instantáneamente para mostrar solo las apps que contienen "note" en el nombre.	<b>Superada</b>
P-08	Filtro de Tipo (Sistema/Usuario)	Seleccionar "Sistema" en el desplegable.	La lista muestra solo las apps marcadas con la etiqueta "Sistema".	<b>Superada</b>
P-09	Filtro de Tipo (Usuario)	Añadir una app manualmente y seleccionar "Usuario".	La lista muestra solo la app añadida manualmente (y otras de usuario si las hubiera).	<b>Superada</b>

Prueba	Funcionalidad	Método de Prueba	Resultado Esperado	Resultado Obtenido
P-10	Ordenación (A-Z / Z-A)	Cambiar el orden en el desplegable de "↑ A-Z" a "↓ Z-A".	El orden de la lista de aplicaciones se invierte alfabéticamente.	<b>Superada</b>
P-11	Extracción de Iconos (IconUtils.kt)	Revisar visualmente la lista tras el escaneo.	Las aplicaciones muestran sus iconos nativos (de .exe en Win, de tema en Linux), no el icono genérico.	<b>Superada</b>
P-12	Estado de Carga (Escaneo)	Pulsar "Escanear" y observar la UI.	El botón se deshabilita y aparece el CircularProgressIndicator hasta que finaliza el escaneo.	<b>Superada</b>

### 5.3. Resumen de Pruebas

Todas las pruebas funcionales y de interfaz de usuario definidas han sido superadas con éxito en los entornos de prueba (Windows 10, Windows 11 y Ubuntu 22.04). El comportamiento de la aplicación es el esperado y cumple con todos los requisitos especificados en el enunciado del proyecto.

## 6. Conclusiones y dificultades encontradas

### 6.1. Conclusiones

El proyecto ha cumplido exitosamente sus objetivos, entregando un lanzador de aplicaciones funcional y probado en Windows (10 y 11) y Linux. La elección de Kotlin junto a Compose for Desktop demostró ser ideal para construir una interfaz de usuario moderna y reactiva, facilitando la gestión del estado. La arquitectura,

separando claramente la UI (App.kt), el escaneo (Scanner.kt) y la extracción de iconos (IconExtractor.kt), fue clave para la mantenibilidad del proyecto.

## 6.2. Dificultades Encontradas

Durante el desarrollo surgieron varios desafíos técnicos significativos:

1. **Configuración del Entorno:** Se invirtió tiempo considerable en resolver conflictos entre versiones de JDK y en estabilizar las dependencias de Gradle para asegurar un entorno de compilación estable.
2. **Extracción de Iconos de Alta Calidad:** Fue el reto más complejo. En Windows, implicó un trabajo minucioso con librerías AWT/Swing para extraer los iconos de .exe en alta resolución (64x64) y no solo los de baja calidad (16x16), requiriendo un escalado cuidadoso. En Linux, supuso replicar la lógica de búsqueda de iconos en los directorios de temas del sistema.
3. **Filtrado del Escáner:** Diseñar una heurística eficaz para el Scanner.kt fue difícil. Se necesitó un proceso iterativo para definir las reglas (listas de palabras ignoradas, rutas excluidas y tamaño mínimo) para desechar "aplicaciones basura" (como instaladores, *updaters* o *plugins*) y listar solo los ejecutables principales.
4. **Interoperabilidad Compose-AWT:** Integrar componentes de AWT (como el FileDialog para la adición manual) dentro del *framework* declarativo de Compose requirió una gestión específica para que ambos sistemas de UI coexistieran correctamente.

## 7. Referencias

- [1] ResuaCode (PSP), "Programación de Servicios y Procesos – documentación," [resuacode.es](https://resuacode.es/psp/), 2025. [En línea]. Disponible: <https://resuacode.es/psp/> [Accedido: nov. 2025].
- [2] JetBrains, "Compose Multiplataforma," [JetBrains.com](https://www.jetbrains.com/es-es/compose-multiplatform/), 2025. [En línea]. Disponible: <https://www.jetbrains.com/es-es/compose-multiplatform/>. [Accedido: 12-nov-2025].

[3] JetBrains, "Documentación oficial de Kotlin," [kotlinlang.org](https://kotlinlang.org/docs/home.html), 2025. [En línea]. Disponible: <https://kotlinlang.org/docs/home.html>. [Accedido: 12-nov-2025].

[4] Oracle, "Java SE 17 & JDK 17 API Documentation," [docs.oracle.com](https://docs.oracle.com/en/java/javase/17/docs/api/), 2025. [En línea]. Disponible: <https://docs.oracle.com/en/java/javase/17/docs/api/>. [Accedido: 12-nov-2025].

[5] Stack Exchange, Inc., "Stack Overflow," *Sitio web de la comunidad*, 2025. [En línea]. Disponible: <https://stackoverflow.com>. [Accedido: 12-nov-2025].

## 8. Anexos

### Anexo A: Repositorio de Código Fuente

Tal como se requiere para los proyectos de desarrollo de software, el código fuente completo de esta aplicación está disponible públicamente en el siguiente repositorio de GitHub:

<https://github.com/davsark/LanzadorApps>

### Anexo B: Uso de Herramientas de Inteligencia Artificial en el Proyecto

#### B.1 Descripción de las herramientas de IA utilizadas

Se empleó un conjunto de herramientas de IA de forma complementaria:

- **Asistentes de Código** (ej. GitHub Copilot, IntelliJ AI Assistant): Integrados en el IDE (IntelliJ IDEA) para la sugerencia y autocompletado de fragmentos de código Kotlin en tiempo real.
- **Claude (Anthropic)**: Utilizado como asistente conversacional principal para la revisión de lógica de código, la refactorización y la depuración de la lógica de negocio (Scanner.kt, IconExtractor.kt).
- **Gemini (Google)**: Utilizado como asistente conversacional para la estructuración y redacción final de esta memoria del proyecto.

#### B.2 Ejemplos de integración y conversaciones relevantes (Prompts)

La IA se integró en todas las fases del desarrollo, actuando como un asistente de programación (*pair programmer*) y un consultor técnico. A continuación, se describen los flujos de conversación y *prompts* utilizados:

### **Fase 1: Planificación Inicial**

Tras recibir el enunciado del proyecto, se utilizó la IA para realizar un planteamiento inicial de la arquitectura.

#### **Prompt (a Claude):**

"Tengo que hacer este proyecto [Se adjunta el enunciado del Proyecto 3: Lanzador de Juegos]. Necesito un planteamiento inicial. ¿Qué clases principales de Kotlin crees que debería crear? ¿Cómo debería estructurar el escaneo de archivos en Windows y Linux?"

### **Fase 2: Asistencia a la Codificación y Revisión**

#### **Prompt (Gemini):**

"Tengo esta clase Scanner.kt [Se adjunta el código]. La lógica para filtrar 'aplicaciones basura' no funciona bien, sigue listando instaladores. Ayúdame a revisar y mejorar los filtros (keywordsIgnoradas y TAMANO\_MINIMO\_MB) para que sea más preciso."

#### **Prompt (Gemini):**

"Tengo una lista LazyColumn que muestra los iconos de los .exe con FileSystemView.getSystemIcon, pero la calidad es horrible. ¿Hay alguna forma de obtener el ícono de alta resolución de un .exe en Windows para que se vea bien en una Card?"

### **Fase 3: Depuración y Configuración del Entorno**

#### **Prompt (Gemini):**

"Estoy desesperado. Mi proyecto de Compose for Desktop no arranca y da el error: LibraryLoadException: Cannot find skiko-windows-x64.dll. He probado a limpiar la caché (.gradle/caches), mover el proyecto a mi disco local y comprobar mi JDK, pero sigue fallando. Mi JAVA\_HOME usa JDK 25. ¿Puede ser ese el problema?"

**Prompt (a Claude):**

"Mi proyecto de Compose for Desktop no compila. Me da un error de incompatibilidad con la versión de Java (JDK). Estoy usando la versión 21. ¿Qué versión de JDK es totalmente compatible con la versión de Compose que estoy usando? Ayúdame a corregir las versiones en mi archivo build.gradle.kts."

**Fase 4: Implementación de Funcionalidad Avanzada de UI (Sustituye a "Empaquetado")**

*(Esta fase describe mejor nuestro trabajo que la de "Empaquetado", ya que no llegamos a esa parte).*

**Promp (Gemini):**

"Ya tengo la lista y la barra de búsqueda. Ahora quiero añadir filtros más avanzados. ¿Cómo puedo añadir dos menús desplegables (DropdownMenu)? Uno para 'Filtrar por:' (TODAS, SISTEMA, USUARIO) y otro para 'Ordenar por:' (A-Z, Z-A), y que la lista principal se actualice automáticamente."

**Prompt (Gemini):**

"Me da un error al usar ExposedDropdownMenuItem en Compose for Desktop. He investigado y creo que ese componente es solo para Android. ¿Podemos reescribirlo usando un Box, un OutlinedButton y un DropdownMenu estándar que sí funcione en Desktop?"

**Fase 5: Generación de la Memoria****Prompt (Gemini):**

"Tengo estos 8 archivos.kt de mi proyecto [Se adjuntan los archivos] y esta plantilla de memoria [Se adjuntan los requisitos del documento]. Analiza el código y ayúdame a redactar una guía de los apartados a modo de checklist para que redacte el documento.

**Prompt (Gemini):**

"Dame un resumen de los prompts para incluir en el anexo del trabajo."