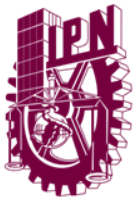


**INSTITUTO POLITÉCNICO NACIONAL**

**Escuela Superior de Cómputo**

**Ingeniería en Sistemas Computacionales**


# Desarrollo de Aplicaciones Móviles Nativas

## Práctica 3: Aplicaciones Nativas

**De la Vega Márquez Anuar**

**2022630058**

**Profesor:**

**Gabriel Hurtado Avilés**

**Fecha:**

**19/10/2025**

# Índice

# Introducción

Este informe detalla el desarrollo de una aplicación nativa para Android como parte de la Práctica 3. El objetivo principal fue implementar un Gestor de Archivos funcional (Ejercicio 1), demostrando el manejo de componentes de UI de Android, acceso al sistema de archivos, gestión de permisos, persistencia de datos y personalización de la interfaz. La aplicación permite a los usuarios explorar, visualizar y administrar archivos y carpetas en el almacenamiento de su dispositivo.

## Marco Teórico

### Aplicaciones Nativas vs Híbridas vs Web Apps

- **Aplicaciones Nativas:** Desarrolladas específicamente para una plataforma (como Android, usando Kotlin/Java y el SDK de Android). Ofrecen el mejor rendimiento, acceso completo a las características del dispositivo y la experiencia de usuario más integrada. Este proyecto es una aplicación nativa.
- **Aplicaciones Híbridas:** Desarrolladas usando tecnologías web (HTML, CSS, JavaScript) y envueltas en un contenedor nativo (ej., usando frameworks como React Native, Flutter). Pueden ejecutarse en múltiples plataformas con una sola base de código, pero pueden tener limitaciones de rendimiento y acceso a hardware.
- **Web Apps:** Básicamente, sitios web diseñados para funcionar bien en navegadores móviles. No requieren instalación, pero dependen de la conectividad y tienen acceso limitado a las funciones del dispositivo.

### Arquitectura Android y Componentes Principales

Android sigue una arquitectura basada en componentes, donde las aplicaciones se construyen a partir de bloques reutilizables con ciclos de vida definidos:

- **Activities:** Representan una única pantalla con una interfaz de usuario. La aplicación Gestor_Archivos utiliza principalmente MainActivity, TextViewerActivity y ImageViewerActivity.
- **Services:** Realizan operaciones en segundo plano sin una interfaz de usuario visible (no utilizado directamente en este ejercicio, aunque las operaciones de archivo podrían beneficiarse de ellos para tareas largas).
- **Content Providers:** Gestionan el acceso a un conjunto compartido de datos de la aplicación. FileProvider es un subtipo especializado utilizado aquí para compartir archivos de forma segura con otras apps.
- **Broadcast Receivers:** Responden a anuncios de difusión de todo el sistema (ej., batería baja, conectividad). No utilizado directamente en este ejercicio.
- **Layouts (XML):** Definen la estructura visual de la interfaz de usuario (ej., LinearLayout, CoordinatorLayout, RecyclerView, Toolbar).

- **ViewModel & LiveData (Arquitectura MVVM):** Aunque no se implementó explícitamente un ViewModel completo, la separación de la lógica de UI (MainActivity) de los datos (FileItem, AppDatabase) sigue principios similares a MVVM (Model-View-ViewModel) o MVC (Model-View-Controller).

### Patrones de Diseño Utilizados

- **Adapter Pattern:** Fundamental en Android para conectar datos a vistas complejas como RecyclerView. Se utilizaron FileAdapter, BreadcrumbAdapter y RecentFilesAdapter.
- **Singleton Pattern:** Implementado en AppDatabase.kt (companion object con getInstance) para asegurar una única instancia de la base de datos en toda la aplicación, optimizando recursos.
- **Observer Pattern (implícito):** Aunque no se usó LiveData directamente, la actualización de la UI (notifyDataSetChanged en los adapters) después de operaciones en segundo plano (coroutines con Room) sigue un patrón similar de observar cambios en los datos.
- **Repository Pattern (simplificado):** Los DAOs (FavoriteDao, RecentFileDao) actúan como una abstracción sobre la fuente de datos (la base de datos Room), similar a un repositorio.

# Ejercicio 1: Gestor de Archivos

## Descripción General y Flujo de Usuario

La aplicación Gestor_Archivos permite al usuario interactuar con el sistema de archivos de su dispositivo Android.

**Flujo Principal:**

1. **Inicio:** La aplicación solicita los permisos de almacenamiento necesarios. Una vez concedidos, muestra el contenido del directorio raíz del almacenamiento externo.
2. **Navegación:**
    - El usuario puede tocar una carpeta para entrar en ella. La lista principal se actualiza y la barra de "breadcrumbs" (ruta) en la parte superior también se actualiza.
    - El usuario puede tocar un segmento en la barra de "breadcrumbs" para saltar directamente a esa carpeta padre.
    - El botón "Atrás" del sistema permite retroceder a la carpeta anterior.
3. **Visualización:**
    - Los archivos y carpetas se muestran con iconos distintivos.
    - El usuario puede cambiar entre vista de lista y vista de cuadrícula usando un botón en la barra de herramientas.

4. **Acciones:**
    - **Pulsación Larga:** Abre un menú contextual con opciones:
        - **Añadir/Quitar favoritos:** Marca/desmarca el archivo/carpeta (muestra una estrella).
        - **Copiar/Mover:** Inicia la operación y muestra una barra inferior ("Pegar"/"Cancelar"). El usuario navega a otra carpeta y pulsa "Pegar".
        - **Renombrar:** Muestra un diálogo para cambiar el nombre.
        - **Eliminar:** Muestra un diálogo de confirmación antes de borrar.
        - **Detalles:** Muestra un diálogo con la ruta, tamaño y fecha de modificación.
    - **Pulsación Simple (Archivo):**
        - Si es texto o imagen, se abre en un visor interno (TextViewerActivity o ImageViewerActivity).
        - Si es otro tipo, se usa un Intent para abrirlo con una app externa.
    - **Botón Flotante (+):** Muestra un diálogo para crear una nueva carpeta en el directorio actual.
5. **Búsqueda:** El usuario puede tocar el icono de lupa en la barra de herramientas para filtrar la lista actual por nombre.
6. **Recientes:** Una sección colapsable en la parte superior muestra los últimos archivos abiertos.
7. **Personalización:** Desde el menú de 3 puntos, el usuario puede seleccionar el tema de color (IPN/ESCOM) y el modo de apariencia (Claro/Oscuro/Automático).

## Arquitectura Técnica

- **Estructura:** Aplicación de una sola actividad principal (MainActivity) que gestiona la mayor parte de la lógica y la UI. Se utilizan actividades secundarias (TextViewerActivity, ImageViewerActivity) para los visores internos.
- **Clases Principales:**
    - MainActivity.kt: Controla la UI principal, la navegación, las acciones, la gestión de permisos, las preferencias y la interacción con los adapters y la base de datos.
    - FileItem.kt: Modelo de datos (data class) que representa un archivo o carpeta en la lista principal.
    - FileAdapter.kt: Adapter para el RecyclerView principal, maneja la visualización de archivos/carpetas, el filtrado y la actualización de favoritos.

- BreadcrumbAdapter.kt / PathSegment.kt: Adapter y modelo de datos para la barra de ruta horizontal.
- RecentFilesAdapter.kt: Adapter para el RecyclerView de archivos recientes.
- AppDatabase.kt, Favorite.kt, FavoriteDao.kt, RecentFile.kt, RecentFileDao.kt: Componentes de la base de datos Room para persistencia de favoritos e historial.
- TextViewerActivity.kt / activity_text_viewer.xml: Actividad y layout para el visor de texto.
- ImageViewerActivity.kt / activity_image_viewer.xml: Actividad y layout para el visor de imágenes (usando la librería PhotoView).

## Acceso a Recursos Nativos

El recurso nativo principal utilizado es el Sistema de Archivos. El acceso se realiza mediante:

1. **API java.io.File:** Para listar directorios (listFiles()), obtener información (name, path, isDirectory, length, lastModified), crear carpetas (mkdir()), renombrar (renameTo()) y eliminar (delete(), deleteRecursively()).
2. **Environment.getExternalStorageDirectory():** Para obtener la ruta raíz del almacenamiento externo principal.
3. **FileProvider:** Para generar URIs seguras (content://...) que se pasan a otras aplicaciones mediante Intent.ACTION_VIEW, permitiendo abrir archivos sin exponer rutas directas del sistema de archivos, lo cual es crucial por seguridad en Android moderno.
4. **Permisos de Almacenamiento:** Gestión explícita de READ_EXTERNAL_STORAGE (lectura, Android <= 9), WRITE_EXTERNAL_STORAGE (escritura, Android <= 9) y MANAGE_EXTERNAL_STORAGE (gestión completa, Android >= 11) a través de comprobaciones en tiempo de ejecución y solicitudes al usuario (incluyendo dirigirlo a la configuración del sistema para MANAGE_EXTERNAL_STORAGE).

**Ejemplo Comentado (Gestión de Permisos en MainActivity.kt):**

**Kotlin MainActivity.kt**

```
private fun checkAndRequestPermissions() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) { // Android 11+
        if (!Environment.isExternalStorageManager()) {
            // Si no tiene el permiso MANAGE_EXTERNAL_STORAGE
            try {
                // Intenta abrir la pantalla específica de la app para este permiso
```

```
                val intent = Intent(Settings.ACTION_MANAGE_APP_ALL_FILES_ACCESS_PERMISSION)
                intent.data = Uri.fromParts("package", packageName, null)
                startActivity(intent)
            } catch (e: Exception) {
                // Fallback a la pantalla genérica si la específica falla
                val intent = Intent(Settings.ACTION_MANAGE_ALL_FILES_ACCESS_PERMISSION)
                startActivity(intent)
            }
        } else {
            // Permiso ya concedido, cargar archivos
            navigateToDirectory(currentPath)
        }
    } else { // Android 10 y anteriores
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.READ_EXTERNAL_STORAGE)
            != PackageManager.PERMISSION_GRANTED) {
            // Si no tiene READ_EXTERNAL_STORAGE, solicitarlo con el diálogo estándar
            ActivityCompat.requestPermissions(
                this,
                arrayOf(Manifest.permission.READ_EXTERNAL_STORAGE),
                REQUEST_CODE_READ_STORAGE
            )
        } else {
            // Permiso ya concedido, cargar archivos
            navigateToDirectory(currentPath)
        }
    }
}

// Maneja la respuesta del diálogo de permisos estándar (para Android <= 10)
override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<out String>, grantResults: IntArray) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults)
    if (requestCode == REQUEST_CODE_READ_STORAGE) {
        if (grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            // Permiso concedido
            navigateToDirectory(currentPath)
```

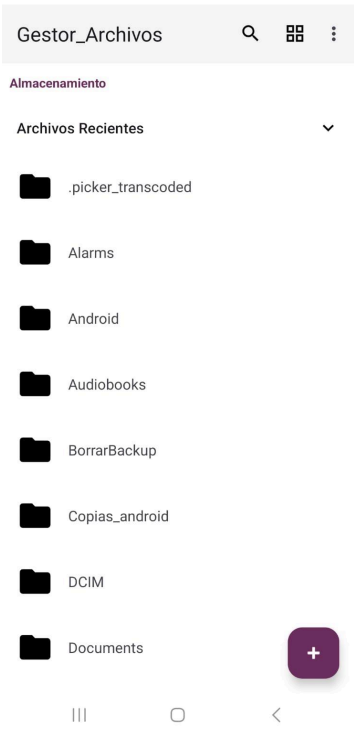
```
        } else {
            // Permiso denegado
            Toast.makeText(this, "Permiso de almacenamiento denegado.", Toast.LENGTH_LONG).show()
        }
    }
}
```

## Interfaz de Usuario

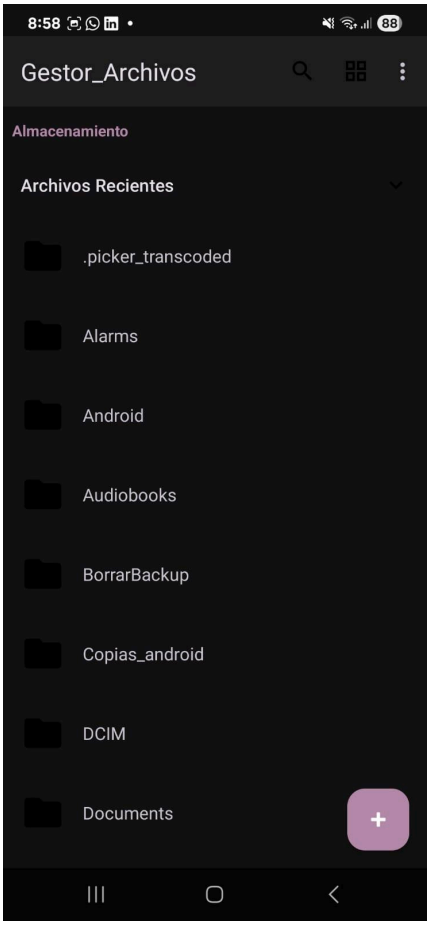
- **Componentes Principales:**
  - CoordinatorLayout como raíz para manejar interacciones entre AppBarLayout, RecyclerView y FloatingActionButton.
  - AppBarLayout con MaterialToolbar para el título y menú de opciones.
  - RecyclerView horizontal (breadcrumbRecyclerView) para la barra de ruta.
  - LinearLayout vertical para agrupar la sección de Recientes y la lista principal.
  - RecyclerView vertical (recyclerView) para mostrar la lista de archivos/carpetas (usando LinearLayoutManager o GridLayoutManager).
  - RecyclerView vertical (recentFilesRecyclerView) para la lista de recientes.
  - FloatingActionButton (fab_add_folder) para crear carpetas.
  - CardView (paste_bar) oculta que aparece para las operaciones de Copiar/Mover.
  - AlertDialog para menús contextuales, diálogos de confirmación y entrada de texto (renombrar/crear carpeta).
- **Temas y Modos:**
  - Se implementaron dos temas basados en Theme.Material3.DayNight.NoActionBar: Theme.Gestor_Archivos.IPN (Guinda) y Theme.Gestor_Archivos.ESCOM (Azul).
  - La aplicación respeta el modo claro/oscuro del sistema por defecto, pero el usuario puede forzar el modo Claro u Oscuro desde el menú.
  - Se utiliza SharedPreferences para guardar la preferencia de tema y modo.
  - Se aplica el tema/modo guardado en onCreate antes de setContentView usando setTheme() y AppCompatDelegate.setDefaultNightMode().
  - La actividad se recrea (recreate()) al cambiar de tema/modo para aplicar los cambios visuales.

**Capturas de Pantalla (4 combinaciones):**

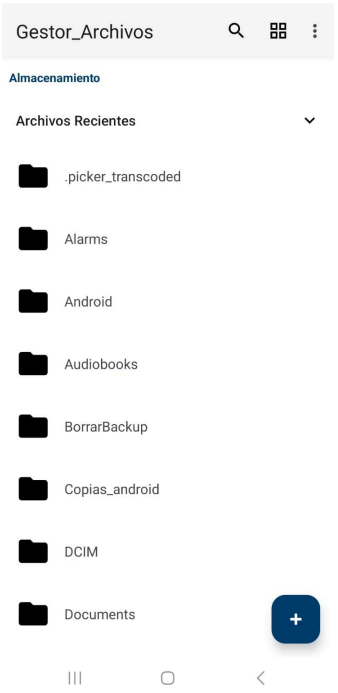
## Tema Guinda, Modo Claro:



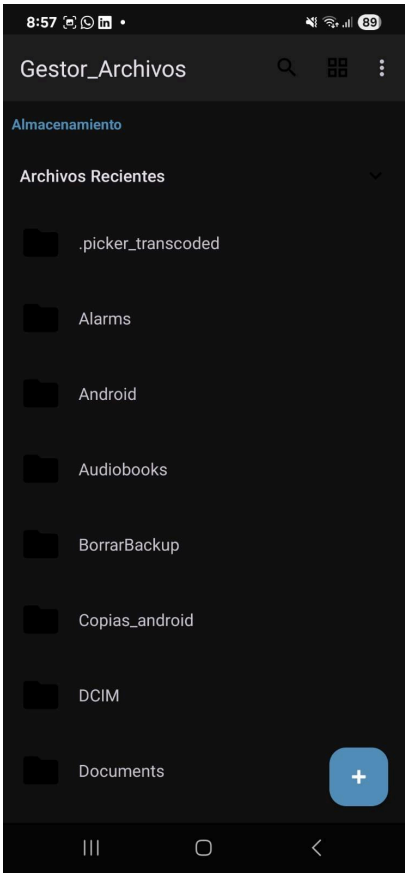

## Tema Guinda, Modo Oscuro:



## Tema Azul, Modo Claro:




## Tema Azul, Modo Oscuro:



## Persistencia de Datos

Se utiliza la biblioteca Room Persistence Library para almacenar datos localmente de forma estructurada y eficiente.

- **Tecnología Usada:** Room (abstracción sobre SQLite).
- **Esquema de BD:**
    - **Tabla favorites_table:** (Entidad: Favorite.kt)
        - path (TEXT, Primary Key): Ruta absoluta del archivo/carpeta.
        - name (TEXT): Nombre del archivo/carpeta.
        - isDirectory (INTEGER/Boolean): Indica si es una carpeta.
    - **Tabla recent_files_table:** (Entidad: RecentFile.kt)
        - path (TEXT, Primary Key): Ruta absoluta del archivo.
        - name (TEXT): Nombre del archivo.
        - lastOpenedTimestamp (INTEGER/Long): Momento (en milisegundos) en que se abrió el archivo, usado para ordenar.
- **Acceso:** Se definieron interfaces DAO (FavoriteDao.kt, RecentFileDao.kt) con anotaciones (@Dao, @Insert, @Query, @Delete) para definir las operaciones SQL.
- **Instancia:** Se usa un Singleton (AppDatabase.kt) para gestionar la instancia única de la base de datos.
- **Operaciones:** Las operaciones de lectura/escritura se realizan en segundo plano utilizando Coroutines (lifecycleScope.launch) para no bloquear el hilo principal.

## Gestión de Permisos

- **Lista:** READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE (limitado a API <= 28), MANAGE_EXTERNAL_STORAGE.
- **Justificación:** Ver sección "Acceso a Recursos Nativos". Son esenciales para leer y modificar el almacenamiento.
- **Implementación:**
    - Declarados en AndroidManifest.xml.
    - Comprobación en tiempo de ejecución en MainActivity.kt (checkAndRequestPermissions).
    - Solicitud mediante ActivityCompat.requestPermissions para READ_EXTERNAL_STORAGE (API <= 29).
    - Redirección a la configuración del sistema (Settings.ACTION_MANAGE_APP_ALL_FILES_ACCESS_PERMISSION) para MANAGE_EXTERNAL_STORAGE (API >= 30).
    - Manejo de la respuesta del usuario en onRequestPermissionsResult (API <= 29) y re-comprobación en onResume (API >= 30).

# Secciones Finales

## Pruebas Realizadas

- **Dispositivos/Emuladores Usados:**
  - Samsung SM-A346M API 35
  - Medium Phone API 36.0
  - Small Phone API 36.0
- **Casos de Prueba con Resultados:**
  - Navegación entre carpetas y uso de breadcrumbs: OK
  - Crear/Renombrar/Eliminar carpeta: OK
  - Copiar archivo entre directorios: OK
  - Mover carpeta entre directorios: OK
  - Abrir archivo de texto: OK (Visor interno)
  - Abrir imagen: OK (Visor interno con zoom)
  - Abrir PDF: OK (Abre app externa)
  - Búsqueda por nombre: OK (Filtra correctamente)
  - Añadir/Quitar Favoritos: OK (Estrella aparece/desaparece, persiste)
  - Historial de Recientes: OK (Archivos abiertos aparecen en la sección)
  - Cambio de Vista Lista/Cuadrícula: OK (Persiste)
  - Cambio de Tema (IPN/ESCOM): OK (Colores cambian, persiste)
  - Cambio de Modo (Claro/Oscuro/Auto): OK (Apariencia cambia, persiste)
  - Manejo de permisos al inicio: OK
  - Manejo de denegación de permisos: OK (Muestra Toast)
- **Pruebas de Compatibilidad, Permisos y Rendimiento:**
  - **Compatibilidad:** Se probó en [Versión Mínima] y [Versión Máxima] de Android (ver dispositivos). La UI se adapta razonablemente a diferentes tamaños (aunque no se diseñó explícitamente para tablets).
  - **Permisos:** Se verificó el flujo de solicitud y manejo de denegación en diferentes versiones de Android.
  - **Rendimiento:** La navegación es fluida. La carga de imágenes mejora notablemente con el caché de Glide. Operaciones de Copiar/Mover en archivos grandes podrían bloquear la UI brevemente (mejora potencial: usar Service/WorkManager). El filtrado de búsqueda es rápido para listas de tamaño moderado.

# Conclusiones

El desarrollo del Gestor de Archivos permitió aplicar conceptos fundamentales de Android, incluyendo el manejo del ciclo de vida de Activity, diseño de UI con RecyclerView y Material Components, acceso al sistema de archivos con las APIs

adecuadas y las consideraciones de seguridad (FileProvider, permisos), persistencia de datos con Room y Coroutines, y personalización de la interfaz mediante temas y preferencias.

**Aprendizajes:**

- La importancia de manejar los permisos de almacenamiento correctamente, especialmente con las restricciones de Scoped Storage y MANAGE_EXTERNAL_STORAGE.
- La eficiencia de RecyclerView para mostrar grandes listas de datos y la necesidad de patrones como ViewHolder y DiffUtil (aunque no se usó explícitamente) para optimizarlo.
- La utilidad de Room y Coroutines para realizar operaciones de base de datos de forma segura y sin bloquear el hilo principal.
- La flexibilidad del sistema de temas y estilos de Android para la personalización.

**Dificultades:**

- Comprender e implementar correctamente la lógica de permisos para diferentes versiones de Android.
- Manejar el estado de la UI de forma consistente (ej., la barra de pegado, el resaltado de selección).
- Asegurar que las operaciones de archivo (especialmente Copiar/Mover recursivo) no bloqueen la UI.

**Soluciones y Mejoras Futuras:**

- La gestión de permisos se resolvió siguiendo la documentación oficial y probando en diferentes APIs.
- El manejo de estado se centralizó en MainActivity con variables y funciones de actualización de UI.
- Para operaciones largas, se podrían implementar Services o WorkManager en el futuro.
- Añadir selección múltiple, búsqueda avanzada y visualizadores más completos mejoraría la funcionalidad.

# Referencias Bibliográficas


- Android Developers. (s.f.). App data and files. Recuperado el 19 de octubre de 2025, de https://developer.android.com/guide/topics/data
- Android Developers. (s.f.). Save data using Room. Recuperado el 19 de octubre de 2025, de https://developer.android.com/training/data-storage/room
- Banes, C. (s.f.). PhotoView [Repositorio de GitHub]. GitHub. Recuperado el 19 de octubre de 2025, de https://github.com/chrisbanes/PhotoView