Instituto Politécnico Nacional

Escuela Superior De Computo

Desarrollo De Aplicaciones Móviles Nativas

Práctica 3

Aplicaciones Nativas

Nombre Del Alumno:

García Quiroz Gustavo Ivan | 2022630278

Grupo: 7CV3

Nombre Del Profesor: Hurtado Avilés Gabriel

Fecha De Entrega: 05/04/2025

Índice

1 Introducción

La Práctica 3: "Aplicaciones Nativas" tiene como objetivo principal la implementación y comprensión del desarrollo de aplicaciones móviles que aprovechan las capacidades nativas de los dispositivos Android. Esta práctica busca familiarizar al estudiante con las herramientas, tecnologías y metodologías empleadas en el desarrollo de aplicaciones que interactúan directamente con el hardware y las API específicas de la plataforma, proporcionando experiencias de usuario optimizadas y de alto rendimiento.

A través de dos ejercicios prácticos enfocados en la gestión de archivos y la utilización de componentes multimedia (cámara y micrófono), se pretende explorar la implementación de funcionalidades que requieren permisos especiales y acceso a recursos del sistema. Estos ejercicios permiten comprender cómo aprovechar eficientemente las capacidades nativas de Android mientras se mantienen buenas prácticas de desarrollo y se garantiza una experiencia de usuario fluida y segura.

Para el desarrollo de las aplicaciones nativas en esta práctica se ha utilizado un entorno de desarrollo moderno y optimizado para Android:

Android Studio es el entorno de desarrollo integrado (IDE) oficial para Android, basado en IntelliJ IDEA. Para esta práctica se utilizó Android Studio en su versión más reciente, que incluye el conjunto completo de herramientas para el desarrollo, depuración y prueba de aplicaciones Android:

1. **Editor de código inteligente**: Con asistencia de código, refactorización, análisis de código y otras características avanzadas específicas para Kotlin y Android.
2. **Android Emulator**: Permite probar las aplicaciones en diferentes configuraciones de dispositivos virtuales sin necesidad de hardware físico.
3. **Sistema de construcción basado en Gradle**: Proporciona flexibilidad para configurar el proceso de construcción y gestionar dependencias. En el

segundo ejercicio se utiliza específicamente Gradle con Kotlin Script (KTS), que ofrece mejor verificación de tipos y asistencia de código.

4. **Herramientas de depuración y perfilado**: Permiten analizar el rendimiento de la aplicación, identificar cuellos de botella y optimizar el consumo de recursos.
5. **Sistemas de control de versiones**: Integración con Git para gestionar el código fuente y colaborar con otros desarrolladores.

El desarrollo de las aplicaciones se realizó utilizando:

- **Kotlin** como lenguaje de programación principal
- **SDK de Android** en sus versiones más recientes, con compatibilidad para diferentes niveles de API
- **Jetpack Libraries** para implementar arquitecturas modernas y seguir las mejores prácticas
- **Gradle** como sistema de construcción y gestión de dependencias
- **Git** para el control de versiones

Este entorno proporciona las herramientas necesarias para implementar aplicaciones nativas de alta calidad que aprovechan eficientemente las capacidades del hardware y las API del sistema Android, como se demuestra en los ejercicios desarrollados en esta práctica.

2 Desarrollo

2.1 Ejercicio 1: Gestor de Archivos para Android (Kotlin)

2.1.1 Descripción y objetivos

El Gestor de Archivos para Android es una aplicación nativa desarrollada en Kotlin que permite a los usuarios explorar, visualizar y gestionar archivos almacenados en el dispositivo. La aplicación proporciona una interfaz intuitiva para navegar por las estructuras de directorios, visualizar diversos tipos de archivos (texto, código, imágenes, etc.) y realizar operaciones básicas de gestión de archivos como copiar, mover, renombrar y eliminar.

La aplicación se diseñó siguiendo las directrices de Material Design y se implementaron temas personalizados que representan los colores institucionales del IPN (guinda) y la ESCOM (azul), con adaptación automática al modo claro y oscuro del sistema.

2.1.2 Arquitectura de la aplicación

La aplicación sigue una arquitectura orientada a componentes, donde cada parte del sistema tiene una responsabilidad bien definida:

Actividades: Gestionan la interfaz de usuario y coordinan los componentes

- MainActivity: Gestiona la navegación de directorios y listado de archivos
- FileViewerActivity: Maneja la visualización de diferentes tipos de archivo
- SearchActivity: Proporciona funcionalidad de búsqueda de archivos

Fragmentos: Implementan visualizadores especializados

- TextViewerFragment: Para archivos de texto plano
- CodeViewerFragment: Para archivos JSON, XML y código
- ImageViewerFragment: Para imágenes con zoom y rotación
- Adaptadores: Conectan los datos con las vistas
- FileAdapter: Muestra la lista de archivos y directorios

Utilidades: Encapsulan funcionalidades específicas

- FileUtils: Operaciones relacionadas con la identificación y manejo de archivos
- ErrorUtils: Gestión de errores y mensajes de usuario
- ThemeManager: Gestión de temas y apariencia
- PermissionsManager: Control de permisos de acceso al almacenamiento

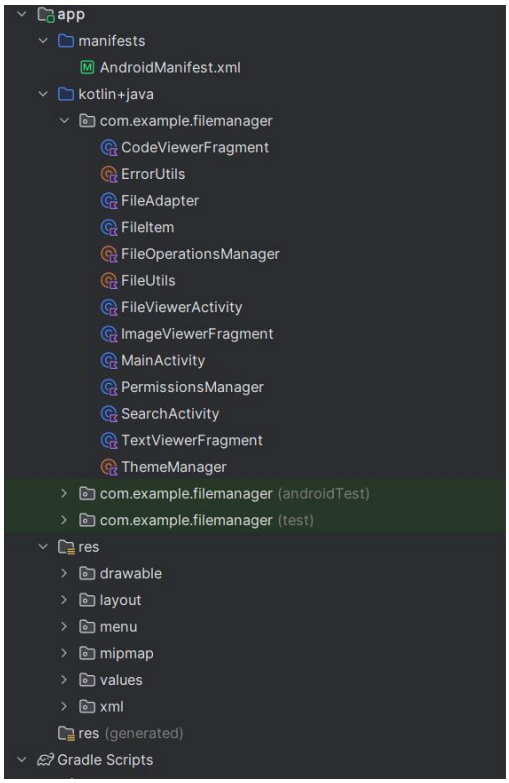


Figura 1 Estructura del proyecto Gestor de Archivos

2.1.3 Implementación

2.1.3.1 Exploración de directorios

La exploración de directorios se implementa como la funcionalidad central de la aplicación, permitiendo al usuario navegar por la estructura jerárquica de archivos y carpetas del dispositivo. Las características principales incluyen:

- Visualización de carpetas y archivos con iconos distintivos según el tipo de archivo
- Navegación intuitiva mediante una estructura de breadcrumbs que muestra la ruta actual

- Ordenación de elementos por nombre, fecha o tamaño
- Acceso a almacenamiento interno y externo (si está disponible)
- Información visual sobre el espacio disponible en cada unidad de almacenamiento
- La implementación utiliza MainActivity.kt como punto de entrada principal, donde se gestiona la navegación entre directorios y la visualización de los elementos mediante el adaptador FileAdapter.kt. La clase FileItem.kt representa cada elemento del sistema de archivos con sus metadatos relevantes.

```kotlin
// Ejemplo simplificado de cómo se cargan los archivos en un directorio
private fun loadDirectory(directory: File) {
    val files = directory.listFiles() ?: return
    val fileItems = files.map { file ->
        FileItem(
            name = file.name,
            path = file.absolutePath,
            isDirectory = file.isDirectory,
            size = if (file.isFile) file.length() else 0,
            lastModified = file.lastModified(),
            type = FileUtils.getFileType(file)
        )
    }.sortedWith(compareBy({ !it.isDirectory }, { it.name.lowercase() }))

    fileAdapter.updateFiles(fileItems)
    currentPath = directory.absolutePath
    updateBreadcrumbs()
}
```

Figura 2 Funcionalidad central de MainActivity.kt

2.1.3.2 Visualización de archivos (texto, JSON, XML, imágenes)

La aplicación implementa visualizadores especializados para diferentes tipos de archivo:

TextViewerFragment: Visualiza archivos de texto plano (.txt) con opciones de ajuste de texto y búsqueda.

CodeViewerFragment: Proporciona visualización con resaltado de sintaxis para archivos JSON, XML y otros formatos de código, mejorando la legibilidad mediante el uso de colores y sangrías adecuadas.

ImageViewerFragment: Permite visualizar imágenes con funcionalidades de zoom y rotación. Implementa gestos multitáctiles para una experiencia intuitiva.

```
fun openFile(file: File) {
    when (FileUtils.getFileType(file)) {
        FileType.TEXT, FileType.JSON, FileType.XML -> {
            val intent = Intent(this, FileViewerActivity::class.java).apply {
                putExtra(FileViewerActivity.EXTRA_FILE_PATH, file.absolutePath)
                putExtra(FileViewerActivity.EXTRA_FILE_TYPE, FileUtils.getFileType(file).name)
            }
            startActivity(intent)
        }
        FileType.IMAGE -> {
            val intent = Intent(this, FileViewerActivity::class.java).apply {
                putExtra(FileViewerActivity.EXTRA_FILE_PATH, file.absolutePath)
                putExtra(FileViewerActivity.EXTRA_FILE_TYPE, FileType.IMAGE.name)
            }
            startActivity(intent)
        }
        else -> {
            // Para archivos que la app no puede abrir directamente
            openWithExternalApp(file)
        }
    }
}
```

Figura 3 Funcionalidad central de Visualización de archivos

2.1.3.3 Sistema de favoritos y archivos recientes

La aplicación implementa un sistema de favoritos y archivos recientes que mejora la experiencia del usuario al proporcionar acceso rápido a los elementos más utilizados:

- **Favoritos**: Permite marcar/desmarcar archivos y carpetas como favoritos, que se guardan utilizando SharedPreferences para persistencia entre sesiones.
- **Archivos recientes**: Mantiene un historial de los últimos archivos accedidos por el usuario, limitado a los 20 más recientes para optimizar el rendimiento.

Ambas funcionalidades se acceden fácilmente desde el menú principal de la aplicación y utilizan el mismo adaptador de archivos con visualizaciones ligeramente personalizadas.

2.1.3.4 Operaciones con archivos (copiar, mover, renombrar, eliminar)

La clase FileOperationsManager centraliza todas las operaciones de gestión de archivos, incluyendo:

- **Copiar**: Replica archivos o directorios completos a una nueva ubicación
- **Mover**: Traslada archivos o directorios a otra ubicación
- **Renombrar**: Cambia el nombre de archivos o carpetas
- **Eliminar**: Borra archivos o directorios completos con confirmación

Todas las operaciones incluyen manejo de errores y retroalimentación adecuada al usuario mediante la clase ErrorUtils, que formatea mensajes de error comprensibles. Las operaciones más complejas se ejecutan en hilos separados para mantener la capacidad de respuesta de la interfaz de usuario.

```kotlin
// Ejemplo de implementación para renombrar un archivo
fun renameFile(context: Context, sourceFile: File, newName: String,
               onSuccess: () -> Unit, onError: (String) -> Unit) {

    if (newName.isBlank()) {
        onError("El nombre no puede estar vacío")
        return
    }

    val parentDir = sourceFile.parentFile
    if (parentDir == null) {
        onError("No se puede acceder al directorio padre")
        return
    }

    val newFile = File(parentDir, newName)
    if (newFile.exists()) {
        onError("Ya existe un archivo con ese nombre")
        return
    }

    try {
        if (sourceFile.renameTo(newFile)) {
            onSuccess()
        } else {
            onError("No se pudo renombrar el archivo")
        }
    } catch (e: Exception) {
        onError("Error: ${e.message}")
    }
}
```

Figura 4 Ejemplo de implementación para renombrar un archivo

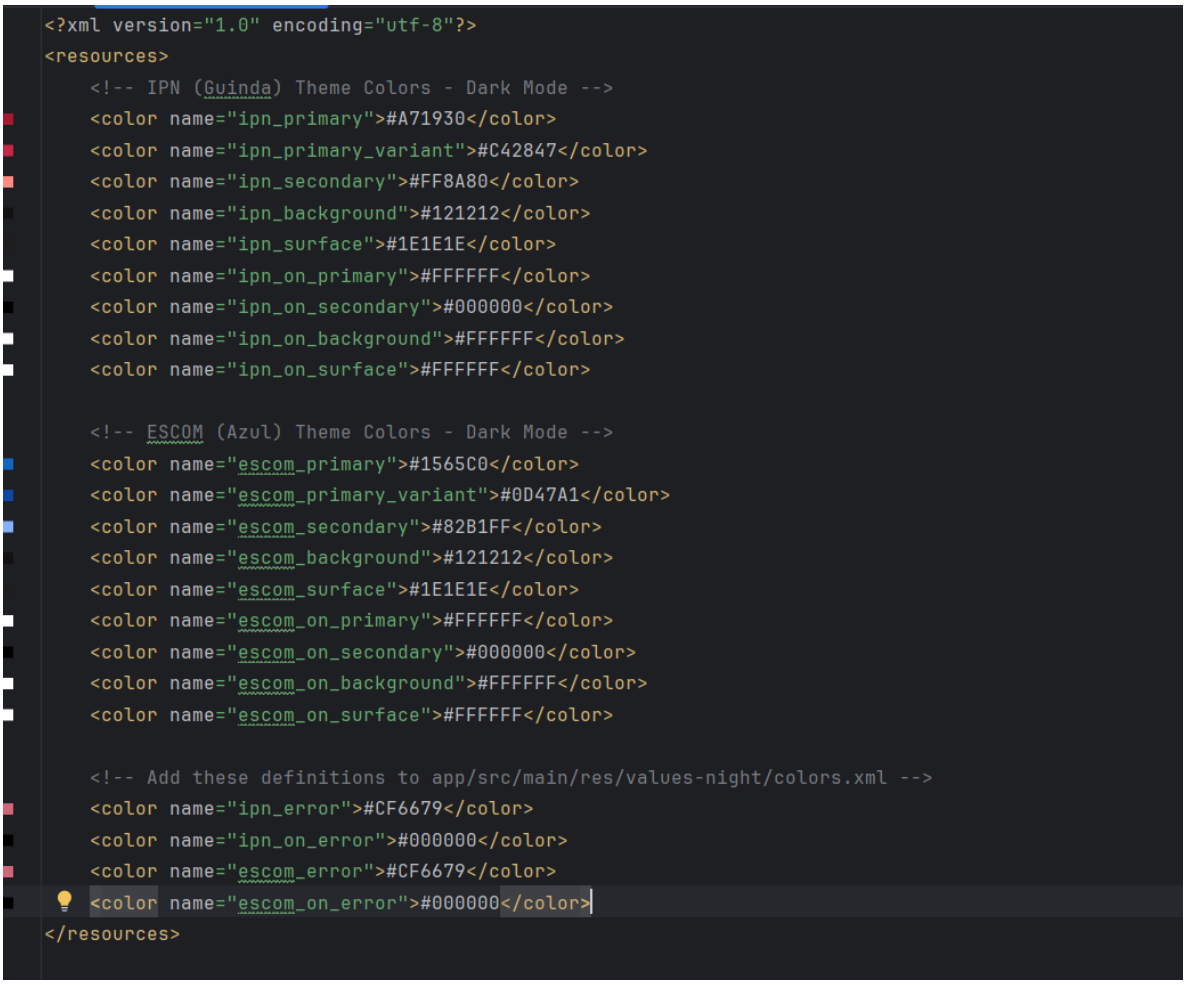
2.1.4 Interfaz de usuario

2.1.4.1 Temas personalizados (Guinda IPN, Azul ESCOM)

La aplicación implementa dos temas personalizados que representan los colores institucionales:

1. **Tema Guinda (IPN)**: Utiliza el color guinda característico del Instituto Politécnico Nacional (#8A1538) como color principal, con variaciones para elementos secundarios y de acento.
2. **Tema Azul (ESCOM)**: Emplea el color azul representativo de la Escuela Superior de Cómputo (#003B70) como color principal, con complementos apropiados.

La implementación de los temas se realiza mediante la clase ThemeManager, que gestiona la aplicación del tema seleccionado y lo guarda en las preferencias de la aplicación.



```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- IPN (Guinda) Theme Colors - Dark Mode -->
    <color name="ipn_primary">#A71930</color>
    <color name="ipn_primary_variant">#C42847</color>
    <color name="ipn_secondary">#FF8A80</color>
    <color name="ipn_background">#121212</color>
    <color name="ipn_surface">#1E1E1E</color>
    <color name="ipn_on_primary">#FFFFFF</color>
    <color name="ipn_on_secondary">#000000</color>
    <color name="ipn_on_background">#FFFFFF</color>
    <color name="ipn_on_surface">#FFFFFF</color>

    <!-- ESCOM (Azul) Theme Colors - Dark Mode -->
    <color name="escom_primary">#1565C0</color>
    <color name="escom_primary_variant">#0D47A1</color>
    <color name="escom_secondary">#82B1FF</color>
    <color name="escom_background">#121212</color>
    <color name="escom_surface">#1E1E1E</color>
    <color name="escom_on_primary">#FFFFFF</color>
    <color name="escom_on_secondary">#000000</color>
    <color name="escom_on_background">#FFFFFF</color>
    <color name="escom_on_surface">#FFFFFF</color>

    <!-- Add these definitions to app/src/main/res/values-night/colors.xml -->
    <color name="ipn_error">#CF6679</color>
    <color name="ipn_on_error">#000000</color>
    <color name="escom_error">#CF6679</color>
    <color name="escom_on_error">#000000</color>
</resources>
```

Figura 5 Ejemplo de implementación para renombrar un archivo

2.1.5 Capturas de pantalla

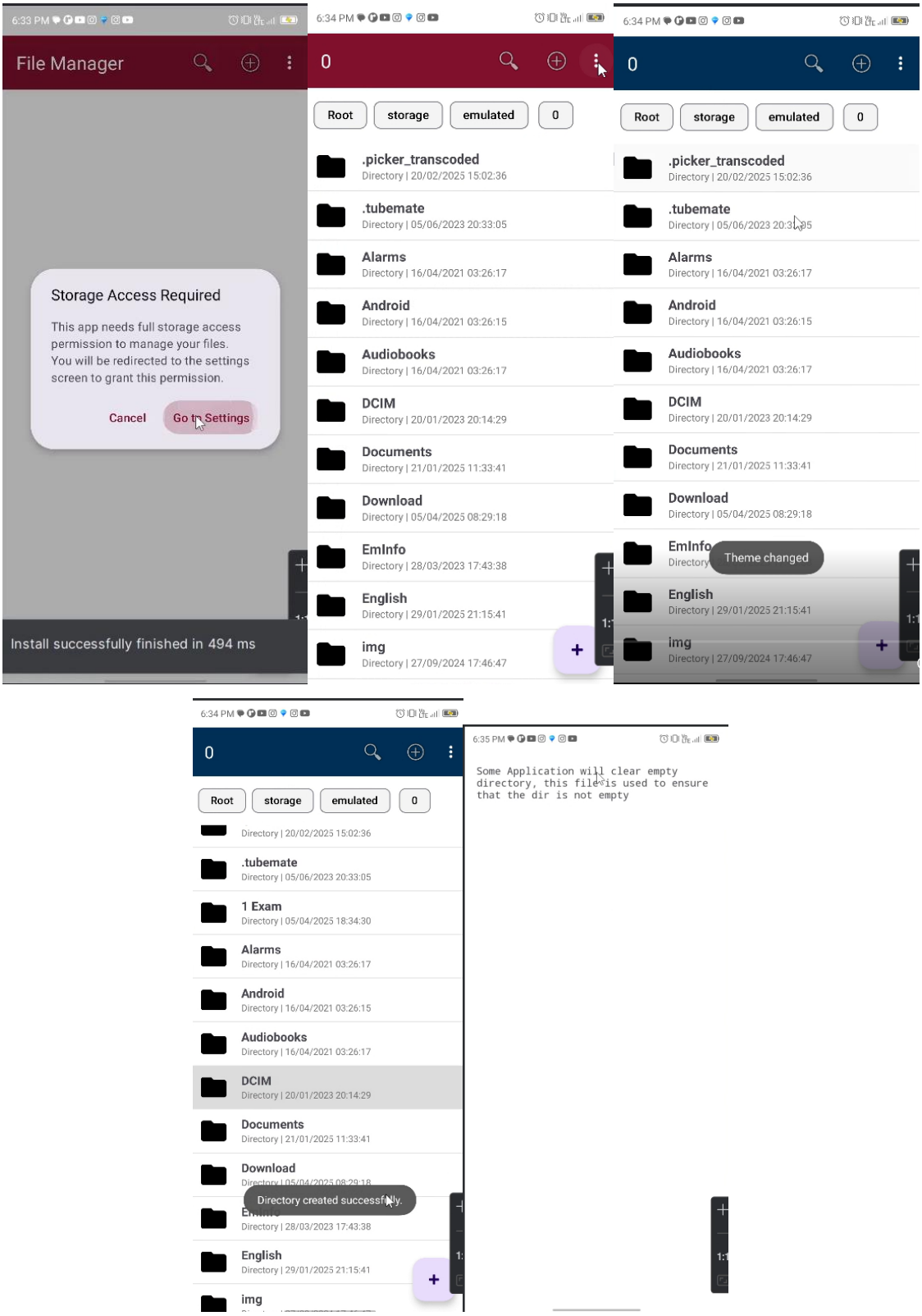


Figura 6 Capturas de pantalla

2.2 Ejercicio 2: Aplicación de Cámara y Micrófono para Android (Kotlin)

2.2.1 Descripción y objetivos

La aplicación de Cámara y Micrófono para Android es un proyecto desarrollado en Kotlin que aprovecha las capacidades nativas del dispositivo para capturar y procesar contenido multimedia. Este ejercicio tiene como objetivo principal implementar una aplicación moderna que utilice Jetpack Compose y CameraX, las API más recientes de Android para el desarrollo de interfaces de usuario y la gestión de la cámara respectivamente.

La aplicación jetpack-camera-app está diseñada para proporcionar a los usuarios una experiencia fluida y moderna al utilizar la cámara y el micrófono de sus dispositivos. Ofrece funcionalidades como la captura de imágenes, grabación de videos, grabación de audio, y configuraciones personalizables para adaptar la experiencia a las necesidades del usuario. El proyecto emplea arquitecturas modernas como MVVM (Model-View-ViewModel) y utiliza componentes de Jetpack para garantizar un código mantenible, testeable y escalable.

2.2.2 Implementación

2.2.2.1 Estructura del proyecto

El proyecto jetpack-camera-app sigue una estructura organizada basada en el sistema de construcción Gradle con KTS (Kotlin Script), lo que proporciona un entorno de desarrollo más robusto y con verificación de tipos en tiempo de compilación. La estructura principal del proyecto se divide en varias carpetas y archivos:

La carpeta raíz del proyecto contiene archivos de configuración fundamentales como:

- build.gradle.kts: Archivo principal de configuración de Gradle para todo el proyecto.
- settings.gradle.kts: Define los módulos incluidos en el proyecto.

- gradle/libs.versions.toml: Gestiona las versiones de las dependencias de manera centralizada.
- local.properties: Contiene configuraciones específicas del entorno local del desarrollador.

La carpeta principal src se divide en dos secciones principales:

- androidTest: Contiene pruebas instrumentadas que se ejecutan en dispositivos Android reales o emuladores.
- main: Incluye el código fuente principal de la aplicación, recursos y el archivo AndroidManifest.xml.

El código de la aplicación sigue una estructura jerárquica dentro de src/main/java/com/google/jetpackcamera, con paquetes organizados por funcionalidad. Uno de los paquetes principales identificados es settings, que contiene las clases relacionadas con la configuración de la aplicación, siguiendo el patrón de arquitectura MVVM:

- SettingsScreen.kt: Define la interfaz de usuario para la pantalla de configuración usando Jetpack Compose.
- SettingsUiState.kt: Modela el estado de la UI para la pantalla de configuración.
- SettingsViewModel.kt: Contiene la lógica de negocio y gestiona los datos para la pantalla de configuración.

Adicionalmente, existe un subpaquete ui que contiene componentes reutilizables de la interfaz de usuario y definiciones de temas:

- SettingsComponents.kt: Componentes Compose específicos para la pantalla de configuración.
- TestTags.kt: Constantes utilizadas para pruebas de UI.
- theme/Theme.kt: Definiciones de temas y estilos para la aplicación.

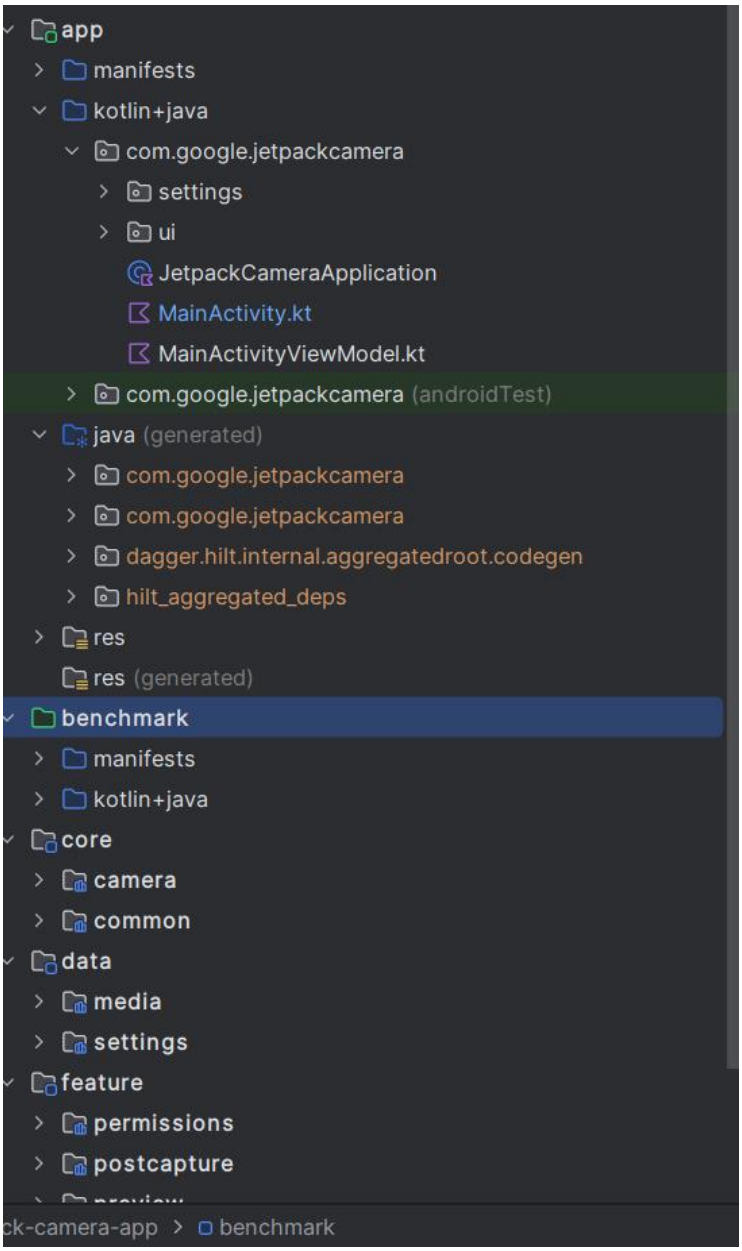


Figura 7 Estructura del proyecto jetpack-camera-app

2.2.2.2 Gestión de permisos de cámara y micrófono

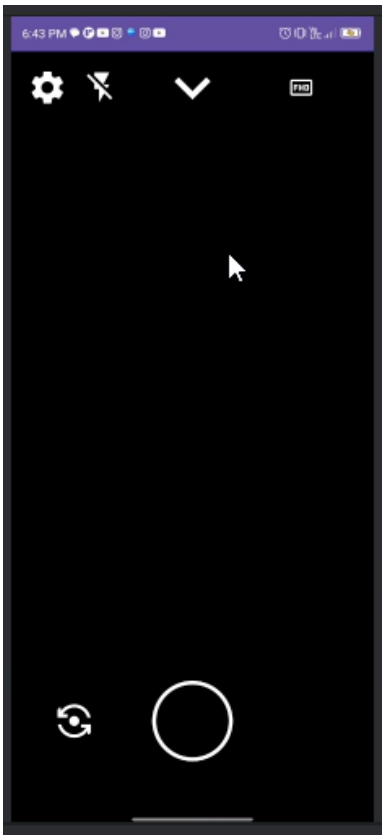
Un aspecto fundamental de la aplicación es la correcta gestión de permisos para acceder a hardware sensible como la cámara y el micrófono. Android requiere que estas aplicaciones soliciten permiso explícito al usuario, y la implementación maneja esto de forma robusta y amigable.

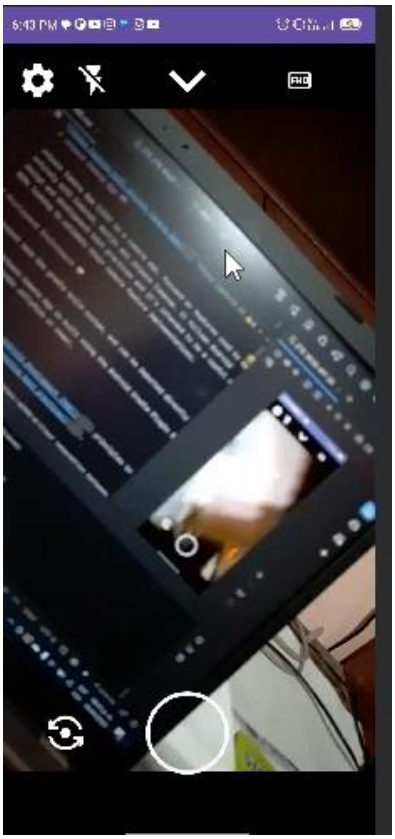
El archivo AndroidManifest.xml declara los permisos necesarios:

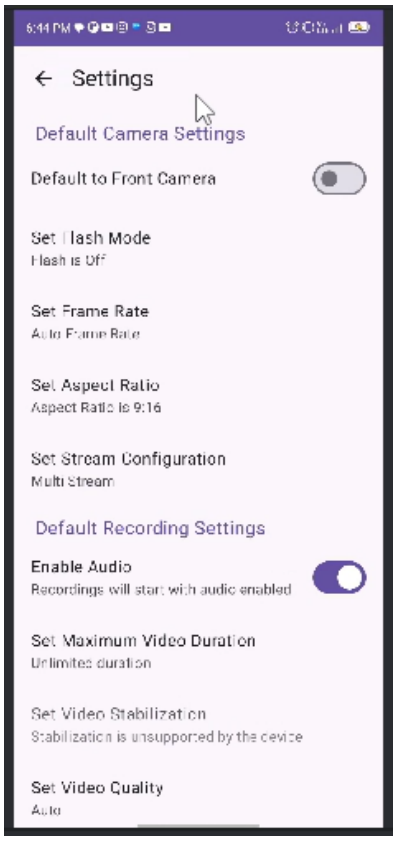
```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
                 android:maxSdkVersion="28" />
```

Figura 8 Gestión de permisos de cámara y micrófono

2.3 Capturas de pantalla






Settings
Default Camera Settings
Default to Front Camera
Set Flash Mode
Flash is Off
Set Frame Rate
Set Aspect Ratio
Aspect Ratio is 9:16
Set Stream Configuration
Multi Stream
Default Recording Settings
Enable Audio
Recordings will start with audio enabled
Set Maximum Video Duration
Unlimited duration
Set Video Stabilization
Stabilization is unsupported by the device
Set Video Quality
Auto


Settings
Default Camera Settings
Default to Front Camera
Set Flash Mode
Auto
On
Low Light Boost
Off
Close
Unlimited duration
Set Video Stabilization
Stabilization is unsupported by the device
Set Video Quality
Auto

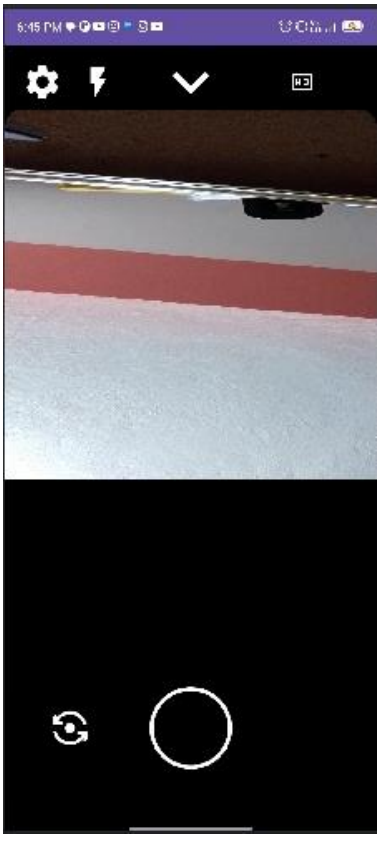


Figura 9 Capturas de pantalla y pruebas

3 Conclusiones

El desarrollo de aplicaciones nativas para Android utilizando Kotlin ha proporcionado una valiosa experiencia en la creación de soluciones móviles que aprovechan al máximo las capacidades del sistema operativo. A través de los dos ejercicios implementados el Gestor de Archivos y la Aplicación de Cámara y Micrófono se ha logrado comprender los fundamentos y las técnicas avanzadas para el desarrollo de aplicaciones Android modernas.

El trabajo con Kotlin como lenguaje principal ha demostrado ser una elección acertada, ofreciendo ventajas significativas como la concisión del código, la seguridad de tipos, y características funcionales que facilitan el desarrollo. La transición desde Java hacia Kotlin representa una evolución natural en el ecosistema de Android, proporcionando herramientas más potentes y expresivas para abordar los desafíos de programación móvil.

La comprensión de los diferentes componentes de la arquitectura Android, como Activities, Fragments, ViewModels, y la implementación de patrones de diseño como MVVM, ha sido fundamental para crear aplicaciones estructuradas, mantenibles y escalables. La práctica ha evidenciado la importancia de seguir las mejores prácticas y directrices de Google para garantizar aplicaciones que no solo funcionen correctamente, sino que también ofrezcan una experiencia de usuario óptima.

4 Bibliografía

Android Developers. (2025). *Android Developer Documentation*. Recuperado de https://developer.android.com/docs

Spring Framework. (2025). *Spring Boot Reference Documentation*. Recuperado de https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/