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 "Manejo de archivos en Android" se enfoca en el desarrollo de aplicaciones nativas que implementan técnicas avanzadas para gestionar archivos en el sistema operativo Android. En un entorno donde el almacenamiento y manipulación de datos es fundamental para casi cualquier aplicación móvil, esta práctica aborda específicamente dos implementaciones complementarias que exploran diferentes aspectos del manejo de archivos.

El primer componente es un Gestor de Archivos completo que permite a los usuarios navegar por la estructura de directorios del dispositivo, visualizar diferentes tipos de archivos (texto, JSON, XML, imágenes) y realizar operaciones básicas como copiar, mover, renombrar y eliminar archivos. Esta aplicación proporciona una interfaz intuitiva adaptada tanto para modo claro como oscuro, y utiliza temas personalizados basados en los colores institucionales del IPN y la ESCOM.

El segundo componente consiste en un juego de tipo Slice Puzzle que incorpora mecanismos de persistencia de datos mediante diferentes formatos de archivo. Esta implementación demuestra cómo una aplicación puede mantener y recuperar su estado utilizando almacenamiento de archivos en texto plano, XML y JSON, lo que permite al usuario guardar su progreso y configuración. El juego implementa dos niveles de dificultad y un sistema de puntuación, proporcionando una experiencia completa que pone en práctica los conceptos de manejo de archivos en un contexto diferente al de un gestor tradicional.

Ambas aplicaciones funcionan sin necesidad de conexión a internet, respetan las restricciones y permisos del sistema Android y están diseñadas con interfaces responsivas que se adaptan a diferentes tamaños y orientaciones de pantalla. La práctica se enmarca dentro del plan de estudios de Ingeniería en Sistemas Computacionales, específicamente en la unidad de aprendizaje "Desarrollo de aplicaciones móviles nativas", y busca fortalecer las competencias en

programación móvil, gestión de datos persistentes y diseño de interfaces de usuario.

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

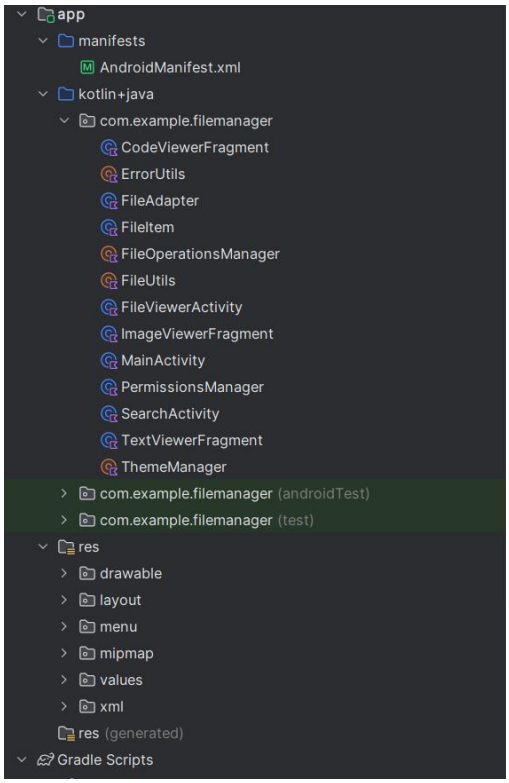


Figura 1 Estructura del proyecto Gestor de Archivos

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

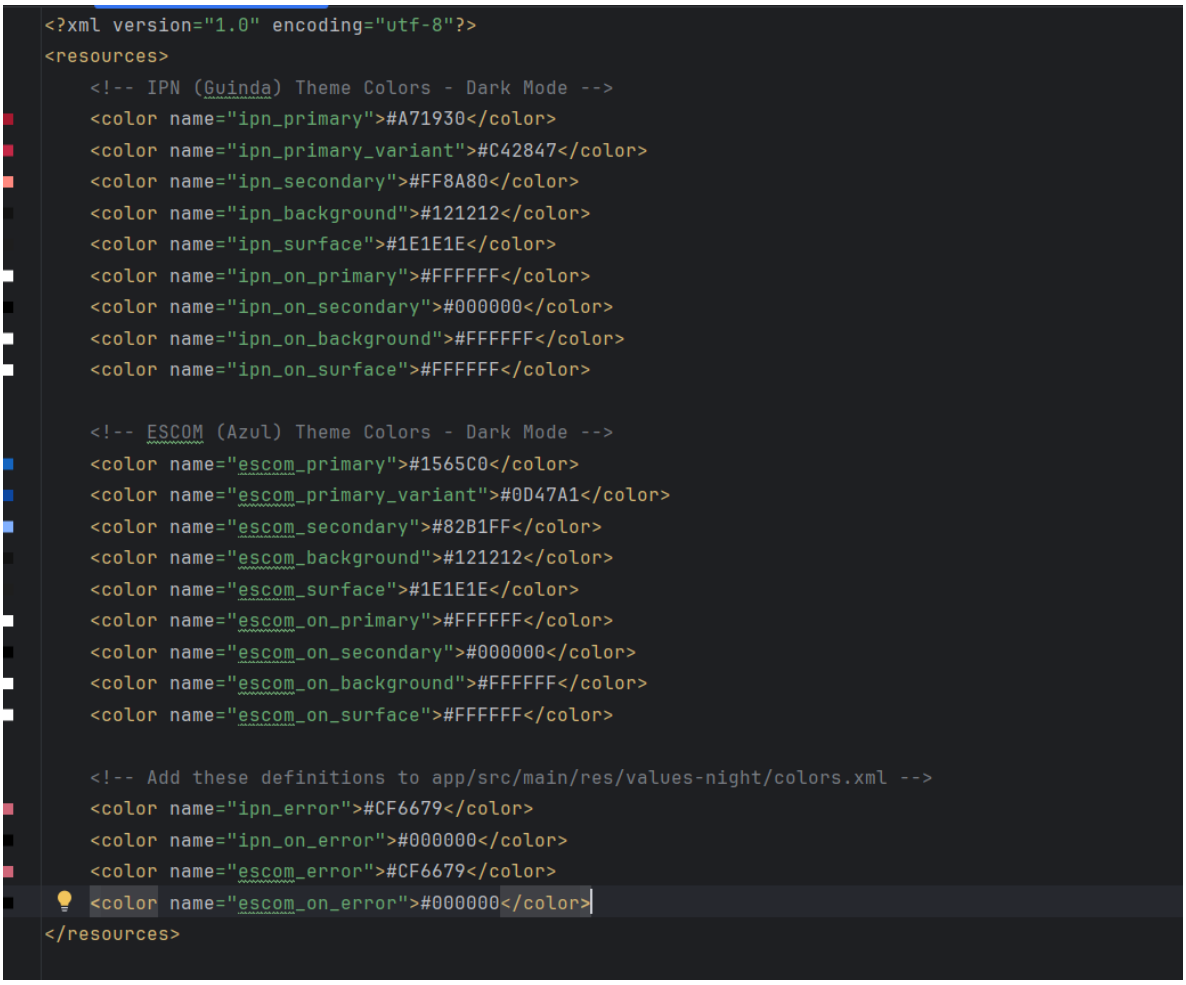
2.1.4 Interfaz de usuario

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

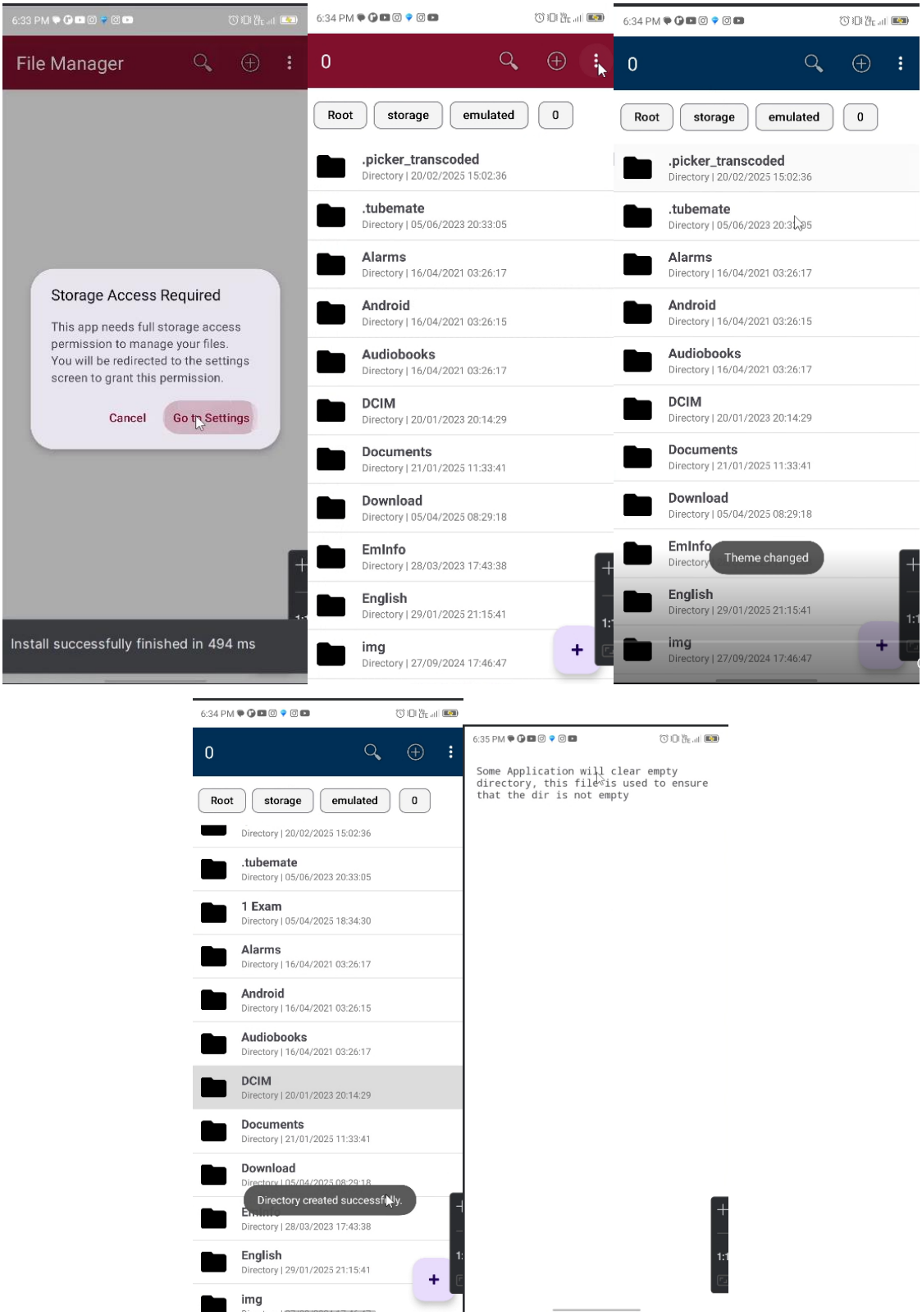


Figura 6 Capturas de pantalla

2.2 Parte 2: Juego Slice Puzzle con Gestión de Archivos

2.3 Descripción del juego

2.3.1 Mecánicas de juego

El Slice Puzzle es un juego clásico de deslizamiento de piezas donde el objetivo principal es reorganizar un conjunto de baldosas numeradas que están en desorden para formar una secuencia específica. El juego se compone de una cuadrícula (comúnmente 3x3 o 4x4) donde cada celda contiene una baldosa numerada, excepto una celda que está vacía. El jugador puede deslizar las baldosas hacia el espacio vacío, reorganizándolas progresivamente hasta alcanzar la configuración objetivo.

La implementación del juego utiliza la clase PuzzleView para renderizar la interfaz visual del tablero y la clase GameBoard para gestionar la lógica del estado del juego, incluyendo la validación de movimientos y la detección de cuando se ha completado el puzzle.

2.3.2 Niveles y modos de juego

El juego ofrece dos modos principales de dificultad:

1. **Modo Fácil**: Utiliza un tablero de 3x3 (8 piezas), ideal para principiantes o partidas rápidas.
2. **Modo Difícil**: Implementa un tablero de 4x4 (15 piezas), ofreciendo un desafío más complejo que requiere mayor planificación estratégica.

Cada modo se inicializa con una configuración aleatoria de baldosas que garantiza que el puzzle sea resoluble, verificando la paridad de la configuración inicial según las reglas matemáticas del juego.

2.3.3 Sistema de puntuación

El sistema de puntuación se basa en varios factores:

- **Movimientos realizados**: Menos movimientos resultan en una puntuación más alta.

- **Tiempo empleado**: Se registra el tiempo transcurrido desde el inicio hasta completar el puzzle.
- **Bonificaciones**: Se otorgan puntos extra por resolver el puzzle en menos movimientos que el promedio histórico del jugador.

La fórmula de puntuación se define como:

$$\boldsymbol{puntuación} = (\mathbf{1000} \times \boldsymbol{dificultad}) - (\boldsymbol{movimientos} \times \mathbf{5}) - (\boldsymbol{segundos} \times \mathbf{2})$$

Donde la dificultad es un multiplicador que varía según el modo de juego (1 para Fácil, 2 para Difícil).

2.4 Gestión de archivos en el juego

2.4.1 Implementación de guardado en formato texto plano

La clase TextGameStateStorage implementa la interfaz GameStateStorage para gestionar el guardado y carga de partidas en formato de texto plano (.txt). Los datos se guardan siguiendo una estructura específica:

```
MODE:HARD
TIME:320
MOVES:145
SCORE:560
BOARD:15,2,1,12,8,5,6,11,4,9,10,7,3,14,13,0
DATE:2025-04-02T14:30:22
```

Figura 7 formato texto plano (.txt).

Cada línea representa un atributo del estado del juego, facilitando la legibilidad por humanos. La implementación incluye métodos para serializar y deserializar el estado del juego, manejando las conversiones entre los objetos de modelo (GameState) y texto plano.

2.4.2 Implementación de guardado en formato XML

La clase XmlGameStateStorage gestiona el guardado y carga del estado del juego en formato XML. El archivo generado sigue una estructura jerárquica que representa claramente los elementos del estado del juego:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<gameState>
    <mode>HARD</mode>
    <time>320</time>
    <moves>145</moves>
    <score>560</score>
    <board>
        <tile>15</tile>
        <tile>2</tile>
        <!-- ... resto de las baldosas ... -->
        <tile>0</tile>
    </board>
    <savedDate>2025-04-02T14:30:22</savedDate>
</gameState>
```

Figura 8 formato XML

La implementación utiliza las APIs estándar de Java para XML (DOM) para crear, leer y manipular documentos XML, garantizando una representación estructurada y mantenible de los datos del juego.

2.4.3 Implementación de guardado en formato JSON

La clase JsonGameStateStorage implementa el guardado y carga del estado del juego en formato JSON. Este formato ofrece una representación más compacta y eficiente que facilita la serialización y deserialización:

```json
{
  "mode": "HARD",
  "time": 320,
  "moves": 145,
  "score": 560,
  "board": [15, 2, 1, 12, 8, 5, 6, 11, 4, 9, 10, 7, 3, 14, 13, 0],
  "savedDate": "2025-04-02T14:30:22"
}
```

Figura 9 formato JSON.

La implementación utiliza la biblioteca estándar de Android para JSON (JSONObject y JSONArray) para manipular la estructura de datos, ofreciendo un alto rendimiento en las operaciones de guardado y carga.

2.4.4 Sistema de carga de partidas

El sistema de carga de partidas está diseñado para detectar automáticamente el formato del archivo y utilizar el mecanismo de carga apropiado. La clase GameStateStorageFactory implementa un patrón factory que selecciona la implementación adecuada de GameStateStorage basándose en la extensión del archivo o analizando su contenido.

El sistema presenta una interfaz unificada al usuario mediante un diálogo que muestra las partidas guardadas con metadatos relevantes, permitiendo seleccionar cualquier archivo independientemente de su formato.

2.4.5 3.2.5 Exportación e importación de partidas

La clase GameExporter gestiona la exportación de partidas guardadas desde el almacenamiento interno de la aplicación a ubicaciones externas accesibles por el usuario o por otras aplicaciones. La implementación:

- Permite exportar partidas guardadas al almacenamiento externo en cualquiera de los tres formatos soportados.
- Genera nombres de archivo únicos basados en la fecha y hora de la exportación.
- Implementa mecanismos de seguridad para garantizar que la exportación cumpla con los permisos y restricciones de Android.
- Incluye opciones para compartir los archivos exportados a través de otras aplicaciones utilizando intents de Android.

2.4.6 Interfaz de usuario

2.4.6.1 Temas personalizados (Guinda IPN, Azul ESCOM)

La aplicación implementa dos temas personalizados acordes con las indicaciones de la práctica:

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

Estos temas se implementan utilizando la clase ThemeManager que gestiona la aplicación de los temas según la preferencia del usuario, almacenada en SharedPreferences.

2.4.7 Estructura del proyecto

El juego se implementa siguiendo un patrón de arquitectura MVC (Modelo-Vista-Controlador):

- **Modelo**: Representado por las clases en el paquete models, incluye:
    - GameState: Encapsula el estado completo del juego.
    - GameBoard: Representa el tablero de juego y su estado actual.
    - Tile: Representa una baldosa individual con su valor y posición.
- **Vista**: Implementada principalmente por:
    - PuzzleView: Vista personalizada que renderiza el tablero de juego.
    - Layouts XML: Definen la estructura de la interfaz de usuario.
- **Controlador**: Principalmente implementado en:
    - MainActivity: Coordina las interacciones entre la vista y el modelo.
    - Clases storage: Gestionan la persistencia del estado del juego.

Las clases de almacenamiento (TextGameStateStorage, XmlGameStateStorage, JsonGameStateStorage) implementan la interfaz común GameStateStorage, que define métodos para guardar y cargar el estado del juego.

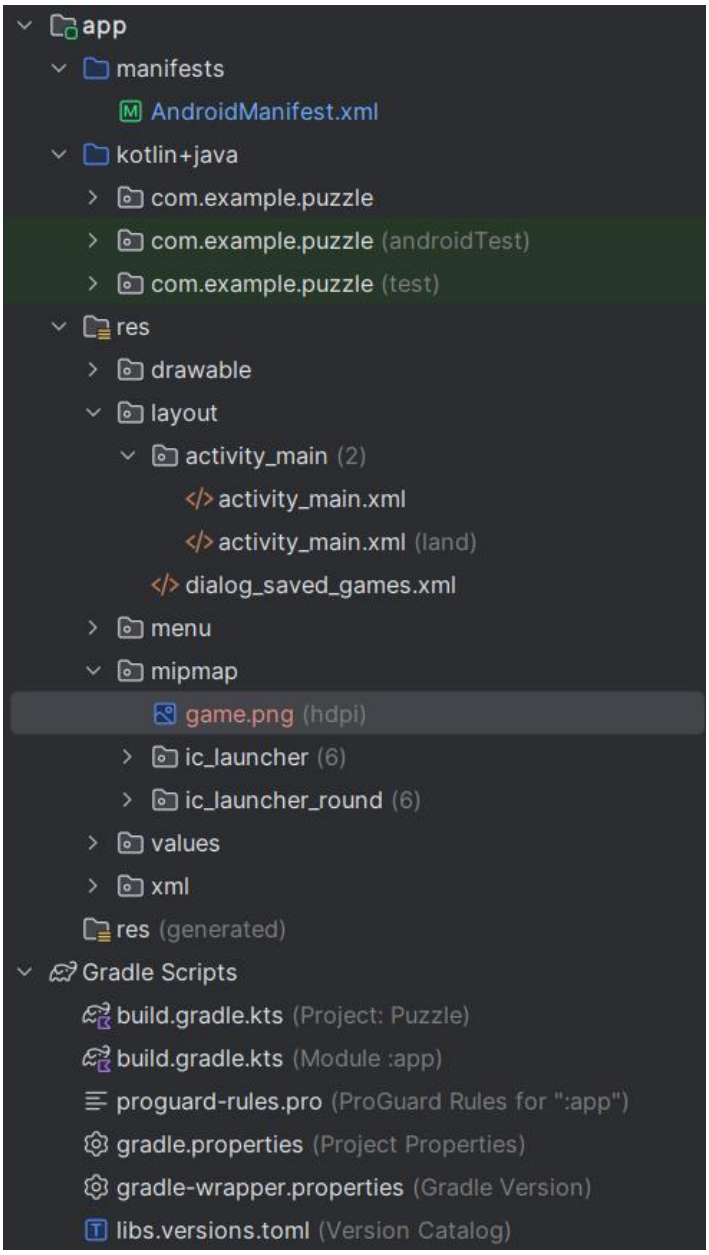


Figura 10 Estructura del proyecto

2.5 Capturas de pantalla

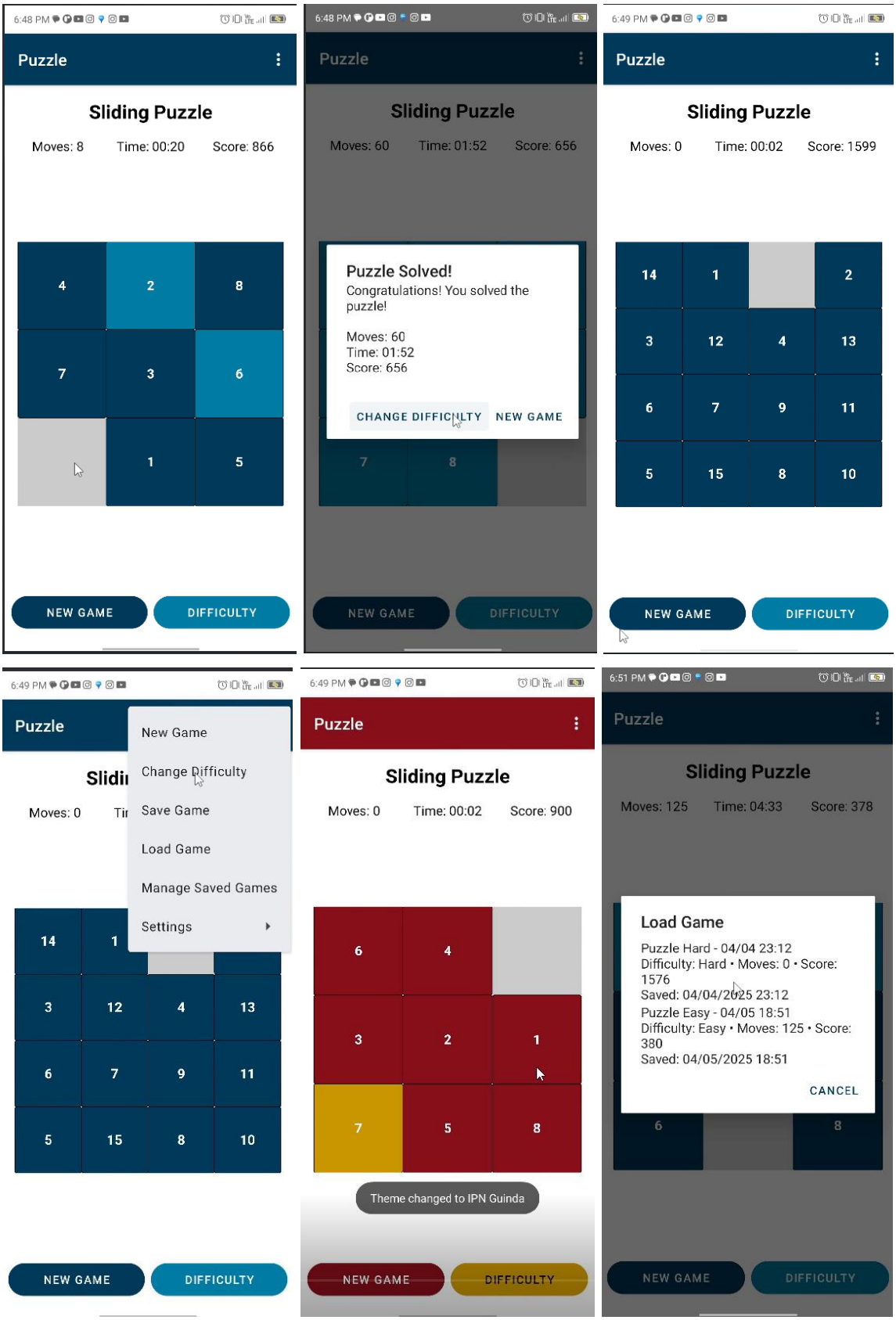

Puzzle
Sliding Puzzle
Moves: 8 Time: 00:20 Score: 866
4 2 8
7 3 6
1 5
NEW GAME
DIFFICULTY
Puzzle
Sliding Puzzle
Moves: 60 Time: 01:52 Score: 656
Puzzle Solved!
Congratulations! You solved the puzzle!
Moves: 60
Time: 01:52
Score: 656
CHANGE DIFFICULTY
NEW GAME
NEW GAME
DIFFICULTY
Puzzle
Sliding Puzzle
Moves: 0 Time: 00:02 Score: 1599
14 1 2
3 12 4 13
6 7 9 11
5 15 8 10
NEW GAME
DIFFICULTY
Puzzle
New Game
Change Difficulty
Save Game
Load Game
Manage Saved Games
Settings
Moves: 0
14 1
3 12 4 13
6 7 9 11
5 15 8 10
NEW GAME
DIFFICULTY
Puzzle
Sliding Puzzle
Moves: 0 Time: 00:02 Score: 900
6 4
3 2 1
7 5 8
Theme changed to IPN Guinda
NEW GAME
DIFFICULTY
Puzzle
Sliding Puzzle
Moves: 125 Time: 04:33 Score: 378
Load Game
Puzzle Hard - 04/04 23:12
Difficulty: Hard • Moves: 0 • Score: 1576
Saved: 04/04/2025 23:12
Puzzle Easy - 04/05 18:51
Difficulty: Easy • Moves: 125 • Score: 380
Saved: 04/05/2025 18:51
CANCEL
6 8
NEW GAME
DIFFICULTY

Figura 11Capturas de pantalla

3 Conclusiones

El desarrollo de esta práctica ha permitido profundizar en los conceptos y técnicas avanzadas de manejo de archivos en Android, demostrando la versatilidad de las soluciones de almacenamiento disponibles en el entorno móvil. A través de la implementación de un Gestor de Archivos y un juego con persistencia de datos, hemos podido aplicar conocimientos teóricos a casos prácticos con diferentes enfoques y requerimientos.

La implementación del Gestor de Archivos nos enfrentó a desafíos importantes como la gestión de permisos en versiones recientes de Android, que se han vuelto cada vez más restrictivas para proteger la privacidad del usuario. La visualización de diferentes tipos de archivo requirió entender los formatos subyacentes y crear interpretadores específicos para cada uno, lo que amplió nuestra comprensión sobre cómo se estructuran los datos en diferentes contextos. El manejo de excepciones y errores también fue crucial para garantizar una experiencia de usuario fluida incluso en situaciones inesperadas.

Por otro lado, el desarrollo del juego Slice Puzzle nos permitió explorar la serialización y deserialización de estados complejos en diferentes formatos (texto plano, XML y JSON), cada uno con sus ventajas e inconvenientes. Esta experiencia ha sido valiosa para comprender cómo seleccionar el formato más adecuado según los requisitos específicos de cada aplicación, considerando factores como la legibilidad humana, el tamaño de almacenamiento y la eficiencia en el procesamiento.

La integración de temas personalizados y la adaptación automática al modo del sistema fue un ejercicio importante en el diseño de interfaces modernas que respetan las preferencias del usuario. Este aspecto, aunque aparentemente cosmético, es fundamental para la aceptación y usabilidad de las aplicaciones en el competitivo mercado actual.

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/