Informe de la práctica Tarea UT02

Jose Luis Fuentes Parra 2 DAM

En esta actividad he aprendido a usar mucho mejor diversas funciones de Android. Durante el desarrollo me he encontrado con varios problemas y he aplicado soluciones bastante interesantes:

---

1. Estilizado Global y Herencia (BaseActivity)

Uno de los primeros problemas fue poder aplicar estilo a todas las actividades por igual desde las preferencias. Como no quería repetir el mismo código 80 veces en cada actividad, decidí crear una actividad padre denominada BaseActivity. A esta actividad se le aplican los estilos y configuraciones que comparten todas las demás. Ahora, en lugar de extender de una clase de la API de Android, todas extienden de BaseActivity y automáticamente se aplican los estilos antes de que se cree la actividad.

La idea es muy buena, pero tenía un inconveniente: en elementos que no son actividades, como las preferencias, no se aplicaba correctamente. Esto ocurría porque las preferencias cargaban demasiado rápido y no llegaban a ejecutar esos métodos. Por ello decidí que cada vez que se haga un cambio de color o tamaño se recree la actividad, pero no desde onCreate, sino desde attachBaseContext, que se ejecuta mucho antes. Así puedo aplicar los estilos a todo lo que exista en la actividad. De esta forma, los estilos quedan globalizados en una sola clase, mejorando el control y el mantenimiento del resto del proyecto.

2. Comunicación entre Actividades (ActivityResultLauncher)

Otro quebradero de cabeza fue pasar información de una pantalla a otra, por ejemplo al crear una tarea y volver a la lista principal. Antiguamente se utilizaba startActivityForResult, pero está obsoleto y resultaba poco legible.

La solución fue implementar ActivityResultLauncher. Esto me permite registrar un “contrato” al principio de la clase y olvidarme de gestionar códigos de petición poco claros. Ahora el código es más limpio y seguro, ya que cada acción (crear, editar, etc.) tiene su propio callback bien definido.

3. Guardar objetos complejos en Base de Datos (TypeConverters)

Al usar Room con SQLite, me encontré con que la base de datos solo entiende números o textos simples. Sin embargo, yo necesitaba guardar objetos como Date para las fechas y Enums para los tipos de archivos.

La solución fue crear una clase Converters. Gracias a las anotaciones @TypeConverter, Room sabe que cuando recibe una fecha debe transformarla a un número largo (Long) para guardarla, y volver a convertirla cuando la lee. Esto me evitó tener que hacer conversiones manuales en cada consulta.

4. Ordenación dinámica según el usuario (RawQuery)

El usuario puede elegir si quiere ordenar por título, fecha o progreso, y además si lo quiere de forma ascendente o descendente. Si utilizara únicamente consultas normales de Room, tendría que escribir muchas funciones distintas para cubrir todas las combinaciones posibles.

La solución fue usar SupportSQLiteQuery junto con la anotación @RawQuery. En el repositorio construyó el comando SQL dinámicamente, añadiendo el ORDER BY y el ASC o DESC según lo que el usuario haya seleccionado en sus preferencias. Así, una sola función me sirve para todo el filtrado y ordenación.

5. El caos de los hilos (Patrón Repository y Executors)

Un problema crítico en Android es que si intentas leer o escribir en la base de datos desde el hilo principal, la aplicación se cierra inesperadamente. Gestionar hilos manualmente puede ser arriesgado.

La solución fue implementar el patrón Repository. Centralicé todo el acceso a datos en TareaRepository, que utiliza un ExecutorService para ejecutar las operaciones pesadas en segundo plano sin bloquear la interfaz. Para notificar a la interfaz cuando los datos están listos, utilizo un Handler con Looper.getMainLooper(), devolviendo el resultado al hilo principal de forma segura.

6. Almacenamiento Interno vs SD (FileStorageHelper)

Gestionar los archivos adjuntos (fotos, audios, documentos) era complicado, ya que cada teléfono puede tener rutas distintas y el usuario puede elegir guardarlos en la tarjeta SD o en la memoria interna.

La solución fue crear un FileStorageHelper. Esta clase consulta al sistema cuál es el directorio seleccionado en las preferencias. Si el usuario cambia de “Memoria Interna” a “SD”, la aplicación no falla: el helper comienza a devolver la nueva ruta y gestiona automáticamente la creación de carpetas, sin que el resto del código tenga que preocuparse por dónde se almacenan realmente los archivos.

7. Formularios en varios pasos (Fragments y ViewModel)

Al crear una tarea, dividí el proceso en dos pasos usando Fragments. El problema era que al pasar del paso 1 al 2, los datos podían perderse o resultaba complicado pasarlos mediante Bundles.

La solución fue usar un FormularioViewModel compartido. Al asociar el ViewModel al ciclo de vida de la Activity principal, ambos fragmentos pueden leer y escribir en las mismas variables. Así, el usuario puede avanzar y retroceder sin perder la información introducida.

8. Selección de fechas sin errores (DatePickerDialog)

Toda aplicación de tareas necesita fechas, pero permitir que el usuario las escriba manualmente puede generar errores de formato.

La solución fue implementar un DatePickerFragment. Encapsulé el diálogo de selección de fecha de Android para que, con un solo clic, aparezca un calendario visual. Esto garantiza que la fecha siempre llegue con el formato correcto y mejora notablemente la experiencia de usuario.

9. Estadísticas en tiempo real (SQL Aggregations)

Quería mostrar un resumen de tareas completadas o el progreso medio. Hacer estos cálculos recorriendo toda la lista en Java cada vez sería poco eficiente si hay muchas tareas.

La solución fue aprovechar directamente las funciones de agregación de SQL en el DAO. Creé consultas utilizando AVG(progreso) o COUNT(*) con filtros específicos. Así, la base de datos devuelve el resultado ya calculado en muy poco tiempo, ahorrando memoria y batería.

10. Reaccionar a los cambios (Preference Listeners)

Cuando el usuario cambia una configuración, como el orden de la lista, espera que se aplique inmediatamente sin tener que salir y volver a entrar.

La solución fue implementar OnSharedPreferenceChangeListener. Configuré las actividades para que escuchen cualquier cambio en las preferencias. En cuanto el usuario modifica una opción, la actividad recibe el aviso y ejecuta automáticamente el método cargarTareas(), haciendo que la aplicación se sienta dinámica y reactiva.

11. Limpieza automática de archivos (File Cleanup)

Un error común es eliminar una tarea de la base de datos pero dejar sus archivos adjuntos ocupando espacio en el dispositivo.

La solución fue integrar una lógica de limpieza en el proceso de borrado. Cuando el repositorio elimina una tarea, no solo borra la fila en SQLite, sino que también llama al FileStorageHelper para localizar y eliminar físicamente todos los archivos asociados. Así se evita dejar archivos innecesarios en el dispositivo.

Este proyecto me ha servido para comprender que dedicar tiempo a una buena arquitectura desde el inicio ahorra mucho trabajo a largo plazo y mejora considerablemente la estabilidad y el mantenimiento de la aplicación.