Instituto Politécnico Nacional

Escuela Superior de Cómputo

Desarrollo de Aplicaciones Móviles Nativas

“Proyecto”

Alumno:

Malagón Baeza Alan Adrian

Profesor:
M. en C. José Asunción Enríquez Zárate

Grupo: 7CM1

Problemática

La problemática que abordaremos es la falta de una aplicación móvil que permita a los usuarios administrar sus tareas de manera efectiva. Muchas personas tienen dificultades para realizar un seguimiento de sus tareas diarias, lo que puede llevar a la procrastinación y la falta de productividad.

Solución

La solución propuesta es desarrollar una aplicación móvil en Kotlin que permita a los usuarios gestionar y organizar sus tareas de manera eficiente. La aplicación proporcionará funcionalidades como agregar nuevas tareas, editarlas y eliminarlas.

Objetivos

1. Desarrollar una aplicación móvil intuitiva y fácil de usar para la gestión de tareas.

2. Proporcionar una interfaz atractiva y moderna que mejore la experiencia del usuario.

3. Permitir a los usuarios agregar, editar y eliminar tareas de manera rápida y sencilla.

5. Mejorar la productividad de los usuarios al mantener un seguimiento efectivo de sus tareas diarias.

Requerimientos

1. La aplicación debe permitir a los usuarios agregar nuevas tareas proporcionando un campo de entrada de texto y un botón de agregar.

2. Debe haber una lista de tareas donde los usuarios puedan ver todas sus tareas.

5. Se deben implementar funciones de edición y eliminación de tareas para permitir a los usuarios realizar cambios según sea necesario.

6. La interfaz de usuario debe ser atractiva y seguir una paleta de colores predominante, con una experiencia de usuario fluida y receptiva.

7. La aplicación debe ser compatible con dispositivos móviles que ejecuten el sistema operativo Android.

8. Se deben utilizar tecnologías como SQLite, Room, SharedPreferences, ListView/RecyclerView para la implementación de la aplicación.

Modelo Relacional

Columna	Tipo	Clave Primaria	Autoincremental
id	Long	Sí	Sí
name	String	No	No

Diccionario de Datos

Tabla: tasks

- Columna: id
  - Tipo de dato: Long
  - Clave primaria: Sí
  - Autoincremental: Sí
  - Descripción: Identificador único de la tarea.
- Columna: name
  - Tipo de dato: String
  - Clave primaria: No
  - Autoincremental: No
  - Descripción: Nombre de la tarea.

Conceptos

SQLite

SQLite es un sistema de gestión de base de datos relacional (RDBMS) que se destaca por ser liviano, fácil de usar y de implementar. A diferencia de otros RDBMS como MySQL o PostgreSQL, SQLite no funciona como un servidor independiente, sino que se integra directamente en la aplicación que lo utiliza.

SQLite se utiliza ampliamente en aplicaciones móviles y de escritorio para almacenar datos localmente, especialmente en entornos donde se requiere una solución de base de datos ligera y autónoma. Es una opción popular para aplicaciones que necesitan un almacenamiento de datos sencillo sin tener que utilizar un servidor de base de datos completo.

Base de datos

Una base de datos SQLite es un archivo único que contiene todas las tablas, índices, vistas y otros objetos relacionados con la estructura y los datos almacenados. Este archivo se puede copiar y transferir fácilmente entre diferentes sistemas.

Tablas

Una base de datos SQLite está formada por una o más tablas, que son estructuras de datos organizadas en filas y columnas. Cada columna tiene un nombre y un tipo de datos asociados, como texto, número o fecha. Las filas representan los registros individuales de datos.

Consultas

SQLite utiliza el lenguaje SQL (Structured Query Language) para realizar consultas y manipulación de datos. Puedes utilizar sentencias SELECT para recuperar datos de la base de datos, INSERT para insertar nuevos registros, UPDATE para modificar registros existentes y DELETE para eliminar registros.

Acceso a datos

El acceso a datos con Room y SQLite se refiere a la forma en que se interactúa y se gestiona una base de datos SQLite utilizando la biblioteca Room en el desarrollo de aplicaciones de Android.

Room es una biblioteca de persistencia desarrollada por Google que proporciona una capa de abstracción sobre SQLite y simplifica el proceso de acceso a la base de

datos desde una aplicación Android. Room combina las características de SQLite, como su rendimiento y confiabilidad, con una sintaxis más sencilla y una gestión más eficiente de la base de datos.

El uso de Room simplifica la administración de la base de datos SQLite en una aplicación Android al proporcionar una capa de abstracción más alta y una sintaxis más intuitiva. Room se encarga de muchos aspectos relacionados con SQLite, como la creación y actualización de la base de datos, la generación de consultas SQL y la administración eficiente de los datos.

Entidades

Las entidades en Room son las clases que representan las tablas en la base de datos SQLite. Cada entidad se mapea a una tabla y cada instancia de la entidad se considera un registro en esa tabla. Las entidades se definen utilizando anotaciones, como `@Entity`, y cada entidad tiene una o más propiedades que corresponden a las columnas de la tabla.

DAO (Data Access Object

Los DAO en Room son interfaces o clases abstractas que definen los métodos para acceder a la base de datos. Los métodos en los DAO pueden incluir consultas SQL personalizadas o métodos predefinidos como `insert()`, `update()`, `delete()` para operaciones de CRUD (crear, leer, actualizar, eliminar) en la base de datos. Los DAO se anotan con `@Dao` en Room.

Base de datos

La clase de base de datos en Room es una clase abstracta que representa la base de datos SQLite. Esta clase se anota con `@Database` y define la lista de entidades que componen la base de datos, así como su versión. La clase de base de datos también puede proporcionar los DAO asociados y configuraciones adicionales, como migraciones de esquema.

Migraciones

Las migraciones en Room se utilizan para actualizar el esquema de la base de datos cuando se realizan cambios en las entidades o en la estructura de la base de datos. Una migración es una clase que implementa la interfaz `Migration` y define los cambios necesarios para pasar de una versión anterior a una nueva versión de la base de datos. Las migraciones permiten preservar los datos existentes durante las actualizaciones.

Consultas y relaciones

Room ofrece un conjunto de anotaciones y características adicionales para definir consultas complejas y relaciones entre entidades. Puedes utilizar anotaciones como `@Query` para ejecutar consultas personalizadas y `@Relation` para definir relaciones entre entidades y facilitar la recuperación de datos relacionados.

Shared Preferences

Shared Preferences es un mecanismo de almacenamiento ligero y simple proporcionado por Android para almacenar datos en pares clave-valor. Se utiliza principalmente para almacenar preferencias de usuario, configuraciones de la aplicación o cualquier otro dato pequeño que deba persistir en la aplicación.

Las Shared Preferences se almacenan en un archivo XML en el directorio de datos de la aplicación. Cada par clave-valor representa un dato individual que se puede acceder y modificar utilizando la clave correspondiente

Desarrollo

1. **MainActivity.kt**: Esta es la clase principal de la aplicación, que extiende la clase AppCompatActivity. En esta clase, se configura la vista principal de la aplicación y se inicializan los componentes principales, como el RecyclerView y el adaptador. También se manejan eventos como la selección de temas y la interacción con los botones de agregar y eliminar tareas.

2. **Task.kt**: Esta clase es un modelo de datos que representa una tarea. Contiene dos propiedades: el ID de la tarea y el nombre de la tarea.

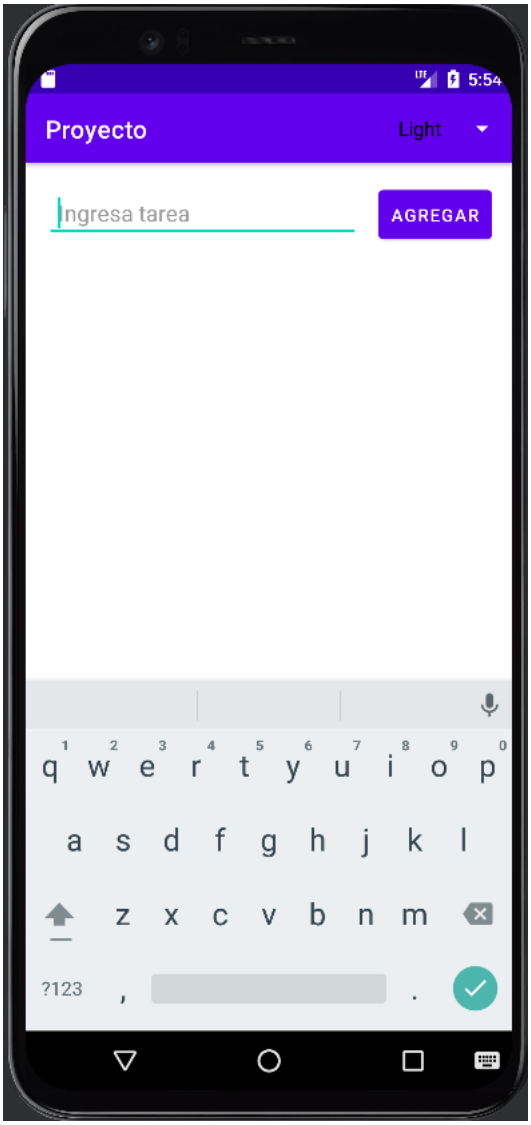
3. **TaskAdapter.kt**: Esta clase es un adaptador personalizado para el RecyclerView. Extiende la clase RecyclerView.Adapter y se encarga de inflar la vista de cada elemento de la lista y vincular los datos correspondientes. También maneja los eventos de clic en los elementos de la lista y proporciona métodos para agregar, eliminar y actualizar tareas en la lista.

4. **task_item.xml**: Este archivo XML define la apariencia de un elemento individual en el RecyclerView. Contiene una vista de texto para mostrar el nombre de la tarea y un botón para eliminar la tarea.

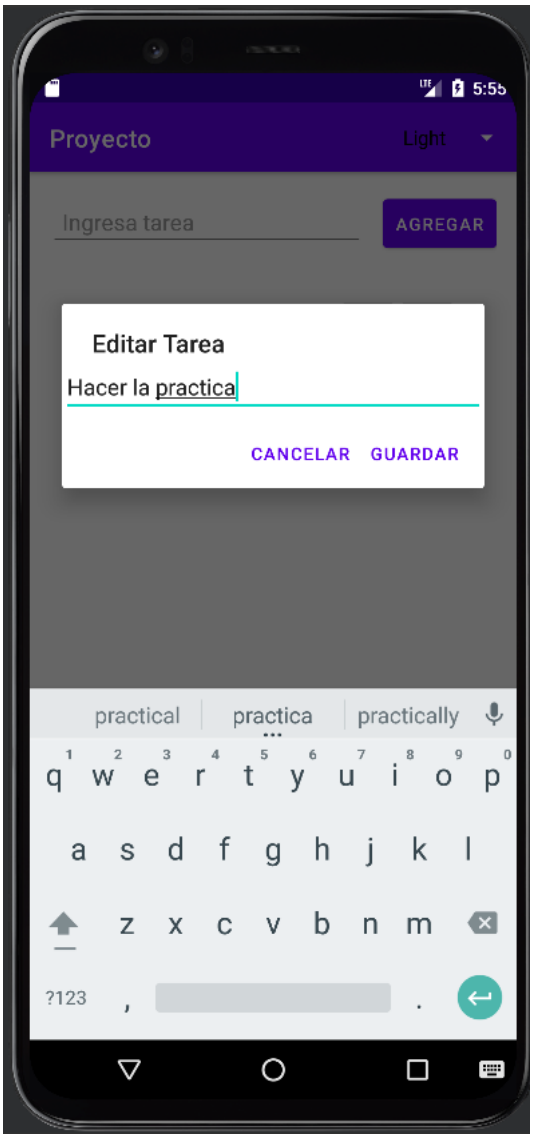
5. **activity_main.xml**: Este archivo XML define la vista principal de la actividad MainActivity. Contiene un RecyclerView para mostrar la lista de tareas y un botón flotante para agregar nuevas tareas.

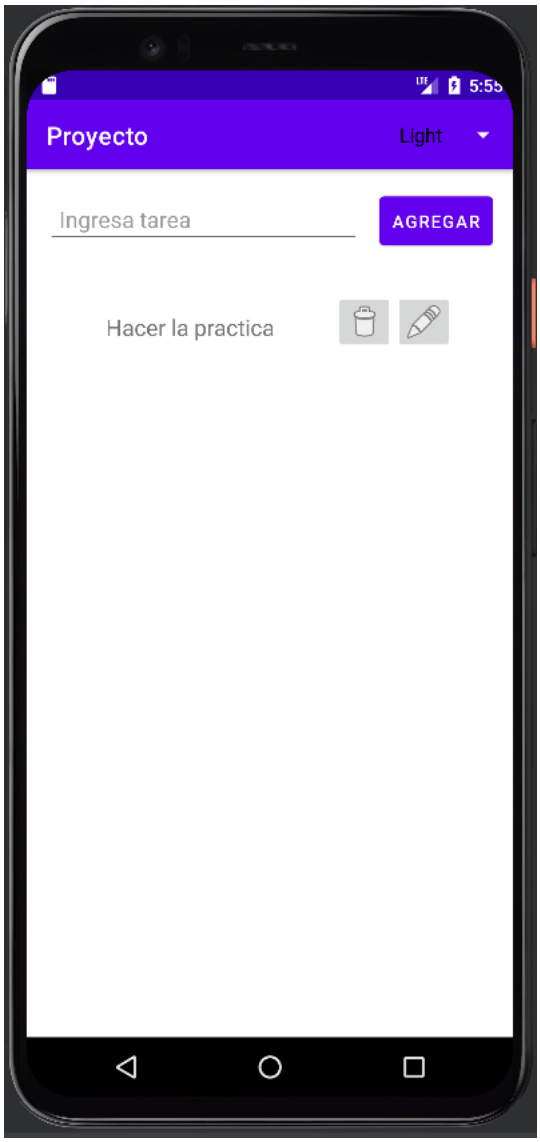
7. **menu_main.xml**: Este archivo XML define el menú de opciones en la barra de aplicaciones. Contiene elementos como la opción para cambiar el tema de la aplicación.

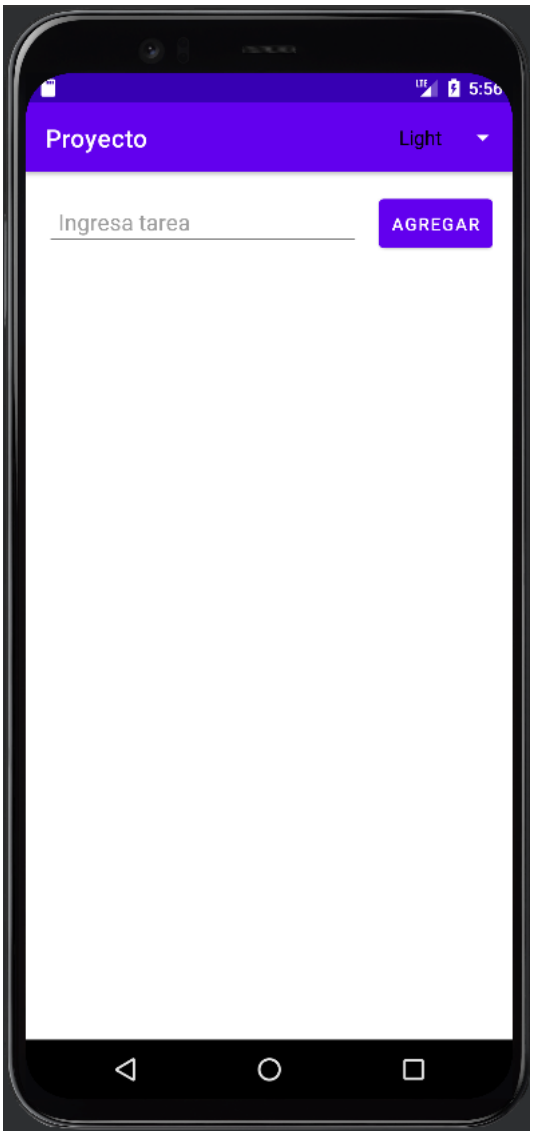
8. **TaskDAO.kt**: Esta interfaz es una parte clave de la arquitectura de Room en Android. Define los métodos de acceso a la base de datos para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la tabla de tareas.

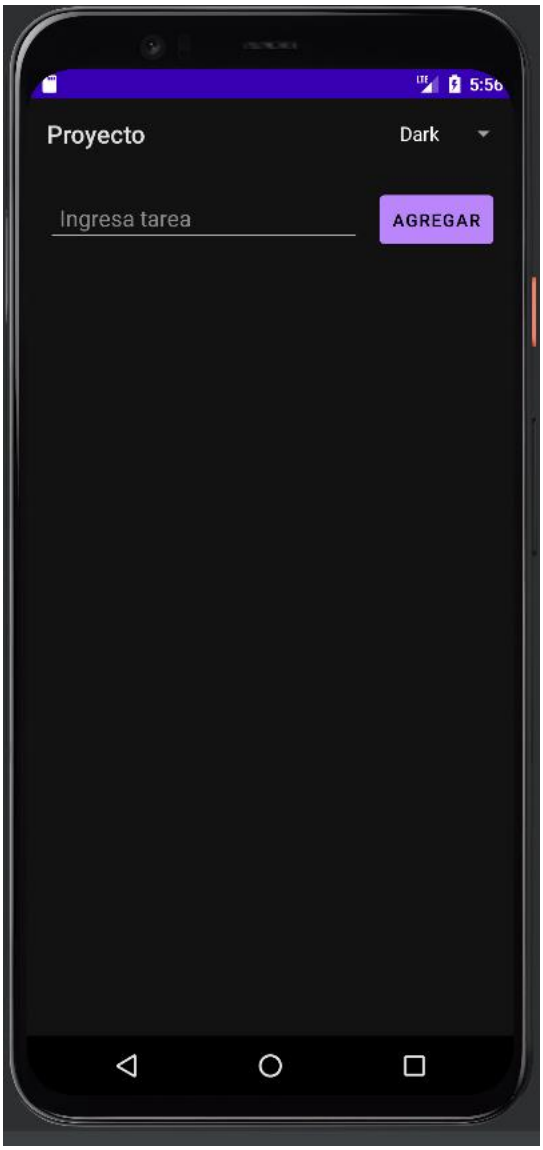

5:54
Proyecto
Light
Ingresa tarea
AGREGAR


5:55
Proyecto
Light
Ingresa tarea
AGREGAR
Hacer la practca


5:55
Proyecto
Light
Ingresa tarea
AGREGAR
Editar Tarea
Hacer la practica
CANCELAR
GUARDAR
practical
practica
practically
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
?123


5:55
Proyecto
Light
Ingresa tarea
AGREGAR
Hacer la practica


5:56
Proyecto
Light
Ingresa tarea
AGREGAR


5:56
Proyecto
Dark
Ingresa tarea
AGREGAR

Conclusiones

En conclusión, hemos desarrollado una aplicación móvil de gestión de tareas utilizando Kotlin y varios componentes de Android, como SQLite, Room, SharedPreferences y ListView/RecyclerView. A lo largo de este proyecto, hemos logrado los siguientes resultados:

1. Implementación de funcionalidades clave: Hemos creado una interfaz de usuario intuitiva que permite a los usuarios agregar, editar y eliminar tareas. Utilizamos una base de datos SQLite y la biblioteca Room para almacenar y administrar las tareas de manera eficiente. Además, utilizamos SharedPreferences para almacenar la configuración del tema de la aplicación.

2. Interfaz de usuario moderna: Mejoramos la apariencia de la aplicación utilizando colores predominantes y un diseño atractivo. Implementamos ListView y luego lo actualizamos a RecyclerView para mostrar la lista de tareas de manera más eficiente y optimizada.

3. Uso de patrones de diseño: Aplicamos el patrón de arquitectura Modelo-Vista-Controlador (MVC) para organizar y separar las responsabilidades de la aplicación. Utilizamos clases y componentes adecuados para cada parte del proyecto, lo que mejoró la estructura general y la mantenibilidad del código.

4. Persistencia de datos: Aprovechamos la funcionalidad de almacenamiento local proporcionada por SQLite y Room para persistir las tareas en la base de datos del dispositivo. Esto permite a los usuarios mantener sus tareas incluso después de cerrar la aplicación.

5. Mejora de la productividad: Nuestra aplicación proporciona una forma efectiva de administrar y organizar tareas, lo que puede ayudar a los usuarios a mejorar su productividad y cumplir con sus objetivos diarios.

En general, este proyecto nos ha brindado una base sólida para desarrollar una aplicación móvil de gestión de tareas en Kotlin. A partir de aquí, hay muchas oportunidades para expandir y agregar más funcionalidades, como notificaciones, etiquetas de categorías, sincronización en la nube, entre otros.

A través de este proyecto, hemos adquirido experiencia en el uso de varios componentes de Android, trabajando con bases de datos locales y diseñando una interfaz de usuario atractiva. Estos conocimientos pueden ser aplicados en proyectos futuros para crear aplicaciones móviles más completas y funcionales.

Referencias Bibliográficas

- Android Developers. (s.f.). Almacenamiento de datos en Android con Room. Recuperado de https://developer.android.com/training/data-storage/room?hl=es-419#groovy
- Android Developers. (s.f.). RecyclerView. Recuperado de https://developer.android.com/guide/topics/ui/layout/recyclerview?hl=es-419#:~:text=RecyclerView%20es%20el%20ViewGroup%20que,un%20objeto%20contenedor%20de%20vistas.
- Android Developers. (s.f.). Room. Recuperado de https://developer.android.com/jetpack/androidx/releases/room?hl=es-419.
- Mozilla. (s.f.). Modelo-Vista-Controlador (MVC). Recuperado de https://developer.mozilla.org/es/docs/Glossary/MVC.
- Android Developers. (s.f.). RecyclerView. Recuperado de https://developer.android.com/guide/topics/ui/layout/recyclerview?hl=es-419.
- Android Developers. (s.f.). Intents y filtros de intención. Recuperado de https://developer.android.com/guide/components/intents-filters?hl=es-419.
- Android Developers. (s.f.). Diseño de cuadrícula. Recuperado de https://developer.android.com/guide/topics/ui/layout/grid?hl=es-419.
- Android Developers. (s.f.). Diseño relativo. Recuperado de https://developer.android.com/guide/topics/ui/layout/relative?hl=es-419.