Ingeniería económica.

Nombre: Luis Ismael López Urbina

Docente: Luis Guido.

Fecha: 06 de Julio de 2024.

Contenido

1. Introdcción 3

2. ¿Qué es Room? 4

2.1 Ventajas de Room 4

3. Componentes Principales de Room 5

3.1 Base de datos: 5

3.2 Entidad: 5

3.3 DAO (Data Access Object): 5

4. Funcionalidades Clave de Room 5

4.1 Relaciones entre Entidades: 5

4.2 Consultas Personalizadas: 6

4.3 Migraciones de Base de Datos: 6

5. Integración con Otros Componentes 6

5.1 ViewModel y LiveData: 6

5.2 RecyclerView: 6

5.3 Kotlin Coroutines: 7

6. Buenas Prácticas y Consideraciones 7

6.1 Gestión de Hilos y Operaciones Asíncronas 7

7. Ejemplo Práctico 8

7.1 Creación de una Base de Datos con Room 8

1. Introducción

Las aplicaciones Android manejan una gran cantidad de datos, desde configuraciones del usuario hasta información compleja de la aplicación. La persistencia de datos se refiere a la capacidad de almacenar estos datos de forma segura y duradera, incluso después de que la aplicación se cierre o el dispositivo se reinicie.

SQLite es la base de datos relacional más comúnmente utilizada en Android. Sin embargo, administrar SQLite directamente puede ser complejo y propenso a errores. Room surge como una biblioteca que simplifica la interacción con SQLite, proporcionando una capa de abstracción y automatizando tareas repetitivas.

Similar a los ORM de las aplicaciones web Room facilita operaciones desde el código fuente a SQLlite a treves de predicados y palabras reservadas se pueden crear, modificar, consultar y eliminar tablas y registros de la base de datos si recurrir a Querys nativas de SqlLite

Este informe tiene como objetivo proporcionar una comprensión profunda de la biblioteca Room para la persistencia de datos en Android. Se cubrirán los siguientes aspectos:

-Definición y componentes principales de Room

-Ventajas y funcionalidades clave

-Integración con otros componentes de Android

-Buenas prácticas y consideraciones

-Ejemplo práctico de uso

-Casos de uso y aplicaciones del mundo real

2. ¿Qué es Room?

Room es una biblioteca de persistencia de datos SQLite para Android, que forma parte del conjunto de herramientas Jetpack de Google. Simplifica el acceso a la base de datos, reduce el código repetitivo y minimiza los errores.

Room fue lanzado por primera vez en 2016 como parte de Android Architecture Components. Ha evolucionado con el tiempo, incorporando nuevas funciones y mejoras en el rendimiento.

2.1 Ventajas de Room

-Simplicidad: Abstrae la complejidad de SQLite, haciendo que la persistencia de datos sea más fácil y accesible.

-Seguridad: Protege contra errores comunes de SQLite y ayuda a prevenir inyecciones SQL.

-Productividad: Reduce el código repetitivo y agiliza el desarrollo.

-Confiabilidad: Garantiza la integridad y consistencia de los datos.

-Verificación en tiempo de compilación: Detecta errores en las consultas SQL durante la compilación, evitando problemas en tiempo de ejecución.

3. Componentes Principales de Room

3.1 Base de datos:

-Define la configuración de la base de datos, incluyendo el nombre y las entidades que contiene.

-Se denota con @Database y proporciona métodos para acceder a los Data Access Objects (DAO).

3.2 Entidad:

La entidad representa una tabla de la base de datos y se denota con @Entity, los campos que perteneces a la tabla deben ser mapeados como propiedades de la clase que contiene la definición de la entidad

3.3 DAO (Data Access Object):

El DAO es una interfaz que encapsula las operaciones CRUD aplicables a una entidad es similar a los Repositorios del ORM JPA de SpringBoot. Estos se denotan con @Dao.

4. Funcionalidades Clave de Room

4.1 Relaciones entre Entidades:

Es posible definir relaciones entre entidades en cualquiera de las cardinalidades existente, uno a uno, uno a varios y varios a varios. Estas relaciones se definen como un decorador de la propiedad que corresponde a la columna que contiene la clave foránea en la tabla, dicho decordor se denota @Relation

4.2 Consultas Personalizadas:

Los Dao además de facilitar la interacción crud con la entidad, también permiten la integración de queries nativas a las funciones de la interfaz, estas queries nativas se colocar dentro de @Query y pueden lo complejamente necesarias.

4.3 Migraciones de Base de Datos:

Las migraciones a la base de datos permiten reflejar los cambios de una entidad en la tabla que corresponde a esta, así como crear tablas desde 0 a partir de la definición de una entidad.

Las migraciones se definen con @Migration y por lo general se ejecutan una sola vez, en caso de realizar múltiples cambios a una entidad se pueden crear las migraciones que sean necesarias.

5. Integración con Otros Componentes

5.1 ViewModel y LiveData:

-Se integra perfectamente con ViewModel y LiveData para implementar el patrón MVVM.

-Permite observar los cambios en los datos de la base de datos y actualizar automáticamente la interfaz de usuario.

5.2 RecyclerView:

-Simplifica la visualización de datos persistidos en listas o cuadrículas utilizando RecyclerView.

-Permite adaptar los datos de la base de datos a elementos de la interfaz de usuario.

5.3 Kotlin Coroutines:

-Facilita la ejecución de operaciones de base de datos de forma asíncrona utilizando Kotlin Coroutines.

-Permite manejar la concurrencia y evitar bloqueos en el hilo principal.

6. Buenas Prácticas y Consideraciones

Las buenas practicas en cualquiera de los contextos del desarrollo de software siempre ayudan a facilitar la lectura y la optimización de escritura de código además de ayudarnos a ser mejores desarrolladores, algunas de las buenas practicas aplicables en Room son:

-Utilizar transacciones para garantizar la atomicidad e integridad de las operaciones de base de datos.

-Optimizar consultas para mejorar el rendimiento y evitar cuellos de botella, habrá casos en que utilizaremos queries nativas para interactuar con la base de datos y es menester que estas sean lo complejamente necesarias para evitar retrasos en las consultas.

6.1 Gestión de Hilos y Operaciones Asíncronas:

La gestión de hilos es importante para la ejecución de consultas en segundo plano sin obstruir al hilo principal de la aplicación, imaginemos que en nuestra aplicación debamos cargar una lista de países para agregar un contacto, pues bien esta consulta al abrir nuestra aplicación podemos gestionarla en segundo plano de modo que no retrase la carga de la página principal.

7. Ejemplo Práctico

7.1 Creación de una Base de Datos con Room

Este ejemplo demuestra la creación de una base de datos simple para almacenar libros.

```
abstract class AppDatabase : RoomDatabase() {
    abstract fun bookDao(): BookDao
}

@Entity
data class Book(
    @PrimaryKey(autoGenerate = true) val id: Int,
    val title: String,
    val author: String,
    val pages: Int
)

@Dao
interface BookDao {
    @Insert
    suspend fun insert(book: Book)

    @Query("SELECT * FROM Book")
    suspend fun getAllBooks(): List<Book>

    @Query("SELECT * FROM Book WHERE id = :id")
    suspend fun getBookById(id: Int): Book

    @Update
    suspend fun updateBook(book: Book)

    @Delete
    suspend fun deleteBook(book: Book)
}
```