Integrantes: **Alejandra Chávez**

Grupo: **IC-VA-MAT-LabA**

Carrera: **Ingeniería en Computación**

Asignatura: **Programacion Movil**

Prof.: Ing. Luis Guido

Informe de Investigación: Room en Android

1. Introducción

Este informe tiene como objetivo proporcionar una visión detallada sobre Room en el desarrollo de aplicaciones Android. Room es una biblioteca de persistencia que proporciona una capa de abstracción sobre SQLite, permitiendo un acceso más fluido y robusto a la base de datos.

2. What is Room?

Room es una biblioteca de persistencia para Android que forma parte del Android Jetpack. Room facilita el trabajo con bases de datos SQLite mediante la creación de una capa de abstracción que maneja las operaciones de la base de datos de forma más segura y eficiente. Las ventajas de utilizar Room incluyen la verificación en tiempo de compilación de las consultas SQL, la integración con LiveData y la reducción de código boilerplate.

3. Components of Room

Room consta de tres componentes principales: Entity, DAO (Data Access Object) y Database. Las Entity representan las tablas en la base de datos, los DAO son las interfaces que contienen los métodos para acceder a los datos, y la Database es la clase que extiende RoomDatabase y actúa como un punto de acceso a la base de datos.

4. Setting up Room

Para configurar Room en un proyecto Android, es necesario agregar las dependencias de Room en el archivo build.gradle. A continuación se muestra un ejemplo de cómo configurar Room:

groovy

```
dependencies {
    implementation 'androidx.room:room-runtime:2.3.0'
    annotationProcessor 'androidx.room:room-compiler:2.3.0'
}
```

5. Defining an Entity

Una Entity en Room representa una tabla en la base de datos. Se define utilizando la anotación @Entity y debe incluir al menos una clave primaria. A continuación se muestra un ejemplo de cómo definir una Entity:

java

```
@Entity
public class User {
    @PrimaryKey
    public int uid;
    public String firstName;
    public String lastName;
}
```

6. Data Access Object (DAO)

El DAO (Data Access Object) es una interfaz que define los métodos para acceder a la base de datos. Los métodos en un DAO pueden incluir consultas SQL, inserciones, actualizaciones y eliminaciones. A continuación se muestra un ejemplo de cómo definir un DAO:

java

```
@Dao
public interface UserDao {
    @Query("SELECT * FROM user")
    List<User> getAll();

    @Insert
    void insertAll(User... users);
```

```
    @Delete
    void delete(User user);
}
```

7. Room Database

La clase RoomDatabase es una clase abstracta que extiende RoomDatabase y actúa como el punto de acceso a la base de datos. Debe incluir referencias a todos los DAO que se utilizarán. A continuación se muestra un ejemplo de cómo definir una RoomDatabase:

```
java
@Database(entities = {User.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract UserDao userDao();
}
```

8. Performing CRUD Operations

Las operaciones CRUD (Create, Read, Update, Delete) son fundamentales para cualquier aplicación que utilice una base de datos. Room facilita la realización de estas operaciones mediante el uso de DAO. A continuación se muestra un ejemplo de cómo realizar operaciones CRUD con Room:

```
java
// Insertar un usuario
User user = new User();
user.uid = 1;
user.firstName = "John";
user.lastName = "Doe";
db.userDao().insertAll(user);
```

```
// Leer todos los usuarios
List<User> users = db.userDao().getAll();

// Actualizar un usuario
user.firstName = "Jane";
db.userDao().update(user);

// Eliminar un usuario
db.userDao().delete(user);
```

9. LiveData and Room

LiveData es una clase de datos observable que se integra perfectamente con Room para proporcionar datos reactivos. Al usar LiveData con Room, los cambios en la base de datos se reflejan automáticamente en la interfaz de usuario. A continuación se muestra un ejemplo de cómo utilizar LiveData con Room:

```
java
@Dao
public interface UserDao {
    @Query("SELECT * FROM user")
    LiveData<List<User>> getAllUsers();
}
```

10. Migration and Versioning

La migración y el versionado de la base de datos son importantes para manejar cambios en la estructura de la base de datos sin perder datos existentes. Room proporciona una forma sencilla de definir migraciones utilizando la clase RoomDatabase.Builder. A continuación se muestra un ejemplo de cómo manejar migraciones en Room:

```
java
Room.databaseBuilder(getApplicationContext(), AppDatabase.class, "database-name")
    .addMigrations(MIGRATION_1_2)
    .build();


static final Migration MIGRATION_1_2 = new Migration(1, 2) {
    @Override
    public void migrate(@NonNull SupportSQLiteDatabase database) {
        database.execSQL("ALTER TABLE user ADD COLUMN birthday TEXT");
    }
};
```

11. Conclusión

En conclusión, Room es una herramienta poderosa en el desarrollo de aplicaciones Android, ofreciendo una forma eficiente y segura de gestionar la persistencia de datos. Comprender sus componentes y cómo utilizarlos adecuadamente es crucial para crear aplicaciones robustas y mantenibles.