Data and File Storage

Android uses a file system that's similar to disk based file systems on other platforms. The system provides several options for you to save your app data:

- **Appspecific storage: Store files that are meant for your app's use only, either in dedicated directories within an internal storage volume or different dedicated directories within external storage. Use the directories within internal storage to save sensitive information that other apps shouldn't access.**
- **Shared storage: Store files that your app intends to share with other apps, including media, documents, and other files.**
- **Preferences: Store private, primitive data in keyvalue pairs.**
- **Databases: Store structured data in a private database using the Room persistence library.**

1. Appspecific files:

- **Type of content: Files meant for your app's use only**
- **Access method:**
  From internal storage: `getFilesDir()` or `getCacheDir()`
  From external storage: `getExternalFilesDir()` or `getExternalCacheDir()`
- **Permissions needed:**
  Never needed for internal storage
  Not needed for external storage when your app is used on devices that run Android 4.4 (API level 19) or higher
- **Can other apps access?: No**
- **Files removed on app uninstall?: Yes**

2. Media:

- **Type of content: Shareable media files (images, audio files, videos)**
- **Access method: MediaStore API**
- **Permissions needed:**
  `READ_EXTERNAL_STORAGE` when accessing other apps' files on Android 11 (API level 30) or higher

`READ_EXTERNAL_STORAGE` or `WRITE_EXTERNAL_STORAGE` when accessing other apps' files on Android 10 (API level 29)
Permissions are required for all files on Android 9 (API level 28) or lower

- **Can other apps access?:** Yes, though the other app needs the `READ_EXTERNAL_STORAGE` permission
- **Files removed on app uninstall?:** No

3. Documents and other files:

- **Type of content:** Other types of shareable content, including downloaded files
- **Access method**: Storage Access Framework
- **Permissions needed:** None
- **Can other apps access?:** Yes, through the system file picker
- **Files removed on app uninstall?:** No

4. App preferences:

- **Type of content:** Keyvalue pairs
- **Access method:** Jetpack Preferences library
- **Permissions needed:** None
- **Can other apps access?:** No
- **Files removed on app uninstall?:** Yes

5. Database:

- **Type of content:** Structured data
- **Access method:** Room persistence library
- **Permissions needed:** None
- **Can other apps access?:** No
- **Files removed on app uninstall?:** Yes

SharedPreferences

Local data storage in Android can be achieved using SharedPreferences, a simple keyvalue pair storage mechanism provided by the Android framework. SharedPreferences allows you to store primitive data types such as boolean, float, int, long, and string persistently.

SharedPreferences in Android is a interface that allows applications to manage and access preferences, which are essentially settings or configuration data.When you create a set of preferences using `Context.getSharedPreferences(String, int)`, there's a single special copy that all parts of the app can use. If you want to make changes to these preferences, you need to use an "Editor" object. This ensures that modifications are handled consistently and saved properly. The objects you get using various "get" methods should be treated like pictures; avoid directly changing them and, instead, use the "Editor" object for any adjustments. In simpler terms, SharedPreferences helps apps store and retrieve their settings, ensuring a consistent and controlled approach to managing preference data.

Save data in a local database using Room

The Room persistence library provides an abstraction layer over SQLite to allow fluent database access while harnessing the full power of SQLite. In particular, Room provides the following benefits:

- **Compiletime verification of SQL queries.**
- **Convenience annotations that minimize repetitive and errorprone boilerplate code.**
- **Streamlined database migration paths.**

Because of these considerations, google highly recommend that you use Room instead of using the SQLite APIs directly.

The Room Persistence Library is a part of the Android Jetpack suite of libraries introduced by Google to simplify and improve the development of Android

applications. Specifically, Room is designed to provide a higherlevel, more abstract layer on top of SQLite, the traditional relational database system used in Android applications.

Here are some key features and components of the Room Persistence Library:

1. Entity:

Annotated classes that represent tables in the database. Each instance of the class corresponds to a row in the table.

2. DAO (Data Access Object):

Annotated interface or abstract class that defines the database interactions (queries, inserts, updates, deletes). It serves as a contract for how the application can interact with the database.

3. Database:

Annotated class that represents the database itself. It is an abstraction over the SQLiteOpenHelper class and serves as the main access point for the underlying database.

4. LiveData:

A part of the Android Architecture Components, LiveData is used to observe changes in the database and automatically update the UI when the underlying data changes.

5. ViewModel:

Another component of the Android Architecture Components, ViewModel is used to store and manage UIrelated data. It is often used in conjunction with Room to handle datarelated tasks.

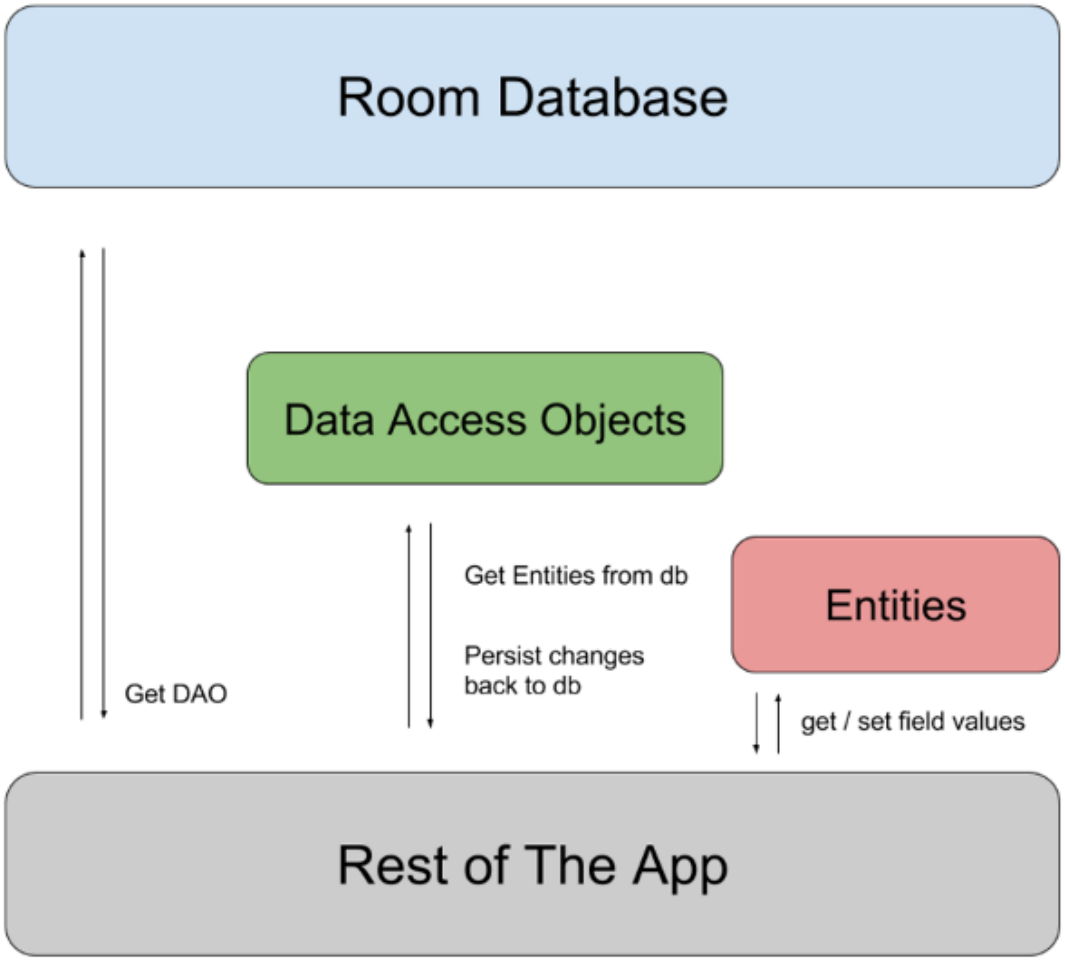
Room simplifies the process of working with SQLite databases in Android applications by providing a set of abstractions and annotations that eliminate much of the boilerplate code traditionally associated with database operations. It enforces compiletime checks for SQL queries and allows developers to work with the database using plain Java or Kotlin methods, making the code more readable and maintainable.

Primary components

There are three major components in Room:

- **The database class that holds the database and serves as the main access point for the underlying connection to your app's persisted data.**
- **Data entities that represent tables in your app's database.**
- **Data access objects (DAOs) that provide methods that your app can use to query, update, insert, and delete data in the database.**

The database class provides your app with instances of the DAOs associated with that database. In turn, the app can use the DAOs to retrieve data from the database as instances of the associated data entity objects. The app can also use the defined data entities to update rows from the corresponding tables, or to create new rows for insertion. Figure 1 illustrates the relationship between the different components of Room.



Here's a simplified example of how you might define a simple entity, DAO, and database using Room:

```
// Define an entity
@Entity(tableName = "user")
public class User {
   @PrimaryKey
   public int userId;

   @ColumnInfo(name = "user_name")
   public String userName;
}

// Define a DAO
@Dao
public interface UserDao {
   @Query("SELECT * FROM user")
   List<User> getAllUsers();

   @Insert
   void insert(User user);

   @Delete
   void delete(User user);
}

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

This example represents a simple User entity with associated DAO methods, and it defines an abstract database class (`AppDatabase`) that extends `RoomDatabase`. Developers can then use these components to interact with the database without directly dealing with SQLiterelated complexities.