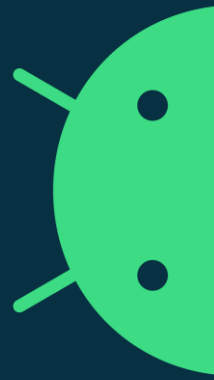
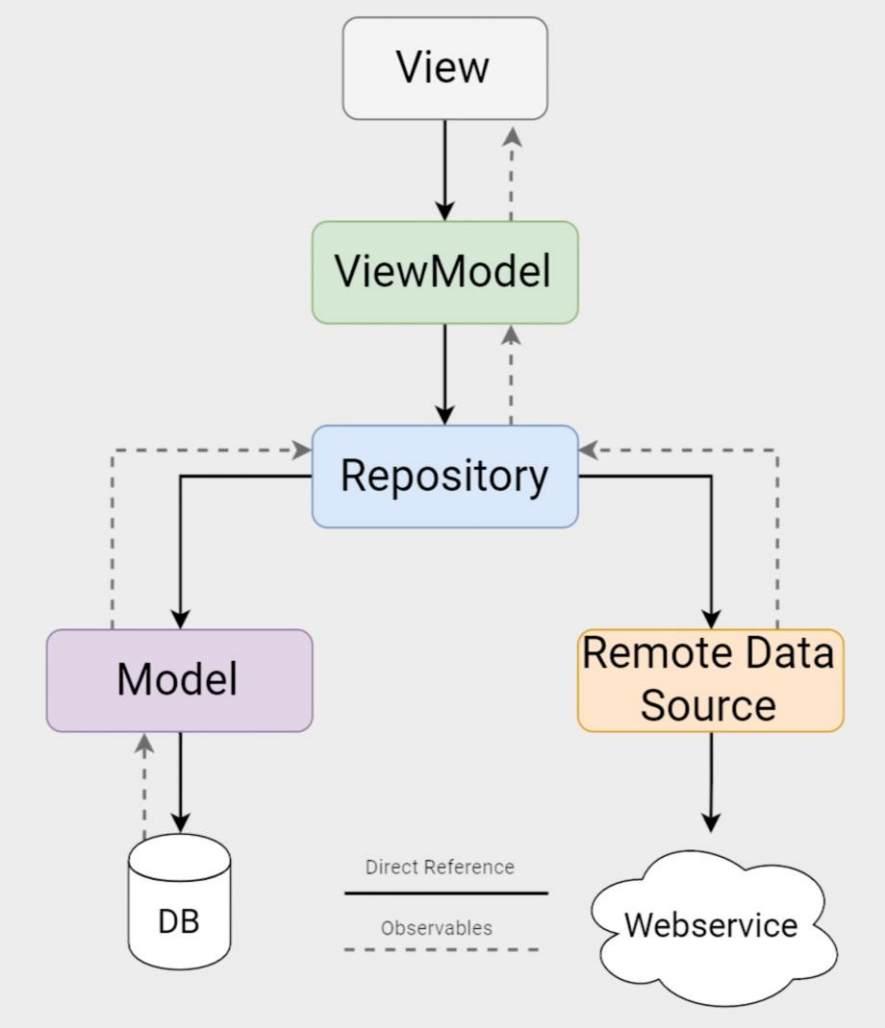
Data layer: (1) Room
CMPS 312



MVVM

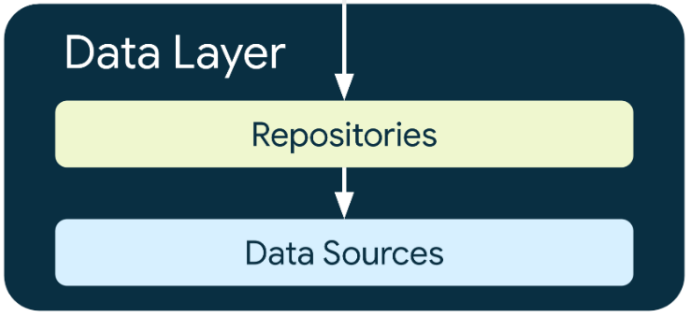
Model
View
ViewModel



[Data layer | Android Developers]

The data layer:
- contains application data and business logic.
- determines how application data must be created, stored, and changed.

The data exposed by this layer should be immutable



Repository classes are responsible for the following tasks:
- Exposing data to the rest of the app.
- Centralizing changes to the data.
- Resolving conflicts between multiple data sources.
- Abstracting sources of data from the rest of the app.
- Containing business logic.

Often, when a repository only contains a single data source and doesn't depend on other repositories, developers merge the responsibilities of repositories and data sources into the repository class.

Data source classes are the bridge between the application and the system for data operations.
- Each data source class should have the responsibility of working with only one source of data, which can be a file, a network source, or a local database.

Classes in the data layer generally expose functions to perform one-shot Create, Read, Update and Delete (CRUD) calls or to be notified of data changes over time.

androidx.room | Android Developers

Room is a Database Object Mapping library that makes it easy to access database on Android applications.

Primary components in Room:

- **Database**: This annotation marks a class as a database. It should be an abstract class that extends **RoomDatabase**.
- **Entity**: This annotation marks a class as a database row. For each Entity, a database table is created to hold the items.
- **Dao**: This annotation marks a class or interface as a **D**ata **A**ccess **O**bject.

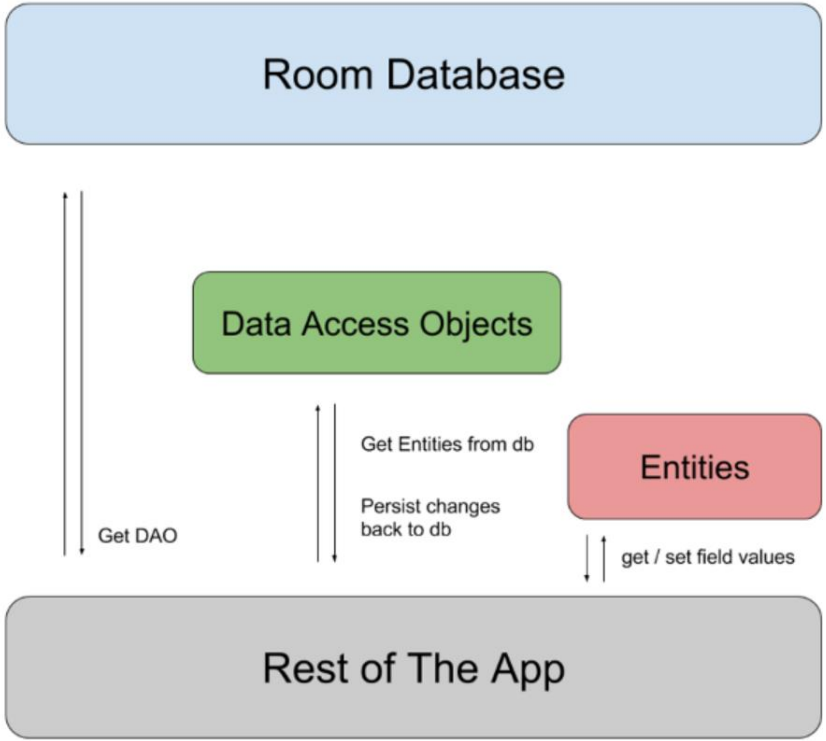
[Save data in a local database using Room | Android Developers]

The **Room** persistence library provides an abstraction layer over **SQLite**.

- To use Room in your app, add the required dependencies to your app's build.gradle file.

Primary 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.



[Save data in a local database using Room | Android Developers]

Implementation:

1. Create Data entity
2. Create Data access object (DAO)
3. Create Database
4. Create An Instance of The Database
5. Get An Instance of The DAO Using The Database Instance
6. Use The Methods in The DAO Instance to Interact With The Database

[Save data in a local database using Room | Android Developers]

Implementation:

1. Create Data entity

```
@Entity
data class User(
    @PrimaryKey val uid: Int,
    @ColumnInfo(name = "first_name") val firstName: String?,
    @ColumnInfo(name = "last_name") val lastName: String?
)
```

[Save data in a local database using Room | Android Developers]

Implementation:

2. Create Data access object (DAO)

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

    @Query("SELECT * FROM user WHERE uid IN (:userIds)")
    fun loadAllByIds(userIds: IntArray): List<User>

    @Query("SELECT * FROM user WHERE first_name LIKE :first AND " +
           "last_name LIKE :last LIMIT 1")
    fun findByName(first: String, last: String): User

    @Insert
    fun insertAll(vararg users: User)

    @Delete
    fun delete(user: User)
}
```

[Save data in a local database using Room | Android Developers]

Implementation:

3. Create Database

```
@Database(entities = [User::class], version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun userDao(): UserDao
}
```

[Save data in a local database using Room | Android Developers]

Implementation:

4. Create An Instance of The Database

```
val db = Room.databaseBuilder(
        applicationContext,
        AppDatabase::class.java, "database-name"
    ).build()
```

[Save data in a local database using Room | Android Developers]

Implementation:

5. Get An Instance of The DAO Using The Database Instance

```
val userDao = db.userDao()
```

[Save data in a local database using Room | Android Developers]

Implementation:

6. Use The Methods in The DAO Instance to Interact With The Database

```
val users: List<User> = userDao.getAll()
```

Team Study

In preparation to effectively use Room DB you need to conduct **Team studies** on the topics below:

1. **SQLite**
2. **Kotlin Coroutines**

SQLite Home Page

SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine.

- SQLite understands most of the standard SQL language. But it does omit some features while at the same time adding a few features of its own.
- SQLIite Language Query Language Understood by SQLite
- SQLite Tutorial SQLite Tutorial - An Easy Way to Master SQLite Fast
- Room Annotations, for example @Entity Entity | Android Developers Other annotations can be accessed too with this link.

Team Study

SQLite

Collectively as a team:

- Decide on DB tables “Entities” needed for you project
- Implement these tables “Entities” as required by Room DB
- Write all SQL queries needed by your project
- For each query, write an abstract method having a name, parameter(s), and return type
- Implement all of these methods in a DAO as required by Room DB

Coroutines guide | Kotlin (kotlinlang.org)

Kotlin Coroutines

Coroutines provide mechanisms for asynchronous or non-blocking programming in Kotlin.

- Coroutines on Kotlinkang.org Coroutines guide | Kotlin (kotlinlang.org)
- Kotlin coroutines on Android https://developer.android.com/kotlin/coroutines
- Use Kotlin Coroutines in your Android App https://developer.android.com/codelabs/kotlin-coroutines#0
- Additional resources for Kotlin coroutines and flow Additional resources for Kotlin coroutines and flow | Android Developers

Team Study

Kotlin Coroutines

- Each member scans through relevant material using the provided resources.
- Discuss your understanding within the team

Collectively as a team:

- Decide where and why you need to use coroutines in your project and