CMPS 312 Mobile App Development
Lab 10 – Data Layer

Objective

In this Lab, you will **build a Todo app that persists data in a local SQLite database.** You will use Room library and coroutines and practice the following skills:

- Create Entity classes.
- Create Data Access Objects (DAO) to map DAO methods to SQL queries.
- Perform database CRUD operations.
- Create and interact with a SQLite database using Room library.
- Handle database relations such as one to many relationships.
- Create cascade delete and enforce integrity checks using foreign keys.
- Use Database Inspector to interact with the SQLite database.

Figure 1 illustrates the Data Layer you will implement in this Lab as part of MVVM architecture.

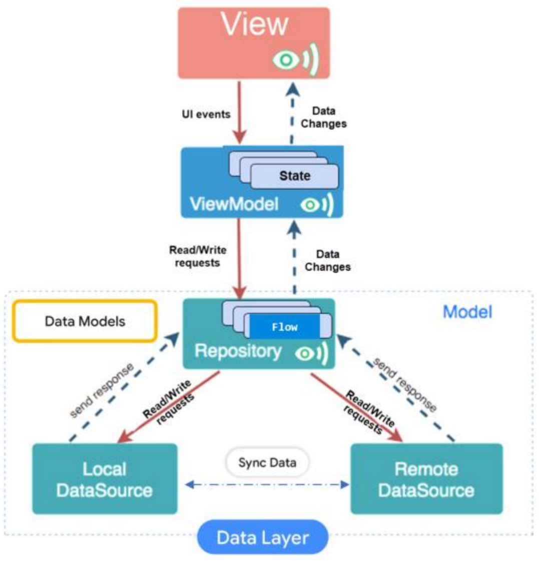


Figure 1. MVVM architecture

Figure 2 depicts the interconnection among the ToDo repository, Project and ToDo models, ProjectDoa and TodoDao data sources, and the ToDo Database.

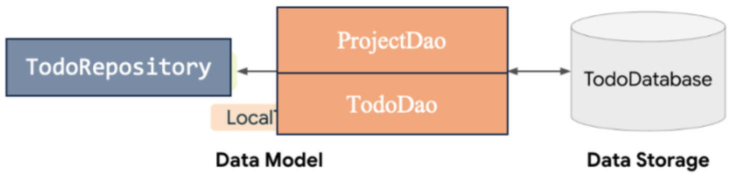


Figure 2. Relationship between ToDo repository, Project and ToDo models, ProjectDoa and TodoDao data sources, and ToDo Database

Preparation

1. Sync the Lab GitHub repo and copy the **Lab 10-Data Layer** folder into your repository.
2. Add the following inside plugins of app's `build.gradle`

```
plugins {
    id ("com.google.devtools.ksp") version "1.9.10-1.0.13"
}
```

Then add the following room dependencies

```
    val roomVersion = "2.6.0"
    implementation("androidx.room:room-runtime:$roomVersion")
    annotationProcessor("androidx.room:room-compiler:$roomVersion")
    // Kotlin Symbol Processing (KSP) - for processing annotations
    ksp("androidx.room:room-compiler:$roomVersion")
    // Kotlin Extensions and Coroutines support for Room
    implementation("androidx.room:room-ktx:$roomVersion")

    // Kotlin Datetime
    implementation("org.jetbrains.kotlinx:kotlinx-datetime:0.4.1")

    // Required to be able to use collectAsStateWithLifecycle
    implementation("androidx.lifecycle:lifecycle-runtime-compose:2.6.2")
```

PART A: Implementing the Todo App

Implement a Todo app to allows users to track todo tasks per projects. The user can add a project and subtasks under each project. The user can also update and delete both projects and todos. If the user deletes a project, then all associated todos should also be deleted.

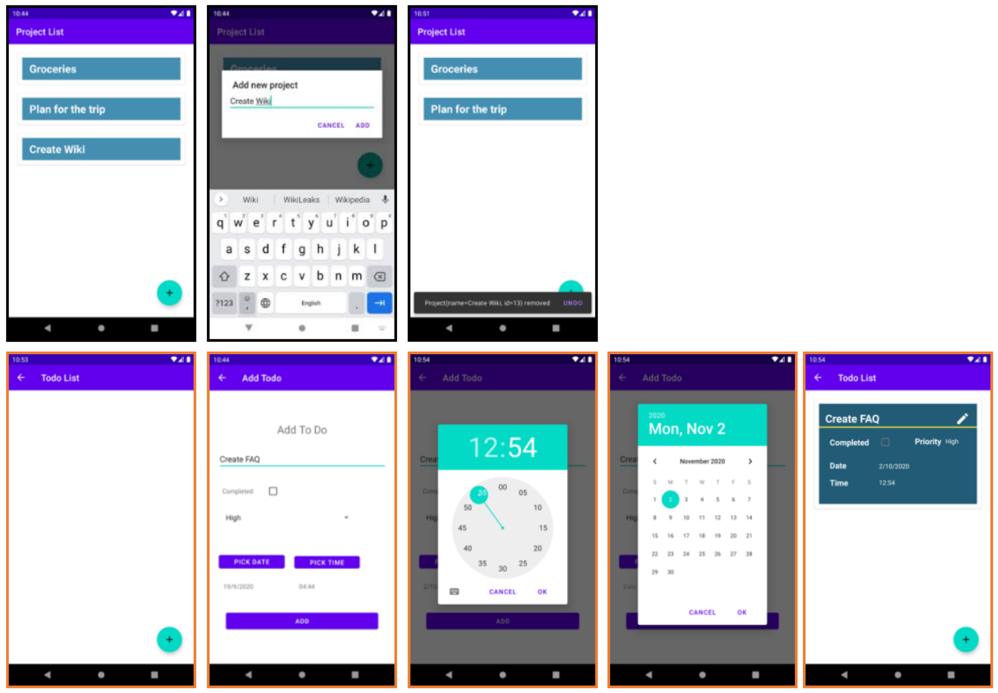


Figure 3: ToDo app UI design

Create the entities

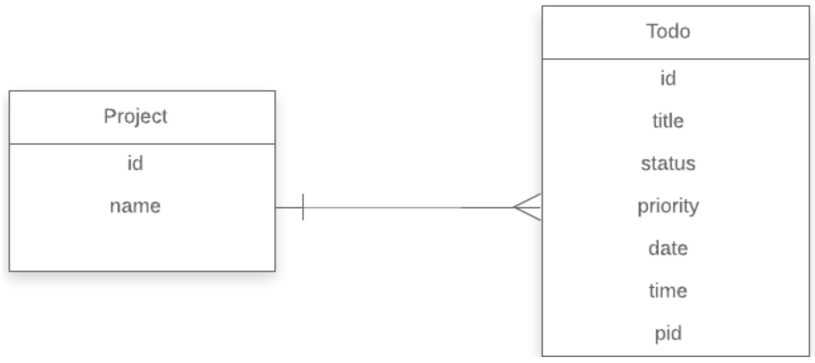


Figure 4. Todo App Entity Relations (ER) diagram

1. Open the entity package entity and create two data classes as shown in the Entity Relations (ER) diagram presented in Figure 4.
2. Annotate the Project class with @Entity annotation.
   Annotate the id parameter of the Project as a primary key **@PrimaryKey(autoGenerate = true)**
3. Annotate the Todo entity with the following annotation to create a **one to many** relationship with the Project entity.

```
@Entity(
    foreignKeys = [
        ForeignKey(
            entity = Project::class,
            parentColumns = ["id"],
            childColumns = ["pid"],
            onDelete = ForeignKey.CASCADE,
            onUpdate = ForeignKey.CASCADE
        )
    ]
)
```

- Annotate the id with **@PrimaryKey(autoGenerate = true)**
- Annotate the pid (project id) with @ColumnInfo(index = true)

I. Create the data sources as DAO Interface

1. Create **`ProjectDao`** interface under the **`datasource`** package and annotate with @Dao. Then add the following methods. Annotate the methods with the appropriate @Query, @Delete , @Insert , @Upsert annotations. E.g.,

```
@Query("select * from Project")
fun observeProjects(): Flow<List<Project>>
suspend fun addProject(project: Project)
suspend fun deleteProject(project: Project)
```

2. Create **`TodoDao`** interface under the **`datasource`** package and annotate with @Dao. Then add the following methods. Annotate the methods with the appropriate @Query, @Delete , @Insert , @Upsert annotations.

```
fun observeTodos(pid : Int): Flow<List<Todo>>
suspend fun getTodo(id: Int): Todo
suspend fun upsertTodo(todo: Todo) : Long
suspend fun deleteTodo(todo: Todo): Int
```

II. Create the Room Database class

Create a Room database abstract class that extends RoomDatabase and annotated with `@Database`. It should provide a singleton dbInstance object created using `Room.databaseBuilder()` to create (if does not exit) and connect to the database. Also, it serves as the main access point to get the DAOs to interact with the database.

1. Create a public abstract class named `TodoDB` that extends `RoomDatabase`. The class is abstract because Room will generate the implementation.

   Annotate the class with **`@Database`** and pass as arguments: list the app entities and the version number.

```
@Database(entities = [Todo::class, Project::class], version = 1)
```

2. Inside the class define 2 abstract methods to return the return the DAOs created earlier. Room will generate the implementation body.

```
abstract fun projectDao(): ProjectDao
abstract fun todoDao(): TodoDao
```

3. Create a companion object that returns an instance of `TodoDB`. Only one instance (a singleton) of the database object is needed for the whole app.

   Use `Room.databaseBuilder()` to create the database only if it doesn't already exist. Otherwise, return the existing instance.

The complete code for the companion object is shown below.

```
companion object {
    private var db: TodoDB? = null

    fun getDatabase(context: Context): TodoDB {
        if (db== null) {
            db= Room.databaseBuilder(
                context.appContext,
                TodoDB::class.java,
                "todoDB"
            ).fallbackToDestructiveMigration().build()
        }
        return db as TodoDB
    }
}
```

III. Create the Repository

1. Implement `TodoRepository` class that call the methods on `ProjectDao` and `TodoDao` to read/write data from the database.

   Create an instance of the todoDao by instantiating the database object and getting the dao instance

```
private val projectDao by lazy {
    TodoDB.getDatabase(context).projectDao()
}


private val todoDao by lazy {
    TodoDB.getDatabase(context).todoDao()
}
```

2. Implement the repository functions by calling the corresponding `ProjectDao` and `TodoDao` functions. For the repository should allow the following:
   - **Add** a new project
   - **Update** an existing project
   - **Get** all projects
   - **Delete** a project **and** all associated **Todos**
   - **Add** a new todo
   - **Update** an existing todo
   - **Get** all todos for a project id
   - **Delete** a specific todo.
3. Run and test your implementation.

IV. Query One to Many relationship

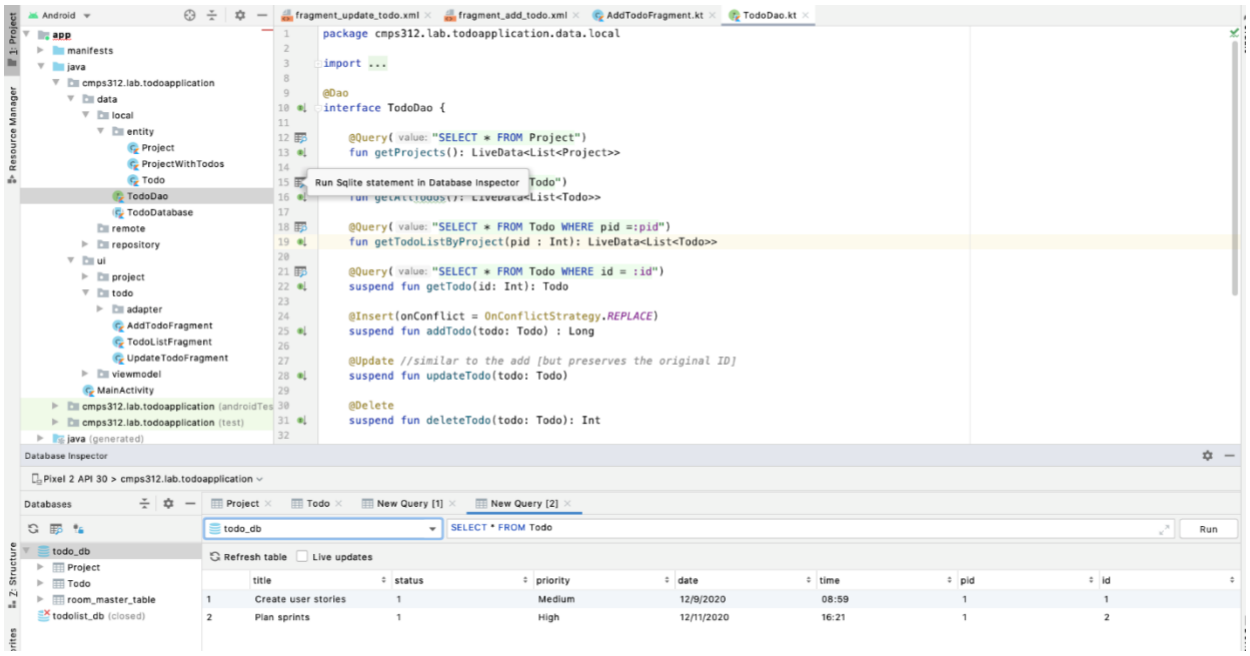
An important feature of a relational database is the ability to query data from multiple tables. Using @Relation annotation you can easily get a project and its associated todos in one query.

1. Create a new **ProjectWithTodos** data class
2. Under the class create two properties. A project property of type Project and another list property of type Todo named todos.
3. Annotate the project property with `@Embedded val project: Project`
4. Annotate the todos property with `@Relation(parentColumn = "id", entityColumn = "pid")`
5. Add `getProjectWithTodos` to TodoDAO class to get all the projects with their to-dos. Make sure you add @Transaction as this method will run multiple SQL statements. @Transaction will ensure that all of them are executed as one unit of work.

```
@Transaction
@Query ("SELECT * FROM Project")
suspend fun getProjectWithTodos(): List<ProjectWithTodos>
```

V. Test the database queries using Database Inspector

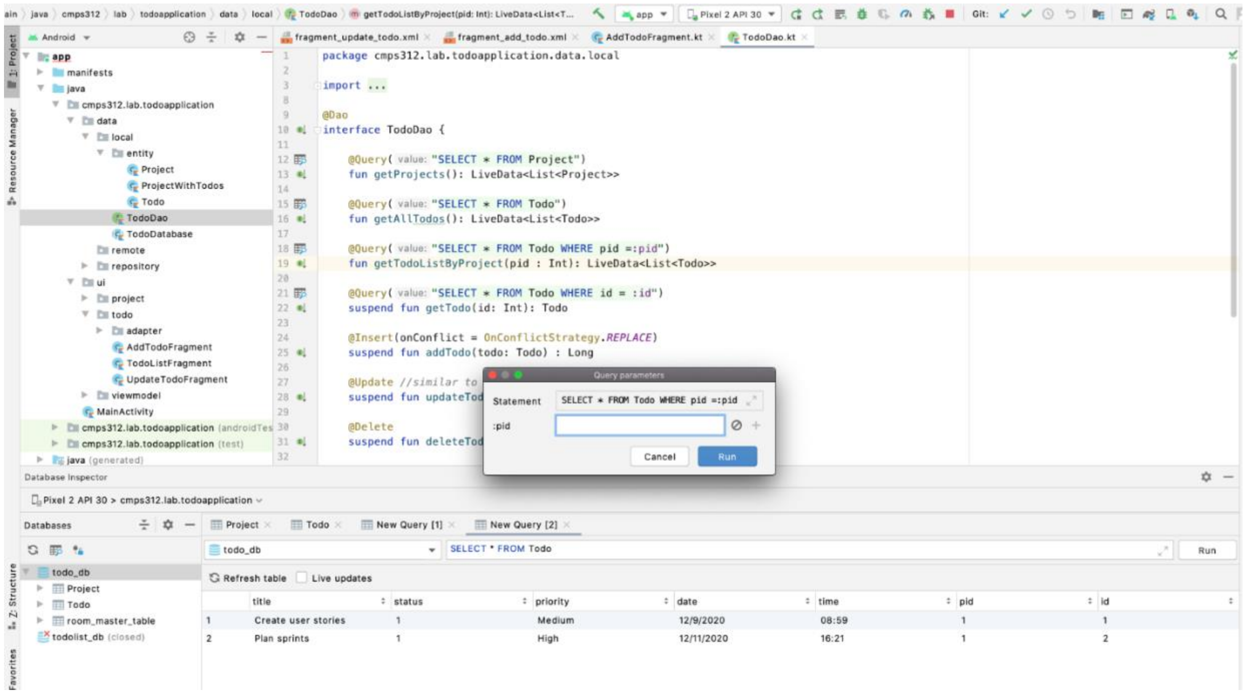
Test the app queries using Android Studio *Database Inspector*. This helps you write and test your queries before using them in the DAOs. Try to run all the queries used the DAOs interface. Try other queries that we did not implement.



```
package cmps312.lab.todoapplication.data.local

import ...

@Dao
interface TodoDao {

    @Query( value: "SELECT * FROM Project")
    fun getProjects(): LiveData<List<Project>>

                                             Todo")
    fun getAllTodos(): LiveData<List<Todo>>

    @Query( value: "SELECT * FROM Todo WHERE pid =:pid")
    fun getTodoListByProject(pid : Int): LiveData<List<Todo>>

    @Query( value: "SELECT * FROM Todo WHERE id = :id")
    suspend fun getTodo(id: Int): Todo

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun addTodo(todo: Todo) : Long

    @Update //similar to the add [but preserves the original ID]
    suspend fun updateTodo(todo: Todo)

    @Delete
    suspend fun deleteTodo(todo: Todo): Int

```

```
SELECT * FROM Todo
```

	title	status	priority	date	time	pid	id
1	Create user stories	1	Medium	12/9/2020	08:59	1	1
2	Plan sprints	1	High	12/11/2020	16:21	1	2



```
package cmps312.lab.todoapplication.data.local

import ...

@Dao
interface TodoDao {

    @Query( value: "SELECT * FROM Project")
    fun getProjects(): LiveData<List<Project>>

    @Query( value: "SELECT * FROM Todo")
    fun getAllTodos(): LiveData<List<Todo>>

    @Query( value: "SELECT * FROM Todo WHERE pid =:pid")
    fun getTodoListByProject(pid : Int): LiveData<List<Todo>>

    @Query( value: "SELECT * FROM Todo WHERE id = :id")
    suspend fun getTodo(id: Int): Todo

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun addTodo(todo: Todo) : Long

    @Update //similar to
    suspend fun updateTod

    @Delete
    suspend fun deleteTod
```

Query parameters

Statement: `SELECT * FROM Todo WHERE pid =:pid`

:pid

Cancel | Run

Database Inspector

Pixel 2 API 30 > cmps312.lab.todoapplication

`SELECT * FROM Todo`

	title	status	priority	date	time	pid	id
1	Create user stories	1	Medium	12/9/2020	08:59	1	1
2	Plan sprints	1	High	12/11/2020	16:21	1	2

PART B: Implement the Banking Apps Database

Following the same techniques as Part A, extend the Banking App introduced in the previous week's 9 implementation, which interacted with the Bank Web API, to use a local database. Your task is first to

1. Cache the data we retrieved from the WebAPI into the Local database
2. Make the applications viewmodel talk to the Local database instead of the server web api.
3. The app should work the same way as before.

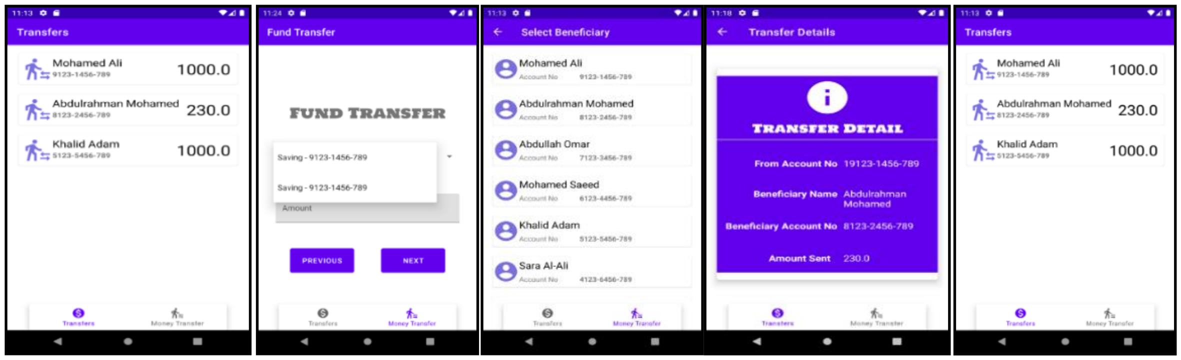


Figure 1 Banking App