CMPS 312 Mobile App Development
Lab 10 – Data Management

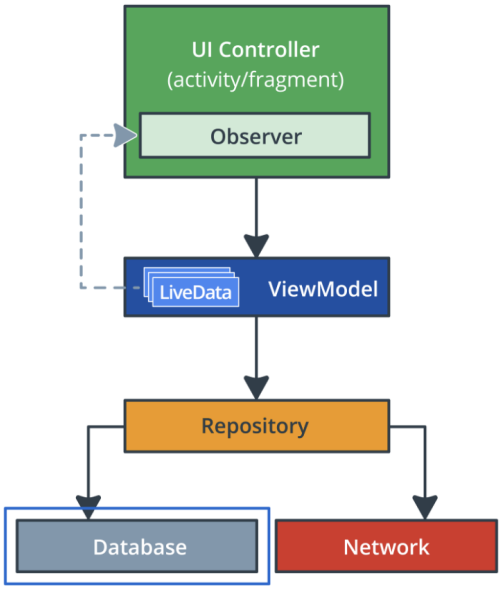
Objective

In this Lab, you will **build a to-do list application that persists data offline.** You will be using the room database library in conjunction with coroutines to get, add, update, and delete to-dos in Sqlite Database.

By the end of this Lab, you will learn,

- How to create and interact with a Room database to persist data.
- How to create a data class that defines a table in the database.
- How to use a data access object (DAO) to map Kotlin functions to SQL queries.
- How to perform database CRUD operations
- How to create Database relations using room such as one to many
- How to create cascade delete and update records using foreign keys
- How to use the Database Inspector to interact with the SQLite database

The image below shows how the Room database fits in with the overall architecture recommended in our Lab.



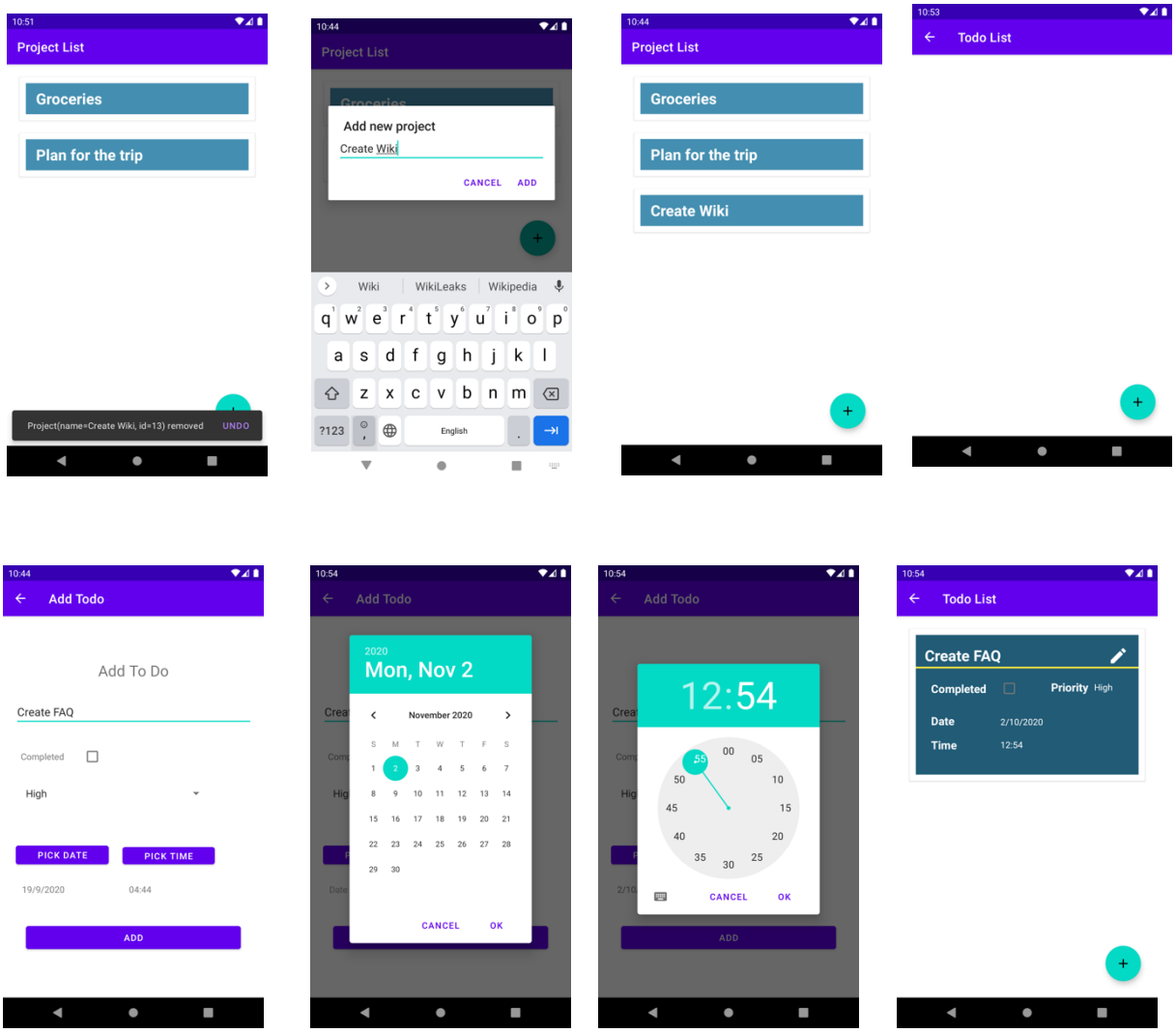
Preparation

1. Sync the Lab GitHub repo and copy the **Lab 10-Data Management** folder into your repository.
2. Add the following room dependencies

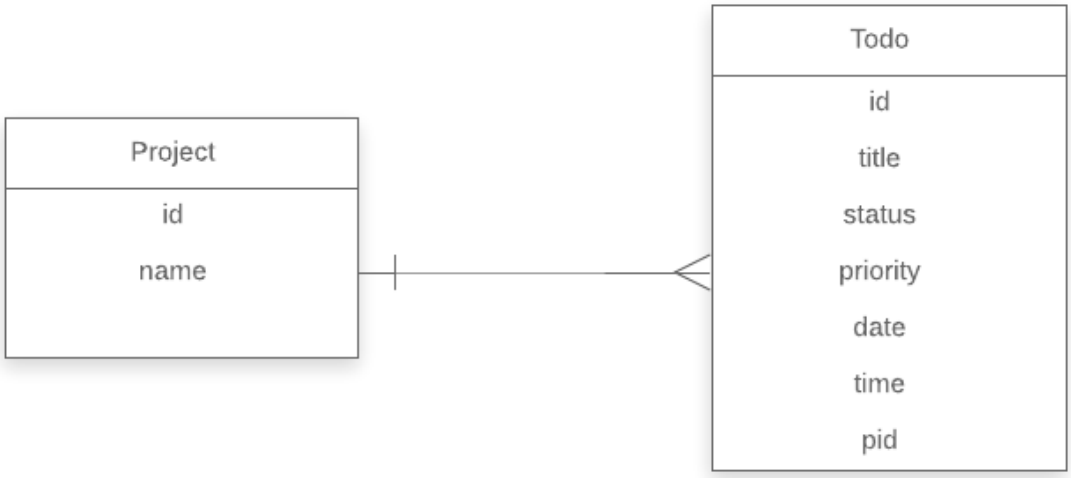
```
//     Room components
    def room_version = "2.2.5"

    implementation "androidx.room:room-runtime:$room_version"
    kapt "androidx.room:room-compiler:$room_version"

    // optional - Kotlin Extensions and Coroutines support for Room
    implementation "androidx.room:room-ktx:$room_version
```

PART A: Implementing the To-do List Application

You will be implementing the following todo list application shown below. The application allows users to add projects. And under each project, the user can then add subtasks or todos under each project. The user also can update and delete both projects as well as the todos. If the user deletes a project then all sub-tasks or todos should also be deleted.



Task 1: Creating the entities



1. Open the entity package inside the data/local/entity and create two data classes. The classes should have the above parameters.
2. Annotate the Project class with @Entity annotation.
3. Annotate the id parameter of the Project as a primary key **@PrimaryKey(autoGenerate = true)**
4. Open the Todo class and add the following annotation.

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

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

Task 2: Creating the DAO Interface

For the todo app database, you need to be able to do the following:

- **Insert** new project, todo.
- **Update** an existing project
- **Update** an existing todo
- **Get all the todos** for a specific project based on the project key
- **Get all projects**, so you can display them.
- **Delete** a project **and** all associated **Todos.**
- **Delete** a specific todo.

1. Under the data/local package create a new kotlin file and select interface. You should name the interface TodoDao

```
@Dao
interface TodoDao {}
```

2. Inside the body of the interface, add all the below functions to accomplish the above database operations.

```
fun getProjects(): LiveData<List<Project>>
fun getTodoListByProject(pid : Int): LiveData<List<Todo>>

suspend fun getTodo(id: Int): Todo
suspend fun addTodo(todo: Todo) : Long
suspend fun updateTodo(todo: Todo)
suspend fun deleteTodo(todo: Todo): Int
suspend fun deleteProject(project: Project)
suspend fun addProject(project: Project)
```

3. Use @Query, @Update, @Delete and @Insert annotations to achieve the above database operations.
   Eg. To get all projects you should annotate the getProjects function as follows.

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

Task 3: Creating the Room Database

In this task, you create a Room database that uses the Entity and DAO that you created in the previous task. This class has one method that either creates an instance of the database if the database doesn't exist, or returns a reference to an existing database.

1. Create a public abstract class named TodoDatabse that extends RoomDatabase. This class is to act as a database holder for our todo list application. The class is abstract because Room creates the implementation for you.

   ```
   abstract class TodoDatabase : RoomDatabase()
   ```

2. Annotate the class with **@Database**. In the arguments, declare the entities for the database and set the version number.

   ```
   @Database(entities = [Todo::class, Project::class], version = 2,
   exportSchema = false)
   ```

3. Inside the class define an abstract method or property that returns a **TodoDao**. The room will generate the body for you.

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

4. Create a companion object that will return the instance of the todo list database. You only need one instance of the Room database for the whole app, so make the RoomDatabase a singleton.

5. Use Room's database builder to create the database only if the database doesn't exist. Otherwise, return the existing database.

The complete code for the companion object is shown below.

```
companion object {
    @Volatile // [we don't want it to be cached..always the users should
get the updated instance]
    private var database: TodoDatabase? = null

    /*protected from concurrent execution by multiple threads by the
monitor of the instance*/
    @Synchronized
    fun getDatabase(context: Context): TodoDatabase {
        if (database == null) {
            database = Room.databaseBuilder(
                context.applicationContext,
                TodoDatabase::class.java,
                "todo_db"
            ).fallbackToDestructiveMigration().build()
        }
        return database as TodoDatabase
    }
}
```

Task 4: Creating the Repository

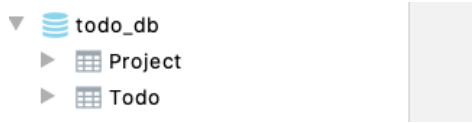
1. Open the repository class and add implements to the TodoDao Interface. It should generate similar functions to the DAO interface. Once the functions are generated remove the implements TodoDao and Override annotation from the methods.
2. Create an instance of the todoDao by instantiating the database object and getting the dao instance

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

3. Implement all the functions that you generated by calling the respective dao function.
4. Run and Test your code.

Task 5: Implementing one to Many Relationship

An important part of designing a relational database the ability to pull data from multiple tables in meaningful ways. In Room, we have support for all possible relations between tables: one-to-one, one-to-many and many-to-many, with one annotation: @Relation. In the following task, we will try aggregate data from the two tables Todo and Project and return all projects with their todos in one query.

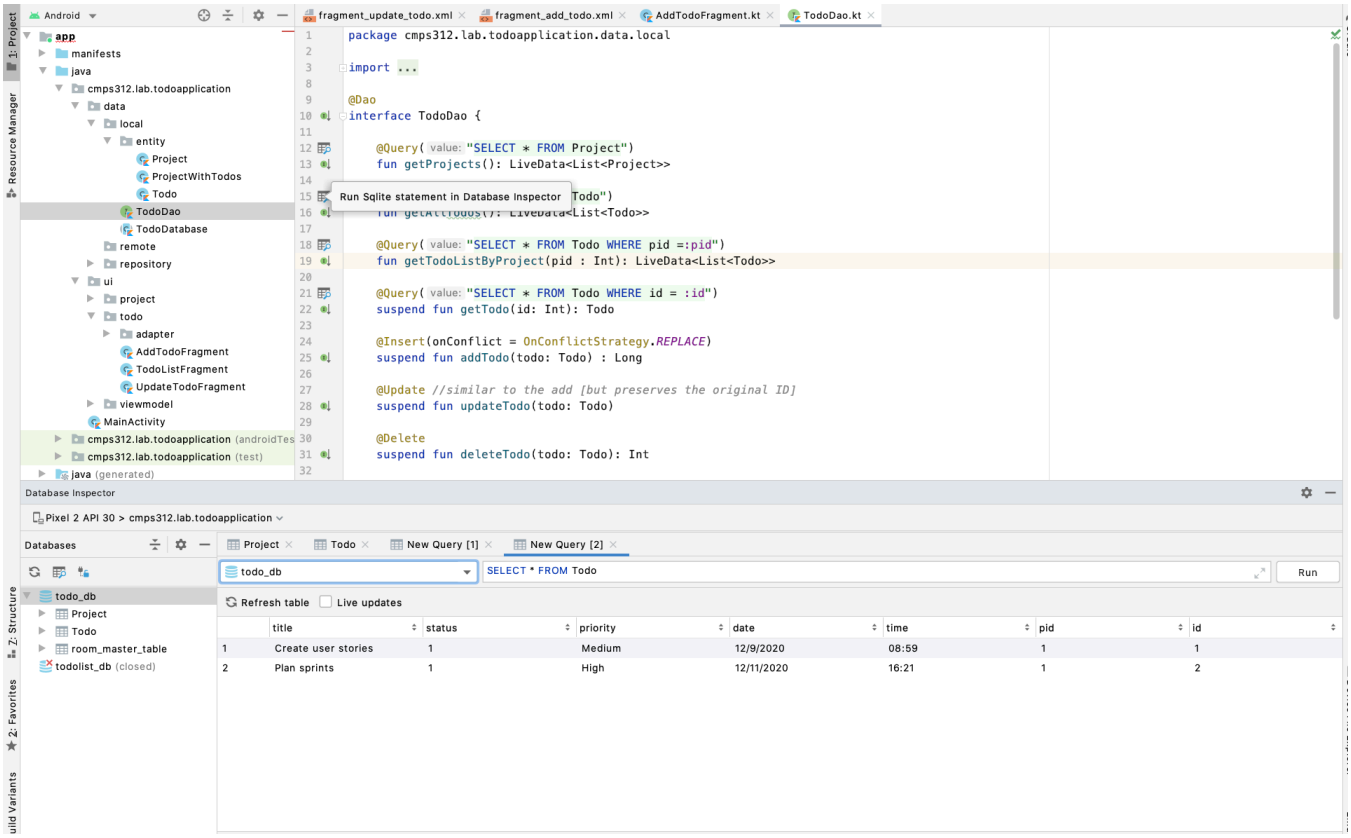


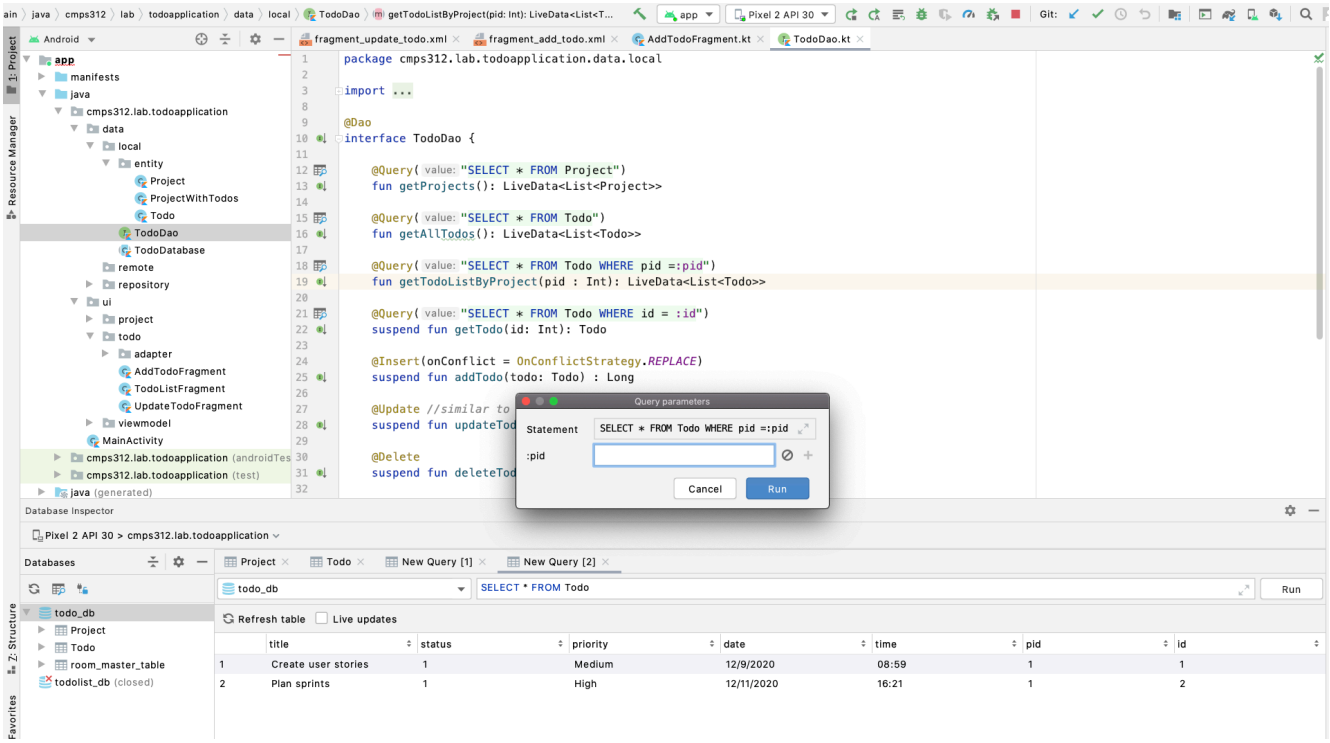
1. Create a new data class and name it **ProjectWithTodos**
2. Under the class create two properties. A project property of type Project and another list property of type Todo named todos.
3. Annotate the project property with `@Embedded val project: Project`
4. Annotate the todos property with `@Relation(parentColumn = "id", entityColumn = "pid")`
5. Open the TodoDAO class and use this new one to many relational class to get all the project with their to-dos, Make sure you add @Transaction as this type of query runs multiple SQL statements.

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

Task 6: Testing the database queries using Database Inspector

Test the app using the inspector. This helps ensure that the database works before you build more functionalities to it. Try to run all of the queries inside 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>>
@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
Query parameters
Statement
SELECT * FROM Todo WHERE pid =:pid
:pid
Cancel
Run
Database Inspector
Databases
todo_db
Project
Todo
room_master_table
todolist_db (closed)
SELECT * FROM Todo
Run
Refresh table
Live updates
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