**TodoList App Documentation**

**1. Project Overview**

TodoList App is a mobile application built with Kotlin that handles multiple to-do lists. This is Assignment 01 for the Mobile Development module and focuses on the integration of Kotlin with Jetpack Compose for UI and Room for database operations, giving it the full feel of managing your to-do list. The documentation details every component and code file in detail, with explanations of classes, functions, and the overall logic of main features.

**2. Core Classes and Their Roles**

**2.1 MainActivity.kt**

MainActivity is the application's entry point; it creates UI and sets up the components needed.

**Properties**

taskViewModel: TaskViewModel instance responsible for managing data and interactions with Room Database

**Functions**

onCreate: Initializes taskViewModel, which sets up the main content by using setContent, then sets up a click handler to go to AddTaskActivity.

onResume :reload any lists of tasks if user leaves the main screen to some other screen and returns from that screen to the main one, so all the edits they made on that new screen are reflected

**TaskApp Composable Function**

it is the main UI: here lists and tasks appear. It has: variables of state management to edit a task, lists, to move a task dynamic listing with tasks according to the current ViewModel data in the present: number of tasks; the due dates.

Dialogs: Have a dialog specially for editing a task's name, one to edit list's names, and dialog for task moving. Such dialogs are given that is followed by an associated ViewModel's functions in order to modify the data accordingly.

**2.2 Task.kt**

This data class denotes information about a single task.

**Member variables**

id: auto-generated unique integer to be used to identify.

listId: the id to know which specific list task belongs

name: Not necessary, has to be provided during the creation process; It denotes the name of that specific task

dueDate: a non-essential filed as a task can either be given without a date to due on, or there could be a specific time that after how much date, or when will be done.

completed: a boolean flag specifying that this task is considered completed or not

**2.3 TaskList.kt**

The data class of every list of tasks the application creates.

**Values**

id: it would automatically produce a unique number to represent an id to identify it among various TaskLists.

name: Each list could be identified through this and the app prohibits to enter some names so they get appropriately identified

**2.4 TaskDao.kt**

The TaskDao interface performs DAO functionalities dealing with a Task, an action performing on TaskTable in relation to insert function.

**Critical functions**

insertTask: The user inserts new entry of his task to table by making method call.

getTasksForList: Returns all the tasks of a list specified with the listId.

updateTaskCompletionStatus: Set the completion status of the task based on the interaction between the user and the program.

updateTaskDetails: Change the name or due date when a task is edited.

deleteTaskById: Remove task based on id

updateTaskListId: It changes the list by putting a task into a list. This can be achieved when we update the listId

**2.5 TaskListDao.kt**

**Interface for the DAO operations over the TaskList is performed**

**Methods Used for Operations**

getAllTaskLists: Returns all the task list in the database

insertTaskList: Inserts new TaskList.

getListById: Retrieve a list by its id, especially for validation.

updateListName: Update the name of the list. So that List always has unique names.

**2.6 TaskViewModel.kt**

TaskViewModel is what you might call the data handler, working in collaboration with TaskDao and TaskListDao to do all operations with respect to data as well as managing state. ViewModel makes an use of Kotlin Flow; it observes as well as emits changes to UI and assures real-time data updates on the entire app.

**Properties**

database : An instance of AppDatabase that links the ViewModel to Room database.

taskListDao, taskDao: DAO references to execute on the database.

\_taskLists, \_tasksByListId, \_taskCountsByListId: The flow of mutable state. Holds the task lists, tasks, and task counts correspondingly.

**Key Methods**

loadTaskLists: Load task lists from the TaskListDao and update the \_taskLists. Load tasks for each list by triggering

loadTasksForAllLists: Load all tasks found in the database along with the list.

updateTaskCounts: Calculate the count of all tasks and complete ones for every list for indicators in the UI.

addTaskList: It adds a new task list and then reloads lists to update the UI.

updateTaskCompletionStatus: It updates the completion status of a task and refreshes task counts.

updateListName: It changes the name of a particular task list.

updateTask: This feature allows the user to edit the name or due date of a task.

deleteTask: It deletes a task by ID and removes it from the current list.

getTaskCountsForList: it returns counts (total and completed) on a specific list, whose purpose is only to display the completion status of the task in the UI;

moveTaskToList: it acts by changing a task's listId to move from one list to another;

**3. User Interface and UX Design**

**3.1 UI Overview**

TodoList App employs Jetpack Compose for building declarative UI, which itself changes its shape according to the data's state.

**Major UI elements are :**

Task List View: The main view will display all the task lists along with their count and number of tasks done. The user can make new lists, rename existing list names, and delete a list.

Tasks in List View: All the tasks of the lists will be viewed with a checkbox option to mark it as complete or not, title of the task and due date if created

**3.2 UI Elements**

Add Task List Input On the main screen, add new lists which comes with the ability to input a task through input field and button on a screen that utilizes ViewModel state management for real-time update.

Add Tasks button for every list For each list, there is the option of adding the user's tasks, where adding a button will open an intent to open the AddTaskActivity and get details inputted.

Task Completion Checkbox: A checkbox is placed beside each task to indicate completion and ViewModel updates the status of that task in the database to synchronize

Due Date Indicators: The color of a due date is different from others for a task, yellow for the tasks which are to be due today while red for the over due tasks.

Editing and Moving Tasks: Editable dialogs can be used to alter task names, due dates, and to move tasks from one list to another. Such dialogs will keep the application interactive and user-friendly.

**3.3 Error Handling and Input Validation**

The application checks for user input in order to avoid the following situations.

Duplicate List Names : The application does not allow duplicate names because each list has to have a unique name.

Empty Input Prevention: The task list and tasks must have a non-empty name to create.

**4. Development Challenges and Solutions**

Real-Time UI Synchronization: This was one of the difficult challenges to maintain UI updates in real-time. With Kotlin Flow, the ViewModel has updated the UI in real-time with respect to data changes. This makes the user experience seamless.

Due Date Indicators: Dynamic date-based indicators needed due dates to be calculated with respect to the current date, thereby facilitating effective prioritization of tasks.

Task Movement Across Lists: The movement of tasks between lists required updating the listId in the database and managing state updates in the ViewModel. This was done using a dialog and ensuring smooth data flow updates.

**5. Final Thoughts and Reflection**

The TodoList App project puts into effect a comprehensive task management experience by including some of the most important notions, such as data persistence with Room, UI creation with Jetpack Compose, efficient data flow with ViewModel and Kotlin Flow. The design and development of the project thus reflect the best practices in the mobile development domain, including, among others, modularity, real-time data treatment, and intuitive use interface.

Every class within this project has been put to use in achieving some functionalities needed. The MainActivity will be the main screen used to display lists and control whatever the user is doing, while TaskViewModel ensures that all the data is in place and hence the app is responsive, and DAOs, room entities provide reliability in storage of data, thus when all data is lost in the app, it would not crash.

The UI is declarative and dynamic but also follows modern design principles when using Jetpack Compose. The application structure can be enhanced further to include reminders or connection to online storage.

TodoList App is a rich introduction to Android app development because it allows students to get hands-on experience in building, managing, and extending real-world applications.