**Module 3 Critical Thinking Assignment: To Do List App**

Alexander Ricciardi

Colorado State University Global

CSC475: Platform-Based Development

Professor Herbert Pensado

March 2, 2025

**Module 3 Critical Thinking Assignment: To Do List App**

This documentation is part of the Module 3 Critical Thinking Assignment from CSC475: Platform-Based Development at Colorado State University Global. The documentation provides an overview of the Android application's functionality and testing scenarios including application pseudocode, Kotlin code lines, and the application’s output screenshots. It also reflects on the obstacles faced during the application’s development, and the skills acquired. The application is coded in Kotlin 1.9.24 and Jetpack Compose 2024.04.01, and is named “To Do List App.”

**The Assignment Direction:**

Option #1: “To Do List”

Challenge: Develop a to-do list application that allows users to add, delete, and mark tasks as completed. Implement data storage using SQLite database to store and retrieve the to-do items. Focus on understanding data persistence concepts in Android.

Please ensure that your submission includes the following components:

Source code file(s) containing the program implementation.

A 1-page paper explaining the program's purpose, the obstacles faced during its development, and the skills acquired. The paper should also include screenshots showcasing the successful execution of the program.

Compile and submit your pseudocode, source code, and screenshots of the application executing the application, the results, and the GIT repository in a single document.

**Program Description:**

The program is a small Android application that allows the user to manage a to do list.

* The app uses the Model-View-ViewModel (MVVM) architecture.
* The app uses Jetpack Compose to generate its UI.
* The user can add, delete, and complete tasks.
* The tasks are prioritized by relevance.
* The tasks can be displayed sorted by priority.
* The app uses SQLite to store task data.

**⚠️My notes:**

* I added a sort functionality allowing the user to display tasks by priorities from Urgent to Low.
* Methods to update tasks and update task priorities are added to the database backend operation, but not implemented in the front end, this needs to be implemented in the future version of the app.

**Git Repository:**

This is a picture of my GitHub page:

A screenshot of a computer

AI-generated content may be incorrect.

I use [GitHub](https://github.com/) as my Distributed Version Control System (DVCS), the following is a link to my GitHub, [Omegapy](https://github.com/Omegapy).

My GitHub repository that is used to store this assignment is named [My-Academics-Portfolio](https://github.com/Omegapy/My-Academics-Portfolio) and the link to this specific assignment is:

<https://github.com/Omegapy/My-Academics-Portfolio/tree/main/Platform-Based-Dev-Android-CSC475/Module-3-Critical-Thinking>

**Project Map:**

* Module 3 Critical Thinking Assignment.docx (this file, App documentation)

The project files from the app, file structure:

├── MainActivity.kt # (view layer)

├── model/ # Model layer

│ └── Task.kt # Core data structures

├── data/ # ViewModel layer

│ ├── TaskRepository.kt # Data operations

│ ├── DatabaseInitializer.kt # Database setup

│ └── database/ # Database

│ └── TaskDatabase.kt # SQLite operations

└── ui/ # View layer

├── screens/ # Main screens

│ └── TodoListScreen.kt # Main task list screen

├── components/ # UI components

│ ├── TaskItem.kt # Individual task

│ └── AddTaskDialog.kt # Task creation dialog

└── theme/ # UI styling

├── Theme.kt # Material theme

├── Color.kt

└── Type.kt

**Reflection**

This app was personally challenging as it is my first program that I can truly categorize as an application. The process of making it was difficult; I had to scrap my project on Android Studio and start over several times. Although Jetpack Compose is integrated into Android Studio as part of the Android Jetpack libraries, managing the composable functions’ dependencies was the most problematic, in the sense that I had to constantly import composable library components, and I had to ask myself many times the following question: "Am I missing an import, or is it just a typo? Or maybe it’s a deprecated function?".

I was using composable functions from a Udemy tutorial that was using outdated data information. I even rebuilt my gradle.kts file several times to support my outdated composable functions, and in some instances, this created more problems than it fixed. I settled for the last version of Jetpack Compose by extensively using the “Jetpack Compose Documentation” (Android Developers, n.d.a). I also went through the four lessons of the “Jetpack Compose Tutorial” (Android Developers, n.d.b) provided by Android. The lessons are basic but very useful to use the fundamentals of Compose.

On the other hand, implementing SQLite for data persistence was a piece of cake compared to implementing the UI with Compose. I use a *TaskDatabase* class, which extends *SQLiteOpenHelper* to handle task data operations.

* The *SQLiteOpenHelper* class is a helper class from Android's SQLite database libraries, it helps manage CRUD operations, it acts as an abstraction layer between the Kotlin code and SQLite.
* The *TaskDatabase* class manages the task data (CRUD) at the SQL database level by implementing the *SQLiteOpenHelper* class functionality (methods).

I created a data class named *Task* to manipulate Task objects representing task data to be created, retrieved, deleted, or updated from the SQLite database. Where each row in a SQL table named *tasks* represents a *Task* object, see code snippets below

**Code Snippet 1**

*SQL tasks Table Code*

CREATE TABLE tasks (

      id INTEGER PRIMARY KEY AUTOINCREMENT,

      description TEXT NOT NULL,

      priority TEXT NOT NULL,

      is\_completed INTEGER NOT NULL DEFAULT 0

  )

*Note:* The SQL code is a presentation of the *tasks* table

**Code Snippet 2**

*Task Objects*

data class Task(  
 val id: Long = 0,  
 val description: String,  
 val priority: Priority,  
 val isCompleted: Boolean = false  
)

enum class Priority {  
 HIGH,  
 MEDIUM,  
 LOW,  
 URGENT  
}

*Note:* The Code Snippet represents the *Task* class used to instantiate a *Task* object, the enumerator *Priority* class used by the *Task* class to set the priority of each *Task* object.

I use the Model-View-ViewModel (MVVM) approach to structure my project, separating the application logic from the UI elements of the application. The tables below provide a description of each class, Composable Function, and helper function in the project, listing what MVVM layer they belong to and in which project file they can be found.

**Table 1**

*Model Layer*

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | File | Description |
| TasK | Data Class | Task.kt | data representing a single task with ID, description, priority, and completion status |
| Priority | Enum | Task.kt | Priority levels (URGENT, HIGH, MEDIUM, LOW) for tasks |
| TaskDatabase | Class | TaskDatabase.kt | SQLite database helper for managing task data persistence |

**Table 2**

*Model Layer TaskDatabase Class Methods*

|  |  |  |
| --- | --- | --- |
| Method | Type | Description |
| getAllTasks | Getter | Gets all tasks from the database |
| getTask | Getter | Gets a specific task by ID |
| getTasksByCompletionStatus | Getter | Gets tasks filtered by completion status |
| getTasksByPriority | Getter | Gets tasks filtered by priority |
| insertTask | Modifier | Inserts a new task into the database |
| updateTask | Modifier | Updates an existing task |
| deleteTask | Modifier | Deletes a task from the database |
| deleteCompletedTasks | Modifier | Deletes all completed tasks |

**Table 3**

*ViewModel Layer*

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | File | Description |
| TaskRepository | Class | TaskRepository.kt | Mediator for data access to the View layer |
| DatabaseInitializer | Class | data/DatabaseInitializer.kt | Utility/test for initializing the database with fake data |

**Table 4**

*ViewModel TaskRepository Class Methods*

|  |  |  |
| --- | --- | --- |
| Method | Type | Description |
| getTasks | Getter | Gets all tasks from the database |
| getTaskById | Getter | Gets a specific task by ID |
| getCompletedTasks | Getter | Gets all completed tasks |
| getActiveTasks | Getter | Gets all active (not completed) tasks |
| getTasksByPriority | Getter | Gets tasks filtered by priority |
| addTask | Modifier | Adds a new task to the database |
| updateTask | Modifier | Updates an existing task |
| deleteTask | Modifier | Deletes a task from the database |
| deleteCompletedTasks | Modifier | Deletes all completed tasks |
| updateTaskCompletionStatus | Modifier | Updates the completion status of a task |
| updateTaskPriority | Modifier | Updates the priority of a task |

**Table 5**

*ViewModel DatabaseInitializer Class Methods*

|  |  |  |
| --- | --- | --- |
| Method | Type | Description |
| initializeWithSampleData | Modifier | Initializes the database with fake tasks |

**Table 6**

*View Layer*

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | File | Description |
| MainActivity | Class | MainActivity.kt | Main Class, sets up the UI and repository |
| TodoApp | Composable | MainActivity.kt | Main composable function |
| TodoListScreen | Composable | TodoListScreen.kt | Main screen displaying the list of tasks |
| TaskItem | Composable | TaskItem.kt | Displays a single task with checkbox, description, priority, and delete button |
| AddTaskDialog | Composable | AddTaskDialog.kt | Dialog for adding new tasks with description and priority selection |
| PrioritySelector | Composable | AddTaskDialog.kt | Component for selecting task priority |
| capitalize | Extension Function | Multiple files | Utility to capitalize the first letter of a string (used for priority display) |

*Note*: The tables illustrate the MVVM structure of the application, they also describe the classes, composable, and helper functions.

The relation between the class and Composable functions can be illustrated as follows:

Model Classes ↔ Repository Class ↔ MainActivity ↔ Composable Functions

(Data) (Operations) (Coordination) (UI Rendering)

In conclusion, making this app was challenging but through the process of creating, I learned valuable skills in Android development (compose Function and SQLite database manipulation) and app development in general. Skills such as implementing the MVVM approach to app structures, creating responsive and dynamic user interfaces with Jetpack Compose, implementing proper data persistence with SQLite databases, managing application state, handling user interactions, and debugging complex UI and application environment.

**Screenshots**

This section demonstrates the functionality of the application by using images illustrating the user interactions with the application and the application outputs from those interactions.

*See next page*

**Figure 1**

*Main Screen*

A screenshot of a cell phone

AI-generated content may be incorrect.

*Note:* The image shows the main screen of the To Do List application. The app is titled "My To Do List" at the top of the screen. It also displays the list of tasks, each with a colored circle on the left indicating priority level, a checkbox [ ] to mark completion status, the task description, and a red trash icon **🗑** on the right for deletion. In the bottom right corner, there's a blue square button [+] with a plus sign, which is used to add new tasks to the list. The tasks are color-coded by priority, with red representing "Urgent" tasks, orange for "High" priority, yellow for "Medium" priority, and green for "Low" priority tasks. The three-dot menu icon (⋮) in the top right corner of the screen is the "more options" menu, when clicked, it reveals a dropdown with three specific options, sort by priority, clear complete, and delete all. The data was populated by the *DatabaseInitializer* class.

**Figure 2**

*The More Options Drop Down Menu*

*A screenshot of a phone

AI-generated content may be incorrect.*

*Note:* When clickedthe three-dot menu icon (⋮) (topBar -> DropdownMenu section in the TodoListScreen()Composable function) of the "more options" menu reveals Sort by priority, Clear completed, and Delete all options.

*See next page*

**Figure 3**

*Sort by priority output*

A screenshot of a phone

AI-generated content may be incorrect.*A screenshot of a phone

AI-generated content may be incorrect.*

*A screenshot of a cell phone

AI-generated content may be incorrect.*A screenshot of a cell phone

AI-generated content may be incorrect.

*Note:* Clicking on the Sort by priority option ((isSortedByPriority) "Reset sort order" else "Sort by priority") sorts the tasks by priorities from urgent to low, clicking on the reset order, the sort is undone and the tasks would be displayed in the database order of entry.

**Figure 4**

*Add Task*

A screenshot of a phone

AI-generated content may be incorrect.A screenshot of a cell phone

AI-generated content may be incorrect.

*Note:* Clicking on the [+] add button (showAddTaskDialog(), trigger in the TodoListScreen() Composable function) it will open the add dialog window (AddTaskDialog() Composable function), there the user can add the new task. Note that the task was added at the end of the list and has a medium priority.

*See next page*

**Figure 5**

*Delete A Task*

A screenshot of a cell phone

AI-generated content may be incorrect.A screenshot of a phone

AI-generated content may be incorrect.

*Note*: Clicking on the **🗑** trach icon in the main screen (whenTaskDelete in the TodoApp() Composable function) will delete the task here, the first task in the list was selected for deletion

*See next page*

**Figure 6**

*Selecting Completed Tasks and Removing Them*

A screenshot of a cell phone

AI-generated content may be incorrect.A screenshot of a cell phone

AI-generated content may be incorrect.A screenshot of a cell phone

AI-generated content may be incorrect.

*Note:* Selecting task as complete [✓] (whenTaskCheckedChange in the TodoApp() Composable function) will mark the task as done. Using the three-dot menu icon (⋮) (topBar -> DropdownMenu section in the TodoListScreen()Composable function of the "more options" menu and clicking on the ✓ Clear completed, DropdownMenuItem(text = { Text("Clear completed") }, …), will remove all the completed tasks.

*See next page*

**Figure 7**

*Deleting All the Tasks*

A screenshot of a cell phone

AI-generated content may be incorrect.*A screenshot of a phone

AI-generated content may be incorrect.A screen shot of a cell phone

AI-generated content may be incorrect.*

*Note:* Clicking on the 🗑, DropdownMenuItem( text = **{** Text("Delete all") **}**,...), trash icon within the three-dot menu icon (⋮) of the "more options" menu will delete all the tasks.

As shown in figures 1 to 7, the app works as intended.

**References:**

Android Developers (n.d.). *Jetpack Compose Documentation │ Get started with Jetpack Compose*. Android. https://developer.android.com/develop/ui/compose/documentation