# TO DO LIST APP

***Submitted by***

**SHANMUGA DIVYA K** **(2116220701261)**

***in partial fulfillment for the award of the degree***

***of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**



**RAJALAKSHMI ENGINEERING COLLEGE**

**ANNA UNIVERSITY, CHENNAI**

**MAY 2025**

# RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

## BONAFIDE CERTIFICATE

Certified that this Project titled **"TO DO LSIT APP"** is the bonafide work of **"SHANMUGA DIVYA K (2116220701261)"** who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.


**SIGNATURE**

Dr. N. Durai Murugan ., M.E., Ph.D.,

**SUPERVISOR**

Assistant Professor

Department of Computer Science and Engineering,

Rajalakshmi Engineering College, Chennai-602 105.


Submitted to Project Viva-Voce Examination held on _____

**Internal Examiner** **External Examine**

# ABSTRACT

This project focuses on the development of a simple and efficient To-Do List mobile application using Kotlin in Android Studio. The main objective of the application is to provide users with a minimalistic and intuitive platform for managing their daily tasks. Users can easily add tasks, view them in a list, mark them as complete, and remove them when finished. The user interface is designed with simplicity in mind, using essential UI components such as EditText, Button, and Recycler View to enable dynamic task management.

The application uses Kotlin as the programming language for its modern syntax and interoperability with Android APIs, while Android Studio serves as the integrated development environment (IDE) for building and testing the app. The app utilizes a RecyclerView to efficiently handle and display task items, ensuring a smooth experience even with a large number of tasks. Task data is stored in-memory for simplicity, making the app lightweight and ideal for beginner-level development.

The design pattern follows standard Android architecture principles, offering a solid base for potential future enhancements like persistent storage or notification reminders. Overall, this project provides a practical introduction to Android app development using Kotlin and serves as a useful tool for understanding fundamental concepts such as UI layout, event handling, and list management.

## ACKNOWLEDGMENT

Initially we thank the Almighty for being with us through every walk of our life and showering his blessings through the endeavor to put forth this report. Our sincere thanks to our Chairman Mr. S. MEGANATHAN, B.E, F.I.E., our Vice Chairman Mr. ABHAY SHANKAR MEGANATHAN, B.E., M.S., and our respected Chairperson Dr. (Mrs.) THANGAM MEGANATHAN, Ph.D., for providing us with the requisite infrastructure and sincere endeavoring in educating us in their premier institution. Our sincere thanks to Dr. S.N. MURUGESAN, M.E., Ph.D., our beloved Principal for his kind support and facilities provided to complete our work in time. We express our sincere thanks to Dr. P. KUMAR, M.E., Ph.D., Professor and Head of the Department of Computer Science and Engineering for his guidance and encouragement throughout the project work. We convey our sincere and deepest gratitude to our internal guide, Dr. S. VINOD KUMAR , M.Tech., Ph.D., Professor of the Department of Computer Science and Engineering. Rajalakshmi Engineering College for his valuable guidance throughout the course of the project. We are very glad to thank our Project Coordinator, Mr. M. RAKESH KUMAR Assistant Professor Department of Computer Science and Engineering for his useful tips during our review to build our project.

**SHANMUGA DIVYA K  2116220701261**

# TABLE OF CONTENTS

# CHAPTER 1
# INTRODUCTION

## 1.1 GENERAL

It is a mobile-based To-Do List application developed using Kotlin programming language in Android Studio. The main goal of this system is to offer users a simple yet effective tool to manage daily tasks with a focus on usability, performance, and clarity. The app provides the ability to add, delete, and manage a list of tasks in an organized manner, all within a single-screen interface. This project is ideal for personal productivity and serves as an entry-level example of mobile app development using modern Android practices.

## 1.2 OBJECTIVE

The main objective of this project is to develop a simple and user-friendly mobile To-Do List application using the Kotlin programming language in Android Studio. The app helps users manage their daily activities by allowing them to add, view, and delete tasks easily from a single screen. It is designed to boost productivity and ensure that users do not forget their important tasks. The interface is clean and minimal so that even first-time users can use it without confusion. The focus is on keeping the app lightweight, fast, and responsive without using too many phone resources. It also works offline, so users can access their tasks anytime without needing internet connectivity. This project is perfect for students or beginner developers who want to learn how to build Android applications. It helps them understand the use of modern development tools like Android Studio, Kotlin, and UI components. Overall, this app combines learning and practicality, showing how to create a real-world mobile application that can actually be used in daily life. The idea is to keep things simple and efficient, offering only the core features required to manage personal tasks without any ads, extra permissions, or complicated settings.

## 1.3 EXISTING SYSTEM

Currently, there are many To-Do List applications available in the Play Store like Google Tasks, Microsoft To Do, Todoist, and many others. These apps are feature-rich, offering reminders, recurring tasks, cloud sync, and more. However, for a simple user who just wants to write down tasks and check them off, these apps may be too heavy or complicated. Most of them require internet access, login with Google or Microsoft accounts, and have advertisements or premium features that may distract users. Also, they consume more storage, RAM, and battery. For someone who needs only the basic functions like adding, deleting, and viewing tasks, these apps may not be ideal. The existing systems are not always beginner-friendly for developers either, as their codebases are complex. So, the proposed system solves this by providing a clean, offline-first app with no login or internet requirements. It is built with simplicity and clarity in mind, suitable for everyday users and also helpful as a beginner-level Android development project. It avoids all the extra features and focuses only on what's truly needed — managing daily tasks in the easiest possible way without any confusion or heavy setup.

# CHAPTER-2

# PROPOSED SYSTEM

## 2.1 GENERAL

The proposed system is a mobile-based To-Do List application developed using Kotlin programming language in Android Studio. The main goal of this system is to offer users a simple yet effective tool to manage daily tasks with a focus on usability, performance, and clarity. The app provides the ability to add, delete, and manage a list of tasks in an organized manner, all within a single-screen interface. This project is ideal for personal productivity and serves as an entry-level example of mobile app development using modern Android practices. The application will use a user-friendly graphical interface, enabling users to interact through basic UI components such as EditText, Button, and RecyclerView. The RecyclerView serves as the primary display mechanism for the list of tasks. It allows dynamic data handling and efficient updates, providing a smooth scrolling experience regardless of the number of items. When a user inputs a task in the EditText field and presses the "Add" button, the task will immediately appear in the list. Each task item in the list will also include a "Delete" button to remove the task and an optional checkbox or visual cue to indicate completion. The proposed system operates entirely on the client side, with in-memory data storage. This means that the tasks are not stored permanently and will be lost when the app is closed. While this approach is suitable for demonstration and learning purposes, the system is built in a modular way that allows easy future integration of persistent storage options like SQLite, Room Database, or Shared Preferences. From a software development perspective, the system adheres to modern Android development practices. It separates UI logic from data handling logic and uses Kotlin’s concise syntax to reduce boilerplate code. The design emphasizes simplicity and clarity, making it easy to maintain, understand, and extend. The use of lifecycle-aware components and support libraries ensures compatibility with a wide range of Android devices.

## 2.2 SYSTEM ARCHITECTURE DIAGRAM

The system architecture diagram shows the high-level design of how different parts of the To-Do List app work together. In this app, the architecture follows the MVVM (Model-View-ViewModel) pattern, which is commonly used in modern Android development. The **View** is the user interface where users add, view, or delete tasks. The **ViewModel** acts as a bridge between the UI and the **Model**, managing data and app logic. The **Model** includes the task data stored in a local database like Room. This layered structure helps keep the app clean, organized, and easy to maintain.

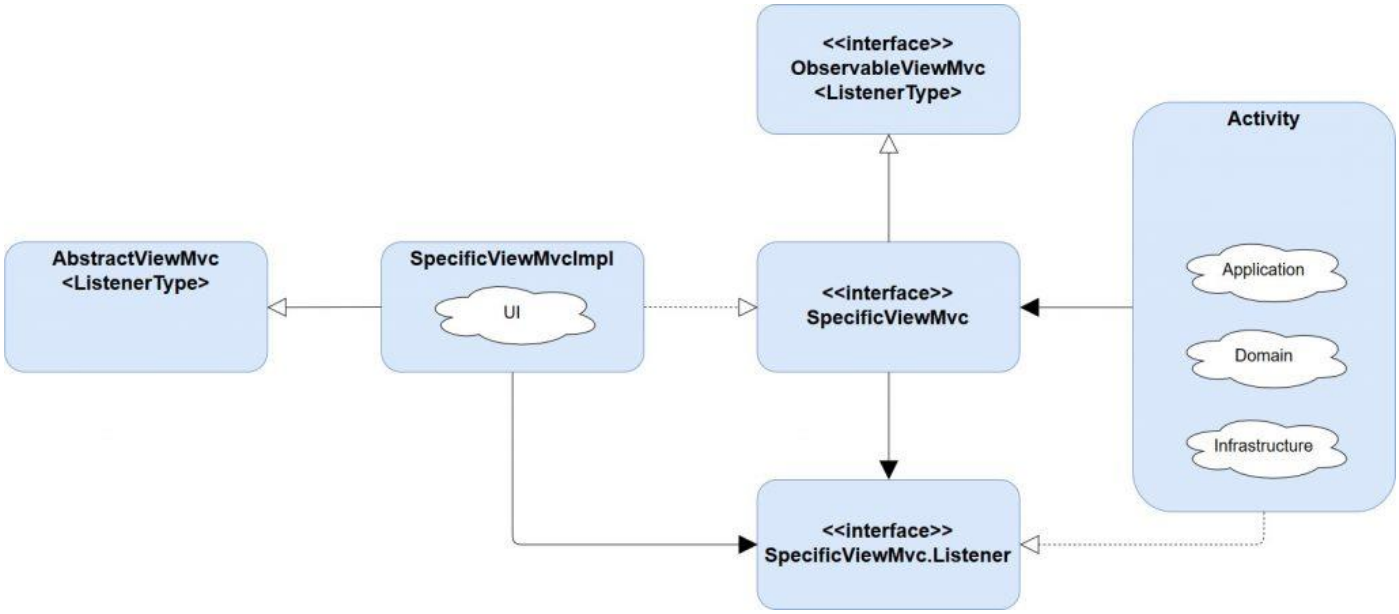



**Fig 2.1 System Architecture**

## 2.3 DEVELOPMENTAL ENVIRONMENT

### 2.3.1 HARDWARE REQUIREMENTS

To run and test the proposed To-Do List mobile application, basic hardware requirements must be met. Additionally, a minimum screen resolution of 1280x720 pixels is ideal for optimal app visibility.

**TABLE 2.1 HARDWARE REQUIREMENTS**

| Component | Specification |
|---|---|
| Processor | Inter Core i ( $10^{th}$ Gen or above)/AMD Ryzen 3 |
| RAM | 8 GB RAM |
| Storage | 512 GB HDD/SSD |
| Network | Stable Internet Connection |

### 2.3.2 SOFTWARE REQUIREMENTS

The development of this application requires Android Studio IDE, which includes built-in support for Kotlin. The latest stable version of Android Studio should be installed, along with the necessary SDK tools and an emulator for testing. Kotlin plugin support is essential as the app is developed entirely in Kotlin. The Gradle build system is used for compiling and managing dependencies. Additionally, Java Development Kit (JDK) version 8 or above is required. For version control and collaboration, Git may be used. On the mobile side, the app runs on Android OS version 6.0 or higher, with no additional software dependencies.

**TABLE 2.2 SOFTWARE REQUIREMENTS**

| Component | Specification |
|---|---|
| Operating system | Windows, macOS, or Linus (cross platform) |
| Programming Language | Kotlin |

## 2.4 DESIGN OF THE ENTIRE SYSTEM

### 2.4.1 ACTIVITY DIAGRAM

The activity diagram shows the flow of actions from the user's point of view. It begins with the app launch, where the main screen is shown. The user can then choose to add a new task, mark it as completed, or delete a task. Each user action triggers a response in the app. This diagram helps understand how the user interacts with the app step-by-step, covering all possible flows.

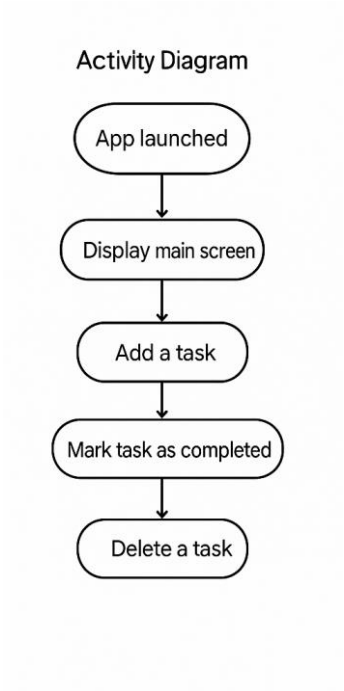



**Fig 3.2 Activity Diagram**

### 2.4.2 DATA FLOW DIAGRAM

The DFD shows how data moves inside the app. It includes the user, input (task details), processing (saving to database), and output (displaying tasks). It helps visualize how the app handles and processes user data in a structured way.

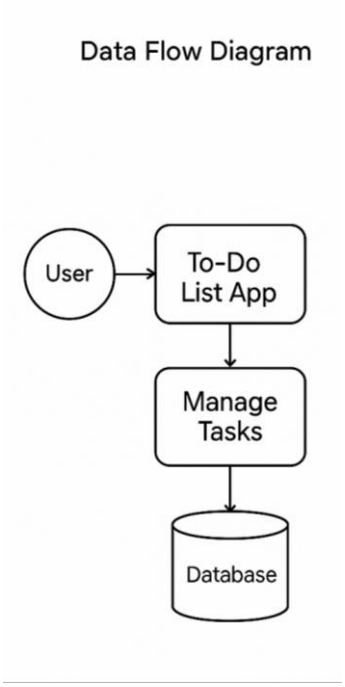



**Fig 3.3 Data Flow Diagram**

# CHAPTER-3

# MODULE DESCRIPTION

## 3.1 SYSTEM ARCHITECTURE

### 3.1.1 USER INTERFACE DESIGN

The User Interface (UI) of the To-Do List app is designed with simplicity, clarity, and usability in mind. It follows a single-screen design where users can interact with all features from the main interface. At the top, there is a header or title bar named "My Tasks." Below that, there is a text input field where users can type their task and a button labeled “Add Task” to save it. Tasks are displayed in a scrollable list view with checkboxes and delete icons beside each item. Checked items are marked as completed. The color scheme is minimal — mostly white backgrounds with accent colors for buttons and status highlights to ensure a clean and calm look.The design follows Android’s Material Design principles. Buttons are responsive, text is legible, and spacing ensures ease of touch. Animations like smooth transitions and swipe-to-delete make the app more interactive. Overall, the UI design ensures that users can quickly understand and use the app without any learning curve, especially helpful for first-time users or those looking for a fast and straightforward task manager.

### 3.1.2 BACKEND INFRASTRUCTURE

The backend infrastructure of the app is built using Android’s local storage system. It uses Room Database, a modern SQLite wrapper in Android Jetpack, to store and manage tasks efficiently. The architecture follows the MVVM pattern (Model-View-ViewModel), which separates UI, business logic, and data layers for better performance and maintainability. The Model represents the data class (Task), which defines the structure of each task (like task ID, name, and status). The ViewModel handles business logic, updating the UI based on changes in the database. It communicates with the Repository, which acts as a bridge between the ViewModel and the Room database. This helps in managing operations like insert, update, and delete with clean separation. Data is stored locally using Room, meaning no internet is needed. This ensures fast performance, offline access, and privacy. All database queries are handled on background threads to avoid freezing the user interface. LiveData and Kotlin coroutines are used to observe data changes and perform tasks asynchronously, ensuring the app stays smooth even when managing many tasks. No external servers or APIs are used, keeping the backend lightweight and secure for personal use.

## 3.2 DATA COLLECTION AND PREPROCESSING

- ✓ In this app, data collection and preprocessing are straightforward since it deals only with user-entered text tasks. The user types a task into the input field and presses the "Add Task" button. The entered data is first validated — blank or duplicate tasks are not allowed. After basic validation, the task is processed and converted into a data object with fields like taskName, isCompleted, and timestamp.

- ✓ Each task is stored as a record in the Room database. While there's no complex preprocessing like in machine learning, the app ensures that each task is trimmed of unnecessary spaces and converted to proper format before saving. Tasks are automatically sorted by their timestamp or status depending on implementation. Preprocessing also involves UI adjustments.

- ✓ For example, completed tasks can be shown as strikethrough or grayed out. When a task is checked or deleted, the corresponding database record is updated or removed. These changes are instantly reflected in the UI via LiveData observers.

- ✓ Overall, even though the data is simple, proper validation and structured storage ensure the app remains bug-free and efficient. This also lays the foundation for future enhancements like sync, analytics, or priority tagging.

## 3.3 SYSTEM WORKFLOW

The system workflow starts when the user opens the app. The **MainActivity** initializes and loads the ViewModel, which fetches data from the Room database and displays it. Users can enter a new task, which is validated and passed to the ViewModel. The ViewModel calls a repository function to insert the task into the database. Once added, the database sends updates to the UI using LiveData, so the task list refreshes automatically. If the user taps the checkbox next to a task, its completion status is updated. If the delete icon is tapped, the task is removed from the database. These updates are handled using Kotlin coroutines to run in the background, avoiding any lag. The app also handles lifecycle events. If the app is minimized and reopened, the data persists since it's stored locally. The entire workflow is reactive — the UI changes based on database changes, ensuring smooth user experience. The system remains responsive, minimal, and organized thanks to the MVVM structure, Room for storage, and Kotlin features for reactive programming.

# CHAPTER-4

# IMPLEMENTATION AND RESULTS

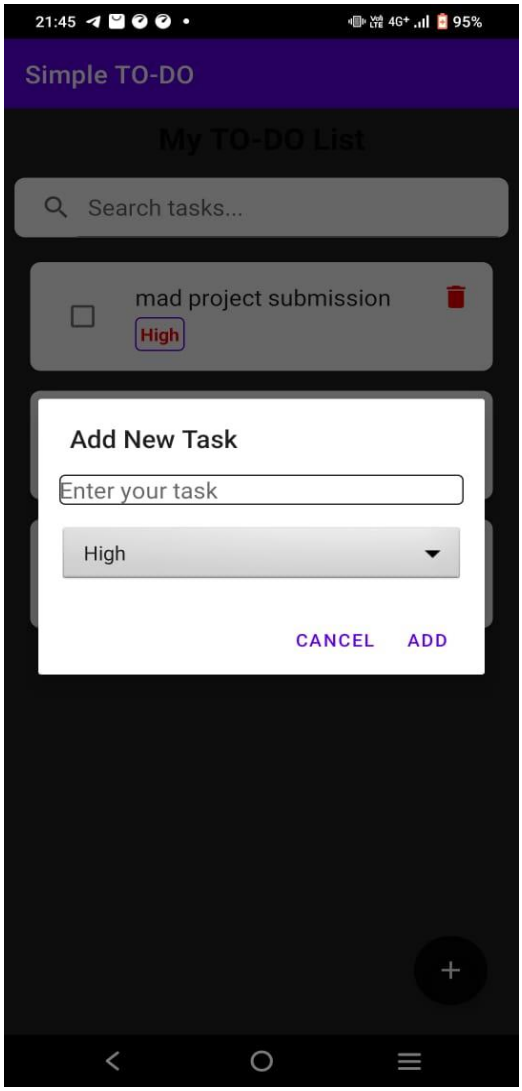
## 4.1 IMPLEMENTATION

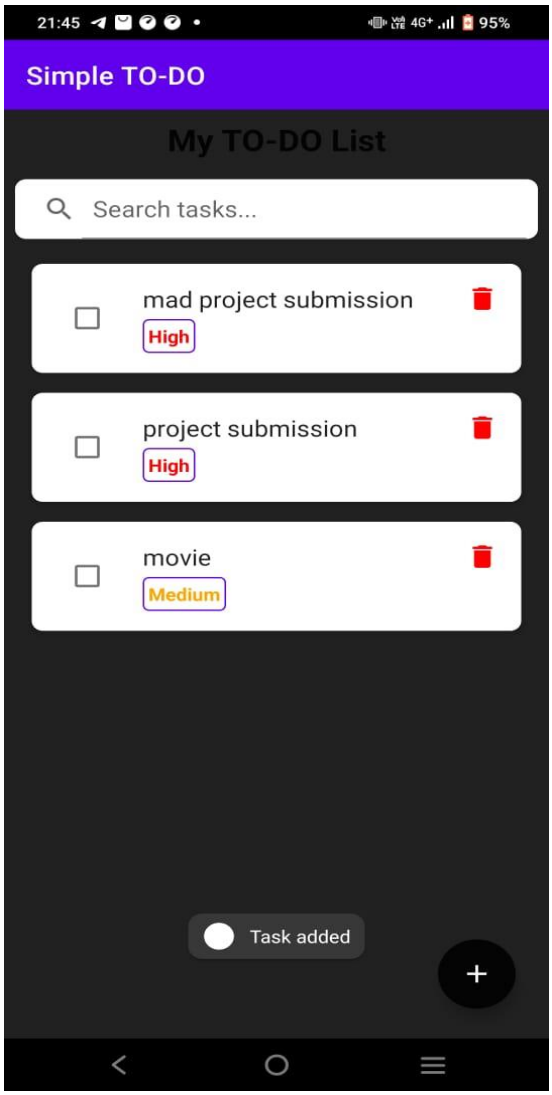
The To-Do List app is implemented using Android Studio with Kotlin as the programming language. First, the project structure is created with necessary dependencies for Room, Live Data, View Model, and Coroutines. A data class Task.kt defines the task structure. Then, DAO (Data Access Object) interfaces and Room Database classes are created to handle database operations like insert, update, and delete. The MainActivity.kt sets up the UI using XML layout files and observes Live Data from the View Model. When the "Add Task" button is pressed, the input is collected and sent to the View Model. If the input is valid, it is added to the Room database through the Repository. Live Data ensures that the Recycler View showing the task list gets updated automatically. The output is a functioning mobile app where users can add new tasks, mark them as completed, or delete them. Completed tasks appear in a different style (like strikethrough text). If a user restarts the app, the task list remains, showing data persistence. All operations like adding, deleting, and updating are fast, and the app does not crash even with rapid actions due to proper use of coroutines.The final result is a clean, working Android app that can be installed on any phone running Android 6.0 or higher. It meets the goal of simple daily task management without the need for internet or sign-in.

## 4.2 OUTPUT SCREENSHOTS

The output screenshot of the To-Do List application showcases a clean and minimalistic user interface, designed with ease of use in mind. At the top of the screen, there is a header displaying the app title, such as “My Tasks.” Below the header, a text input field allows users to type their new tasks. Beside the input field, an “Add” button is visible, which the user can tap to add the task to the list.Once a task is added, it appears immediately in a scrollable list view below the input section. Each task is displayed in a card or row with a checkbox on the left, the task text in the center, and a delete icon on the right. If the user marks the checkbox, the task is visually updated—typically with a strikethrough effect and a faded color—to indicate completion. When the delete icon is tapped, the task is instantly removed from the list. The screenshot also reflects real-time responsiveness. As tasks are added, updated, or removed, the changes are immediately visible on the screen without needing to refresh. This live update is made possible by the use of LiveData and the MVVM pattern. The app’s output also remains consistent when the app is reopened or rotated, thanks

to local data persistence using Room Database.The screenshot clearly demonstrates that the app is fully functional, visually neat, and capable of handling task management in a straightforward, user-friendly way without unnecessary clutter or complexity.



# CHAPTER-5

# CONCLUSION AND FUTURE ENHANCEMENT

## 5.1 CONCLUSION

The To-Do List mobile application successfully achieves its objective of providing a simple, offline, and user-friendly task management solution. Developed using Kotlin in Android Studio, it demonstrates key Android development practices like MVVM architecture, Room database integration, and responsive UI design. By focusing only on essential features—adding, viewing, completing, and deleting tasks—the app avoids unnecessary complexity. It ensures fast performance, data persistence, and ease of use, making it perfect for daily personal use or for beginners in Android development. Overall, the app serves both as a practical productivity tool and a learning project that reflects best practices in modern mobile development.

## 5.2 FUTURE ENHANCEMENT

While the current version of the app is simple and functional, there are several future enhancements that can greatly improve its utility and user experience:

1. **Task Notifications and Reminders:** Adding alarm or notification support can remind users about due tasks.

2. **Priority Tagging:** Users can categorize tasks as high, medium, or low priority using color codes or icons.

3. **Subtasks and Notes:** Allowing users to add subtasks or extra notes for each task can make the app more flexible.

4. **Cloud Sync and Backup:** Integrating cloud storage using Firebase or Google Drive can help users access their tasks across multiple devices and prevent data loss.

5. **Dark Mode Support:** Offering dark mode based on system theme would improve battery life and reduce eye strain.

6. **Voice Input:** Letting users add tasks using voice commands can make the app more accessible.

**REFERENCES:**

The development of this mobile-based To-Do List application was guided by several trusted references, including official documentation, open-source community contributions, and academic resources related to Android development. Below is a summary of the key references that supported the implementation:


1. **Android Developers Documentation**

   Google's official Android developer guide was the primary source of information. It provided detailed explanations of essential components like Activities, ViewModel,LiveData,and Room Database.

   *Source:* https://developer.android.com

2. **Kotlin Language Documentation**

   The official Kotlin documentation by JetBrains was used to understand language-specific features like null safety, data classes, lambda expressions, and coroutines. *Source:* https://kotlinlang.org/docs/home.html

3. **Android Jetpack Components**

   Jetpack libraries such as Lifecycle, ViewModel, LiveData, and Room were used extensively for building the app architecture and managing data in a lifecycle-aware manner.
   *Source:* https://developer.android.com/jetpack

4. **Stack Overflow Community**

   Community forums were helpful for troubleshooting coding errors, understanding best practices, and optimizing application logic.

   *Source:* https://stackoverflow.com