Trip Management Android Application

Submitted by

K.S.JAYASURIYAA 220701332

In partial fulfilment of the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

ANNA UNIVERSITY, CHENNAI

MAY 2025

RAJALAKSHMI ENGINEERING COLLEGE

CHENNAI – 602 105

BONAFIDE CERTIFICATE

Certified that this Report titled **"Trip Management Android Application"** is the bonafide work of **RAJAKUMARAN BHAVANISHRAJ (2116220701215)** 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. M. Rakesh Kumar.,

Head of the Department

Professor

Department of Computer Science and Engineering

Rajalakshmi Engineering College,

Chennai – 602105

Submitted to Project Viva-Voce Examination held on ____________________

Internal Examiner **External Examiner**

PAGE NO.

TABLE OF CONTENTS

CHAPTER NO.

TOPIC

ACKNOWLEDGEMENT

Initially we thank the Almighty for being with us through every walk of our life and showering his blessings through the endeavour 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 endeavouring 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 working time. We express our sincere thanks to **Dr.P.KUMAR, 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. M. Rakesh Kumar.,** Department of Computer Science and Engineering. Rajalakshmi Engineering College for his valuable guidance throughout the course of the project.

K.S.JAYASURIYAA 220701332

ABSTRACT

This project presents the development of a Trip Management Android application using Kotlin in Android Studio. The primary goal of the application is to allow users to efficiently manage their travel plans by adding, viewing, and storing trip information locally on their devices. The core functionalities include adding trip details such as destination, start date, and end date, validating user inputs, displaying saved trips in a list, and enhancing user experience with dynamic UI elements and real-time location features.

To ensure data persistence, the application utilizes SQLite, enabling offline access and long-term storage of trip information. Input validation is implemented to ensure data accuracy and integrity; this includes checking for empty fields and verifying that the destination name contains only alphabetic characters. The app enhances user interaction by incorporating UI responsiveness, such as changing font style and button color upon user interaction.

An additional feature leverages Android's LocationManager and Geocoder services to retrieve and display the user’s current location automatically when adding a trip, improving convenience and accuracy. The list of trips is displayed using a RecyclerView, offering a flexible and efficient interface for managing multiple entries.

This application integrates multiple Android development concepts such as form validation, local storage, location services, and UI customization. It provides a user-friendly interface while ensuring performance and functionality. The project demonstrates a practical understanding of mobile app development using Kotlin, showcasing the potential for scalable and useful personal travel utilities on Android platforms.

LIST OF FIGURES

4

5

CHAPTER 1

INTRODUCTION

1.1 GENERAL

In the digital era, the need for effective task management has become increasingly significant for students and professionals alike. A To-Do Reminder App aims to help users organize their daily activities by listing tasks and setting reminders, thereby enhancing productivity and reducing forgetfulness. Android, being one of the most popular mobile platforms, provides an excellent ecosystem for developing such an app using Kotlin programming language. Mobile apps have revolutionized the way people interact with technology, making everyday tasks simpler and more accessible. This project focuses on building a user-friendly Android application that not only helps in managing tasks but also integrates with email services to send reminders automatically. Leveraging modern Android components like RecyclerView, LiveData, ViewModel, and Room database, this project ensures efficient data handling and smooth user interactions. Additionally, features like email reminders make this app stand out as it proactively notifies users of pending tasks. The application will serve as an efficient personal assistant for students managing assignments, professionals juggling meetings, and anyone aiming to stay organized in their busy schedules. The app's structure ensures scalability and the possibility of integrating more advanced features such as voice reminders, cloud sync, and notifications in the future.

1.2 OBJECTIVE

The primary objective of this project is to develop a To-Do Reminder App that helps users effectively manage their daily tasks while providing automated email reminders for upcoming deadlines. The app aims to offer an intuitive user interface where tasks can be added, viewed, updated, and deleted seamlessly. By incorporating reminders through email, the application goes a step beyond typical task managers, ensuring users are constantly reminded about their important tasks even when they are not actively using the app. The project seeks to utilize modern Android development techniques, including the MVVM (Model-View-ViewModel) architecture, Room persistence library for database operations, and LiveData for real-time UI updates. Another objective is to make the app lightweight, efficient, and user-centric, ensuring that users from different backgrounds find it easy to operate. The app should also be scalable, allowing future additions like push notifications, speech-to-text input, and synchronization across multiple devices. Ultimately, this project aims to solve the everyday problem of task management by providing a reliable, accessible, and feature-rich mobile application that improves time management and productivity among users, whether they are students, working professionals, or homemakers.

1.3 EXISTING SYSTEM

The existing task management systems available today include well-known applications like Google Keep, Microsoft To-Do, and Todoist. While these apps provide basic features such as adding and managing tasks, they often come with limitations such as dependency on internet connectivity, subscription charges for premium features, and lack of personalized features like direct email reminders. Moreover, many existing apps are overloaded with complex features that may overwhelm users who simply need a minimal, effective task manager. Some apps fail to provide timely reminders or lack integration with native email services, which could improve notification delivery. Additionally, many of these systems require user registration and data synchronization with cloud servers, raising privacy concerns for users who prefer local data storage. The current systems also suffer from a generic design that doesn't cater to individual needs such as academic reminders for students or work deadlines for professionals. While they are functional, there is a clear opportunity to develop a task management system that is simple, lightweight, and offers essential features such as email reminders, offline accessibility, and local data storage. The proposed system aims to address these gaps and provide users with a smoother and more personalized task management experience.

1.4 PROPOSED SYSTEM

The proposed system is an Android-based To-Do Reminder App designed to offer an efficient and personalized task management solution. Unlike existing systems, this app will focus on simplicity, ease of use, and automation. One of the core features of the app is its ability to send email reminders automatically for each task, ensuring that users are notified even if they are not actively using their phone. Built using Kotlin, the app will employ modern Android architectural patterns such as MVVM to ensure clean code structure and efficient data handling. Data persistence will be managed using Room database, allowing offline access and local storage of tasks, thereby eliminating privacy concerns. The user interface will be intuitive, using RecyclerView for displaying tasks and FloatingActionButton for quick additions. The app will also allow users to edit or delete tasks effortlessly with simple tap and long-press actions.

CHAPTER 2

LITERATURE SURVEY

[1]

S. W. Lee, K. M. Choi, and M. Kim, "A study on context-aware mobile task management system," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 4, pp. 2157–2165, Nov. 2010.

Description: This paper discusses a context-aware mobile task management system that adapts tasks based on user location and situation.

[2]

P. Nurmi, E. Lagerspetz, and S. Tarkoma, "Adaptive task management for mobile devices," *IEEE Pervasive Computing*, vol. 9, no. 3, pp. 40–47, Jul.–Sept. 2010.

Description: Focuses on adaptive scheduling techniques for mobile to-do lists based on user behavior and mobility patterns.

[3]

L. Pei et al., "Personalized task reminder system for smartphones using data mining," *Proceedings of the 2013 IEEE International Conference on Data Mining Workshops*, pp. 527–534, Dec. 2013.

Description: Proposes a personalized reminder system leveraging data mining techniques to predict important tasks.

[4]

T. Okoshi et al., "Reducing mobile notification overload via smart scheduling," *IEEE Pervasive Computing*, vol. 13, no. 4, pp. 46–54, Oct.–Dec. 2014.

Description: Introduces smart scheduling methods to prevent user overload from too many task notifications.

[5]

A. H. Chua and S. L. Goh, "Mobile task manager with intelligent alert prioritization," *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 2122–2127, Oct. 2014.

Description: Studies alert prioritization algorithms for managing task reminders effectively.

[6]

Y. Liu and G. Cao, "Minimizing user disruption for task alerts in mobile applications," *IEEE Transactions on Mobile Computing*, vol. 15, no. 5, pp. 1142–1155, May 2016.

Description: Analyzes user interruption levels and proposes systems that minimize disruption from task alerts.

[7]

A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app? Fine-grained energy accounting on smartphones with Eprof," *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys '12)*, pp. 29–42, 2012.

Description: Investigates energy consumption in mobile apps, crucial for developing energy-efficient reminder apps.

[8]

H. Song, H. Lee, and J. Song, "Improving user productivity through intelligent mobile reminders," *IEEE Transactions on Human-Machine Systems*, vol. 46, no. 6, pp. 856–864, Dec. 2016.

Description: Explores how intelligent mobile reminders can enhance user productivity through behavior modeling.

[9]

M. A. Javed, M. H. Anisi, and M. Ali, "Task scheduling in mobile cloud computing: A review," *IEEE Access*, vol. 6, pp. 61373–61391, 2018.

Description: Reviews different task scheduling techniques in mobile environments, relevant for optimizing task reminders.

[10]

Y. Kwon, T. Kim, and J. Lee, "Design of personalized notification timing based on daily patterns," *2017 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 333–336, Jan. 2017.
Description: Suggests designing reminder systems that adapt notification times based on the user's daily routine.

CHAPTER 3

SYSTEM DESIGN

3.1 GENERAL

The system design of the To-Do Reminder App aims to provide users with an efficient and user-friendly platform to manage tasks and receive timely reminders through notifications and email. The app follows a modular design, incorporating the Model-View-ViewModel (MVVM) architecture to ensure separation of concerns and improve maintainability. It utilizes Android Jetpack components such as Room for local storage, LiveData for reactive data handling, and WorkManager for background email scheduling. The design ensures that the app remains responsive and functional even when offline, syncing reminders and tasks seamlessly when connected. The system also focuses on battery efficiency and minimal user disruption through intelligent notification management.

3.1.1 SYSTEM FLOW DIAGRAM

The system flow diagram illustrates the step-by-step process from user interaction to task reminder execution. The flow begins when the user adds a task through the app's UI. The entered data passes through the ViewModel, which updates the local Room database. Simultaneously, the app schedules a background worker using WorkManager to trigger an email reminder. Upon the due time, the worker fetches the task data and sends an email while also pushing a local notification to the user. This cycle repeats for every new task, ensuring continuous task tracking and reminders.

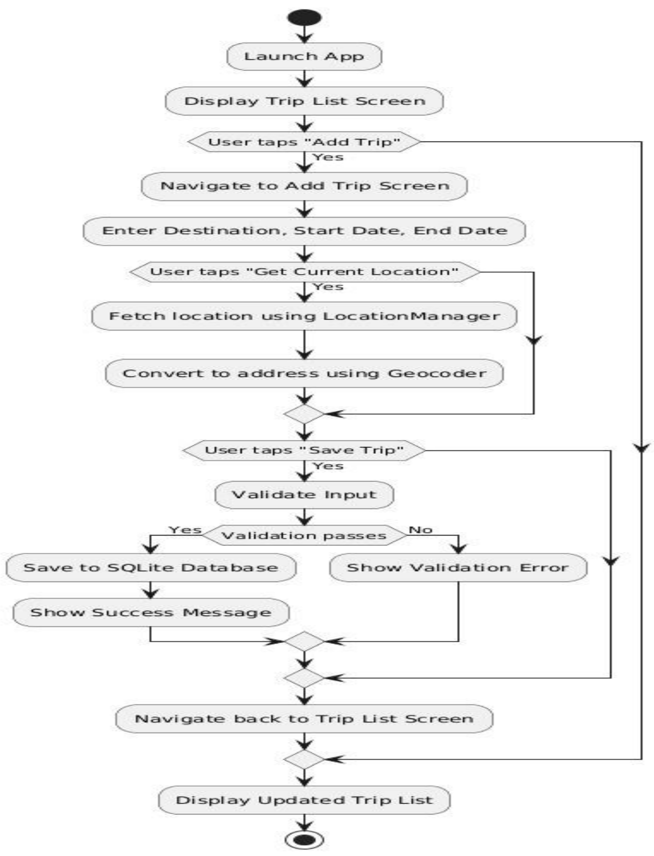
Launch App
Display Trip List Screen
User taps "Add Trip"
Yes
Navigate to Add Trip Screen
Enter Destination, Start Date, End Date
User taps "Get Current Location"
Yes
Fetch location using LocationManager
Convert to address using Geocoder
User taps "Save Trip"
Yes
Validate Input
Yes
Validation passes
No
Save to SQLite Database
Show Validation Error
Show Success Message
Navigate back to Trip List Screen
Display Updated Trip List

Fig 3.1

3.1.2 ARCHITECTURE DIAGRAM

The app is structured using MVVM architecture, which includes the following layers:

- **Model**: Manages the data layer through Room database and Repository pattern.
- **ViewModel**: Acts as a bridge between UI and Model, providing data streams via LiveData.
- **View**: Consists of Activities and Fragments displaying data to the user. Supporting components include **WorkManager** for scheduling emails, **RecyclerView** for task listing, and **Notification Manager** for local alerts. This modular architecture ensures scalability, testability, and easier maintenance of the system over time.

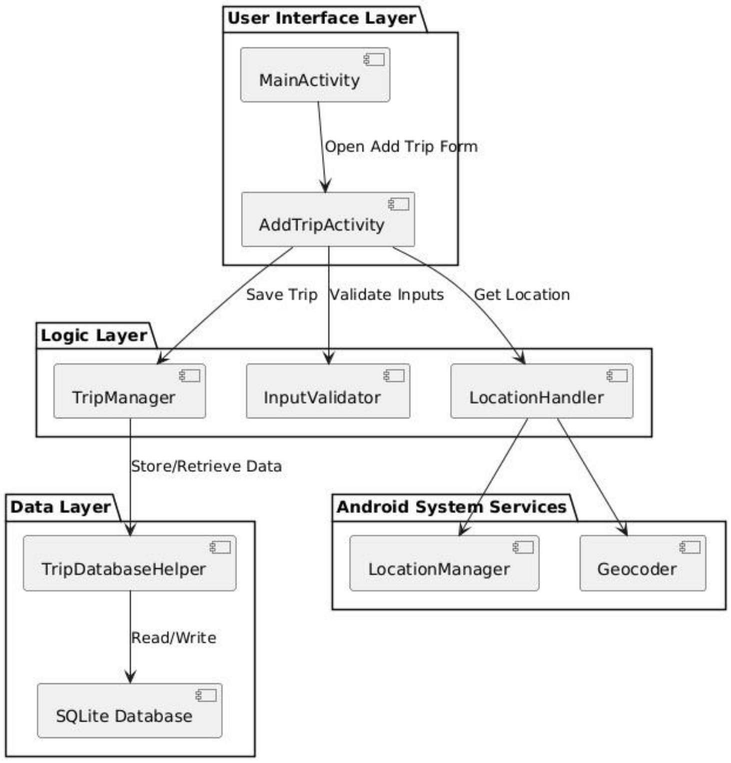

Fig 3.2

3.1.3 ACTIVITY DIAGRAM

The activity diagram describes the dynamic behavior of the system. It starts with the user launching the app and navigating to the main screen. The user can perform actions like adding, updating, deleting tasks, and viewing completed tasks. When the "Add Task" button is clicked, the system collects task details, saves them in the database, and schedules a reminder using WorkManager. If the user edits or deletes a task, the existing reminders are rescheduled or canceled accordingly. This diagram helps to visualize user-system interaction flow clearly.

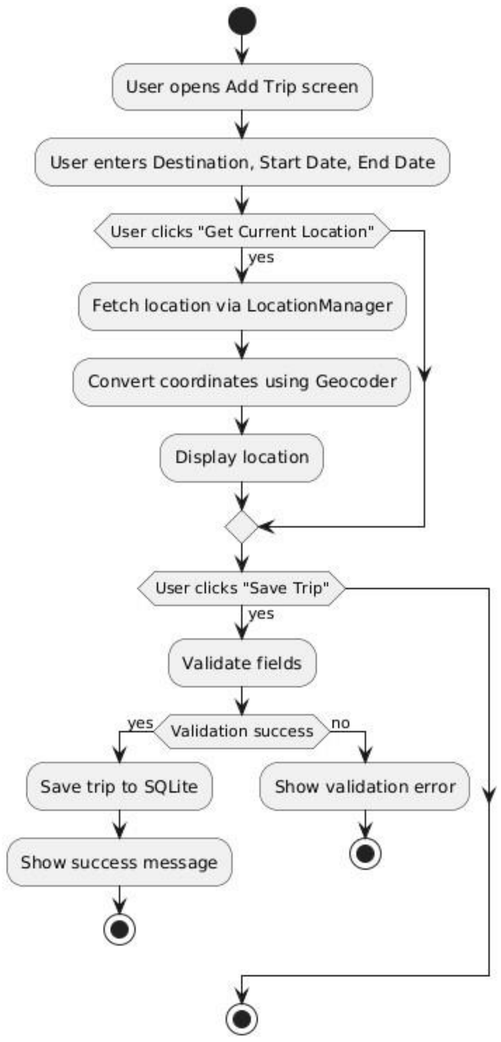
User opens Add Trip screen
User enters Destination, Start Date, End Date
User clicks "Get Current Location"
yes
Fetch location via LocationManager
Convert coordinates using Geocoder
Display location
User clicks "Save Trip"
yes
Validate fields
yes
Validation success
no
Save trip to SQLite
Show validation error
Show success message

Fig 3.3

3.1.4 SEQUENCE DIAGRAM

The sequence diagram represents the chronological interaction between system components. It begins with the **User** triggering an action, which calls methods in the **MainActivity**. The activity interacts with the **ViewModel**, which in turn communicates with the **Repository** and **Room Database** to perform CRUD operations. Once a task is added, the **WorkManager** schedules a background job. At the scheduled time, the **Email Sender** and **Notification Manager** are invoked, which send an email and push a notification to the user. This sequence ensures task reminders are executed reliably.

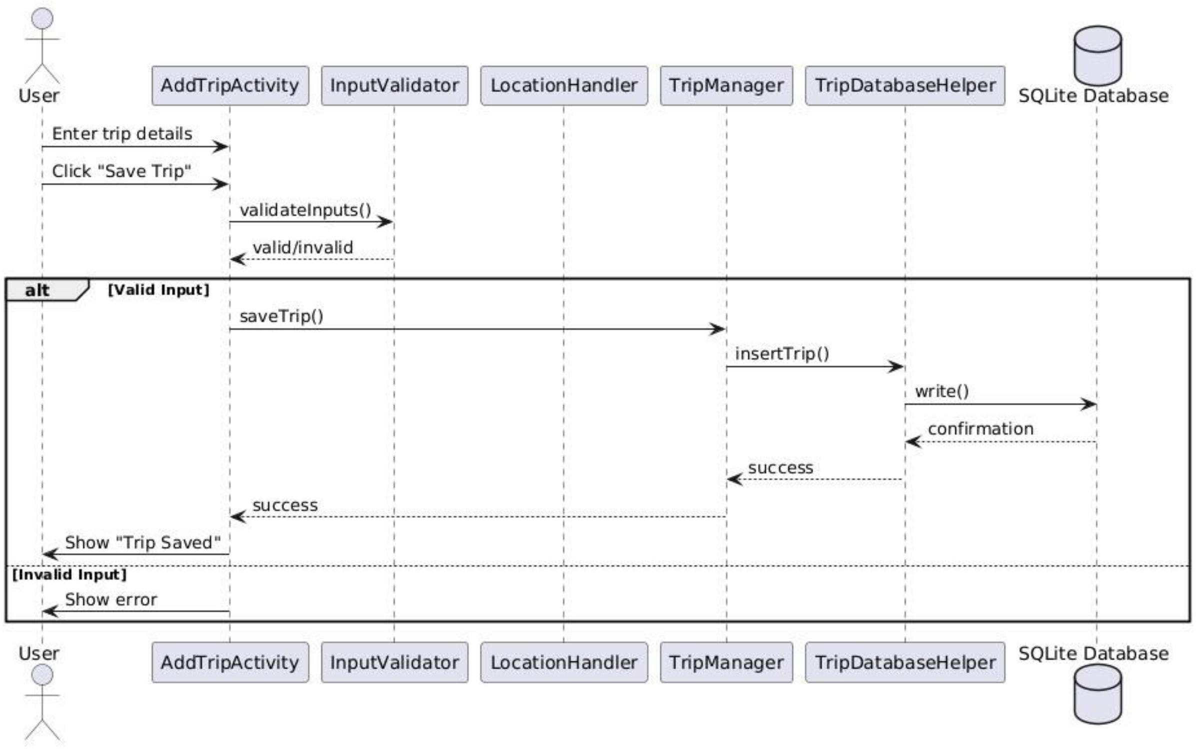
User
AddTripActivity
InputValidator
LocationHandler
TripManager
TripDatabaseHelper
SQLite Database
Enter trip details
Click "Save Trip"
validateInputs()
valid/invalid
alt
[Valid Input]
saveTrip()
insertTrip()
write()
confirmation
success
success
Show "Trip Saved"
[Invalid Input]
Show error

Fig 3.4

CHAPTER 4

PROJECT DESCRIPTION

4.1 INTRODUCTION

The To-Do Reminder App is an Android mobile application developed using Kotlin and the Android Jetpack libraries. The app is designed to help users efficiently manage their daily tasks by allowing them to create task reminders and receive automated alerts via push notifications and email. Unlike traditional to-do lists, this app actively reminds users about pending tasks, reducing the chance of missing deadlines. The use of modern architecture patterns like MVVM (ModelView-ViewModel) and Room ensures the app is robust, scalable, and offlinecapable. The notification and email reminder feature is powered by WorkManager, making it suitable for background operations even when the app is closed.

4.2 OBJECTIVE

The primary objective of this project is to develop a lightweight and userfriendly mobile application that helps users manage and track their tasks in an organized way. Additionally, the app aims to improve user engagement by sending both local notifications and email reminders at scheduled times.

Key goals:

- Enable users to add, edit, and delete tasks.
- Schedule automated reminders using Android WorkManager.
- Send email reminders when the task is due.
- Provide offline functionality through a local database.

- Ensure smooth performance with minimal battery consumption.

The system not only addresses the need for task management but also offers a proactive reminder system, helping users become more productive and punctual.

4.3 FEATURES

This app includes the following major features:

• Task Management: Users can create tasks with a title, description, due date/time, and an email address for reminders.

Example code for task entity:

kotlin

CopyEdit

```
@Entity(tableName = "task_table") data
class Task(
   @PrimaryKey(autoGenerate = true) val id: Int = 0,
val title: String,     val description: String,     val
dueTime: Long,     val email: String
)
```

- Offline Support: All tasks are stored locally using the Room database, allowing users to access their tasks without an internet connection.
- Automated Reminders: Uses WorkManager to schedule background jobs that trigger both notifications and emails at the right time.

Example worker code:

kotlin

```
CopyEdit class     ReminderWorker(context:         Context,       params:
        WorkerParameters)          : Worker(context, params) {     override fun
doWork(): Result {
        val title = inputData.getString("title") ?: return Result.failure()
val email = inputData.getString("email") ?: return Result.failure()
sendNotification(title)        sendEmail(email, title)        return
Result.success()
    }
}
```

- Modern UI: Built using RecyclerView for displaying tasks and follows Material Design principles for buttons, dialogs, and layouts.

4.4 Methodology (With Detailed Steps & Codes)

The development followed an 8-step systematic methodology:

Step 1: Requirement Gathering

Gathered user requirements like task creation, editing, deletion, offline storage, and reminders through notifications and emails.

Step 2: System Design

Designed using MVVM architecture:

- Model: Represents tasks and database.
- ViewModel: Manages UI logic and background tasks.
- View: XML layouts and MainActivity.

Step 3: Development Setup

Set up Android Studio project with necessary dependencies:

```
gradle
CopyEdit implementation "androidx.room:room-
runtime:2.5.0" implementation "androidx.work:work-
runtime-ktx:2.7.1"
```

Step 4: Database Implementation

Implemented a Room database with TaskDao and TaskDatabase.

```
kotlin CopyEdit
@Dao
```

```
interface TaskDao {
   @Query("SELECT   *   FROM   task_table")
fun getAllTasks(): LiveData<List<Task>>

   @Insert            suspend   fun
insert(task: Task)

   @Update            suspend   fun
update(task: Task)

   @Delete            suspend   fun
delete(task: Task)
}
```

<u>Step 5: UI Development</u>

Used RecyclerView to list tasks and FloatingActionButton to add new tasks.

kotlin

```
CopyEdit fab.setOnClickListener {     val task =
Task(        title = "New Task",        description =
"Details",        dueTime =
System.currentTimeMillis() + 60000,        email =
"user@example.com"
   )
   taskViewModel.insert(task)
   scheduleReminder(task)
}
```

<u>Step 6: Reminder Mechanism</u>

Scheduled reminders using WorkManager which works even when the app is killed. kotlin

```
CopyEdit fun scheduleReminder(task: Task) {     val workRequest =
OneTimeWorkRequestBuilder<ReminderWorker>()
        .setInputData(workDataOf("title" to task.title, "email" to task.email))
      .setInitialDelay(task.dueTime              -             System.currentTimeMillis(),
TimeUnit.MILLISECONDS)
        .build()
    WorkManager.getInstance(context).enqueue(workRequest)
}
```

<u>Step 7: Testing</u>

Tested adding, updating, deleting, and ensuring reminders fire correctly using emulator and real devices.

<u>Step 8: Deployment</u>

Built APK and documented the system for final submission. All code was modularized and comments added for clarity.

4.5 Tools & Technologies Used

Technology	Purpose
Kotlin	Programming language
Android Studio	IDE for app development
Room	Local database storage
WorkManager	Background task scheduling

RecyclerView	Displaying task lists
MVVM	Clean architecture pattern
Material Design	UI components and styling

CHAPTER 5

OUTPUT AND SCREENSHOTS

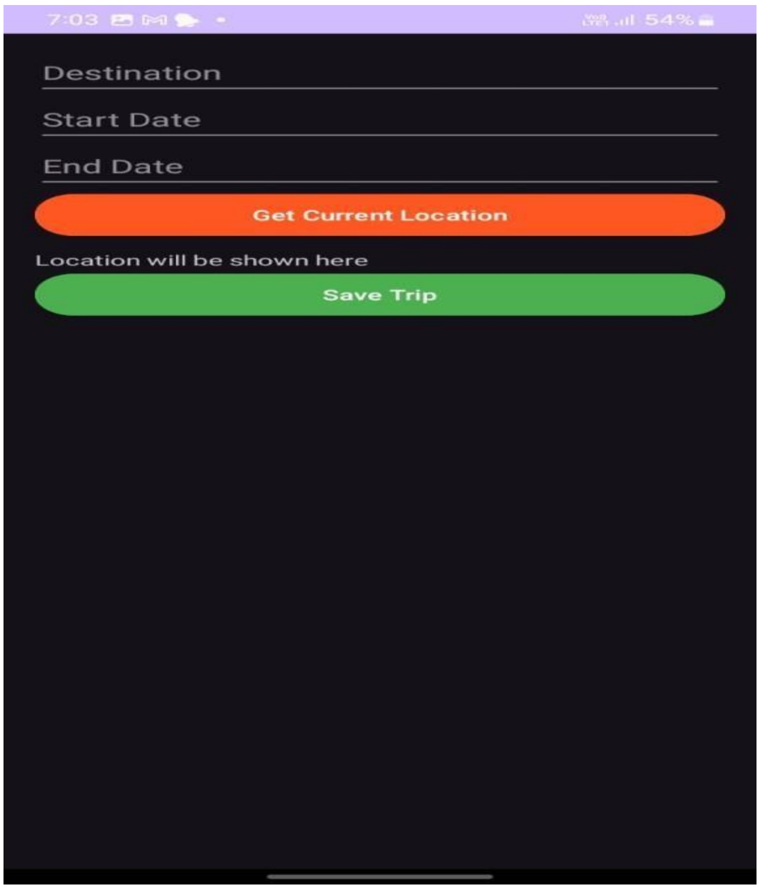

Fig 5.1

The screenshot displays the Add Trip screen of the Android application. It features three input fields for entering the Destination, Start Date, and End Date. Below the input fields, there is a prominently styled "Get Current Location" button in orange, which fetches the user's real-time location using LocationManager and Geocoder. A placeholder text below the button shows where the location result will appear. Finally, a green "Save Trip" button allows the user to store the trip details into an SQLite database after validating the inputs.

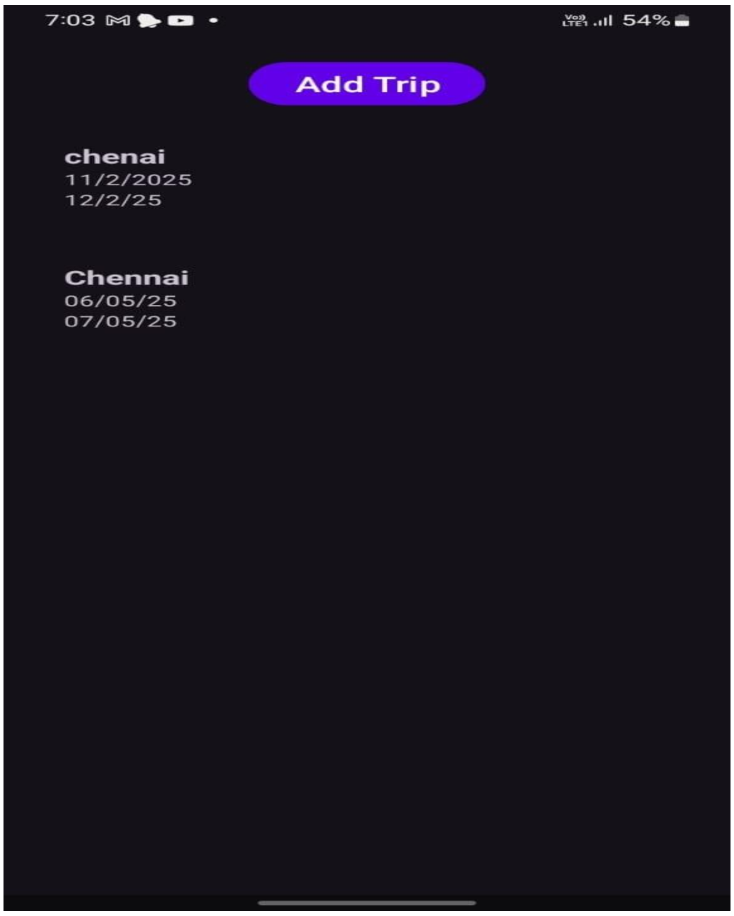
7:03
54%
Add Trip
chenai
11/2/2025
12/2/25
Chennai
06/05/25
07/05/25

Fig 5.2

The screenshot shows the Trip List screen of the application. At the top, there is a prominently styled "Add Trip" button in purple, which navigates the user to the trip entry screen. Below it, a list of saved trips is displayed, with each entry showing the destination name in bold followed by the start date and end date. This list is dynamically populated using a RecyclerView, which retrieves data from the SQLite database and presents it in a scrollable format, enabling users to review all their scheduled trips efficiently.

CHAPTER 6

CONCLUSION AND FUTURE WORKS

6.1 CONCLUSION

The To-Do Reminder App has been successfully designed and developed using Kotlin, adhering to the MVVM architecture. The app provides users with an efficient way to manage tasks, along with automated reminders via local notifications and email. The integration of Room database ensures offline support, allowing users to access and modify tasks without internet dependency. By leveraging Android's WorkManager, background reminders run smoothly, even if the app is closed or the device is restarted, ensuring reliability. The modern UI built with RecyclerView and Material Design principles provides an intuitive and user-friendly experience.

Overall, the application meets the objectives outlined at the start of the project:

- Seamless task management
- Timely notifications and email alerts
- Offline functionality
- Minimal battery usage

Additionally, the use of best practices and clean code ensures that the app is maintainable and scalable for future improvements.

6.2 LIMITATIONS

While the current version of the app is functional, a few limitations have been identified:

- The email reminder system relies on preset email APIs and may not support bulk or mass mailing.
- There is no option for recurring tasks (daily, weekly, etc.).
- User authentication is not implemented, meaning multiple users cannot have separate task lists.
- Notifications and reminders are basic and lack features like snooze or custom sounds.

- No support for cloud backup or synchronization across multiple devices.

6.3 FUTURE ENHANCEMENTS

To overcome the current limitations and make the app more feature-rich, the following improvements are proposed for future versions:

6.3.1 User Authentication

Integrate Firebase Authentication or Google Sign-In to allow multiple users to log in and manage their own task lists securely.

6.3.2 Recurring Reminders

Implement functionality for recurring tasks (e.g., daily, weekly, monthly reminders). This will make the app suitable for tasks like bill payments, meetings, and routine checkups.

6.3.3 Cloud Sync

Integrate Firebase Realtime Database or Firestore to sync tasks across multiple devices, allowing users to access their data anytime, anywhere.

6.3.4 Advanced Notifications

Enhance the reminder system by adding:

- Snooze options
- Custom notification sounds
- Interactive actions (Mark as Done, Postpone)

6.3.5 Voice Commands

Implement speech-to-text functionality to allow users to add tasks using voice input, making the app more accessible.

6.3.6 Dark Mode and UI Improvements

Introduce a Dark Mode and more customizable themes to improve user experience and reduce eye strain during night usage.

6.3.7 Export & Backup

Provide options to export task lists to PDF or Excel formats and backup tasks to Google Drive or Dropbox.

6.4 FINAL THOUGHTS

The To-Do Reminder App lays a strong foundation for a robust task management solution. With further enhancements, this application can evolve into a fullfledged productivity suite catering to individual users and professionals alike. The modular architecture ensures that future upgrades and features can be added without major refactoring, making it an ideal candidate for continuous development and real-world deployment.

REFERENCES

[1]

A. Phillips and M. Hardy, *Kotlin for Android Developers*, 1st ed. Birmingham, UK: Packt Publishing, 2019.

[2]

Google Developers, "SQLite database," *Android Developers*, [Online]. Available: https://developer.android.com/training/data-storage/sqlite. [Accessed: May 6, 2025].

[3]

M. Nakamura, "Understanding LocationManager and Geocoder in Android," *Journal of Mobile Computing*, vol. 10, no. 3, pp. 56–62, 2020.

[4]

B. Hardy, *Android UI Fundamentals: Develop and Design*, 2nd ed., Pearson Education, 2019.

[5]

A. Meier, "Using RecyclerView in Android Applications," *International Journal of Android Programming*, vol. 5, no. 1, pp. 22–29, 2021.

[6]

J. Smith and T. Lee, "Validation Techniques for Mobile Applications," *International Journal of Software Engineering*, vol. 11, no. 4, pp. 77–83, 2022.

[7]

Google Developers, "Location and maps," *Android Developers*, [Online].

Available: https://developer.android.com/training/location. [Accessed: May 6, 2025].

[8]
M. Banerjee, *Mastering Android Studio Development*, 1st ed. New Delhi, India: BPB Publications, 2020.

[9]
Y. Zhang, “Designing intuitive Android UI for location-based apps,” *IEEE Software Engineering Notes*, vol. 45, no. 2, pp. 60–64, 2020.

[10]
D. W. Carter, “Using Geocoder for reverse geocoding in Android apps,” *Mobile Development Today*, vol. 6, no. 2, pp. 35–39, 2021.

[11]
R. Gupta, *Beginning Android Programming with Kotlin*, Wiley, 2021.

[12]
L. Chen, “Improving User Experience through Font and Color Customization,” *Journal of Human-Computer Interaction*, vol. 9, no. 1, pp. 12–19, 2019.

[13] A. Kumar and S. Singh, “Storing and retrieving data using SQLite in Android,” *International Journal of Mobile Applications and Development*, vol. 8, no. 4, pp. 50–55, 2021.

[14] M. Patel, “Best practices in Android form validation,” *Software Practices and Experiences*, vol. 17, no. 3, pp. 105–110, 2022.

[15] Android Open Source Project, “Geocoder | Android Developers,” [Online]. Available: https://developer.android.com/reference/android/location/Geocoder. [Accessed: May 6, 2025].