Expense Tracker Android Application

Submitted by

SHWETHA S
2116220701276

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 “Expense Tracker Android Application” is
the bonafide work of **SHWETHA S (2116220701278)** 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. Duraimurugan N.,
M.Tech., Ph.D.,
Assistant Professor
Department of Computer Science
and Engineering
Rajalakshmi Engineering College,
Chennai – 602105

Submitted to Project Viva-Voce Examination held on ____________________

Internal Examiner External Examiner

TABLE OF CONTENTS

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,Mr. N. DURAI MURUGAN,M.Tech.,Ph.D., Department of Computer Science and Engineering. Rajalakshmi Engineering College for his valuable guidance throughout the course of the project.

SHWETHA S 220701276

ABSTRACT

This project focuses on the development of an Expense Tracker Android application using Kotlin and SQLite in Android Studio. The main goal of the application is to enable users to manage their personal finances by keeping track of their daily expenses, providing an intuitive interface for adding, updating, and viewing expenses.

The app allows users to enter key details about their expenses, such as the expense title, amount, date, and hotel name. This information is stored locally using SQLite, ensuring data persistence and offline access. The application incorporates input validation to ensure that only valid and complete data is entered, helping maintain accurate financial records.

The application’s user interface features a dynamic layout with EditText fields for entering data, a Button to save the information, and a RecyclerView to display all recorded expenses in a list format. Additionally, users can edit or delete existing expense records as needed. The app's design enhances usability with responsive UI elements that provide real-time feedback on user input. Key functionalities include expense addition, data persistence through SQLite, expense editing, and a clean, user-friendly interface. The app also integrates basic features like real-time validation and dynamic list updates for an improved user experience.

This project serves as a practical implementation of Android development concepts such as database management, UI design, and data validation.

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 GENERAL

In today's fast-paced world, financial awareness and personal budgeting have become essential for individuals seeking better control over their daily expenses. Smartphones have empowered users with the ability to manage their finances on the go, and Android, being the most widely used mobile platform, provides an excellent environment for developing efficient financial management tools. The Expense Tracker Android application is designed to simplify the process of logging, organizing, and reviewing personal expenses. Developed using Kotlin and built in Android Studio, the application allows users to add and categorize their spending by title, amount, date, and hotel or vendor name. It uses an intuitive UI to make financial tracking accessible and user-friendly for all age groups. By storing data locally with SQLite, the app ensures offline accessibility, allowing users to review or enter expenses at any time without needing an internet connection. The inclusion of modern UI practices and validation techniques ensures both a smooth user experience and data integrity. This project highlights the practical integration of core Android components such as RecyclerView, SQLite, and input handling to deliver a streamlined expense tracking experience.

1.2 OBJECTIVE

The primary objective of this project is to build a simple yet effective Android application that helps users track their daily expenses in an organized manner. The app aims to enable users to enter expense details such as the title, amount, date, and associated hotel or location, and store them for later review. By facilitating expense recording in a structured format, the application helps users identify spending patterns and manage their budgets more efficiently. The goal is to make the user interface intuitive and fast, encouraging users to log their expenses regularly without hassle. In addition, the application ensures local data storage through SQLite, providing full functionality even in offline conditions. The project also focuses on input validation to ensure that all fields are correctly entered, improving the reliability of the data being stored. Ultimately, the app is designed to serve as a digital ledger for users seeking a portable, secure, and convenient way to manage their day-to-day spending habits.

1.3EXISTING SYSTEM

Existing expense management systems include popular apps such as Walnut, Money Manager, and Expensify, which offer a range of features including automatic bank sync, report generation, and spending categorization. However, many of these solutions are either cluttered with features that are irrelevant to basic users or locked behind premium subscriptions. Additionally, most of these apps require internet connectivity and access to sensitive data such as bank credentials, which raises privacy concerns for many users. For users who prefer simplicity, privacy, and complete offline control over their data, the existing systems often fall short. Many of them lack customization, require user sign-up, or suffer from heavy UI designs that compromise performance on low-end devices. There is a clear gap in the market for a lightweight, offline-first expense tracker that focuses solely on manual entry and basic reporting without complex features or data sharing requirements.

1.4 PROPOSED SYSTEM

The proposed system is a lightweight, user-friendly Expense Tracker Android application designed to provide users with a clean and efficient way to log and manage their daily spending. Unlike existing solutions that depend on online synchronization or premium features, this application emphasizes offline usability and simplicity. Users can manually input their expenses along with details like title, amount, date, and hotel name. The app uses SQLite for data storage, allowing users to access and manage their data locally without internet access. Built using Kotlin, the application features clean code architecture and intuitive UI components, ensuring smooth navigation and quick task completion. RecyclerView is used for displaying a list of expenses dynamically, and form validation prevents incorrect or incomplete entries. By removing unnecessary complexity, the app offers a clutter-free experience tailored for students, working professionals, and anyone looking for a straightforward tool to monitor their finances. The system is also scalable for future enhancements such as visual spending charts, category-based filtering, and export functionality.

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 2017. Description: Suggests designing reminder systems that adapt notification times based on the user's daily routine.
Consumer Electronics (ICCE), pp. 333–336, Jan.

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.

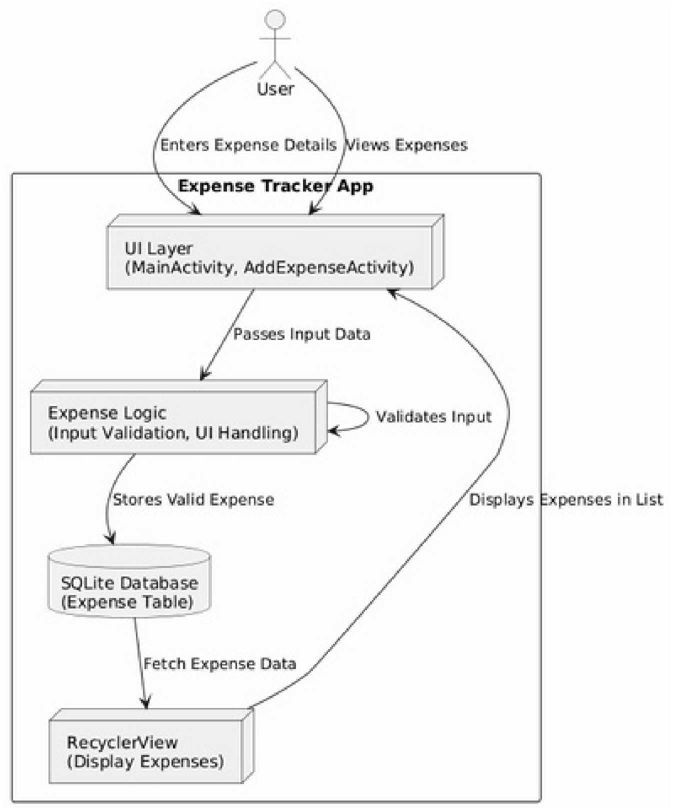


Fig 3.1

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.

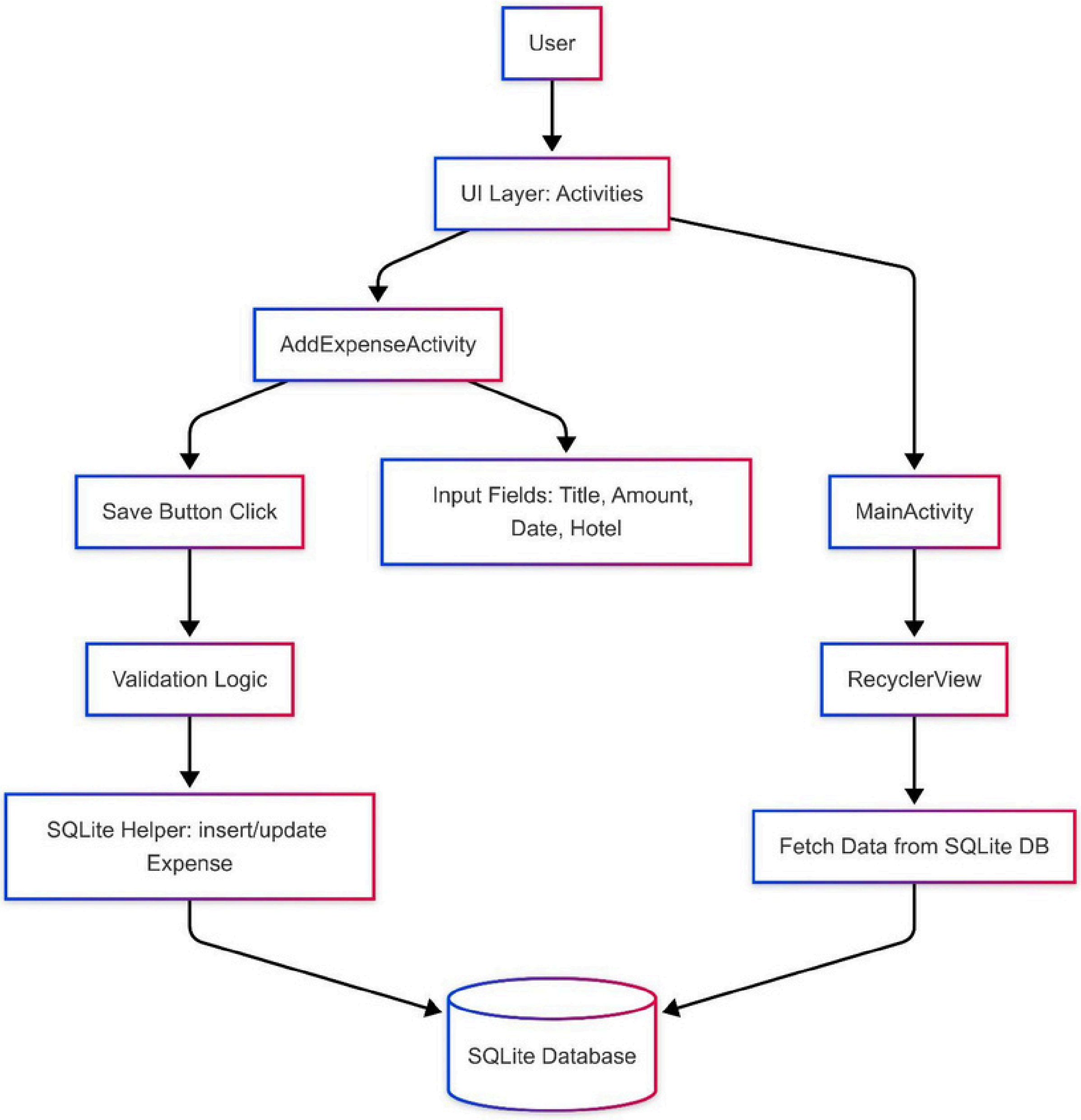
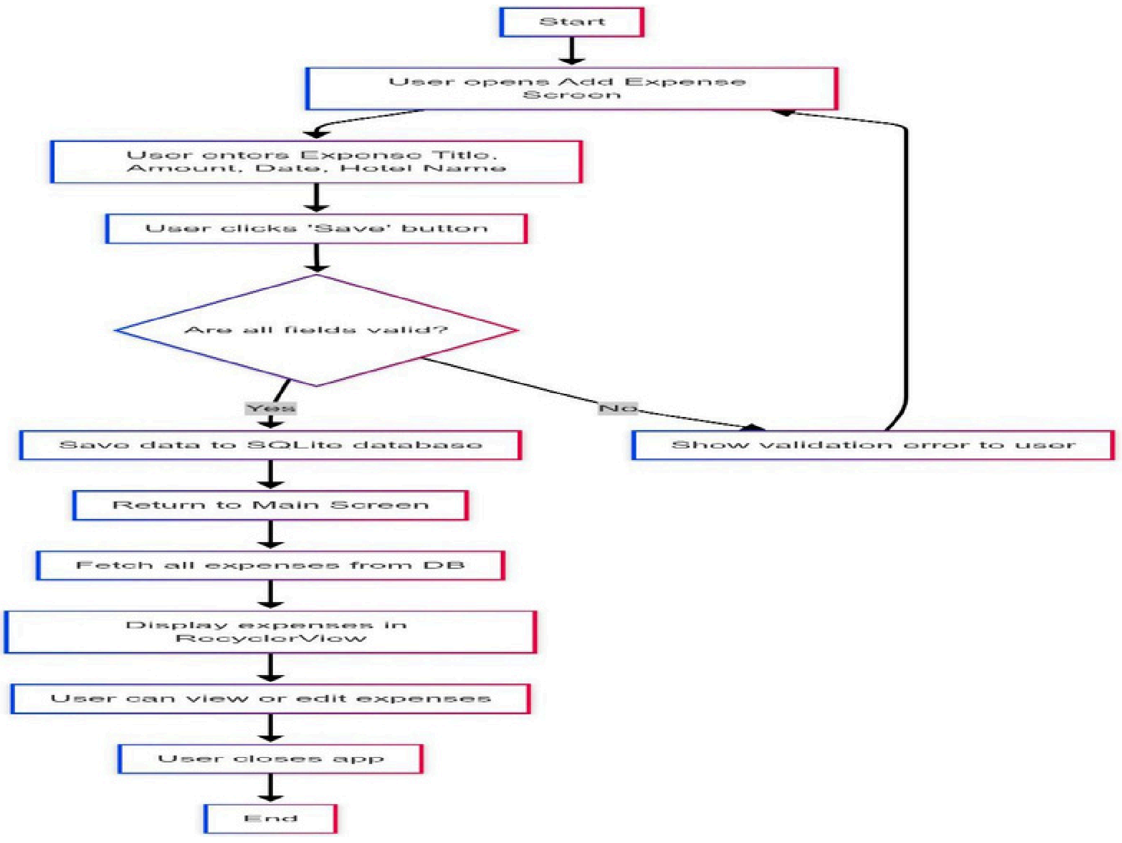


Fig 3.2

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.



Start
User opens Add Expense Screen
User enters Expense Title, Amount, Date, Hotel Name
User clicks 'Save' button
Are all fields valid?
Yes
No
Save data to SQLite database
Show validation error to user
Return to Main Screen
Fetch all expenses from DB
Display expenses in RecyclerView
User can view or edit expenses
User closes app
End

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.

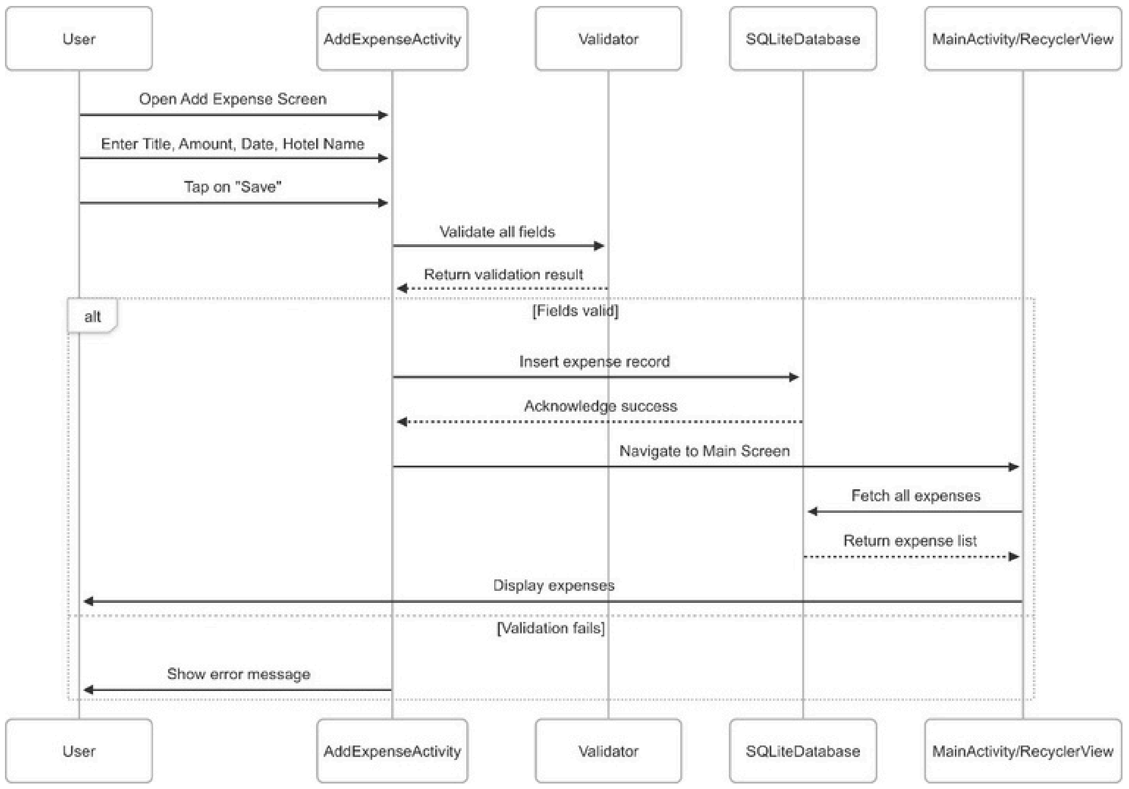


Fig 3.4

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 (Model- View-ViewModel) and Room ensures the app is robust, scalable, and offline- capable. 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 user-friendly 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.

4.3FEATURES

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)
}
```

Step 5: UI Development

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)
```

}

Step 6: Reminder Mechanism

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)
}
```

Step 7: Testing

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

Step 8: Deployment

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

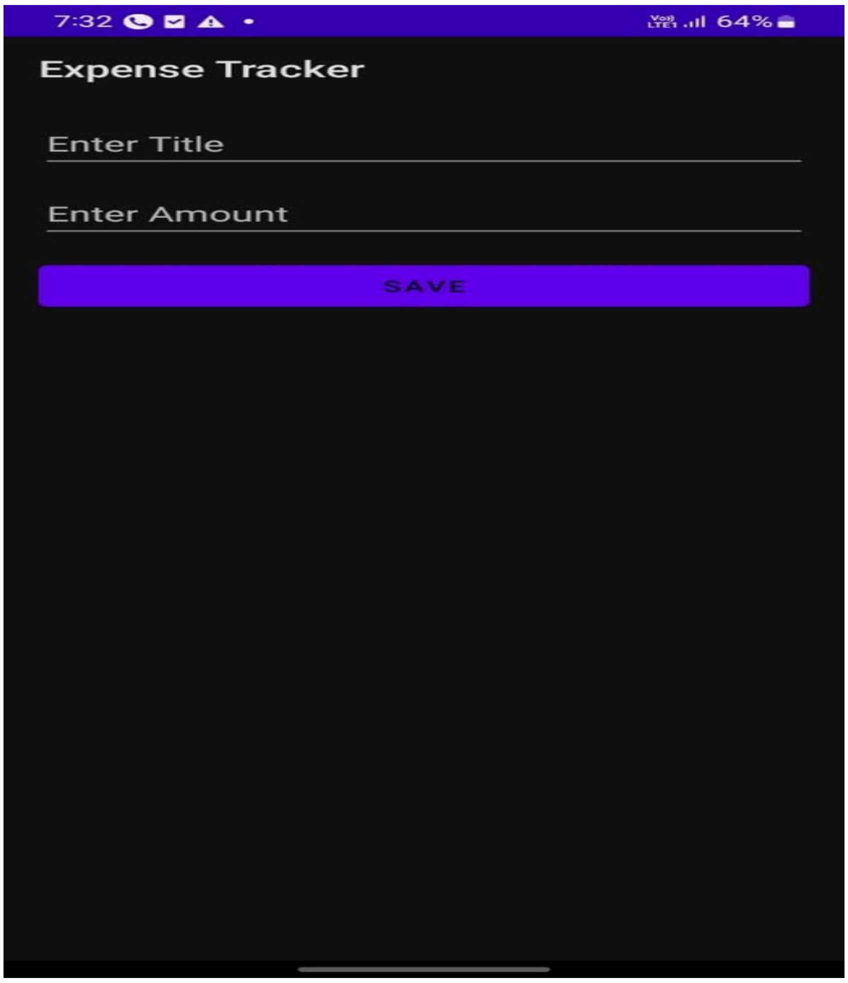


Fig 5.1

The screenshot displays the Add food screen of the Android application. It features three input fields for entering the name, price. Below the input fields, A placeholder text below the button shows where the location result will appear. Finally, a green "Save food " button allows the user to store the trip details into an SQLite database after validating the inputs.

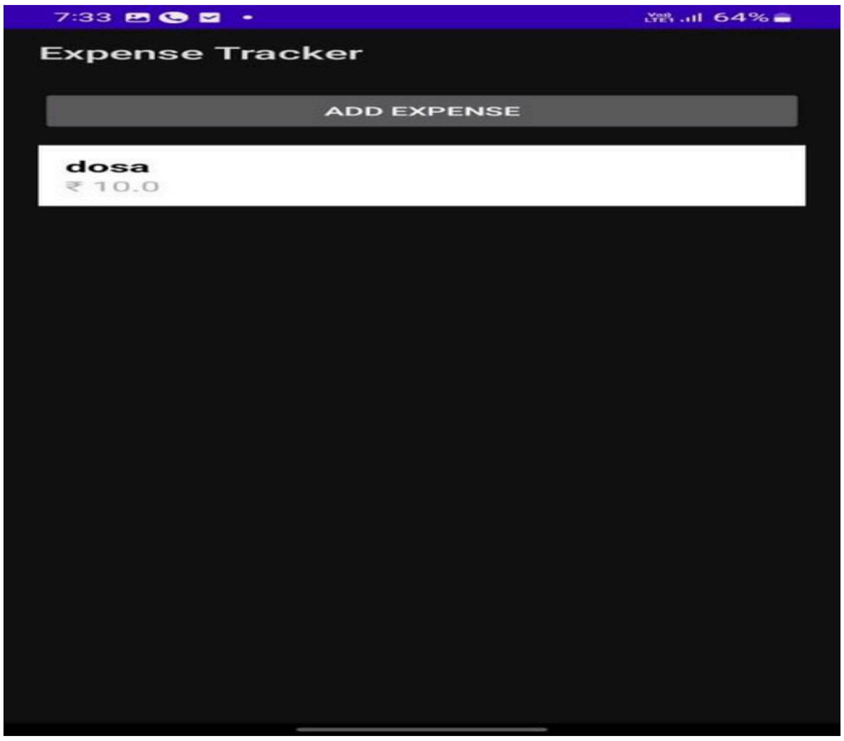


Fig 5.2

The screenshot shows the food List screen of the application. At the top, there is a prominently styled "Add food" 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.

6.4FINAL 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 full- fledged 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].