

# **MULTIFUNCTIONAL FINANCE TRACKING SYSTEM**

## **A PROJECT REPORT**

**Submitted by**

**MOHANAPRIYA K**  
(Reg. No: 24MCR068)

**SATHIYA JOTHI C**  
(Reg. No: 24MCR094)

**SUJITHA K C**  
(Reg. No: 24MCR111)

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

*of*

**MASTER OF COMPUTER APPLICATIONS**

**DEPARTMENT OF COMPUTER APPLICATIONS**



**KONGU ENGINEERING COLLEGE**

**(Autonomous)**

**PERUNDURAI, ERODE – 638 060**

**DECEMBER 2024**

**DEPARTMENT OF COMPUTER APPLICATIONS  
KONGU ENGINEERING COLLEGE**

**(Autonomous)**

**PERUNDURAI, ERODE – 638 060**

**DECEMBER 2024**

**BONAFIDE CERTIFICATE**

This is to certify that the project report entitled “**MULTI FUNCTIONAL FINANCE TRACKING SYSTEM**” is the bonafide record of project work done by **MOHANAPRIYA K (Reg. No: 24MCR068), SATHIYA JOTHI C (Reg. No: 24MCR094), SUJITHA K C (Reg. No: 24MCR111)** in partial fulfilment of the requirements for the award of the Degree of Master of Computer Applications of Anna University, Chennai during the year 2024-2025.

**SUPERVISOR**

**HEAD OF THE DEPARTMENT**

**(Signature with seal)**

**Date:**

Submitted for the end semester viva voce examination held on \_\_\_\_\_

**INTERNAL EXAMINER**

**EXTERNAL EXAMINER**

## DECLARATION

We affirm that the project report entitled “**MULTI FUNCTIONAL FINANCE TRACKING SYSTEM**” being submitted in partial fulfilment of the requirements for the award of Master of Computer Applications is the original work carried out by us. It has not formed the part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidates.

**Date:**

**MOHANAPRIYA K**  
**(Reg. No: 24MCR068)**

**SATHIYA JOTHI C**  
**(Reg. No: 24MCR094)**

**SUJITHA K C**  
**(Reg. No: 24MCR111)**

I certify that the declaration made by the above candidates is true to the best of my knowledge.

**Date:**

Name and Signature of the supervisor

**(Ms.M.SINDHU)**

## **ABSTRACT**

Managing personal and group finances can be difficult in today's busy world. People often face problems like creating budgets, tracking expenses, and remembering to pay bills on time. Groups, such as families, roommates, or friends, struggle with splitting expenses, keeping track of who paid what, and resolving disputes over money. Since there is no single solution to manage both personal and group finances, people end up using different tools, which leads to confusion, wasted time, and incomplete information. Most existing applications either focus only on personal finances or only on group expenses, leaving a big gap for an integrated solution.

The Multi-Functional Finance Tracking System solves these issues by combining tools for both personal and group financial management in one application. It helps individuals track their expenses, understand spending habits, and receive reminders for bills and subscriptions. For groups, the application provides features like automatic bill splitting, real-time updates, clear contribution tracking, and tools for collaborative budgeting. These features make it easier for users to manage their finances transparently and efficiently, whether they are working alone or in a group.

The application is built using Flutter, which ensures a smooth experience on both Android and iOS devices, saving time and cost during development while providing high performance. Firebase is used for the backend, offering secure login, real-time data updates, and reliable cloud storage. Unlike other applications, this system brings personal and group finance tools into one platform, eliminating the need for multiple applications., strong data security, and a user-friendly interface. This application stands out by simplifying financial management, encouraging good financial habits, promoting collaboration, and giving users a complete view of their financial situation.

## ACKNOWLEDGEMENT

We express our sincere thanks to our correspondent **Thiru.A.K.ILANGO BCom., MBA., LLB.,** and other philanthropic trust members of Kongu Vellalar Institute of Technology Trust for having provided with necessary resources to complete this project. We are always grateful to our beloved visionary Principal **Dr.V.BALUSAMY BE(Hons)., MTech., PhD** Kongu Engineering College, Perundurai for providing us with the facilities offered.

We convey our gratitude and heartfelt thanks to our Head of the Department **Dr.A.TAMILARASI Msc., MPhil., PhD., MTech.,** Department of Computer Applications, Kongu Engineering College for her perfect guidance and support that made this work to be completed successfully.

We also like to express our gratitude and sincere thanks to our project coordinator **Mrs.S.HEMALATHA MCA.,** Assistant Professor (Sr.G), Department of Computer Applications, Kongu Engineering college who have motivated us in all aspects for completing the project in scheduled time.

We would like to express our gratitude and sincere thanks to our project guide **Mrs.M.SINDHU MCA.,** Assistant Professor, Department of Computer Applications, Kongu Engineering College for giving her valuable guidance and suggestions which helped us in the successful completion of the project.

We owe a great deal of gratitude to our parents for helping us to overwhelm in all proceedings. We bow our heart and head with heartfelt thanks to all those who thought us their warm services to succeed and achieve our work.

## TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	<b>ABSTRACT</b>	iv
	<b>ACKNOWLEDGEMENT</b>	v
	<b>LIST OF FIGURES</b>	viii
	<b>LIST OF ABBREVIATIONS</b>	ix
<b>1</b>	<b>INTRODUCTION</b>	1
	1.1 ABOUT THE PROJECT	1
	1.2 EXISTING SYSTEM	2
	1.3 DRAWBACKS OF EXISTING SYSTEM	2
	1.3 PROPOSED SYSTEM	2
	1.4 ADVANTAGES OF THE PROPOSED SYSTEM	3
<b>2</b>	<b>SYSTEM ANALYSIS</b>	4
	2.1 IDENTIFICATION OF NEED	4
	2.2 FEASIBILITY STUDY	4
	2.2.1 Technical Feasibility	5
	2.2.2 Operational Feasibility	5
	2.2.3 Economical Feasibility	6
	2.3 SOFTWARE REQUIREMENT SPECIFICATION	6
	2.3.1 Hardware Requirements	6
	2.3.2 Software Requirements	7
<b>3</b>	<b>SYSTEM DESIGN</b>	11
	3.1 MODULE DESCRIPTION	11
	3.2 DATA FLOW DIAGRAM	13
	3.3 DATABASE DESIGN	18

	3.4 INPUT DESIGN	19
	3.5 OUTPUT DESIGN	22
<b>4</b>	<b>IMPLEMENTATION</b>	24
	4.1 CODE DESCRIPTION	24
	4.2 STANDARDIZATION OF THE CODING	24
	4.3 ERROR HANDLING	24
<b>5</b>	<b>TESTING AND RESULTS</b>	26
	5.1 TESTING	26
	5.1.1 Unit Testing	26
	5.1.2 Integration Testing	28
	5.1.3 Validation Testing	29
<b>6</b>	<b>CONCLUSION AND FUTURE ENHANCEMENT</b>	30
	6.1 CONCLUSION	30
	6.2 FUTURE ENHANCEMENT	30
	<b>APPENDICES</b>	31
	A. SAMPLE CODING	31
	B. SCREENSHOTS	46
	<b>REFERENCES</b>	54

## LIST OF FIGURES

FIGURE NO	FIGURE NAME	PAGE NO
3.1	Data Flow Diagram Level 0	13
3.2	Data Flow Diagram Level 1	14
3.3	Use case Diagram	16
3.4	User Login page	20
3.5	SignUp page	21
3.6	Storing in Firebase	22
3.7	Expense Details stored in Firebase	23
B.1	SignUp Page	46
B.2	Login page	47
B.3	Individual expense	48
B.4	Invite Group member	49
B.5	Set reminder	50
B.6	Group expense	51
B.7	Join Group	52
B.8	Group Budget	53



## LIST OF ABBREVIATIONS

DB	Data Base
GB	Gigabyte
DFD	Data Flow Diagram
SDK	Software Development Kit
JSON	JavaScript Object Notation
UID	Unique Identifier
API	Application Programming Interface
UI	User Interface

## **CHAPTER 1**

### **INTRODUCTION**

#### **1.1 ABOUT THE PROJECT**

The purpose of this project is to develop a Multi-Functional Finance Tracking System that simplifies financial management for both individuals and groups. Traditionally, people rely on multiple tools or methods, such as spreadsheets, budgeting applications, or group expense trackers, which often leads to fragmented data, inefficiencies, and confusion. This system aims to provide a unified solution by combining personal finance tracking with group expense management in one application. It helps individuals set budgets, track expenses, and receive reminders for bills. For groups, it enables transparent contribution tracking, automated bill splitting, and collaborative budget management.

The system allows users to easily manage their personal finances by categorizing expenses, and keeping track of bills and subscriptions. For group finances, users can collaborate in real-time, ensuring fairness and transparency in managing shared expenses. By offering a single platform for both individual and group financial management, the application reduces the need for multiple applications, simplifying the overall financial management process.

Overall, this application improves financial literacy, promotes better financial habits, and ensures transparency in group financial dealings, the application empowers users to manage their finances more effectively, making it a valuable tool for individuals and groups.

## **1.2 EXISTING SYSTEM**

Traditional tools like Microsoft Excel and Google Sheets allow manual tracking of expenses but lack automation. Budgeting applications such as Mint, YNAB, and PocketGuard help manage personal finances but don't support group expenses. Group expense applications like Splitwise and Settle Up are useful for splitting bills, but they don't handle personal budgeting. These existing tools offer limited functionality and don't integrate personal and group financial management in one place. As a result, users often have to use multiple applications to manage different aspects of their finances.

## **1.3 DRAWBACKS OF EXISTING SYSTEM**

- Existing tools often focus on personal finances or group finances.
- Group management features like contribution tracking and balancing are usually missing.
- Many tools lack insights for spending patterns
- Spreadsheets need manual effort.
- Some applications are complex.
- Data security controls are often insufficient.
- Real-time updates are missing, which leads to inaccuracies.

## **1.4 PROPOSED SYSTEM**

The Multi-Functional Finance Tracking System is an advanced application that combines personal finance tracking with group expense management to overcome the limitations of traditional financial tools. It offers a range of advanced features, user-friendly design, and modern technology to provide a complete financial management solution for both individuals and groups. It offers automated bill reminders, and alerts for overspending to assist users in managing their finances effectively.

## **1.5 ADVANTAGES OF THE PROPOSED SYSTEM**

- Combines personal and group finance management in one software, eliminating the need for multiple applications.
- Provides real-time updates for users to access the latest financial data.
- Promotes transparency and fairness through group expense tracking, bill splitting, and contribution management.
- Offers analytics to help users understand their financial behavior and improve planning.
- Ensures data security through robust encryption and privacy controls.

## **SUMMARY**

The previous chapter introduces a detailed description of the financial tracking system. It highlights the limitations of current systems and offers a versatile solution for effective personal and group finance management. It demonstrates the system's capabilities and benefits in real-time tracking, detailed analysis, and enhanced security measures. Finally, it offers users clear steps to set up and use the system in their daily routines.

## **CHAPTER 2**

### **SYSTEM ANALYSIS**

#### **2.1 IDENTIFICATION OF NEED**

Effective financial management is essential, yet individuals and groups face unique challenges that existing tools fail to address comprehensively. Personal finances often lack proper tracking and budgeting tools. Group expenses, on the other hand, suffer from disputes and inefficiencies. The disconnect between personal and group management tools leads to fragmented data, making decision-making difficult and less effective. The multi-functional finance tracking system provides real-time tracking, transparency, and actionable insights to help you manage your finances effectively. This system simplifies financial management by encouraging better spending habits and promoting collaboration among users.

#### **2.2 FEASIBILITY STUDY**

A feasibility study is a complete analysis that is conducted to evaluate the practicality and viability of a proposed project. The prime goal of a feasibility study is to determine whether the project is worth pursuing and if it can be successfully implemented within the defined constraints. Each structure has to be thought of in the developing of the project, as it has to serve the end user in a friendly manner. Three considerations involved in feasibility are

- Technical Feasibility
- Operational Feasibility
- Economic Feasibility

### **2.2.1 TECHNICAL FEASIBILITY**

Technical feasibility assesses the technical aspects of the project, such as the availability of technology, required infrastructure, and the necessary expertise for successful implementation. The proposed system is built using Flutter for cross-platform development, ensuring that the application is compatible with both Android and iOS.

Firebase will be used for backend functionalities, such as real-time database management, user authentication, and cloud storage. The tools and frameworks chosen for the project ensure that the application is scalable, responsive, and capable of handling a large number of users. The system will be able to perform all the necessary operations, from managing personal finances to group expense sharing, making it a fully functional and adaptable solution. Moreover, the ease of integration with Firebase ensures a smooth experience for developers, as well as a stable and secure infrastructure for end-users.

### **2.2.2 OPERATIONAL FEASIBILITY**

The Multi-Functional Finance Tracking System has been designed to be user-friendly, allowing both individual users and groups to easily manage their finances. It provides simple navigation and clear functionality, ensuring that even users with minimal technical expertise can easily use it.

The application is designed to solve the problems faced by users in tracking and managing personal and shared finances by offering a comprehensive set of features in one platform. By enabling budget tracking, expense categorization, bill reminders, and group expense management, the system enhances the overall user experience. The intuitive interface and real-time tracking capabilities ensure high user adoption, making it a suitable solution for a wide range of users.

### **2.2.3 ECONOMIC FEASIBILITY**

This application is developed by Flutter, Firebase, and other open-source technologies, which makes it an affordable solution for both developers and users. The use of Flutter for cross-platform development allows the same codebase to be

used for both Android and iOS, significantly reducing development time and costs. Firebase, with its pay-as-you-go pricing model, offers cost-efficient backend services, especially in the early stages of the application. These technologies are widely accessible and free, which keeps the overall development budget within limits. The system's ability to replace multiple financial management tools with a single integrated solution also results in cost savings for users. By offering a unified platform for personal and group financial management, the application not only provides users with better financial control but also ensures long-term savings. Therefore, the system is economically viable, with a focus on reducing both development costs and user expenses.

## **2.3 SOFTWARE REQUIREMENT SPECIFICATION**

A declaration that describes the capability that a system needs in order to satisfy the user's requirements is known as a system requirement. The system requirements for specific machines, software or business operations are general. Taking it all the way down to the hardware and coding that operates the software. System requirements are the most efficient way to address user needs while lowering implementation costs.

### **2.3.1 HARDWARE REQUIREMENTS**

The hardware for the system is selected considering the factors such as CPU processing speed, memory access, peripheral channel access speed, printed speed; seek time & relational data of hard disk and communication speed etc.

Processor	: Intel
core i5 RAM	: 16 GB
Monitor	: 13.3 inch
Disk	: 512
GB Keyboard	: Optical

### 2.3.2 SOFTWARE REQUIREMENTS

The software for the project is selected considering the factors such as working frontend environment, flexibility in the coding language, database knowledge of enhances in backend technology etc.

Scripting Language	: Dart
Frontend	: Flutter
Back End	: Firebase
Database	: Firebase
Tools	: Visual studio code

### FRONT END

Flutter, developed by Google, is an open-source UI toolkit designed for creating natively compiled applications for mobile, web, and desktop platforms from a single codebase. It is known for building visually appealing, fast, and highly responsive applications, making it a preferred choice for developers. Flutter's architecture is based on widgets, which serve as reusable building blocks for creating user interfaces. These widgets are highly customizable and can be nested to build complex layouts, ensuring maintainable and scalable application development.

Using the Dart programming language, Flutter achieves remarkable performance by compiling directly to native code and rendering UI with the Skia graphics engine. This approach ensures consistent performance and smooth animations across all platforms. The hot reload feature further accelerates development by instantly reflecting changes in the application without losing its state, allowing developers to debug and experiment efficiently.

Being open-source, Flutter benefits from an active community, detailed documentation, and regular updates, ensuring continuous improvement. It supports native features like cameras and sensors through platform channels and provides built-



in tools for internationalization, enabling developers to create applications for a global audience. With its efficiency, flexibility, and ability to deliver consistent user experiences, Flutter is ideal for projects ranging from simple prototypes to complex enterprise-grade applications.

## FEATURES OF FLUTTER

There are unique features available on React because that it is widely popular.

**Widget-Based Architecture:** One of Flutter's core building blocks is the widget. In Flutter, everything is a widget, whether it's a button, text, layout, or even the entire screen. Widgets can be reused, customized, and nested to build complex user interfaces efficiently. The modularity of widgets simplifies UI development, as developers can work on smaller parts independently before combining them into a complete application.

**Hot Reload:** Flutter provides a powerful Hot Reload feature that allows developers to see changes in the code instantly without losing the application state. This significantly speeds up development and debugging processes, enabling developers to experiment and iterate quickly.

**Performance:** Flutter delivers exceptional performance because it renders UI components directly to the screen using the Skia Graphics Engine, bypassing the need for native platform components. It uses Dart (an optimized programming language) for faster execution and ensures smooth animations at 60 FPS or even 120 FPS on capable devices.

## BACK END

Firebase is a comprehensive, open-source (partially), and cross-platform Backend-as-a-Service (BaaS) platform developed by Google. It provides ready-to-use services for building scalable, real-time back-end solutions without managing servers. Firebase is not a programming language or framework but a suite of cloud-based tools that handle data storage, authentication, hosting, and more.

It is widely used for building back-end services for web, mobile, and serverless applications. Large organizations, including Google, Alibaba, Lyft, and The New York Times, rely on Firebase for its performance and scalability.

## FEATURES OF FIREBASE

**Real-Time Data Synchronization:** Firebase uses Realtime Database and Cloud Firestore to sync data in real time across all connected clients, enabling fast and interactive applications.

**Event-Driven:** Firebase works asynchronously and supports event-driven architecture with real-time triggers, allowing applications to update data instantly without polling servers.

**Highly Scalable:** Firebase leverages Google Cloud infrastructure, making it highly scalable and capable of handling millions of concurrent users without manual intervention.

**No buffering:** Firebase handles data efficiently by enabling instant data transfer and streaming, ideal for applications with audio, video, or live content.

**Open source tools:** While Firebase itself is proprietary, many Firebase libraries and SDKs are open source and actively maintained by the community.

**License:** Firebase SDKs and tools are available under open-source licenses, and the platform is backed by Google's reliable infrastructure.

## FIREBASE

Firebase is a powerful cloud-based NoSQL database system that provides scalable and flexible data management for real-time applications. It includes two primary database options: Firebase Realtime Database and Cloud Firestore, both designed to handle unstructured or semi-structured data efficiently. Firebase stores data in a JSON-like structure, allowing developers to work with flexible schemas that evolve as the application requirements change.

Unlike traditional relational databases, Firebase is schema-less, which makes it easier to adapt to changing data models without significant overhead. Firebase ensures high performance for real-time data synchronization by leveraging a WebSocket-based connection. It supports horizontal scaling through Google Cloud infrastructure, enabling applications to handle heavy loads efficiently by distributing data across servers.

## FEATURES OF FIREBASE

**JSON-Like Data Model:** Firebase stores data in a hierarchical JSON structure (Realtime Database) or as documents grouped into collections (Cloud Firestore). This makes it easy for developers to store and access data intuitively without rigid tables or relationships.

**Sharding and Horizontal Scaling:** Firebase's databases can scale horizontally by distributing data across multiple servers automatically through Google Cloud infrastructure, ensuring high availability and performance even under heavy loads.

**Replication:** Firebase ensures data redundancy and fault tolerance through replication. Data is automatically replicated across regions, reducing points of failure and ensuring that applications remain available even during server crashes or hardware issues.

**Time Series Data:** Firebase can handle time-stamped data effectively, such as logs, metrics, or sensor data. Cloud Firestore's support for range queries and timestamp fields allows efficient management and querying of time-series data.

## SUMMARY

The system requirements are fully explained in detail for each piece of hardware and software that was used to meet the user and system requirements. The above have explained the features of Flutter which plays main role in the project. The detailed information regarding the system design will be presented in the following chapter 3.

## **CHAPTER 3**

### **SYSTEM DESIGN**

#### **3.1 MODULE DESCRIPTION**

A module description provides detailed information about the module and its supported components, which is accessible in different manners.

The project contains the following module:

- Individual Expense
- Group Expense
- Invite Member
- Split Expense
- Add reminder

#### **INDIVIDUAL EXPENSE MODULE**

The individual expense module focuses on managing a user's personal financial activities. Users can add, edit, or delete expenses or income entries, specifying details like name, amount, and type (income or expense). The module calculates and displays a summary of total income, expenses, and the remaining balance. Search functionality allows users to filter entries for easy tracking. All data is securely stored in Firebase Firestore, linked to the user's UID. This module empowers users to monitor and manage their finances efficiently, helping them track spending patterns and savings over time with an intuitive and visually appealing interface.

#### **GROUP EXPENSE MODULE**

The group expense module enables collaborative financial management for shared events or activities. Users can create a group and assign themselves as the group owner. Within the group, members can add shared expenses, such as trip costs

or group meals. Expenses are stored under the group's unique ID in Firebase Firestore. Group members can view the collective expenditure and contributions in real time. The module promotes transparency by clearly showing how much each member owes or has paid. It is ideal for simplifying financial responsibilities in shared settings, ensuring fair cost-sharing and accurate tracking for all participants.

## **INVITE MEMBER MODULE**

The invite member module allows group owners to expand their collaboration by inviting others. Owners can send invites via email or share the group's unique ID. Invited members can join by accepting the email invite or entering the group ID. This module restricts inviting permissions to group owners, ensuring controlled access to the group. Firebase Firestore maintains member details for easy management. It ensures seamless onboarding and effective collaboration within groups, making it convenient for participants to join and contribute to shared expenses. This feature simplifies group setup, enhancing the application's utility in collaborative financial scenarios.

## **SPLIT EXPENSE MODULE**

The split expense module simplifies shared financial management by dividing group expenses fairly among members. When an expense is added, the module calculates the share for each participant, based on equal or predefined proportions. It displays each member's balance, showing how much they owe or are owed. Updates occur dynamically in Firebase Firestore, ensuring real-time synchronization. The module helps settle balances through clear breakdowns of individual contributions. Ideal for trips or shared events, this feature eliminates manual calculations and promotes fairness, making group financial collaboration smooth and transparent for all users.

## ADD REMINDER MODULE

The add reminder module helps users stay organized by setting reminders for upcoming financial tasks. Users can schedule reminders for bill payments, expense due dates, or recurring expenses. Reminders include details like the name, amount, and due date. Notifications alert users before deadlines, ensuring they never miss an important payment. This module integrates seamlessly with other features, allowing users to link reminders to specific expenses. Stored in Firebase Firestore, reminders are persistent and accessible across sessions. By providing timely alerts, this module supports users in maintaining their financial responsibilities and avoiding late fees or missed payments.

### 3.2 DATA FLOW DIAGRAM

The Data Flow Diagram provides information about the inputs and outputs of each entity and process itself.

#### LEVEL 0

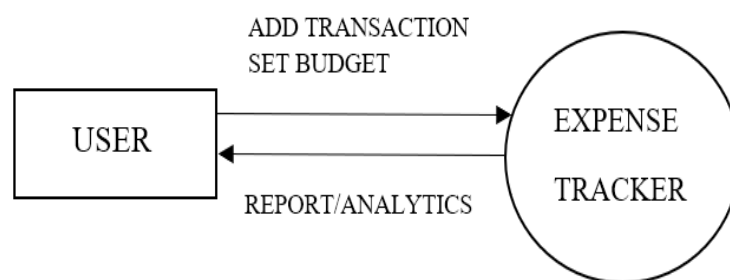


Figure 3.1 Data Flow Diagram Level 0

In Figure 3.1 User can Login to the application and make use of their respective process in the system. In this, user can able to manage the expense details. User can view transaction and report expense.

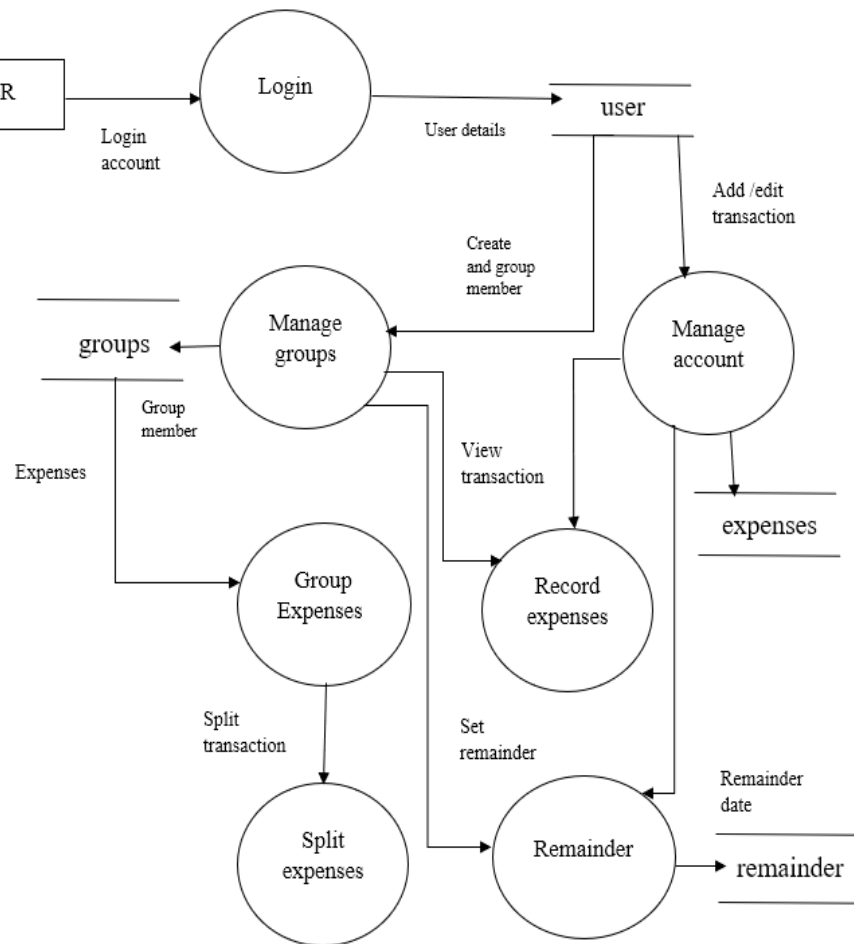
**LEVEL 1**

Figure 3.2 Data Flow Diagram Level 1

**DESCRIPTION:**

## 1. User Login:

- The user logs in by providing credentials.
- The system verifies credentials against the User Database and retrieves user details upon successful authentication.

## 2. User Account Management:

- The user manages their account, including adding/editing transactions, viewing account balances, and setting budgets.
- The system updates the Account Database and stores expense data in the Transaction Database.

## 3. Group Management:

- The user creates groups, adds members, and views group transactions.
- The system interacts with the Group Database to store group-related data.
- Group transactions are managed and stored in the Transaction Database.

## 4. Expense Tracking:

- The user records expenses by entering transaction details.
- The system stores transaction data in the Transaction Database.
- The user can view transaction history by retrieving data from the same database.

## 5. Expense Splitting:

- The user splits expenses among group members.
- The system updates the Transaction Database to assign split amounts to individual members.

## 6. Reminders:

- The user sets reminders for upcoming expenses or bills.
- The system updates the Transaction Database to assign split amounts to individual members.



## USE CASE DIAGRAM

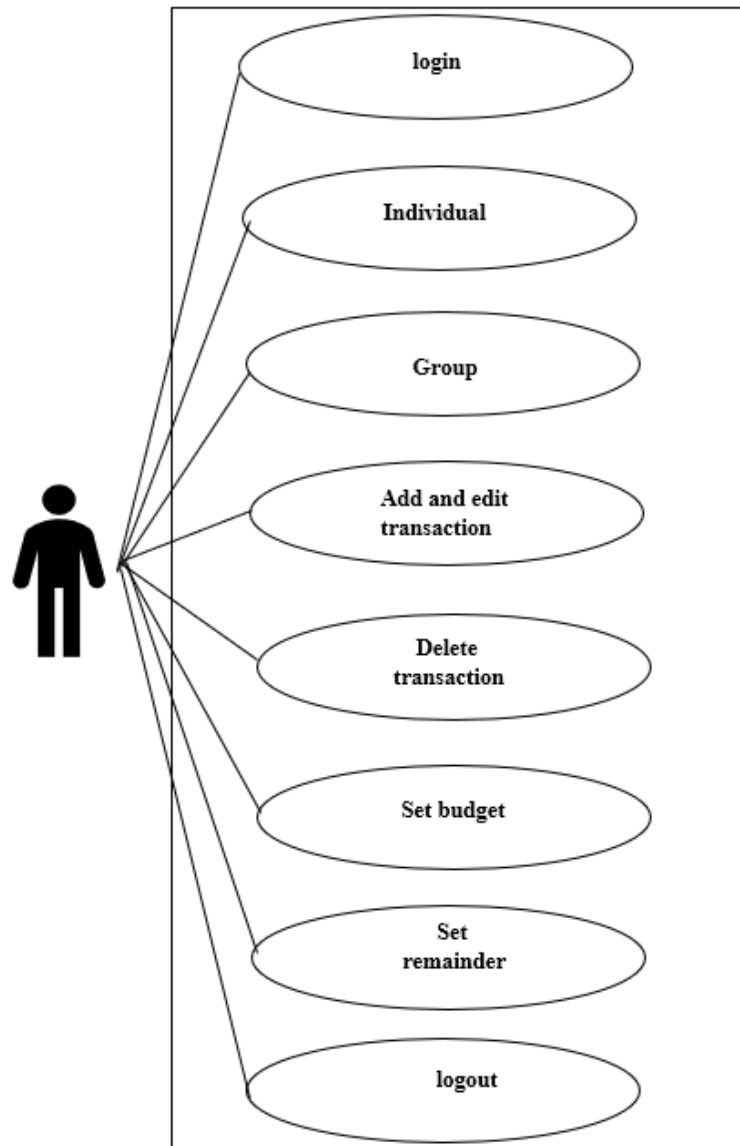


Figure 3.3 Use case diagram

## **DESCRIPTION:**

### **Use Cases:**

1. Login : Allows the user to securely access their account.
2. Individual : Focused on managing personal financial activities like tracking income, expenses, and budgets for individual use.
3. Group : Enables users to collaborate in managing group finances, such as shared expenses or group budgets.
4. Add and Edit Transaction: Users can add or modify financial transactions, such as recording income, expenses, or payments.
5. Delete Transaction: Provides the ability to remove incorrect or unnecessary transactions from the system.
6. Set Budget: Allows users to define a budget for individual or group purposes, ensuring better financial planning.
7. Set Reminder: Users can set reminders for bill payments, dues, or other financial obligations.
8. Logout: Ends the user session and ensures account security.

### **Relationships**

1. The user interacts with all functionalities provided in the system.
2. Each use case is connected to the user through lines indicating the actions they can perform.

### **3.3 DATABASE DESIGN**

Database design is the organization of data according to a database model. The designer determines what data must be stored and how the data elements interrelate. With this information, they can begin to fit the data to the database model. Database management system manages the data accordingly. The term database design can be used to describe many different parts of the design of an overall database system.

Principally, and most correctly, it can be thought of as the logical design of the base data structure used to store the data. In an object database the entities and relationships map directly to object classes and named relationships. However, the term database design could also be used to apply to the overall process of designing, not just the base data structure, but also the forms and queries used as part of the overall database application within the database management system.

### **DATA INTEGRATION**

In a database, information from several files is coordinated, accessed and operated upon as though it is in a single file. Logically, the information is centralized, physically, the data may be located on different devices, connected through data communication facilities. Data integrity means storing all data in one place only and determining how each application has to access it. This leads to less data redundancy, a reduction in the direct access storage requirement.

### **DATA INDEPENDENCE**

Data independence is the insulation of application programs from changing aspects of physical data organization. This objective seeks to allow changes in the content and organization of physical data without reprogramming of applications and to allow modifications to application program without recognizing the physical data. If the database changes and expands over time, it is very important that the changes in one level should not affect the data at other levels of the database. The table needed for each module were designed and the specification of each and every column was given based on the records and details collected during record specification of the system study, is shown in the below.

## **DATA SECURITY**

Data security refers to the process of protecting data from unauthorized access and data corruption throughout its lifecycle. Data security includes data encryption, hashing, tokenization and key management practices that protect data across all applications and platforms. Data security means protecting digital data, such as those in a database, from destructive forces and from the unwanted actions of unauthorized users, such as cyberattacks or a data breach.

Security requirements placed restrictions on the use of this application by the admin and customers of wireless LAN communicator only, control access to the data, provide different kinds of requirements to different people, require the use of passwords. Organizations around the globe are investing heavily in information technology, cyber security capabilities to protect their critical assets.

### **3.4 INPUT DESIGN**

Input design is the process of converting user-originated inputs to a computer understandable format. Input design is one of the most expensive phase of the operation of computerized system and is often the major problem of a system. A large number of problems with a system can usually be tracked back to fault input design and method.

Every moment of input design should be analyzed and designed with utmost care. The decisions made during the input design are the project gives the low time consumption to make sensitive application made simple. Thus, the developed system is well within the budget. This was achieved because most of the technologies used are freely available. Only the customized product had to be purchased. In the project, the forms are designed with easy-to-use option. The coding is being done such that proper validation are made to get the perfect input. No error inputs are accepted.

## LOGIN FORM

This is a user login form, the user can login with their registered mail id and password to access the application. After completing the registration, the user should give their correct register details else the user cannot able to login.

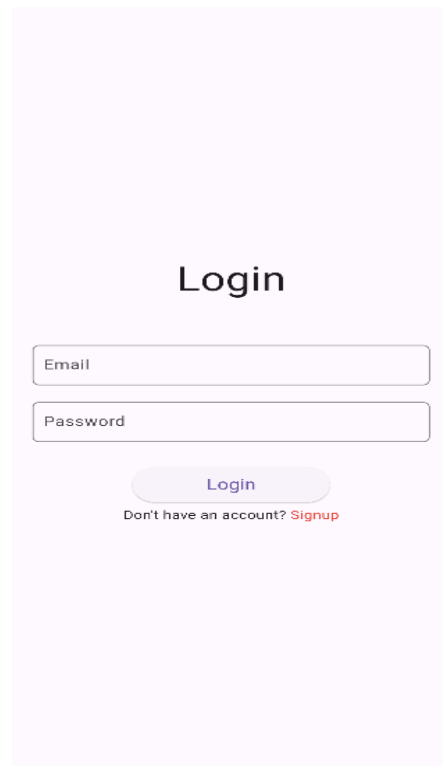
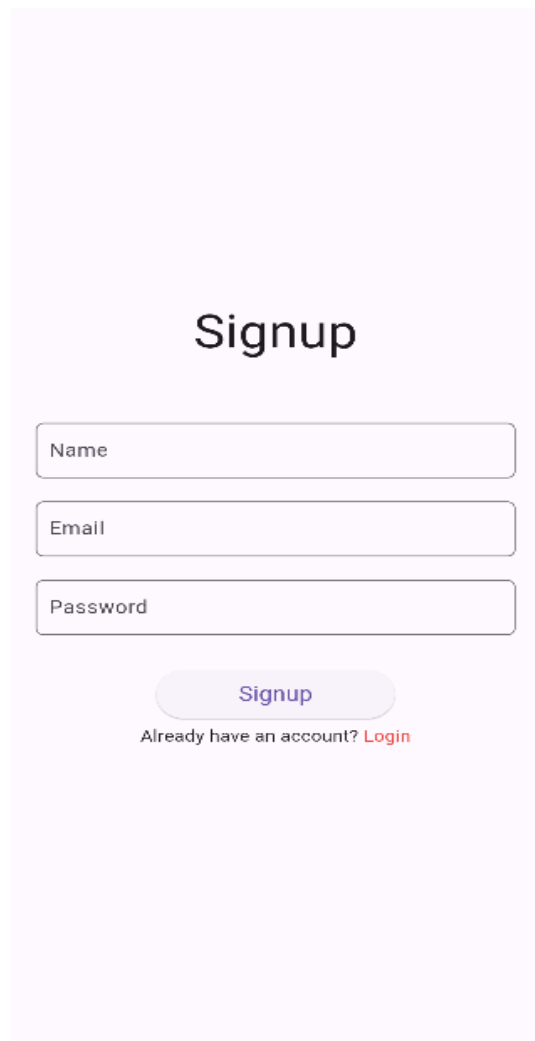
The image shows a user login page with a light purple background. At the top center, the word "Login" is displayed in a bold, black, sans-serif font. Below the title, there are two input fields: the first is labeled "Email" and the second is labeled "Password". Both fields are white with a thin purple border. Below the password field, there is a purple button with the word "Login" in white. At the bottom, there is a link that says "Don't have an account? Signup", where "Signup" is in red and the rest is in black.

Figure 3.4 User Login page

The Figure 3.4 is user login form. Login form has mail id and password. The username should be in proper email format. If the fields are not filled, form will not be submitted. Password must contain at least 8 digit or characters. If the form is submitted successfully the page redirected towards home page.

## SIGNUP FORM



Signup

Name

Email

Password

Signup

Already have an account? [Login](#)

Figure 3.5 SignUp Page

The Figure 3.5 is user has to register to the page by giving the required details. After register user has to login with the authorized mail id so that the user can be redirected to the new transaction. user can now able to manage the expense and transaction, set reminder and manage group expenses by invite group member.

### 3.5 OUTPUT DESIGN

Output design generally refers to the results and information that are generated by the system for many end-users; it should be understandable with the enhanced format. Computer output is most important direct source of information to the user. Output design deals with form design. Efficient output design should improve the interfacing with user. The term output applies to any information produced by an information system in terms of data displayed. When analyst design system output, they identify the specific output that is needed to meet the requirements of end user.

Previewing the output reports by the user is extremely important because the user is the ultimate judge of the quality of the output and in turn, the success of the system. When designing output, system analysis accomplishes more things like, to determine what applications, website or documents are blocked or allowed. The output is designed in such way that is attractive, convenient and informative.

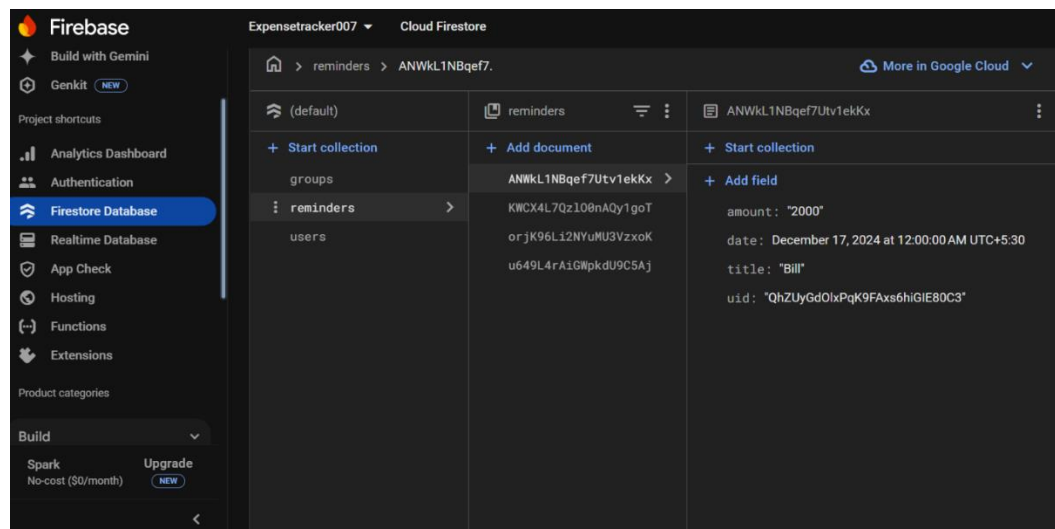


Figure 3.6 Storing in Firebase

The Figure 3.6 shows the information like details of login, user details. In Firebase data is stored as JSON application data.

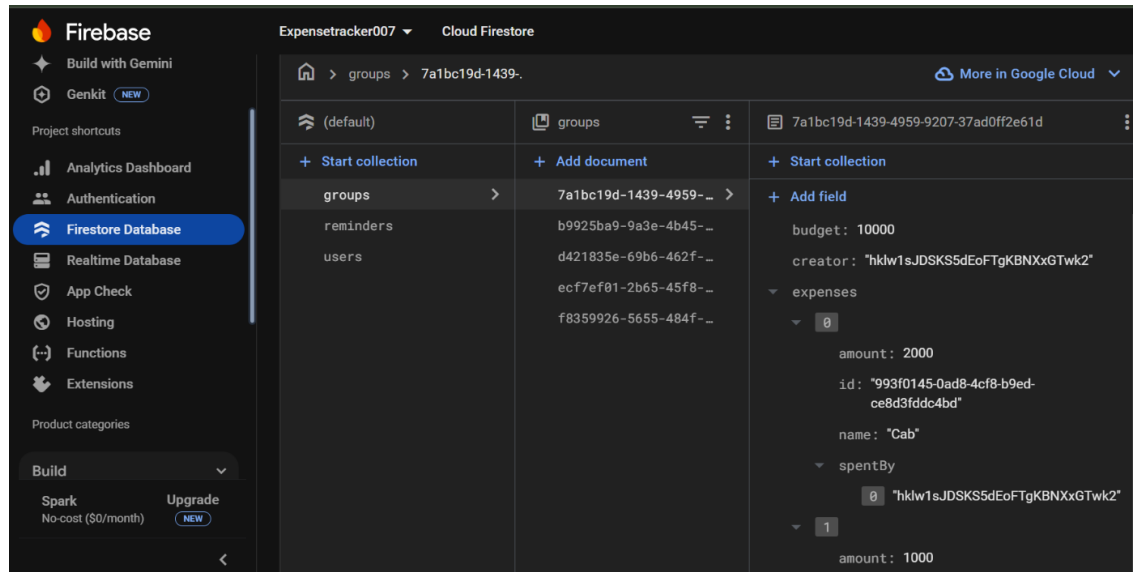


Figure 3.7 Expense Details Stored In Firebase

The Figure 3.7 Shows the information that stores in Firebase about the expense of group member.

## SUMMARY

The system design contains a detailed description of the modules. Login Page, Upload expense details, reminder, individual expense are all mentioned in the module description. The Data flow diagrams depicts the flow of the application from the user entry to the exit. It flows through all the processes.



## **CHAPTER 4**

### **IMPLEMENTATION**

#### **4.1 CODE DESCRIPTION**

Code description can be used to summarize code or to explain the programmer's intent. Good comments don't repeat the code or explain it. They clarify its intent. Comments are sometimes processed in various ways to generate documentation external to the source code itself by document generator or used for integration with systems and other kinds of external programming tools. I have chosen flutter as front end and firebase as back end.

#### **4.2 STANDARDIZATION OF THE CODING**

Coding standards define a programming style. A coding standard does not usually concern itself with wrong or right in a more abstract sense. It is simply a set of rules and guidelines for the formatting of source code. The other common type of coding standard is the one used in or between development teams. Professional code performs a job in such a way that is easy to maintain and debug. All the coding standards are followed while creating this project. Coding standards become easier, the earlier you start. It is better to do a neat job than cleaning up after all is done. Every coder will have a unique pattern than he adheres to such a style might include the conventions he uses to name variables and functions and how he comments his work. When the said pattern and style is standardized, it pays off the effort well in the long.

#### **4.3 ERROR HANDLING**

Exception handling is a process or method used for handling the abnormal statements in the code and executing them. It also enables to handle the flow control of the code/program. For handling the code, various handlers are used that process the exception and execute the code. Flutter provides mechanisms like try-catch blocks to catch exceptions that may occur, such as errors when the application fails to fetch data from Firebase or when users input invalid transaction details.

An error is a serious problem than an application doesn't usually get pass without incident. Errors cause an application to crash, and ideally send an error message offering some suggestion to resolve the problem and return to a normal operating state, there is no way to deal with errors "live" or in production the only solution is to detect them via error monitoring and bug tracking and dispatch a developer or two to sort out the code.

## **USER INTERFACE DESIGN**

Input design is the process of converting user originated inputs to a computer understandable format. Input design is one of the most expensive phases of the operation of computerized system and is often the major problem of a system. A large number of problems with a system can usually be tracked back to fault input design and method. Every moment of input design should be analyzed and designed with utmost care. To provide cost effective method of input. To achieve the highest possible level of accuracy. To ensure that the input is understood by the user. System analysis decide the following input design details like, what data to input, what medium to use, how the data should be arranged or coded, data items and transactions needing validations to detect errors and at last the dialogue to guide user in providing input. Input data of a system may not be necessarily is raw data captured in the system from scratch. These can also be the output of another system or subsystem.

## **SUMMARY**

The above implementation section explained that the purpose of a code description is to summarise the code or to clarify the programmer's intent. Good comments don't repeat or explain the code. A programming style is defined by coding standards. A coding standard isn't usually concerned with what's proper or bad in a broader sense. Exception handling is a method or process for dealing with and executing anomalous statements in code. Next Chapter 5 is shown about the Testing.

## **CHAPTER 5**

### **TESTING AND RESULTS**

#### **5.1 TESTING**

Software testing serves as the final assessment of specifications, designs, and coding and is a crucial component of software quality assurance. The system is tested throughout the testing phase utilizing diverse test data. The preparation of test data is essential to the system testing process. The system under study is tested after the test data preparation. Once the source code is complete, relevant data structures should be documented. The finished project must go through testing and validation, when errors are explicitly targeted and attempted to be found.

The project developer is always in charge of testing each of the program's separate units, or modules. Developers frequently also perform integration testing, which is the testing phase that results in the creation of the complete program structure.

This project has undergone the following testing procedures to ensure its correctness

- Unit testing
- Integration Testing
- Validation Testing

##### **5.1.1 UNIT TESTING**

A testing strategy known as unit and integration testing has been used to check that the system behaves as expected. The testing strategy was based on the functionality and the requirements of the system. In unit checking out, we have to check the applications making up the device.

This enables, to stumble on mistakes in coding and common sense which might be contained with the module alone. The checking out became completed at some stage in programming level itself.

### Test Case 1

Module	: User Authentication
Test Type	: Unit Testing
Input	: Username and Password
Expected Output	: Authenticate user

### Sample Test

Output	: User successfully authenticated
Analysis	: The test ensures that the system can authenticate users based on the provided username and password. If authentication fails for valid credentials or successful authentication occurs for invalid credentials, it indicates a problem with the user authentication module.

### Test Case 2

Module	: Expense Tracking
Test Type	: Unit Testing
Input	: Expense Category, Amount
Expected Output	: Expense successfully recorded

### Sample Test

Output	: Expense recorded successfully in the correct category.
Analysis	: This test ensures the system can record expenses accurately under specific categories.

### 5.1.2 INTEGRATION TESTING

Integration testing is done to test itself if the individual modules work together as one single unit. In integration testing, the individual modules that are to be integrated are available for testing. Thus, the manual test data that used to test the interfaces replaced by that which in generated automatically from the various modules. It can be used for testing how the module would actually interact with the proposed system. The modules are integrated and tested to reveal the problem interfaces.

#### Test Case 1

Module : Authentication & User  
 Test Type : Integration Testing  
 Input : User login  
 Expected Output : Access transaction

#### Sample Test

Output : Successfully accessed transaction  
 Analysis : This test validates that when user login are provided, access their transaction.

#### Test Case 2

Module : Individual & Group Expense  
 Test Type : Integration Testing  
 Input : Add Expense  
 Expected Output : Expense successfully recorded

#### Sample Test

Output : Expense recorded successfully in the correct category.  
 Analysis : This test checks the integration between the expense tracking module and group features. It ensures group expenses are accurately logged, split, and visible to all members in real time.

### 5.1.3 VALIDATION TESTING

Verification and validation checking out are critical tests, which might be achieved earlier than the product has been surpassed over to the customer. This make sure, that the software program checking out lifestyles cycle begins off evolved early. The intention of each verification and validation is to make certain that the product is made in step with the necessities of the customer and does certainly fulfil the supposed purpose.

#### Test Case 1

Module	: Group Expense
Test Type	: Validation Testing
Input	: Add only invited member
Expected Output	: Error message prompting the user not invited to this group

#### Sample Test

Output	: Error message displayed “you are not invited to this group”
Analysis	: This test ensures that the system validates whether a group member can only be added if the user is invited.

### SUMMARY

The preceding chapter discusses the many types of testing, including unit testing, integration testing and system testing. During the system evaluation following the completion of the source code, it is documented as associated data structures. The final project must go through testing and validation, which includes both subtle and overt attempts to find problems.

## **CHAPTER 6**

### **CONCLUSION AND FUTURE ENHANCEMENT**

#### **6.1 CONCLUSION**

The Multi-Functional Finance Tracking System is a comprehensive tool designed to help both individuals and groups manage their finances efficiently. In today's fast-paced world, keeping track of personal and shared expenses is essential for financial well-being. Key features include tools for setting budgets, real-time expense tracking, and financial analytics that give users clear insights into their spending habits.

For individual users, the system allows them to manage their finances by setting budgets, tracking expenditures, and receiving alerts for upcoming bills. For group users, it offers transparent shared expense management, making it easy to track contributions and expenses among members, ensuring fairness and accountability. Security is a priority, with advanced encryption to protect sensitive financial data. By enabling features like collaborative budgeting and expenditure sharing, this project promotes financial literacy and responsible spending. Overall, the Multi-Functional Finance Tracking System empowers individuals and groups to take control of their finances, fostering a culture of financial awareness and stability.

#### **6.2 FUTURE ENHANCEMENT**

- Integrate AI to analyze user spending patterns, providing personalized advice on how to save money, invest wisely, and optimize budgets based on historical data.
- Integrate voice assistants (like Google Assistant or Alexa) to allow users to add expenses or check their budget status through voice commands.
- Implement multi-factor authentication (MFA) for added security, especially for groups and individuals who manage sensitive financial data.

## APPENDICES

### A.SAMPLE CODING

#### HOME\_SCREEN.DART

```
import 'package:flutter/material.dart';
import 'individual_expenses_screen.dart';
import 'group_expenses_screen.dart';
import 'reminders_screen.dart';
import 'package:finance_tracker/auth/login_screen.dart';

class HomeScreen extends StatefulWidget {
  final String uid; // Add uid parameter

  const HomeScreen({super.key, required this.uid}); // Require uid

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  int _currentIndex = 0;

  late final List<Widget> _screens;

  @override
  void initState() {
    super.initState();
    _screens = [
      FirestoreCRUD(uid: widget.uid),
      GroupFinanceTracker(currentUserId: widget.uid),
      Reminders(uid: widget.uid),
    ];
  }
}
```



```
}
```

```
@override
```

```
Widget build(BuildContext context) {
```

```
  return Scaffold(
```

```
    body: _screens[_currentIndex],
```

```
    bottomNavigationBar: BottomNavigationBar(
```

```
      currentIndex: _currentIndex,
```

```
      onTap: (index) {
```

```
        if (index == _screens.length) {
```

```
          Navigator.of(context).pushReplacement(
```

```
            MaterialPageRoute(builder: (context) => const LoginScreen()));
```

```
        } else {
```

```
          setState(() {
```

```
            _currentIndex = index;
```

```
          });
```

```
        }
```

```
      },
```

```
      items: const [
```

```
        BottomNavigationBarItem(
```

```
          icon: Icon(Icons.account_balance),
```

```
          label: "Individual",
```

```
        ),
```

```
        BottomNavigationBarItem(
```

```
          icon: Icon(Icons.group),
```

```
          label: "Group",
```

```
        ),
```

```
        BottomNavigationBarItem(
```

```
          icon: Icon(Icons.alarm),
```

```
          label: "Reminder",
```

```
        ),
```

```
        BottomNavigationBarItem(
```

```
          icon: Icon(Icons.logout),
```

```

        label: "Logout",
      ),
    ],
    backgroundColor: const Color.fromARGB(
      255, 247, 246, 249),
    selectedItemColor: const Color(0xFF6200EA),
    unselectedItemColor: Colors.black54,
  ),
);
}
}

```

## **GROUP EXPENSE SCREEN.DART**

```

import 'package:flutter/material.dart';
import 'package:uuid/uuid.dart';
import 'package:url_launcher/url_launcher.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'group_details_page.dart';

class GroupFinanceTracker extends StatefulWidget {
  final String currentUserId;

  const GroupFinanceTracker({required this.currentUserId});

  @override
  _GroupFinanceTrackerState createState() => _GroupFinanceTrackerState();
}

class _GroupFinanceTrackerState extends State<GroupFinanceTracker> {
  final TextEditingController _groupNameController = TextEditingController();
  final TextEditingController _budgetController = TextEditingController();
  final TextEditingController _groupUidController = TextEditingController();

```

```

final FirebaseFirestore _firestore = FirebaseFirestore.instance;

Future<String> _getUserEmail() async {
  final userDoc =
    await _firestore.collection('users').doc(widget.currentUserId).get();
  return userDoc.data()?['email'] ?? '';
}

void _createGroup() {
  if (_groupNameController.text.isNotEmpty &&
    _budgetController.text.isNotEmpty) {
    final groupId = const Uuid().v4();
    final budget = double.tryParse(_budgetController.text);

    if (budget != null) {
      _firestore.collection('groups').doc(groupId).set({
        'id': groupId,
        'name': _groupNameController.text,
        'members': [
          {'uid': widget.currentUserId}
        ],
        'expenses': [],
        'budget': budget,
        'creator': widget.currentUserId,
        'invitedEmails': [],
      }).then((_) {
        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(
            content: Text('Group "${_groupNameController.text}" created!'),
          ),
        );
        _groupNameController.clear();
        _budgetController.clear();
      });
    }
  }
}

```

```

Navigator.pushReplacement(
  context,
  MaterialPageRoute(
    builder: (context) => GroupDetailsPage(groupId: groupId),
  ),
);
});
}
}
}

void _addExistingGroup() async {
  if (_groupUidController.text.isNotEmpty) {
    final groupId = _groupUidController.text.trim();
    final groupDoc = await _firestore.collection('groups').doc(groupId).get();

    if (groupDoc.exists) {
      final groupData = groupDoc.data() as Map<String, dynamic>;
      final invitedEmails = groupData['invitedEmails'] ?? [];
      final currentUserEmail = await _getUserEmail();

      if (invitedEmails.contains(currentUserEmail)) {
        if (!(groupData['members'] as List)
          .any((member) => member['uid'] == widget.currentUserId)) {
          await _firestore.collection('groups').doc(groupId).update({
            'members': FieldValue.arrayUnion([
              {'uid': widget.currentUserId}
            ]),
          });
        }
      }

      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Successfully joined the group!')),
      );
    }
  }
}

```

```

);

Navigator.push(
  context,
  MaterialPageRoute(
    builder: (context) => GroupDetailsPage(groupId: groupId),
  ),
);
} else {
  ScaffoldMessenger.of(context).showSnackBar(
    const SnackBar(content: Text('You are already a member.')),
  );
}
} else {
  ScaffoldMessenger.of(context).showSnackBar(
    const SnackBar(content: Text('You are not invited to this group.')),
  );
}
} else {
  ScaffoldMessenger.of(context).showSnackBar(
    const SnackBar(content: Text('Invalid UID. No such group found.')),
  );
}
}
}

void _openGroupCreationOptionsDialog() {
  showDialog(
    context: context,
    builder: (context) {
      return AlertDialog(
        title: const Text('Select an Option'),

```

```

content: Column(
  mainAxisAlignment: MainAxisAlignment.min,
  children: [
    ElevatedButton(
      onPressed: () {
        Navigator.pop(context);
        _openExistingGroupDialog();
      },
      child: const Text('Add Existing Group'),
    ),
    const SizedBox(height: 10),
    ElevatedButton(
      onPressed: () {
        Navigator.pop(context);
        _openGroupCreationDialog();
      },
      child: const Text('Create New Group'),
    ),
  ],
),
);
},
);
}

```

```

void _openExistingGroupDialog() {
  showDialog(
    context: context,
    builder: (context) {
      return AlertDialog(
        title: const Text('Enter Group UID'),
        content: TextField(
          controller: _groupUidController,

```

```

        decoration: const InputDecoration(
          labelText: 'Group UID',
          border: OutlineInputBorder(),
        ),
      ),
      actions: [
        TextButton(
          onPressed: () => Navigator.pop(context),
          child: const Text('Cancel'),
        ),
        TextButton(
          onPressed: () {
            Navigator.pop(context);
            _addExistingGroup();
          },
          child: const Text('Join Group'),
        ),
      ],
    );
  },
);
}

```

```

void _openGroupCreationDialog() {
  showDialog(
    context: context,
    builder: (context) {
      return AlertDialog(
        title: const Text('Create New Group'),
        content: Column(
          mainAxisAlignment: MainAxisAlignment.min,
          children: [
            TextField(

```

```

        controller: _groupNameController,
        decoration: const InputDecoration(
          labelText: 'Group Name',
          border: OutlineInputBorder(),
        ),
      ),
    const SizedBox(height: 10),
    TextField(
      controller: _budgetController,
      decoration: const InputDecoration(
        labelText: 'Group Budget',
        border: OutlineInputBorder(),
      ),
      keyboardType: TextInputType.number,
    ),
  ],
),
actions: [
  TextButton(
    onPressed: () => Navigator.pop(context),
    child: const Text('Cancel'),
  ),
  TextButton(
    onPressed: () {
      Navigator.pop(context);
      _createGroup();
    },
    child: const Text('Create'),
  ),
],
);
},
);

```



```
}
```

```
void _sendInvite(String groupId) {
  final TextEditingController emailController = TextEditingController();

  showDialog(
    context: context,
    builder: (context) {
      return AlertDialog(
        title: const Text('Invite Member'),
        content: TextField(
          controller: emailController,
          decoration: const InputDecoration(
            labelText: 'Enter Email Address',
            border: OutlineInputBorder(),
          ),
        ),
        actions: [
          TextButton(
            onPressed: () => Navigator.pop(context),
            child: const Text('Cancel'),
          ),
          TextButton(
            onPressed: () async {
              final email = emailController.text.trim();
              if (email.isNotEmpty) {
                await _firebase.collection('groups').doc(groupId).update({
                  'invitedEmails': FieldValue.arrayUnion([email]),
                });
                Navigator.pop(context);

                final Uri mailUri = Uri(
                  scheme: 'mailto',
```

```

        path: email,
        query:

'subject=Group%20Invitation&body=Your%20group%20UID%20is%20$g
roupId',
    );
    await launch(mailUri.toString());

    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('Invitation sent to $email'),
      ),
    );
  } else {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
        content: Text('Please enter a valid email address.'),
      ),
    );
  }
},
child: const Text('Send Invite'),
),
],
);
},
);
}

void _deleteGroup(String groupId) {
  _firestore.collection('groups').doc(groupId).delete().then((_) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(

```

```

        content: Text('Group deleted successfully'),
      ),
    );
  }).catchError((error) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('Failed to delete group: $error'),
      ),
    );
  });
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Group Finance Tracker',
        style: TextStyle(color: Colors.white)),
      backgroundColor: const Color(0xFF6200EA),
    ),
    body: StreamBuilder<QuerySnapshot>(
      stream: _firestore.collection('groups').where('members',
        arrayContains: {'uid': widget.currentUserid}).snapshots(),
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.waiting) {
          return const Center(child: CircularProgressIndicator());
        }

        if (snapshot.hasError) {
          return Center(child: Text('Error: ${snapshot.error}'));
        }

        if (!snapshot.hasData || snapshot.data!.docs.isEmpty) {

```

```

    return const Center(child: Text('No groups available.'));
  }

  final groups = snapshot.data!.docs;

  return Padding(
    padding: const EdgeInsets.all(16.0),
    child: ListView.builder(
      itemCount: groups.length,
      itemBuilder: (context, index) {
        final group = groups[index];
        final groupId = group['id'];
        final groupName = group['name'];
        final groupOwner = group['creator'] as String;

        return Card(
          margin: const EdgeInsets.symmetric(vertical: 10),
          child: ListTile(
            contentPadding: const EdgeInsets.symmetric(
              vertical: 10, horizontal: 15),
            title: Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              Expanded(
                child: Text(
                  groupName,
                  style: const TextStyle(
                    fontSize: 18, fontWeight: FontWeight.bold),
                  overflow: TextOverflow
                    .ellipsis,
                ),
              ),
              Row(

```

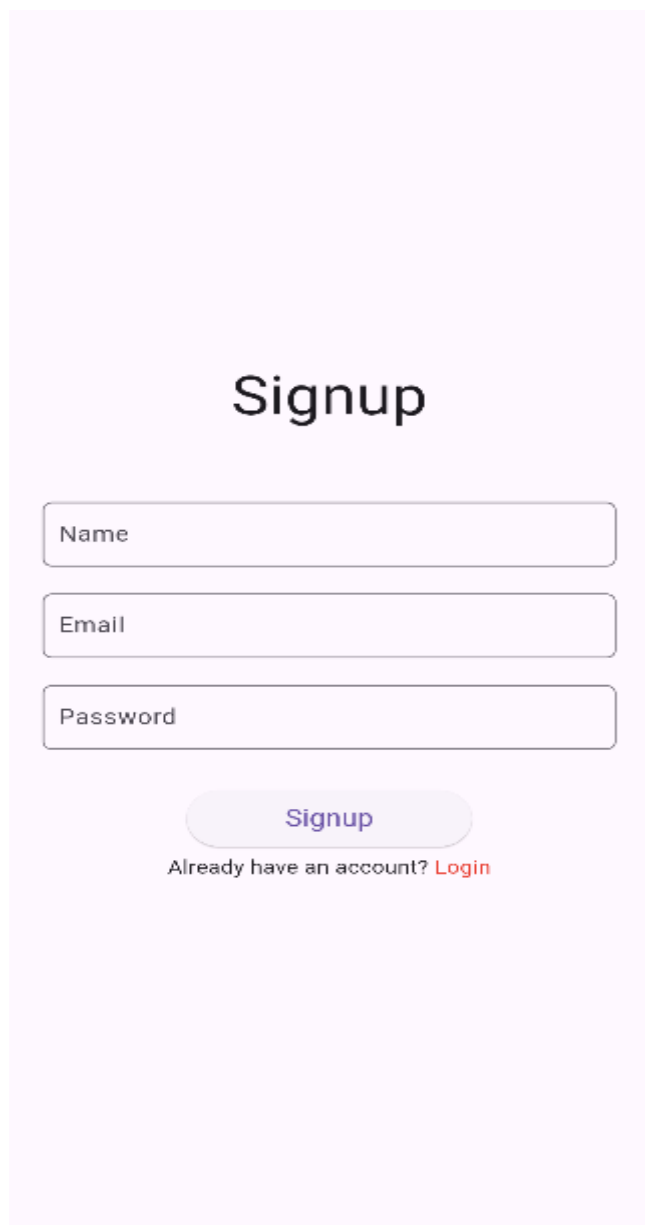
```

children: [
  IconButton(
    icon: const Icon(Icons.group_add),
    onPressed: () {
      if (group['creator'] == widget.currentUserId) {
        _sendInvite(group['id']);
      } else {
        ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(
            content: Text(
              'Only the group owner can invite members.')),
          );
      }
    },
  ),
  IconButton(
    icon: const Icon(Icons.delete),
    color: Colors.red,
    onPressed: () {
      if (group['creator'] == widget.currentUserId) {
        _deleteGroup(group['id']);
      } else {
        ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(
            content: Text(
              'Only the group owner can delete this group.')),
          );
      }
    },
  ),
],
),
],

```



## B.SCREENSHOTS



The image shows a mobile app interface for a 'Signup' page. The background is a solid light purple color. At the top center, the word 'Signup' is written in a large, black, sans-serif font. Below the title, there are three vertically stacked rectangular input fields with rounded corners and thin purple borders. The first field is labeled 'Name', the second 'Email', and the third 'Password'. Below these fields is a rounded rectangular button with a light purple gradient and the word 'Signup' in a medium-sized, dark purple font. At the bottom, the text 'Already have an account?' is followed by the word 'Login' in a red font, which is a hyperlink.

Figure B.1 Sigup Page

User has to register to the page by giving the required details. After register user has to login with the authorized mail id so that the user can be redirected to the new transaction. User can now able to manage the expense and transaction, set reminder and manage group expenses by invite group member.

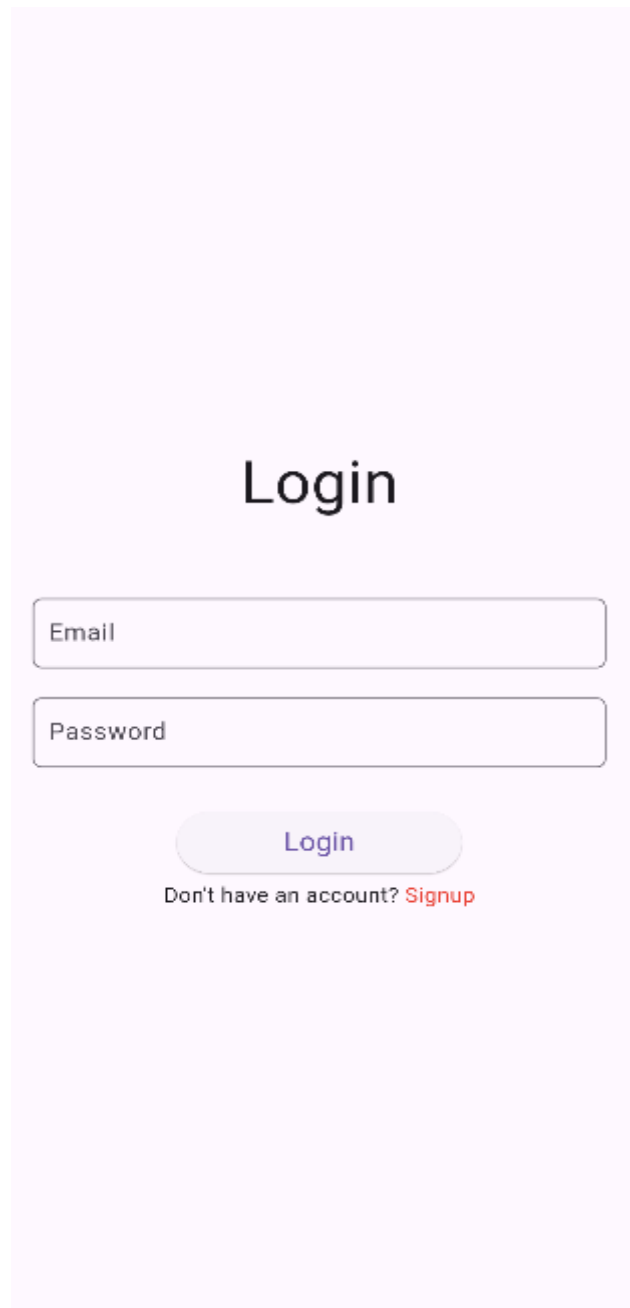
A login form on a light purple background. At the top, the word "Login" is centered in a large, black, sans-serif font. Below it are two rectangular input fields with rounded corners and thin purple borders. The first field is labeled "Email" in a small, grey font. The second field is labeled "Password" in a small, grey font. Below these fields is a rounded rectangular button with a light purple gradient and the word "Login" in a medium-sized, purple font. At the bottom, the text "Don't have an account? Signup" is displayed, where "Signup" is a red hyperlink.

Figure B.2 Login Page

User login form. Login form has mail id and password. The username should be in proper email format. If the fields are not filled, form will not be submitted. Password must contain at least 8 digit or characters. If the form is submitted successfully the page redirected towards home page.



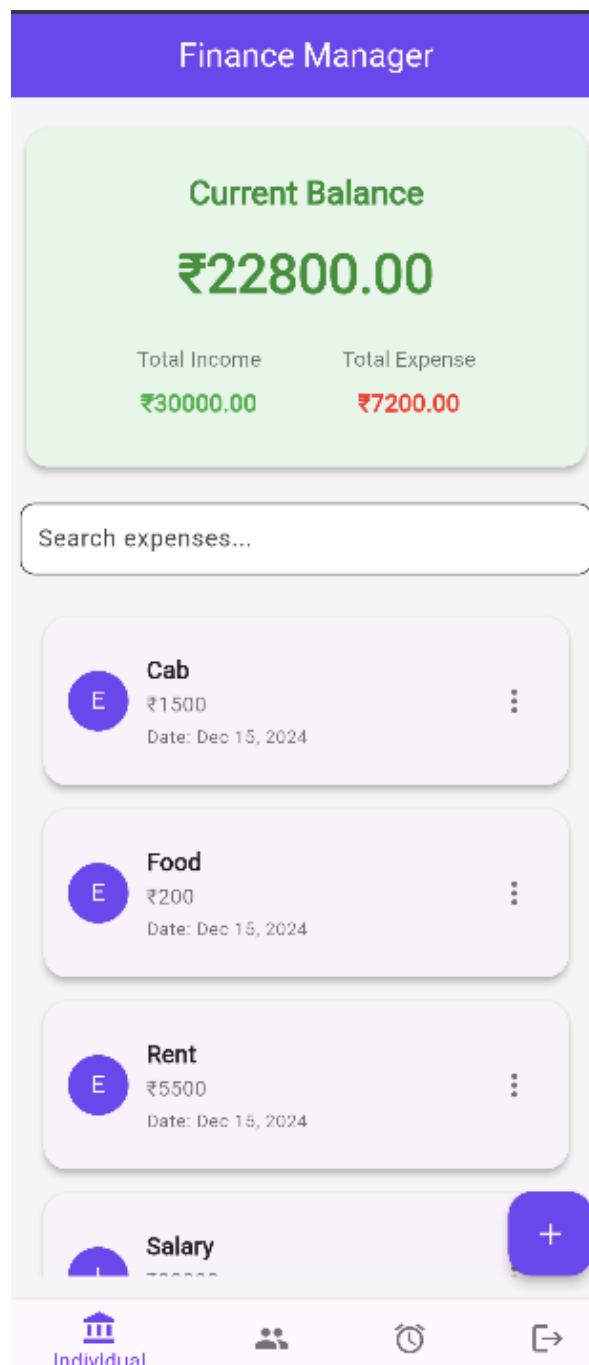


Figure B.3 Individual Expense

In this Individual Expense, users can easily track their income and expenses. The app displays the current balance, total income, and total expenses. Users can search for specific expenses and view detailed information about each transaction, including the date and category.

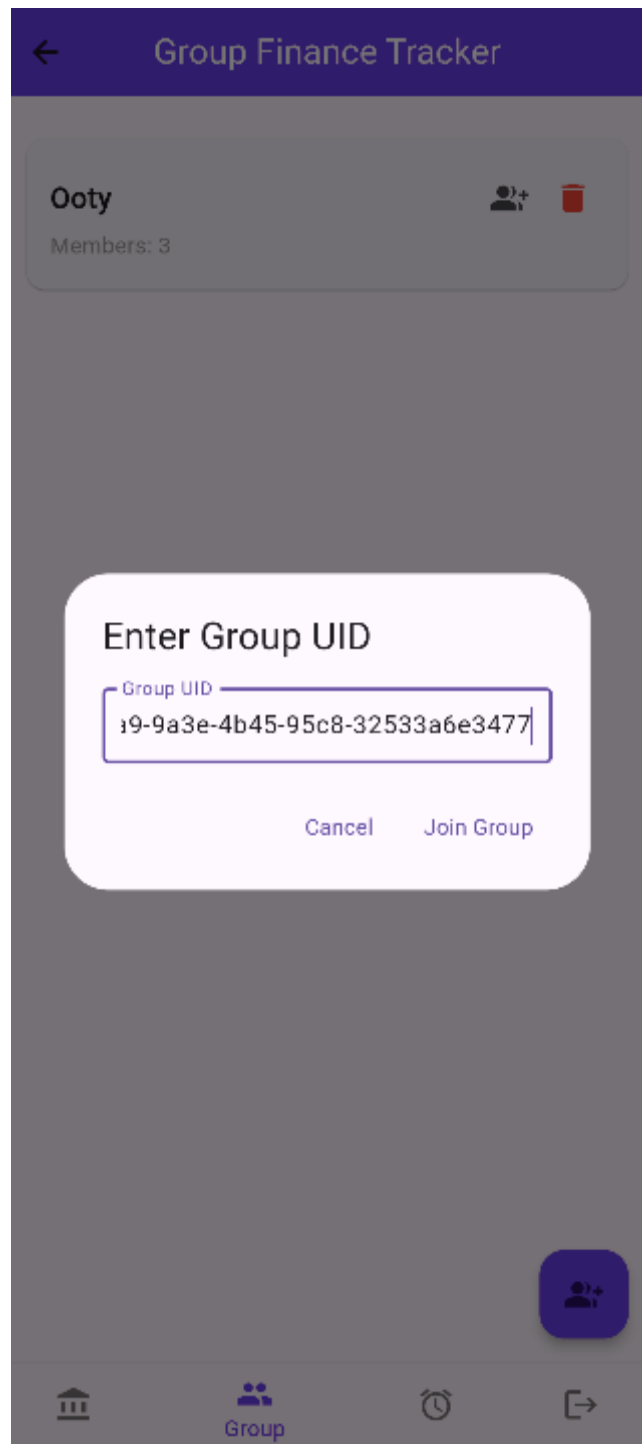


Figure B.4 Invite Group Member

In this Join Group Expense, users can easily join existing groups by entering the unique Group UID. This allows users to collaborate with friends and family on shared expenses and track contributions within specific groups.

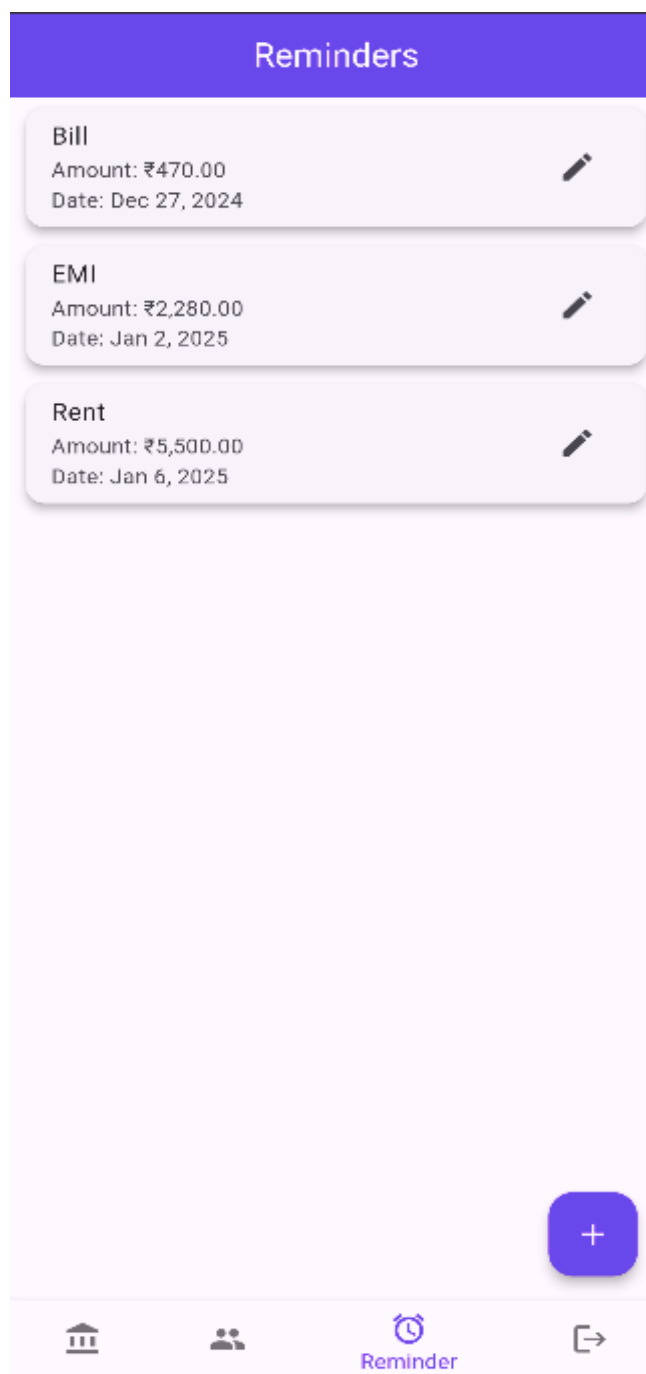


Figure B.5 Set Reminder

In this Reminder, users can easily manage their upcoming expenses. The application displays a list of reminders, including the amount, date, and a quick edit button for each reminder.

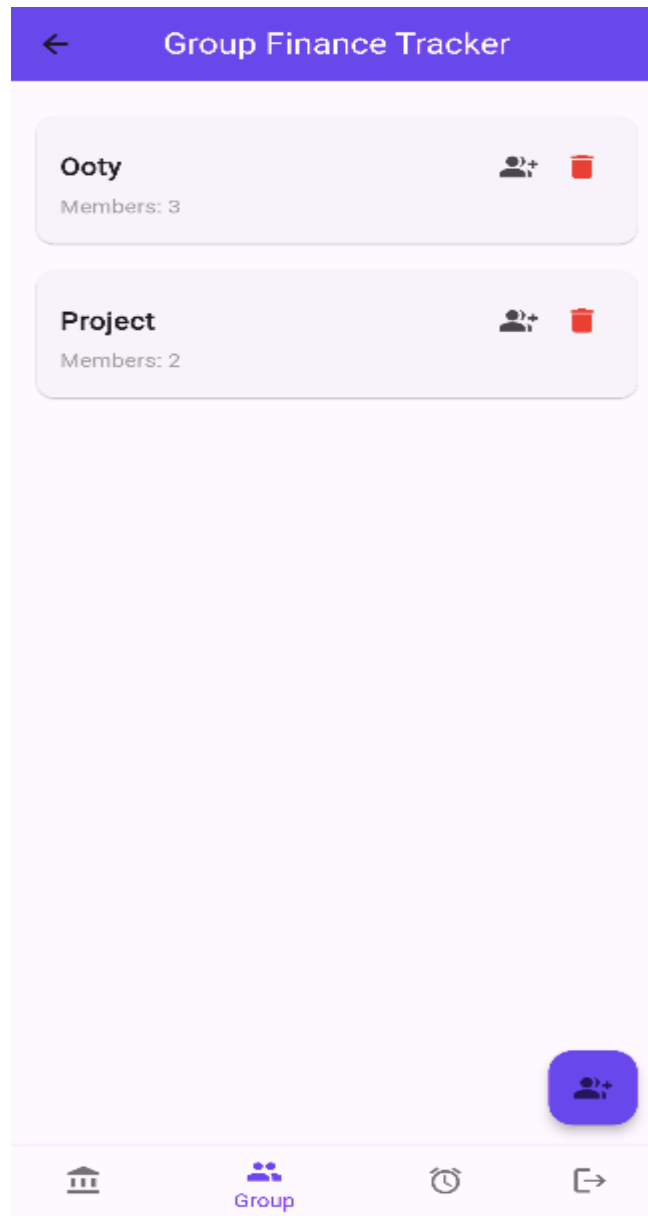


Figure B.6 Group Expense

In this Group Finance, users can easily track shared expenses within groups. The application displays a list of groups, showing the number of members in each group. Users can create new groups, add members, and track expenses and contributions within each group.

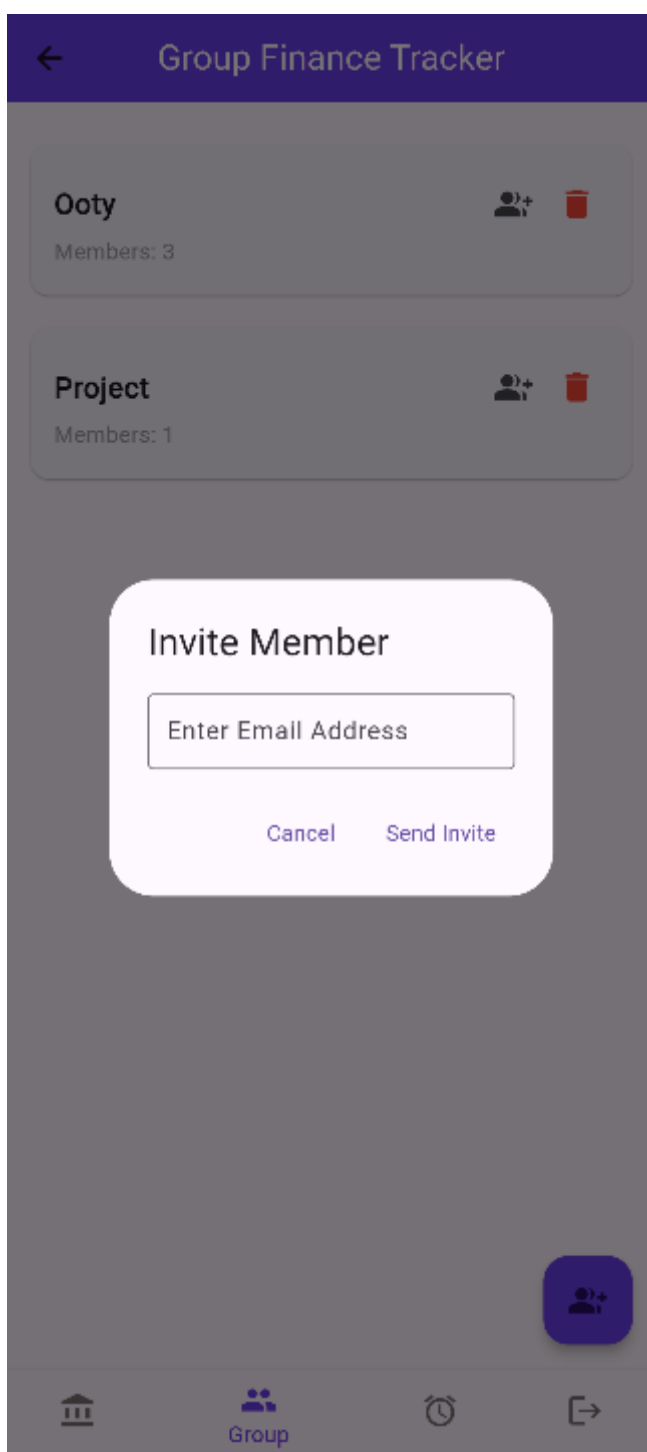


Figure B.7 Join Group

In this Invite Member , users can easily invite new members to their group. By entering the email address of the person they want to invite, they can send an invitation to join the group and start sharing expenses.

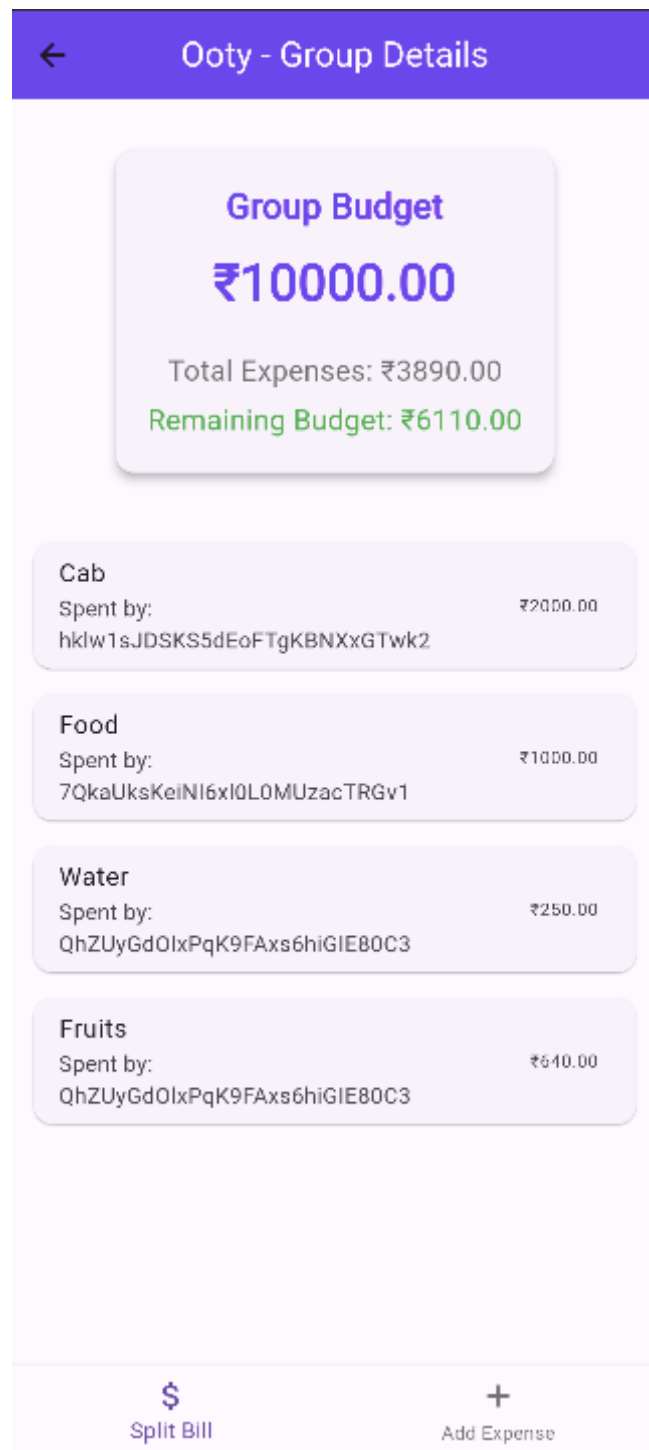


Figure B.8 Group Budget

In this Group Expense Tracker, users can easily track shared expenses within a specific group. The application displays the group's total budget, total expenses, and the remaining budget. It also lists individual expenses, the member who spent it, and the amount.

## REFERENCES

- [1] "Flutter for Beginners: An Introduction to App Development" by Alessandro Biessek (2021, Volume 1, 2nd Edition)
- [2] "Flutter in Action" by Eric Windmill (2020, Volume 1, 1st Edition)
- [3] "Practical Flutter: Improve Your Mobile Development Workflow" by Frank Zammetti (2020, Volume 1, 1st Edition)
- [4] "Firebase Essentials: Mastering Cloud-Driven Development" by Kevin Lee (2019, Volume 1, 2nd Edition)
- [5] "Full Stack Development with Flutter and Firebase" by Matthew Ford (2022, Volume 2, 1st Edition)
- [6] "Building Applications with Flutter and Firebase" by Mark Anderson (2023, Volume 1, 1st Edition)
- [7] <https://flutter.dev/>
- [8] <https://firebase.google.com/>
- [9] <https://www.w3schools.com/> — 'Learn Flutter Basics'