

Money Mate



Session: 2021 - 2025

Submitted by:

Muhammad Yaqoob

2021-CS-118

Supervised by:

Muhammad Laeeq Uz Zaman Khan Niazi

Department of Computer Science
University of Engineering and Technology Lahore
Pakistan

Contents

1	Chapter 1	3
1.1	Introduction	3
1.2	Objectives	3
1.3	Project Features	3
1.3.1	Track Balances	3
1.3.2	Organize Expenses	3
1.3.3	Add Expenses Easily	3
1.3.4	Pay Friends Back	3
1.3.5	The Whole Nine Yards	4
2	Chapter 2	4
2.1	Project Diagrams	4
2.1.1	Low Level Class Diagram	4
2.1.2	Use Case Diagram	5
2.1.3	High Level Class Diagram	6
2.1.4	Sequence Diagram	7
2.1.5	Architecture Diagram	7
3	Chapter 3	8
3.1	Best Practices	8
3.1.1	Comments as Best Practices	8
3.1.2	API Naming Best Practices	8
3.2	Project Management Tool	10
3.3	Project Management	10
3.4	Code Snippets	10
3.5	Front-end	14
3.6	Back-end	15
3.7	Target Audience	15
3.7.1	Housemates and Roommates	16
3.7.2	Travel Enthusiasts	16
3.7.3	Couples and Partners	16
3.7.4	Friend and Social Circles	16
3.7.5	Students and Flatmates	16
4	Chapter 4	17
4.1	Wireframes	17
4.2	List of Packages and Widgets Used	21
4.3	Description of database integrated via APIs	22

4.3.1	GET - Get All Users	22
4.3.2	POST - Sign Up	23
4.3.3	POST - Login	23
4.3.4	GET - Get User by ID	23
4.3.5	PUT - Update User	23
4.3.6	GET - Get All Groups	23
4.3.7	POST - Create Group	23
4.3.8	GET - Get Groups by Token	24
4.3.9	POST - Add Member	24
4.3.10	POST - Add Expense	24
4.3.11	GET - All Expenses	24
4.3.12	POST - Add Friend	24
4.3.13	DELETE - Remove Friend	24
4.3.14	GET - Get All Friends	25
5	Chapter 5	25
5.1	Why Choose Us?	25
5.2	Conclusion	27

1 Chapter 1

1.1 Introduction

Managing shared expenses has never been easier! With our application, you can effortlessly keep track of balances, split expenses, and simplify financial transactions with housemates, on trips, and among friends and family. This introduction provides an overview of the application's core objectives and features, making it the go-to solution for stress-free expense management.

1.2 Objectives

Stress-Free Expense Sharing

Our primary objectives revolve around minimizing stress in various shared financial scenarios. Whether you're living with housemates, embarking on a trip, sharing expenses with your partner, or managing costs with anyone else, our application aims to alleviate the hassles associated with shared financial responsibilities.

1.3 Project Features

1.3.1 Track Balances

Keep an accurate record of shared expenses, balances, and outstanding debts. Our application ensures transparency in financial transactions, allowing users to easily identify who owes what.

1.3.2 Organize Expenses

Efficiently split expenses within any group, whether it's a trip, shared living arrangement, or social gathering. The application adapts to diverse scenarios, providing a flexible and organized approach to expense management.

1.3.3 Add Expenses Easily

Seize the moment and add expenses on the go before details fade away. Our application facilitates quick expense recording, preventing the hassle of trying to remember who made payments.

1.3.4 Pay Friends Back

Settle financial obligations seamlessly by recording cash or online payments within the application. It streamlines the process of reimbursing friends, ensuring clarity and accuracy.

1.3.5 The Whole Nine Yards

This comprehensive feature set covers every aspect of shared expense management:

- **Add Groups and Friends:** Create customized groups and effortlessly add friends to streamline expense tracking.
- **Split Expenses, Record Debts:** Divide expenses among group members, accurately recording individual debts.
- **Equal or Unequal Splits:** Flexibility in splitting expenses equally or unequally, catering to different financial arrangements.
- **Calculate Total Balances:** Instantly calculate total balances within a group, providing a clear financial overview.
- **Suggested Repayments:** Receive smart repayment suggestions based on recorded debts, facilitating fair settlements.
- **Simplify Debts:** Our application simplifies the complexity of debts, ensuring clarity and ease of understanding.
- **Offline Mode:** Enjoy uninterrupted access to essential features even without an internet connection.
- **Cloud Sync:** Safeguard your data with cloud synchronization, ensuring seamless access across multiple devices.
- **Categorize Expenses:** Organize expenses systematically by categorizing them for better financial management.
- **Expense Search:** Easily locate specific expenses through a powerful search feature, enhancing overall usability.

2 Chapter 2

2.1 Project Diagrams

2.1.1 Low Level Class Diagram

This class diagram provides a low-level overview of the primary entities and their relationships within the Money Mate system.

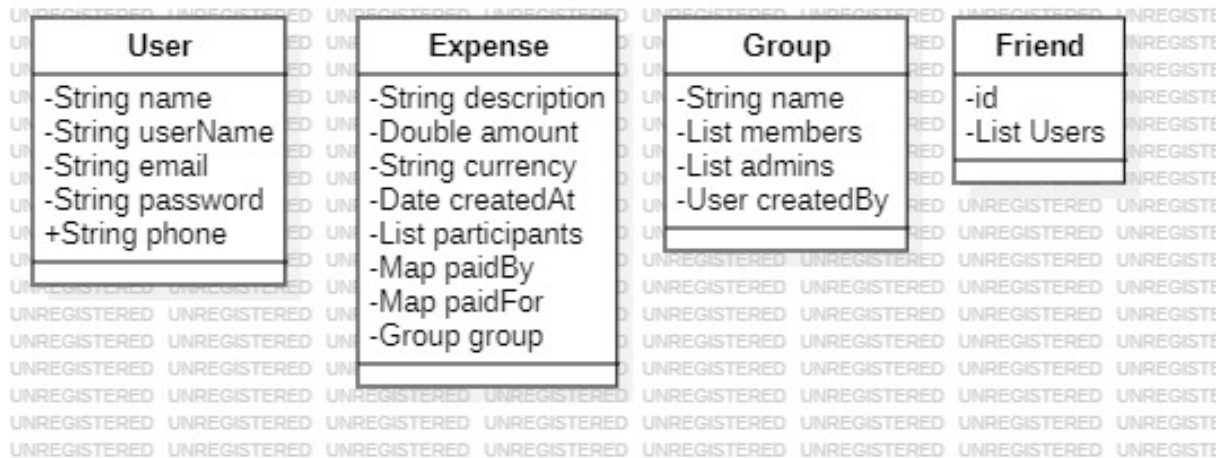


FIGURE 1: Class Diagram of the Project

2.1.2 Use Case Diagram

This use case diagram provides a high-level perspective on how users can navigate and engage with different aspects of the application.

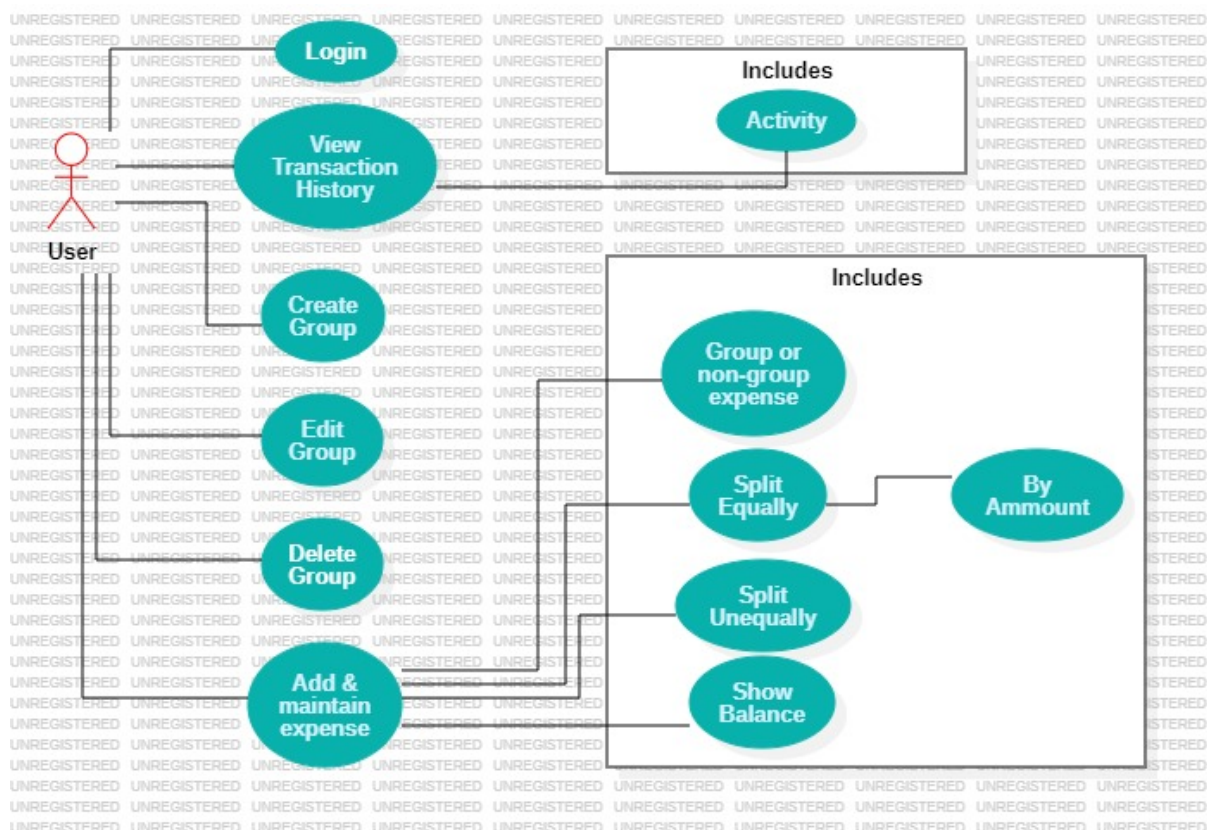


FIGURE 2: Use Case Diagram of the Project

2.1.3 High Level Class Diagram

In this high-level class diagram, the "User" class has one-to-many relationships with "Expenses.Participants," "Group.Members," and "Group.Admins" classes. The "Expenses.Participants" table links users to their expenses. The "Group.Members" table shows user-group associations, and "Group.Admins" identifies users with administrative roles in groups, allowing them to delete groups. This diagram provides a concise overview of the key associations in the system.

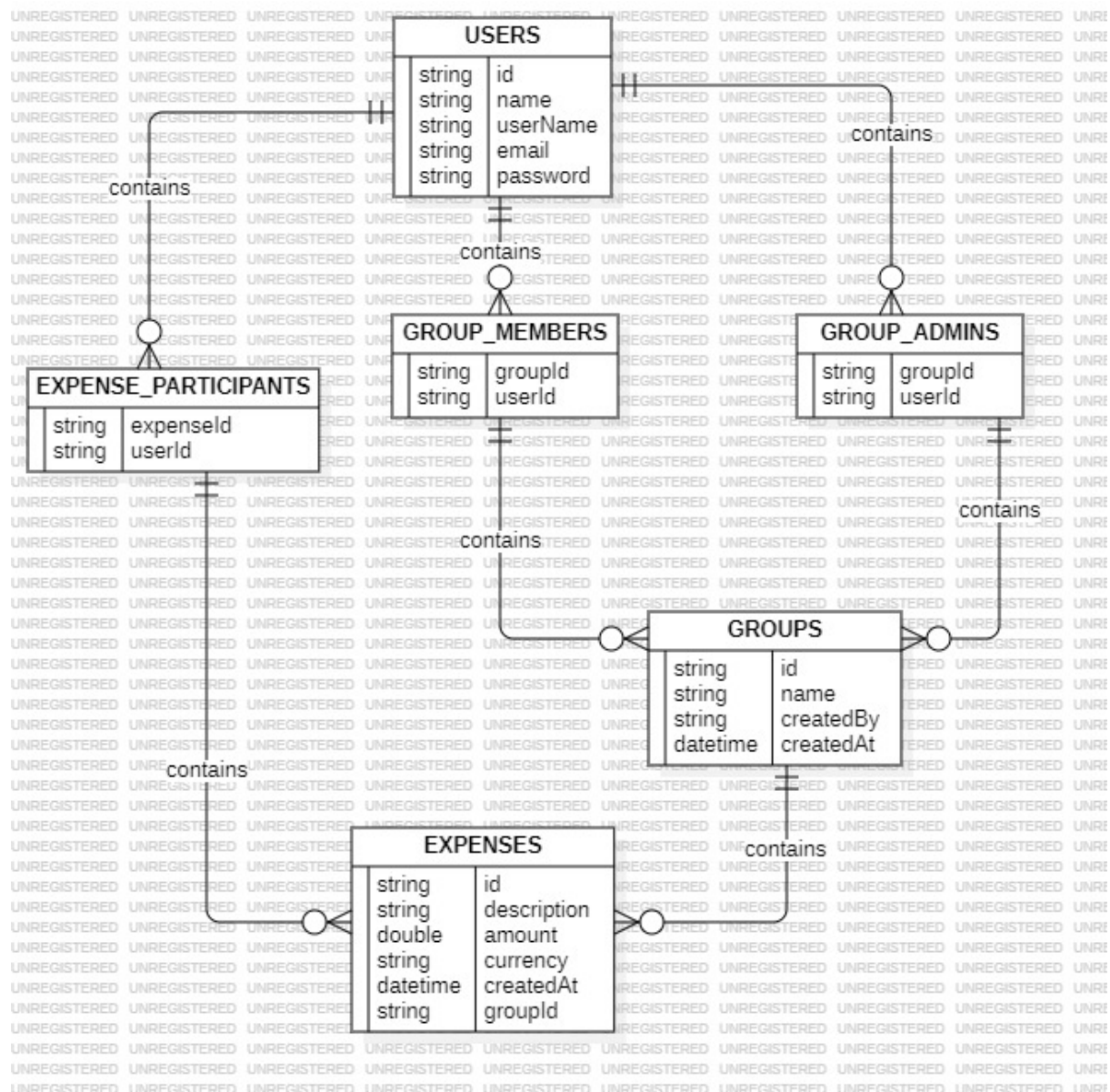


FIGURE 3: Class Diagram of the Project

2.1.4 Sequence Diagram

This sequence diagram highlights the sequential flow of interactions between the user and the system components when adding an expense, emphasizing the collaboration between classes to achieve this functionality.

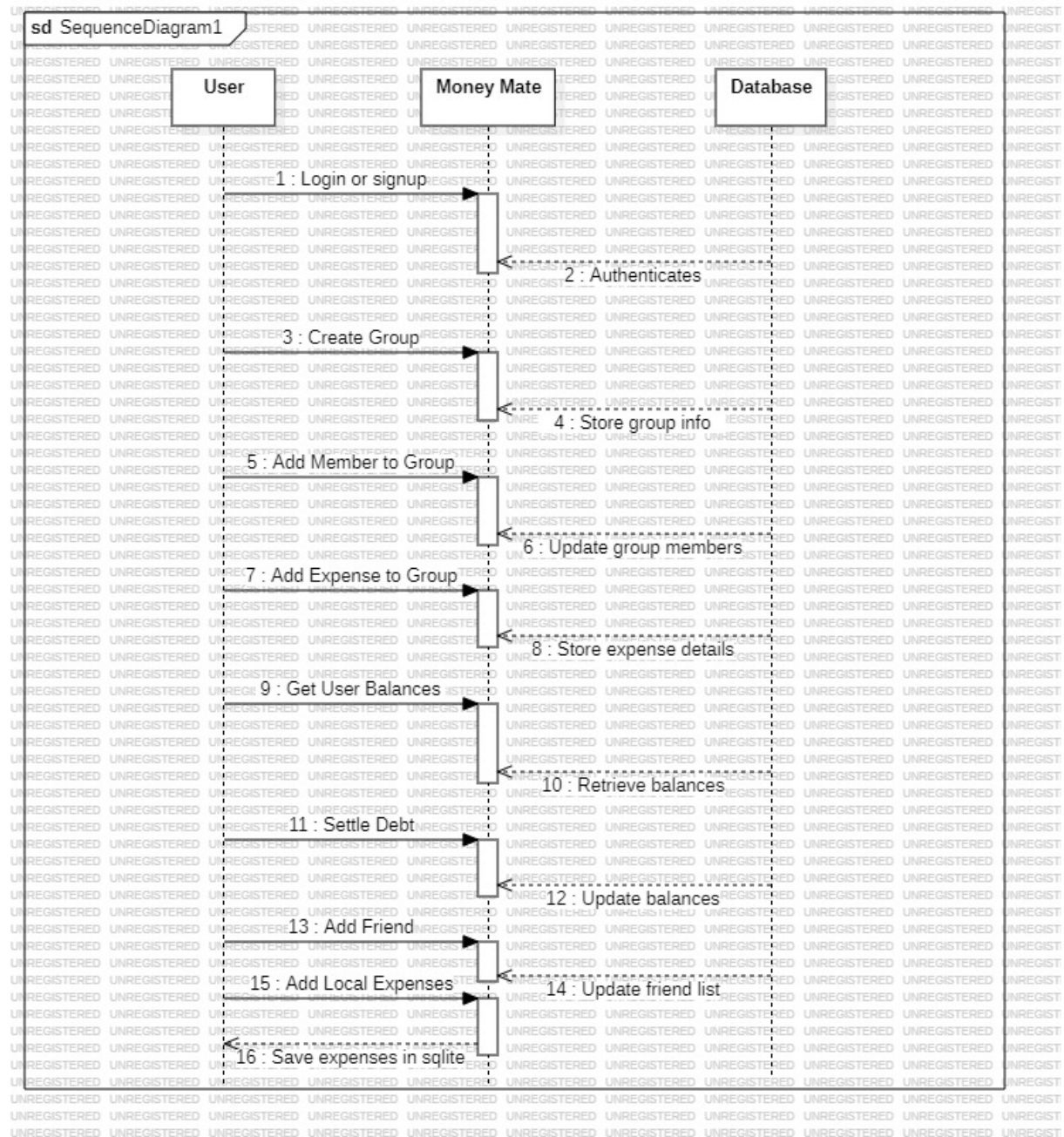


FIGURE 4: Sequence Diagram of the Project

2.1.5 Architecture Diagram

In this architectural diagram, user-related services are centralized within the user database, managing aspects such as user profiles and authentication. Similarly, group

services, responsible for tasks associated with groups, are stored in the group database. Additionally, the expense services, primarily focusing on the creation and management of expenses, are added to and stored in the expense database. This separation allows for a modular and scalable architecture, ensuring efficient organization and retrieval of data based on the distinct functionalities of user, group, and expense services.

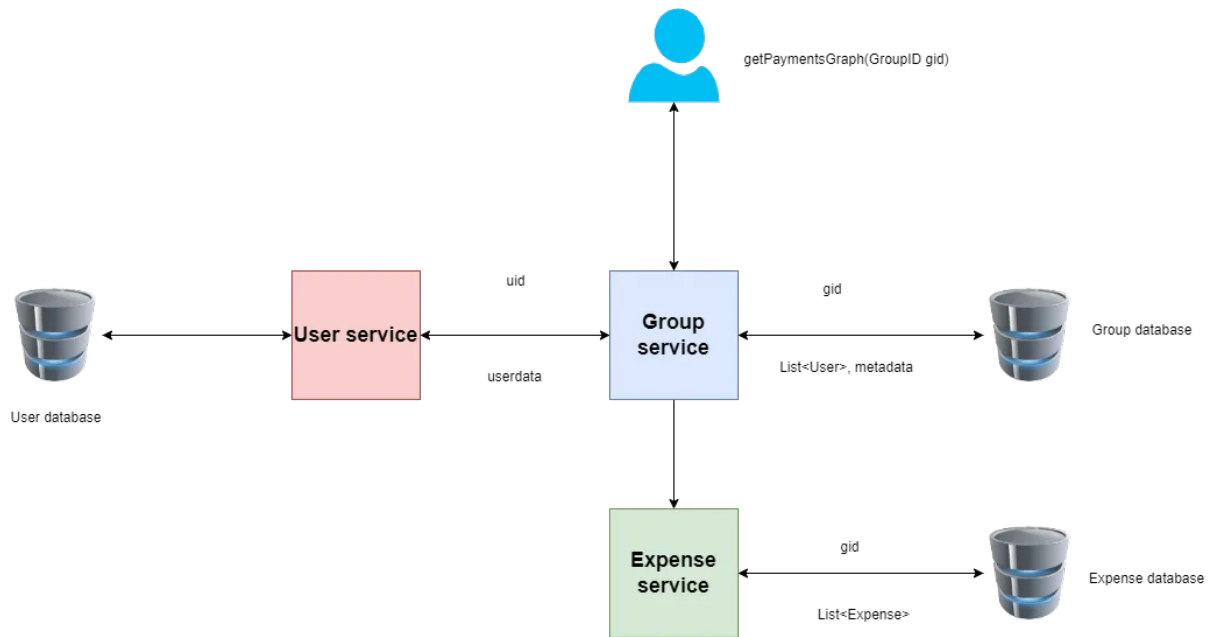


FIGURE 5: Architecture Diagram of the Project

3 Chapter 3

3.1 Best Practices

3.1.1 Comments as Best Practices

The code incorporates best practices in comments, offering clarity by explaining complex algorithms. These comments enhance code readability without introducing redundancy, aligning with industry standards for maintainable and comprehensible code. Screen Short in Figure 6.

3.1.2 API Naming Best Practices

I have adhered to best practices in API naming, ensuring consistent and meaningful endpoint names for APIs with different request types. Screen Short in Figure 7.

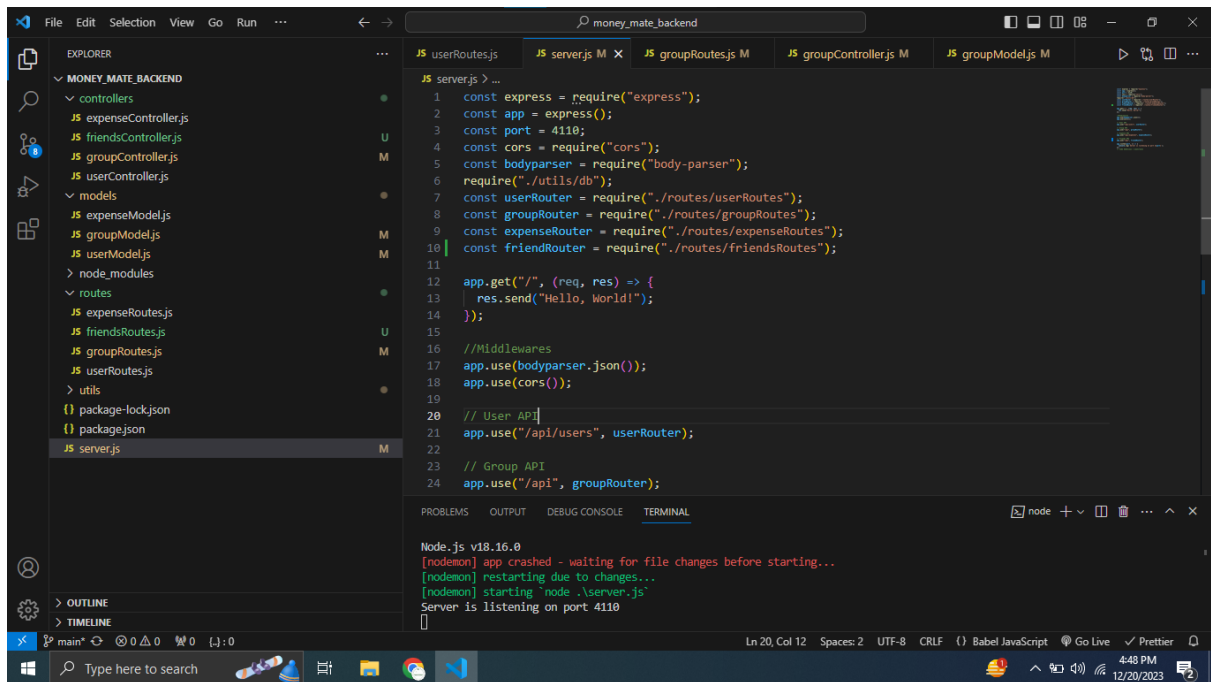


FIGURE 6: Comments as Best Practices

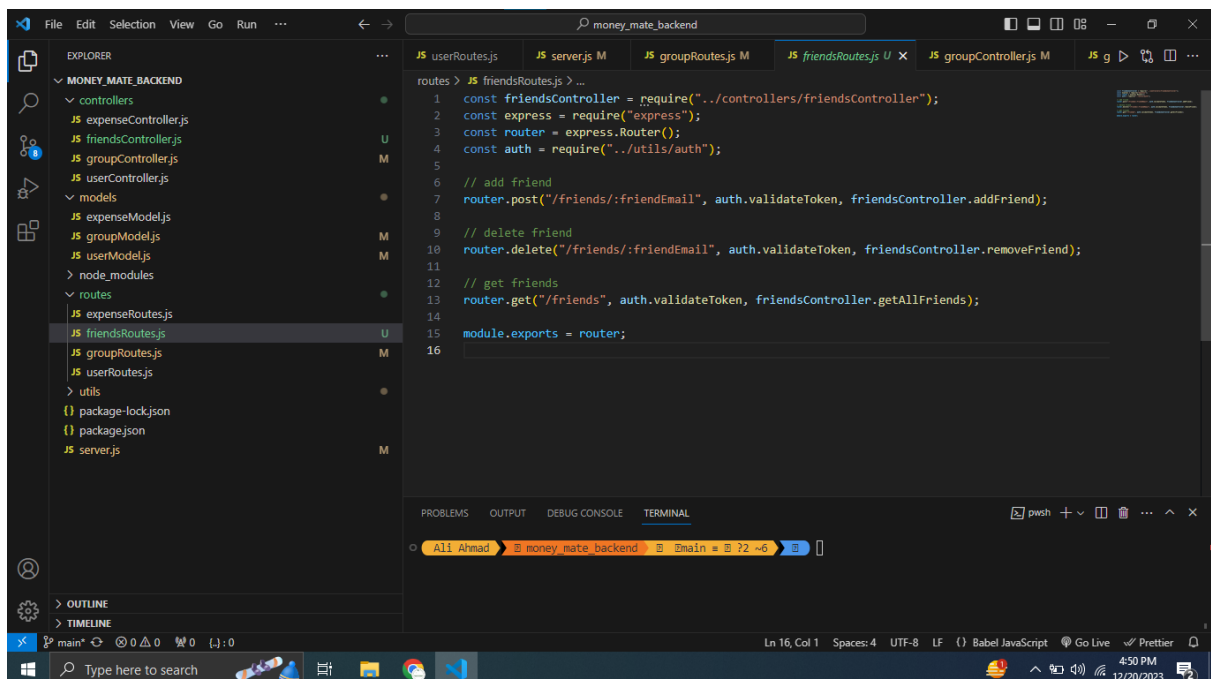


FIGURE 7: API Naming Best Practices

3.2 Project Management Tool

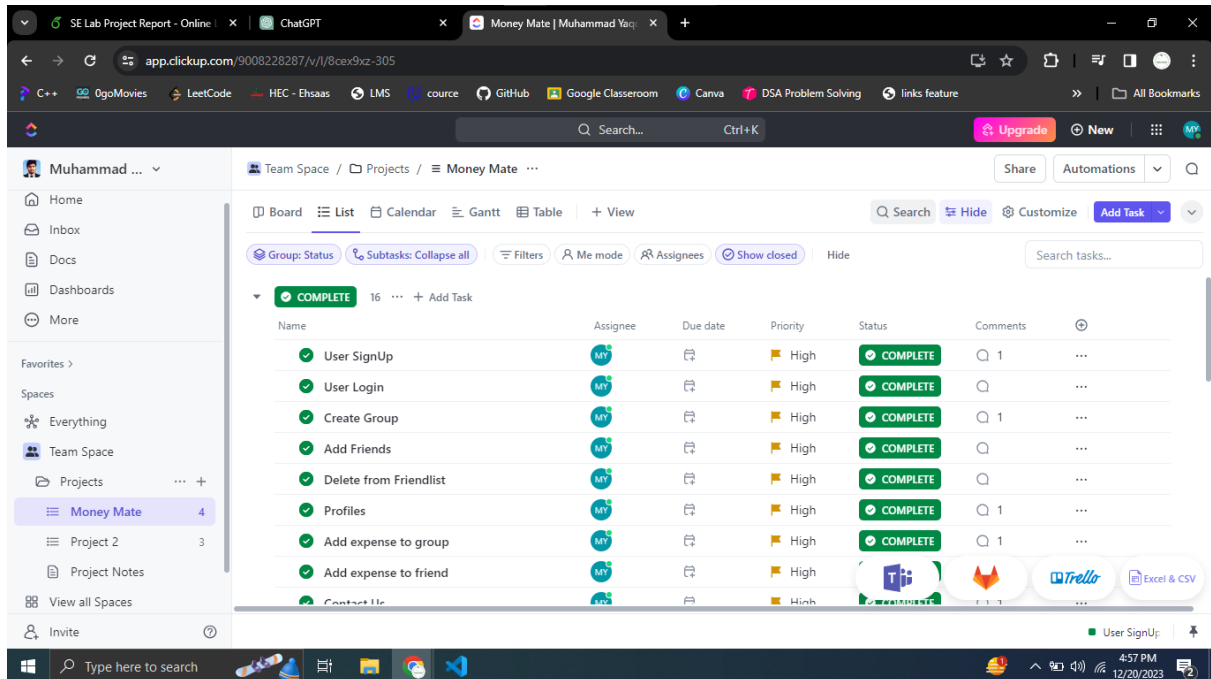


FIGURE 8: Utilizing ClickUp as a Project Management Tool

3.3 Project Management

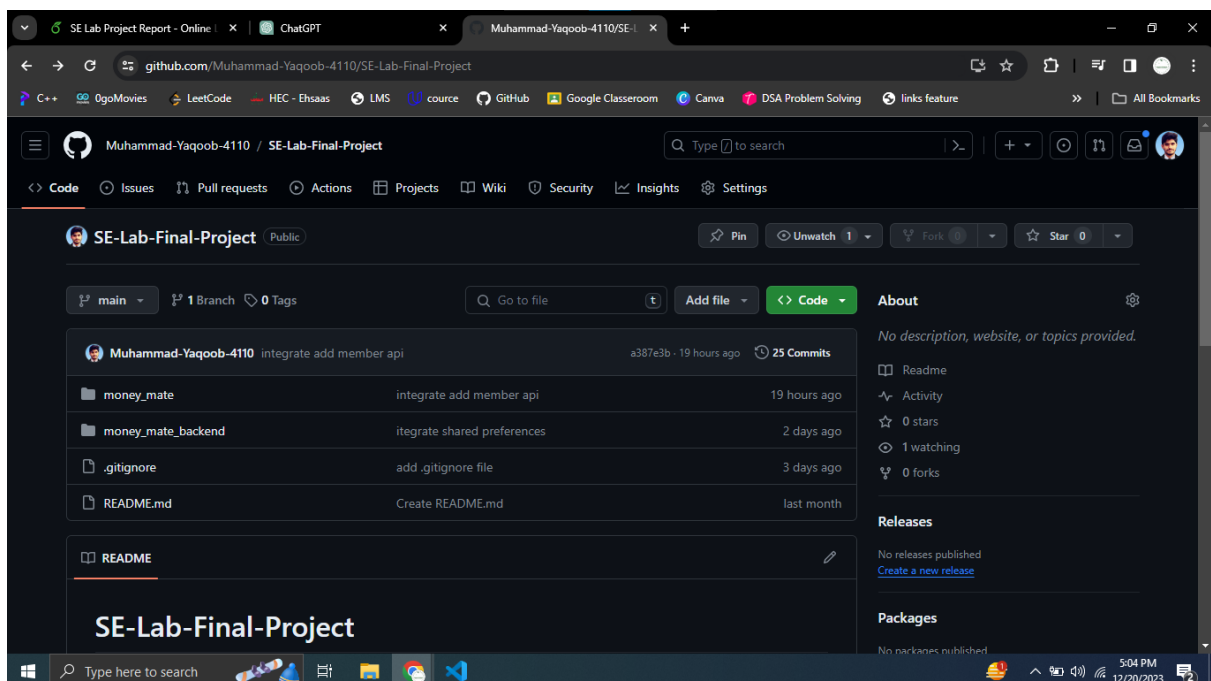


FIGURE 9: Leveraging GitHub as an Integrated Project Management

3.4 Code Snippets

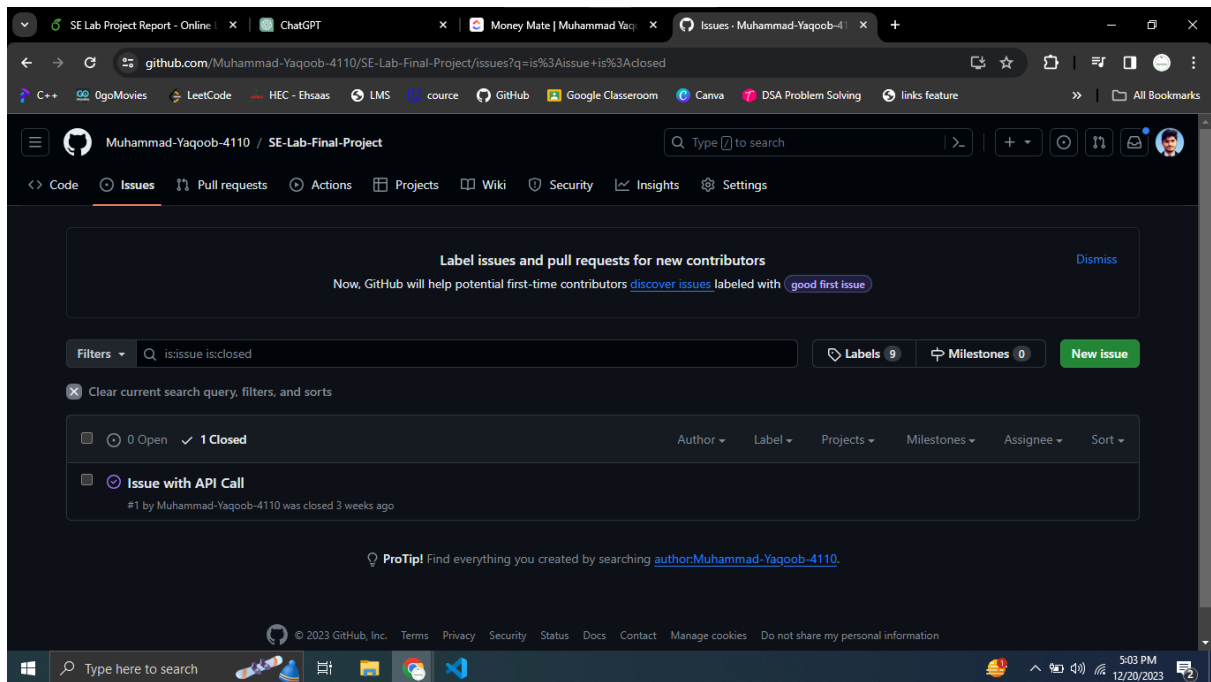


FIGURE 10: GitHub Issues - Navigating Seamless Collaboration

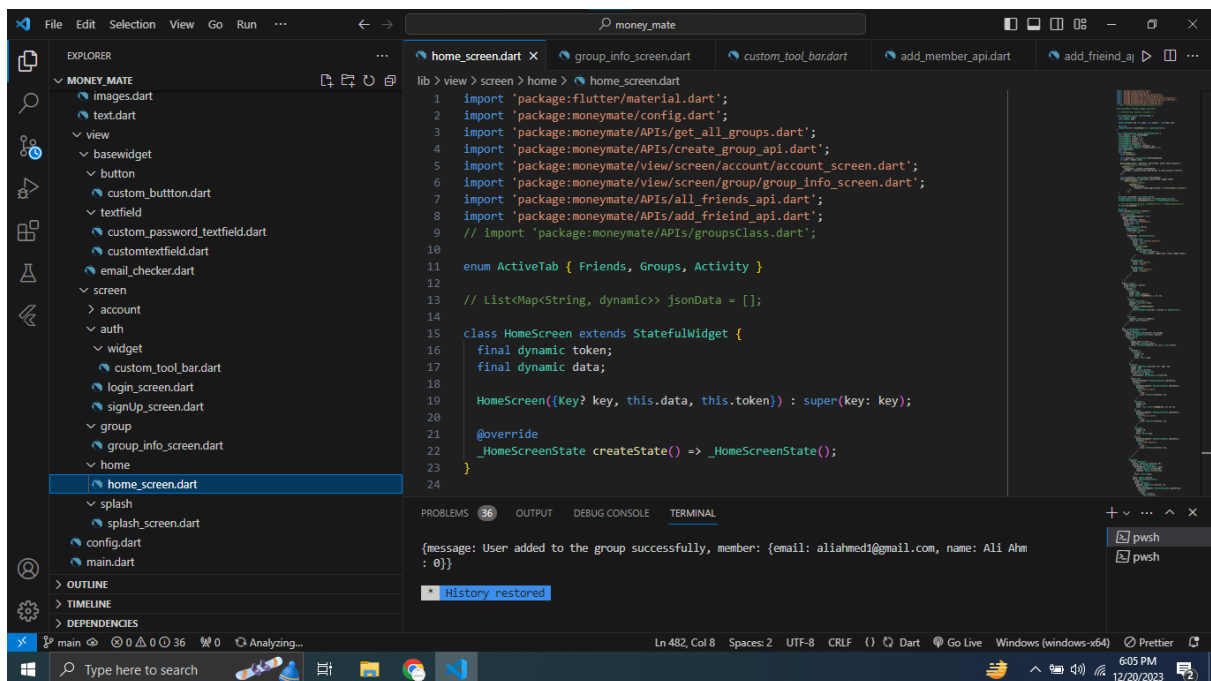


FIGURE 11: Exemplary Code Snippet

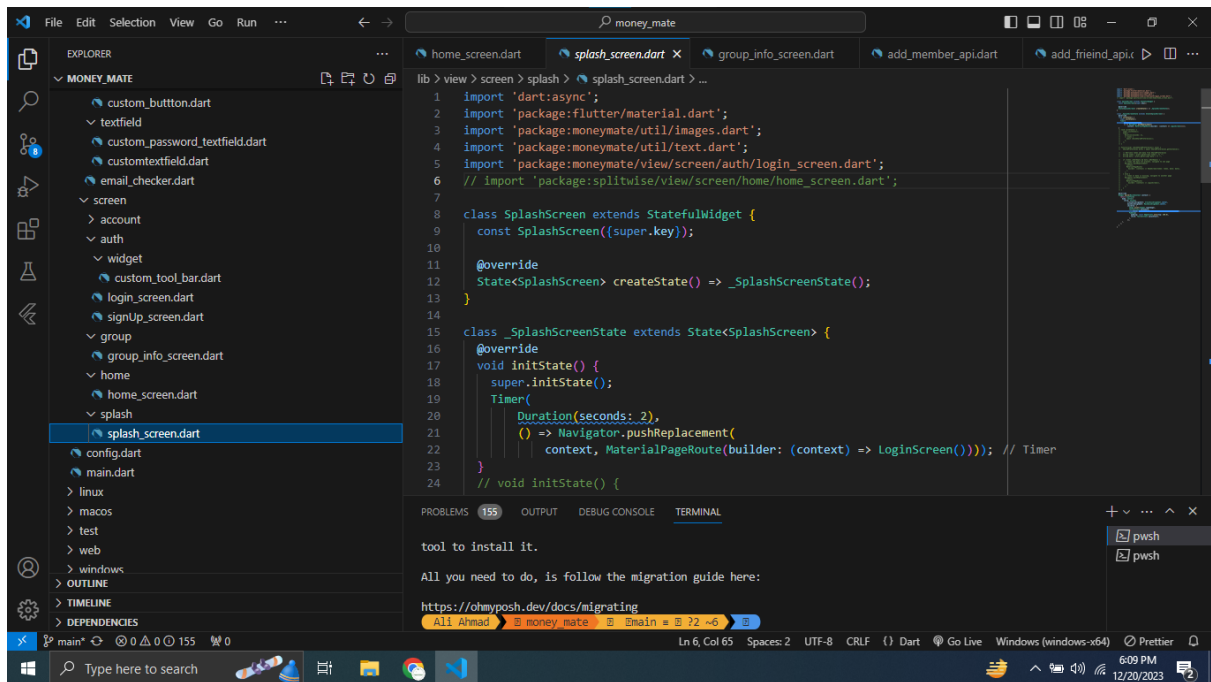


FIGURE 12: Exemplary Code Snippet

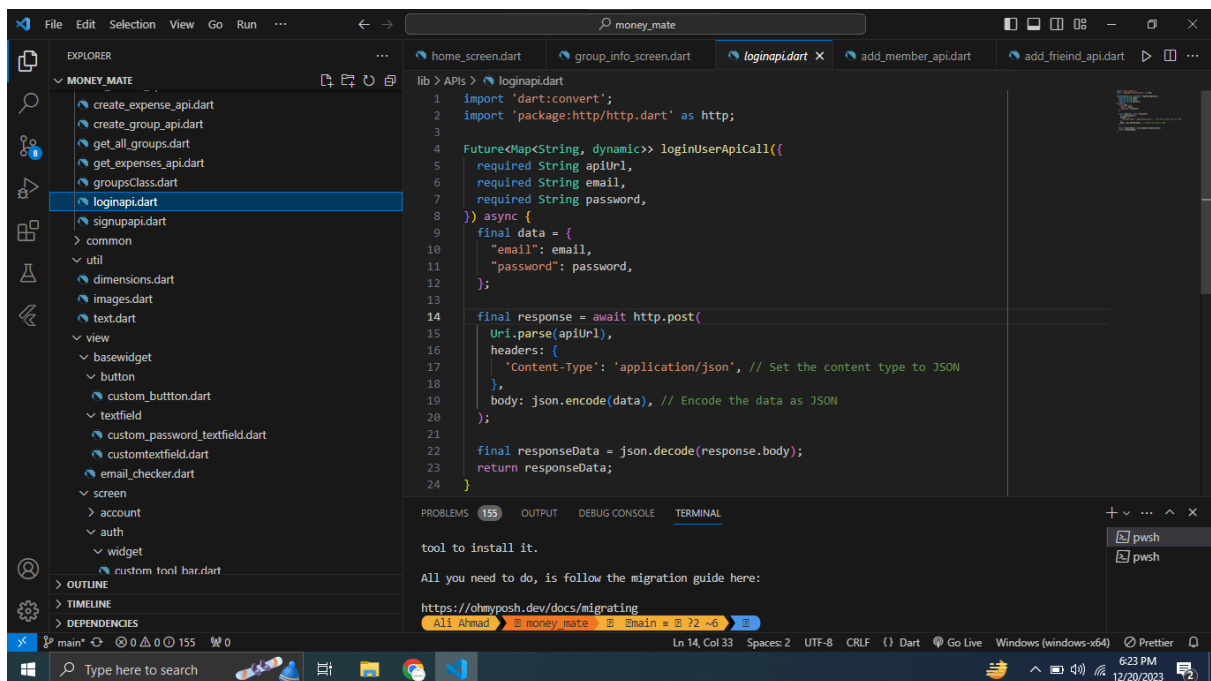
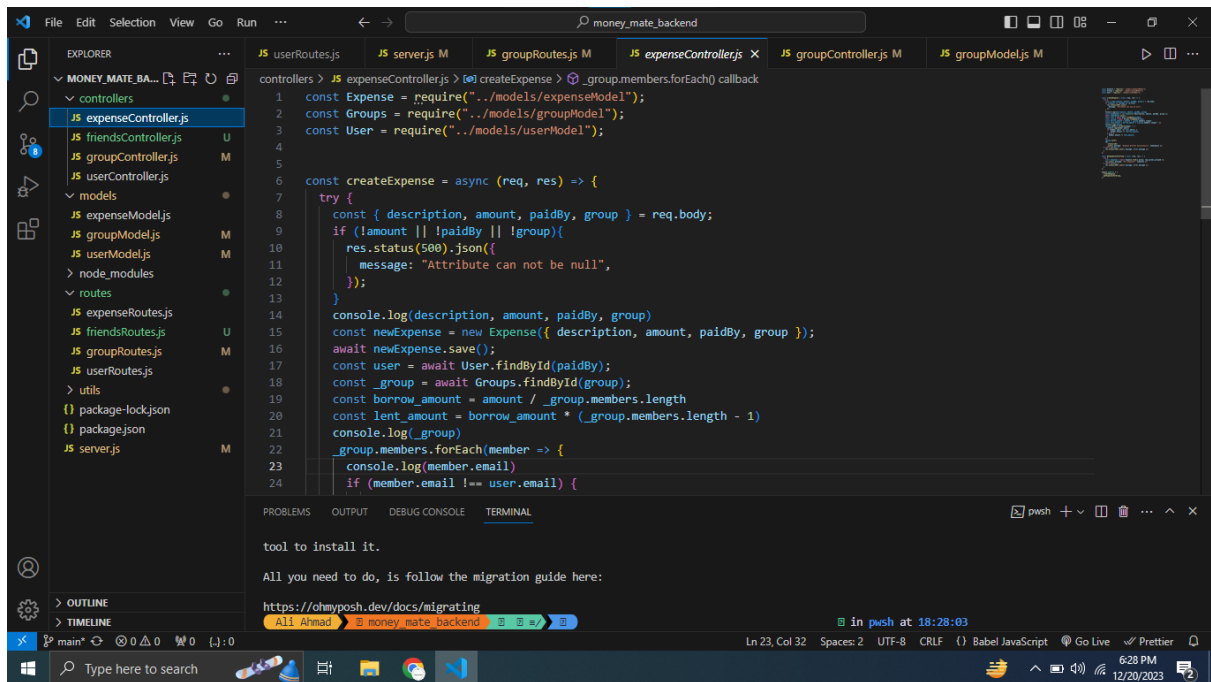
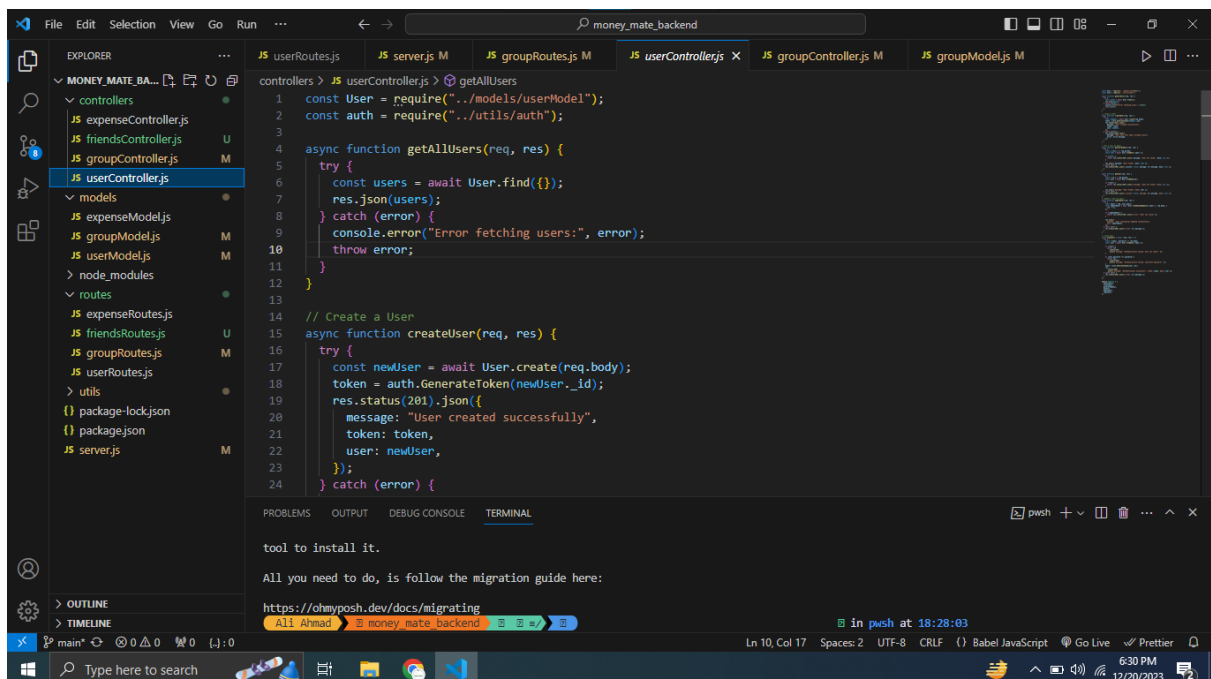


FIGURE 13: Exemplary Code Snippet



```
1 const Expense = require("../models/expenseModel");
2 const Groups = require("../models/groupModel");
3 const User = require("../models/userModel");
4
5
6 const createExpense = async (req, res) => {
7   try {
8     const { description, amount, paidBy, group } = req.body;
9     if (!amount || !paidBy || !group) {
10       res.status(500).json({
11         message: "Attribute can not be null",
12       });
13     }
14     console.log(description, amount, paidBy, group)
15     const newExpense = new Expense({ description, amount, paidBy, group });
16     await newExpense.save();
17     const user = await User.findById(paidBy);
18     const _group = await Groups.findById(group);
19     const borrow_amount = amount / _group.members.length
20     const lent_amount = borrow_amount * (_group.members.length - 1)
21     console.log(_group)
22     _group.members.forEach(member => {
23       console.log(member.email)
24       if (member.email !== user.email) {
```

FIGURE 14: Exemplary Code Snippet



```
1 const User = require("../models/userModel");
2 const auth = require("../utils/auth");
3
4 async function getAllUsers(req, res) {
5   try {
6     const users = await User.find({});
7     res.json(users);
8   } catch (error) {
9     console.error("Error fetching users:", error);
10    throw error;
11  }
12 }
13
14 // Create a User
15 async function createUser(req, res) {
16   try {
17     const newUser = await User.create(req.body);
18     token = auth.GenerateToken(newUser._id);
19     res.status(201).json({
20       message: "User created successfully",
21       token: token,
22       user: newUser,
23     });
24   } catch (error) {
```

FIGURE 15: Exemplary Code Snippet

3.5 Front-end

```
lib/
├── alerts/
│   └── alerts.dart
├── APIs/
│   ├── loginapi.dart
│   ├── signupapi.dart
│   ├── add_frieind_api.dart
│   ├── add_member_api.dart
│   ├── all_friends_api.dart
│   ├── create_expense_api.dart
│   ├── create_group_api.dart
│   ├── get_all_groups.dart
│   ├── get_expenses_api.dart
│   └── groupsClass.dart
├── common/
│   └── alert_widget.dart
├── util/
│   ├── dimensions.dart
│   ├── images.dart
│   └── text.dart
├── local_expenses/
│   ├── database_helper.dart
│   └── expense_model.dart
└── view/
    ├── basewidget/
    │   ├── button/
    │   │   └── custom_butttton.dart
    │   ├── textfield/
    │   │   ├── custom_password_textfield.dart
    │   │   └── customtextfield.dart
    │   └── email_checker.dart
    └── screen/
        ├── account/
        │   └── account_screen.dart
        ├── auth/
        │   └── widget/
        │       └── custom_tool_bar.dart
```

```
├── login_screen.dart
├── signUp_screen.dart
├── group/
│   ├── group_info_screen.dart
├── home/
│   ├── home_screen.dart
├── splash/
│   ├── splash_screen.dart
├── config.dart
└── main.dart
```

3.6 Back-end

```
backend/
├── controllers/
│   ├── expenseController.js
│   ├── friendsController.js
│   ├── groupController.js
│   └── userController.js
├── models/
│   ├── expenseModel.js
│   ├── groupModel.js
│   └── userModel.js
├── routes/
│   ├── expenseRoutes.js
│   ├── friendsRoutes.js
│   ├── groupRoutes.js
│   └── userRoutes.js
├── utils/
│   ├── auth.js
│   ├── config.js
│   └── db.js
├── package.json
└── server.js
```

3.7 Target Audience

Our application is designed to cater to a diverse range of individuals and groups who navigate shared financial responsibilities in various contexts. The target audience includes, but is not limited to:

3.7.1 Housemates and Roommates

For individuals living together, our application simplifies the tracking of shared household expenses. From rent and utilities to groceries and shared services, our features ensure transparency and fairness in managing collective financial commitments.

3.7.2 Travel Enthusiasts

Ideal for travelers embarking on journeys with friends or family, our application facilitates the effortless splitting of travel expenses. Whether it's accommodation, meals, or group activities, users can easily manage and settle shared costs incurred during their adventures.

3.7.3 Couples and Partners

In shared financial partnerships, such as couples managing joint expenses or individuals sharing costs within a relationship, our application provides a user-friendly platform for recording, tracking, and settling financial transactions. It promotes clarity and understanding in monetary dealings between partners.

3.7.4 Friend and Social Circles

For friends organizing events, outings, or social gatherings, our application offers a streamlined solution for managing shared expenses. From restaurant bills to concert tickets, users can efficiently divide costs and keep track of who owes what, ensuring smooth financial interactions within their social circles.

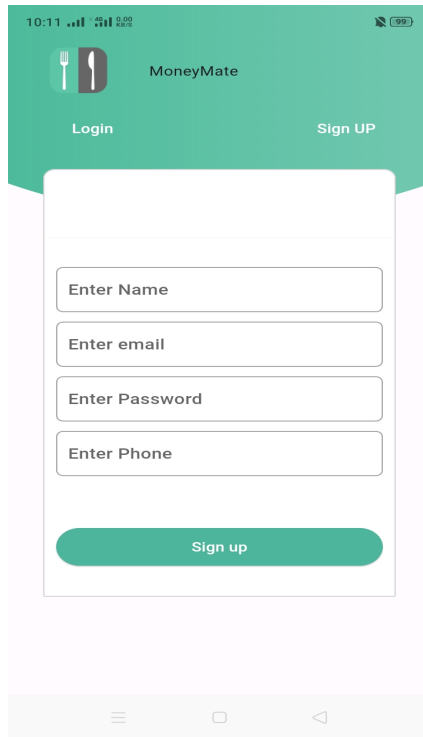
3.7.5 Students and Flatmates

Especially useful for students and flatmates sharing accommodation and resources, our application simplifies the complexities of shared living expenses. Users can easily manage rent, bills, and other joint expenditures, fostering a hassle-free communal living experience.

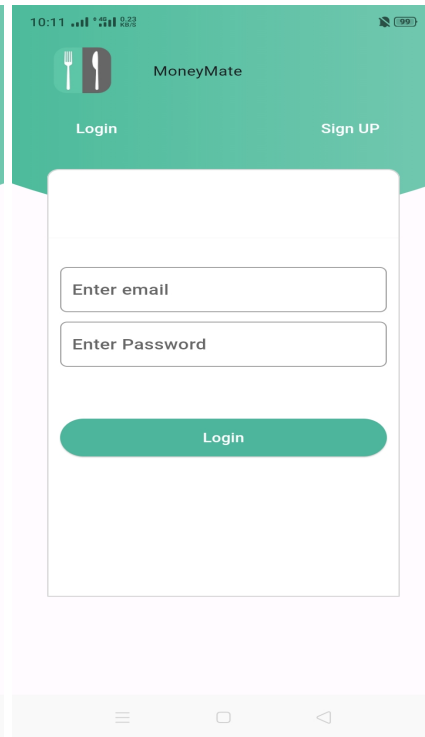
Our target audience encompasses anyone seeking a reliable and intuitive tool for shared expense management. Whether you're part of a household, travel group, couple, social circle, or shared living arrangement, our application is tailored to meet your diverse financial needs.

4 Chapter 4

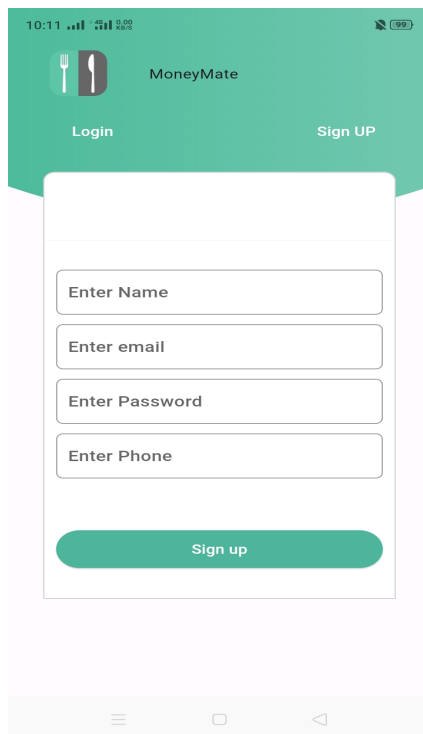
4.1 Wireframes



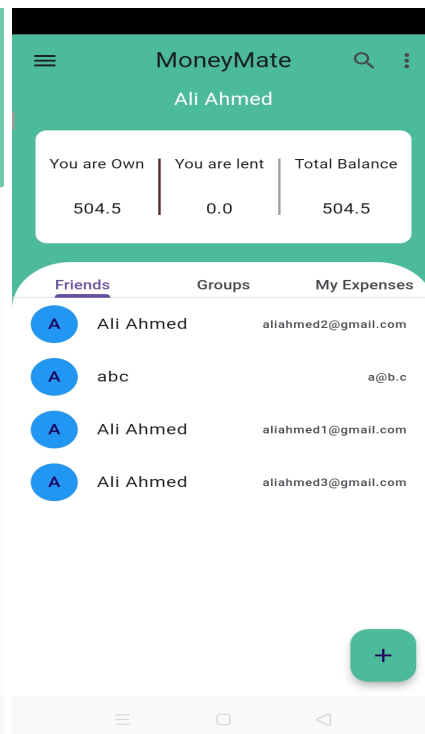
(a) Signup Screen



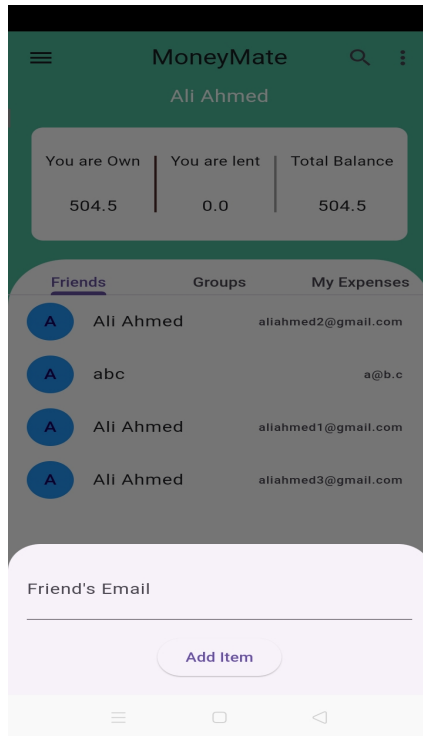
(b) Login Screen



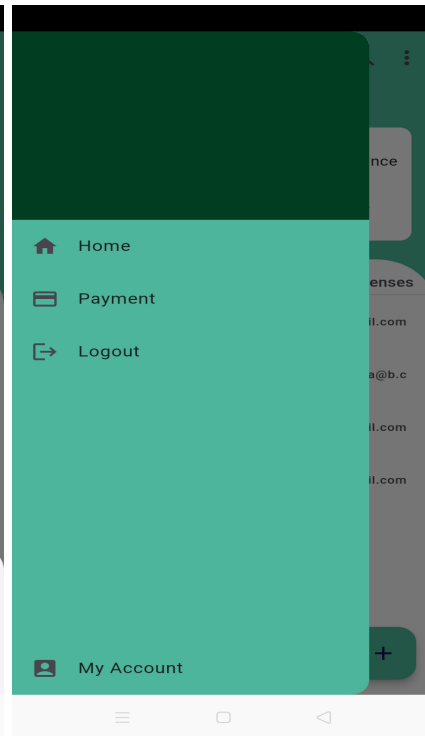
(c) SignUp Screen



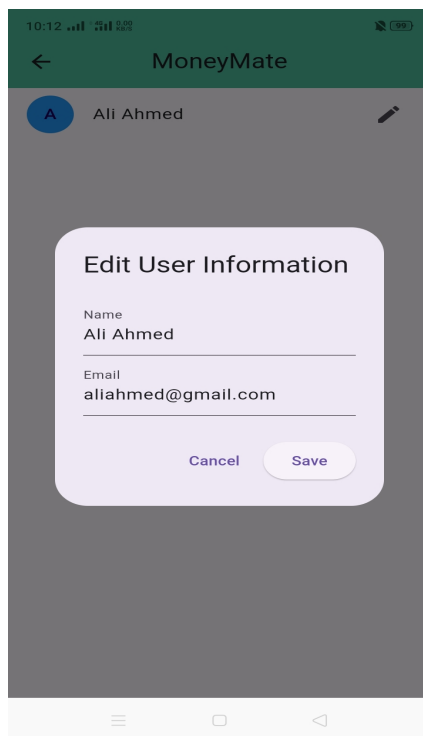
(d) Friends Screen



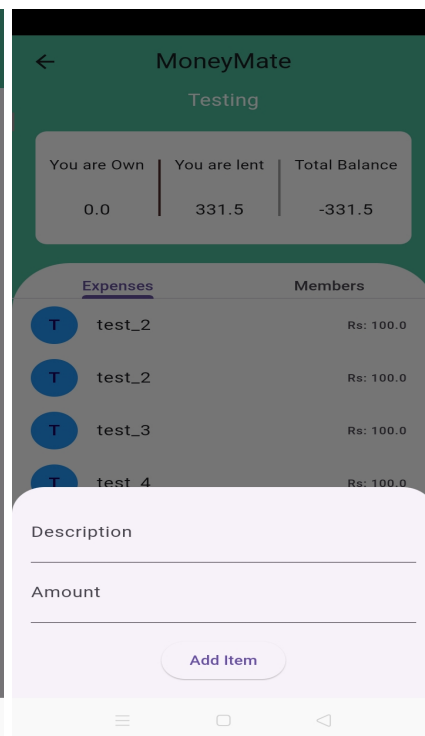
(e) Add Friend Screen



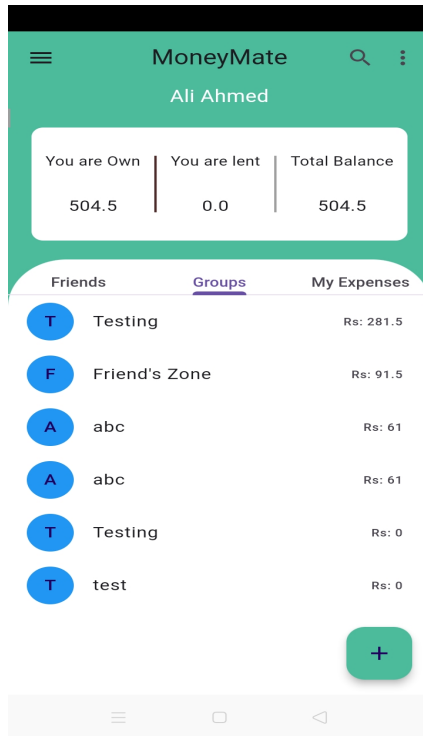
(f) Home Screen Drawer



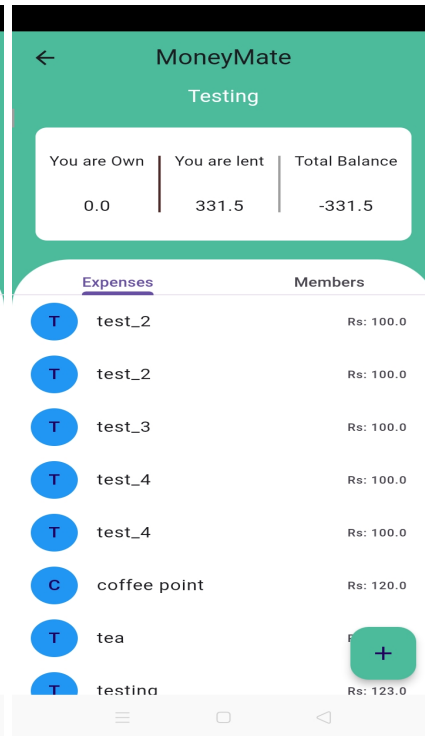
(g) Account Screen



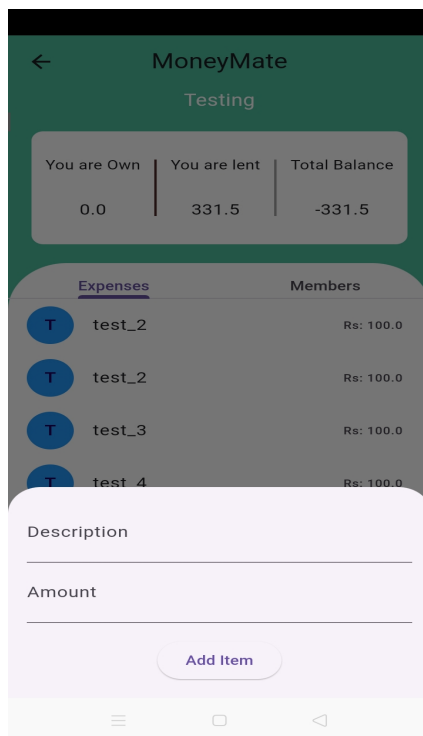
(h) Add Expenses Screen



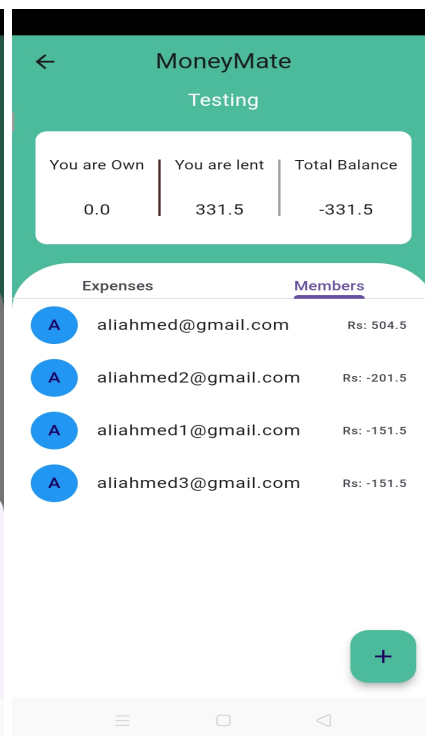
(i) Home Group Screen



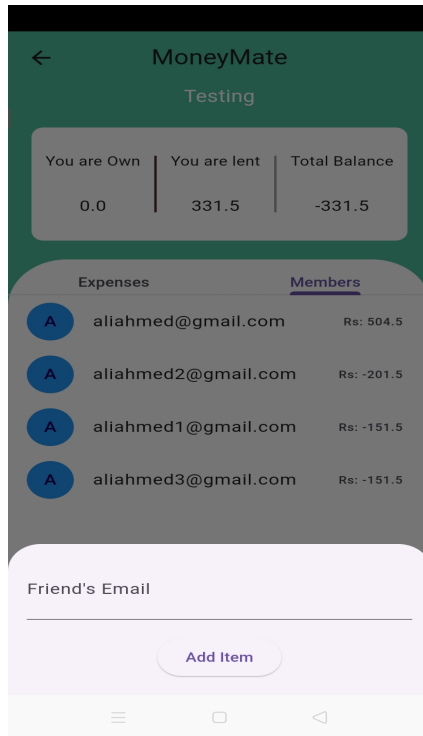
(j) Group Expense Screen



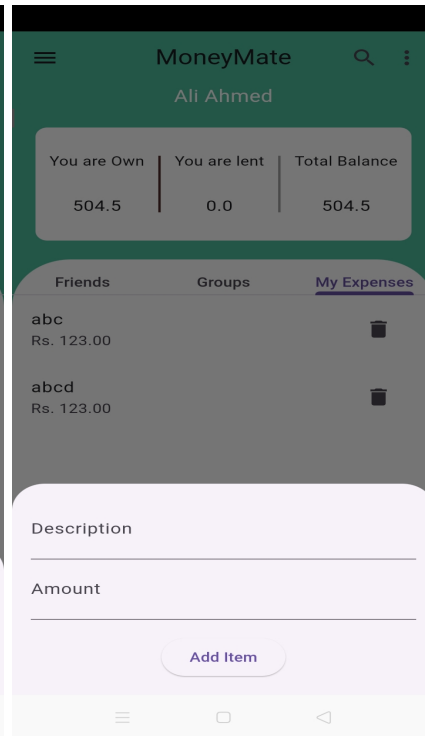
(k) Add Expense Screen



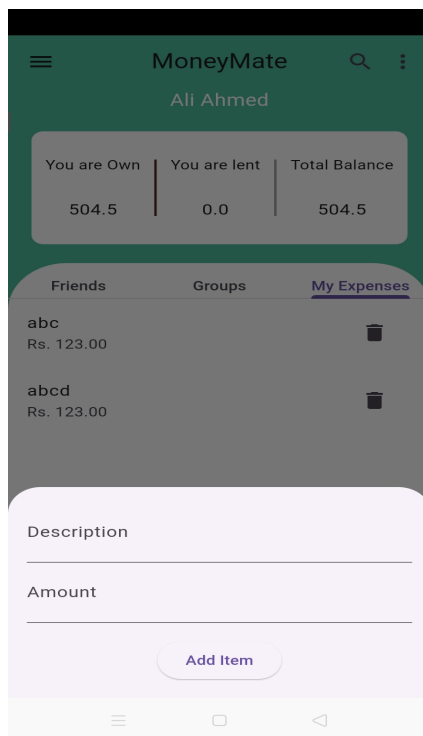
(l) Group Members Screen



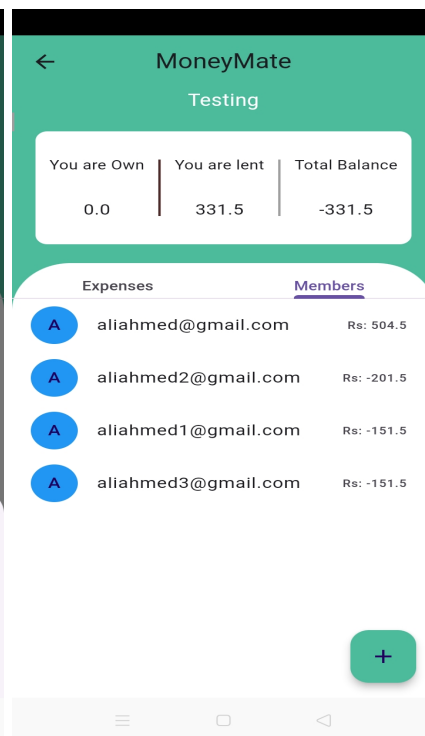
(m) Add Members Screen



(n) My Expenses Screen



(o) Add Local Expenses Screen



(p) Group Members Screen

4.2 List of Packages and Widgets Used

The MoneyMate project utilizes several Flutter packages to enhance its functionality. Here is a comprehensive list of packages along with brief descriptions:

1. **http** (^1.1.2): Facilitates HTTP requests, crucial for handling API communication within the application.
2. **shared_preferences** (^2.0.0): Provides a persistent store for simple data, allowing the application to save and retrieve key-value pairs efficiently.

Additionally, the money mate makes use of the following key Flutter widgets:

- **MaterialApp**: Represents the root widget of a Flutter application, providing a material design.
- **Scaffold**: Implements the basic material design visual structure, incorporating the AppBar and body content.
- **CustomScrollView**: Provides a scrollable view with slivers.
- **SliverAppBar**: Represents the top app bar that typically contains the application title.
 - Expanded height: 200.0
 - Flexible space: CustomAppBar widget as flexible space.
- **CustomAppBar**: A custom widget for the flexible space of the SliverAppBar.
 - Takes a height parameter.
- **SliverToBoxAdapter**: Converts a box widget into a sliver for CustomScrollView.
- **Container**: A box model that contains other widgets.
 - Contains a Column widget.
- **Column**: Organizes children widgets in a vertical array.
- **CustomTextField**: A custom widget for a text input field.
 - Takes hint text and a controller as parameters.

- **CustomPasswordField**: Similar to CustomTextField but designed for password input.
- **Text**: Displays a short piece of text.
 - Shows a validation message in red if there's an error.
- **CustomTextButton**: A custom button widget.
 - Takes text and a callback function as parameters.
 - Triggers the login process and API call.
- **GestureDetector**: Listens to gestures, used to dismiss the keyboard when tapping outside the text fields.
- **_LoginScreenState**: Manages the state of the LoginScreen widget.
 - Uses TextEditingController for email and password inputs.
 - Calls the loginUserApiCall function on login button press.
- **loginUserApiCall**: A function making an API call to attempt user authentication.
- **Navigator.push**: Navigates to the HomeScreen if authentication is successful.

These packages and widgets collectively contribute to the development and functionality of the MoneyMate application.

4.3 Description of database integrated via APIs

The MoneyMate project encompasses a robust set of APIs catering to user management, group creation, and expense tracking. These APIs are instrumental in facilitating seamless interactions between the application and the backend server. Below is a detailed overview of the APIs integrated into the MoneyMate system.

4.3.1 GET - Get All Users

Endpoint: /api/users

Authorization: Bearer Token

Description: Retrieves information for all users, leveraging an authorization helper from the collection.

4.3.2 POST - Sign Up

Endpoint: /api/users

Authorization: Bearer Token

Description: Registers a new user through a sign-up process, utilizing an authorization helper from the collection.

4.3.3 POST - Login

Endpoint: /api/users/login

Authorization: Bearer Token

Description: Allows users to log in by providing their credentials. This request uses an authorization helper from the collection.

4.3.4 GET - Get User by ID

Endpoint: /api/users/:id

Authorization: Bearer Token

Description: Fetches user details based on the unique user ID and employs a Bearer Token for authorization.

4.3.5 PUT - Update User

Endpoint: /api/users/:email

Authorization: Bearer Token

Description: Updates user information, such as full name, password, and phone number, utilizing a Bearer Token for authentication.

4.3.6 GET - Get All Groups

Endpoint: /api/all_groups

Authorization: Bearer Token

Description: Retrieves information for all groups, employing an authorization helper from the collection.

4.3.7 POST - Create Group

Endpoint: /api/groups

Authorization: Bearer Token

Description: Facilitates the creation of a new group, with the group name provided in the request body. This request uses a Bearer Token for authentication.

4.3.8 GET - Get Groups by Token

Endpoint: /api/groups

Authorization: Bearer Token

Description: Fetches groups associated with the provided Token for authenticated users.

4.3.9 POST - Add Member

Endpoint: /api/groups/:groupId/members

Authorization: Bearer Token

Description: Adds a new member to an existing group, with the member's email provided in the request body.

4.3.10 POST - Add Expense

Endpoint: /api/expenses

Authorization: Bearer Token

Description: Records a new expense, including details such as description, amount, payer, and associated group. This request uses an authorization helper from the collection titled "SE Lab Project."

4.3.11 GET - All Expenses

Endpoint: /api/expenses/:groupId

Authorization: Bearer Token

Description: Retrieves information for all expenses associated with the provided Token, leveraging an authorization helper from the collection.

These APIs collectively form the backbone of the MoneyMate application, facilitating user interactions, group management, and transparent expense tracking.

4.3.12 POST - Add Friend

Endpoint: /api/friends/:email

Authorization: Bearer Token

Description: Adds a friend using the specified email address. This request utilizes a Bearer Token for authentication.

4.3.13 DELETE - Remove Friend

Endpoint: /api/friends/:email

Authorization: Bearer Token

Description: Removes a friend using the specified email address. This request requires a Bearer Token for authentication.

4.3.14 GET - Get All Friends

Endpoint: /api/friends

Authorization: Bearer Token

Description: Retrieves information for all friends. This request relies on a Bearer Token for authentication.

5 Chapter 5

5.1 Why Choose Us?

1. **Intelligent Expense Insights:** Money Mate goes beyond basic tracking, providing AI-driven insights for smarter financial decisions.
2. **Tailored Budgeting Tools:** Unlike others, Money Mate's budgeting tools are customized to individual spending patterns, offering a more personalized approach.
3. **Real-Time Visualizations:** Enjoy instant visualizations of your expenses, giving you a dynamic overview of your financial health, a feature not present in competing apps.
4. **Innovative Debt Settlement:** Money Mate introduces an intuitive debt settlement system, making it easier and more efficient than ever.
5. **Adaptable Collaboration:** Facilitates collaborative financial activities with a level of adaptability that surpasses Splitwise.
6. **Advanced Security Measures:** Money Mate prioritizes security with advanced measures, ensuring your financial data remains safe and confidential.
7. **Effortless Expense Categorization:** Easily categorize expenses with Money Mate's intuitive system, simplifying financial organization compared to Expensify.
8. **Smart Notification System:** Stay on top of your finances with a sophisticated notification system, providing timely reminders and updates unmatched by Venmo.
9. **Seamless Cloud Synchronization:** Money Mate offers secure cloud synchronization for seamless accessibility across devices, a feature that sets it apart from others.

10. **Customizable Interface:** Tailor the Money Mate interface to your preferences, ensuring a user-centric experience that surpasses the standard design of competing apps.
11. **Instant Expense Search:** Find specific expenses instantly with Money Mate's powerful search functionality, enhancing usability compared to Splitwise.
12. **Comprehensive Group Management:** Manage group expenses comprehensively, with features such as suggested repayments and simplified debt tracking, surpassing Splitwise.
13. **Holistic Financial Overview:** Money Mate provides a holistic financial overview, including intelligent suggestions, giving users a more complete picture compared to Venmo.
14. **Dynamic Expense Equalization:** Dynamically equalize expenses within a group, ensuring fairness in repayment, a feature not as robust in Splitwise.
15. **Enhanced Security Infrastructure:** Money Mate implements cutting-edge security measures, ensuring a higher level of data protection compared to Expensify.
16. **Efficient Receipt Scanning:** Effortlessly scan and organize receipts within Money Mate, streamlining the documentation process, a feature not as advanced in Venmo.
17. **Innovative Financial Recommendations:** Receive innovative financial recommendations and insights personalized to your spending habits, setting Money Mate apart from Expensify.
18. **Group Expense Synchronization:** Synchronize group expenses seamlessly with Money Mate, providing a more integrated experience than Venmo.
19. **Enhanced User Control:** Money Mate offers greater control over expense tracking, allowing users to customize and adapt the app to their specific needs compared to Splitwise.
20. **Personalized Financial Plans:** Generate personalized financial plans within Money Mate, ensuring users have actionable insights for better financial management, a feature not as robust in Venmo.

5.2 Conclusion

The Money Mate project aims to revolutionize shared expense management, offering a sophisticated and intelligent platform for users to seamlessly track, split, and settle financial obligations. With an intuitive user interface, advanced features, and robust security measures, Money Mate stands out as a comprehensive solution for individuals and groups navigating shared finances.

With its innovative features, advanced capabilities, and user-friendly interface, Money Mate is poised to become the preferred choice for anyone seeking a reliable and intelligent solution for shared expense management. As the financial landscape evolves, Money Mate stands ready to empower users in their journey towards financial transparency and efficiency.