Design Document

for

Brokeo

Version 1.0

Group Number:17 **GroupName:Runtime3rror**

Anjali Patra	230148	anjalip23@iitk.ac.in
Aujasvit Datta	220254	aujasvitd22@iitk.ac.in
Bhavnoor Singh	230293	bhavnoor23@iitk.ac.in
Darshan Sethia	220326	darshands22@iitk.ac.in
Dhriti Barnwal	230364	dhritib23@iitk.ac.in
Marisha Thorat	230637	marishajt23@iitk.ac.in
Rudransh Verma	230881	rudranshv23@iitk.ac.in
Sanjna S	230918	sanjnas23@iitk.ac.in
Solanki Shrey Jigneshbhai	231017	shreyjs23@iitk.ac.in
Suryansh Verma	231061	suryanshv23@iitk.ac.in

Course: CS253

Mentor TA: Mr. Paras Ghodeshwar

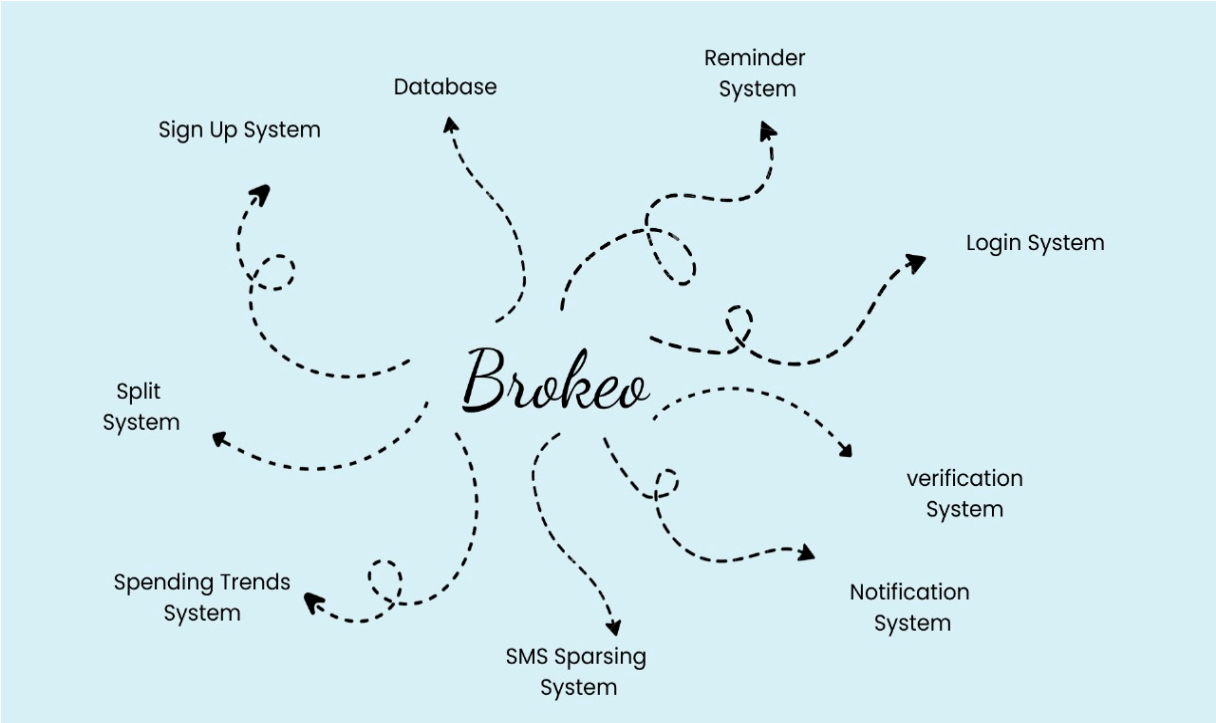
Date: 7 February 2025

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.0	Full Group	Completed first version of the document	07/02/24

1 Context Design

1.1 Context Model



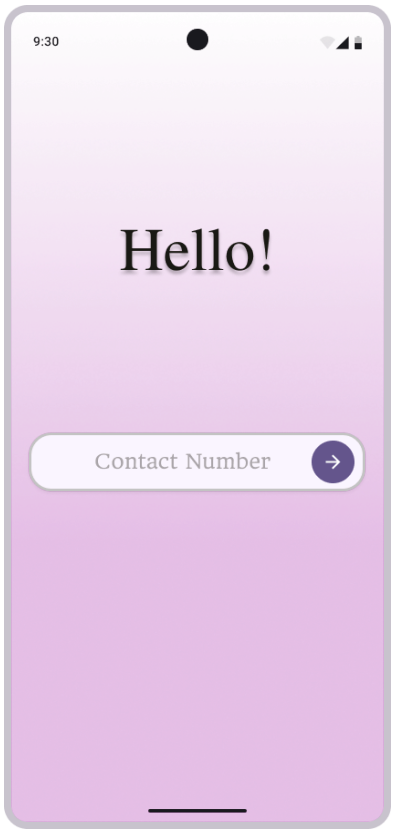
- **Login System** – This system securely handles user authentication and ensures they are redirected to their appropriate landing pages based on their selected category.
- **Sign-Up System** – Responsible for managing user registration, this system facilitates account creation and validates email addresses by communicating with the mailing system.
- **Mailing System** – The mailing system is used to send important notifications to users, including one-time verification links for email authentication during the sign-up process.
- **Notification System** – This system sends alerts to users regarding transactions, budget limits, reminders, and other important updates within the application.
- **Reminder System** – Designed to keep users informed, this system notifies them of upcoming dues, scheduled payments, and other financial commitments.
- **Database** – The database serves as the central repository for storing and organizing user information, transaction records, and spending categories efficiently.
- **Split System** – This system enables users to manage shared expenses by allowing them to split transactions, track outstanding debts, and request payments from contacts.

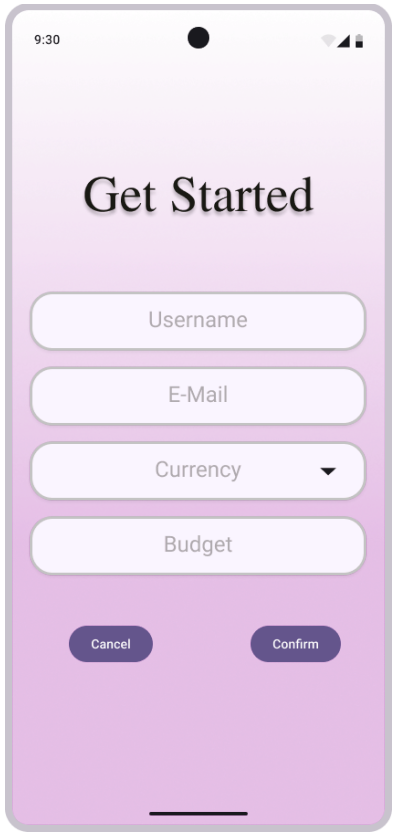
- **Spending Trends System** – By analyzing transaction data, this system provides users with insights into their spending habits, helping them make informed financial decisions.
- **SMS Parsing System** – This system extracts financial transaction details from incoming SMS messages, categorizes the expenses, and integrates them into the user's financial records.

1.2 Human Interface Design

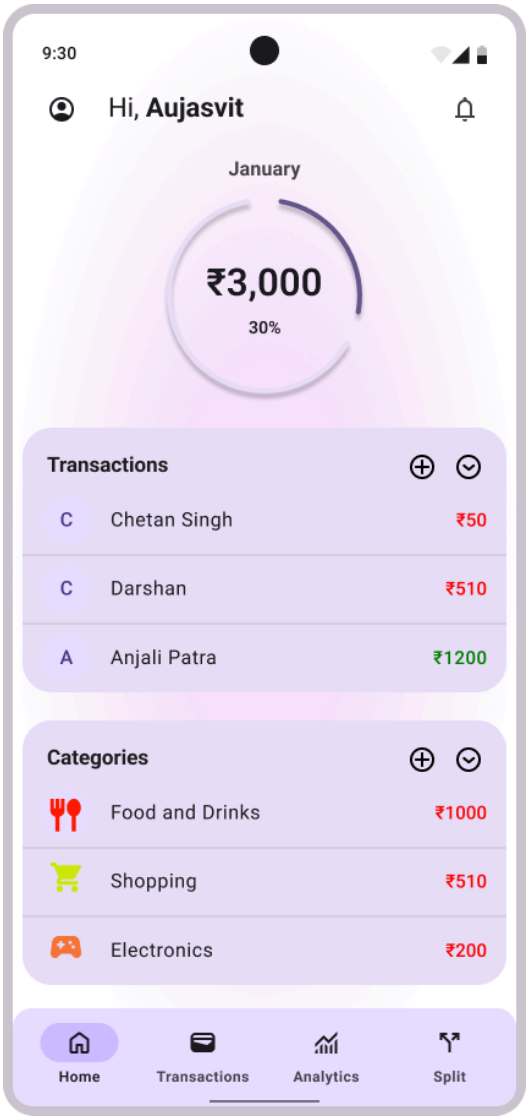
1.2.1 Landing Page:

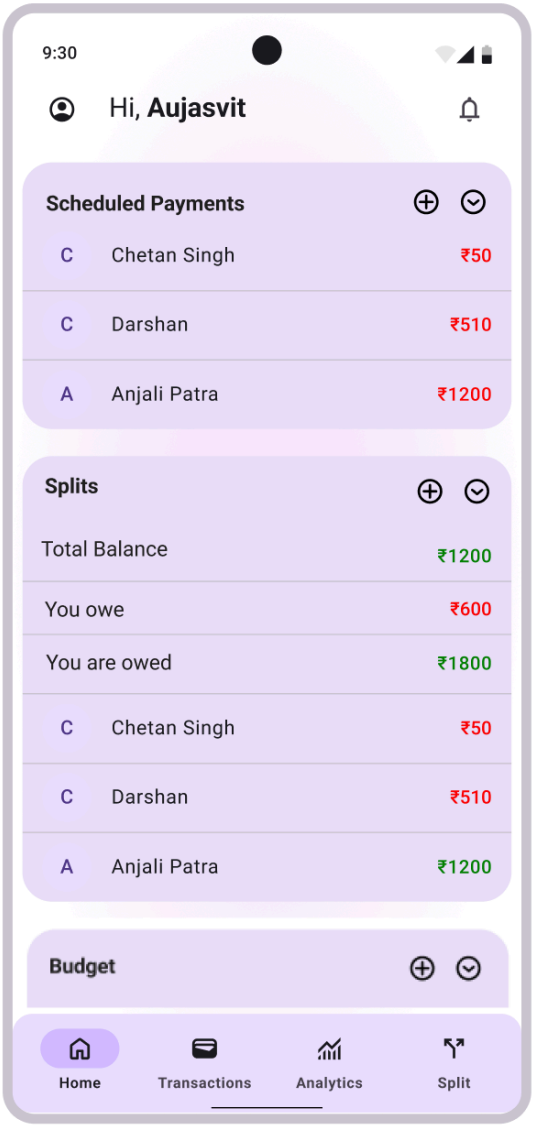
1.2.2 Login and Sign-up:

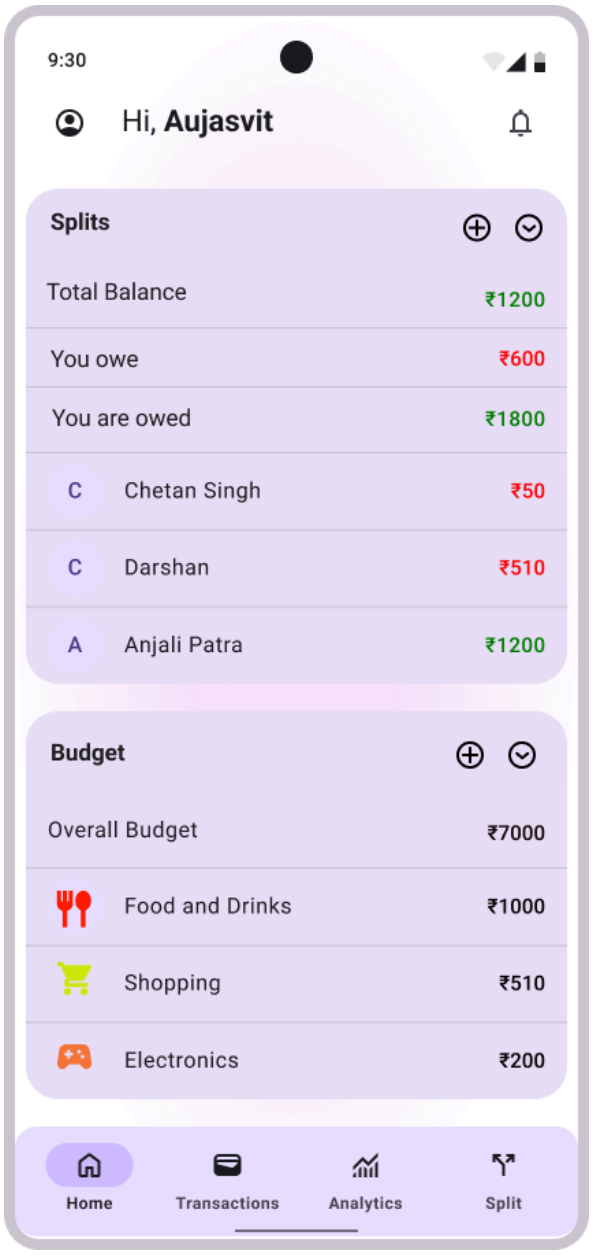


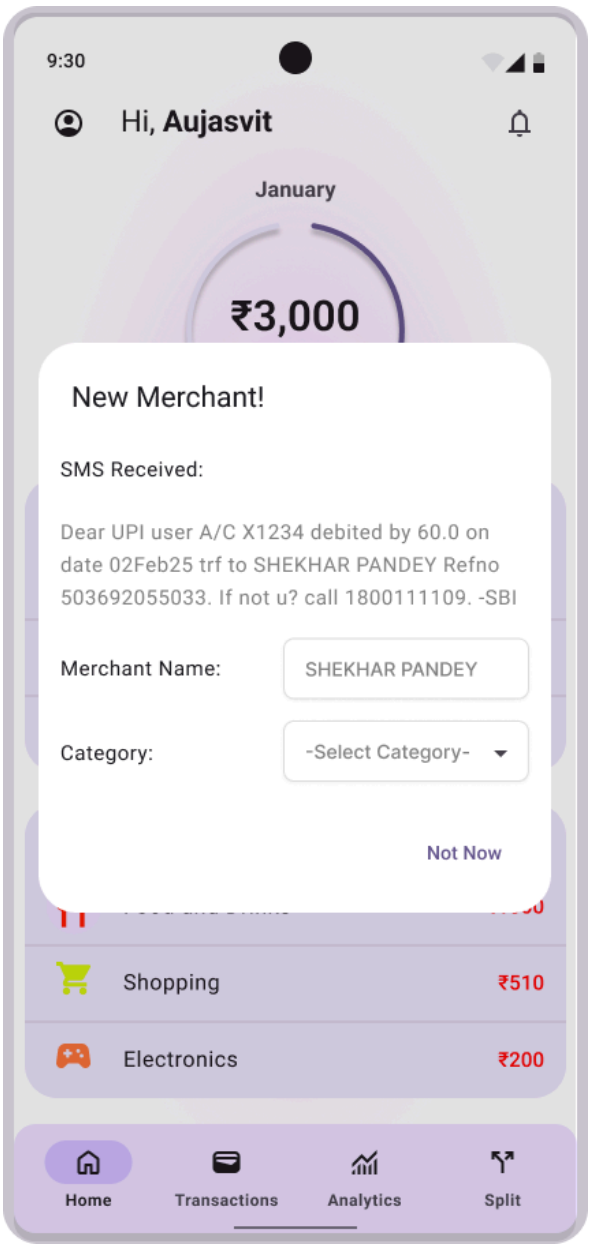


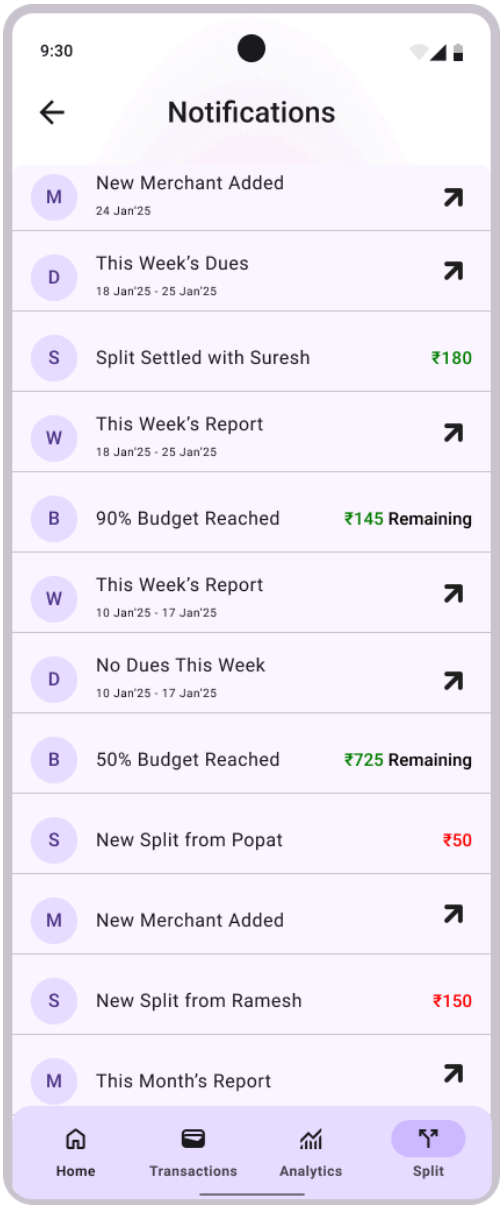
1.2.3 Home Page:



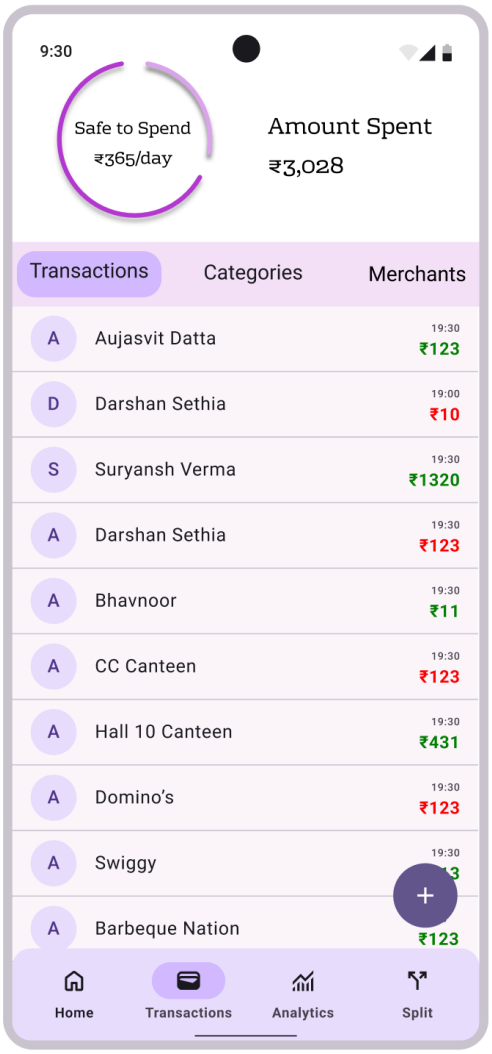


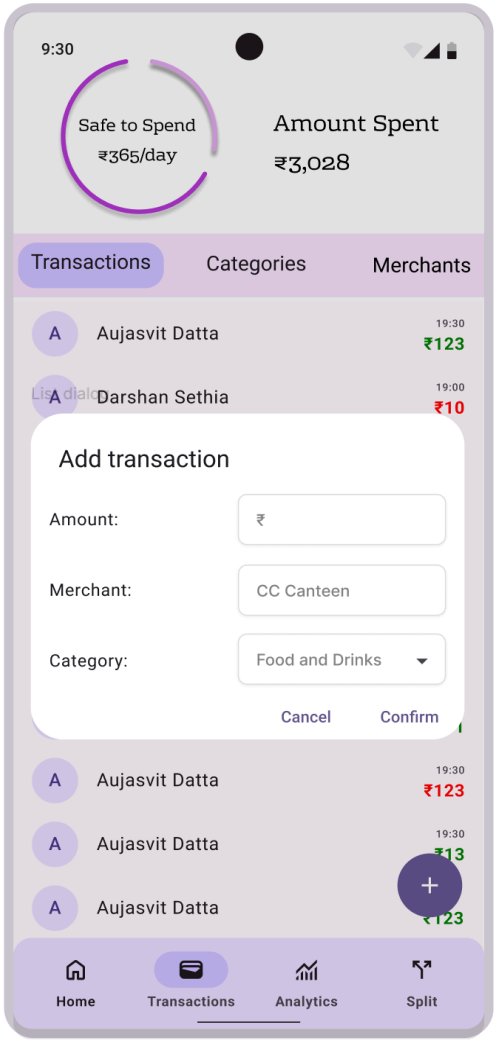

9:30
Hi, Aujasvit
Splits
Total Balance ₹1200
You owe ₹600
You are owed ₹1800
C Chetan Singh ₹50
C Darshan ₹510
A Anjali Patra ₹1200
Budget
Overall Budget ₹7000
Food and Drinks ₹1000
Shopping ₹510
Electronics ₹200
Home
Transactions
Analytics
Split

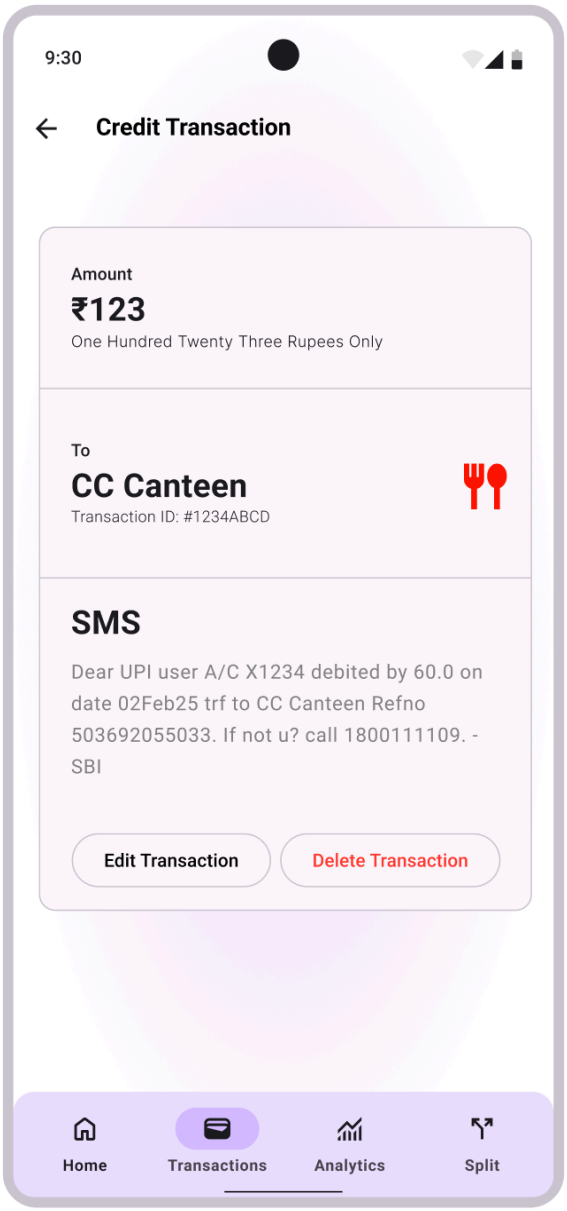

9:30
Hi, Aujasvit
January
₹3,000
New Merchant!
SMS Received:
Dear UPI user A/C X1234 debited by 60.0 on date 02Feb25 trf to SHEKHAR PANDEY Refno 503692055033. If not u? call 1800111109. -SBI
Merchant Name: SHEKHAR PANDEY
Category: -Select Category-
Not Now
Shopping ₹510
Electronics ₹200
Home
Transactions
Analytics
Split

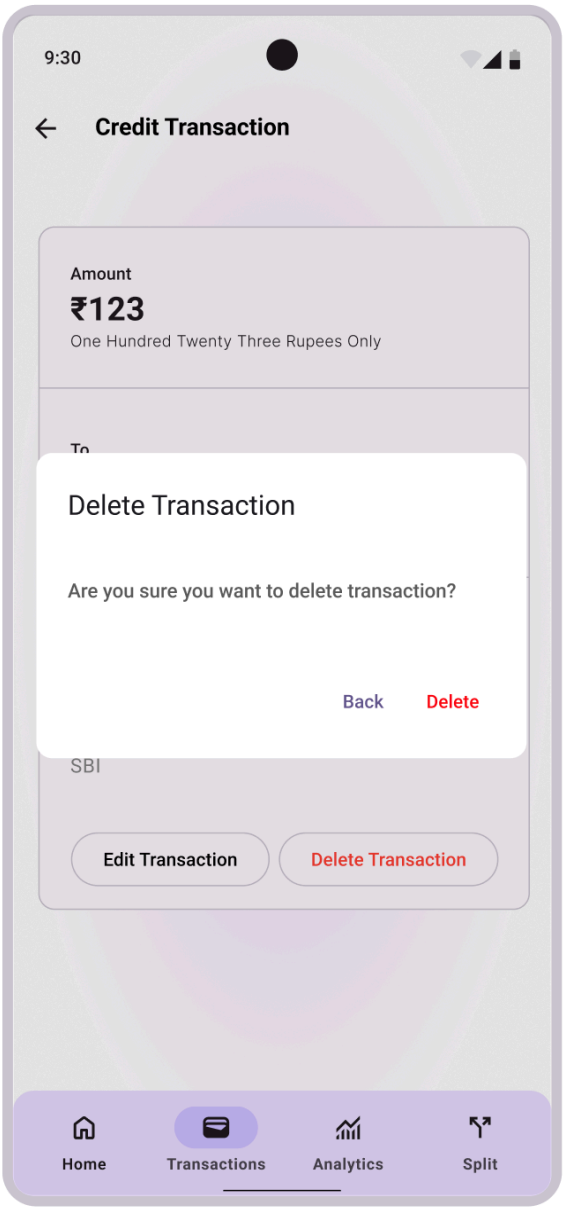

9:30
Notifications
M
New Merchant Added
24 Jan'25
D
This Week's Dues
18 Jan'25 - 25 Jan'25
S
Split Settled with Suresh
₹180
W
This Week's Report
18 Jan'25 - 25 Jan'25
B
90% Budget Reached
₹145 Remaining
W
This Week's Report
10 Jan'25 - 17 Jan'25
D
No Dues This Week
10 Jan'25 - 17 Jan'25
B
50% Budget Reached
₹725 Remaining
S
New Split from Popat
₹50
M
New Merchant Added
S
New Split from Ramesh
₹150
M
This Month's Report
Home
Transactions
Analytics
Split

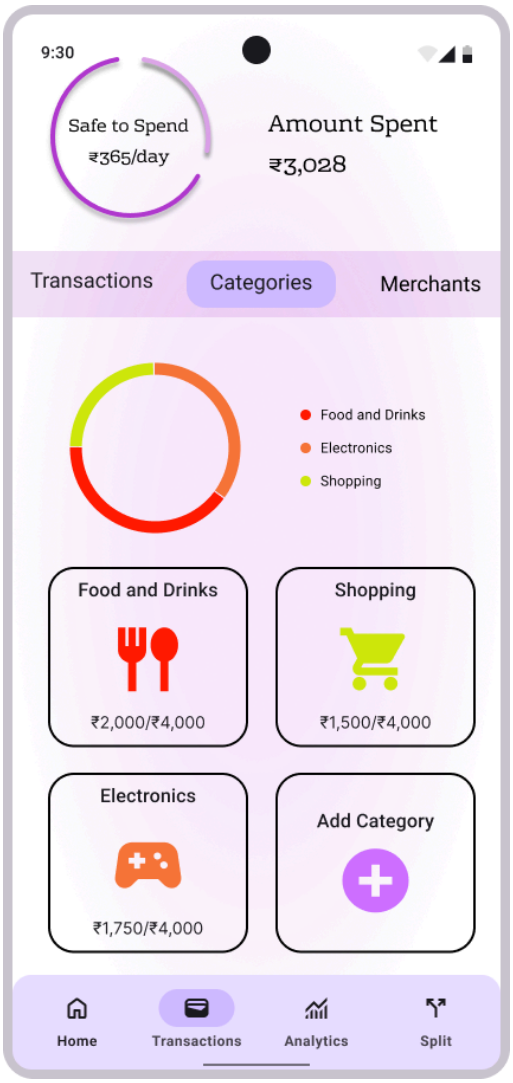
1.2.4 Transaction Page:

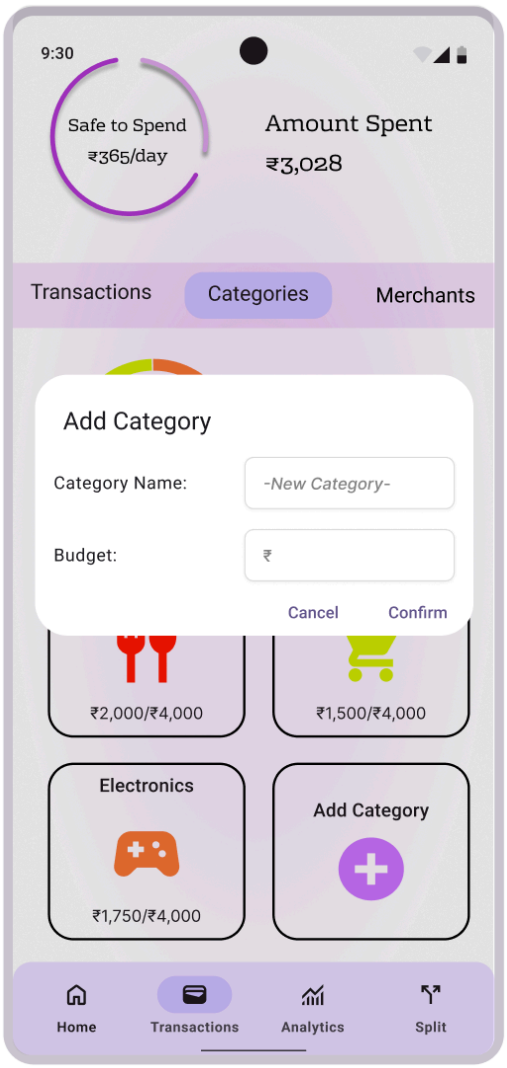


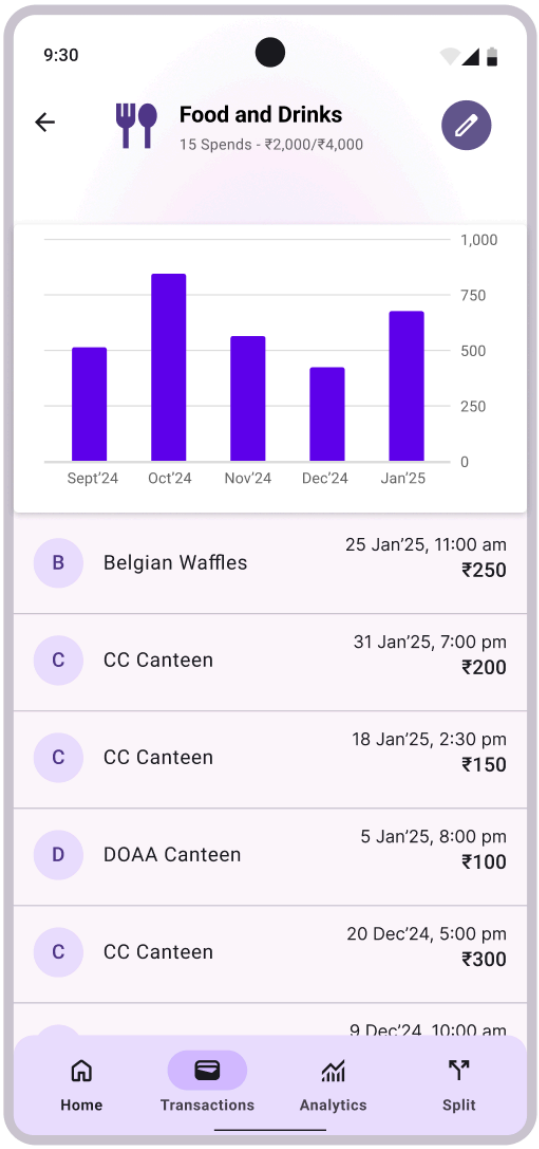



9:30
Credit Transaction
Amount
₹123
One Hundred Twenty Three Rupees Only
To
CC Canteen
Transaction ID: #1234ABCD
SMS
Dear UPI user A/C X1234 debited by 60.0 on date 02Feb25 trf to CC Canteen Refno 503692055033. If not u? call 1800111109. - SBI
Edit Transaction
Delete Transaction
Home
Transactions
Analytics
Split

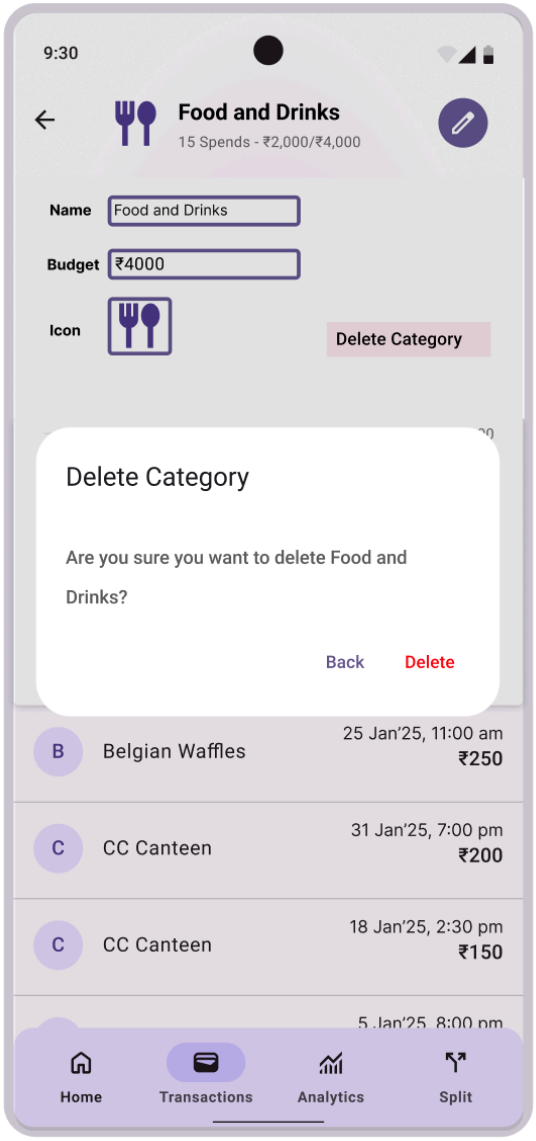

9:30
Credit Transaction
Amount
₹123
One Hundred Twenty Three Rupees Only
To
Delete Transaction
Are you sure you want to delete transaction?
Back
Delete
SBI
Edit Transaction
Delete Transaction
Home
Transactions
Analytics
Split

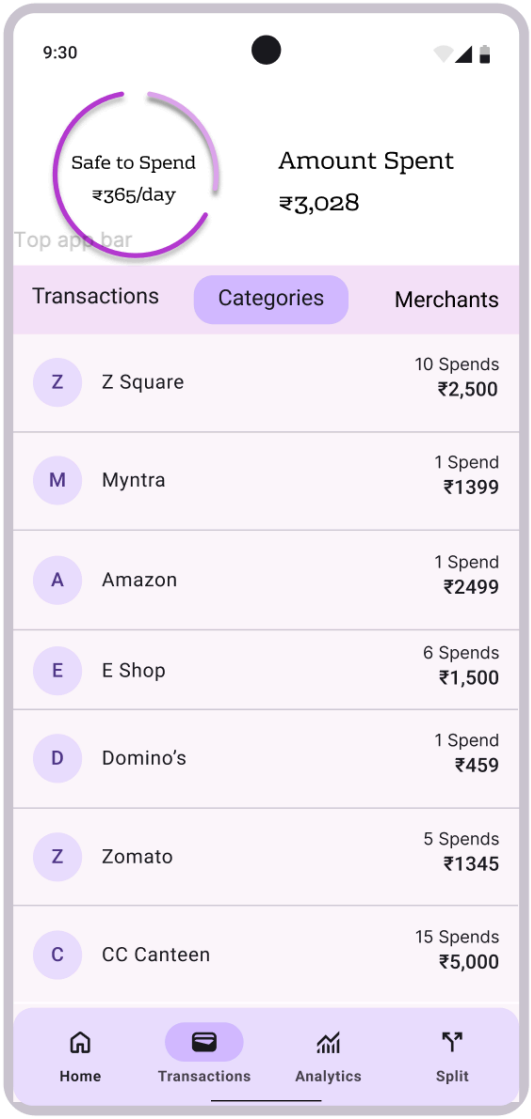

9:30
Safe to Spend
₹365/day
Amount Spent
₹3,028
Transactions
Categories
Merchants
Food and Drinks
Electronics
Shopping
Food and Drinks
₹2,000/₹4,000
Shopping
₹1,500/₹4,000
Electronics
₹1,750/₹4,000
Add Category
Home
Transactions
Analytics
Split

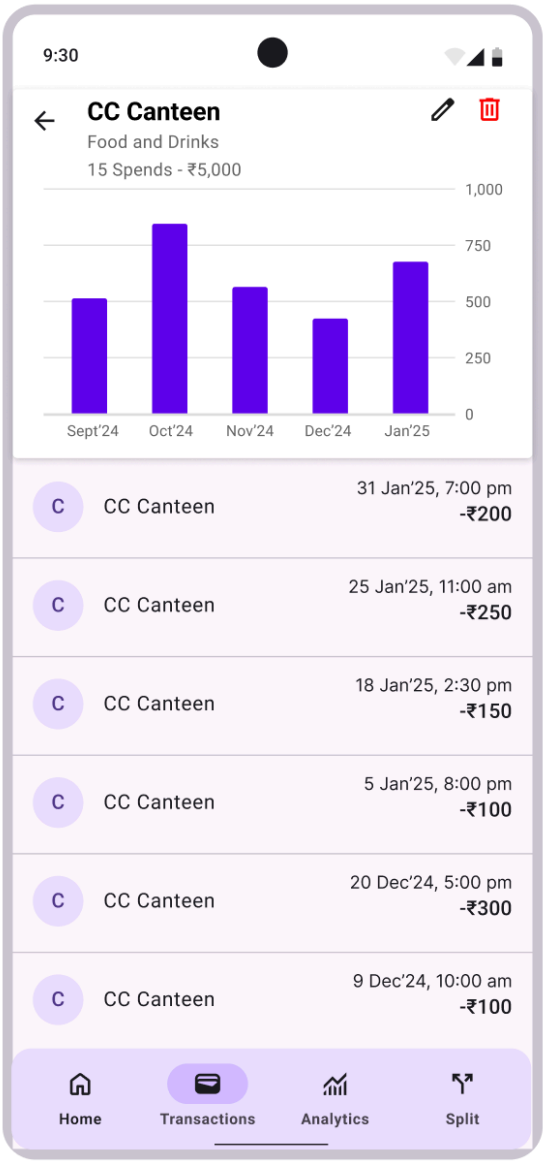

9:30
Safe to Spend
₹365/day
Amount Spent
₹3,028
Transactions
Categories
Merchants
Add Category
Category Name:
-New Category-
Budget:
₹
Cancel
Confirm
₹2,000/₹4,000
₹1,500/₹4,000
Electronics
₹1,750/₹4,000
Add Category
Home
Transactions
Analytics
Split

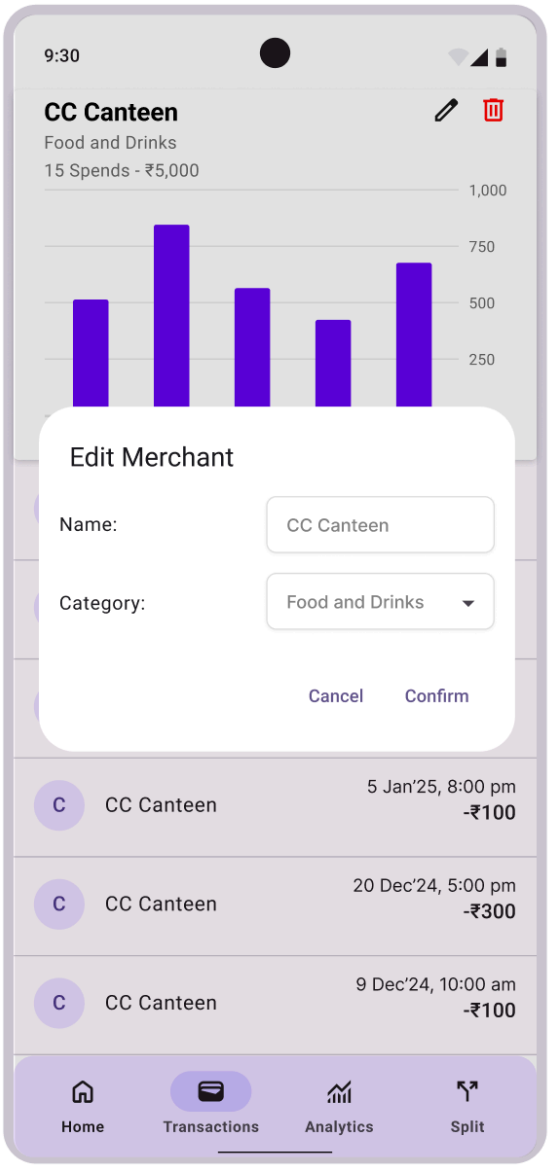

9:30
Food and Drinks
15 Spends - ₹2,000/₹4,000
1,000
750
500
250
0
Sept'24
Oct'24
Nov'24
Dec'24
Jan'25
B Belgian Waffles 25 Jan'25, 11:00 am ₹250
C CC Canteen 31 Jan'25, 7:00 pm ₹200
C CC Canteen 18 Jan'25, 2:30 pm ₹150
D DOAA Canteen 5 Jan'25, 8:00 pm ₹100
C CC Canteen 20 Dec'24, 5:00 pm ₹300
Home
Transactions
Analytics
Split

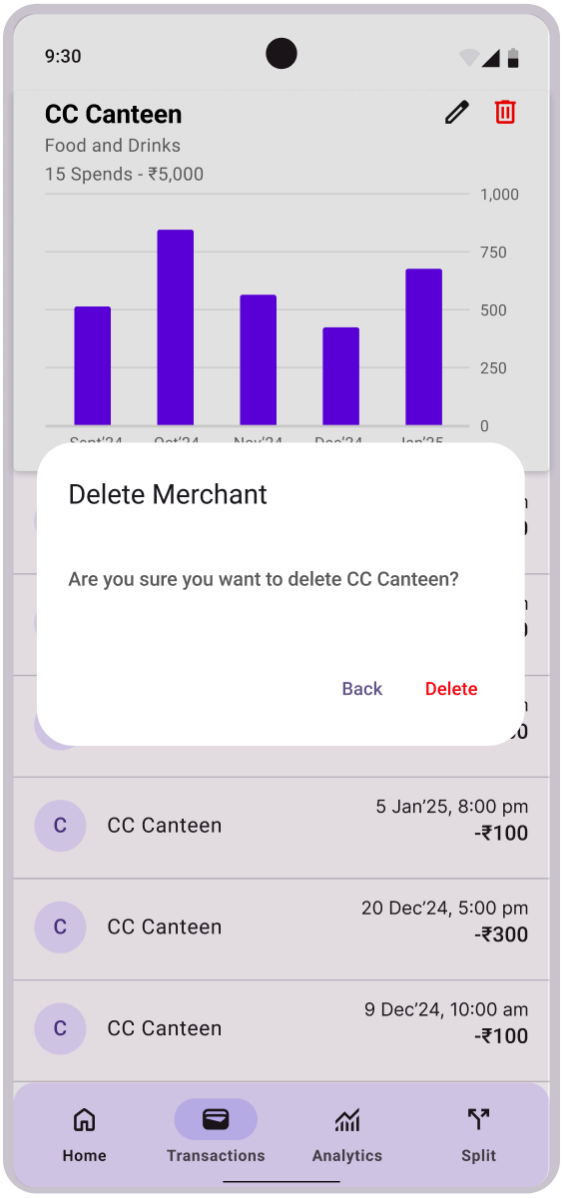

9:30
Food and Drinks
15 Spends - ₹2,000/₹4,000
Name Food and Drinks
Budget ₹4000
Icon
Delete Category
1,000
750
500
250
0
Sept'24
Oct'24
Nov'24
Dec'24
Jan'25
B Belgian Waffles 25 Jan'25, 11:00 am ₹250
C CC Canteen 31 Jan'25, 7:00 pm ₹200
C CC Canteen 18 Jan'25, 2:30 pm ₹150
Home
Transactions
Analytics
Split


9:30
Food and Drinks
15 Spends - ₹2,000/₹4,000
Name
Food and Drinks
Budget
₹4000
Icon
Delete Category
Delete Category
Are you sure you want to delete Food and Drinks?
Back
Delete
B
Belgian Waffles
25 Jan'25, 11:00 am
₹250
C
CC Canteen
31 Jan'25, 7:00 pm
₹200
C
CC Canteen
18 Jan'25, 2:30 pm
₹150
Home
Transactions
Analytics
Split

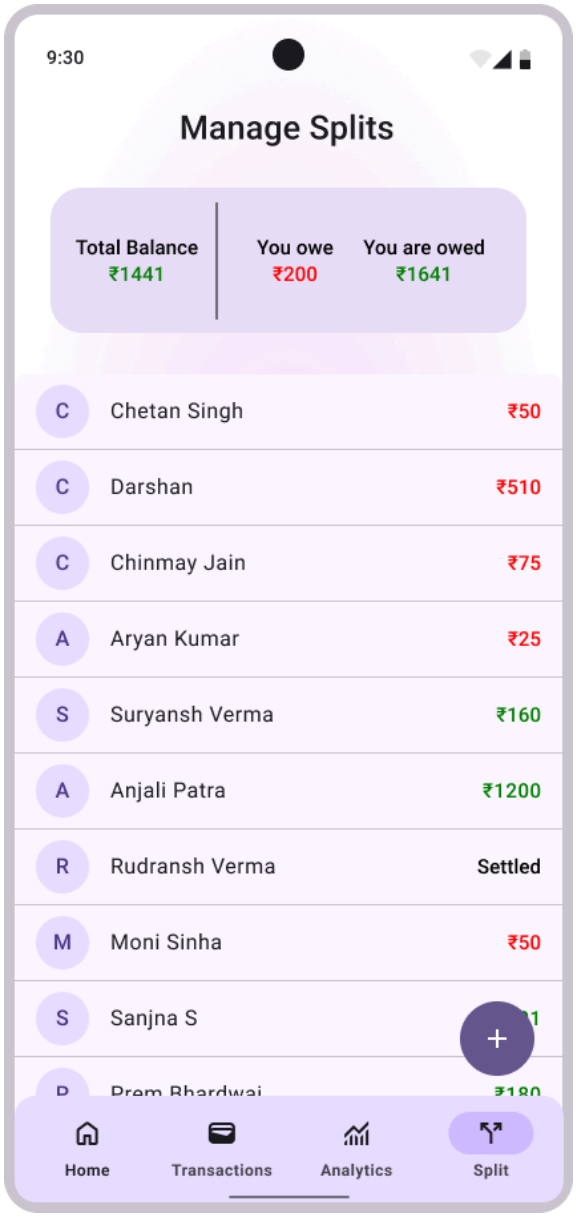

9:30
Safe to Spend
₹365/day
Amount Spent
₹3,028
Top app bar
Transactions
Categories
Merchants
Z
Z Square
10 Spends
₹2,500
M
Myntra
1 Spend
₹1399
A
Amazon
1 Spend
₹2499
E
E Shop
6 Spends
₹1,500
D
Domino's
1 Spend
₹459
Z
Zomato
5 Spends
₹1345
C
CC Canteen
15 Spends
₹5,000
Home
Transactions
Analytics
Split

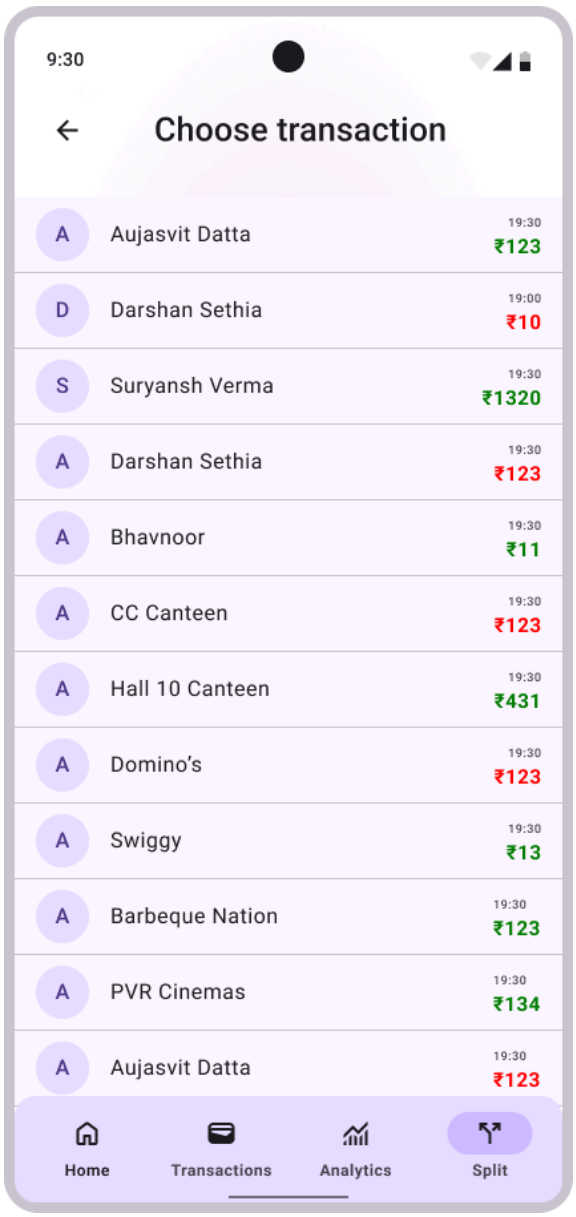

9:30
CC Canteen
Food and Drinks
15 Spends - ₹5,000
1,000
750
500
250
0
Sept'24
Oct'24
Nov'24
Dec'24
Jan'25
C CC Canteen 31 Jan'25, 7:00 pm -₹200
C CC Canteen 25 Jan'25, 11:00 am -₹250
C CC Canteen 18 Jan'25, 2:30 pm -₹150
C CC Canteen 5 Jan'25, 8:00 pm -₹100
C CC Canteen 20 Dec'24, 5:00 pm -₹300
C CC Canteen 9 Dec'24, 10:00 am -₹100
Home
Transactions
Analytics
Split

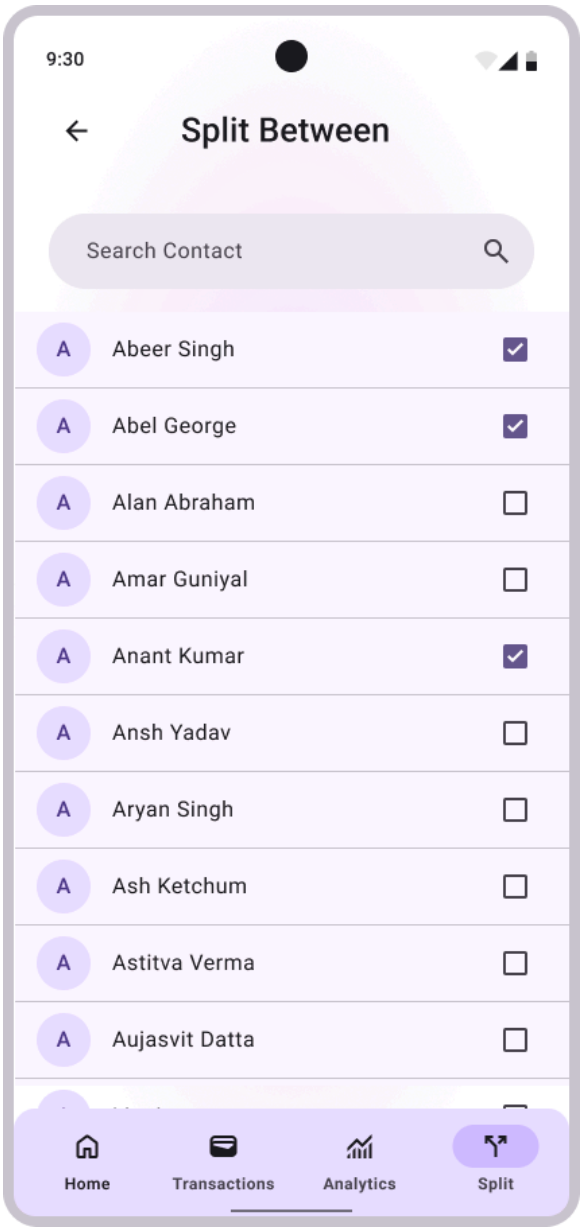

9:30
CC Canteen
Food and Drinks
15 Spends - ₹5,000
1,000
750
500
250
Edit Merchant
Name:
CC Canteen
Category:
Food and Drinks
Cancel
Confirm
C CC Canteen 5 Jan'25, 8:00 pm -₹100
C CC Canteen 20 Dec'24, 5:00 pm -₹300
C CC Canteen 9 Dec'24, 10:00 am -₹100
Home
Transactions
Analytics
Split

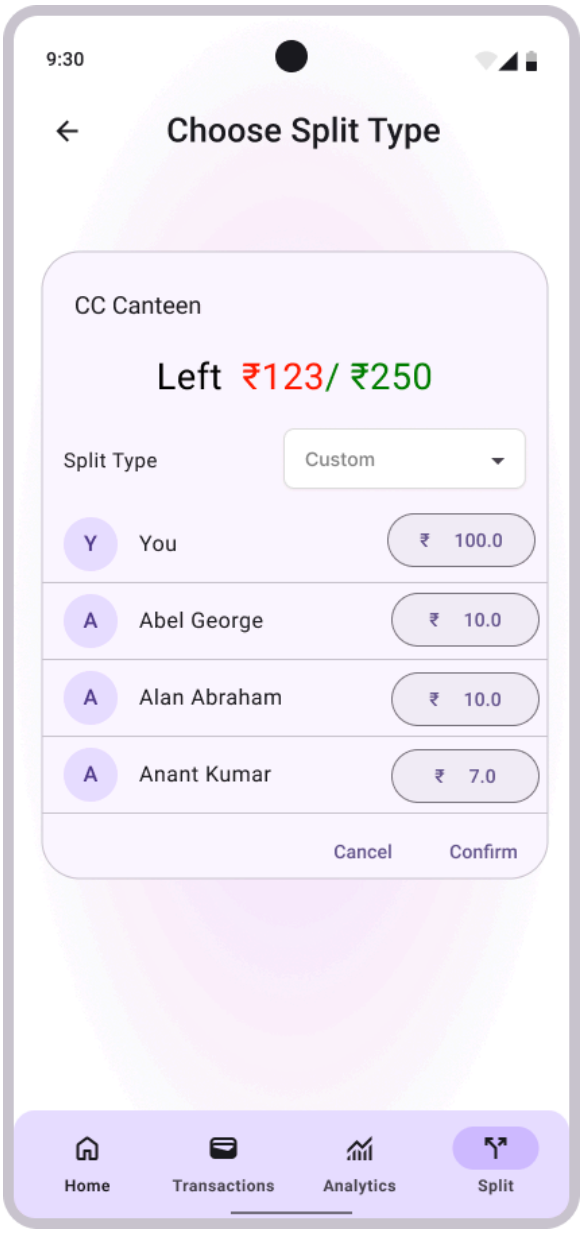

9:30
CC Canteen
Food and Drinks
15 Spends - ₹5,000
1,000
750
500
250
0
Delete Merchant
Are you sure you want to delete CC Canteen?
Back
Delete
C
CC Canteen
5 Jan'25, 8:00 pm
-₹100
C
CC Canteen
20 Dec'24, 5:00 pm
-₹300
C
CC Canteen
9 Dec'24, 10:00 am
-₹100
Home
Transactions
Analytics
Split

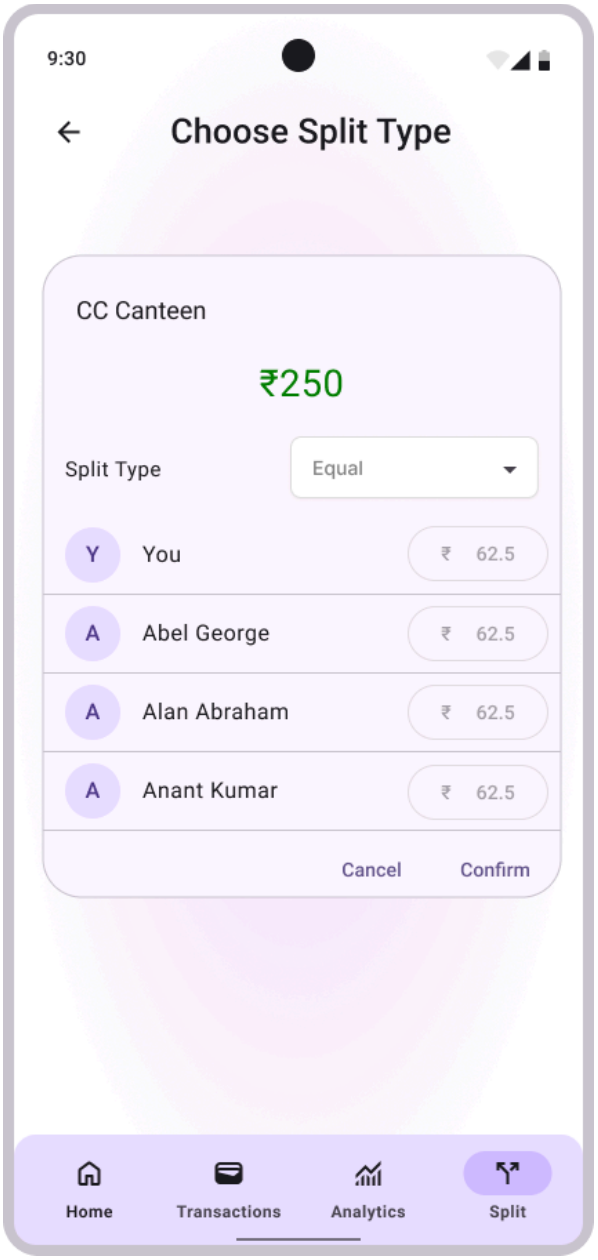
1.2.5 Split Page:



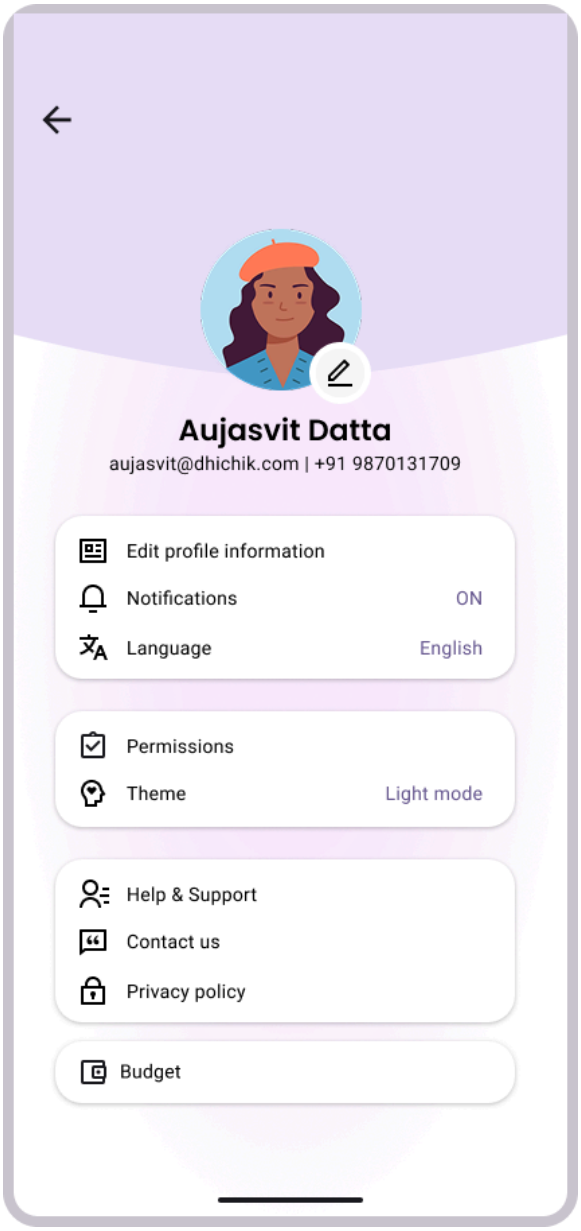


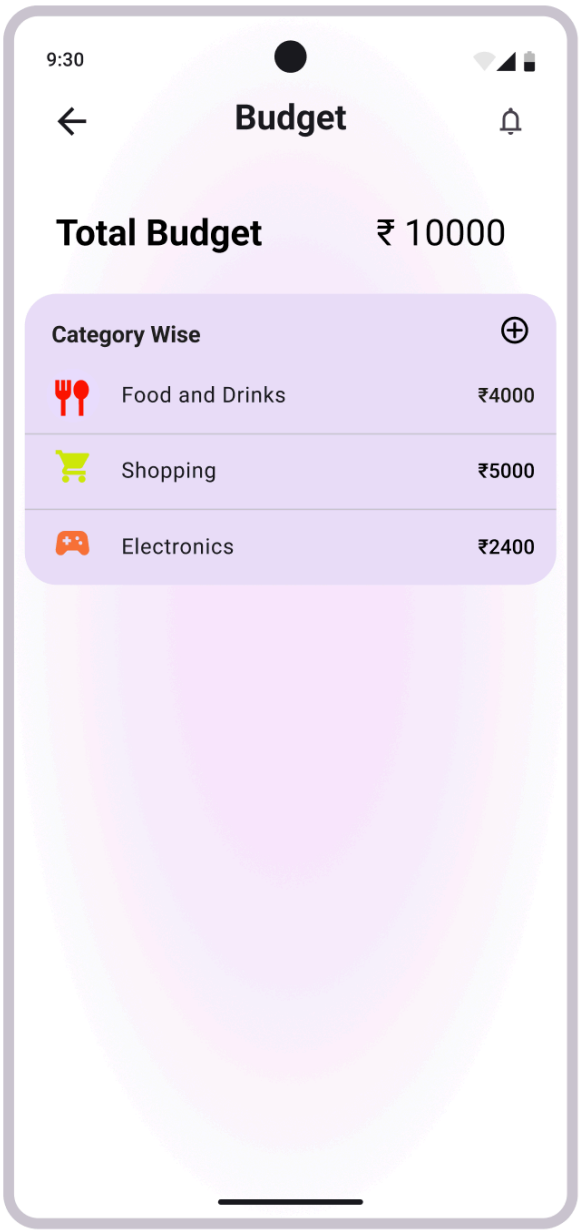

9:30
Split Between
Search Contact
A Abeer Singh
A Abel George
A Alan Abraham
A Amar Guniyal
A Anant Kumar
A Ansh Yadav
A Aryan Singh
A Ash Ketchum
A Astitva Verma
A Aujasvit Datta
Home
Transactions
Analytics
Split

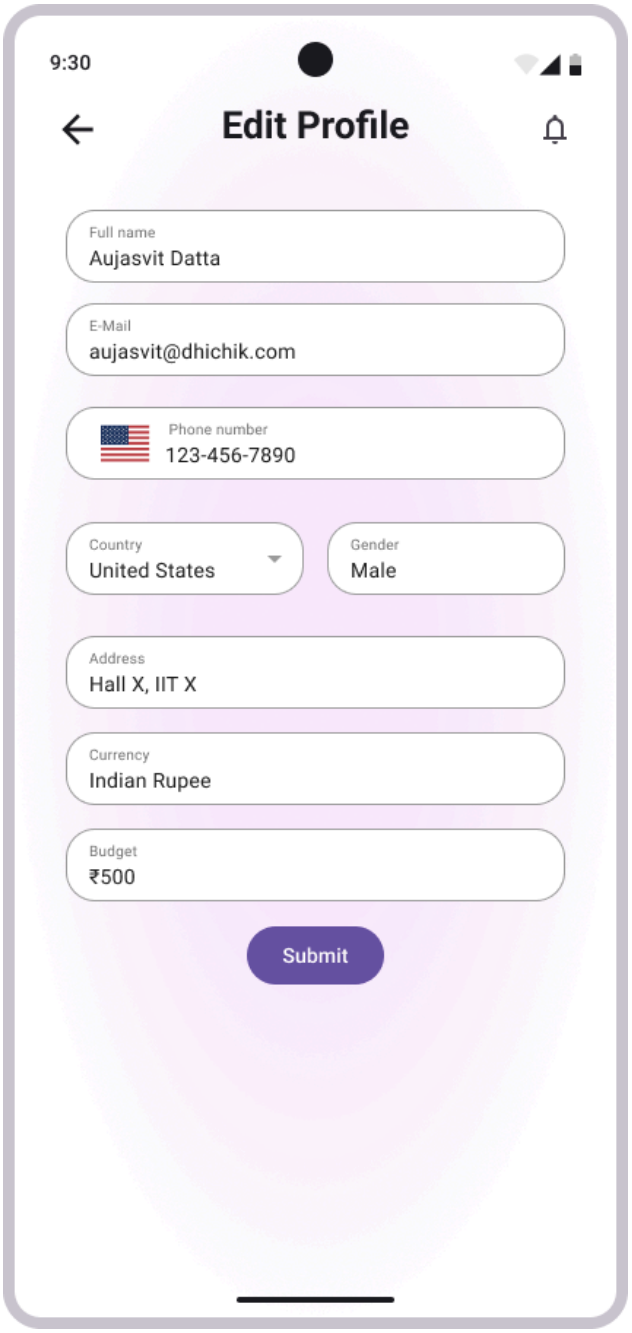

9:30
Choose Split Type
CC Canteen
Left ₹123/ ₹250
Split Type
Custom
Y You ₹ 100.0
A Abel George ₹ 10.0
A Alan Abraham ₹ 10.0
A Anant Kumar ₹ 7.0
Cancel
Confirm
Home
Transactions
Analytics
Split


9:30
Choose Split Type
CC Canteen
₹250
Split Type
Equal
Y
You
₹ 62.5
A
Abel George
₹ 62.5
A
Alan Abraham
₹ 62.5
A
Anant Kumar
₹ 62.5
Cancel
Confirm
Home
Transactions
Analytics
Split

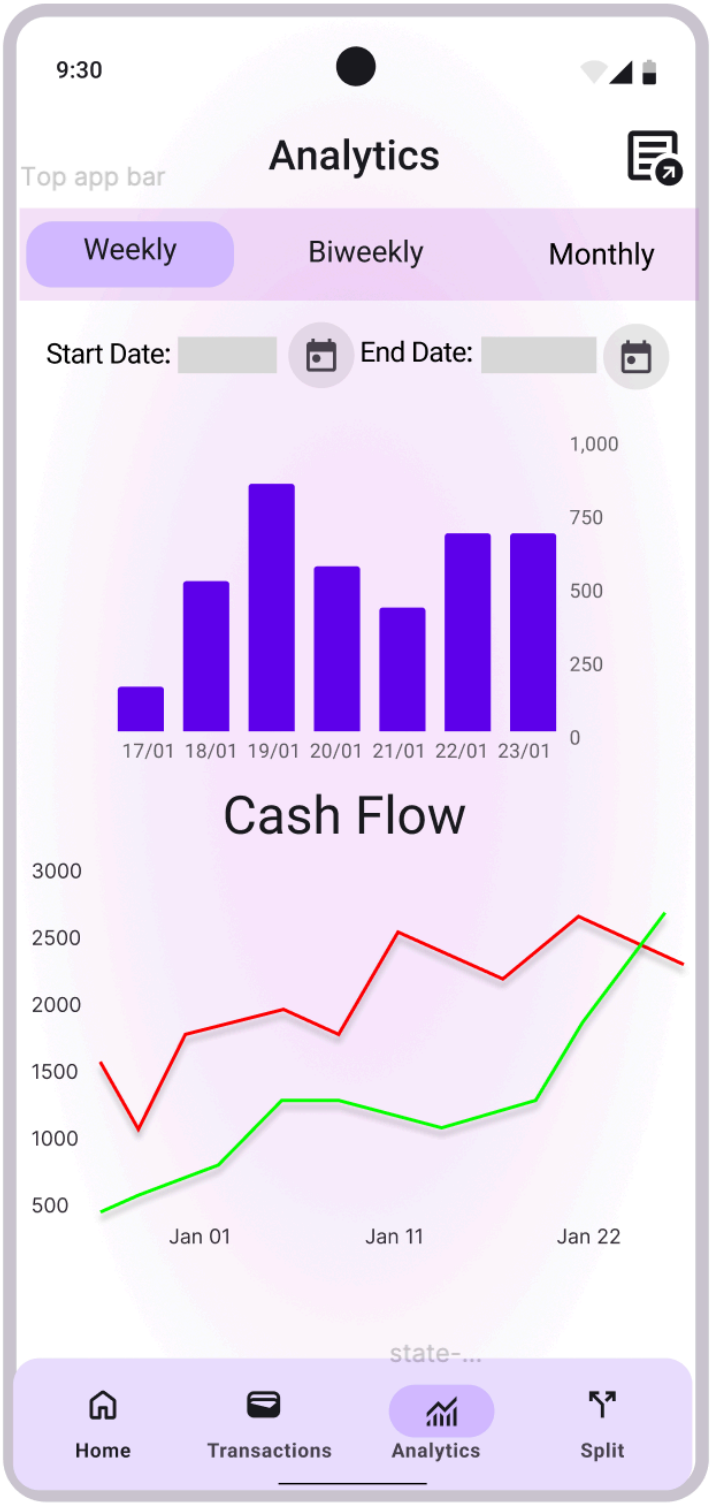
1.2.6 User Profile:






9:30
Edit Profile
Full name
Aujasvit Datta
E-Mail
aujasvit@dhichik.com
Phone number
123-456-7890
Country
United States
Gender
Male
Address
Hall X, IIT X
Currency
Indian Rupee
Budget
₹500
Submit

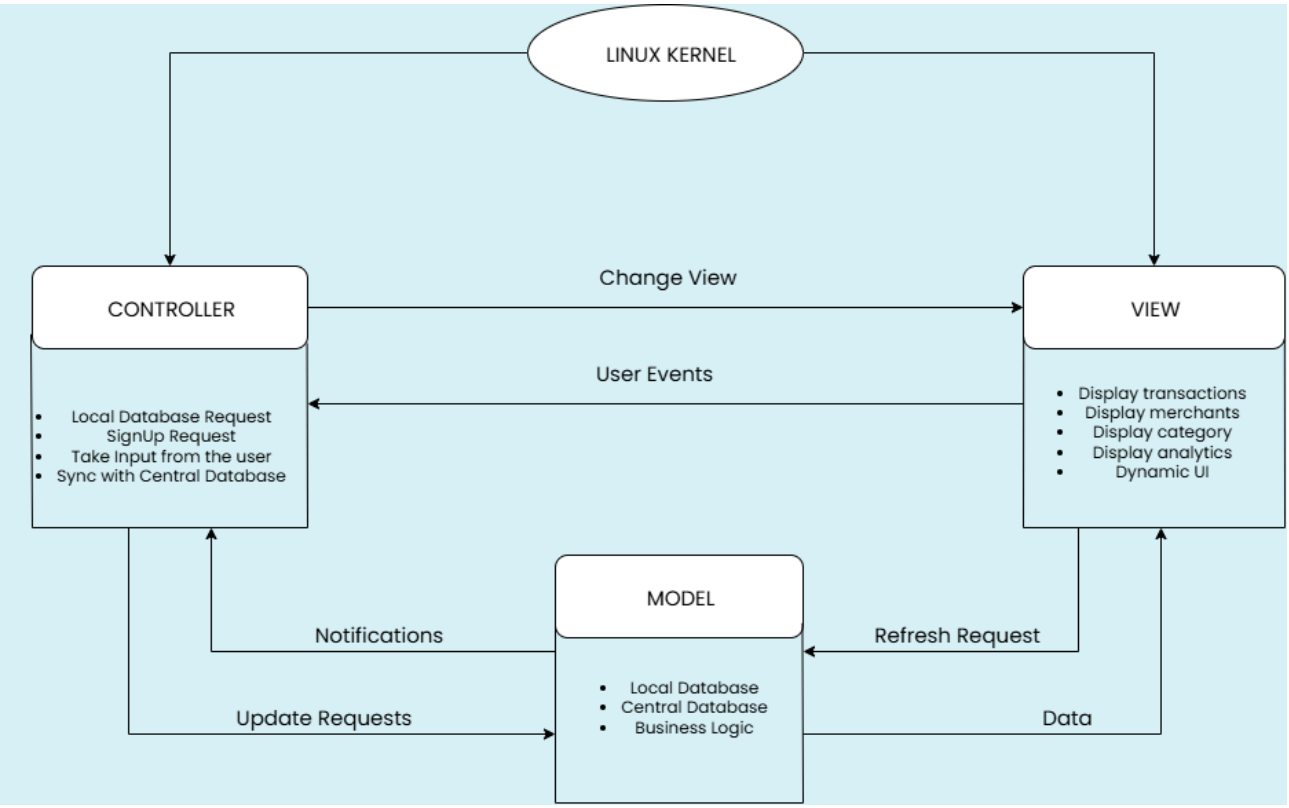
1.2.7 Analytics Page:



2 Architecture Design

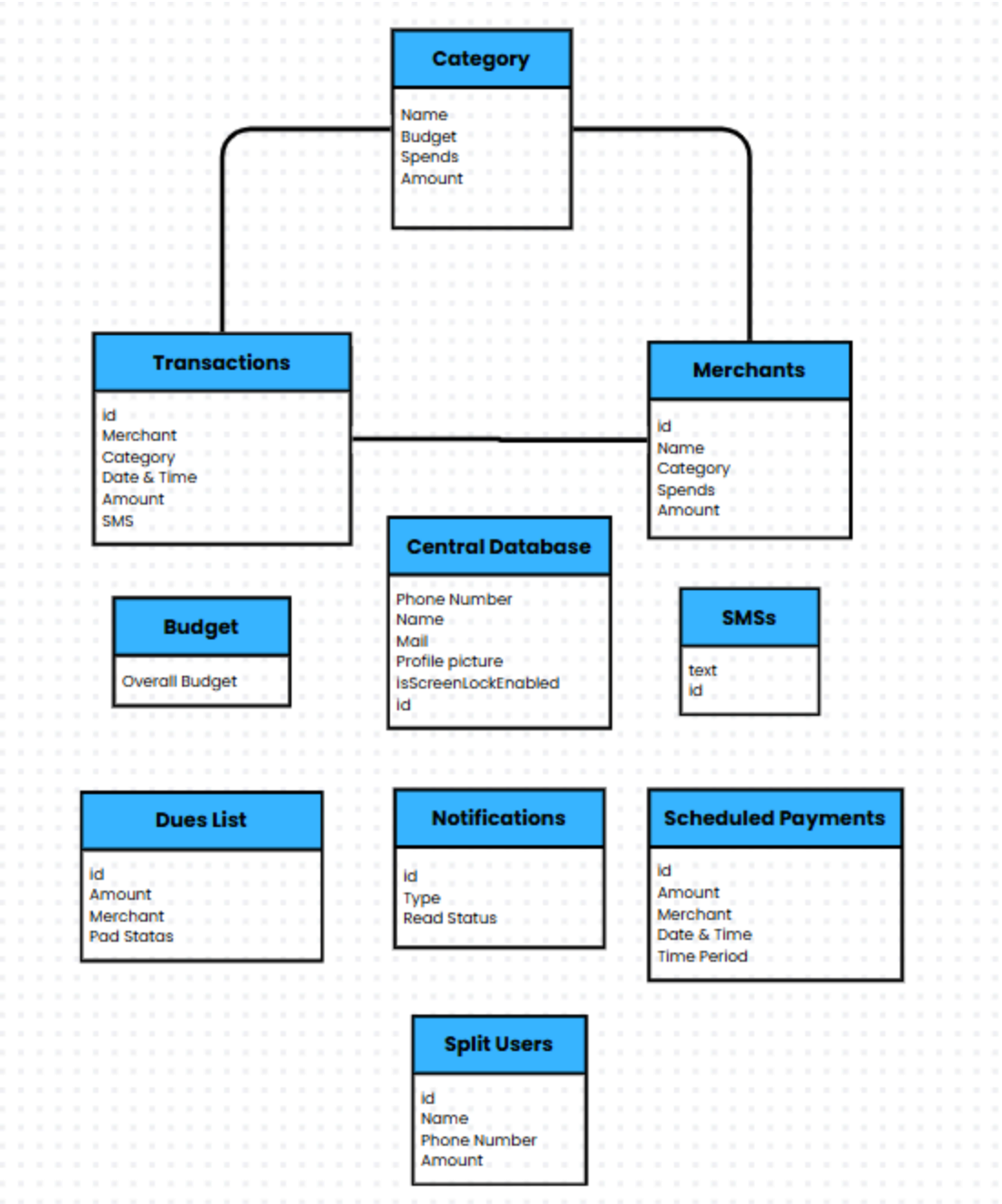
Model View Control Architecture

We will implement the Model-View-Controller (MVC) architecture in Brokeo to efficiently manage and present financial data. This pattern is well-suited for applications requiring complex data interactions, such as tracking expenses, splitting transactions, analyzing spending trends, and managing notifications. Additionally, MVC provides flexibility, allowing the system to scale and incorporate new features as needed in the future.



Model

The Model will interact with central databases as well as between themselves to support Brokeo's core functionalities, including transaction tracking, merchant categorization, split payments, budget management, notifications, and scheduled dues. It will actively fetch, store, and update data to keep financial records accurate and up to date.



Controller

The Controller acts as the bridge between the Model and the View, processing user requests and ensuring smooth interaction. Its key functions include:

- **Requests** – Handling user interactions, such as adding transactions, modifying categories, or exporting reports.
- **Authorization** – Verifying login credentials and managing user authentication to ensure secure access.

View

The View represents the User Interface (UI), dynamically updating based on user input and backend data. Different users will interact with tailored views based on their needs. Broadly classified, these views include:

- **Dashboard** – Provides users with insights into transactions, spending trends, and budget summaries.
- **Transaction Management** – Allows users to add, modify, split, and categorize transactions.
- **Split Expense View** – Displays shared expenses, outstanding balances, and payment requests.
- **Budget & Analytics View** – Shows spending breakdowns, trends, and category-wise expenses.
- **Notification & Reminder Panel** – Lists alerts for upcoming payments, budget limits, and new transactions.
- **Sign-in & Login Pages** – Facilitates user authentication and account creation.

Each view updates in real-time, ensuring users have access to the latest financial data while making Brokeo intuitive and interactive.

3 Object Oriented Design

3.1 Use Case Diagrams

3.1.1 Use Case #1: Track Transactions Automatically (U1)

Authors: This use case was written by Anjali Patra and Marisha Thorat.

Purpose: To capture all user transactions automatically via SMS parsing for financial tracking.

Requirements Traceability: The system detects online transactions from incoming messages, identifies the merchant, and categorizes the purchase by merchant and spending category.

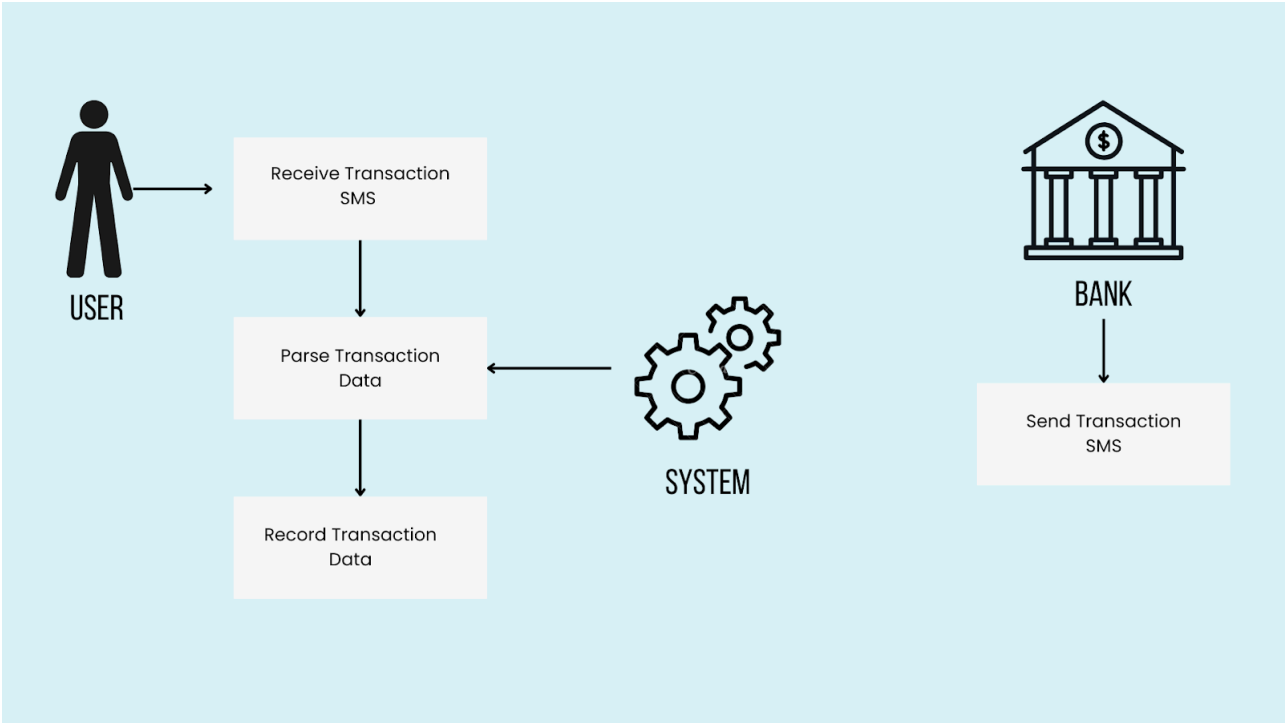
Priority: High - Critical for automatic transaction capture and accurate financial tracking.

Preconditions: The user must grant SMS permissions and enable transaction notifications from their bank or payment provider. The system requires an SMS parsing engine to detect transaction patterns and a secure database to store transaction data.

Post-conditions: Transaction data is automatically captured, parsed, and stored. Transactions are securely stored in the database and categorized.

Actors: User, SMS Parsing Engine, Transaction Database

Exceptions: Invalid or unreadable SMS format – the system cannot parse transaction details. SMS permissions not granted – transaction data cannot be captured.



3.1.2 Use Case #2: View and Manage Transactions (U2)

Authors: This use case was written by Rudransh Verma and Bhavnoor Singh.

Purpose: To allow users to view detailed information for each transaction, edit existing transactions, or add new ones.

Requirements Traceability: The system allows users to add, modify, and delete transactions. When adding a transaction, they can enter the amount, select or create a merchant, choose a spending category, and split it with contacts. Modifications allow changes to these fields, and users can delete any recorded transaction.

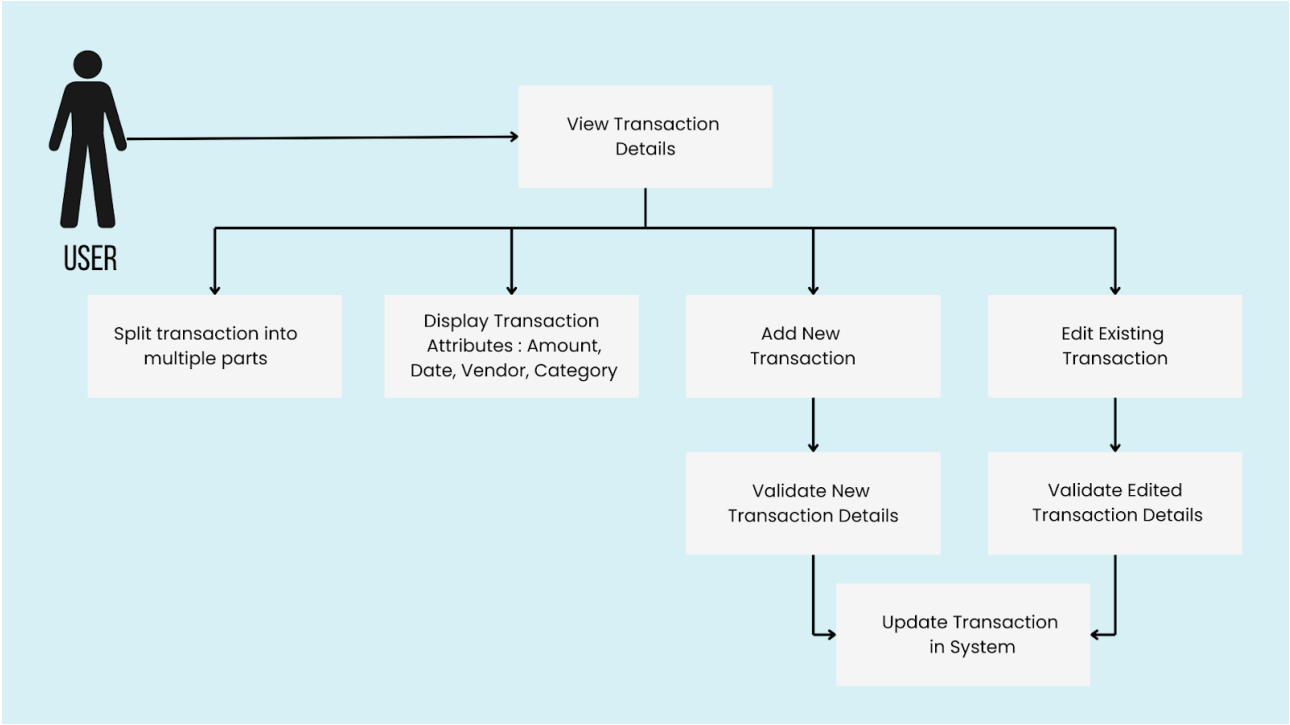
Priority: High - Essential for users to manage and edit their transactions effectively.

Preconditions: The system must have existing transaction data stored in the database and provide an interface that allows users to view, edit, and add transactions. Each transaction must be associated with defined categories and attributes. Additionally, the system must enforce validation rules for user-input transaction details to ensure the accuracy and integrity of the data.

Post-conditions: Transaction details are updated or new transactions are added. Validated transaction data is stored securely.

Actors: User, Transaction Database, System

Exceptions: Invalid transaction data input – system flags or rejects the entry.



3.1.3 Use Case #3: Set and Manage Budgets (U3)

Authors: This use case was written by Aujasvit Datta and Suryansh Verma.

Purpose: To set and track overall and category-specific (e.g., Food, Travel) budgets.

Requirements Traceability: The system lets users track spending by category (e.g., food, medical, groceries) and manage categories by adding, editing, or removing them. Users can assign categories to new merchants, change transaction or merchant categories, and set monthly budgets per category, viewing spent and remaining amounts.

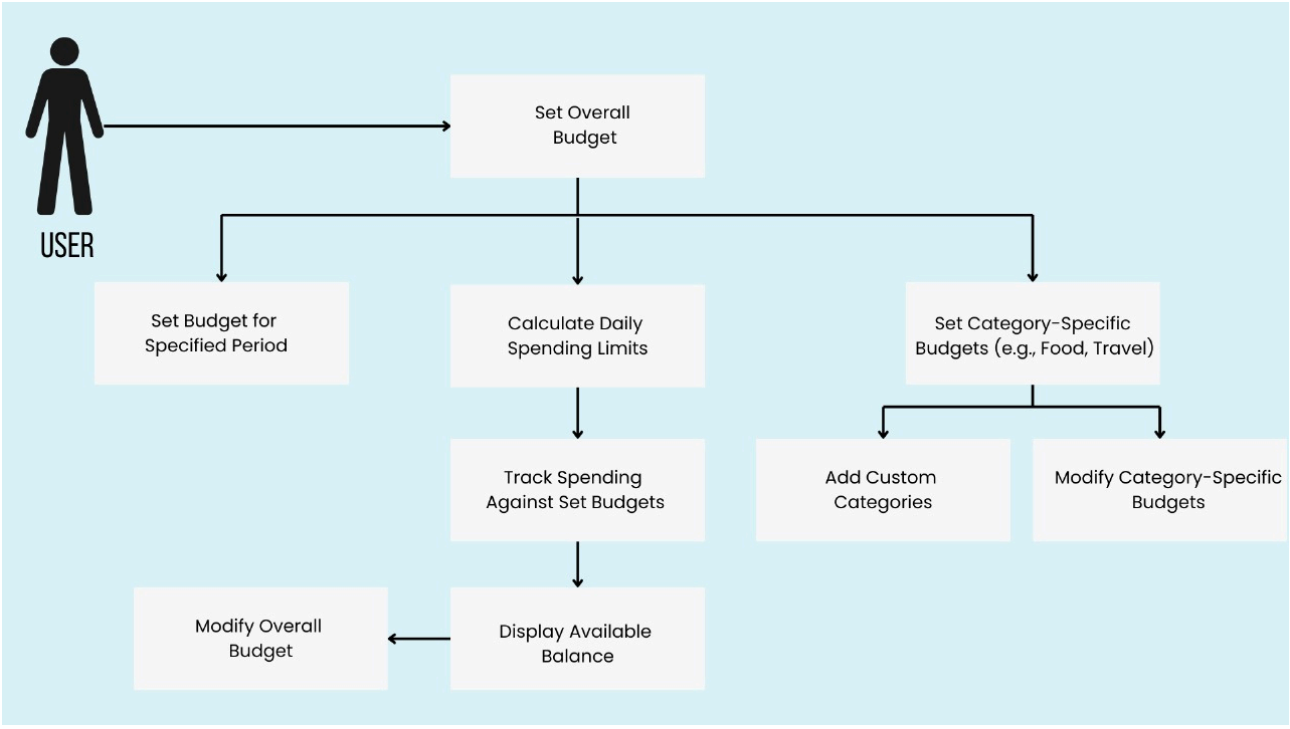
Priority: High - Core feature for managing overall and category-specific budgets to track expenses.

Preconditions: The system must have predefined categories or allow users to create custom ones. The system must have access to historical spending data for calculating daily limits. The system must have a budget setup interface available to users.

Post-conditions: Budgets are successfully set, modified, and tracked. Spending limits are updated, and users are notified of overspending.

Actors: User, System

Exceptions: Budget limit exceeded– user is alerted.



3.1.4 Use Case #4: Vendor Categorization and Expense Tracking (U4)

Authors: This use case was written by Darshan Sethia and Sanjna S.

Purpose: To categorize vendors into predefined or custom categories (e.g., Food, Travel) and track spending, including total amount and visit frequency.

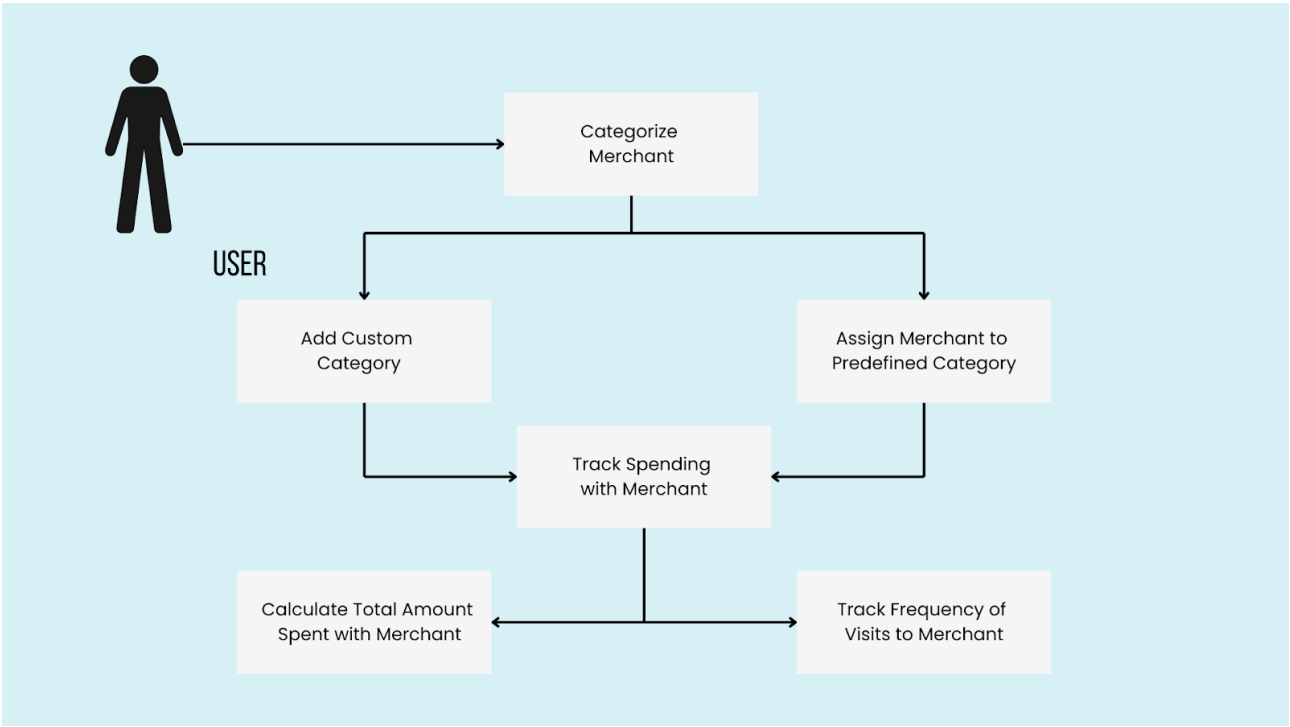
Requirements Traceability: Users can track spending by category (e.g., food, medical, groceries) and manage categories by adding, editing, or removing them. They can assign categories to new merchants, modify transaction or merchant categories, and set monthly budgets with usage details. Transactions can be viewed by merchant, showing total spending and transaction count for the month. Users can also assign or update merchant categories.

Priority: Medium - Important for categorizing vendors and tracking spending but secondary to core functions.

Preconditions: The system must have stored transaction data with associated vendor details. Predefined and user-defined categories must be configured in the system. The system must have mechanisms to track spending amounts and frequency for each vendor.

Post-conditions: Vendors are categorized, and expenditure is tracked. Total amounts and visit frequencies are updated and displayed.

Actors: User, Transaction database, System



3.1.5 Use Case #5: Analyze Trends of Transactions (U5)

Authors: This use case was written by Solanki Shrey Jigneshbhai and Dhriti Barnwal.

Purpose: To view spending trends by month, week, or day.

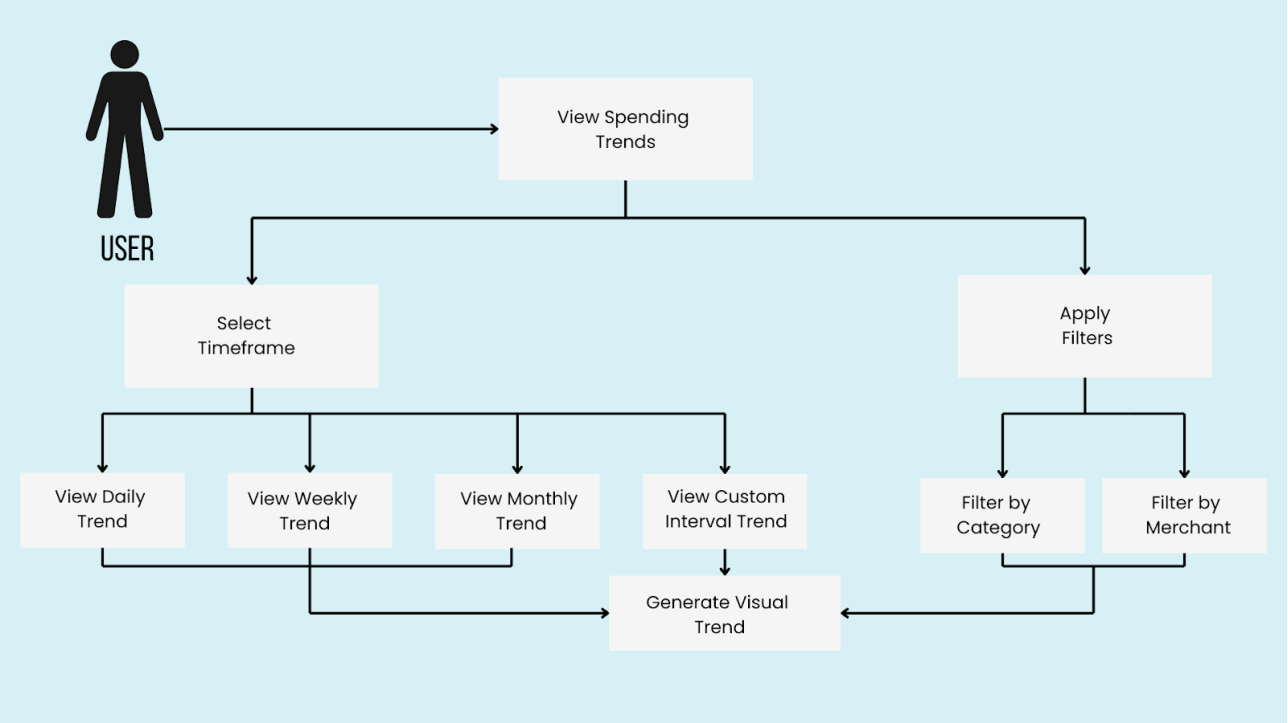
Requirements Traceability: Users can view detailed analytics for a selected time interval (week, month, quarter, year, or custom range), including total spending, average spending, comparisons with previous periods, and category-wise breakdowns. They can also export transaction data for any interval as a CSV file, including individual transactions, category totals, and overall spending.

Priority: Medium - Useful for users to analyze spending trends but not a critical feature.

Preconditions: The system must have stored transaction data categorized by timeframes and categories. The system must have a visualization interface for displaying trends. Users must be able to apply filters to trends based on categories or merchants.

Post-conditions: Spending trends are visualized and can be filtered by categories or merchants.

Actors: User, System, Transaction Database



3.1.6 Use Case #6: Export Expense Report (U6)

Authors: This use case was written by Anjali Patra and Marisha Thorat.

Purpose: To allow users to export expense reports for specified timeframes (monthly, weekly, or custom).

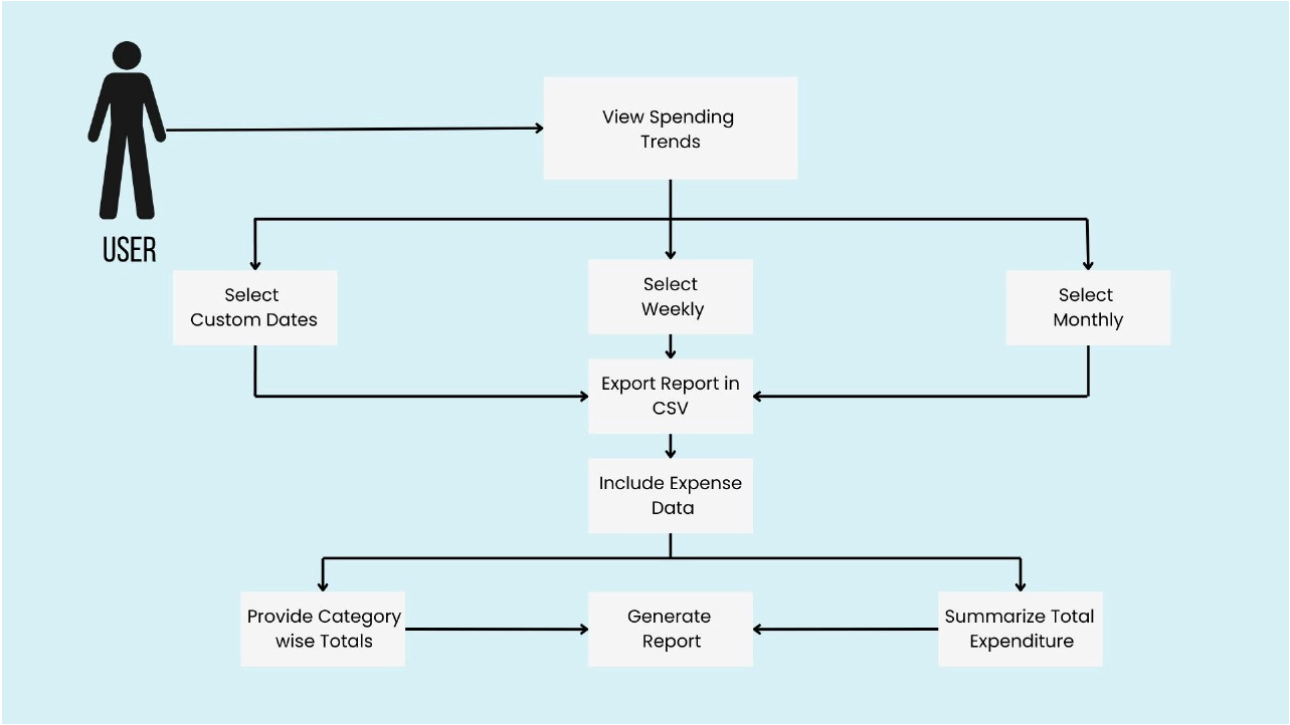
Requirements Traceability: The system allows users to export expense reports for specified timeframes (monthly, weekly, or custom) as a CSV file.

Priority: Medium - Valuable for exporting detailed reports, but not as essential as real-time tracking.

Preconditions: The system must have stored transaction data categorized by timeframes and attributes. The system must have an export interface available for users to select timeframes and data options.

Post-conditions: The expense report is successfully exported in the chosen format. Reports can be shared via email or other platforms (if configured).

Actors: User, System, Transaction Database



3.1.7 Use Case #7: Visualize Expenditure with Charts (U7)

Authors: This use case was written by Rudransh Verma and Suryansh Verma.

Purpose: To provide a visual representation of overall expenditure and category-wise (e.g., Food, Travel) spending using charts.

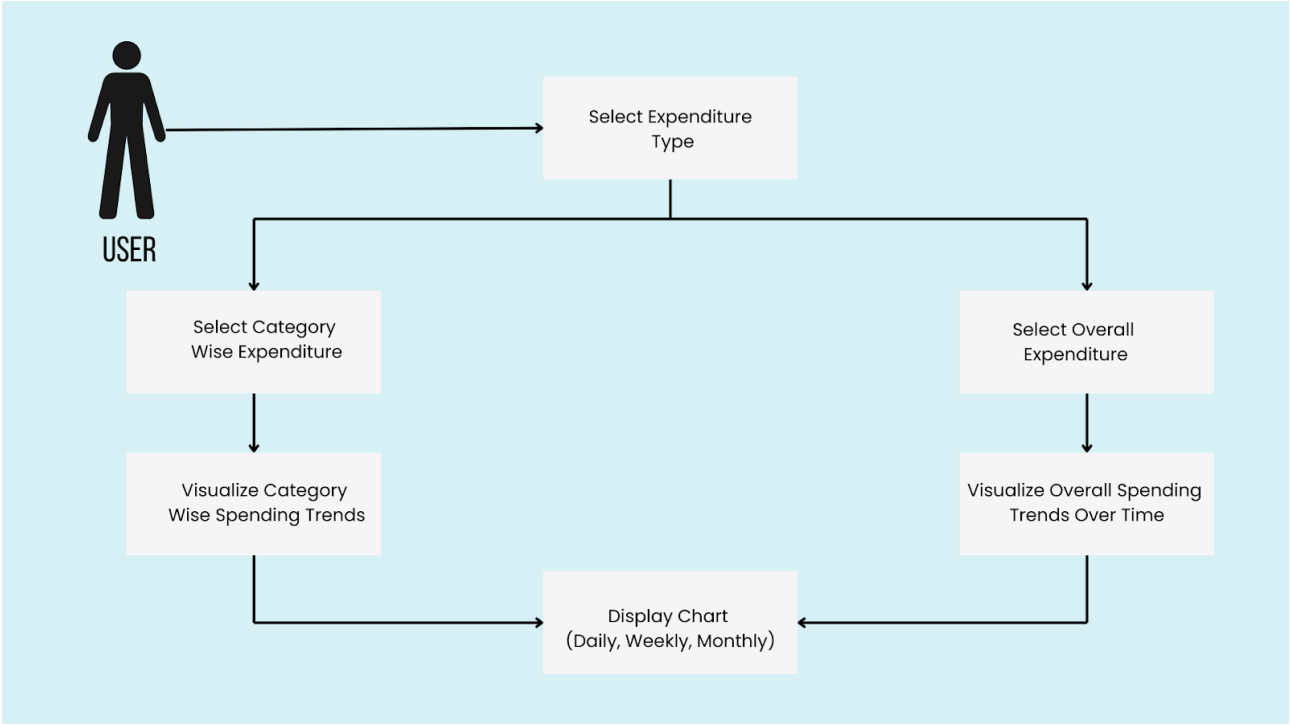
Requirements Traceability: The system provides users with visual representations of their overall and category-wise spending through charts. Users can filter expenditure data by category or timeframe for more precise insights..

Priority: Medium - A helpful visualization tool, but not central to financial management.

Preconditions: The system must have stored and categorized transaction data. The system must have a charting or visualization tool integrated into the application. Users must be able to apply filters for categories or timeframes for more accurate visualizations.

Post-conditions: The user can visualize expenditure trends via charts. Filters for categories or time frames are applied for better visual insights.

Actors: User, System, Transaction Database



3.1.8 Use Case #8: View and Manage Dues and Upcoming Bills/Payments (U8)

Authors: This use case was written by Aujasvit Datta and Bhavnoor Singh.

Purpose: To view and manage pending dues and upcoming bills.

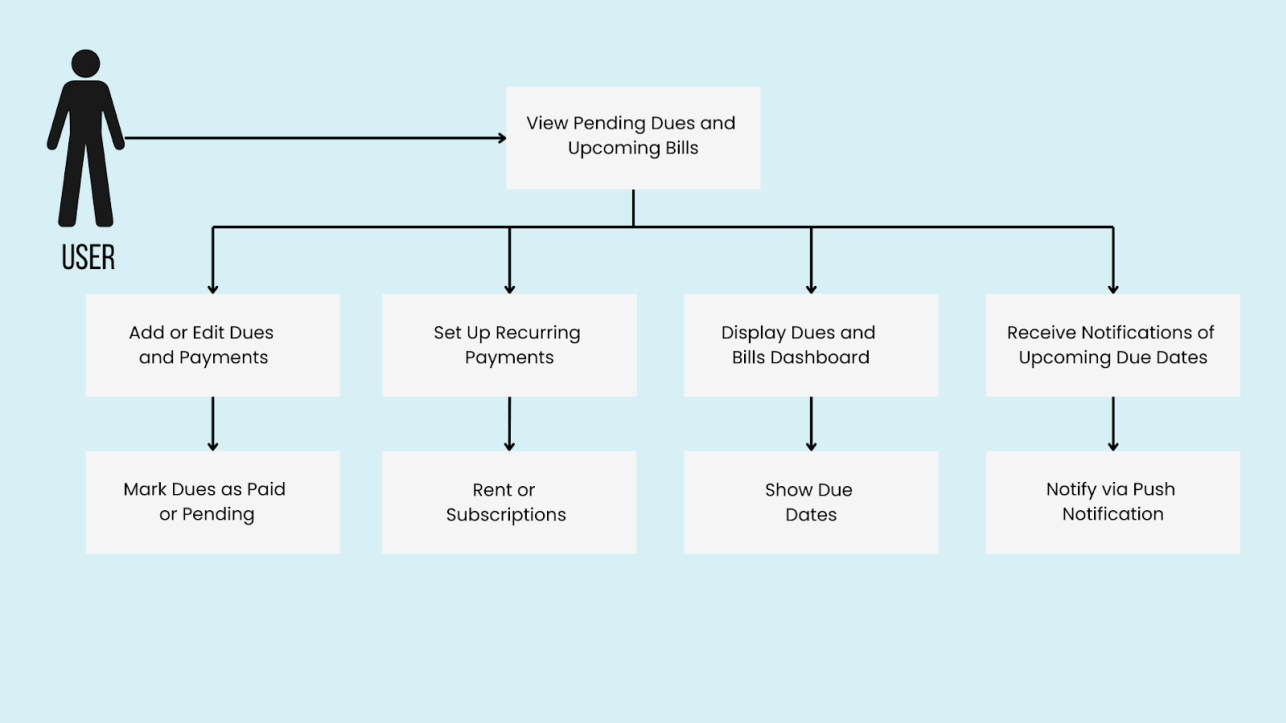
Requirements Traceability: The system displays a dashboard of pending dues and upcoming payments with due dates. Users receive push notifications, can manually add or edit payments, mark dues as paid or pending, and set up recurring payments (e.g., subscriptions, mess fees).

Priority: High - Crucial for managing bills and avoiding missed payments, impacting user financial health.

Preconditions: The system must have existing dues and bill data (either manually entered or imported). Recurring payments must be set up in the system if users wish to track them.

Post-conditions: Dues and bills are displayed on the dashboard. Users are notified of upcoming due dates. Recurring payments are tracked and notified appropriately.

Actors: User, System



3.1.9 Use Case #9: Split Bills and Track Debts (U9)

Authors: This use case was written by Darshan Sethia and Dhriti Barnwal.

Purpose: To split bills and track owed or borrowed amounts.

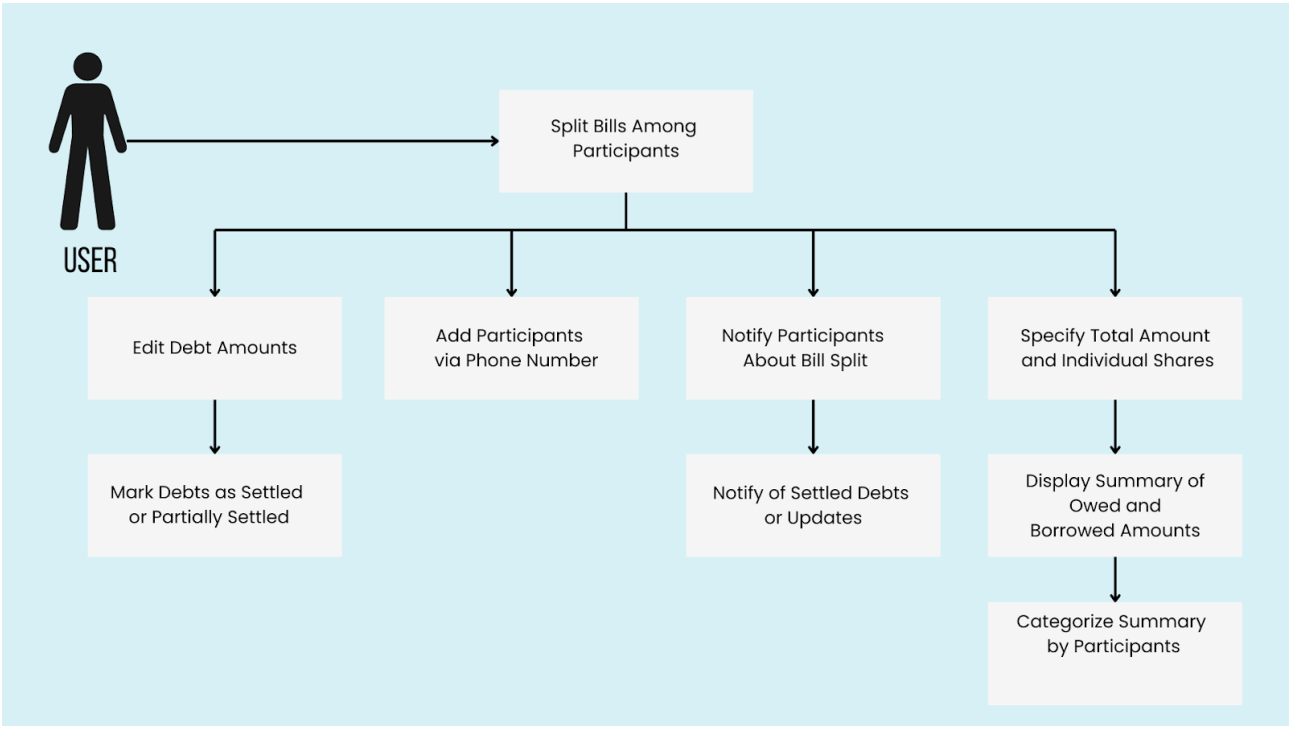
Requirements Traceability: The system uses phone numbers to manage transaction splits. Users can split new or existing transactions, view split history, track outstanding amounts, and record cash payments to settle debts. They can notify contacts for repayment and view their net balance (owed or due).

Priority: Medium - Helpful for managing shared expenses but not critical to core financial tracking.

Preconditions: The system must have an interface for splitting bills and managing debts. The system must have a notification system to inform participants about updates (email, SMS, or app notifications).

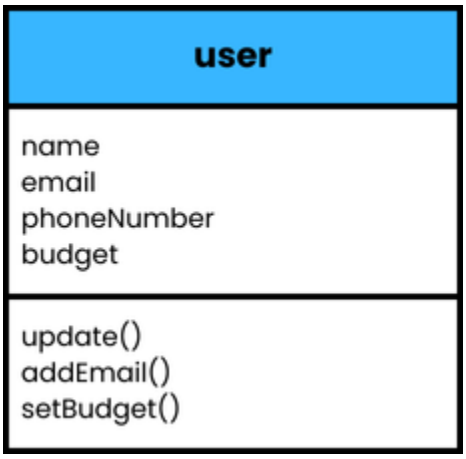
Post-conditions: Bills are split and tracked. Debt amounts are updated and participants are notified. Settled debts are marked appropriately.

Actors: Users, System



3.2 Class Details

1. **user class:**

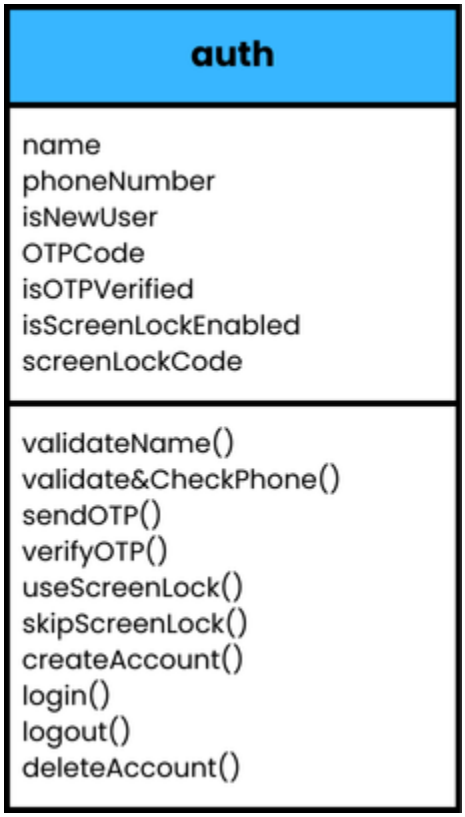


   - **Attributes**
     i. **name**: Name of the user
     ii. **email**: The email of the user, defaulted to none
     iii. **phoneNumbe**r: Phone number of the user
     iv. **budget**: Budget set by the user
   - **Methods**
     i. **update()**: Takes user data as arguments and updates the data
     ii. **addEmail()**: Takes the email of the user and verifies the email format and adds it
     iii. **setBudget()**: Takes a number as an argument and sets it as the budget of the user

2. **sms class:**

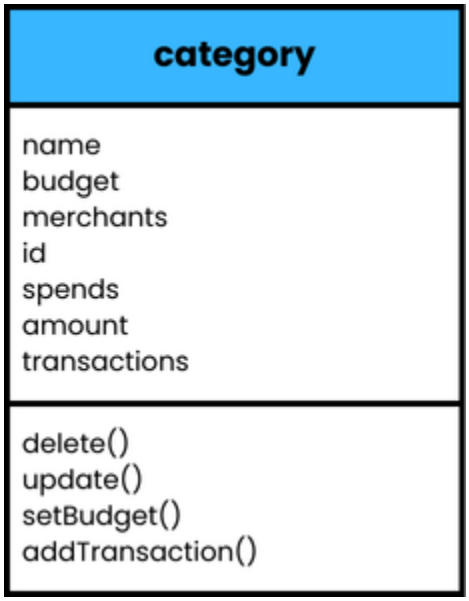
- **Attributes**
  i. **smsList**: Stores the raw relevant SMS messages fetched from the device
  ii. **filterKeywords**: A list of keywords used to identify spending-related SMS messages (e.g., "spent," "paid," "debited")
- **Methods**
  i. **fetch()**: Fetches SMS messages from the device
  ii. **filterSMS()**: Filters the SMS messages in smsList to identify those related to spending, using the keywords defined in filterKeywords and stores them in smsList
  iii. **parseSMS()**: Parses a spending-related SMS message into a structured Transaction object, extracting details such as amount, merchant and date
  iv. **checkPermissions()**: Checks whether the application has the necessary permissions to read SMS messages
  v. **requestPermissions()**: Requests SMS read permissions from the user if they haven't already been granted
  vi. **extractAmount()**: Extracts the transaction amount from the SMS text
  vii. **extractMerchant()**: Extracts the merchant or recipient name from the SMS text
  viii. **extractDate()**: Extracts the transaction date from the SMS text

3. **auth class:**



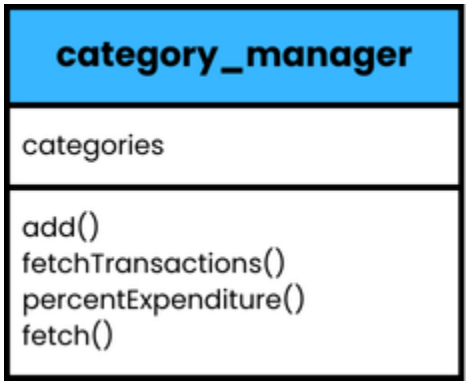
- **Attributes**
  i. **name**: Temporarily stores the user's name during the sign-up process, used to create the User object after all steps are completed
  ii. **phoneNumber**: Temporarily stores the user's phone number, checks if the user already exists, is used for OTP verification and creates the User object (if the user doesn't already exist)
  iii. **isNewUser**: Boolean flag indicating if the user already exists
  iv. **OTPCode**: Stores the OTP sent to the user's phone for verification, which expires after a certain period
  v. **isOTPVerified**: Boolean flag indicating whether the OTP has been successfully verified, ensuring the user cannot proceed without verification
  vi. **isScreenLockEnabled**: Boolean flag indicating whether the user has chosen to enable the phone screen lock feature. Defaults to false if the user skips this option
  vii. **screenLockCode**: Stores the phone screen lock code (e.g., PIN, pattern, or biometric data) if the user opts to enable screen lock. This is optional, and the app will function normally without it
- **Methods**
  i. **validateName()**: Validates the user's name ensuring it is not empty
  ii. **validate&CheckPhone()**: Validates the phone number, ensuring it is in the correct format and checks if the user already exists
  iii. **sendOTP()**: Sends an OTP to the user's phone using the SMSService and stores it in the otpCode attribute for verification
  iv. **verifyOTP()**: Compares the entered OTP with the stored otpCode. If they match, sets isOTPVerified to true
  v. **useScreenLock()**: Allows the user to enable the phone screen lock by setting a screenLockCode. This updates isScreenLockEnabled to true
  vi. **skipScreenLock()**: Allows the user to skip setting up the phone screen lock. This sets isScreenLockEnabled to false, and the app will function normally without it
  vii. **createAccount()**: Creates a new User object if the user doesn't exist using the validated tempName and tempPhoneNumber. If the phone screen lock is enabled, the screenLockCode is securely stored. If skipped, the app remains fully functional without the screen lock
  viii. **login()**: If the user already exists, log in if the OTP is verified
  ix. **logout()**: Logs out the user
  x. **deleteAccount()**: Deletes the account of the use

4. **category class:**



   - **Attributes**
     i. **name**: Stores the name of the category
     ii. **budget**: Budget of the category set by the user
     iii. **merchants**: List of merchants of this category
     iv. **id**: Unique identifier of the category
     v. **spends**: Number of transactions specific to this category
     vi. **amount**: Total expenditure which belongs to this category
     vii. **transactions**: List of transactions of this category
   - **Methods**
     i. **delete()**: Deletes the selected category
     ii. **update()**: Helps in modifying the attributes of this category
     iii. **setBudget()**: Allows the user to set a category-specific budget
     iv. **addTransaction()**: Add the transaction to the list of transactions and change other attributes accordingly

5. **category_manager class:**

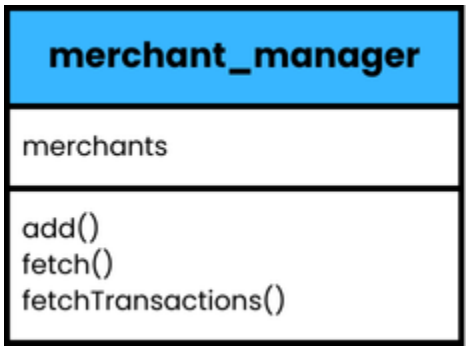


- **Attributes**
  i. **categories**: List of all category objects
- **Methods**
  i. **add()**: Adds a new category to the categories
  ii. **fetchTransactions()**: Fetches and returns transactions corresponding to a category
  iii. **percentExpenditure()**: Stores a mapping between categories and a percentage value calculated based on the total expenditure of the category
  iv. **fetch()**: Fetches and returns the list of categories

6. **merchant class:**

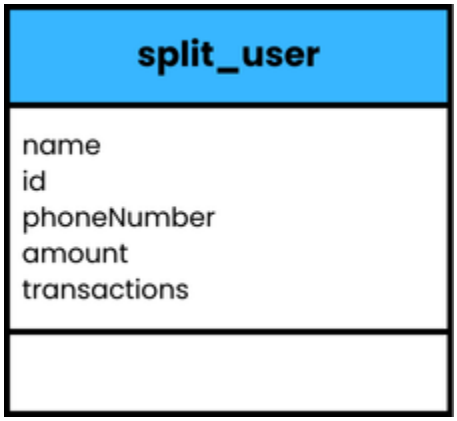
- **Attributes**
  i. **id**: Unique identifier for each merchant
  ii. **name**: Stores the name of the merchant
  iii. **category**: Stores the category to which the merchant belongs
  iv. **spends**: Number of transactions specific to this category
  v. **amount**: Total expenditure which belongs to this category
  vi. **transactions**: List of transactions of this category
- **Methods**
  i. **update()**: Updates the information about a merchant.
  ii. **displayTrends()**: Displays a trend of payments to the merchant for the given time-frequency

7. **merchant_manager class:**



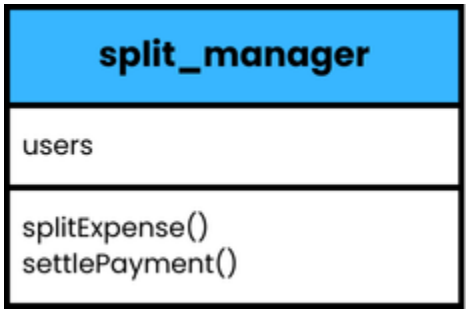
- **Attributes**
  i. **merchants**: List of all merchant objects
- **Methods**
  i. **add()**: Add a merchant
  ii. **fetch()**: Fetches and returns list of all merchants.
  iii. **fetchTransactions()**: Fetches and returns transactions corresponding to a merchant

8. **split_user class:**



- **Attributes**
  i. **name**: Name of the other user with whom there is a split
  ii. **id**: Unique identifier of the user
  iii. **phoneNumber**: Phone number of the other user
  iv. **amount**: Amount of money the other user owes or is owed from the original user
  v. **transactions**: List of transactions with the other user

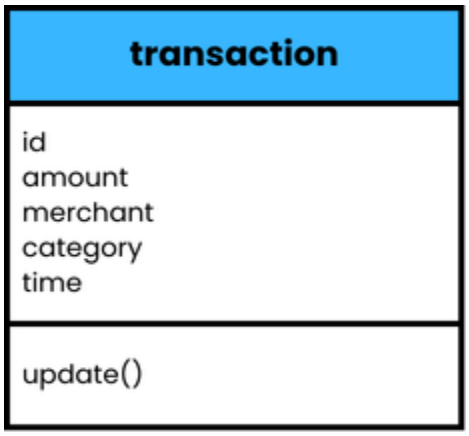
9. **split_manager class:**



- **Attributes**
  i. **users**: List of users of split user class with whom there is a split
- **Methods**
  i. **splitExpense()**: Takes a transaction ID, a list of users to split the transaction between and an identifier to define the type of split as arguments to split the transaction between those users

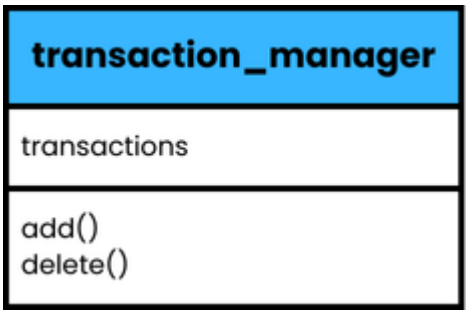
ii. **settlePayment()**: Settle up the amount partially or completely by taking as an argument the amount & record the payment in the list of transactions of that user

10. **transaction class**



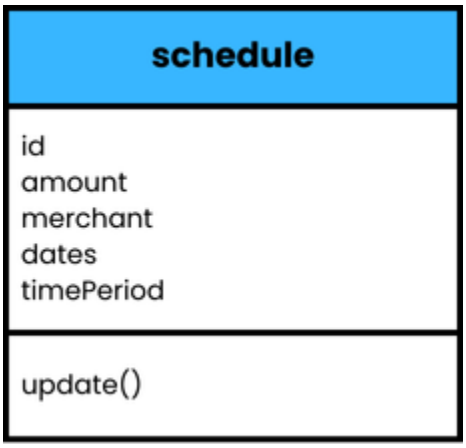
- **Attributes**
  - i. **id**: Unique identifier for each transaction
  - ii. **amount**: The amount involved in the transaction
  - iii. **merchant**: The merchant associated with the transaction
  - iv. **category**: The category under which the transaction falls
  - v. **time**: The timestamp of the transaction
- **Methods**
  - i. **update()**: Allows modification of transaction details

11. **transaction_manager class**



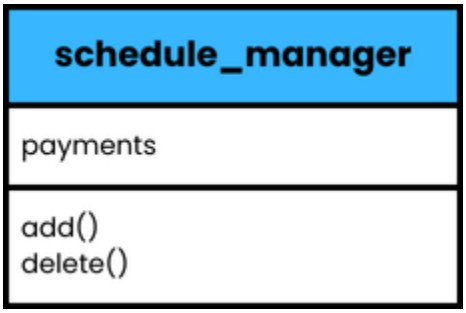
- **Attributes**
  - i. **transactions**: A list that stores multiple Transaction objects
- **Methods**
  - i. **add()**: Inserts a new transaction.
  - ii. **delete()**: Removes a transaction from the list

12. schedule class



- **Attributes**
  i. **id**: Unique identifier of the recurring payment
  ii. **amount**: Object that stores information related to a particular payment
  iii. **merchant**: Stores the details of the recipient
  iv. **dates**: Stores the dates payments have be made
- **Methods**
  i. **update()**: Takes modified payment details as input and modifies the recurring payment

13. schedule_manager class

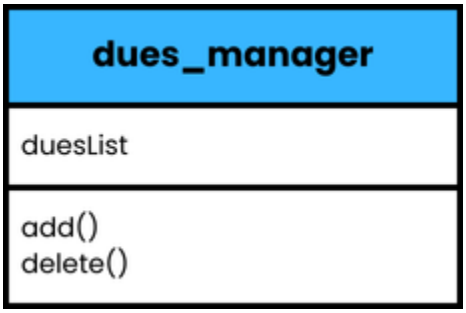


- **Attributes**
  i. **payments**: List of all scheduled payment objects
- **Methods**
  i. **add()**: Takes payment details as input and creates a new payment id
  ii. **delete()**: Takes payment id as input and allows deletion of an existing recurring payment

14. dues class

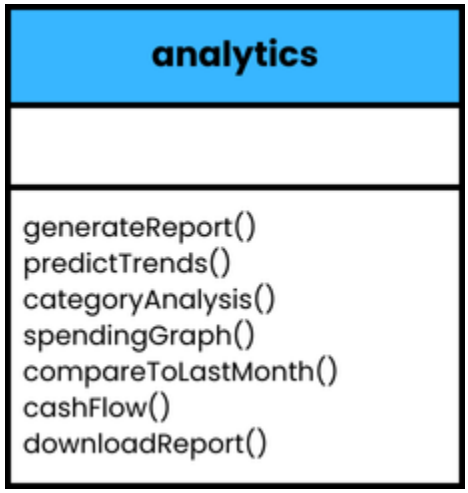
- **Attributes**
  i. **id**: Unique identifier of the due
  ii. **amount**: Object that stores information related to a particular due
  iii. **merchant**: Stores the information of the recipient
  iv. **date**: The date before which the due is to be paid
  v. **paidStatus**: Boolean representing if the dues is paid
- **Methods**
  i. **togglePaidStatus()**: Toggles paid_status
  ii. **update()**: Modifies a due item to the dues list by taking in the due id and new details

15. dues_manager class



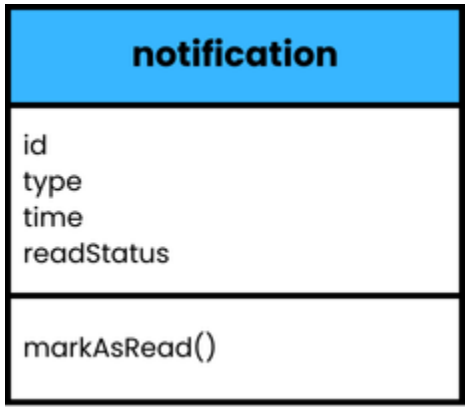
- **Attributes**
  i. duesList: List of all category objects
- **Methods**
  i. add(): adds a due item to the dues list by taking in the due details and assigning a due id
  ii. delete(): Deletes an existing due entry of the provided due id

16. analytics class



- **Methods**
  i. **generateReport()**: Produces spending reports for the past week, month, year, or a user-specified custom duration
  ii. **predictTrends()**: Uses past data to estimate future expenses
  iii. **categoryAnalysis()**: Analyzes spending in a specific category
  iv. **spendingGraph()**: Generates a category-wise spending graph for visualization
  v. **compareToLastMonth()**: Compares the current month's spending with the previous month's to highlight changes
  vi. **cashFlow()**: Calculates credit vs. debit transactions and compares them to the budget
  vii. **downloadReport()**: Downloads the generated report

17. notification class

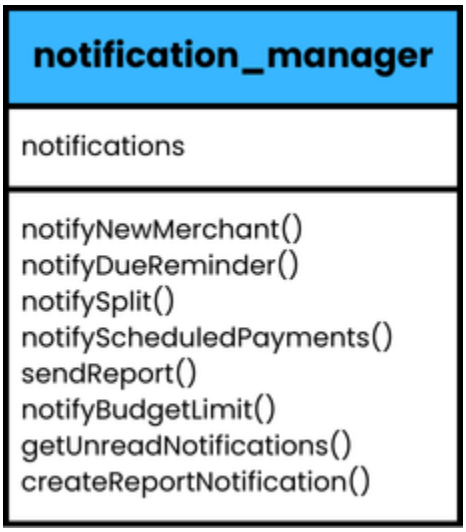


- **Attributes**
  i. **id**: Unique identifier for the notification
  ii. **type**: The system component that triggered the notification (e.g., "Budget Tracker", "Due Reminder")
  iii. **time**: Timestamp of when the notification was sent

iv. **sms:** The sms text from which the transaction object has been created
v. **readStatus**: Boolean flag (True/False) indicating if the notification has been read

- **Methods**
  i. **markAsRead()**: Set read_status to true

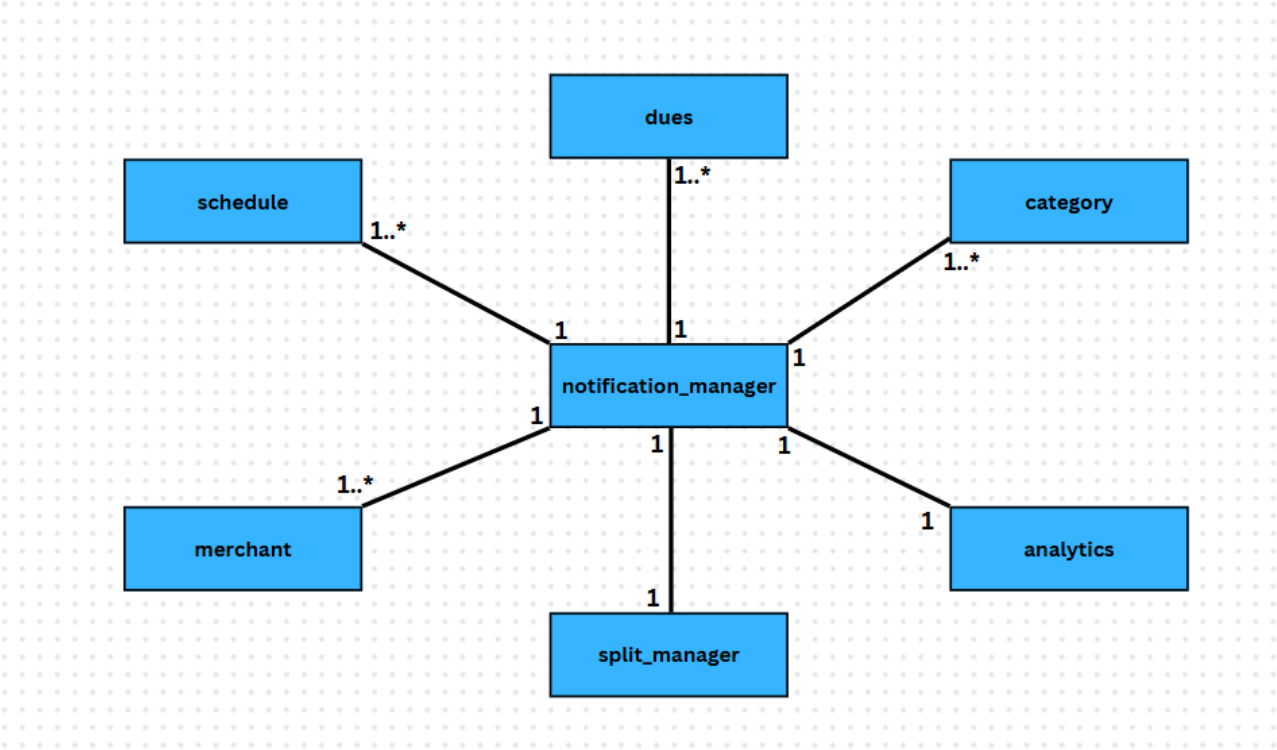
18. notification_manager class



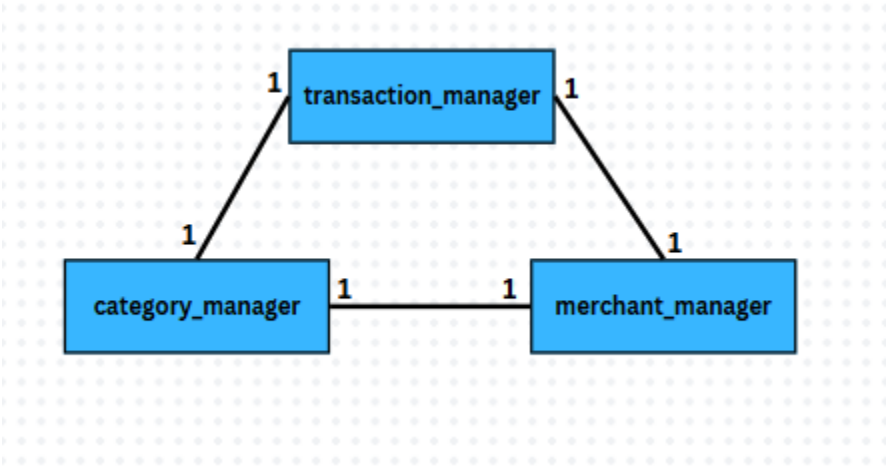
- **Attributes**
  i. **notifications**: List of notifications of class notification
- **Methods**
  i. **notifyNewMerchant()**: Alerts user to categorise a new merchant
  ii. **notifyDueReminder()**: Sends reminders for unpaid dues
  iii. **notifySplit()**: Sends a notification when a user creates a split
  iv. **notifyScheduledPayments()**: Sends reminders for recurring payments
  v. **sendReport()**: Sends a financial report (weekly, monthly, yearly)
  vi. **notifyBudgetLimit()**: Notifies the user when spending in a category reaches 50%, 90%, or 100% of the budget
  vii. **getUnreadNotifications()**: Returns all unread notifications
  viii. **createReportNotification()**: Sends a notification when the report is ready

3.3 Class Diagrams

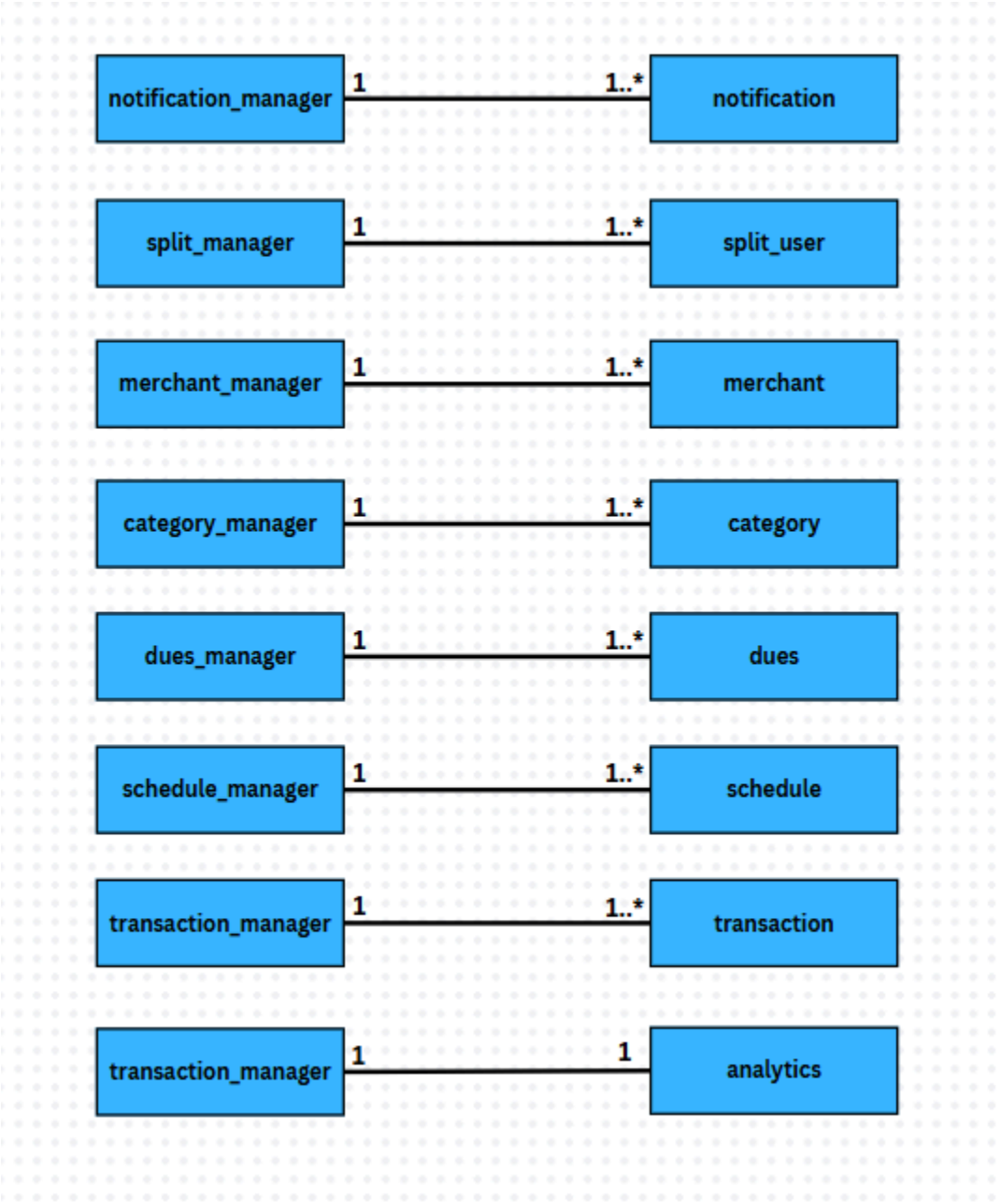
3.3.1 Class Diagram 1



3.3.2 Class Diagram 2

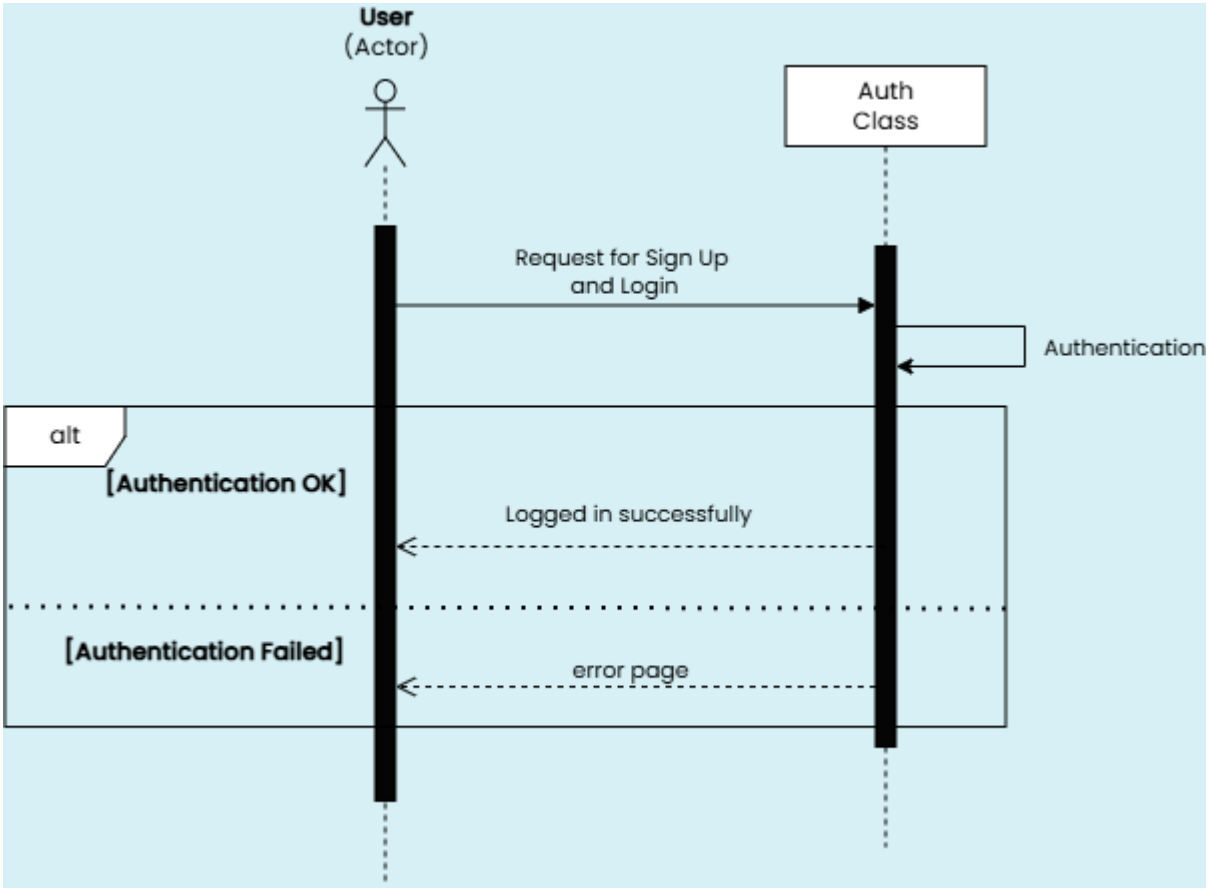


3.3.3 Class Diagram 3

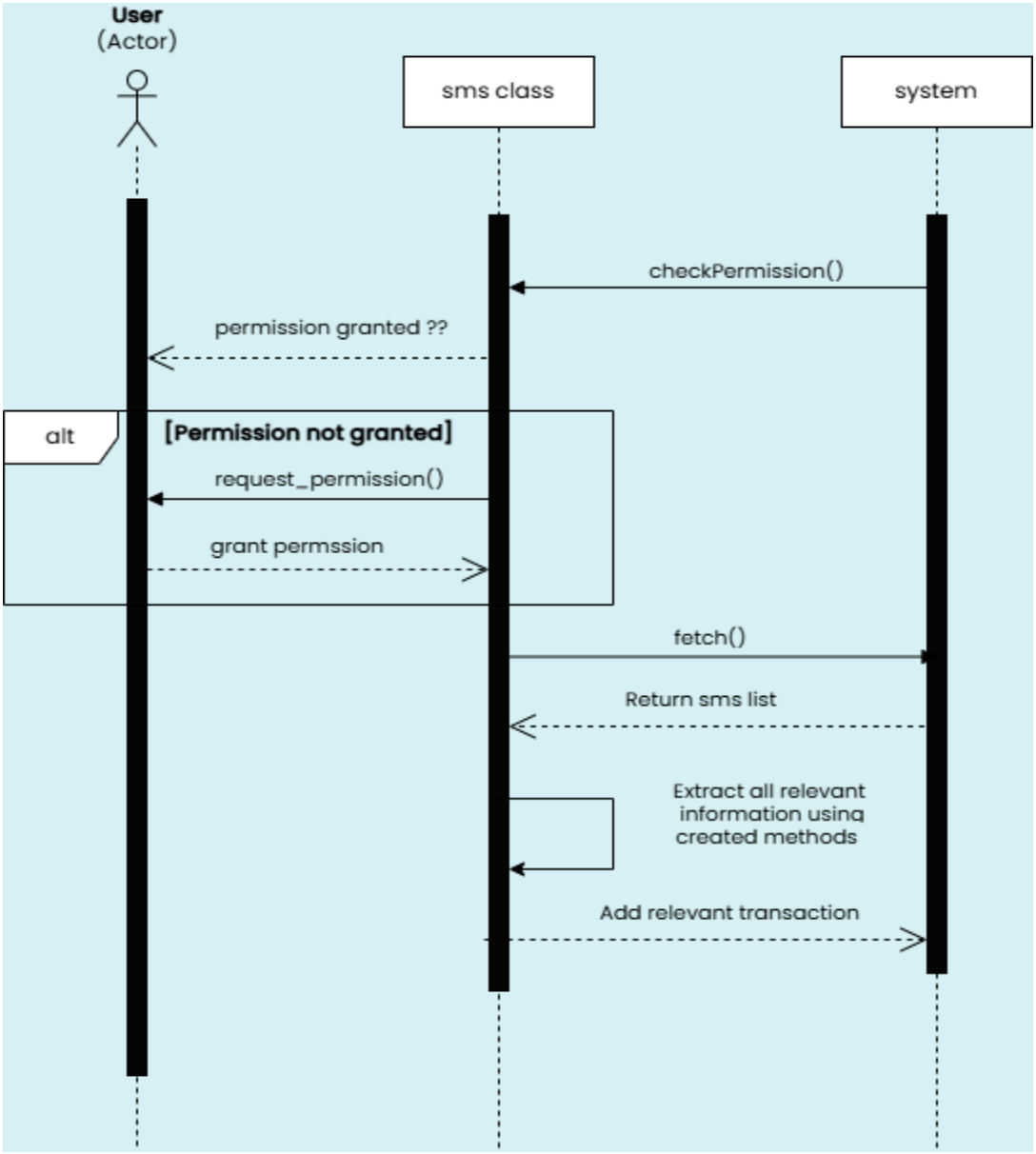


3.4 Sequence Diagrams

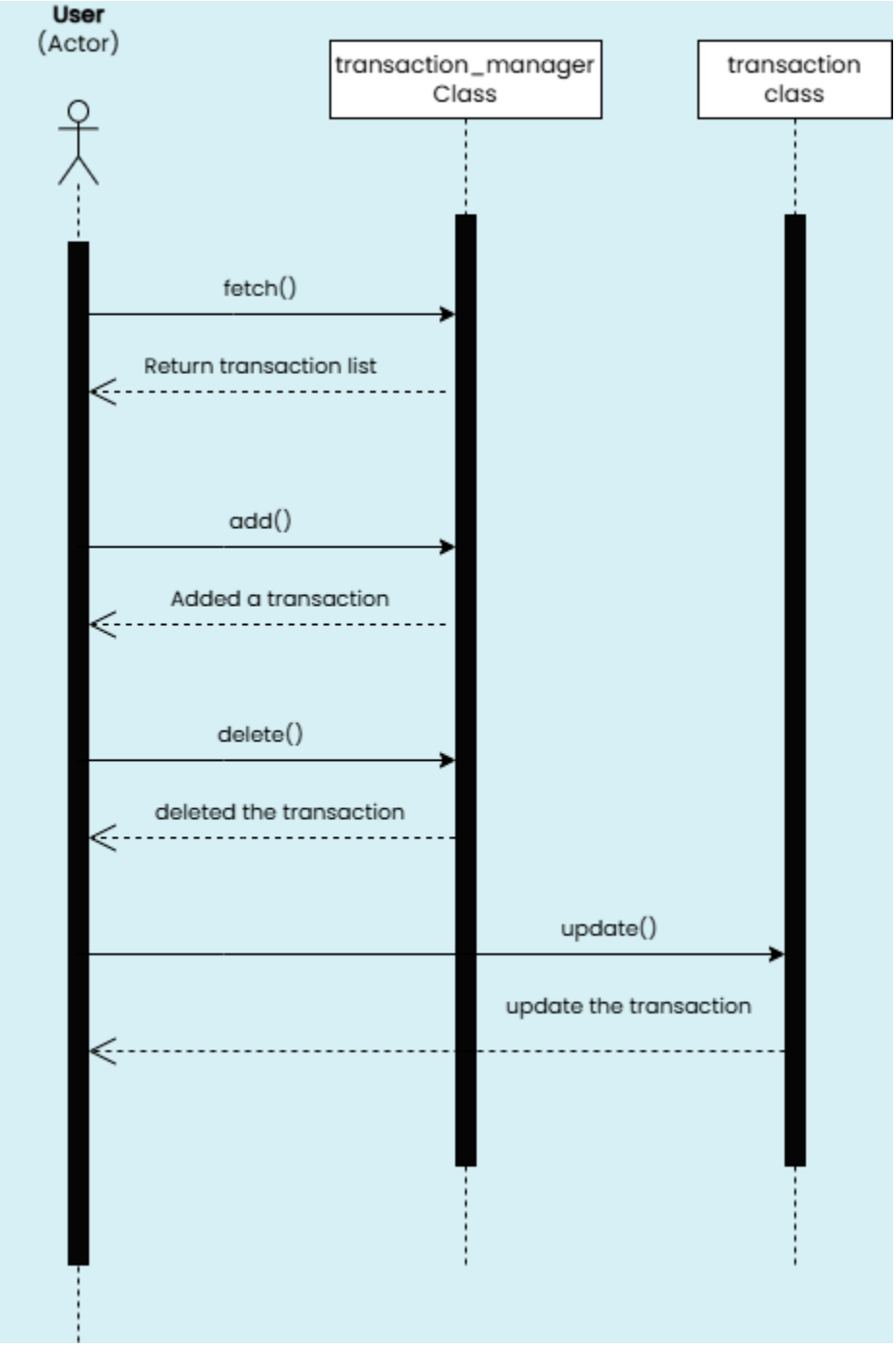
3.4.1 Login and Sign Up



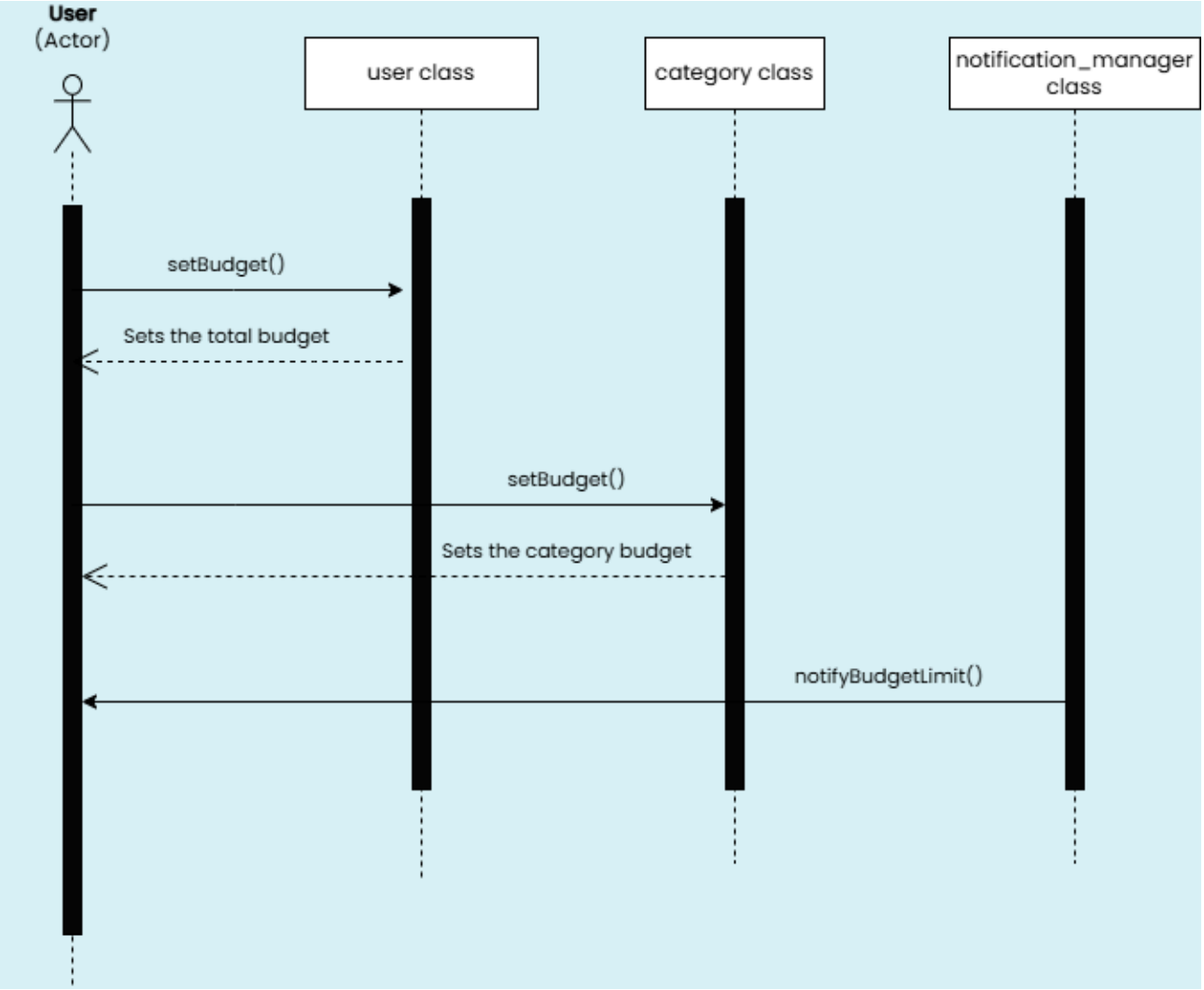
3.4.2 SMS Parsing



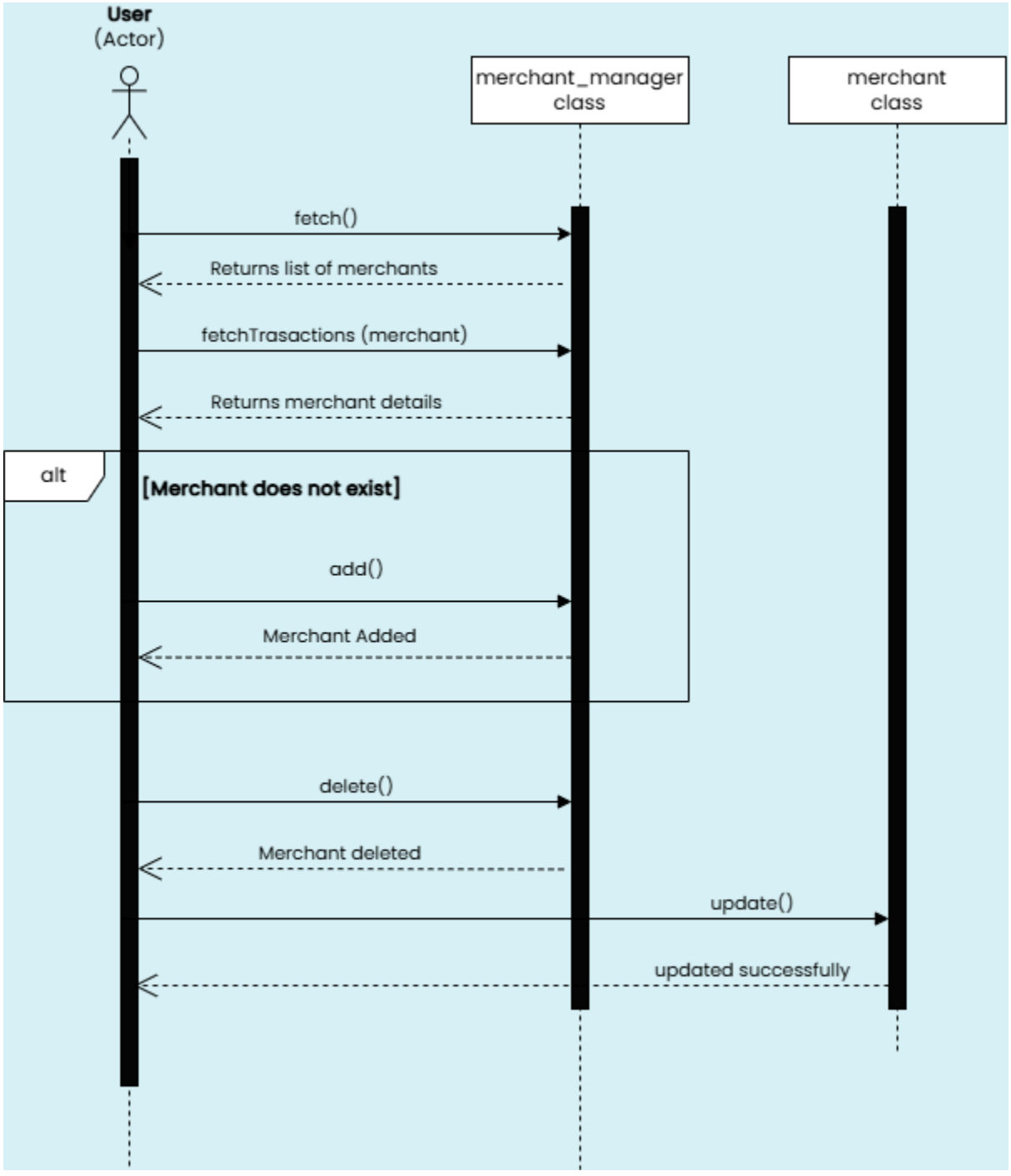
3.4.3 Managing Transactions



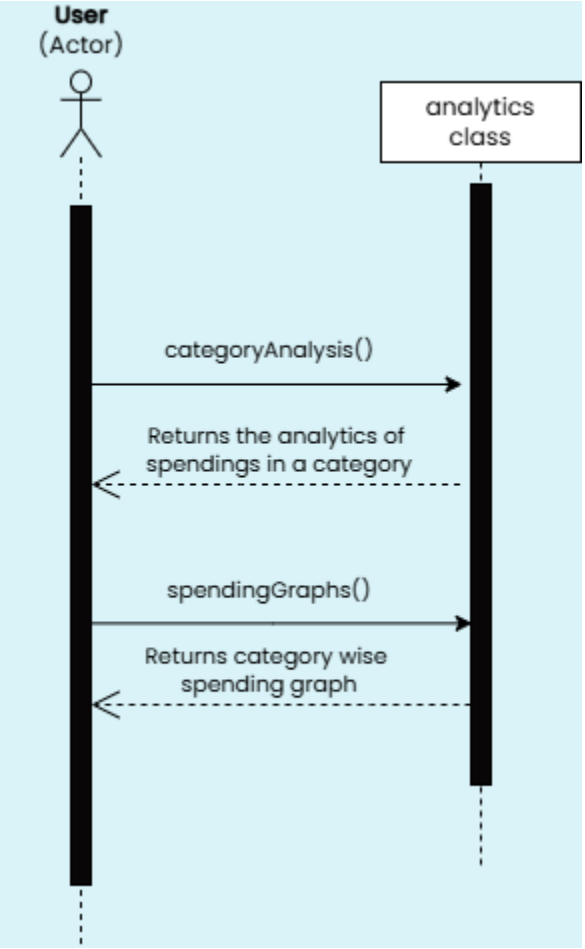
3.4.4 Budgeting Operations:



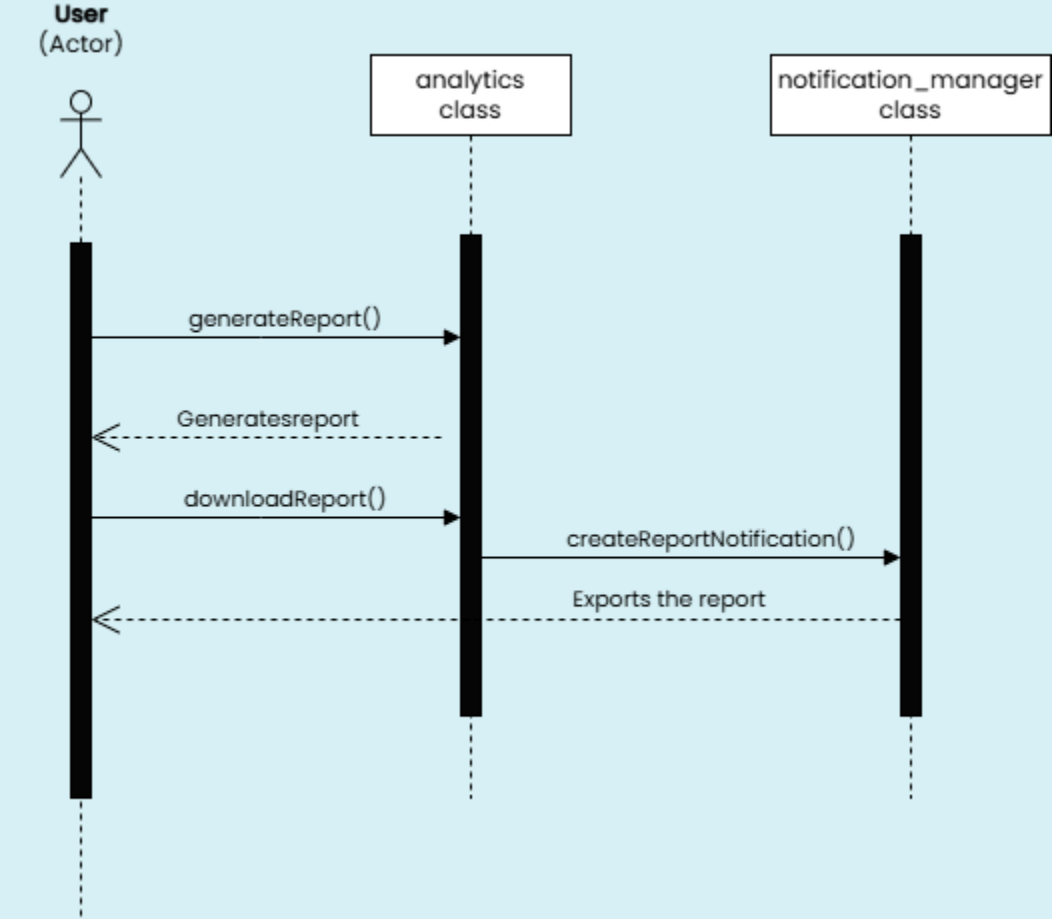
3.4.5 Managing Merchants:



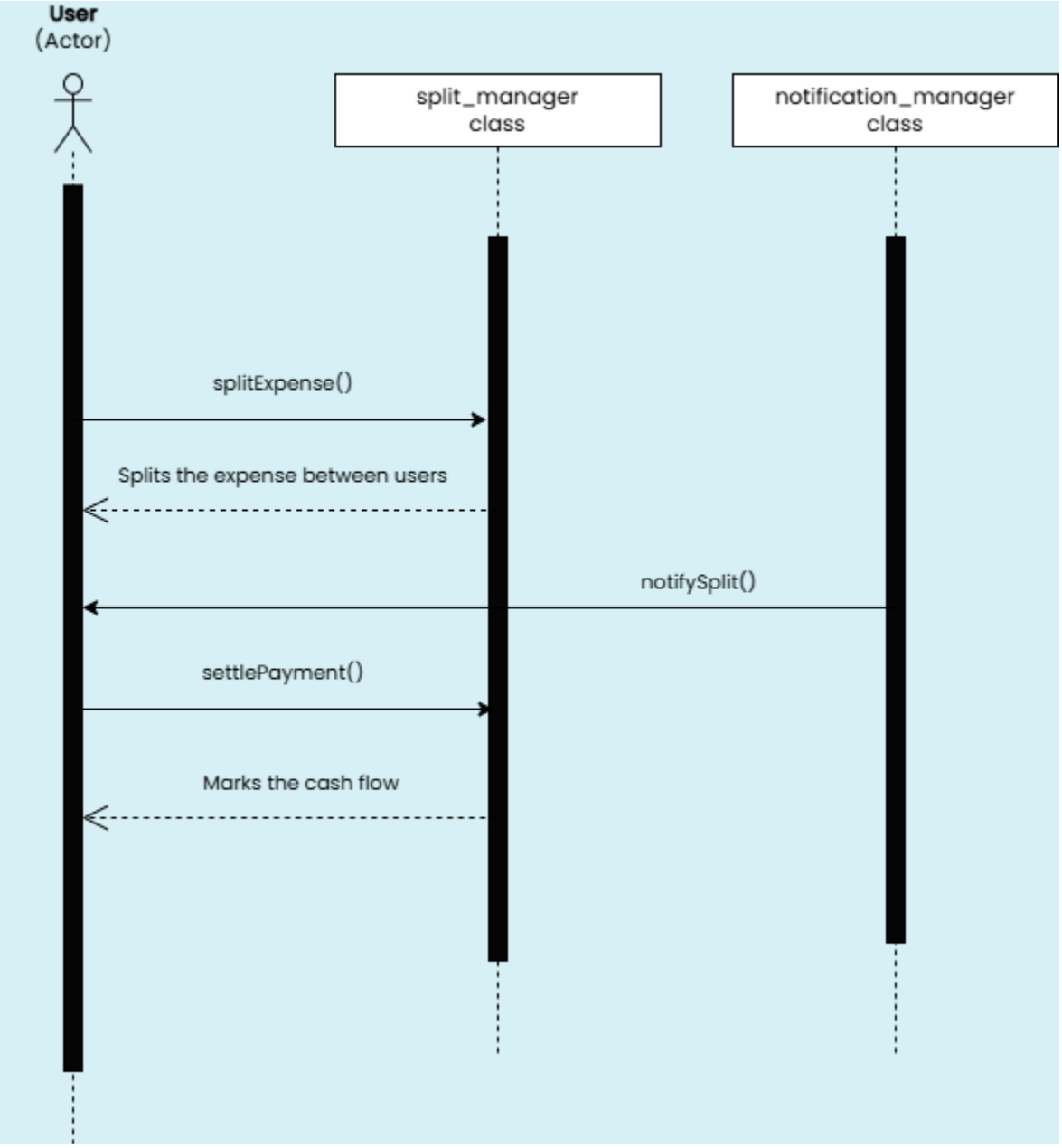
3.4.6 Category Analysis:



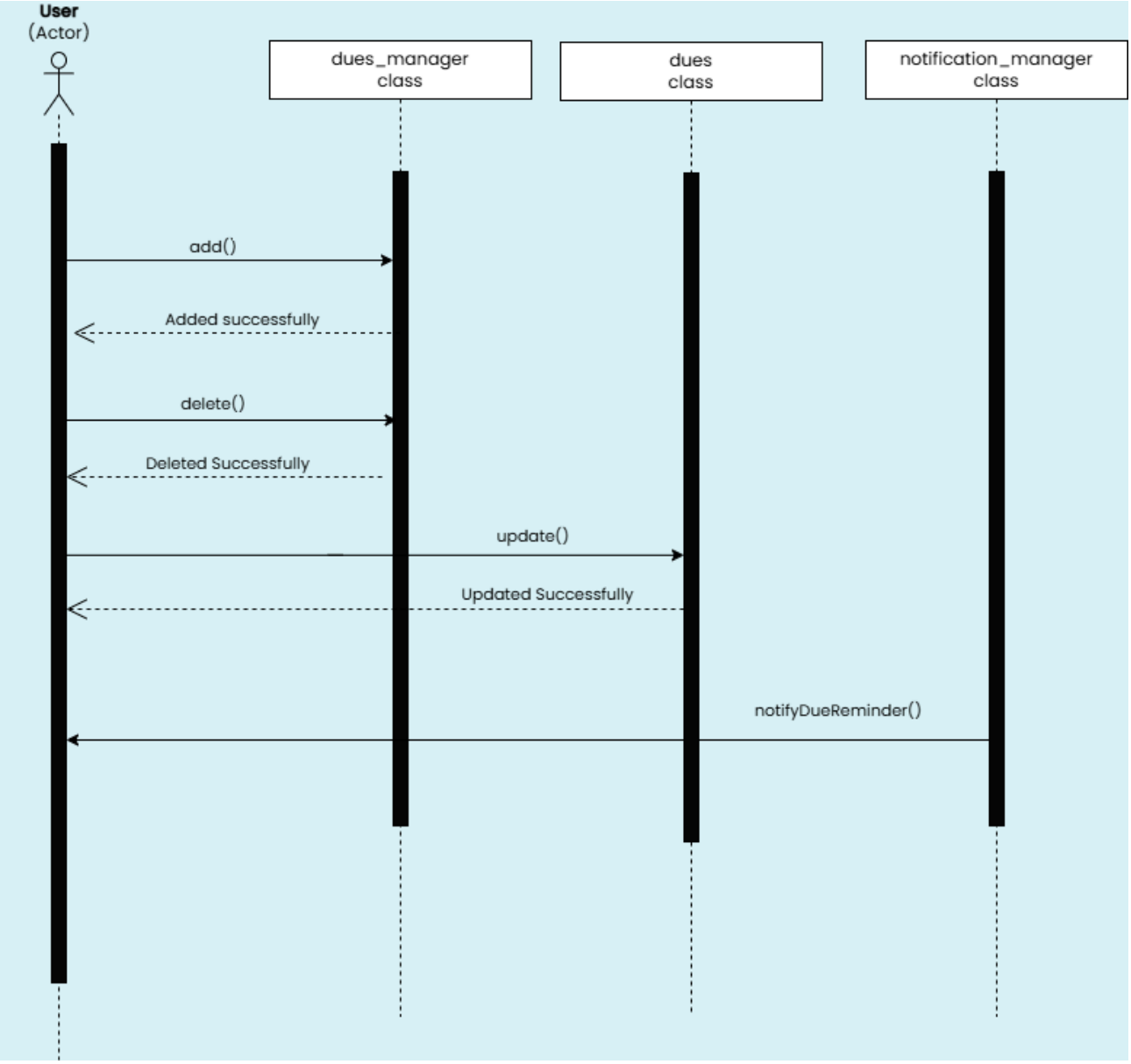
3.4.7 Generation of Reports:



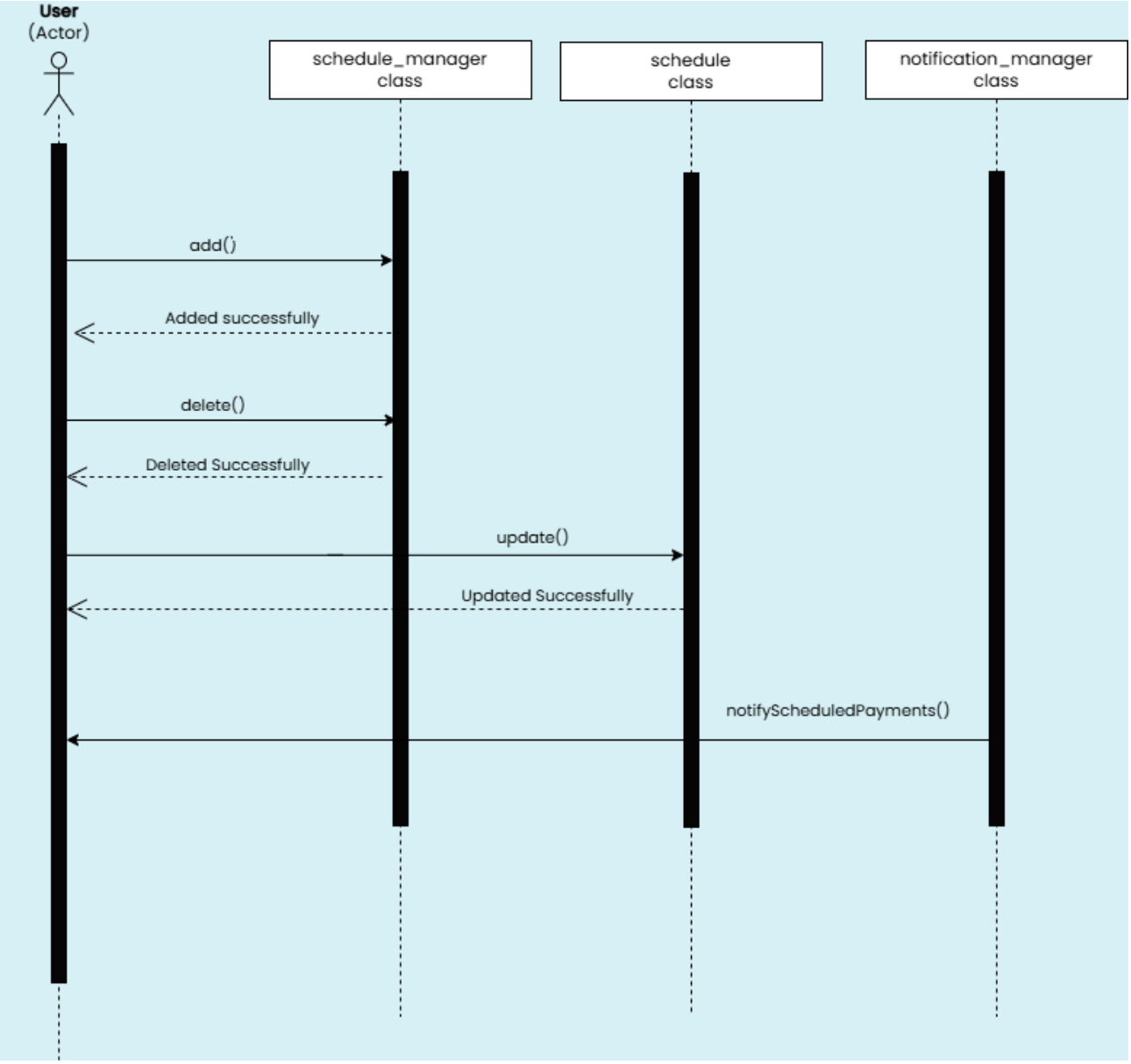
3.4.8 Creating splits:



3.4.9 Managing Dues:

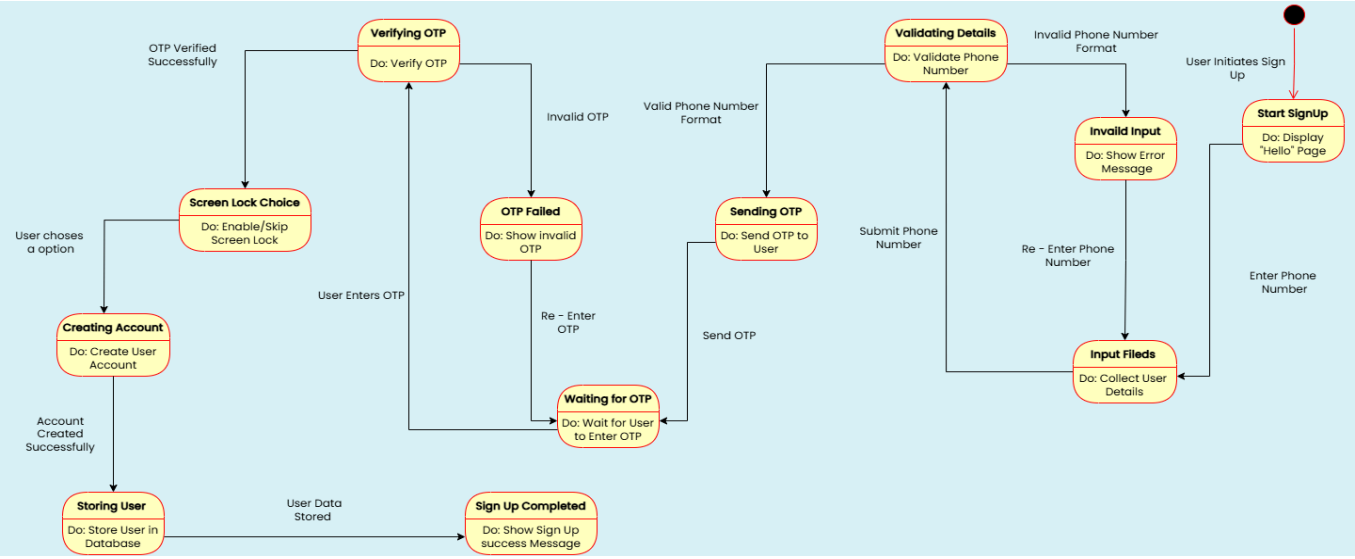


3.4.9 Managing Scheduled Payments:

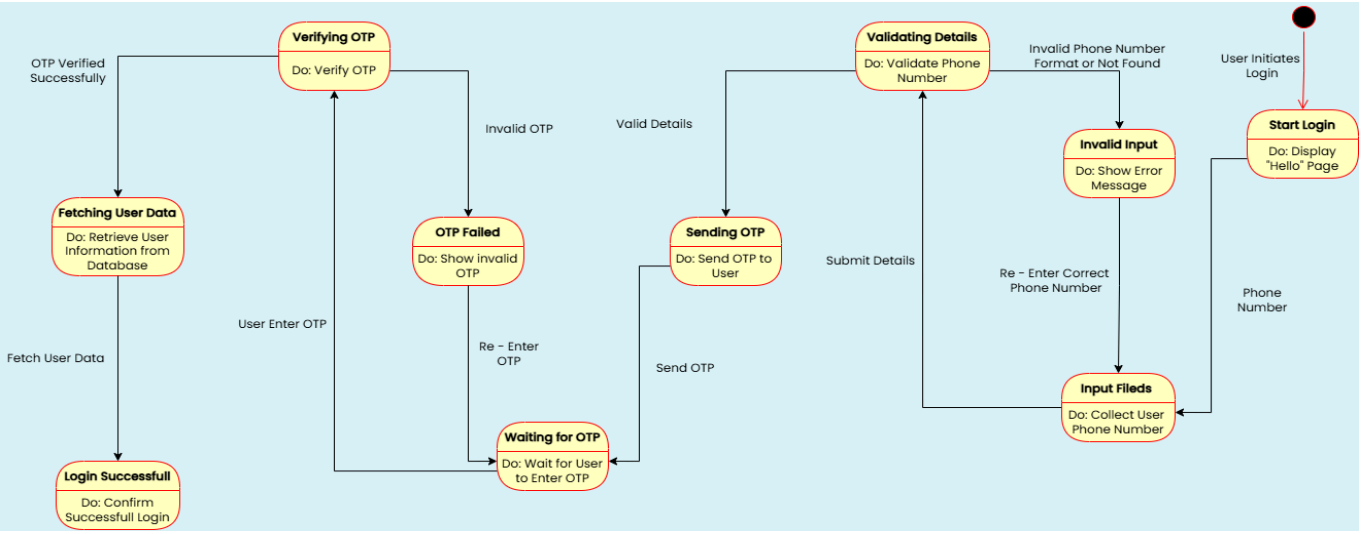


3.5 State Diagrams

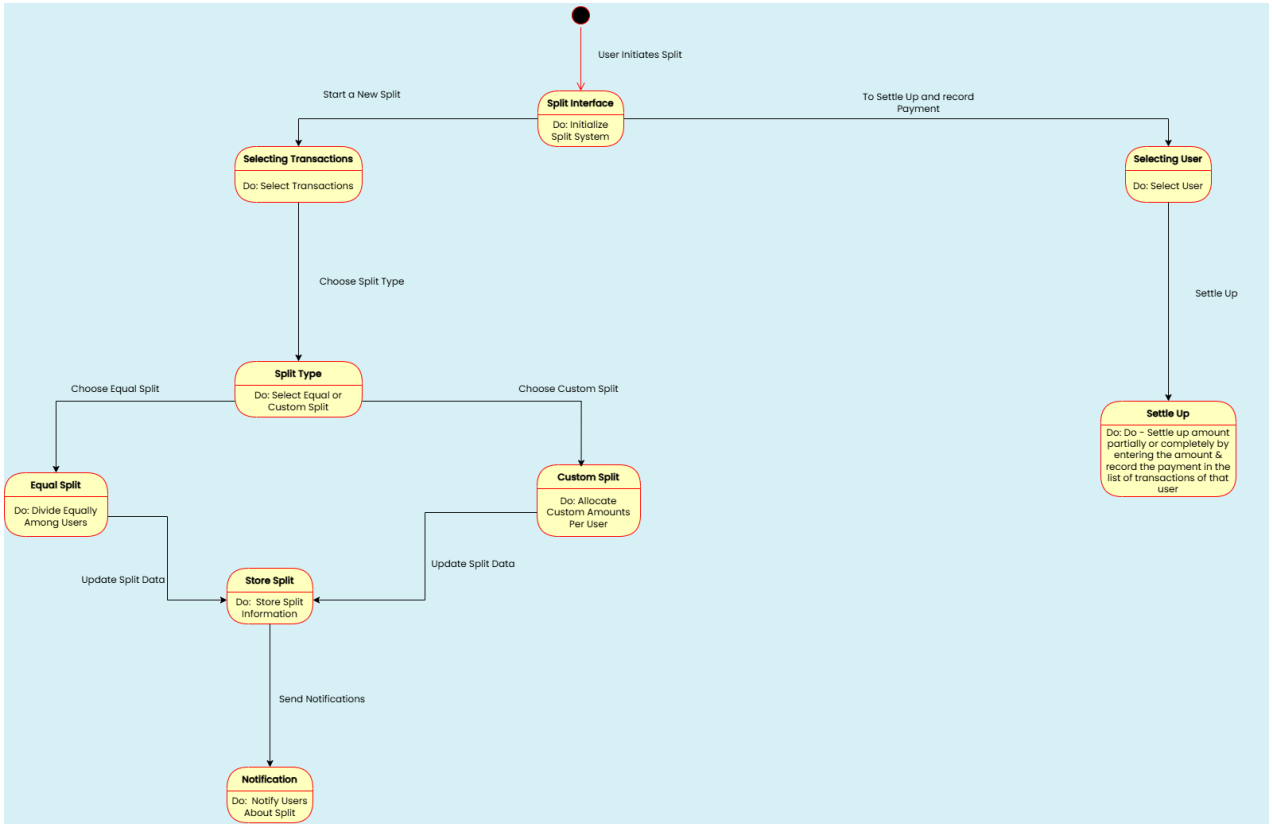
3.5.1 Navigating the Sign Up function:



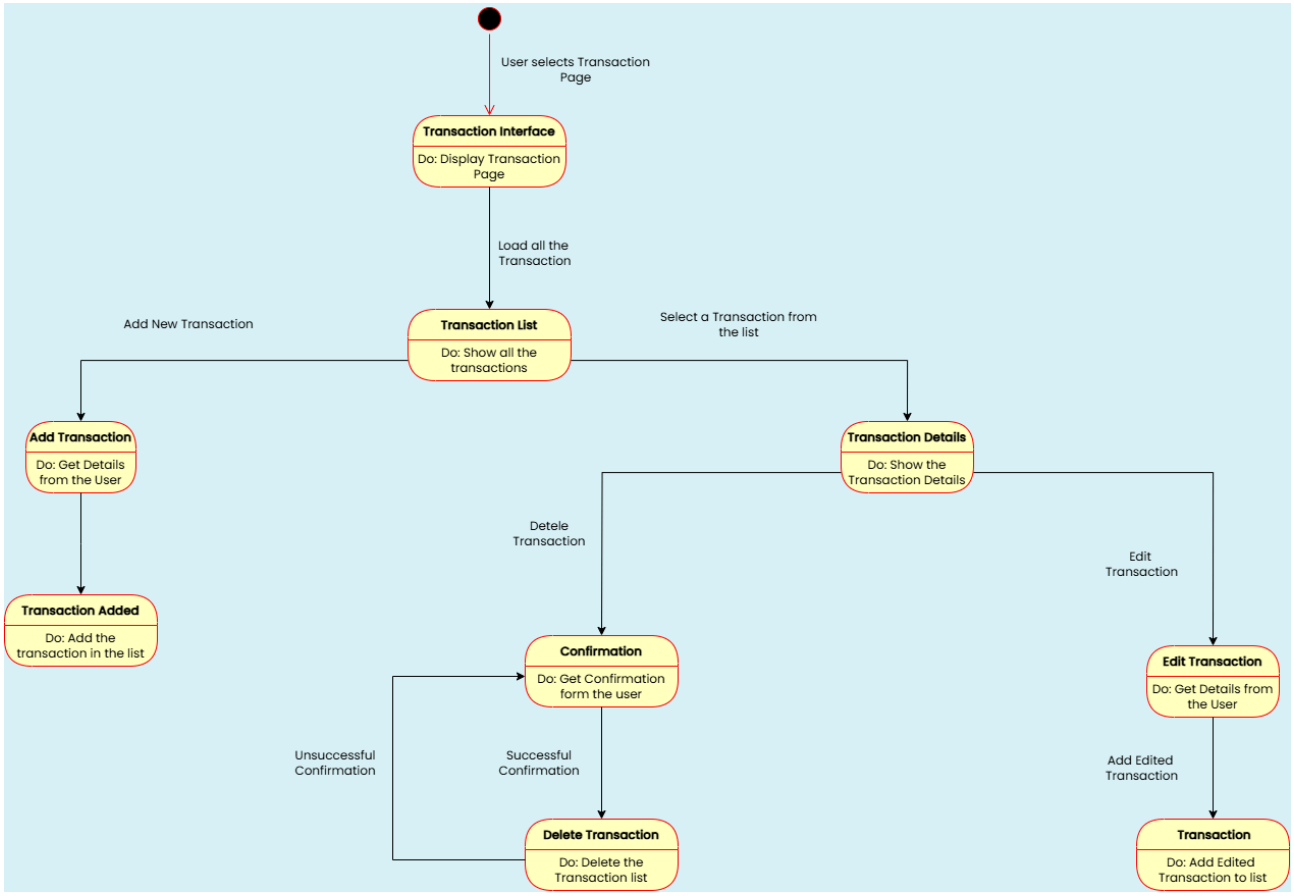
3.5.2 Navigating the login function:



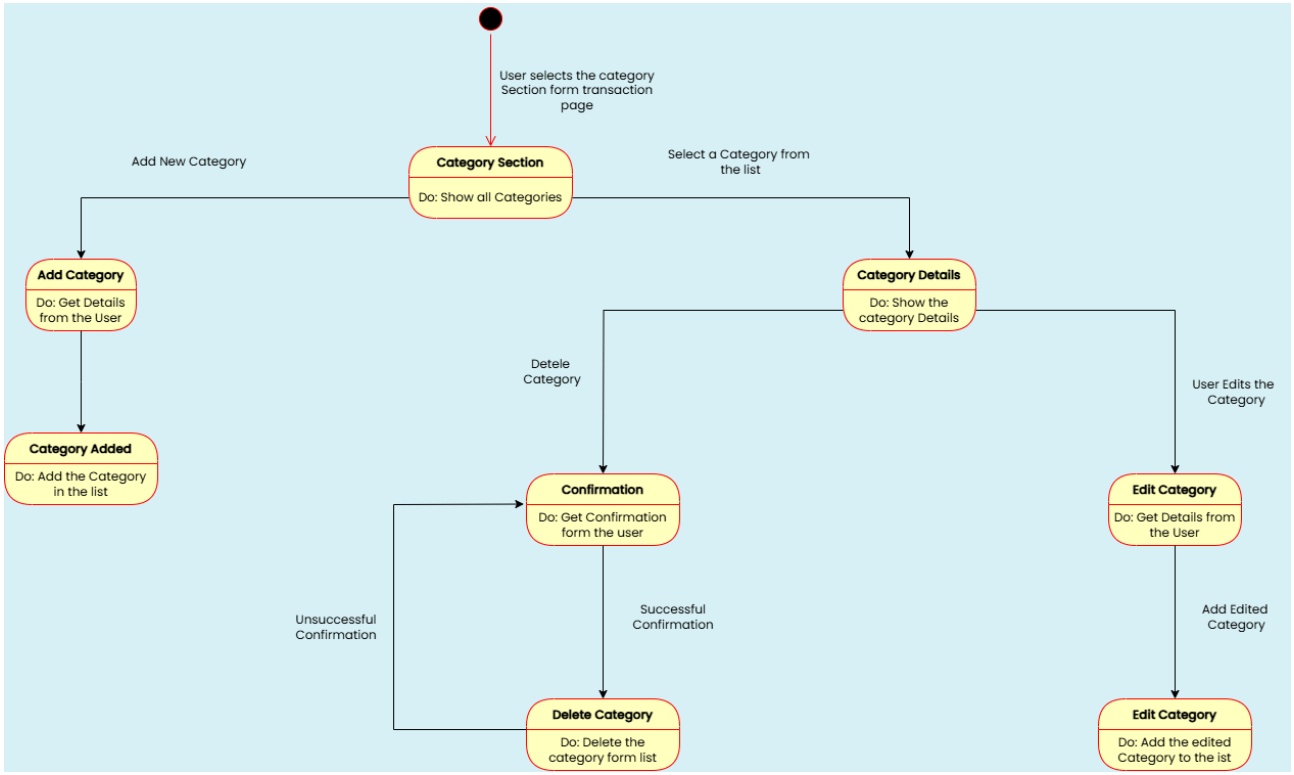
3.5.3 Navigating through the Split Page:



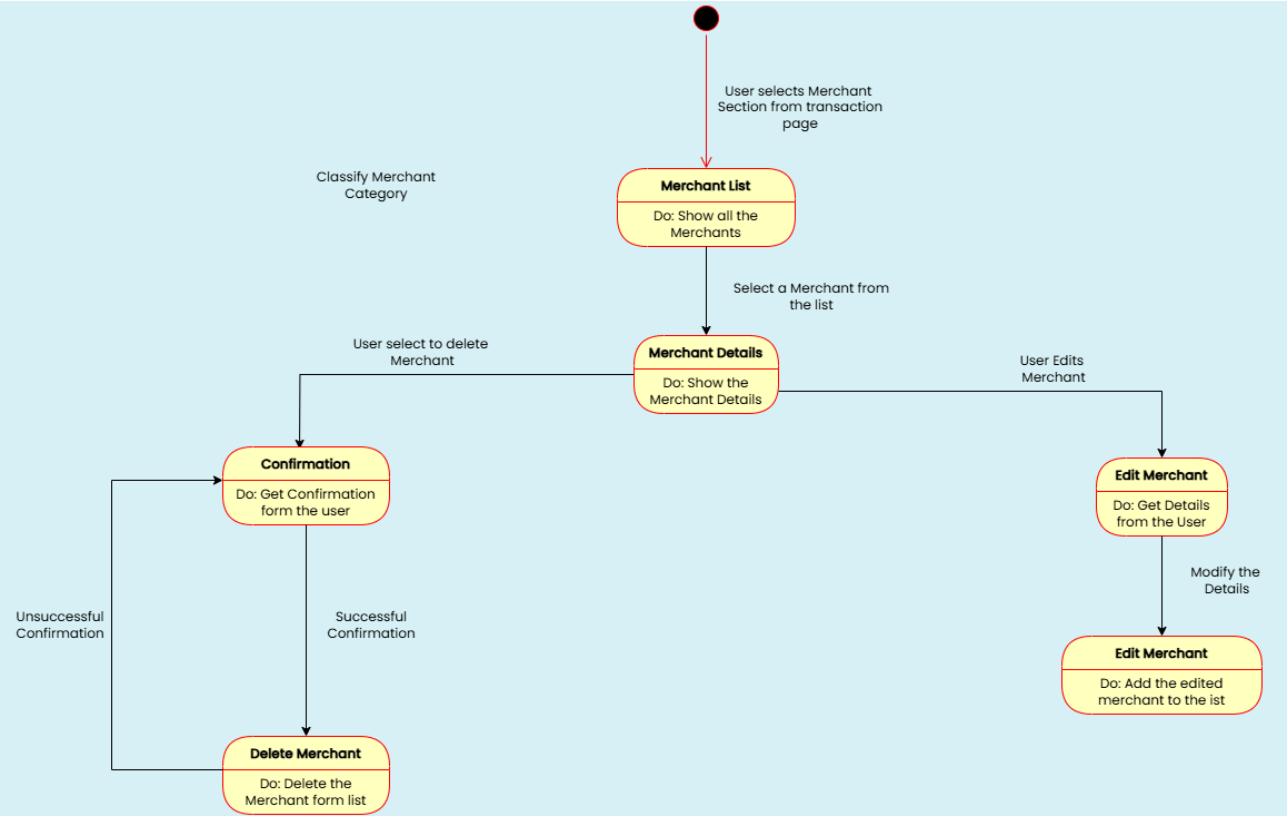
3.5.4 Navigating the Transactions sub-section:



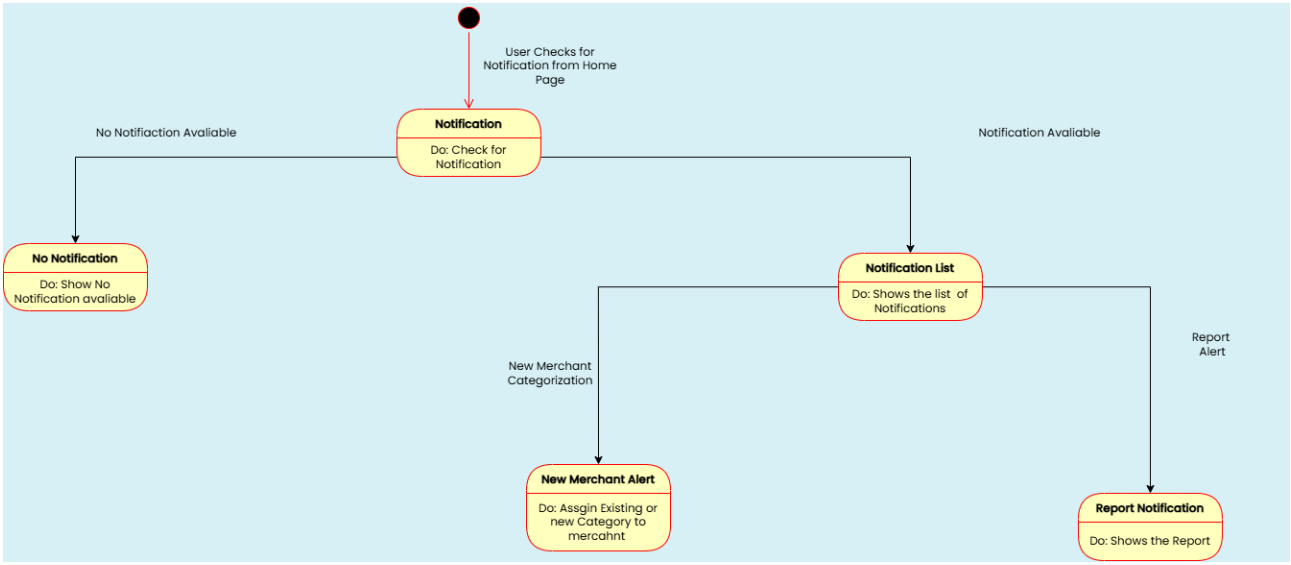
3.5.5 Navigating the Category sub-section:



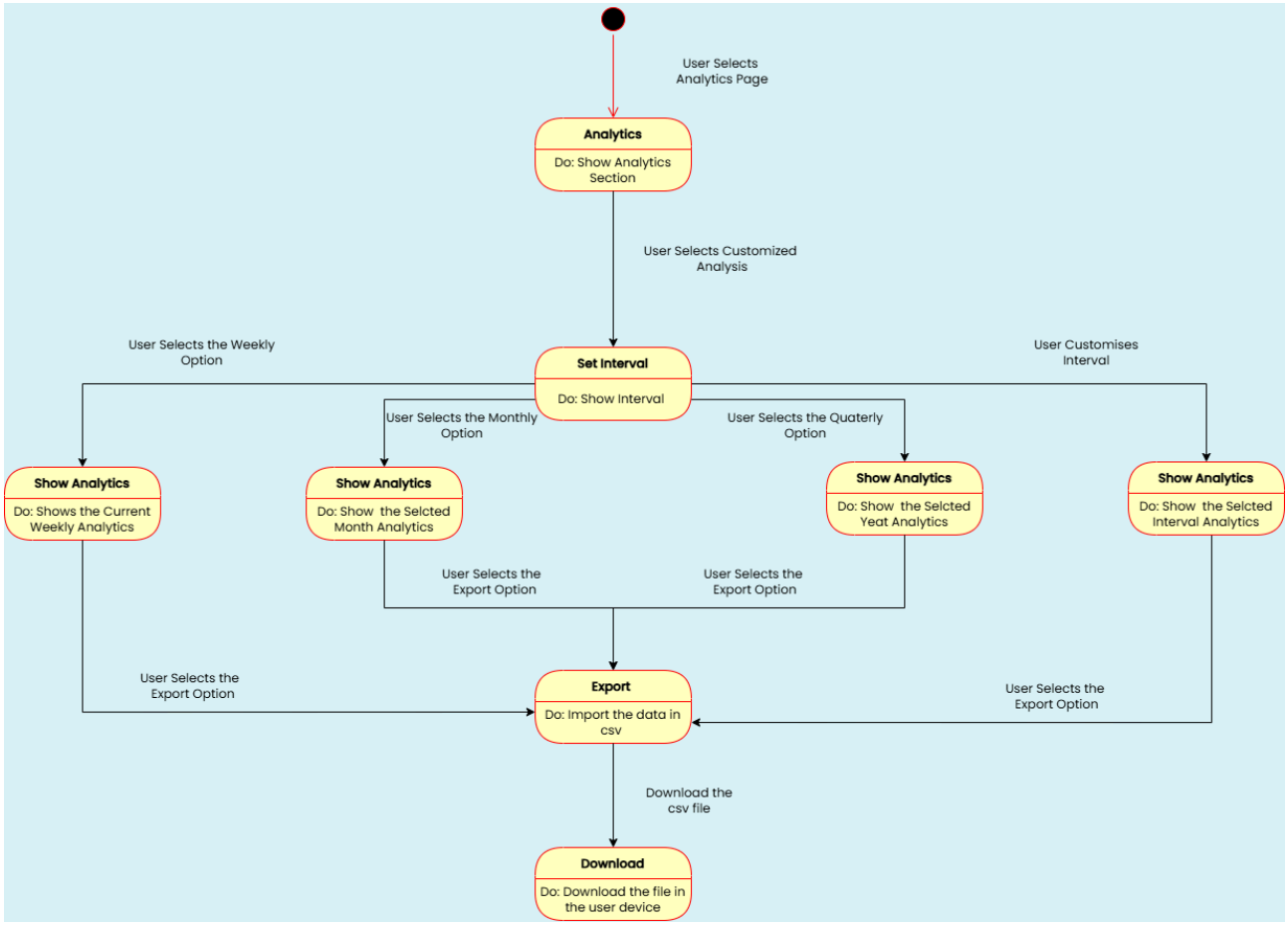
3.5.6. Navigating through the Merchants Sub-section:



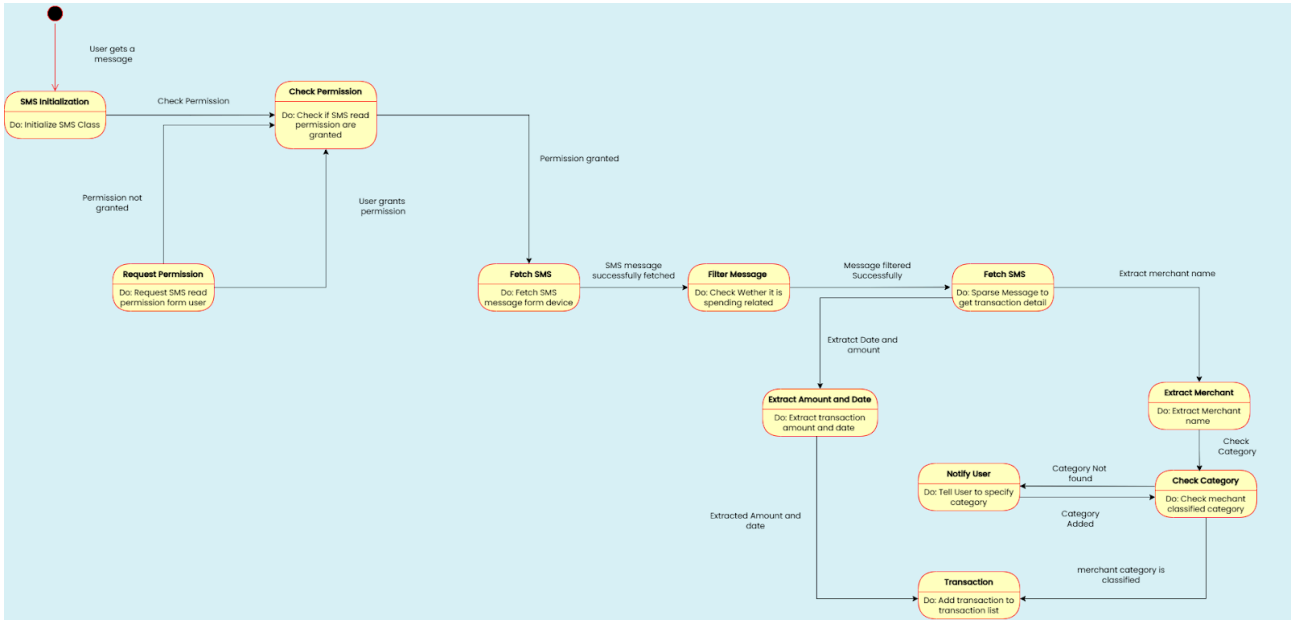
3.5.7 Navigating through the notification option of the Home page:



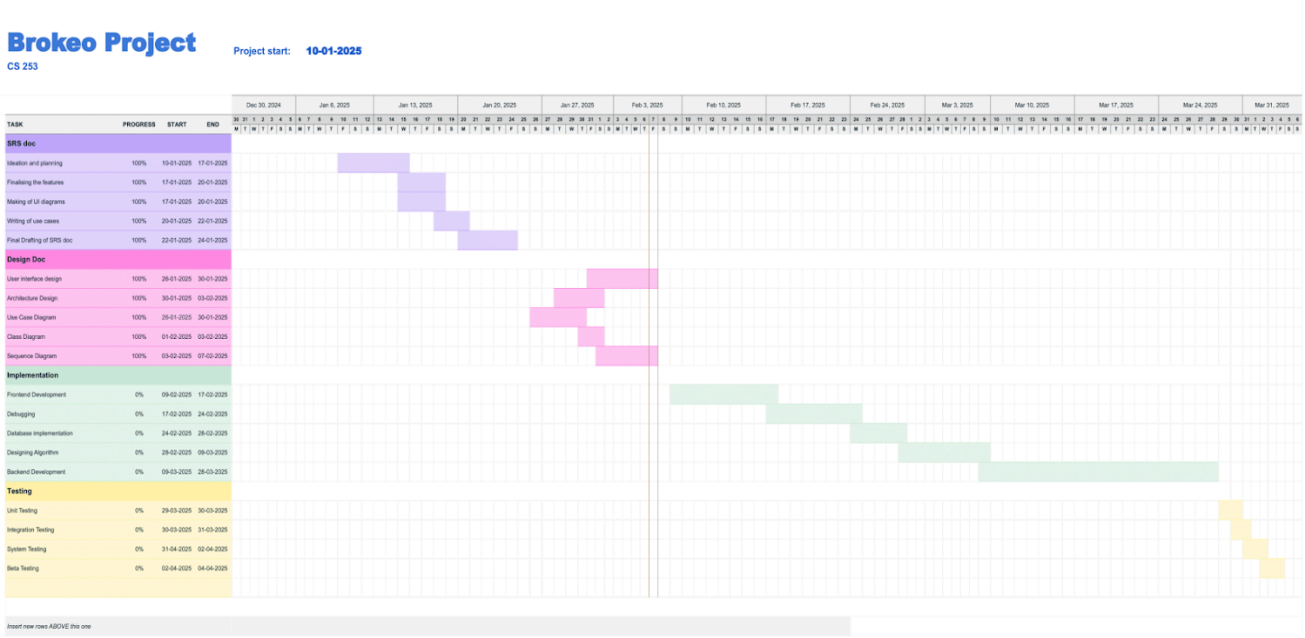
3.5.8 User Analytics



3.5.9 SMS Parsing :



4 Project Plan



MEMBERS	TASK
ANJALI PATRA	FULL STACK IMPLEMENTATION, CODE IMPROVEMENT, ALPHA TESTING
MARISHA THORAT	FULL STACK IMPLEMENTATION, CODE IMPROVEMENT, UNIT TESTING
AUJASVIT DATTA	BACKEND DEVELOPMENT, CODE IMPROVEMENT, ALPHA TESTING
DARSHAN SETHIA	FULL STACK IMPLEMENTATION, CODE IMPROVEMENT, UNIT TESTING
DHRITI BARNWAL	FULL STACK IMPLEMENTATION, SYSTEM TESTING, BETA TESING
RUDRANSH VERMA	BACKEND DEVELOPMENT, INTEGRATION TESTING, ALPHA TESTING
SANJNA S	FRONTEND DEVELOPMENT, UNIT TESTING, ADDRESSING FEEDBACK
SOLANKI SHREY JIGNESHBHAI	FRONTEND DEVELOPMENT, SYSTEM TESTING, ADDRESSING FEEDBACK
SURYANSH VERMA	BACKEND DEVELOPMENT, INTEGRATION TESTING, CODE IMPROVEMENT
BHAVNOOR SINGH	FRONTEND DEVELOPMENT, MANUAL FOR BETA TESTING, BETA TESTING

Appendix A - Group Log

Date	Timings	Duration	Minutes
30th Jan	21:00-23:30	2hrs 30mins	● Discussion regarding the overall framework of the document, classes to work on and work distribution corresponding to it
1st Feb	10:00-12:00	2hrs	● Planning architecture design ● Work done on classes and class diagrams
4 Feb	9:00-14:00	5hrs	● Finalisation of state diagrams, sequence diagrams ● Worked on the UI design with work distribution
6th Feb	20:00-23:00	3hrs	● Refinement of all class, state and sequence diagrams ● Completion of UI
7th Feb	14:00-17:00	3hrs	● Proofread the document