UE21CS352B - Object Oriented Analysis & Design using Java

Mini Project Report

"Expense Tracker And Management"

Submitted by:

NIDA SAMREEN	**PES1UG21CS379**
NIDHI P G	**PES1UG21CS380**
NIKITA SURESH	**PES1UG21CS386**
PRAMATH S	**PES1UG21CS425**

6th Semester G Section

Prof. RAGHU B A RAO
Associate Professor

January - May 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

Index

Sl.No	Title	Page No
1	Problem Statement	2
2	Features	2-3
3	UML Diagrams (Use Case, Class Diagram, Activity Diagrams and State Diagrams of Individual Components)	4-14
4	Architecture Pattern	15-16
5	Design Principles	17-18
6	Design Patterns	19-20
7	Individual Contribution	21-22
8	Output Screenshots	23-34
9	GitHub Link	35

I. Problem Statement:

The Java Spring Boot Expense Tracker empowers users to efficiently manage personal and group expenses. It features robust user authentication, role-based access control, and notification alerts for enhanced security and user engagement. Group management functionalities enable users to create, manage, and analyze shared expenses collaboratively. The application boasts an intuitive interface, responsive design, and interactive visualization tools for seamless navigation and insightful expense analysis. Leveraging modern technologies and frameworks, it sets a new standard in expense management, prioritizing convenience, security, and usability.

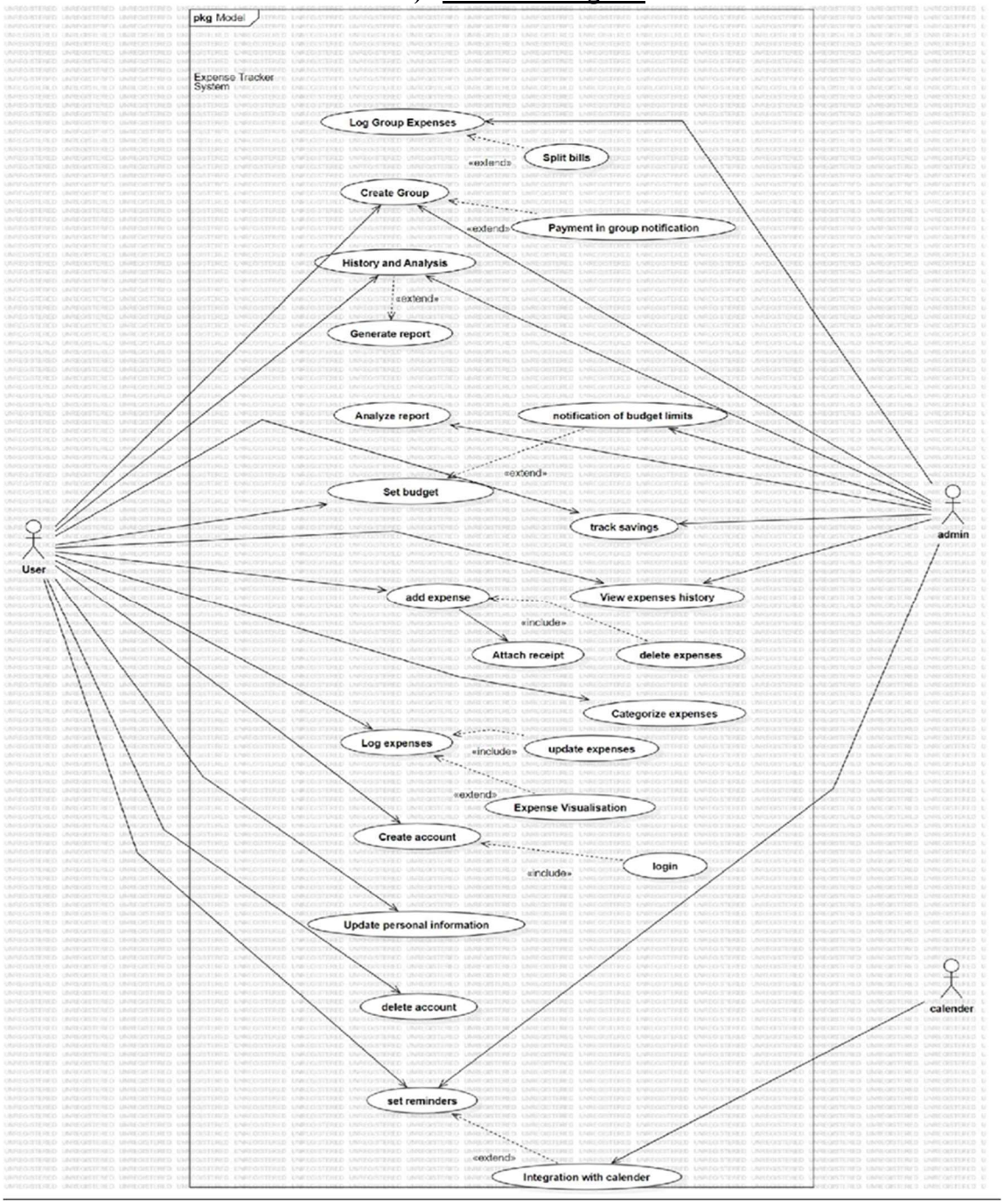
II. Features:

1. User Authentication and Authorization:
    - Allow users to register and log in securely to the application.
2. Personal Expense Management:
    - Enable users to record personal expenses, including details such as date, amount, category, and description.
    - Provide options for categorizing expenses (e.g., food, transportation, utilities, etc.).
    - Allow users to view, edit, and delete their expenses.
    - Implement charts or graphs to visualize personal spending patterns over time.
3. Group Expense Management:
    - Allow users to create groups for managing shared expenses among group members.
    - Provide features for adding and removing group members.
    - Implement functionalities for viewing group expenses.
4. Notification System:
    - Implement a notification system to alert users about upcoming payments, expense approvals, or other relevant activities.
    - Provide options for users to customize notification preferences based on their needs.
5. Financial Reports and Analysis:
    - Generate periodic financial reports summarizing expenses.
    - Implement analytical tools to identify spending trends, categories with the highest expenditure, etc.
    - Allow users to set budget limits and receive alerts when approaching or exceeding predefined thresholds.
6. User Interface and Experience:
    - Design an intuitive and user-friendly interface for easy navigation and interaction.

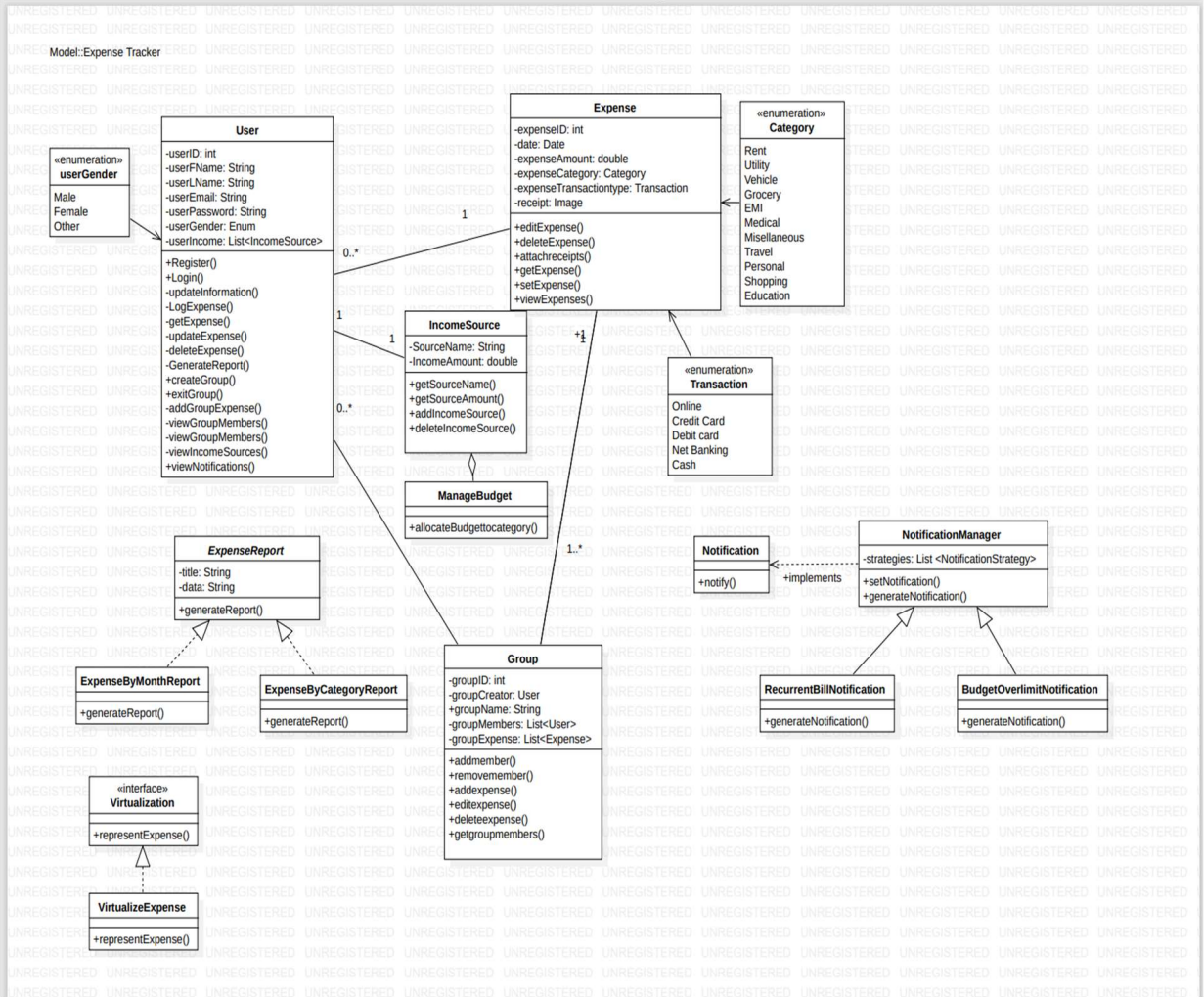
- Ensure responsiveness across various devices (desktop, tablet, mobile) for a seamless user experience.
- Provide search and filtering functionalities to quickly locate specific expenses or transactions.

III. UML Diagrams

1) Use Case Diagram

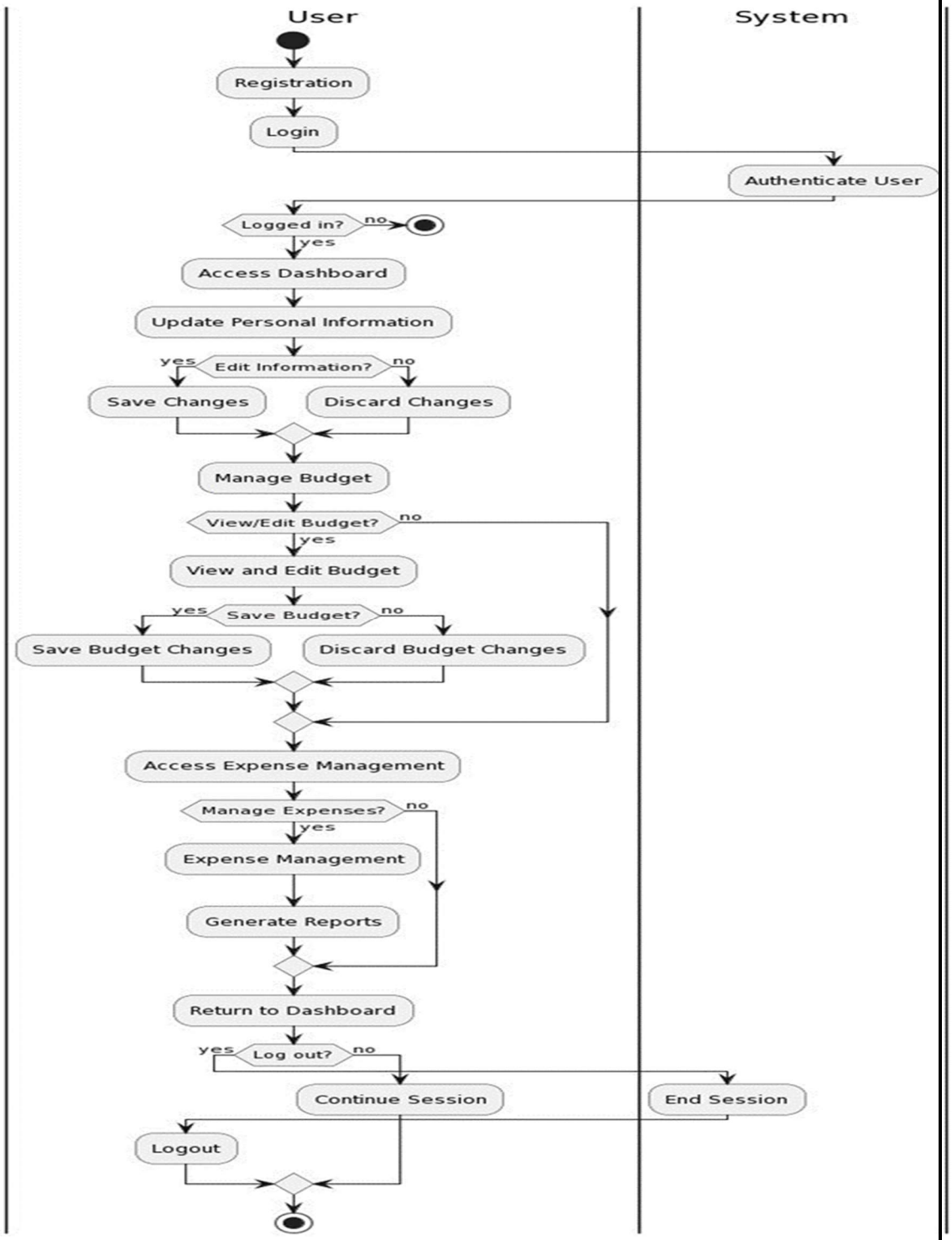


2) Class Diagram

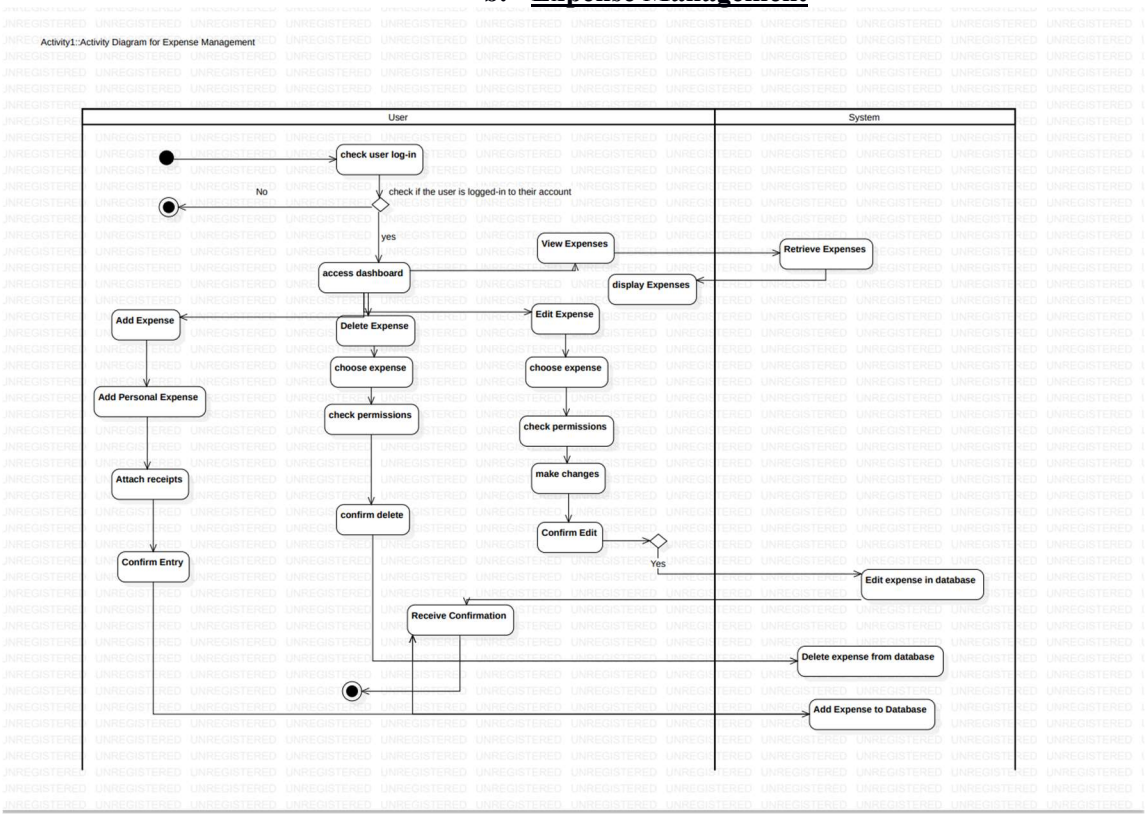


3) Activity Diagrams:

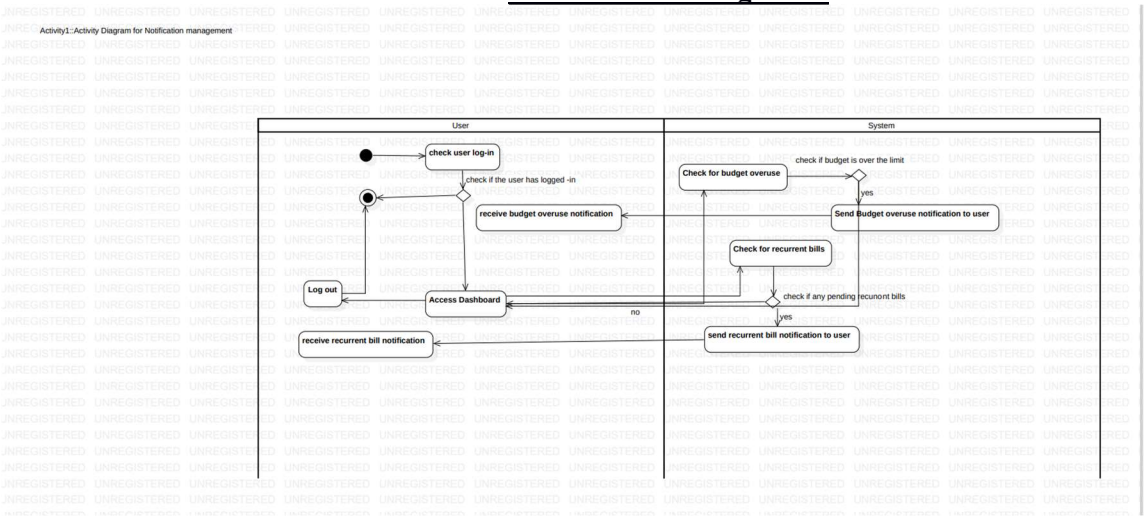
a. User Management



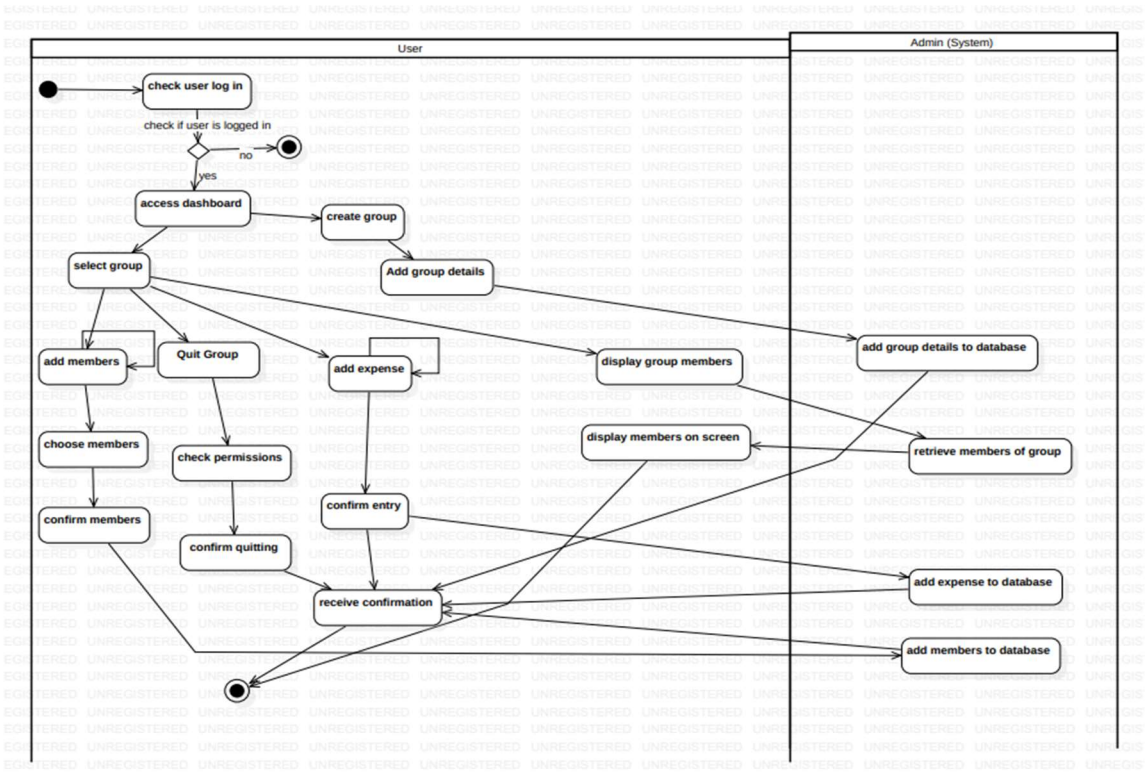
b. Expense Management



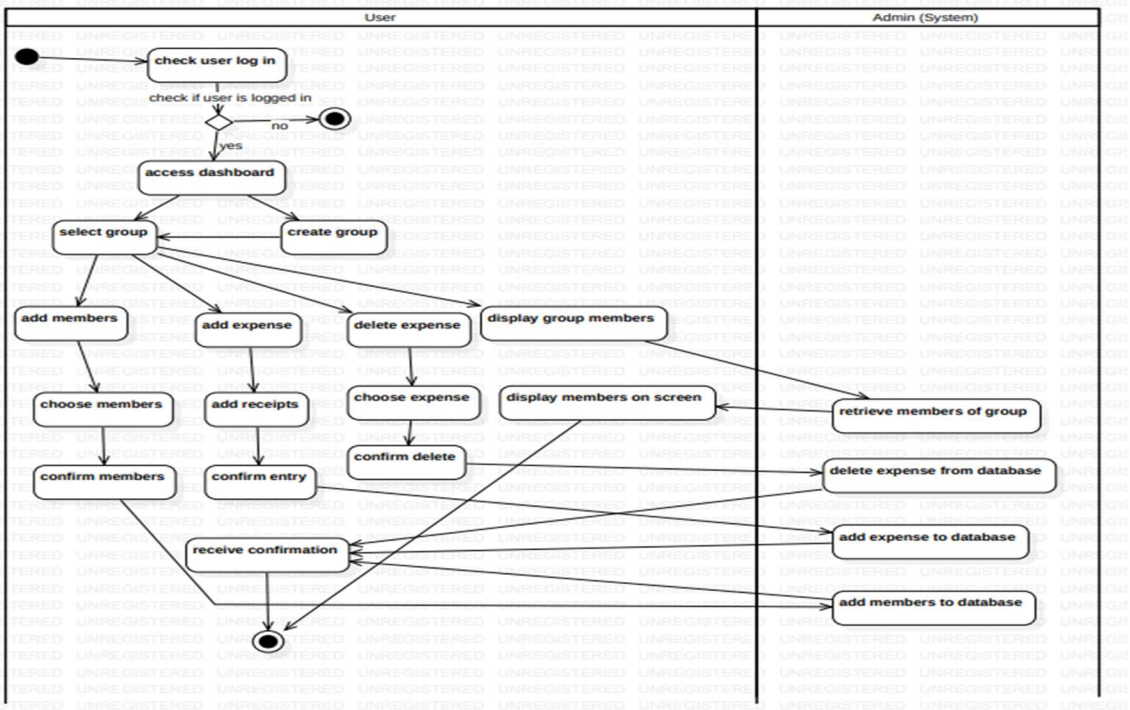
c. Notification Management



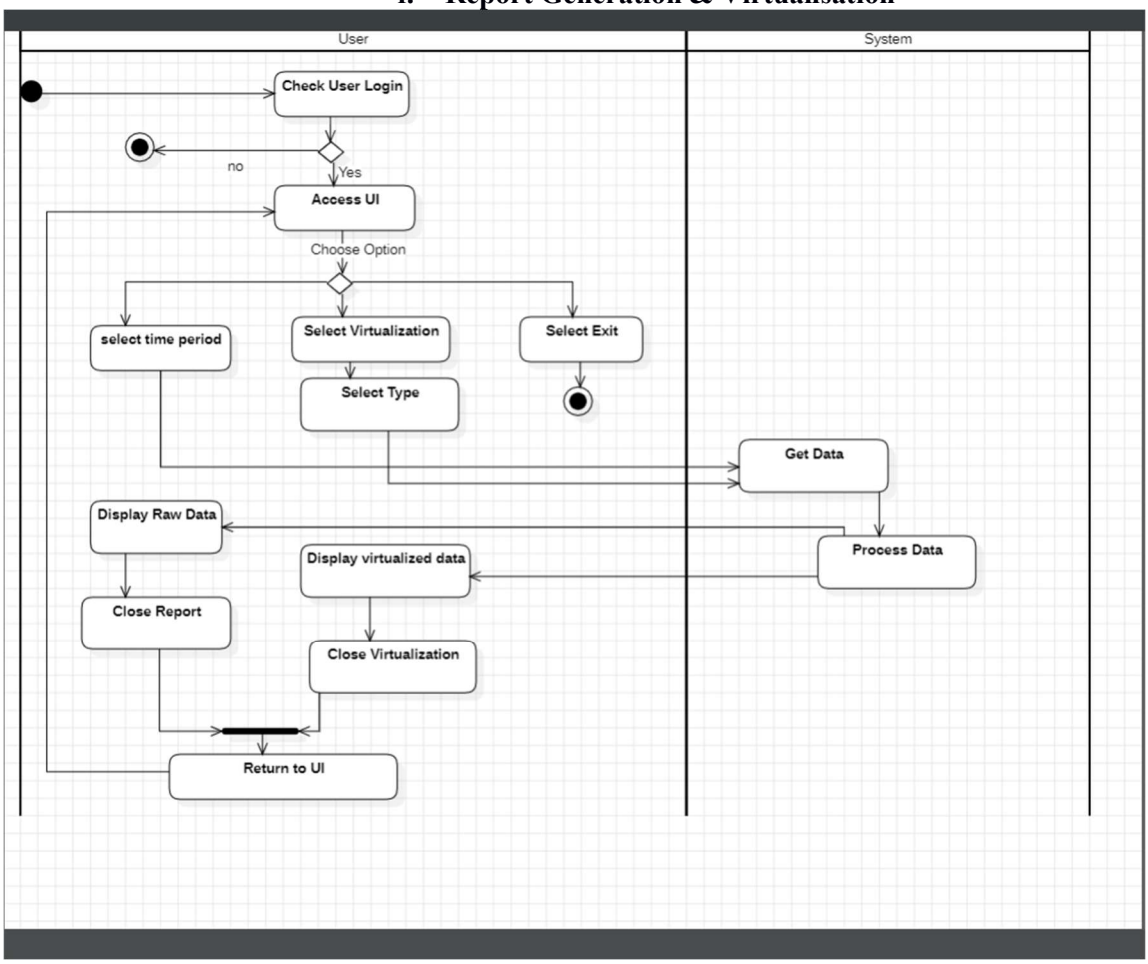
d. Group Management



e. Search Feature

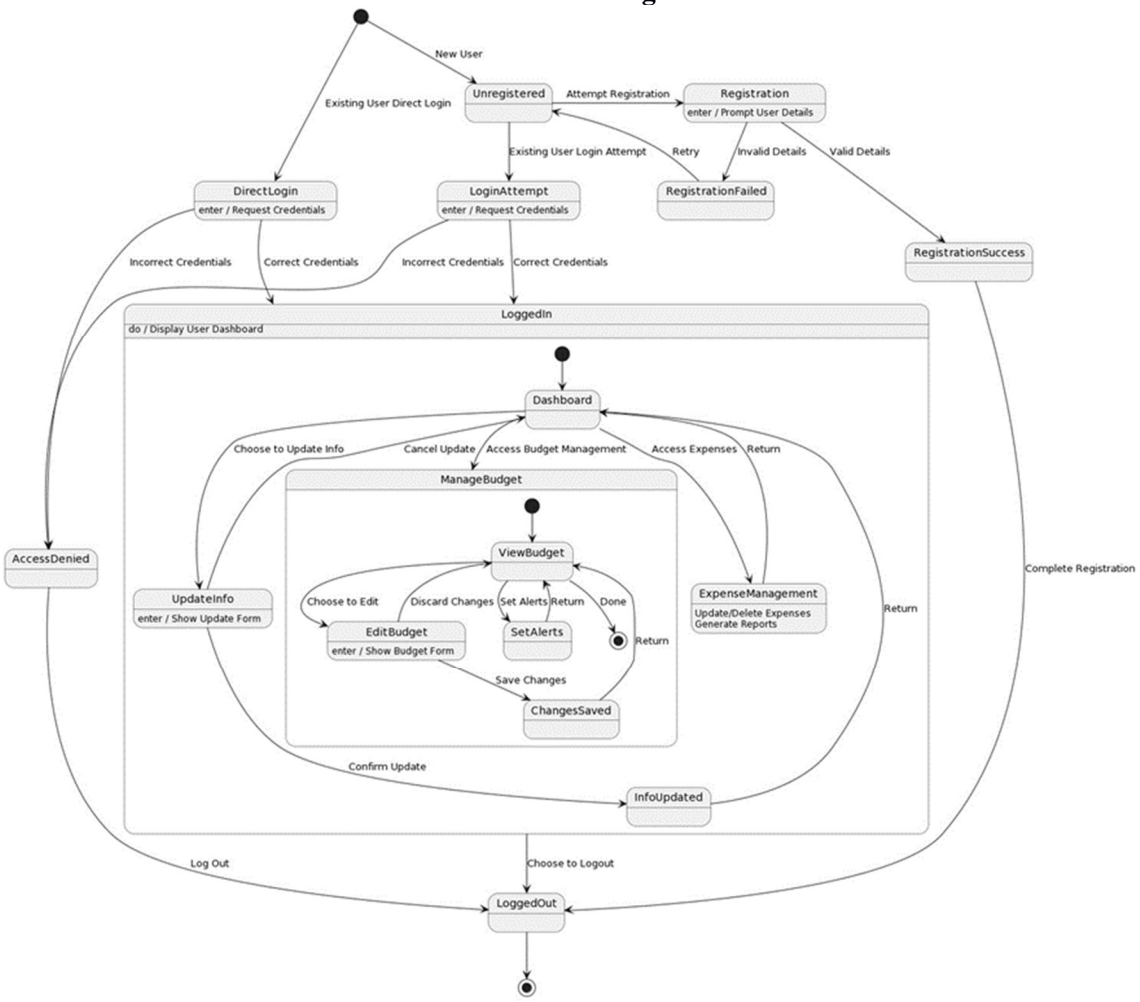


f. Report Generation & Virtualisation

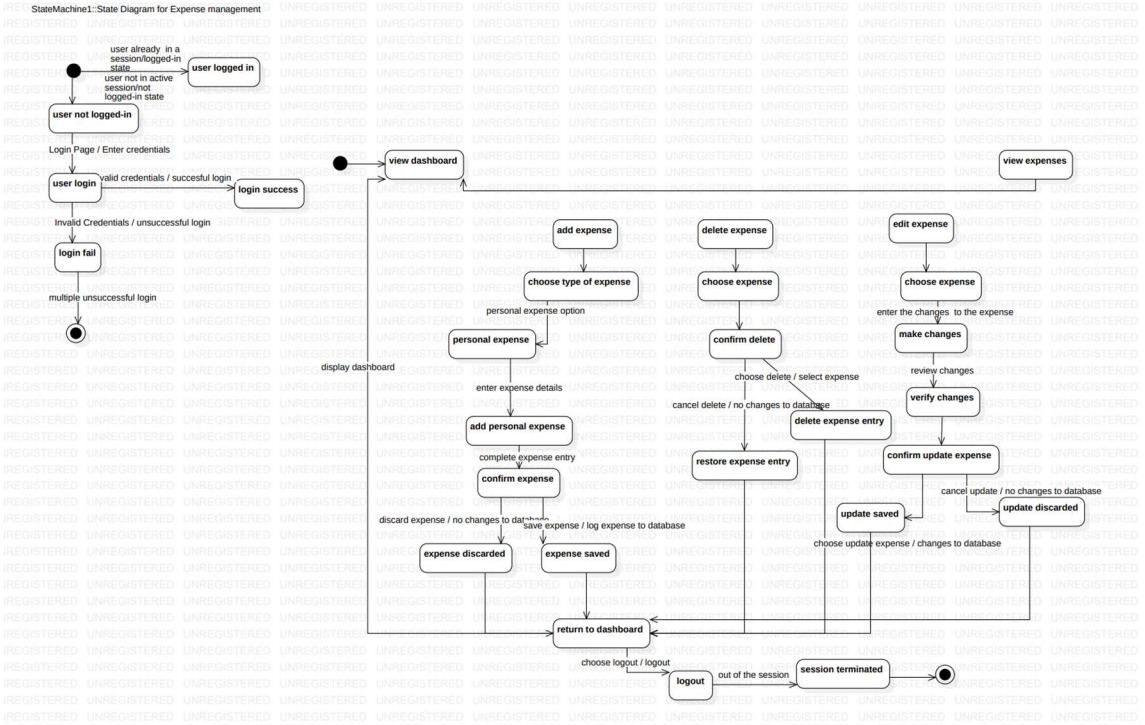


4. State Diagrams

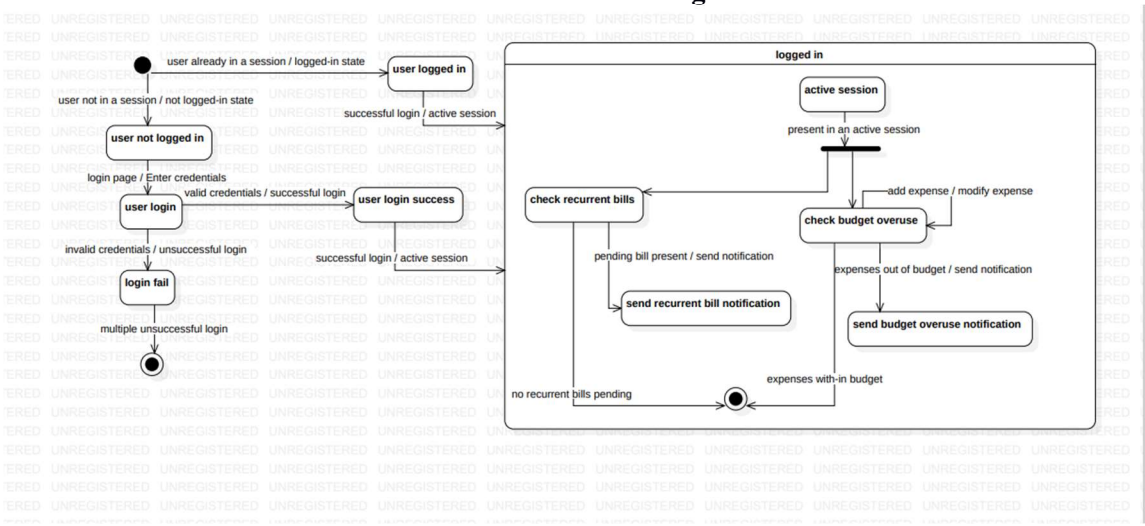
a. User Management



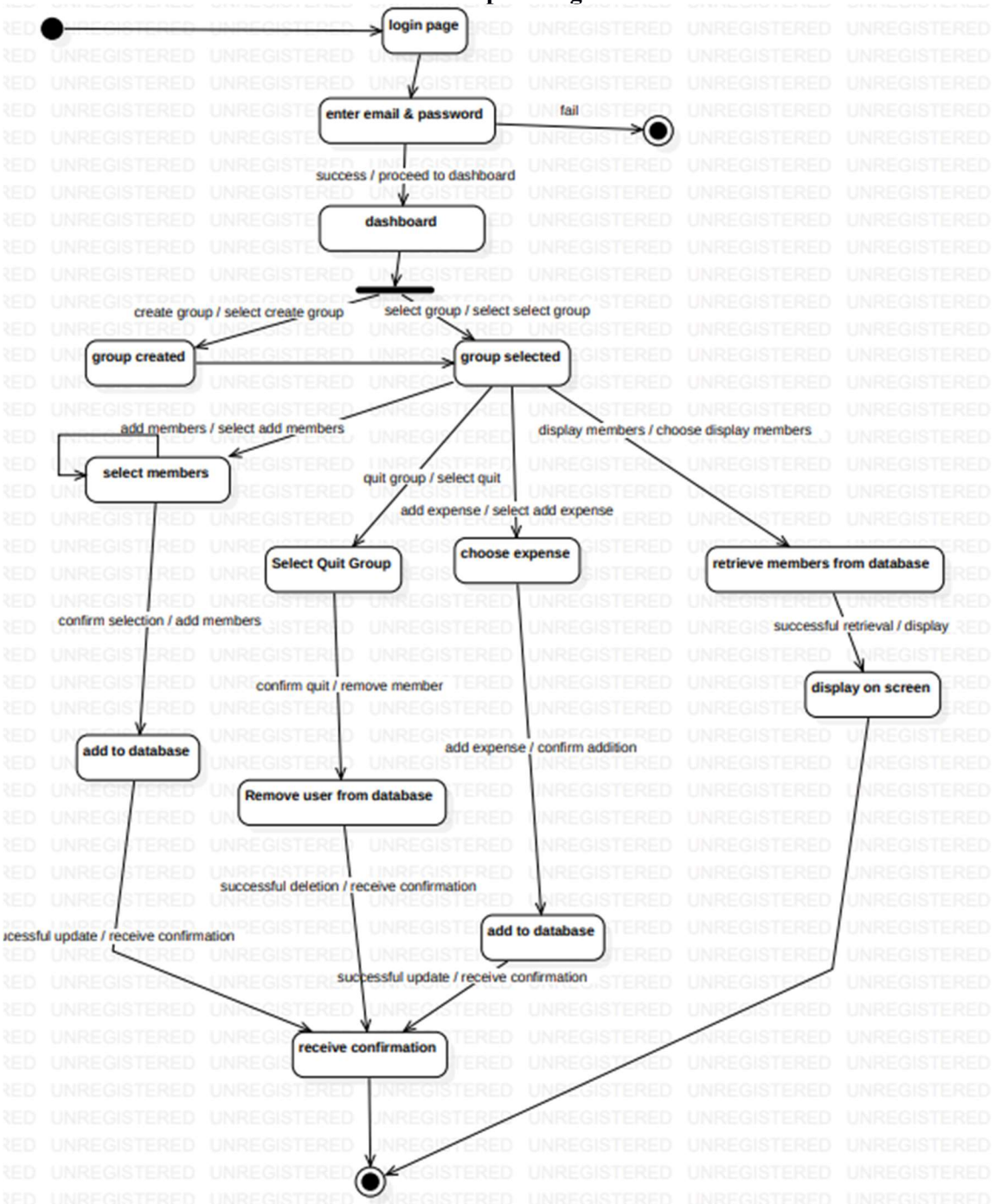
b. Expense Management



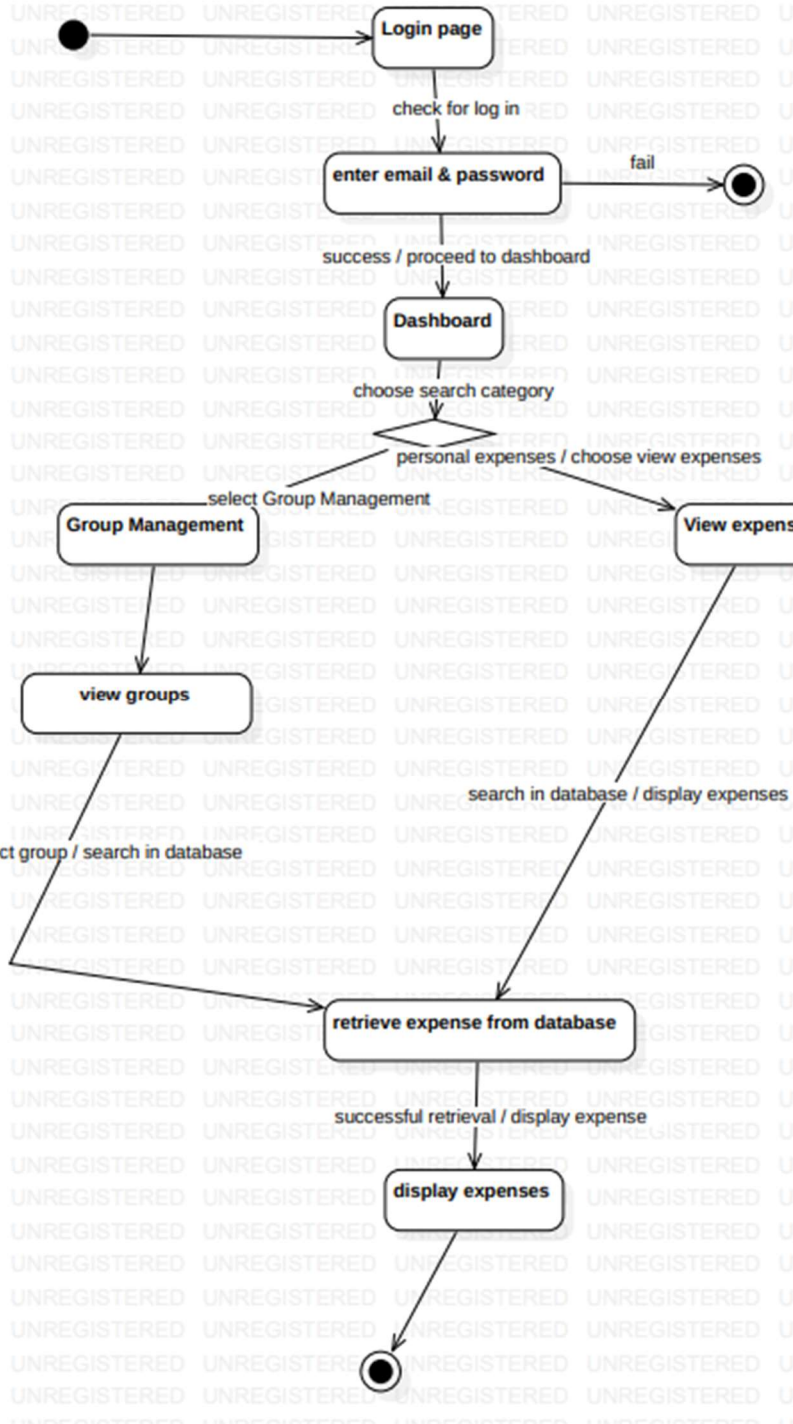
c. Notification Management



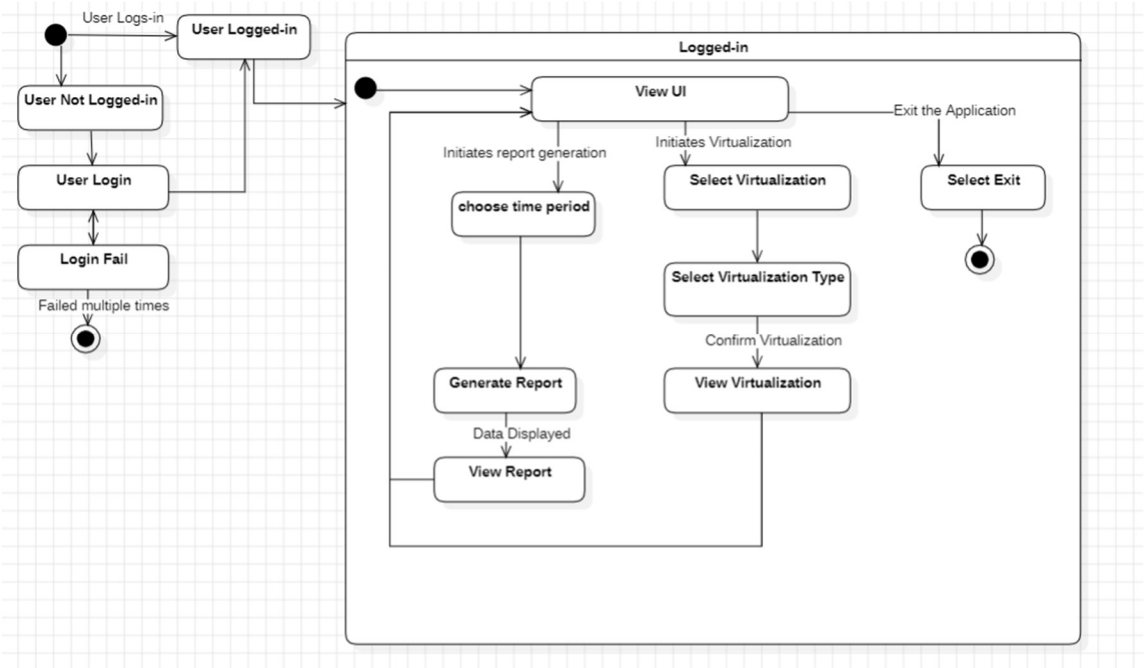
d. Group Management



e. Search Feature



f. Report Generation & Virtualisation



IV. **Architecture Pattern**

MVC Architecture Implementation Report

Our Java Spring Boot application employs the MVC architecture pattern to organize code into three primary concerns: models, views, and controllers. This pattern enhances maintainability, scalability, and separation of concerns. Below is an overview of each component within our application structure:

Model Components:
- Budget: Represents the budget data and its related operations.
- Expense: Encapsulates details and operations for individual expenses.
- ExpenseCategory: Defines categories for expenses.
- ExpenseGroup: Manages group-related data for expenses.
- ExpenseTransactionType: Enumerates the types of expense transactions.
- Gender: Enumerates gender types for user profiles.
- GroupExpense: Relates to expenses shared among a group of users.
- GroupMember: Represents members within an expense group.
- IncomeSource: Describes the sources of income for users.
- User: Represents user details and their related operations.

Controller Components:
- AuthController: Manages authentication-related processes, such as login and registration.
- BudgetController: Handles HTTP requests regarding budget management.
- ExpenseController: Processes requests related to expense tracking.
- GroupMemberController: Oversees actions specific to group members within the application.
- IncomeSourceController: Manages income sources for users, including CRUD operations.
- UserController: Deals with user-related requests, such as fetching and updating user profiles.

Notification Subsystem (Part of Controller or Service Layer):
- BudgetOverlimitNotification: A specific type of notification related to budget limits.
- NotificationManager: Orchestrates the sending of notifications.
- NotificationStrategy: Defines the strategy for notification delivery.
- RecurrentBillNotification: Handles notifications for recurring bills.

Repository Components:
- BudgetRepository: Interfaces with the database for budget-related data operations.
- ExpenseGroupRepository: Manages data access for expense groups.
- ExpenseRepository: Abstracts the data layer for expense management.
- ExpenseRepositoryImplementation: Provides a concrete implementation of the ExpenseRepository.
- GroupExpenseRepository: Facilitates data operations for group expenses.
- GroupMemberRepository: Handles data persistence for group member information.

View Components:
- HTML Templates: Our application uses Thymeleaf templates to render the server-side UI. Key templates include:
  - `addexpense.html`: UI for adding expenses.
  - `groupmanagement.html`: Interface for group-related activities.

- `incomeSources.html`: Display and manage user income sources.
- `profile.html`: User profile page.
- `error.html`, `login.html`, `register.html`: Standard authentication and error pages.
- *Additional HTML files* corresponding to various functionalities within the application.

Data Layer:
- SQL Database: The application relies on an SQL database for persistent storage, utilizing JPA repositories for data access and manipulation.

DTOs:
- Data Transfer Objects: Classes such as `UserLoginDto`, `UserRegistrationDto`, and `IncomeSourceDTO` are used to transfer data between the client and server, optimizing the data payload and enhancing security.

The application leverages the MVC architecture to separate the user interface, business logic, and data access layers, ensuring a clear division of responsibilities. The models define the core data structures, controllers manage request handling, and views are rendered using server-side templates. Additionally, a notification system is incorporated for alerts, and data persistence is achieved through an SQL database, with data transfer objects facilitating secure data exchange.

V. Design Principles:

1. Group Expense Management

Design Principle: Single Responsibility Principle

Since the module deals with managing group-related functionalities such as creating groups, adding members, managing expenses within groups, and retrieving group data, adhering to SRP ensures that each class or component within the module has a single responsibility. This helps keep the codebase organized, maintainable, and easier to understand, as each part of the system is focused on a specific task. Additionally, it facilitates easier testing and future enhancements to the module.

2. **Expense Management**

Design Principle: Single Responsibility Principle (SRP)

In our Expense Management system, the Expense class and ExpenseRepository exemplify the SRP by adhering strictly to managing expense-related operations:
Expense Class: This class encapsulates the attributes and behaviors related to an expense. It focuses solely on representing an individual expense entity, ensuring that any changes to expense-related attributes or behaviors will affect only this class.
ExpenseRepository: This class is responsible for the persistence and retrieval of expense data from a data source, like a database or a file system. By dedicating itself to data storage and retrieval tasks exclusively, it adheres to the SRP, ensuring that any modifications related to data persistence will not affect the business logic or representation of an expense.

3. **Design Principle: High Cohesion in Notification Management**

The NotificationManager and NotificationStrategy classes in the Expense Management system demonstrate high cohesion by focusing on specific and related responsibilities. The NotificationManager oversees the management and orchestration of notifications, while NotificationStrategy defines individual notification methods. This design promotes maintainability, readability, and flexibility, allowing for easy updates or additions to notification strategies without impacting other system components.

4. Report generation and Virtualisation:

Design Principle: Single Responsibility Principle (SRP)

Report generation:
Separation of Concerns: Functions have distinct roles. fetchExpensesData fetches data, groupExpensesByMonth groups it, and updateTableForMonth handles table updates. This division makes code easier to maintain and test.
Modularity: The code is designed for extensibility. New features can be added without changing existing functionality, thanks to the clear responsibilities of each function.

Virtualisation:
Data and Visualization Separation: Functions for fetching data and rendering visualizations are separate. This allows changes in one area without affecting the other.
Consistent Structure: The clear organization makes it easier to add new visualizations or adjust existing ones without disrupting the code's stability.

5.User Management

Design Principle: Single Responsibility Principle (SRP)

In our User Management module, SRP is exemplified through the distinct roles assigned to each component: UserController: Manages HTTP request handling exclusively, delegating all business logic to the service layer. This controller is solely responsible for routing and responding to user-related HTTP requests. UserService: Handles all business operations related to users, such as validation, registration, and data manipulation, ensuring that the UserController remains streamlined and focused only on communication tasks. This separation enhances clarity and maintainability, as modifications in user management processes or business rules will primarily involve changes in the UserService without affecting the UserController.

Service Layer Design Design Principle: Dependency Inversion Principle (DIP)

The application's service layer demonstrates DIP by interacting through interfaces rather than concrete implementations, promoting flexibility and decoupling: UserService and UserRepository: The UserService depends on the UserRepository interface to abstract the details of data access implementations. This setup allows the UserService to remain unchanged even if the underlying data access techniques or technologies evolve. Implementing DIP facilitates easier unit testing and future upgrades or replacements of data access layers, contributing to a more robust and adaptable application architecture.

VI. Design Patterns:

Notification Management

Design Pattern: OBSERVER

The Observer pattern is employed to implement the notification feature within the application, chosen specifically for its ability to provide flexibility and extensibility in notification generation. This design approach fosters extensibility and reusability, allowing for the seamless integration of new notification types without altering existing code. Moreover, it promotes a clear separation of concerns by isolating the responsibility for generating notifications from the core application logic. This not only streamlines the maintenance process but also facilitates

easier future extensions. Additionally, the pattern promotes loose coupling, ensuring that the NotificationManager remains agnostic to the specifics of individual notification strategies. Instead, it leverages the NotificationStrategy interface to seamlessly trigger the appropriate notification generation based on the context, enhancing the overall robustness and adaptability of the notification system.

Strategy Pattern: The NotificationStrategy class defines a family of algorithms (or notification methods) and encapsulates each one, allowing them to be interchangeable. This design enables dynamic selection of the appropriate notification method at runtime, offering flexibility and maintainability without altering the NotificationManager or core notification code.

Group Expense Management:

Design Pattern: Observer

- Notifications: Observer pattern facilitates notifying users about events like adding members, quitting groups, or adding expenses, ensuring timely updates and user engagement.
- Decoupling: By decoupling the subject (e.g., group or expense) from its observers (e.g., UI components or services), Observer pattern enables flexible and maintainable interactions between different system components.
- Dynamic Updates: With Observer pattern, UI elements and other components receive automatic updates upon changes in group or expense states, ensuring real-time responsiveness without manual intervention.
- Scalability: Observer pattern allows easy scalability by accommodating new features or events that require notification mechanisms, ensuring the system can evolve and adapt to changing requirements without significant refactoring.

Report generation and Virtualization:

Design Pattern**: Observer**

The Observer Pattern is demonstrated by attaching event listeners to specific elements and responding to events like button clicks or document loading. For instance, the event listener document.addEventListener('DOMContentLoaded', function() observes the "DOMContentLoaded" event to initialize fetching of data and setting up further interactions.

Similarly, there's an event listener on the "Generate Report" button to trigger report generation when clicked, and an event listener on the dropdown for selecting chart types to trigger chart rendering.

VII. **Individual Contributions:**

Nidhi P G:

Throughout the development phase of the Expense Management project, my primary focus was on the meticulous handling of personal expenses for individual users. I undertook the responsibility of implementing the core functionalities related to personal expenses, which encompassed the operations of adding, viewing, editing, and deleting expenses.

Additionally, I took the lead in establishing the notification management system. This included designing and implementing strategies to deliver timely and relevant notifications to users, tailored to different triggers and conditions.

One of the significant contributions I made was the seamless integration of all project components into a unified application. This entailed ensuring that each module and feature not only functioned independently but also harmoniously interacted with one another. By achieving this integration, I

aimed to enhance the user experience, making the application more intuitive and convenient for users to manage their expenses effortlessly.

In conclusion, my contributions spanned both the individual expense management functionalities and the overarching integration of all project components. Through these efforts, I aimed to deliver a robust, user-friendly, and comprehensive Expense Management system that meets the diverse needs of our users.

Nikita Suresh:

In the project, my contribution primarily focused on developing the group management module within the Expense Tracker application. I spearheaded the design and implementation of features crucial for efficient collaboration, including group creation, member management, and expense-tracking functionalities. Furthermore, I ensured seamless integration of the group management module with the overall application architecture, facilitating smooth interoperability with other essential modules such as user authentication and expense analysis. Through meticulous attention to detail and rigorous testing, my efforts significantly enhanced the application's capability to streamline shared expense management and foster collaborative financial planning among users.

Pramath: User Management

User authentication is crucial for modern web applications, and achieving both security and flexibility can be challenging. This report highlights the Strategy Pattern as an effective solution for creating scalable and secure user authentication systems
The Strategy Pattern effectively decouples authentication logic from user management, allowing for easy interchange and upgrades of authentication mechanisms without affecting other application areas. It includes three main components:
- Strategy Interface: A common interface for all strategies.
- Concrete Strategies: Implementations of the interface, each with a unique authentication algorithm.
- Context: A class that can dynamically switch strategies.
We implemented two strategies: SimpleAuthenticationStrategy for direct password comparison and HashedAuthenticationStrategy for secure password checking using SHA-256 hashing. Using Spring's dependency injection, these strategies are easily integrated and can be swapped dynamically based on configuration or environmental needs.

Nida Samreen: Report generation and Virtualisation:

My primary focus was on obtaining expense data from the database and organizing it by category. I was responsible for implementing core functionalities for this process, including fetching expenses, calculating monthly totals for each category, and generating relevant reports.

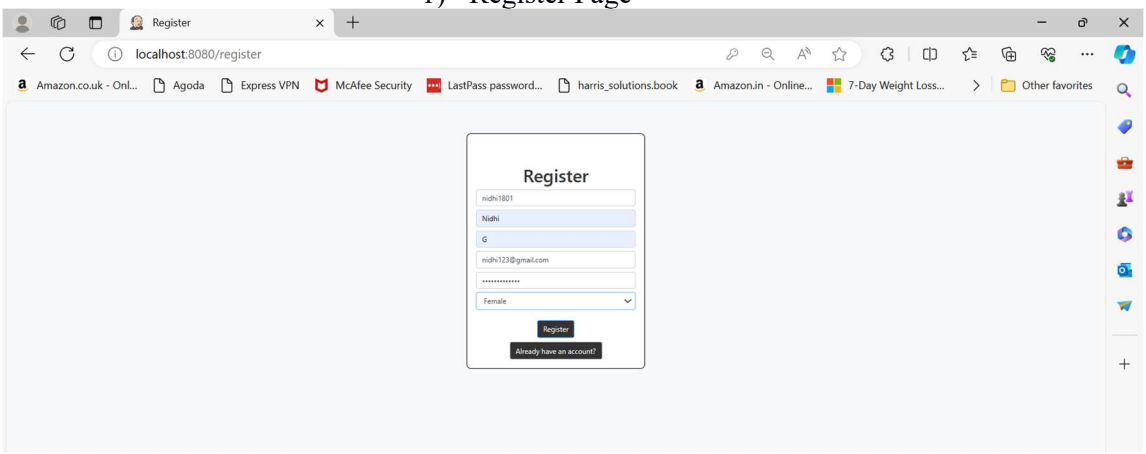
I was also given the responsibility of developing data visualizations for the project. This involved creating pie charts and bar graphs to represent category-wise expenses, providing a more straightforward way to view and analyze the data.

One of the major contributions I made was ensuring the smooth integration of data visualization with the reporting system. This required ensuring that the charts and graphs accurately reflected the calculated data and could easily interact with other components of the project.

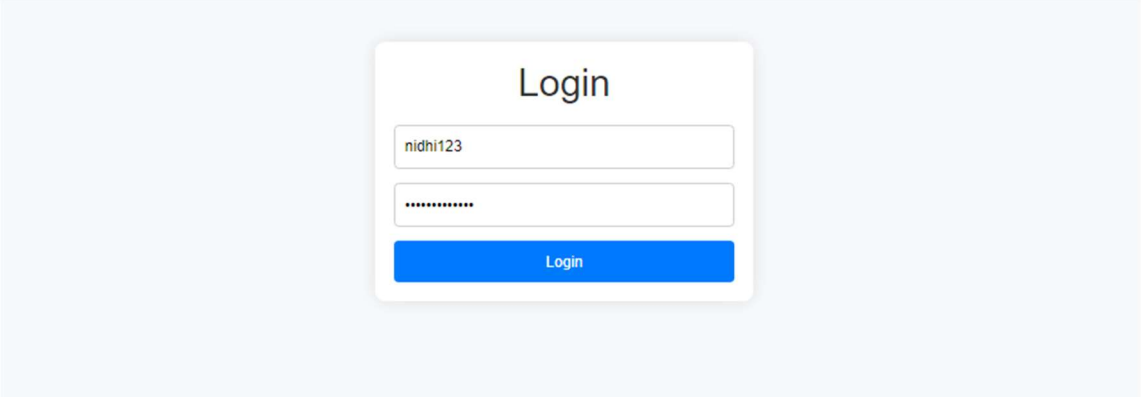
VIII. Output Screenshots:

A) Web Application

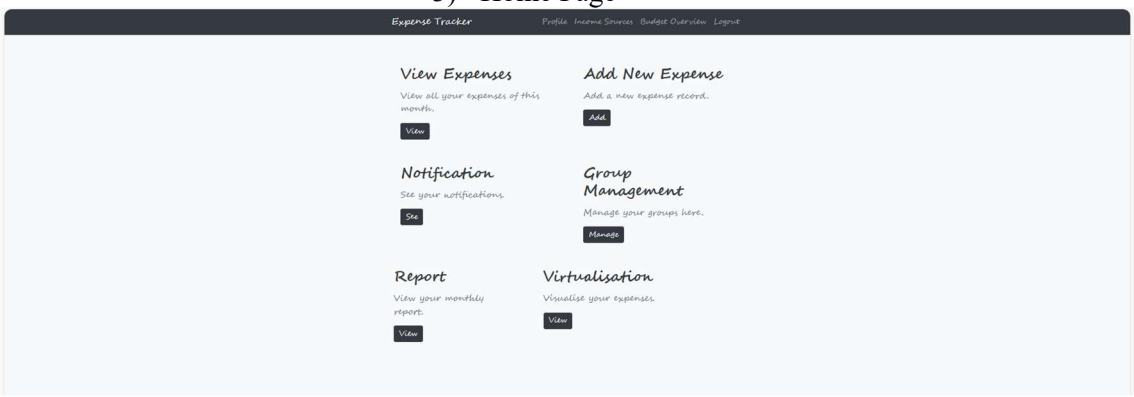
1) Register Page



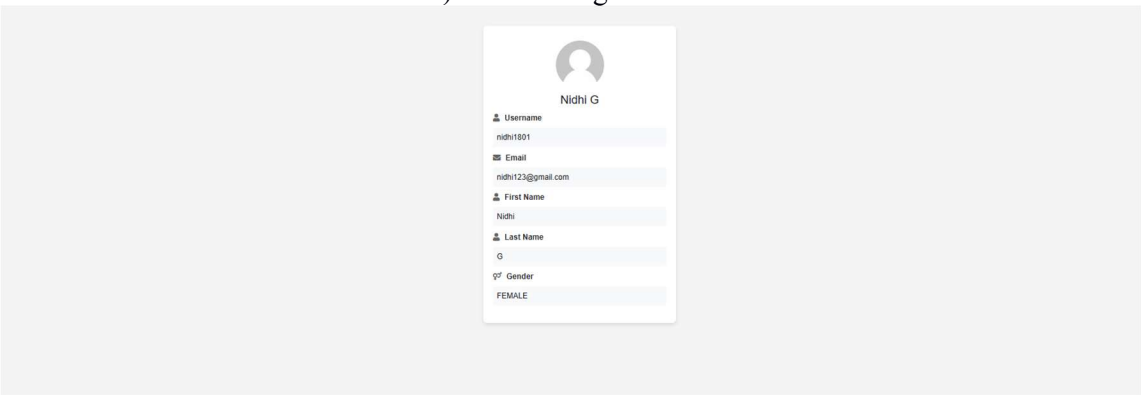
2) Login Page



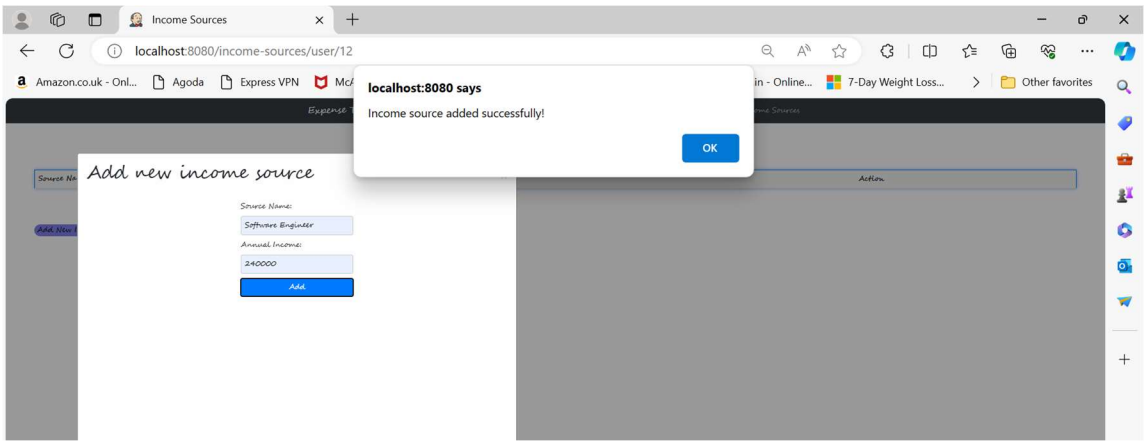
3) Home Page



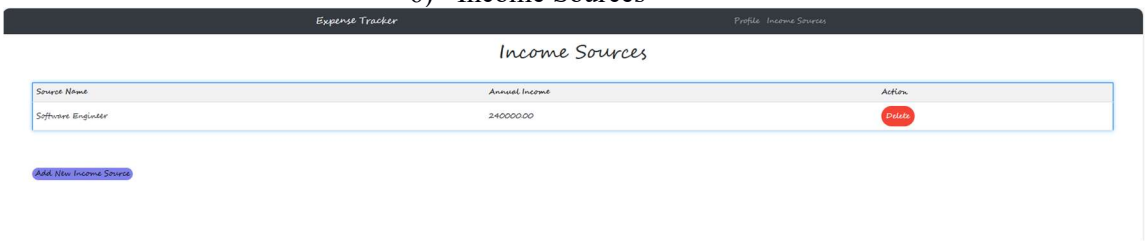
4) Profile Page



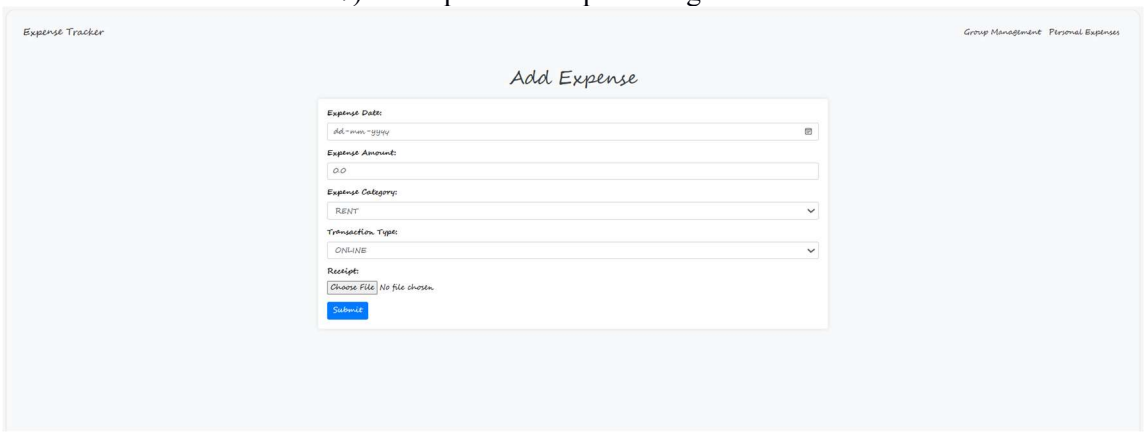
5) Income Sources Page



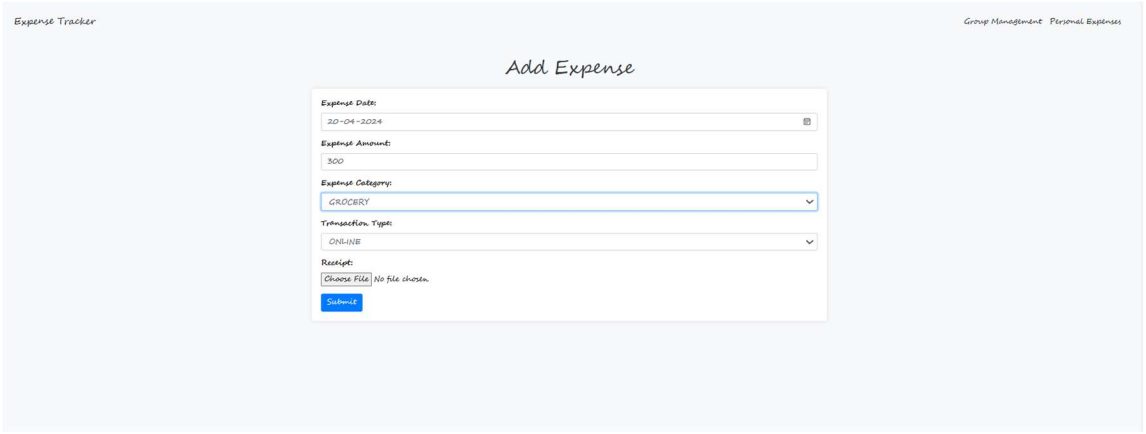
6) Income Sources



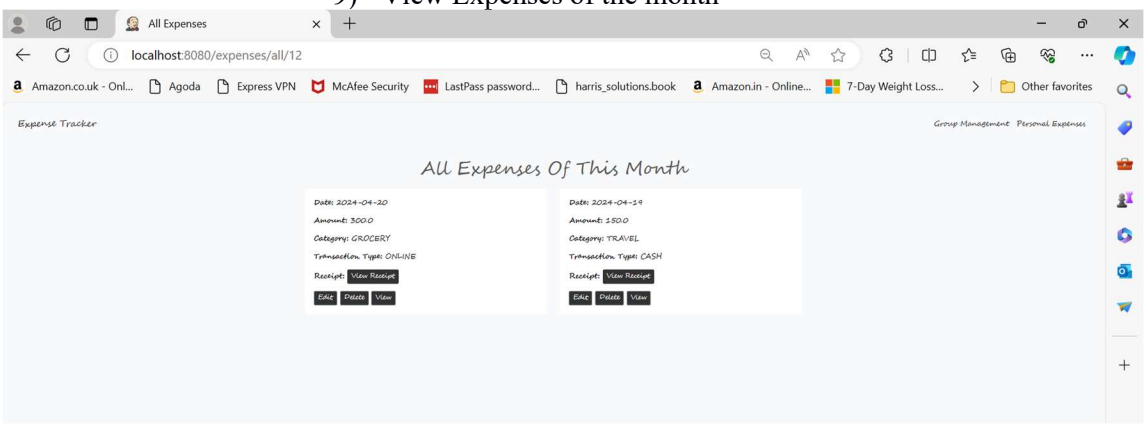
7) Add personal Expense Page



8) Adding an expense

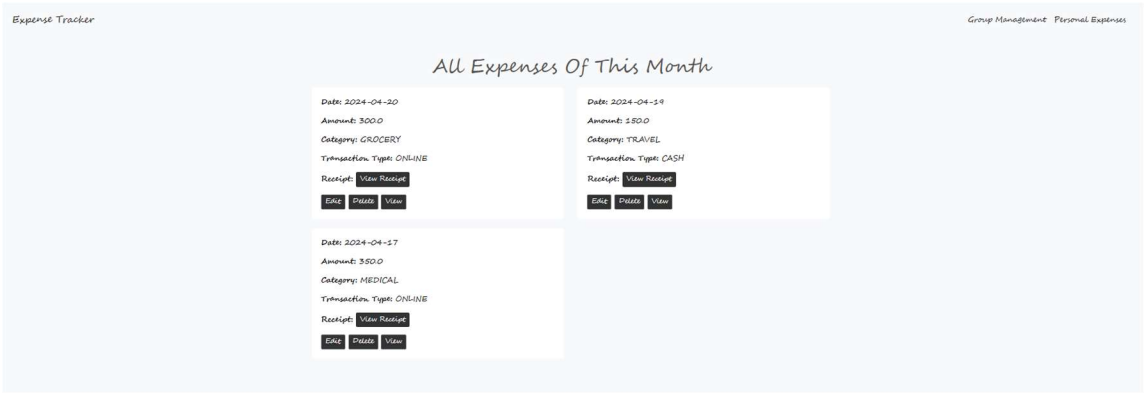


9) View Expenses of the month

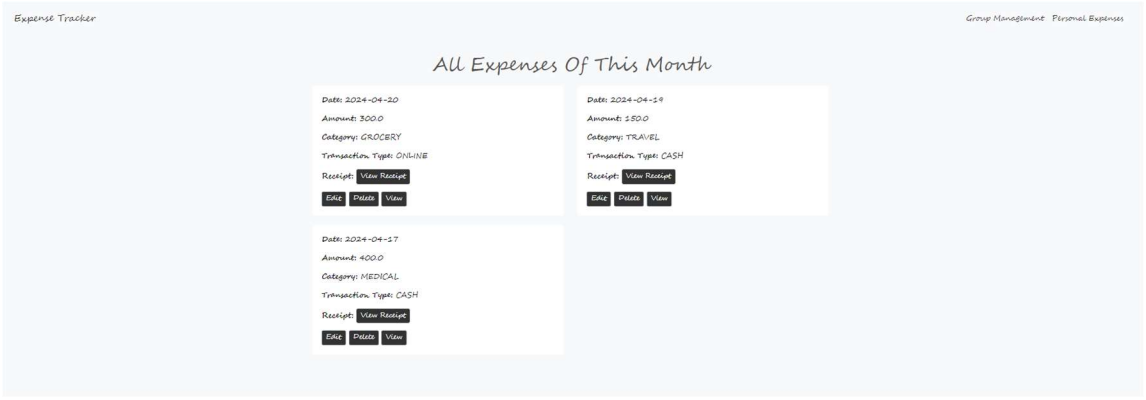


10) Edit an expense

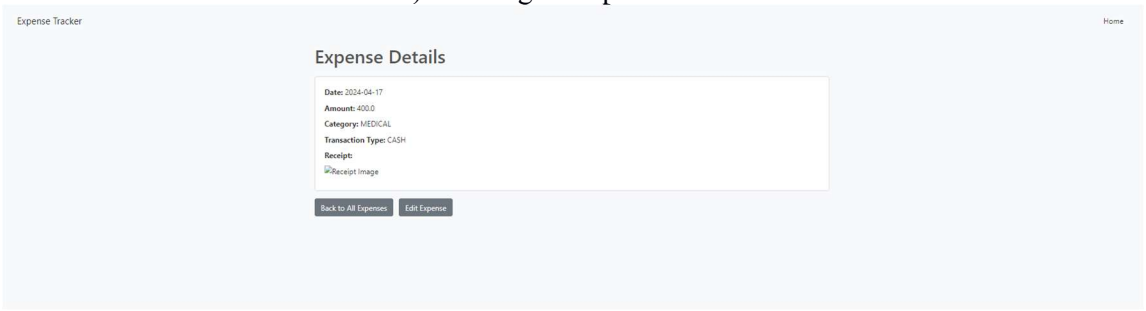
Before:



After:

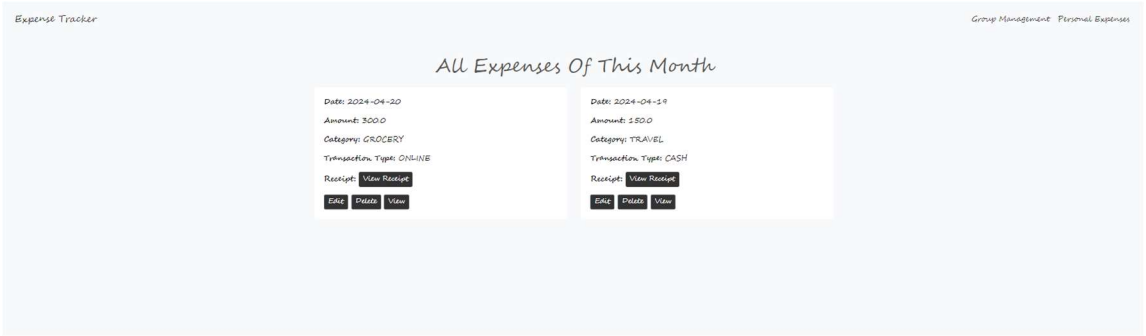


11) Viewing an expense



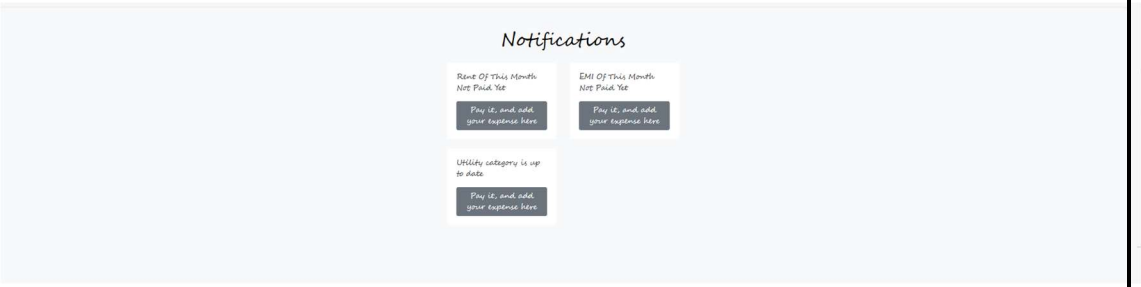
12) Delete an Expense

After Delete:



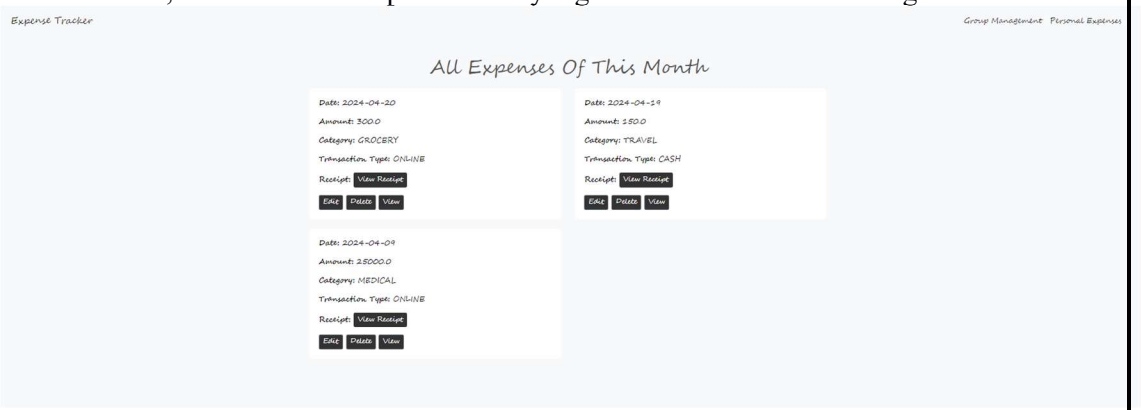
13) Notification Feature

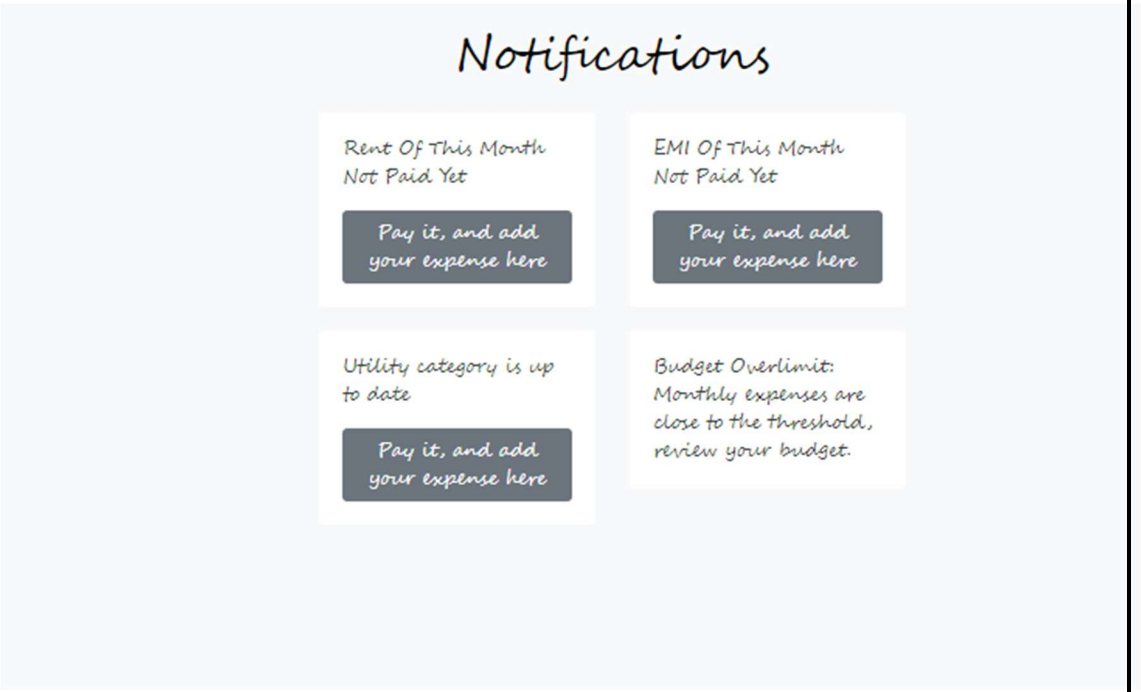
a) Recurrent Bills Notification :



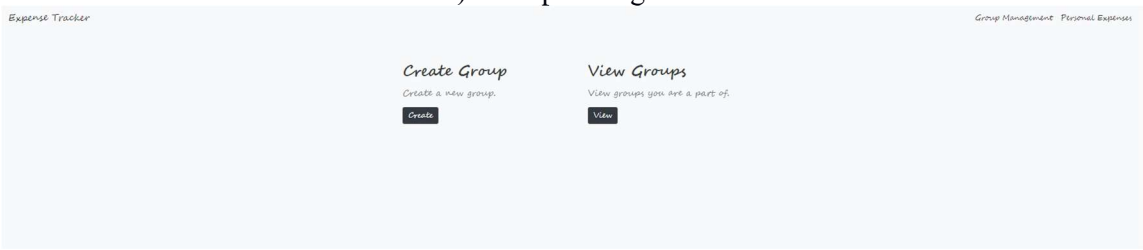
b) Budget Overlimit Notification:

For this, we will add an expense of very high value and see the working.



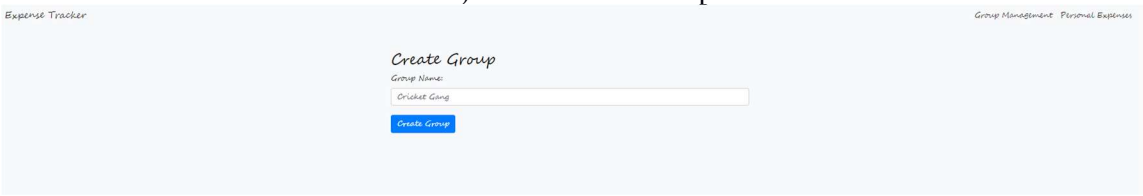


14) Group Management



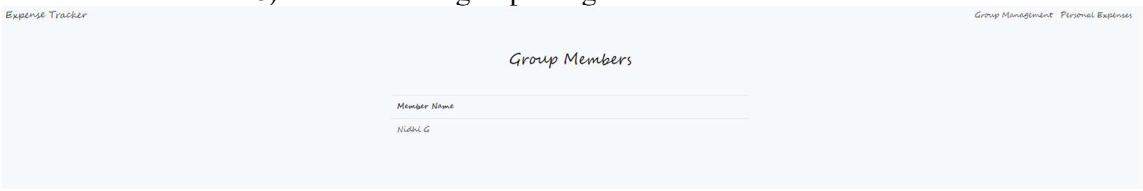
15) View Groups

16) Create a new Group

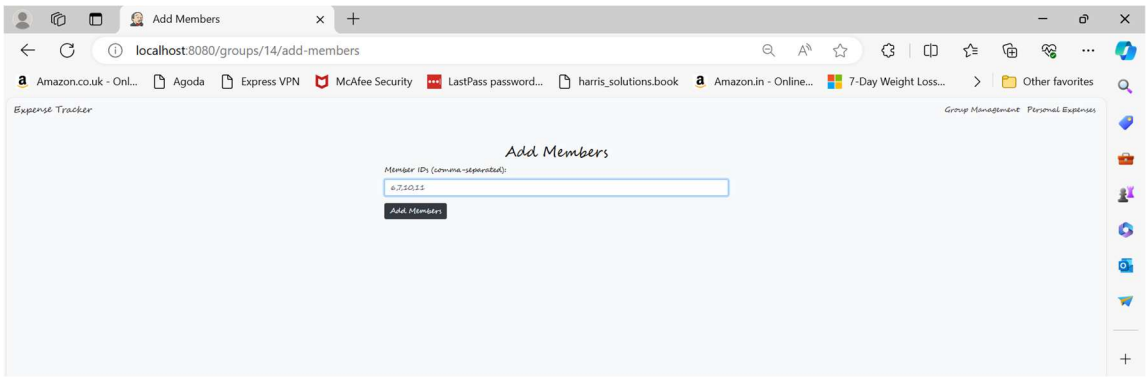


17) View groups after creating

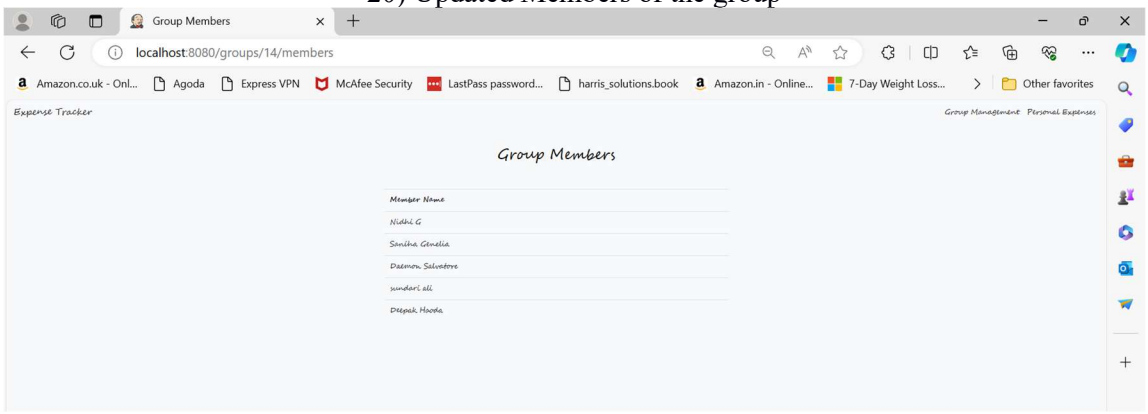
18) Creator of the group being added to the members list



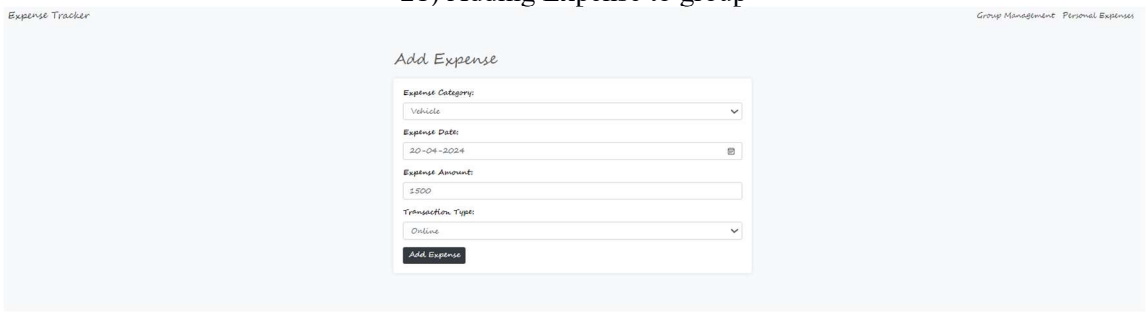
19) Adding Members to the group



20) Updated Members of the group



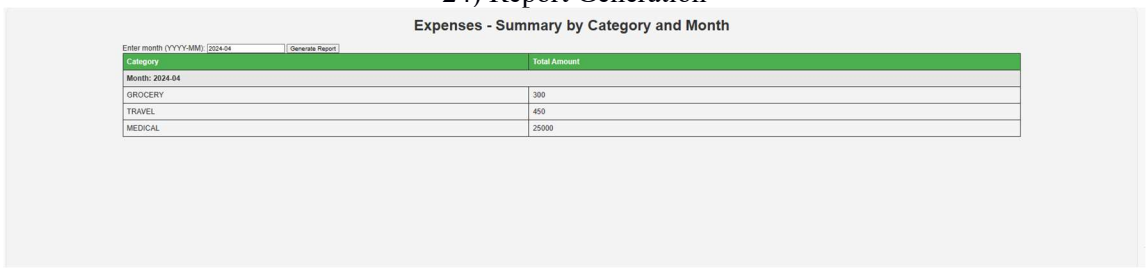
21) Adding Expense to group



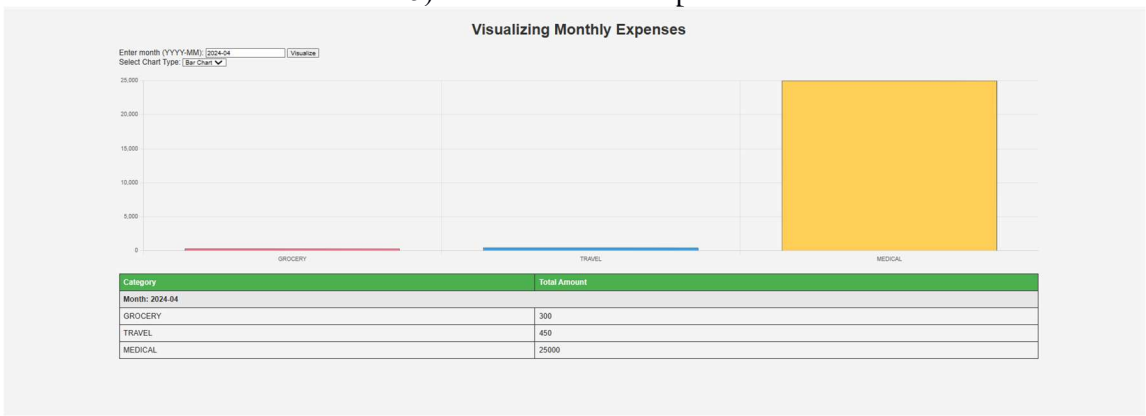
22) View expenses of the group

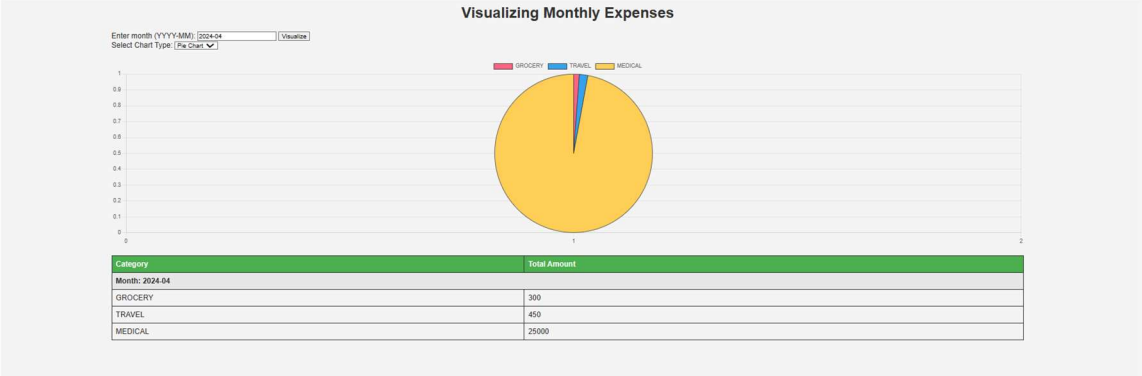
23) Quit group

24) Report Generation

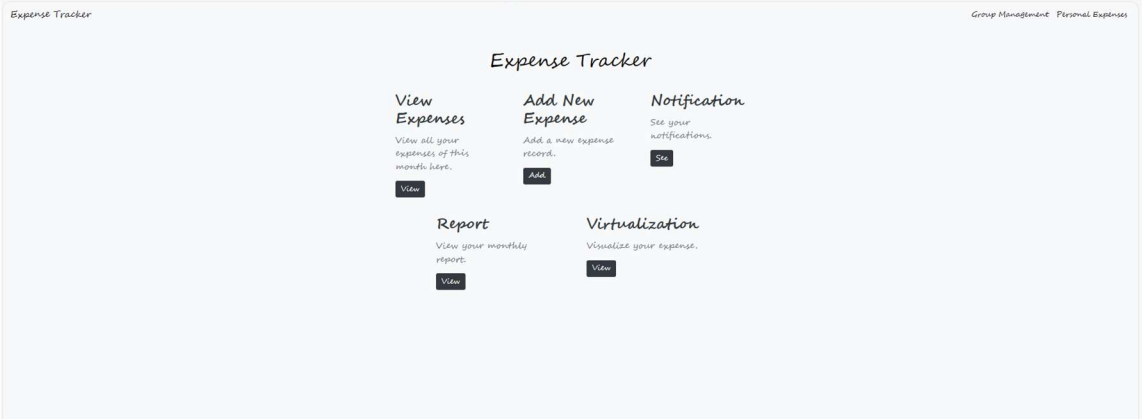


25) Virtualization of Expenses





26) Expenses Dashboard



B) Backend - Database

```
mysql> select * from users;
+---------+-----------------+------------+------------+---------------------------+-----------------+--------+
| user_id | user_name       | first_name | last_name  | email                     | password        | gender |
+---------+-----------------+------------+------------+---------------------------+-----------------+--------+
|       1 | john_doe        | John       | Doe        | john.doe@example.com      | hashed_password1 | MALE   |
|       2 | jane_doe        | Jane       | Doe        | jane.doe@example.com      | hashed_password2 | FEMALE |
|       3 | sam_smith       | Sam        | Smith      | sam.smith@example.com     | hashed_password3 | OTHER  |
|       4 | alex_jones      | Alex       | Jones      | alex.jones@example.com    | hashed_password4 | MALE   |
|       5 | lisa_white      | Lisa       | White      | lisa.white@example.com    | hashed_password5 | FEMALE |
|       6 | saniha          | Saniha     | Genelia    | sanihanoronha123@gmail.com | saniha123       | FEMALE |
|       7 | daemonsalvatore | Daemon     | Salvatore  | daemonsalvatore@gmail.com | daemon123       | MALE   |
|       9 | redbull         | Max        | Verstappen | max@gmail.com             | redbull123      | MALE   |
|      10 | deewani         | sundari    | ali        | deewanimastani@gmail.com  | deewani123      | FEMALE |
|      11 | deepakhooda     | Deepak     | Hooda      | deepakhooda@gmail.com     | deepak123       | MALE   |
|      12 | nidhi1801       | Nidhi      | G          | nidhi123@gmail.com        | nidhinidhi123   | FEMALE |
+---------+-----------------+------------+------------+---------------------------+-----------------+--------+
11 rows in set (0.00 sec)
```

mysql> select * from expenses;

id	user_id	expense_date	expense_amount	expense_category	expense_transaction_type	expense_receipt
3	1	2024-03-31 00:00:00.000000	500	PERSONAL	CASH	NULL
4	1	2024-03-10 00:00:00.000000	500	PERSONAL	CASH	0x433A55736572734E69646869446F776E6C6F61647308696C6C2E706E67
5	1	2024-04-03 00:00:00.000000	1500	UTILITY	CASH	0x433A55736572734E69646869446F776E6C6F61647308696C6C2E706E67
6	3	2024-04-06 00:00:00.000000	1000	MEDICAL	NET_BANKING	0x
7	2	2024-04-07 00:00:00.000000	500	PERSONAL	ONLINE	0x
8	1	2024-04-12 00:00:00.000000	35000	MISCELLANEOUS	CREDIT_CARD	0x
9	6	2024-04-12 00:00:00.000000	300	GROCERY	CASH	0x
10	6	2024-04-13 00:00:00.000000	2000	VEHICLE	CREDIT_CARD	0x
11	6	2024-04-10 00:00:00.000000	50	MISCELLANEOUS	ONLINE	0x
12	6	2024-04-01 00:00:00.000000	100	MEDICAL	ONLINE	0x
13	6	2024-03-14 00:00:00.000000	20	MISCELLANEOUS	ONLINE	0x
14	7	2024-04-11 00:00:00.000000	3000	VEHICLE	DEBIT_CARD	0x
15	9	2024-04-14 00:00:00.000000	2400	VEHICLE	CREDIT_CARD	0x
16	9	2024-04-17 00:00:00.000000	450	GROCERY	ONLINE	0x
17	9	2024-04-16 00:00:00.000000	800	VEHICLE	CASH	0x
18	9	2024-04-16 00:00:00.000000	800	VEHICLE	CASH	0x
19	9	2024-04-16 00:00:00.000000	800	VEHICLE	CASH	0x
20	10	2024-04-15 00:00:00.000000	550	SHOPPING	ONLINE	0x
23	10	2024-04-08 00:00:00.000000	250	TRAVEL	CASH	0x
26	10	2024-04-10 00:00:00.000000	250	TRAVEL	CASH	0x
27	10	2024-04-18 00:00:00.000000	2000	VEHICLE	DEBIT_CARD	0x
28	10	2024-04-16 00:00:00.000000	125	SHOPPING	CASH	0x
29	10	2024-04-17 00:00:00.000000	300	TRAVEL	NET_BANKING	0x
30	11	2024-04-05 00:00:00.000000	350	TRAVEL	ONLINE	0x
42	11	2023-04-01 00:00:00.000000	1200	RENT	ONLINE	NULL
43	11	2023-04-05 00:00:00.000000	300	UTILITY	NET_BANKING	NULL
44	11	2023-04-10 00:00:00.000000	150	GROCERY	DEBIT_CARD	NULL
45	11	2023-04-12 00:00:00.000000	450	VEHICLE	CREDIT_CARD	NULL
46	11	2023-04-01 00:00:00.000000	2000	EMI	ONLINE	NULL
47	12	2024-04-20 00:00:00.000000	300	GROCERY	ONLINE	0x
48	12	2024-04-19 00:00:00.000000	150	TRAVEL	CASH	0x

31 rows in set (0.00 sec)

mysql> select * from expensegroups;

group_id	group_name	user_id
1	Family	1
2	NNNP	1
3	HighFive	3
4	LastBenchParty	10
9	Lalaland	10
10	sample	10
11	Lalaland	10
12	GONEGONE	11
13	sample	11
14	Cricket Gang	12

10 rows in set (0.00 sec)

mysql>

```
mysql> select * from group_members;
+-----------+----------+---------+
| member_id | group_id | user_id |
+-----------+----------+---------+
|         2 |       10 |      10 |
|         3 |       10 |       1 |
|         4 |       10 |       2 |
|         5 |        4 |       1 |
|         6 |        4 |       2 |
|         7 |       10 |       6 |
|         8 |        4 |       6 |
|         9 |       11 |      10 |
|        10 |       11 |       6 |
|        11 |       11 |       7 |
|        12 |       12 |      11 |
|        13 |       12 |       6 |
|        14 |       12 |       7 |
|        15 |       12 |      10 |
|        17 |       14 |      12 |
+-----------+----------+---------+
15 rows in set (0.00 sec)
```

```
mysql> select * from group_members;
+-----------+----------+---------+
| member_id | group_id | user_id |
+-----------+----------+---------+
|         2 |       10 |      10 |
|         3 |       10 |       1 |
|         4 |       10 |       2 |
|         5 |        4 |       1 |
|         6 |        4 |       2 |
|         7 |       10 |       6 |
|         8 |        4 |       6 |
|         9 |       11 |      10 |
|        10 |       11 |       6 |
|        11 |       11 |       7 |
|        12 |       12 |      11 |
|        13 |       12 |       6 |
|        14 |       12 |       7 |
|        15 |       12 |      10 |
|        17 |       14 |      12 |
|        18 |       14 |       6 |
|        19 |       14 |       7 |
|        20 |       14 |      10 |
|        21 |       14 |      11 |
+-----------+----------+---------+
19 rows in set (0.00 sec)
```

```
mysql> select * from groupexpenses;
+----+----------------+------------------+----------------------------+--------------------------+----------+---------+
| id | expense_amount | expense_category | expense_date               | expense_transaction_type | group_id | user_id |
+----+----------------+------------------+----------------------------+--------------------------+----------+---------+
|  1 |        2500.00 | GROCERY          | 2024-04-18 00:00:00.000000 | CASH                     |        4 |      10 |
|  2 |        1000.00 | VEHICLE          | 2024-04-09 00:00:00.000000 | CASH                     |        4 |      10 |
|  3 |        2000.00 | VEHICLE          | 2024-04-10 00:00:00.000000 | CASH                     |        4 |      10 |
|  4 |         500.00 | TRAVEL           | 2024-04-02 00:00:00.000000 | CASH                     |        9 |      10 |
|  5 |         500.00 | VEHICLE          | 2024-04-09 00:00:00.000000 | CASH                     |       11 |      10 |
|  6 |        1000.00 | VEHICLE          | 2024-04-10 00:00:00.000000 | CASH                     |       12 |      11 |
|  7 |        1500.00 | VEHICLE          | 2024-04-20 00:00:00.000000 | ONLINE                   |       14 |      12 |
|  8 |        1500.00 | TRAVEL           | 2024-04-20 00:00:00.000000 | CASH                     |       14 |      12 |
+----+----------------+------------------+----------------------------+--------------------------+----------+---------+
8 rows in set (0.00 sec)
```

IX. Github Link

https://github.com/NidhiPandukalGururaj/ExpenseTracker.git

At this moment the repository is made private.

Mail the collaborators of the project requesting access.

1)nidhigururaj1801@gmail.com
2)nidasamreen1@gmail.com
3)nikkisuresh03@gmail.com
4)pramathpramath30@gmail.com

Thank You!!