**Table of Content**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Subject** | **Page No.** |
| **1** | Project Profile & Company Profile |  |
| **2** | Introduction to Tools |  |
| **3** | System Study (Requirement Analysis) |  |
|  | **3.1** Existing System |  |
|  | **3.2** Proposed System |  |
|  | **3.3** Scope of the Proposed System |  |
|  | **3.4** Aims and Objectives |  |
|  | **3.5** Feasibility Study |  |
|  | **3.5.1** Operational Feasibility |  |
|  | **3.5.2** Technical Feasibility |  |
|  | **3.5.3** Economic Feasibility |  |
| **4** | System Analysis |  |
|  | **4.1** Requirements Specification (With System Modules) |  |
|  | **4.2** Use Case Diagram |  |
|  | **4.3** Activity Diagram |  |
|  | **4.4** Class Diagram / System Flowchart |  |
|  | **4.5** Data Flow Diagram |  |
|  | **4.5.1** Context Level DFD |  |
|  | **4.5.2** First Level DFD |  |
|  | **4.5.3** Second Level DFD |  |
|  | **4.6** ER Diagram |  |
| **5** | System Design |  |
|  | **5.1** Data Dictionary |  |
|  | **5.2** Screen Layouts / UI Design |  |
|  | **5.3** Reports |  |
| **6** | System Implementation |  |
|  | **6.1** Module Implementation |  |
| **7** | System Testing |  |
|  | **7.1** Testing Strategies |  |
|  | **7.2** Test Cases |  |
| **8** | Future Enhancement |  |
| **9** | Bibliography/References |  |
| **10** | Internal Guide’s Report |  |

**1.Project Profile**

**Project Name :** SplitBuddy - The Real Time Group Expense Sharing

**Type of Application :** Web Application

**Project Description:**  
 A web-based platform designed to **split, track, and manage real-time expenses** among groups and their members. The system allows users to create groups, add participants, log shared expenses, and automatically calculate each member’s share. It supports real-time updates, payment tracking, and settlement summaries, ensuring transparent and efficient financial collaboration for activities like trips, events, or shared households.

**Team Members :** 3

**Front End :** React,Bootstrap

**Backend :** Node.js, Express

**Database :** MySql

**Tools Used :** Visual Studio Code

### **Introduction to Tools**

**Front-end Tool:**

React,Bootstrap

**Back-end Tool:**

Node.js,Express

1. **System Study**

**3.1 Existing System**

At present, people usually manage shared expenses in two main ways:

Manual methods such as writing amounts on paper or using spreadsheets.

Basic mobile apps that don’t always offer real-time synchronization or clear visibility for everyone.

Both approaches can easily lead to mistakes, slow updates, and even arguments among group members.

**3.2 Proposed System**

SplitBuddy is designed to solve these problems by offering a single, easy-to-use platform where groups can manage all their shared expenses. The system provides:

A **centralized space** to record and monitor expenses.

**Instant updates** so every member sees changes as soon as they happen.

**Secure settlement tracking** and clear reports to keep everyone on the same page.

**3.3 Scope of the Proposed System**

The application is suitable for any situation where people share costs, including:

Trips, tours, and vacations.

Roommates or shared households.

Group events, office parties, or club activities.

Any other collaborative activity that needs transparent expense management.

**3.4 Aim and Objective of the Proposed System**

SplitBuddy’s main purpose is to make group expense sharing simple, fair, and transparent. Its objectives are to:

Automate how expenses are divided among group members.

Keep a clear history of settlements and provide detailed reports

Enable safe, real-time collaboration on any device with an internet connection.

**3.5 Feasibility Study**

**3.5.1 Operational Feasibility**

The application is web-based with a clean, intuitive interface. Users need little to no training to start managing their expenses.

**3.5.2 Technical Feasibility**

SplitBuddy is built using proven technologies—React for the front end, Node.js with Express for the back end, and MySQL for data storage. These tools are widely supported, scalable, and reliable.

**3.5.3 Economical Feasibility**

Development and upkeep costs are relatively low, and the app can be deployed on affordable cloud hosting services, making it budget-friendly.

1. **System Analysis**

**4.1 Requirements Specification (With System Modules)**

#### Functional Requirements

* User registration and authentication
* Group creation, member management
* Add, edit, delete expenses
* Automatic calculation of each member’s share
* Settlement management
* Notifications
* Admin support (manage user , group monitoring , resolve queries)

#### Non-Functional Requirements

* Responsive UI
* Data security (password hashing, role-based access)
* High availability and scalability
* Real-time updates via APIs/web sockets

#### System Modules

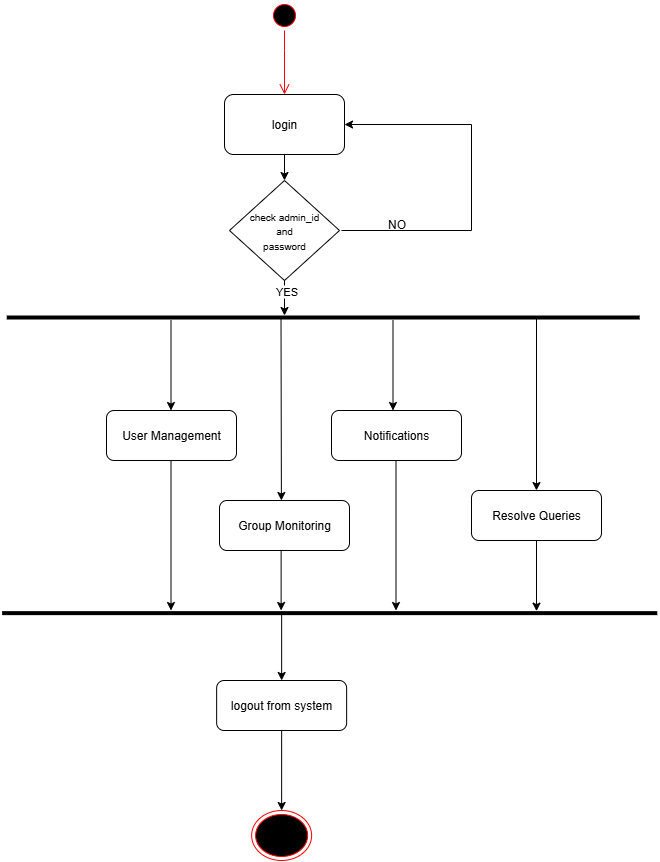
* User Module
* Group & Member Module
* Expense & Split Module
* Settlement Module
* Notification Module
* Admin Support Module

**4.2 Use Case Diagram**

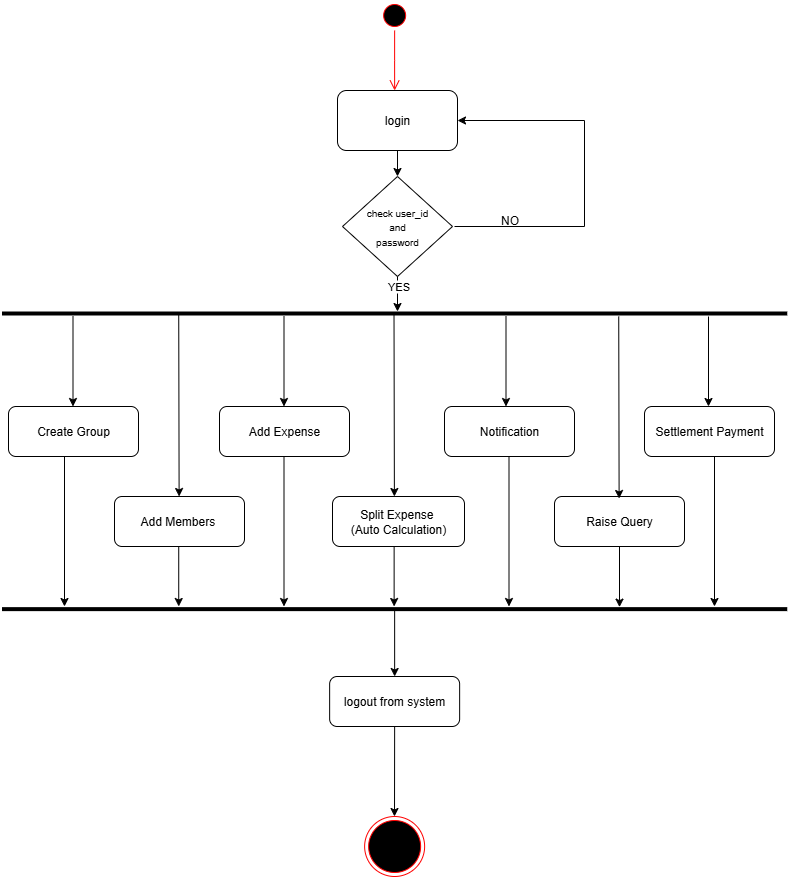


**4.3 Activity Diagram**

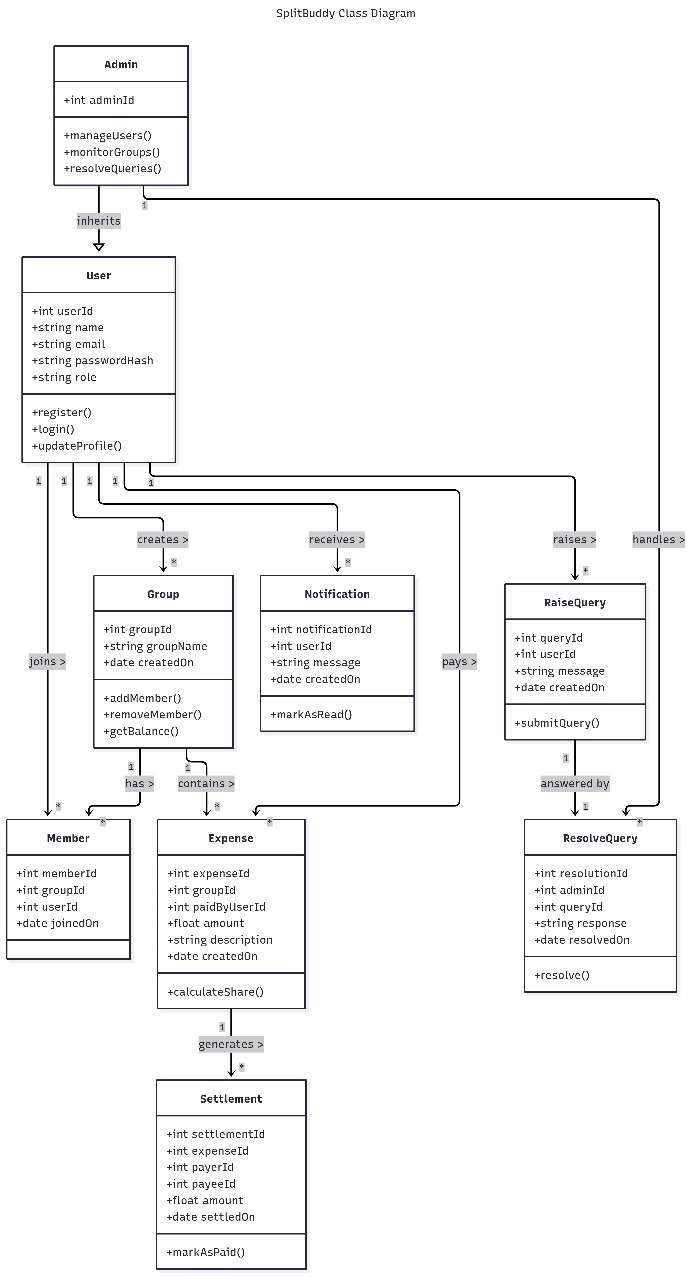
Admin:

****

User:



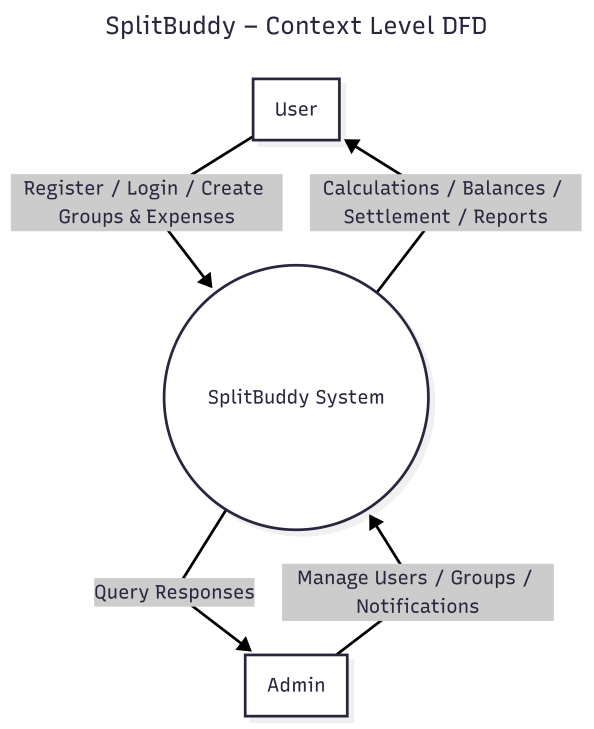
**4.4Class Diagram / System Flowchart**

****

**4.5 Data Flow Diagram**

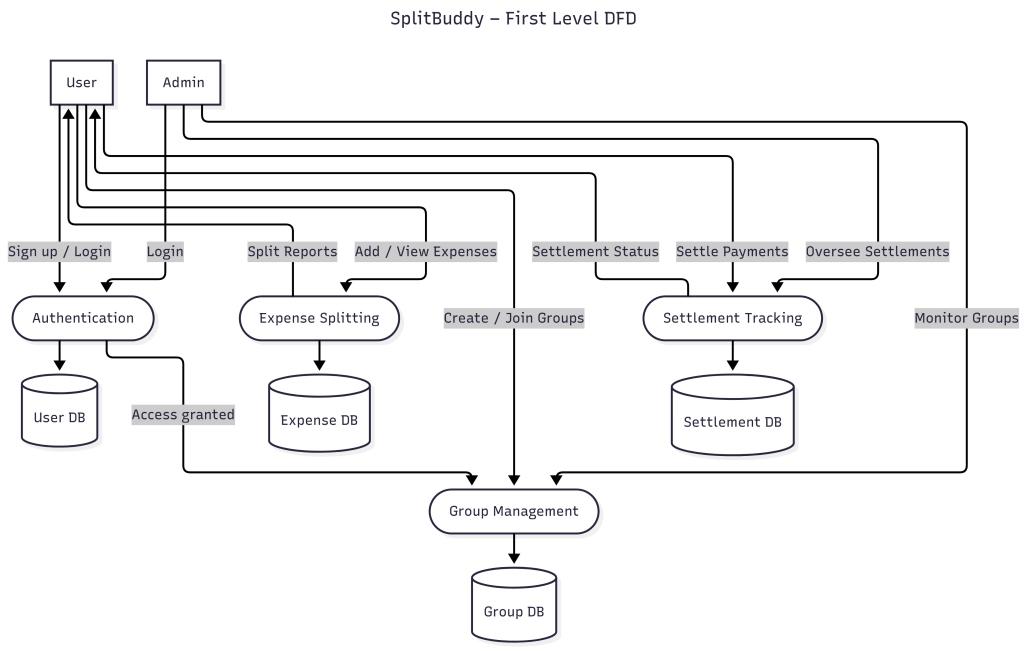
**4.5.1 Context Level DFD**

* System receives input (users, groups, expenses) and outputs calculations/reports.

****

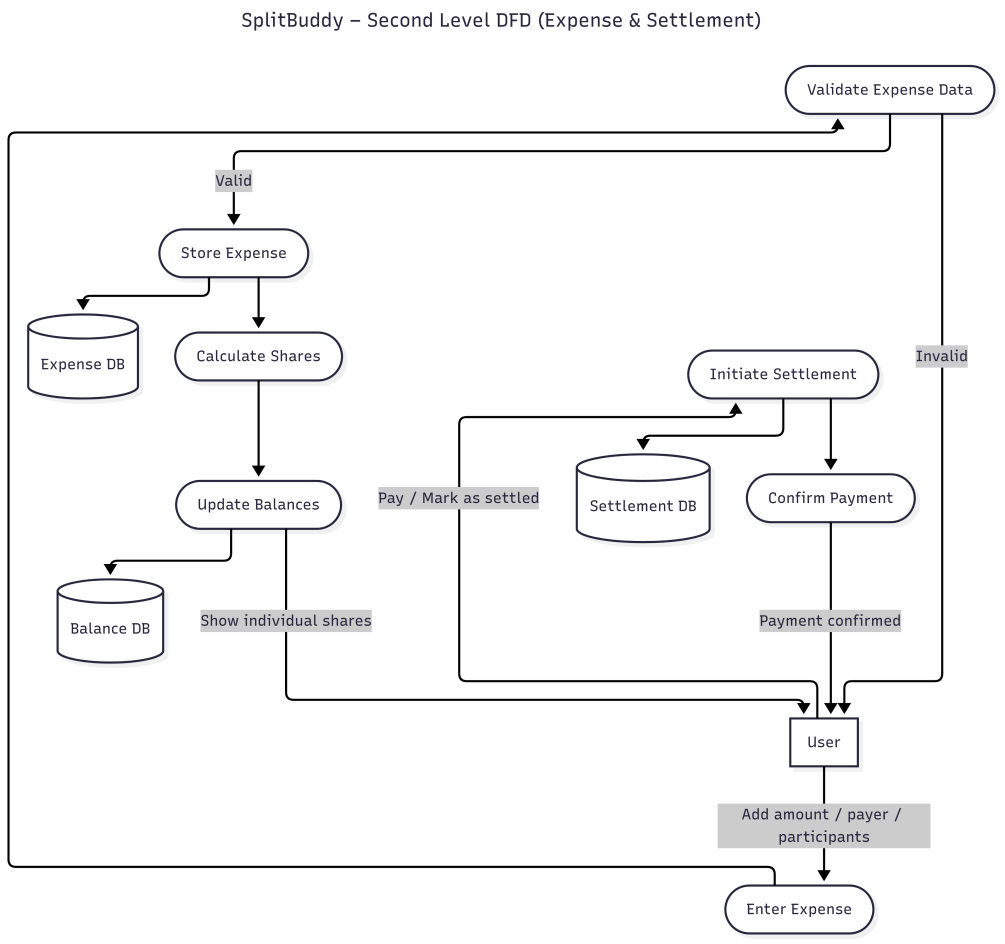
**4.5.2 First Level DFD**

* Split modules: Authentication, Group Management, Expense Splitting, Settlements.

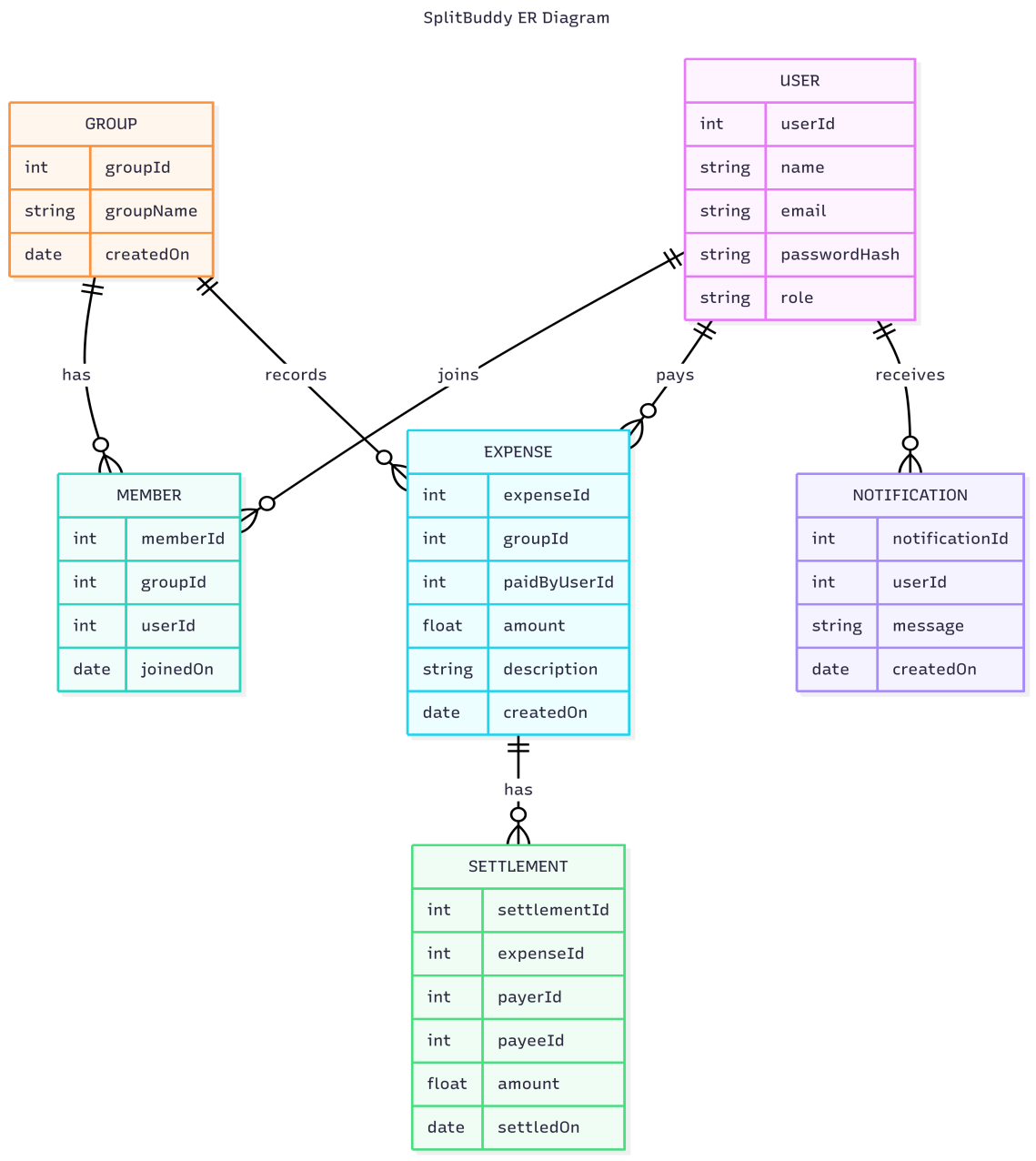
****

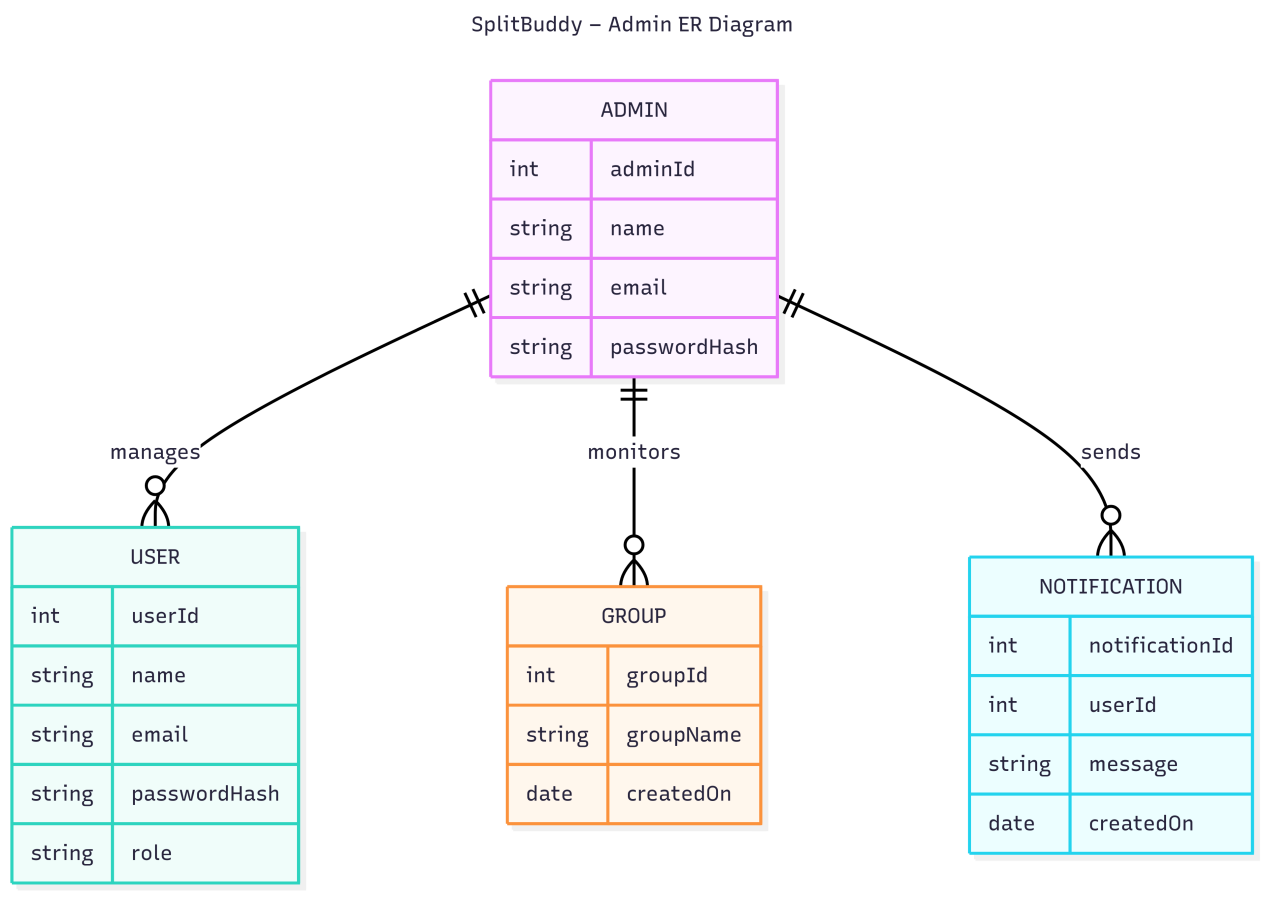
**4.5.3 Second Level DFD**

* Drill down into expense split and settlement calculation.



**4.6 ER Diagram**

****

****

1. **System Design**

**5.1 Data Dictionary**

### Table Name: users

| **Column Name** | **Data Type** | **Constraints / Default** | **Description** |
| --- | --- | --- | --- |
| user\_id | INT (AUTO\_INCREMENT) | **Primary Key**, Auto-increment | Unique identifier for each user. |
| name | VARCHAR(100) | NOT NULL | Full name of the user. |
| email | VARCHAR(100) | UNIQUE, NOT NULL | User’s email address (must be unique). |
| password | VARCHAR(255) | NOT NULL | Encrypted/hashed password for authentication. |
| role | ENUM('admin','user') | Default: 'user' | Role assigned to the user (either admin or user). |
| profile\_photo | VARCHAR(255) | NULL allowed | File path or URL of the user’s profile photo. |
| currency | VARCHAR(10) | Default: 'INR' | Preferred currency of the user. |
| timezone | VARCHAR(50) | Default: 'Asia/Kolkata' | User’s preferred timezone. |
| created\_at | TIMESTAMP | Default: CURRENT\_TIMESTAMP | Timestamp when the record was created. |

### Table Name: groups

| **Column Name** | **Data Type** | **Constraints / Default** | **Description** |
| --- | --- | --- | --- |
| group\_id | INT (AUTO\_INCREMENT) | **Primary Key**, Auto-increment | Unique identifier for each group. |
| group\_name | VARCHAR(100) | NOT NULL | Name of the group. |
| created\_by | INT | NOT NULL, **Foreign Key** → users(user\_id)ON DELETE CASCADE | ID of the user who created the group. |
| created\_at | TIMESTAMP | Default: CURRENT\_TIMESTAMP | Date and time when the group was created. |

### Table Name: group\_members

| **Column Name** | **Data Type** | **Constraints / Default** | **Description** |
| --- | --- | --- | --- |
| member\_id | INT (AUTO\_INCREMENT) | **Primary Key**, Auto-increment | Unique identifier for each membership record. |
| group\_id | INT | NOT NULL, **Foreign Key** → groups(group\_id) ON DELETE CASCADE | ID of the group the user is a member of. |
| user\_id | INT | NOT NULL, **Foreign Key** → users(user\_id) ON DELETE CASCADE | ID of the user who is a member of the group. |
| joined\_at | TIMESTAMP | Default: CURRENT\_TIMESTAMP | Date and time when the user joined the group. |

### Table Name: expenses

| **Column Name** | **Data Type** | **Constraints / Default** | **Description** |
| --- | --- | --- | --- |
| expense\_id | INT (AUTO\_INCREMENT) | **Primary Key**, Auto-increment | Unique identifier for each expense record. |
| group\_id | INT | NOT NULL, **Foreign Key** → groups(group\_id) ON DELETE CASCADE | ID of the group associated with the expense. |
| paid\_by | INT | NOT NULL, **Foreign Key** → users(user\_id) ON DELETE CASCADE | ID of the user who paid for the expense. |
| description | VARCHAR(255) | NOT NULL | Short text describing the expense. |
| category | VARCHAR(50) | NULL allowed | Category or type of the expense (e.g., food, travel). |
| amount | INT | NOT NULL | Total amount of the expense. |
| currency | VARCHAR(10) | Default: 'INR' | Currency in which the expense is recorded. |
| created\_at | TIMESTAMP | Default: CURRENT\_TIMESTAMP | Date and time when the expense was created. |

### Table Name: expense\_splits

| **Column Name** | **Data Type** | **Constraints / Default** | **Description** |
| --- | --- | --- | --- |
| split\_id | INT (AUTO\_INCREMENT) | **Primary Key**, Auto-increment | Unique identifier for each split record. |
| expense\_id | INT | NOT NULL, **Foreign Key** → expenses(expense\_id) ON DELETE CASCADE | ID of the expense this split belongs to. |
| user\_id | INT | NOT NULL, **Foreign Key** → users(user\_id) ON DELETE CASCADE | ID of the user responsible for this share of the expense. |
| share\_amount | INT | NOT NULL | Amount allocated to the user from the total expense. |

### Table Name: settlements

| **Column Name** | **Data Type** | **Constraints / Default** | **Description** |
| --- | --- | --- | --- |
| settlement\_id | INT (AUTO\_INCREMENT) | **Primary Key**, Auto-increment | Unique identifier for each settlement record. |
| group\_id | INT | NOT NULL, **Foreign Key** → groups(group\_id) ON DELETE CASCADE | ID of the group in which the settlement occurs. |
| from\_user | INT | NOT NULL, **Foreign Key** → users(user\_id) | ID of the user who pays the settlement amount. |
| to\_user | INT | NOT NULL, **Foreign Key** → users(user\_id) | ID of the user who receives the settlement amount. |
| amount | INT | NOT NULL | Amount of money being settled. |
| status | ENUM('pending','completed') | Default: 'pending' | Status of the settlement (either pending or completed). |
| settled\_at | TIMESTAMP | NULL allowed | Date and time when the settlement was completed. |

### Table Name: notifications

| **Column Name** | **Data Type** | **Constraints / Default** | **Description** |
| --- | --- | --- | --- |
| notification\_id | INT (AUTO\_INCREMENT) | **Primary Key**, Auto-increment | Unique identifier for each notification. |
| user\_id | INT | NOT NULL, **Foreign Key** → users(user\_id) ON DELETE CASCADE | ID of the user to whom the notification belongs. |
| message | TEXT | NOT NULL | The content or body of the notification. |
| is\_read | BOOLEAN | Default: FALSE | Flag indicating whether the notification has been read (TRUE) or not (FALSE). |
| created\_at | TIMESTAMP | Default: CURRENT\_TIMESTAMP | Date and time when the notification was created. |

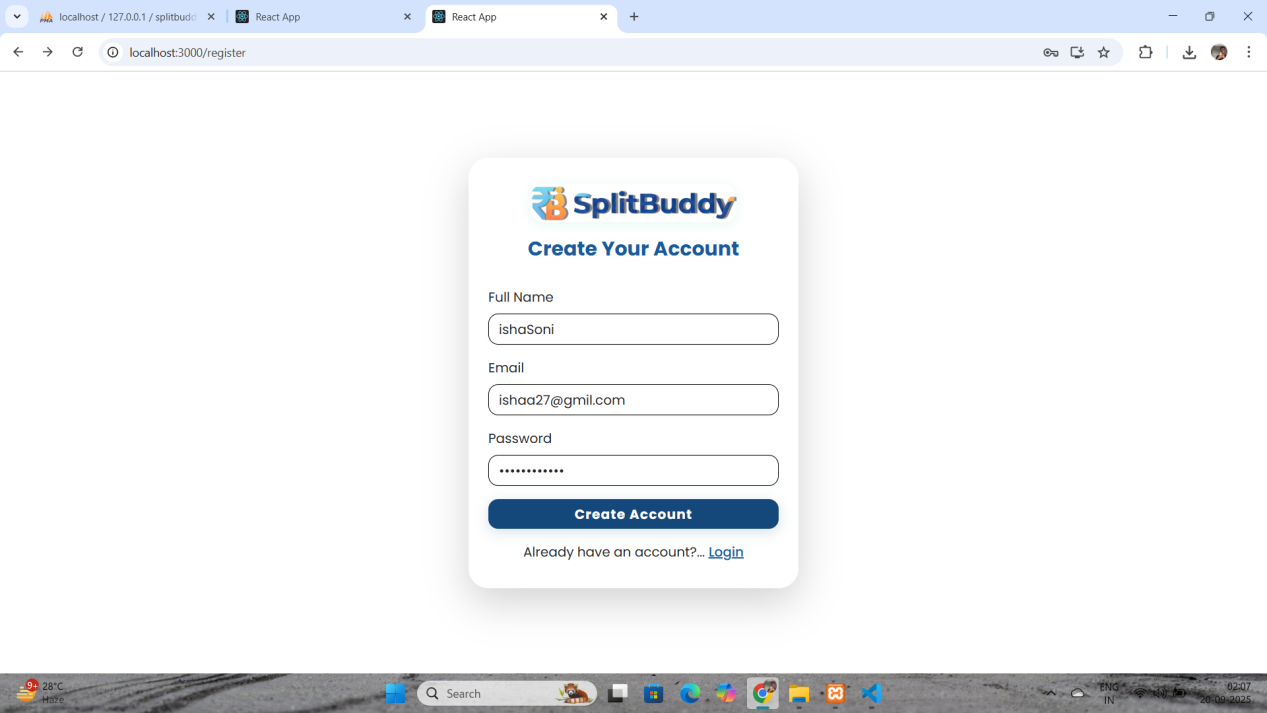
### Table Name: queries (Admin Support)

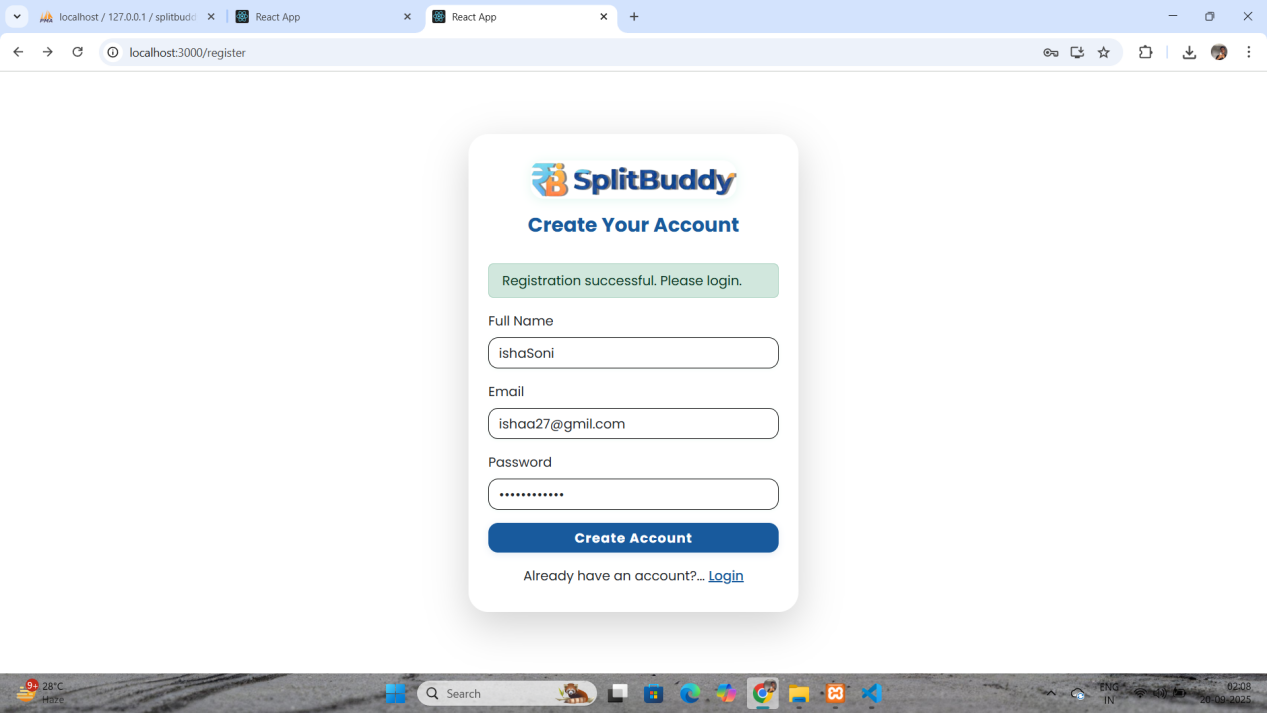
| **Column Name** | **Data Type** | **Constraints / Default** | **Description** |
| --- | --- | --- | --- |
| query\_id | INT (AUTO\_INCREMENT) | **Primary Key**, Auto-increment | Unique identifier for each support query. |
| user\_id | INT | NOT NULL, **Foreign Key** → users(user\_id) ON DELETE CASCADE | ID of the user who submitted the query. |
| category | ENUM('bug','payment','general') | Default: 'general' | Type of query: bug, payment, or general. |
| message | TEXT | NOT NULL | The content or details of the query. |
| status | ENUM('open','resolved') | Default: 'open' | Current status of the query (open or resolved). |
| created\_at | TIMESTAMP | Default: CURRENT\_TIMESTAMP | Date and time when the query was submitted. |

**5.2 Screen Layouts / UI Design**

Key screens:

User Side :Registartion Page :

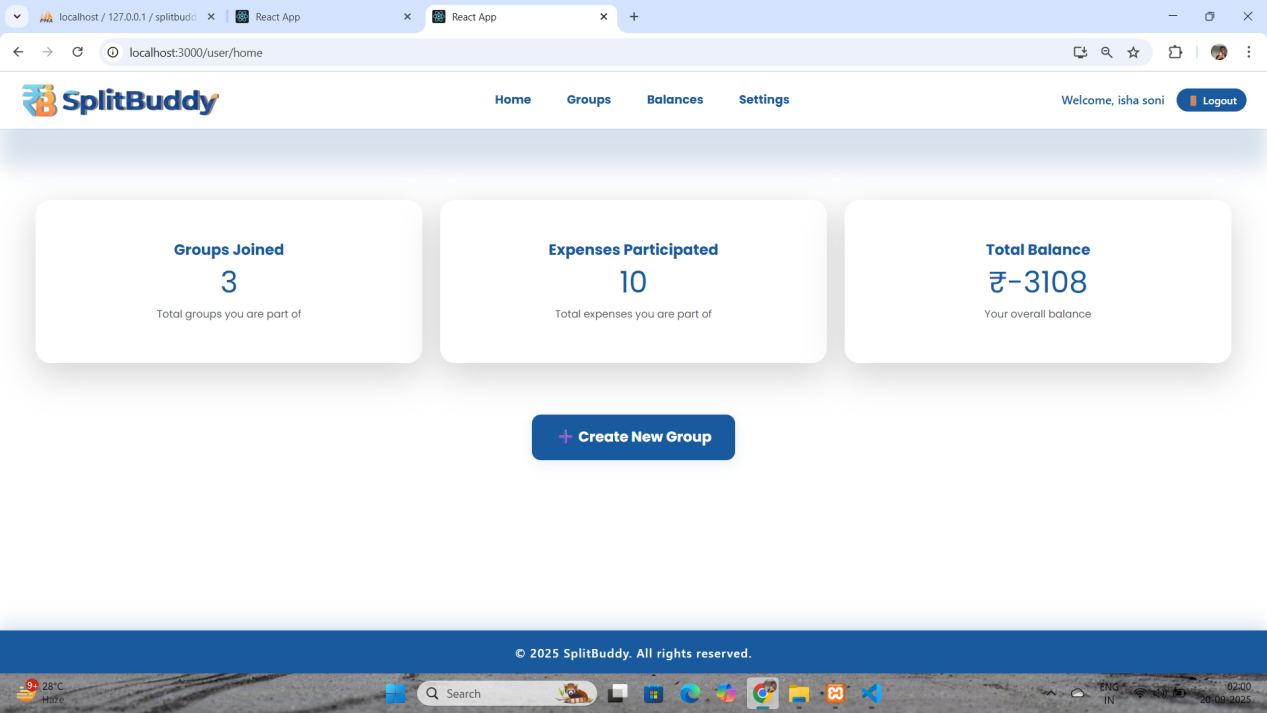




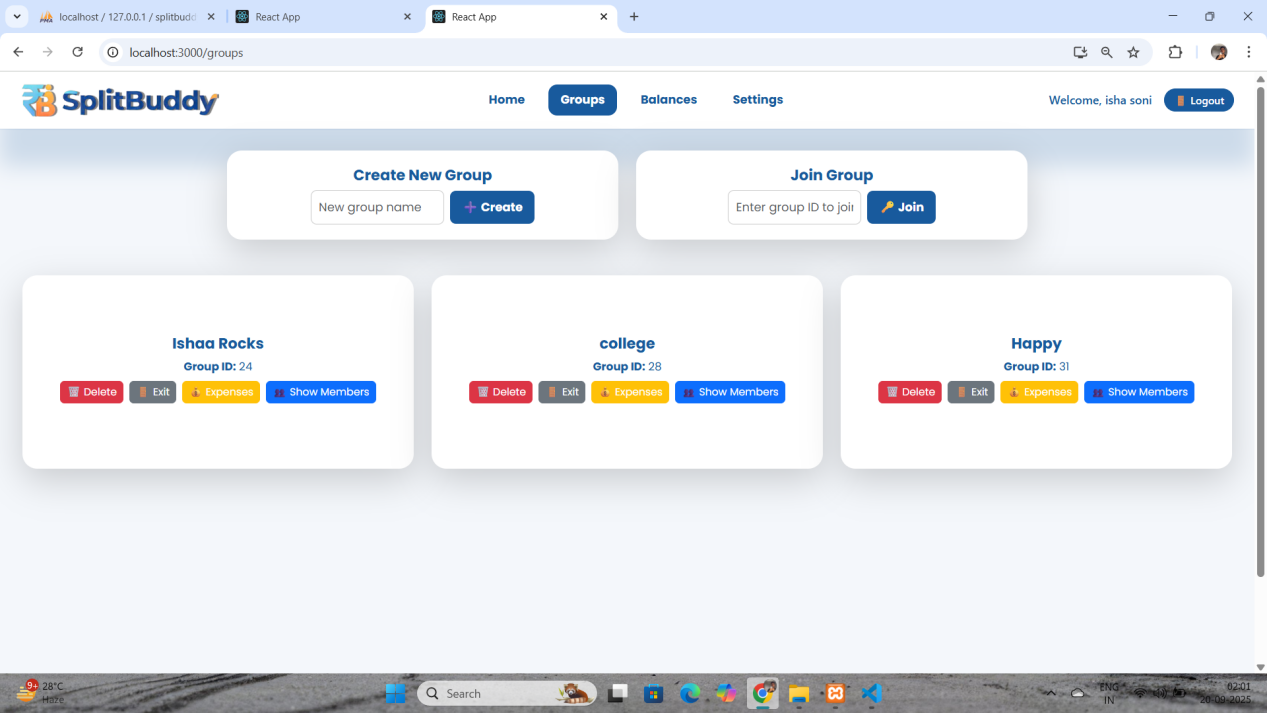
* Login / Sign-up

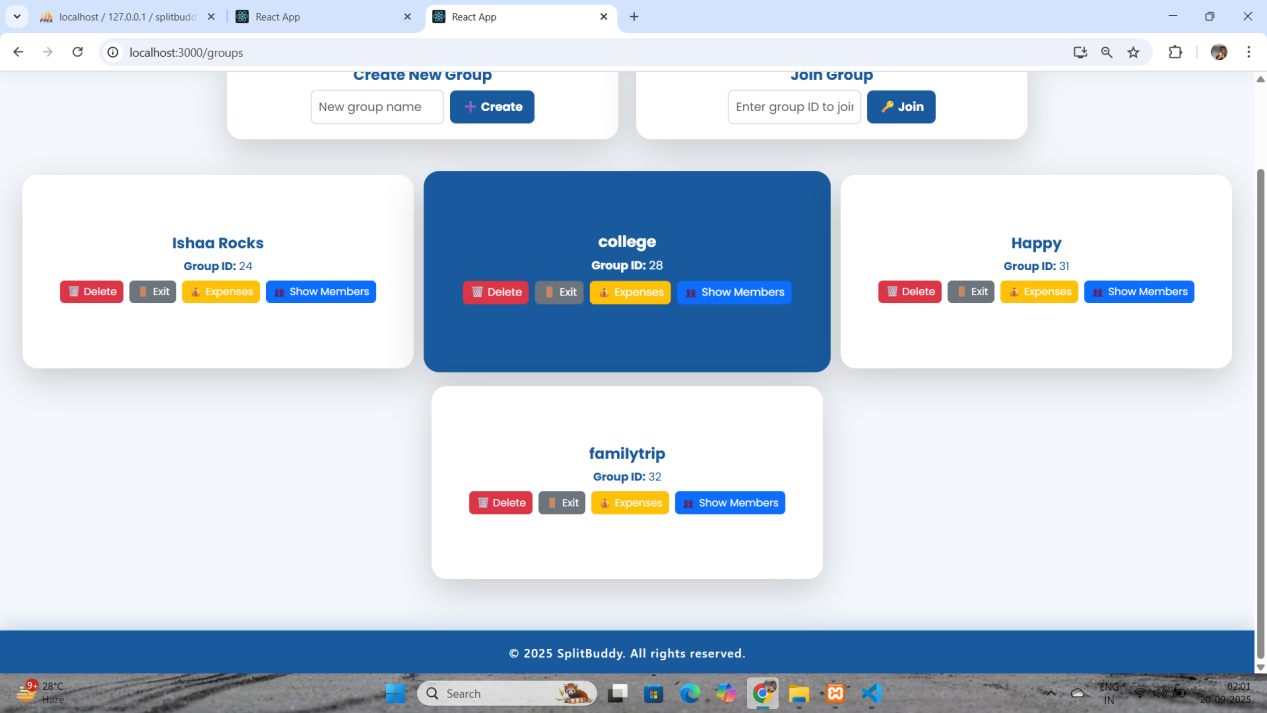


* Dashboard (overview of groups & balances)

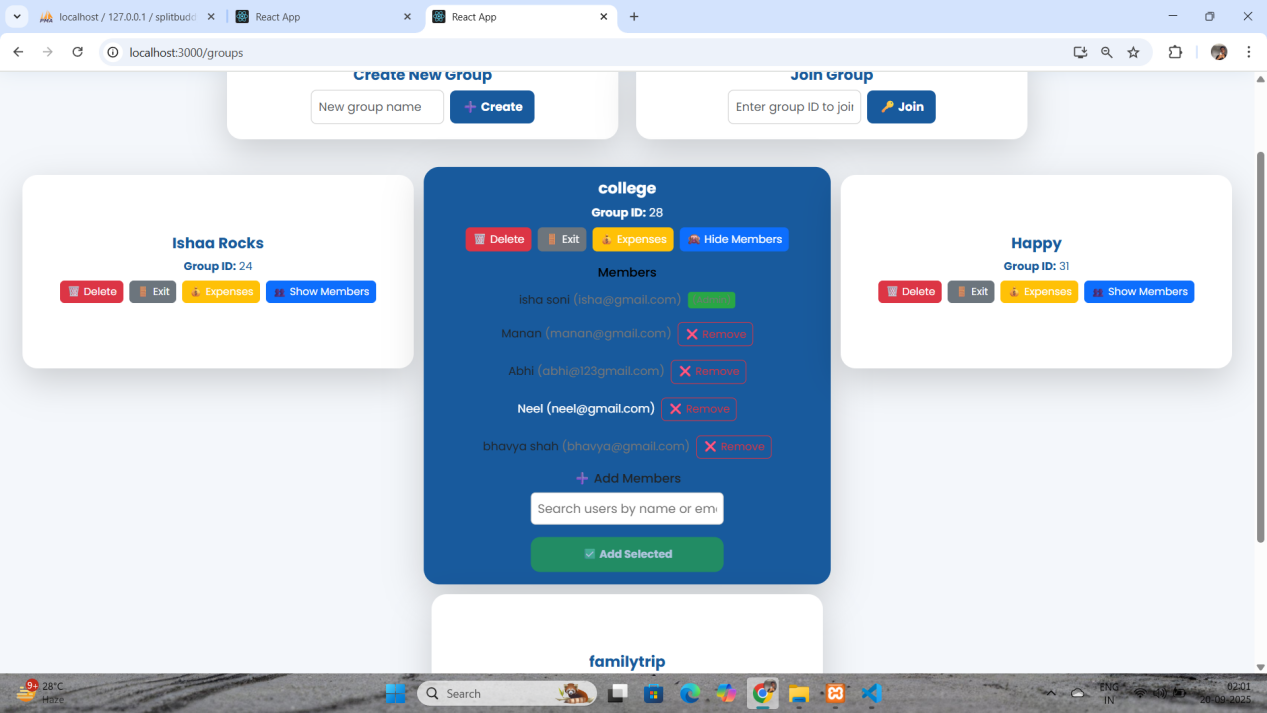


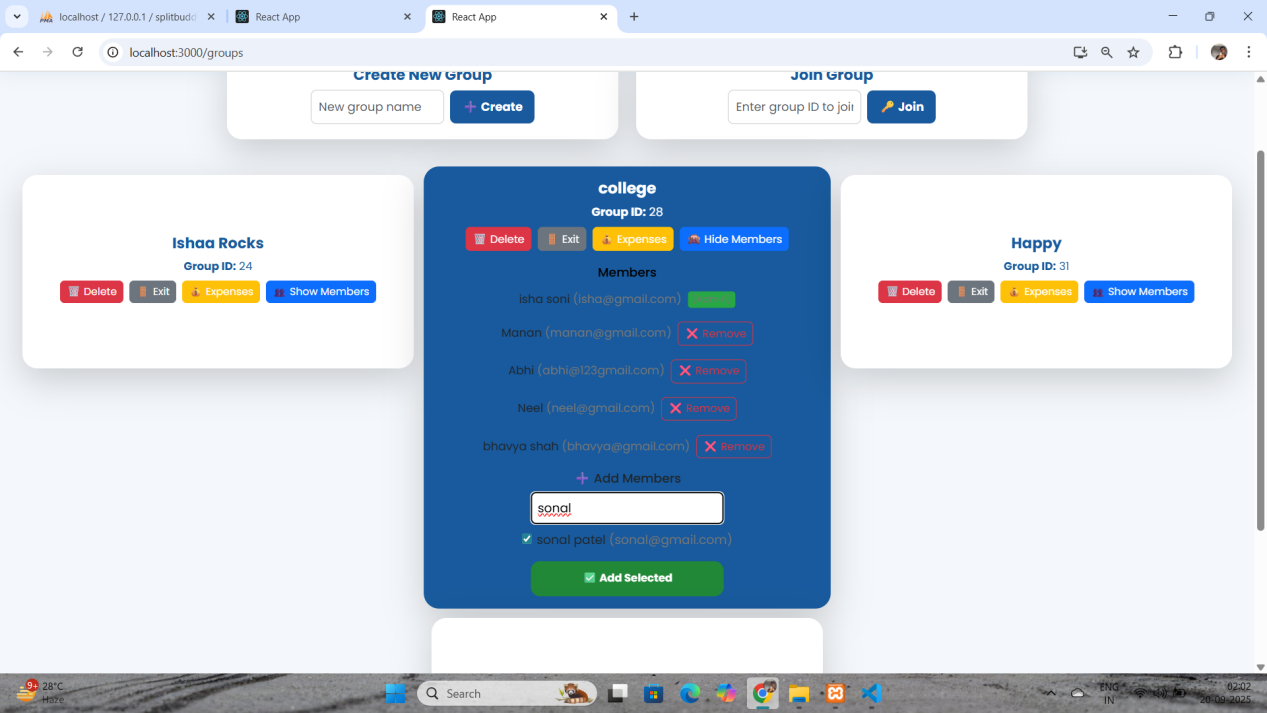
* Group details page





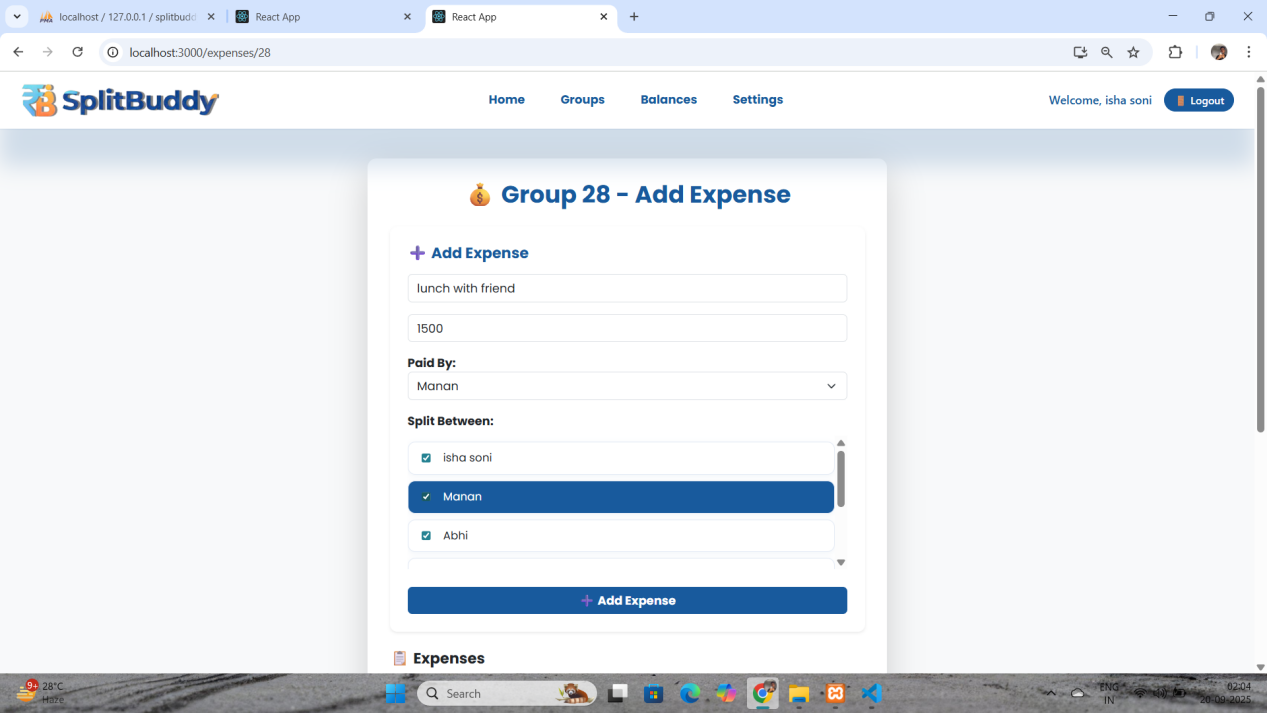
Add Members :



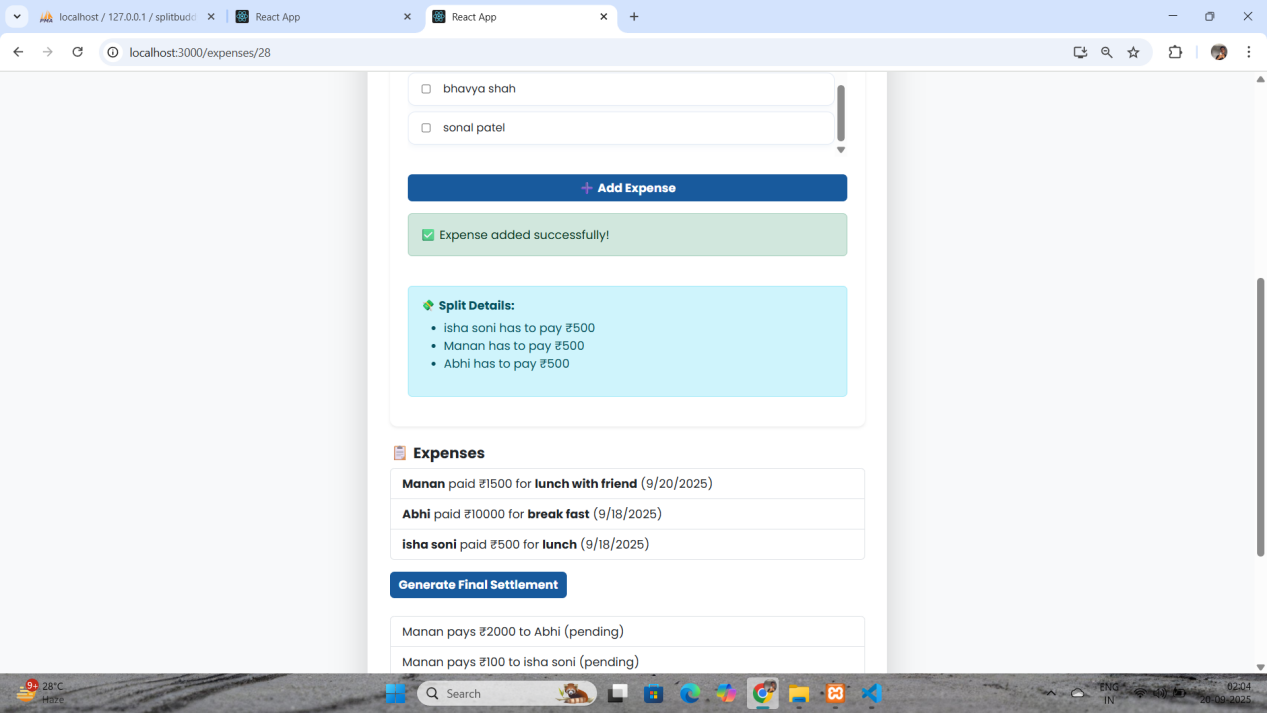




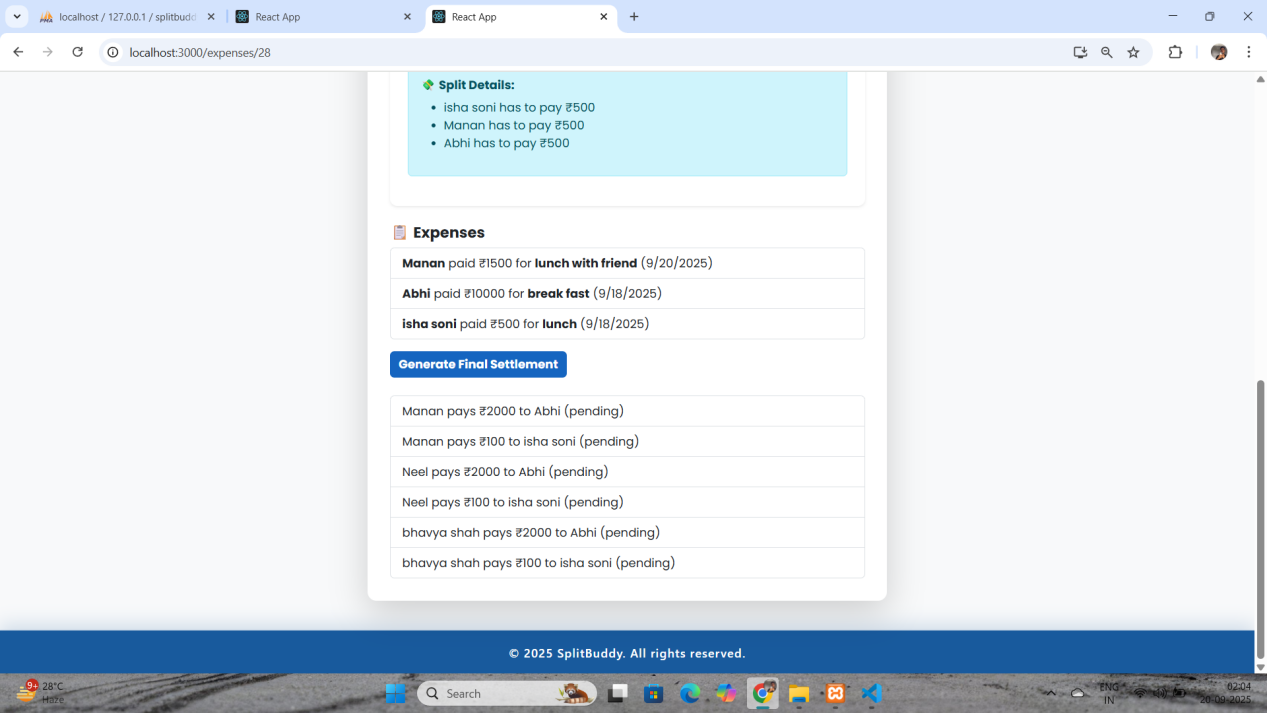
Add Expense :

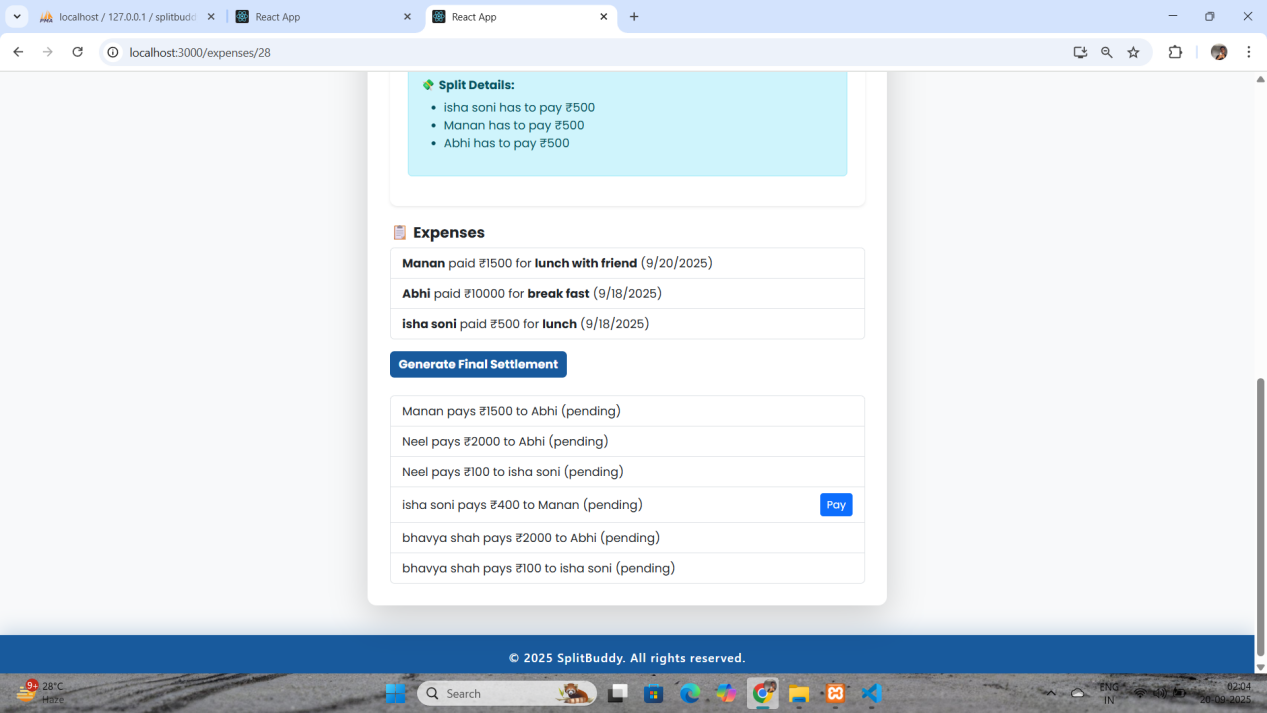


Split Expense :

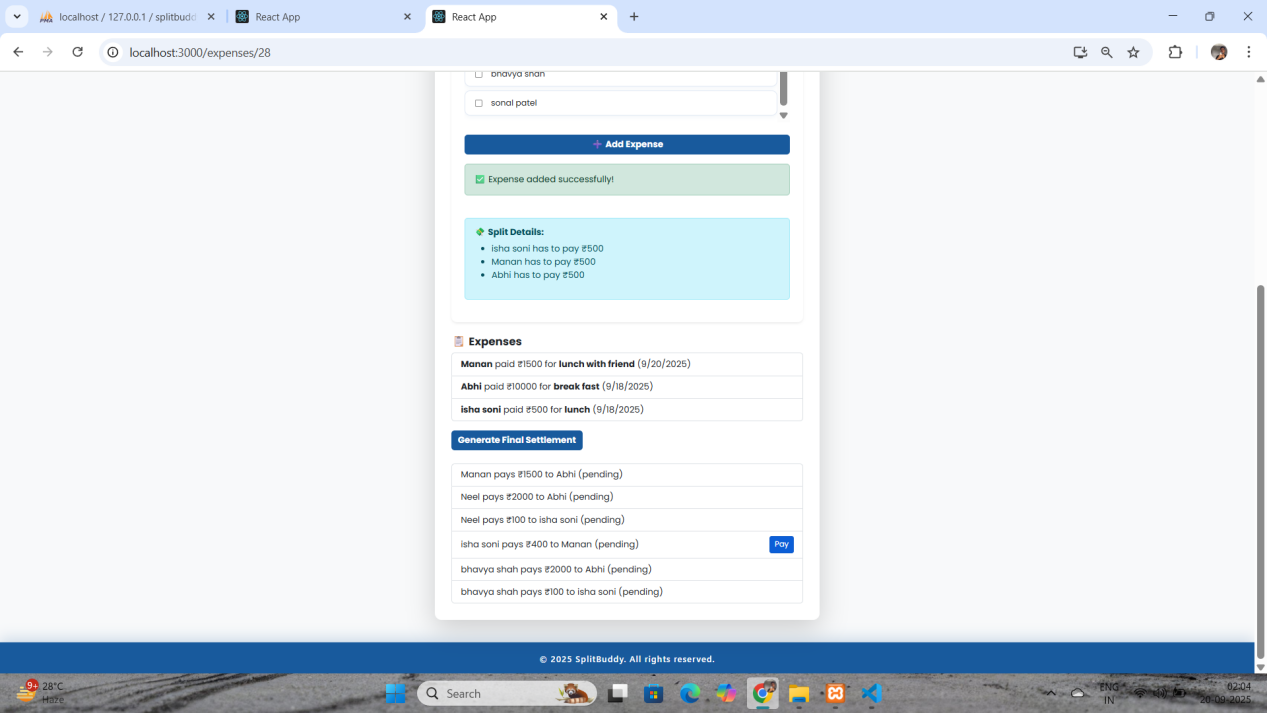


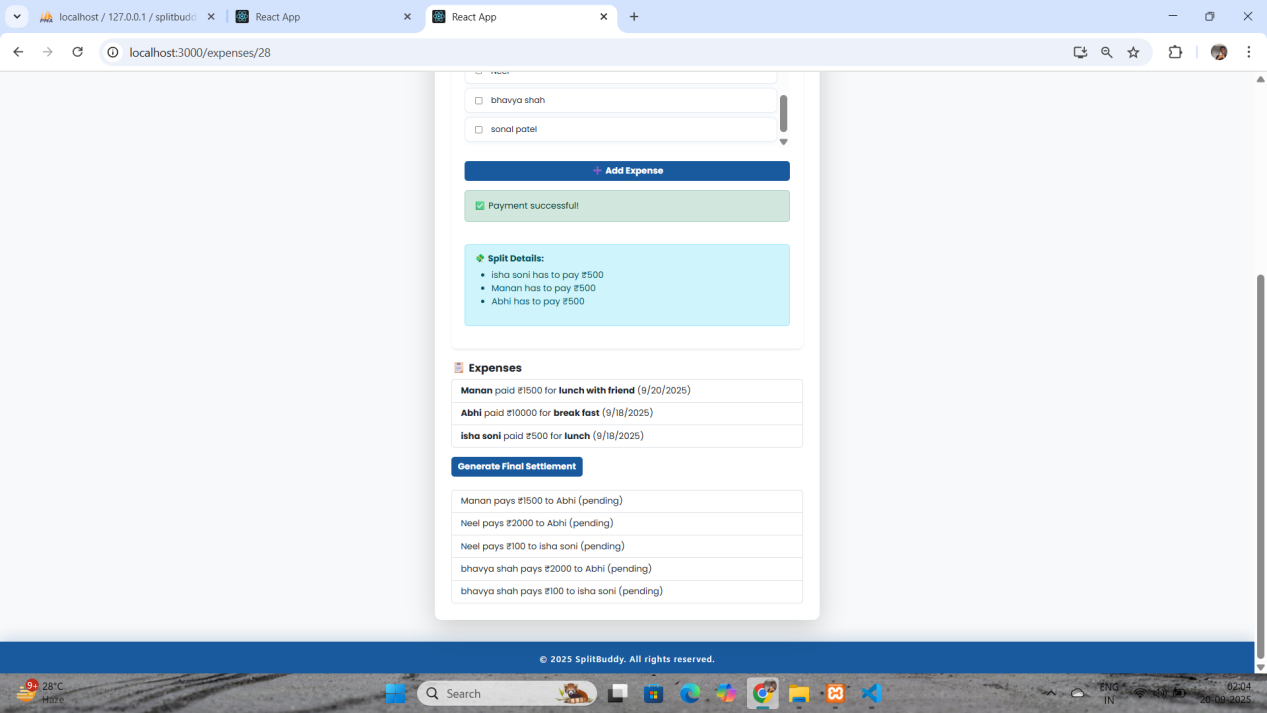
Generate Final Settlement :

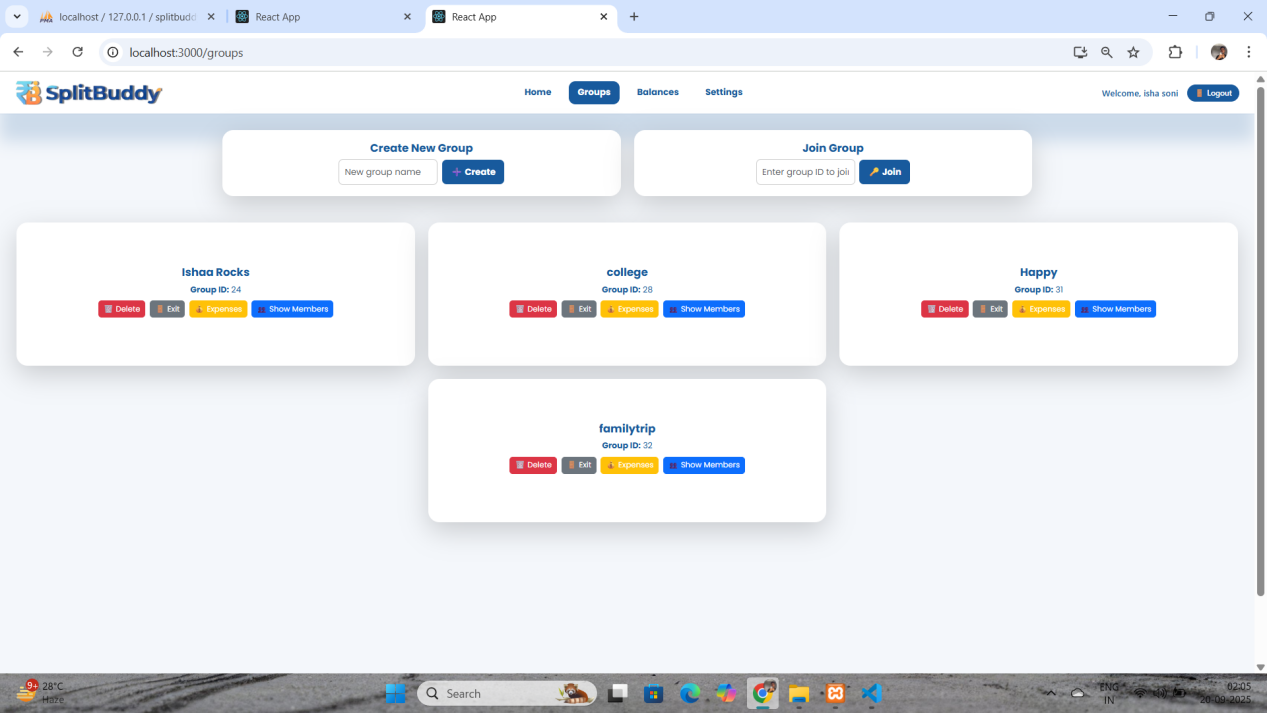




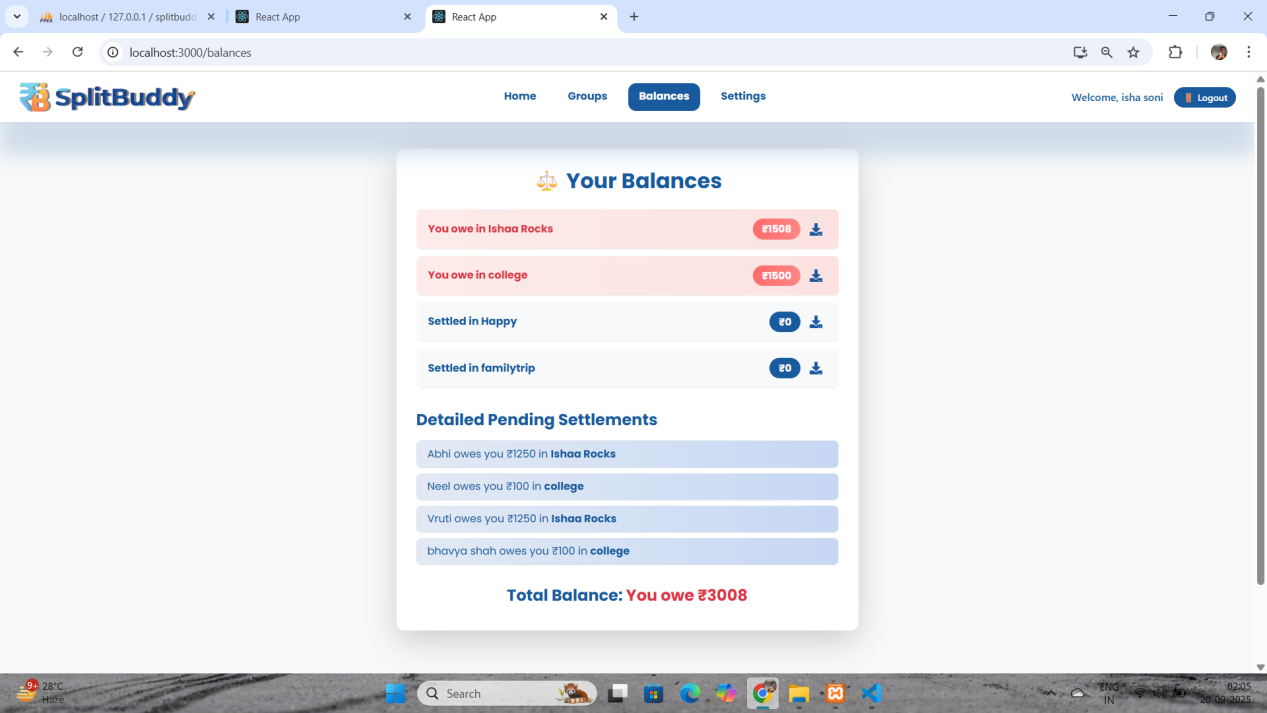
Settlement Payment :



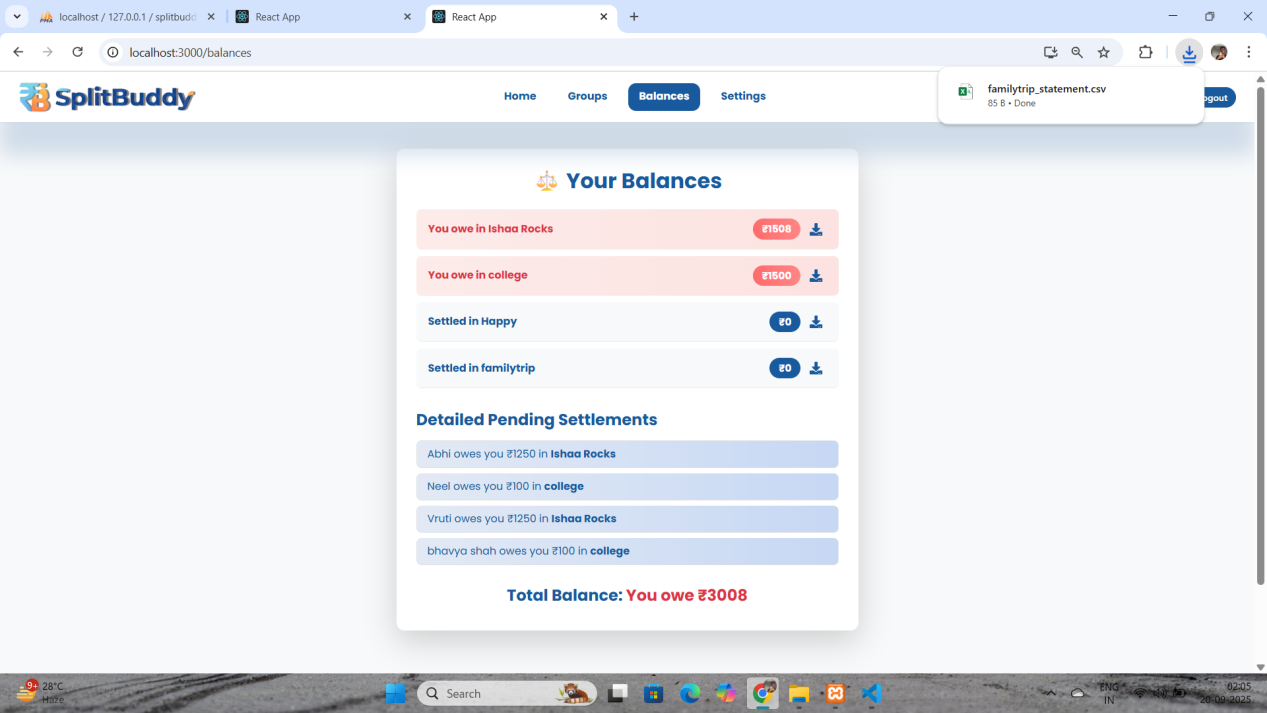




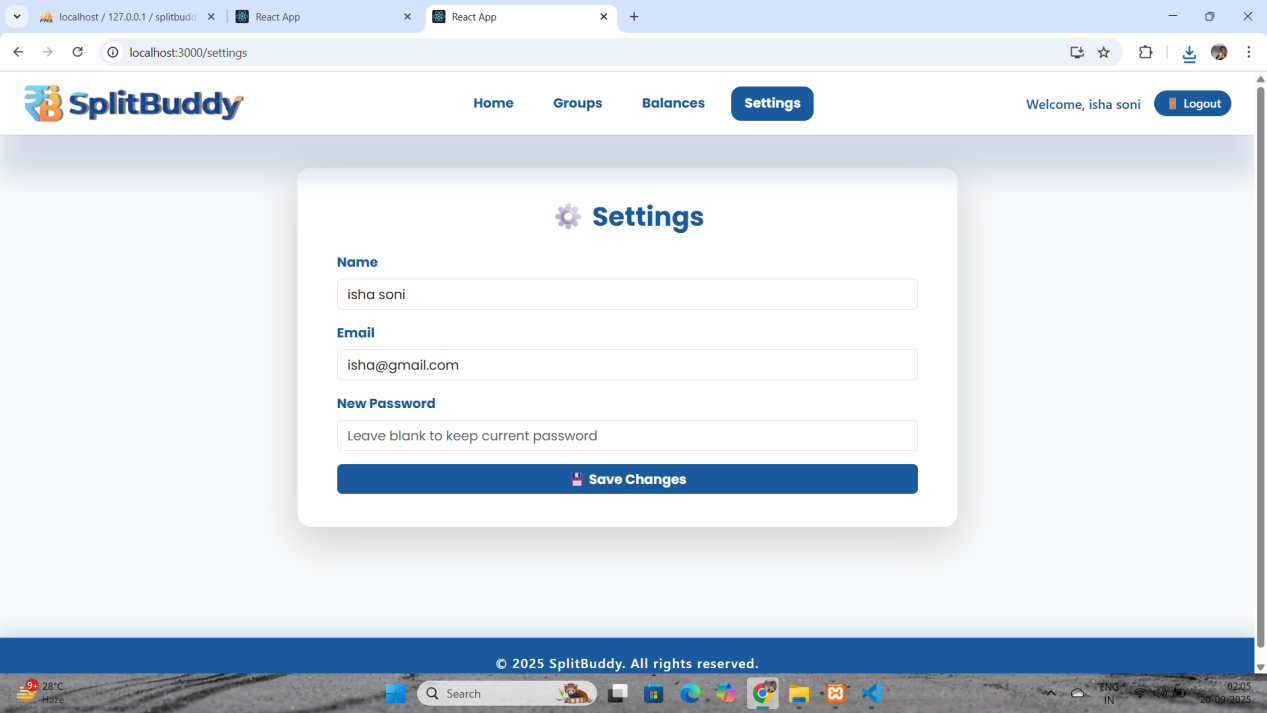
* Reports & summaries

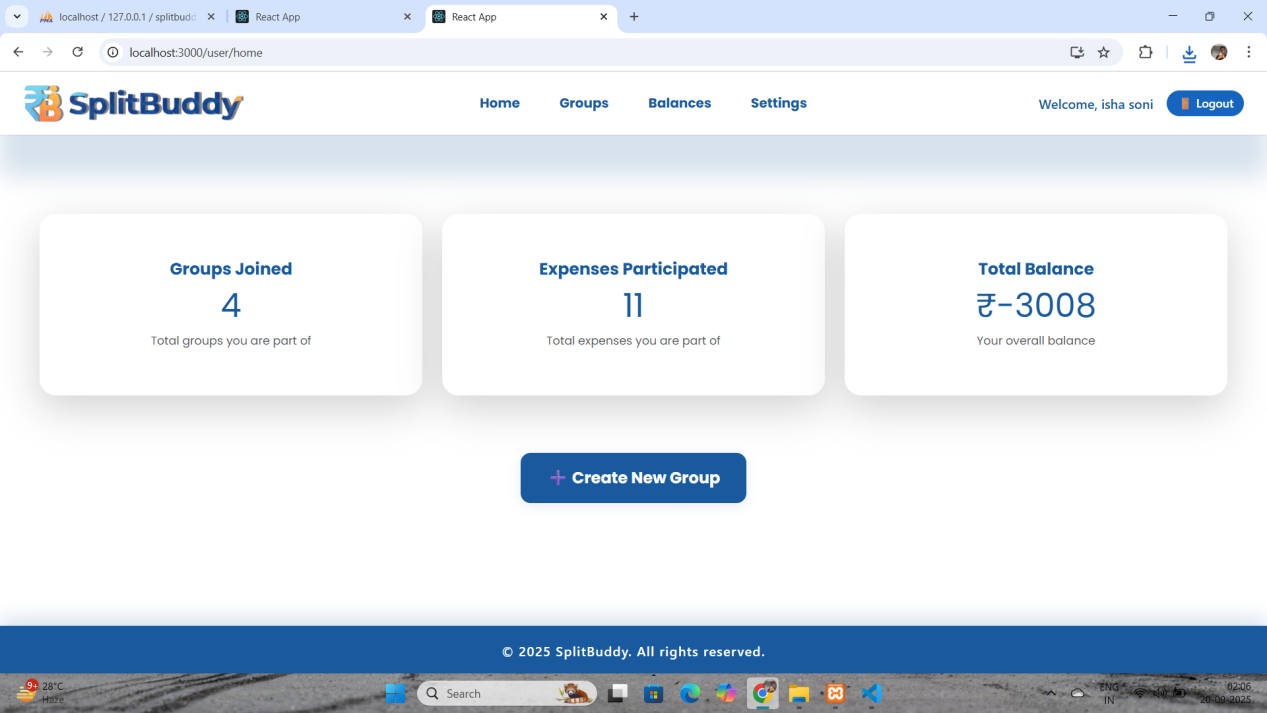


**Report :**

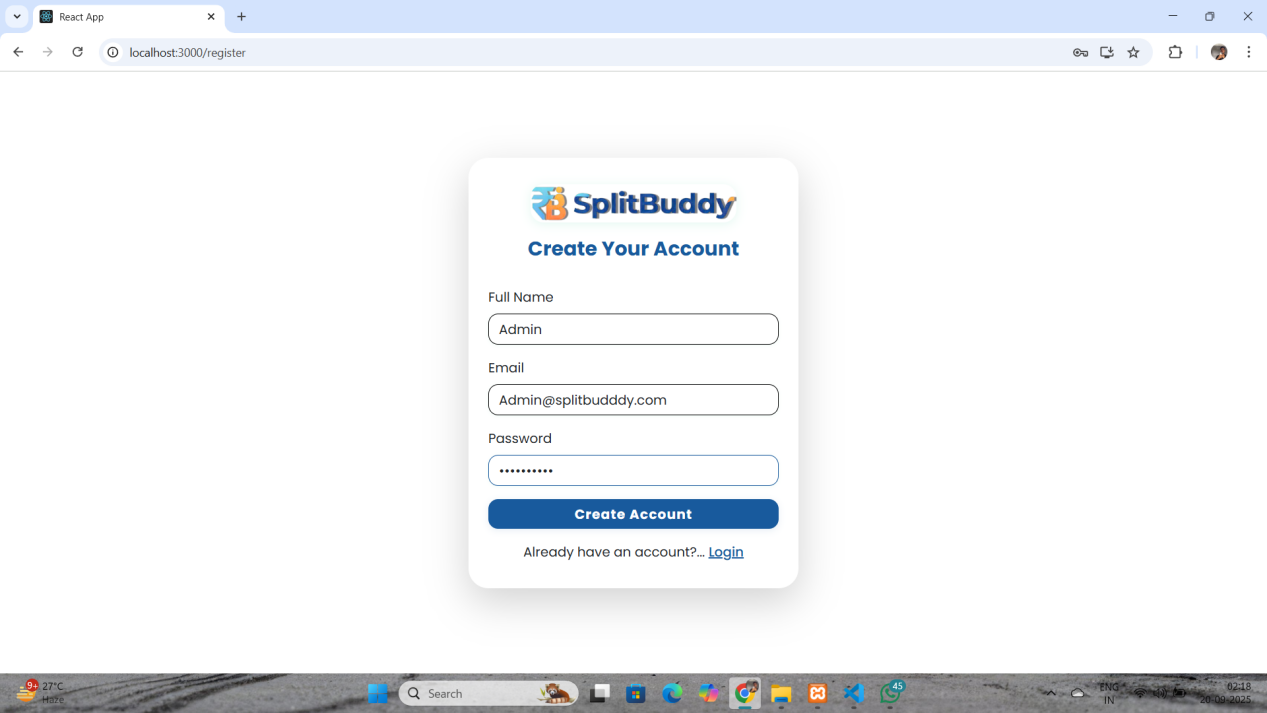
****

**Setting :**

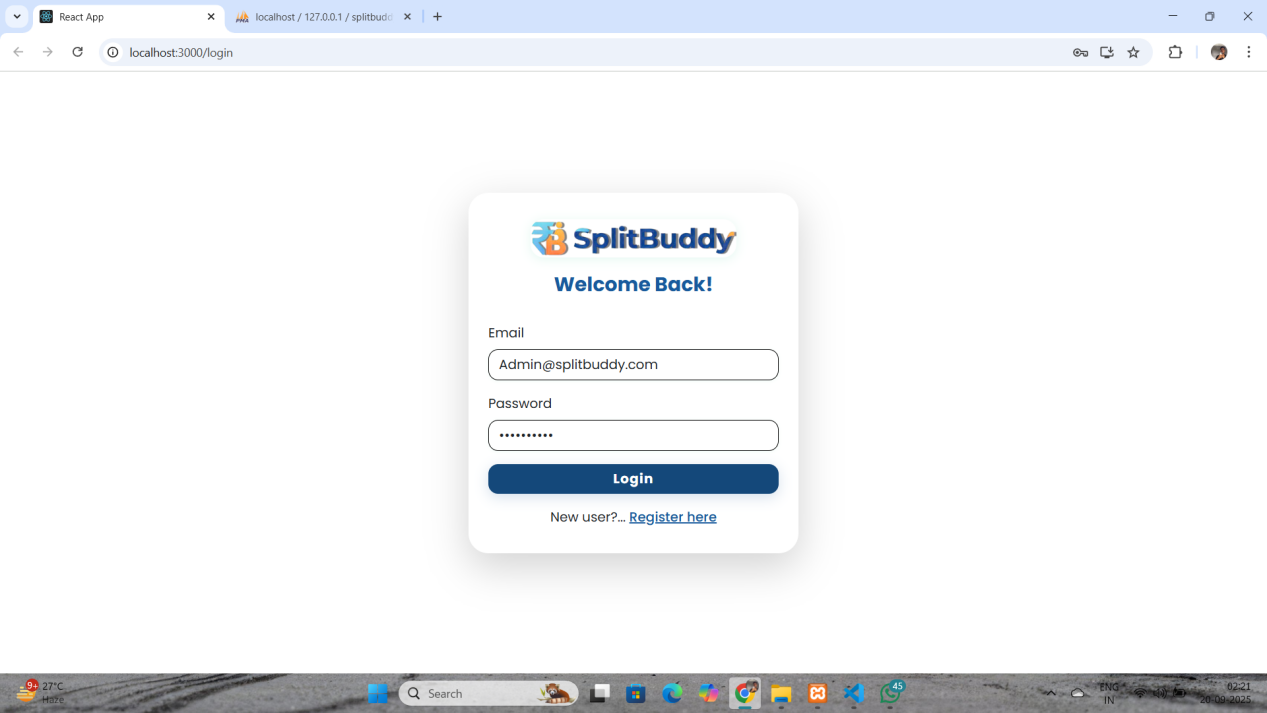
****

****

Admin Side : Registration



Login :

****

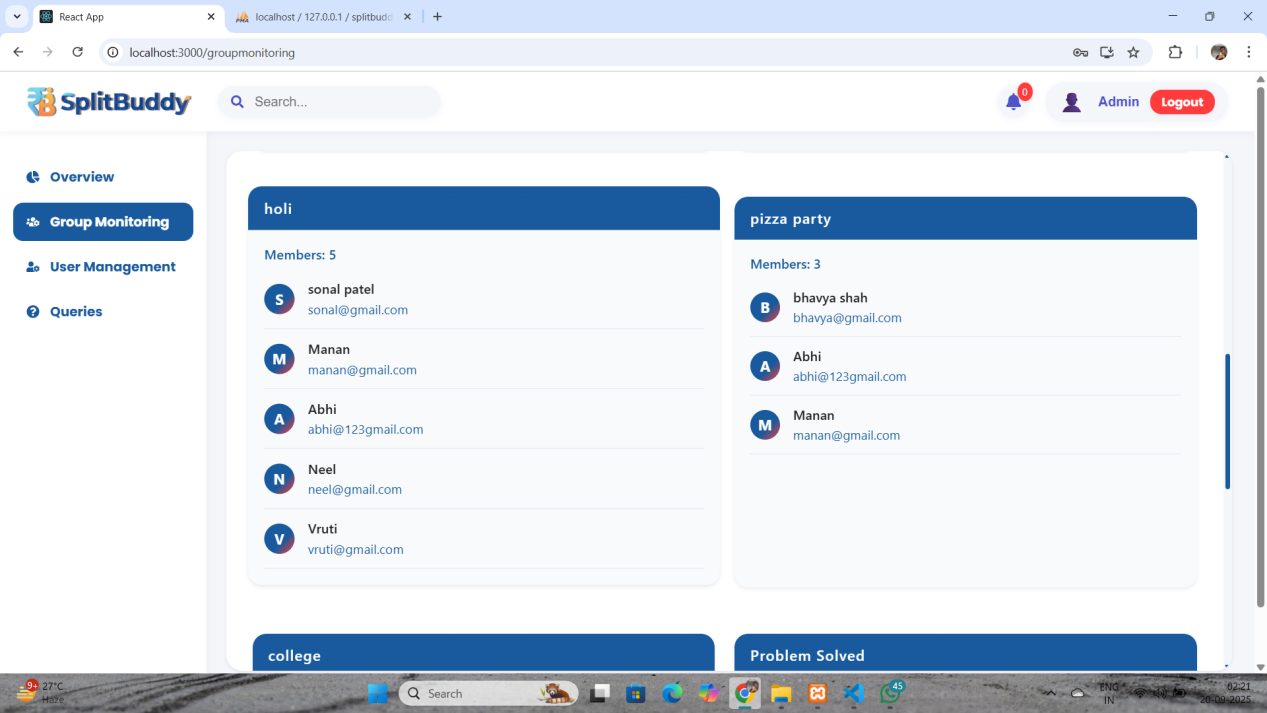
Dashboard :

****

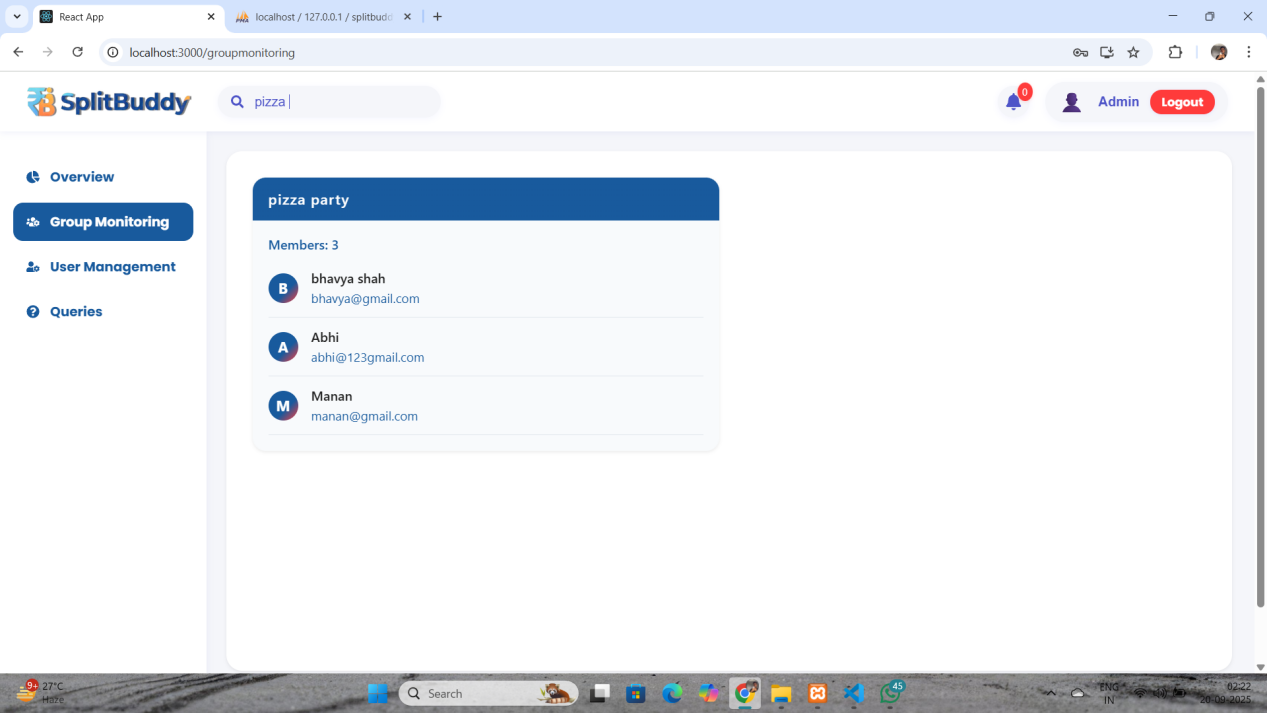
Overview Chart :

****

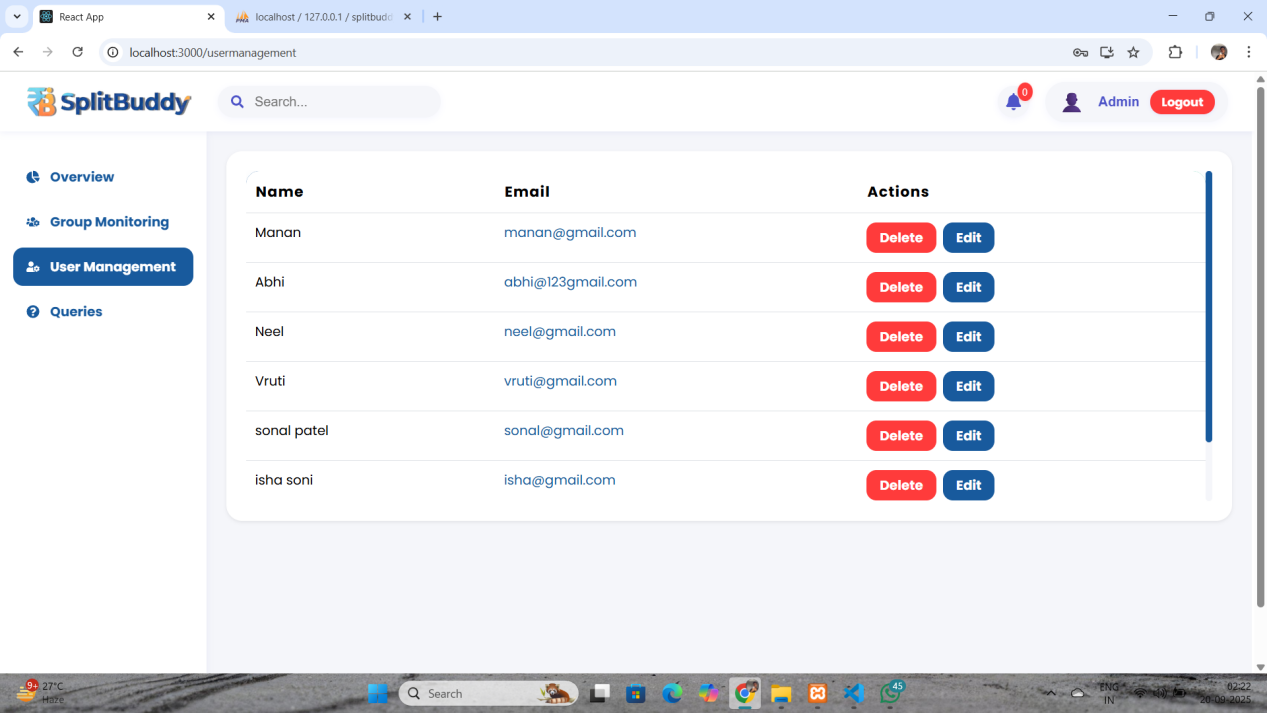
Group Monitoring:

****

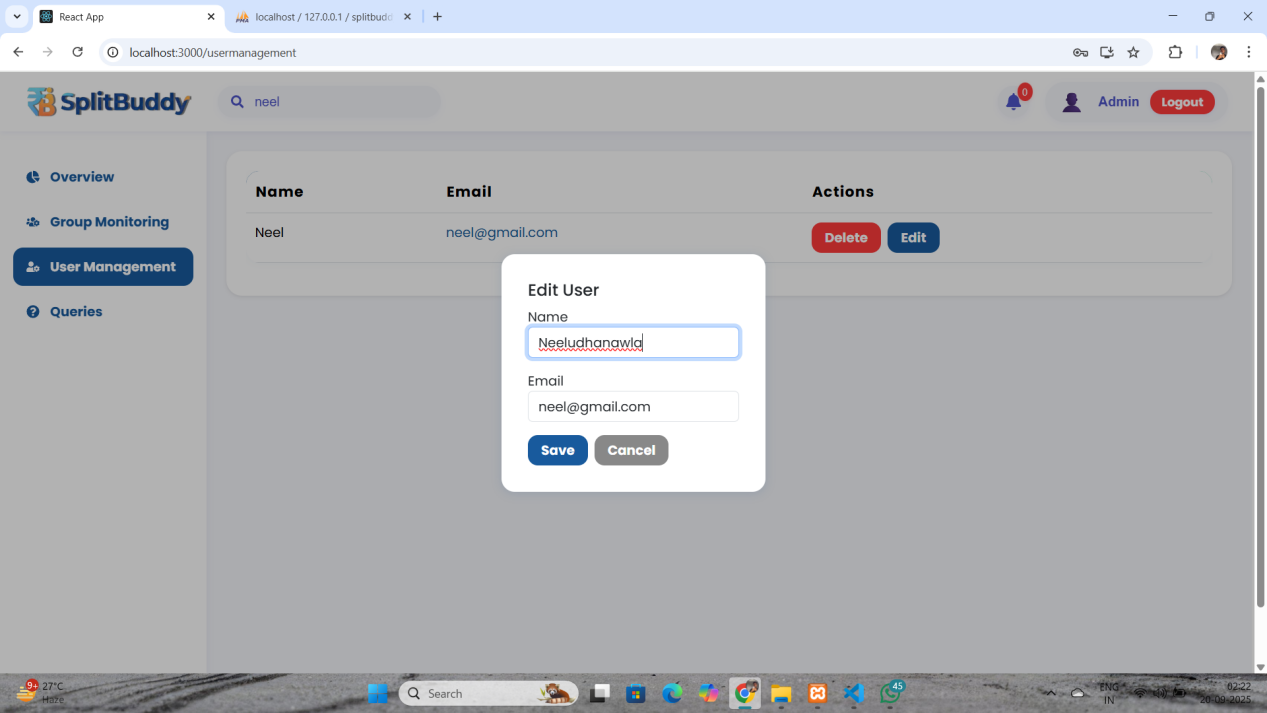
Searching feature : (Serch Group)

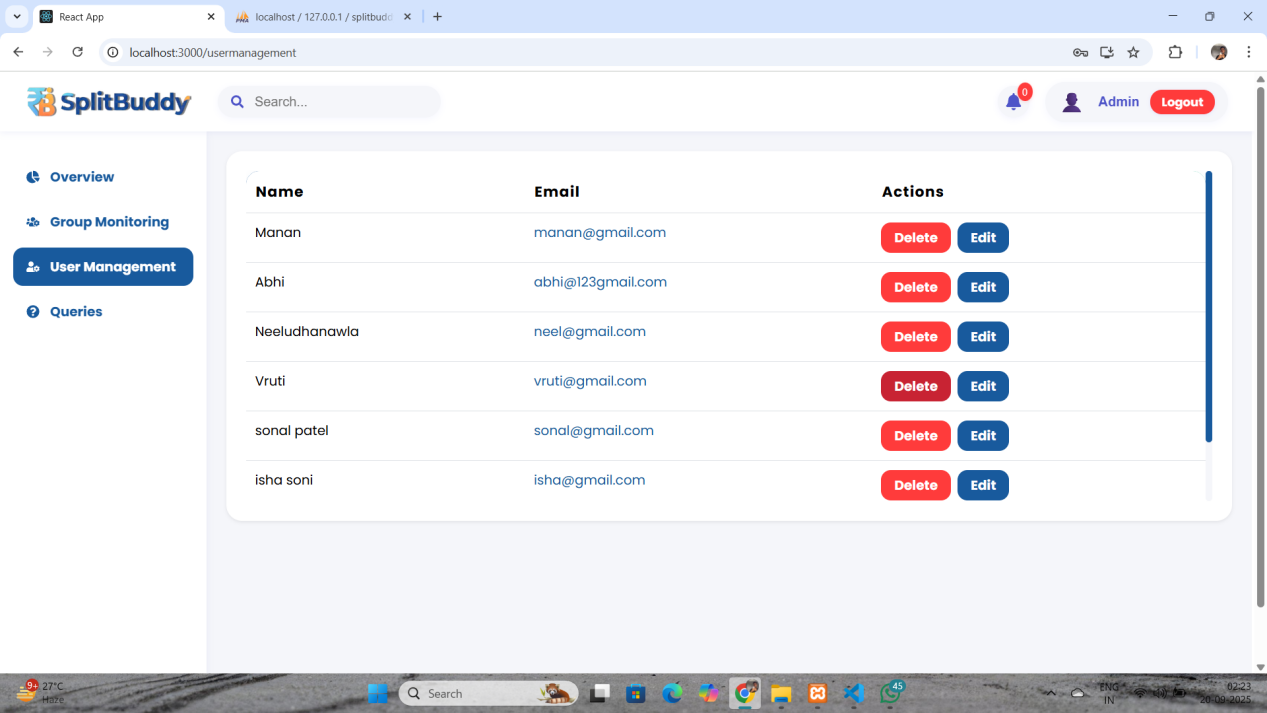
****

User Management :

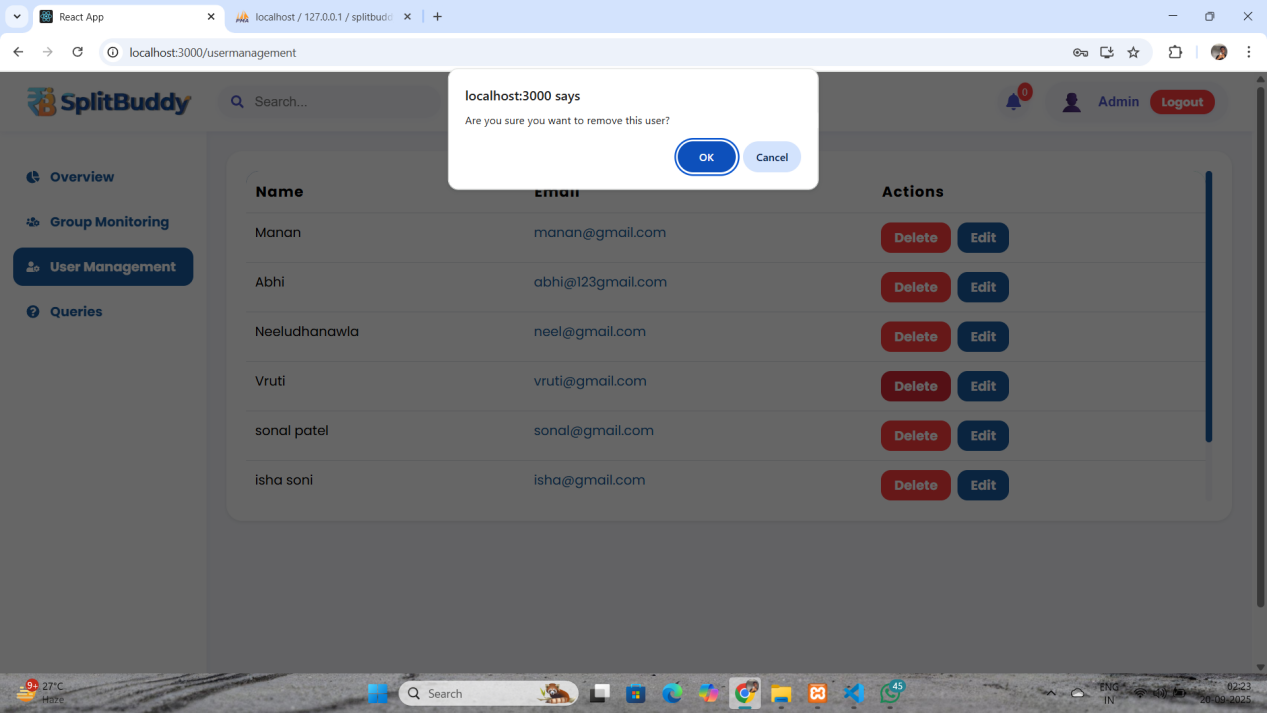
****

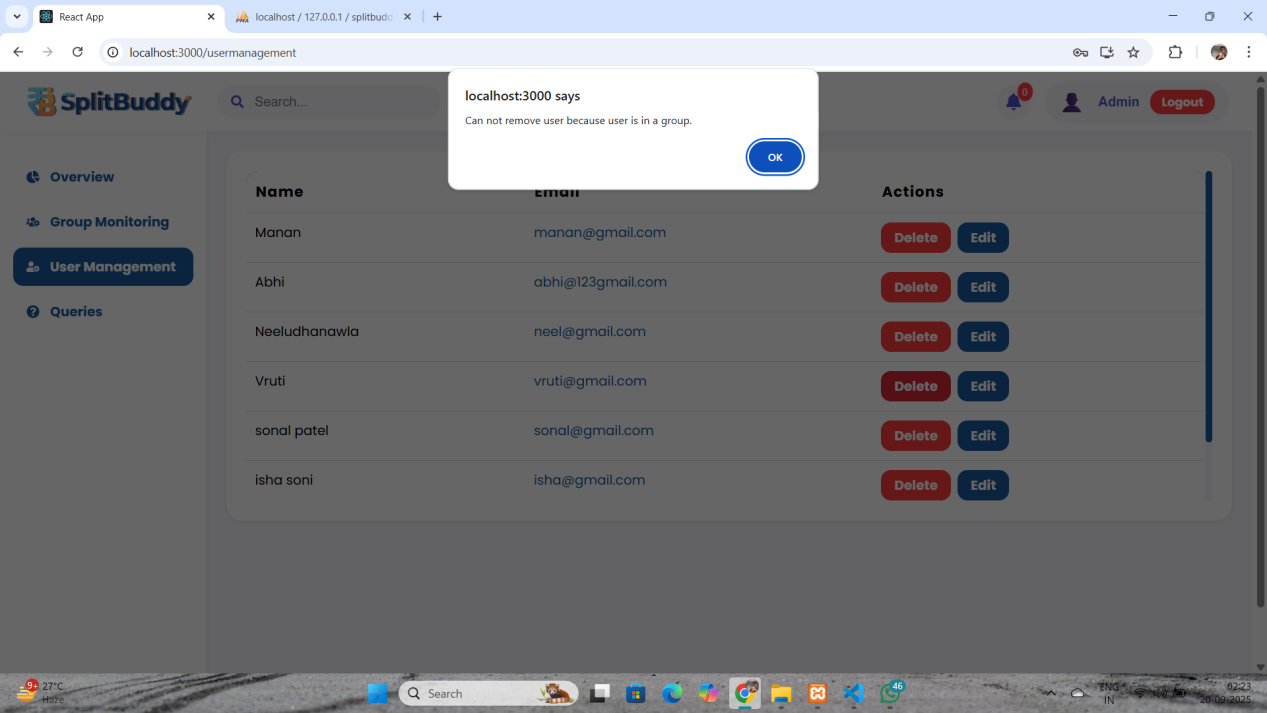
Edit User :

****

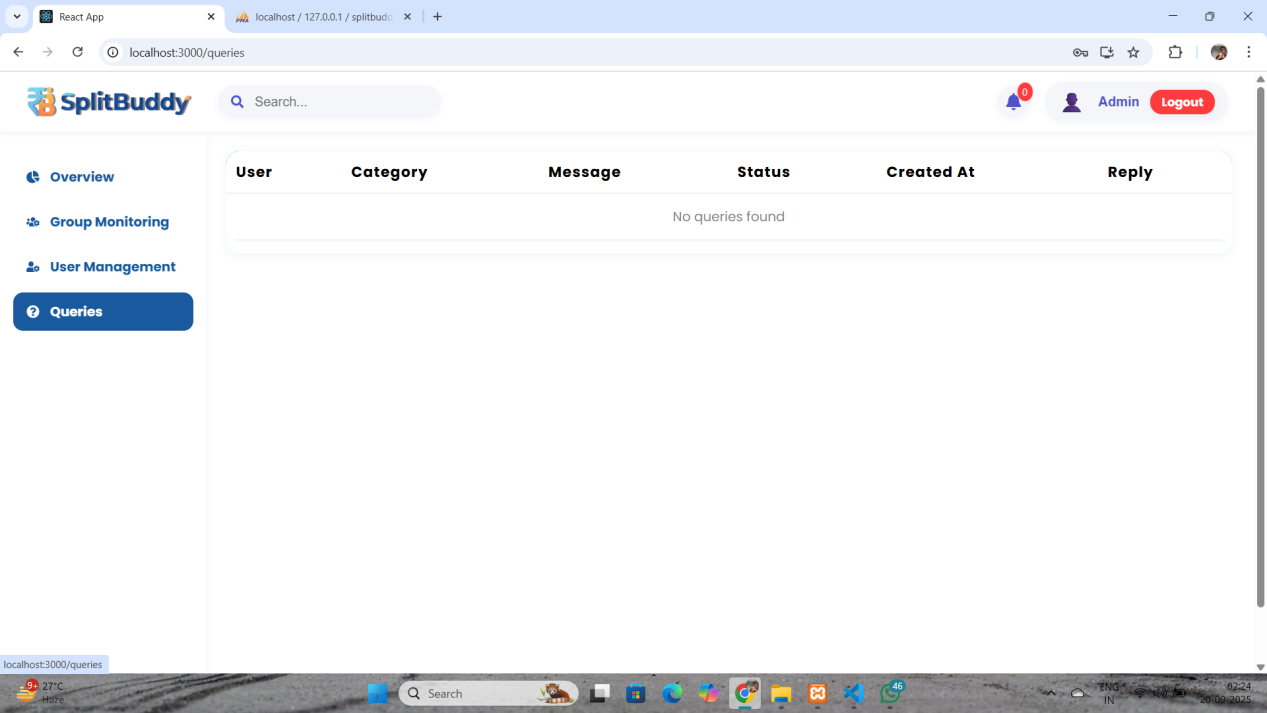
****

Delete User :

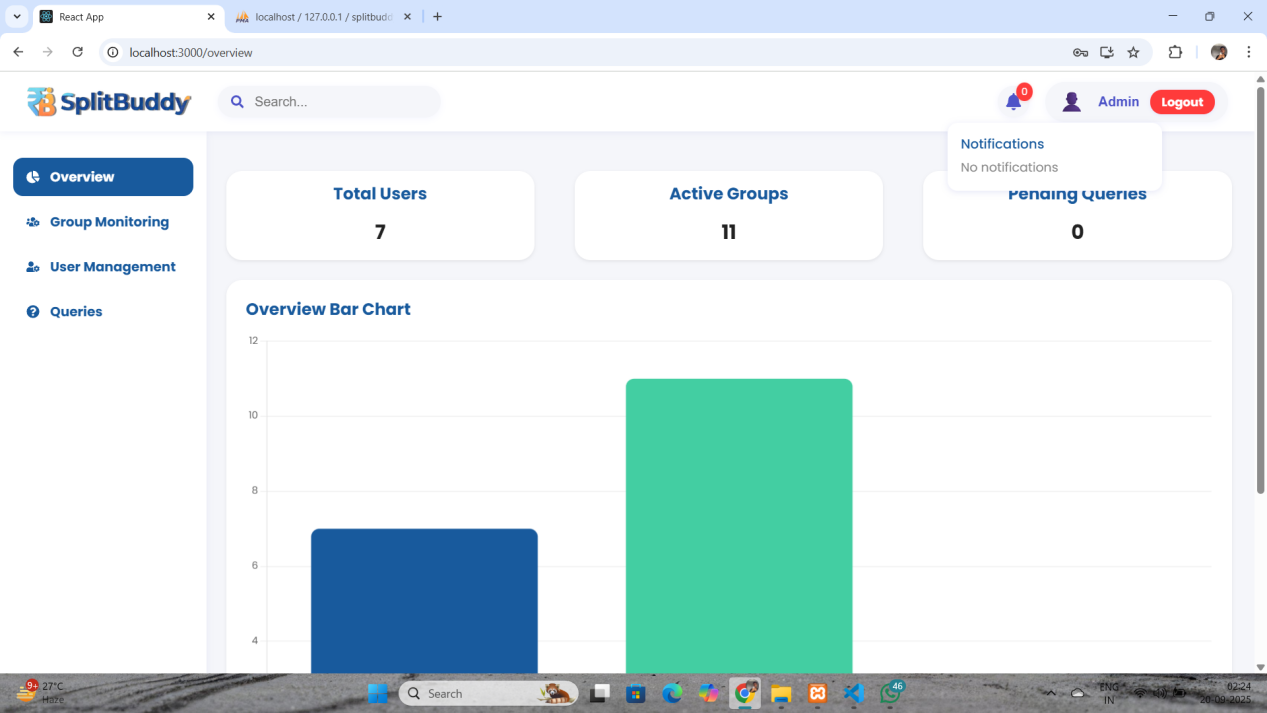
****

****

Resolve Queries :

****

Notification :

****

**5.3 Reports**

* Group-wise expense summary
* Individual user balance report
* Settlement history
* Query/Support report for admin

**6 System Implementation**

**6.1 Module Implementation**

* **User Module**: Authentication (JWT), profile settings
* **Group Module**: CRUD for groups, member invitations
* **Expense Module**: Add/view expenses, split logic
* **Settlement Module**: Track and mark settlements as completed
* **Notifications**: Real-time messaging and alerts
* **Admin Module**: Monitoring groups and User management

- Manage queries and bug reports

**7 System Testing**

**7.1 Testing Strategies**

* **Unit Testing**: Validate individual modules (login, expense split).
* **Integration Testing**: Ensure modules work together (expense + settlement).
* **System Testing**: Verify overall functionality.
* **User Acceptance Testing**: Feedback from sample users.

**7.2 Test Cases**

| **Test Case ID** | **Description** | **Input** | **Expected Output** | **Result** |
| --- | --- | --- | --- | --- |
| TC01 | User login with valid credentials | Correct email & password | Redirect to dashboard | Pass |
| TC02 | User login with **invalid** credentials | Wrong password | Error message: “Invalid username or password” | Pass |
| TC03 | Add expense (normal) | Amount = 500, payer = User A, members = A,B,C | Expense saved and split calculated | Pass |
| TC04 | Add expense with empty fields | Blank amount field | Validation error shown | Pass |
| TC05 | Generate Settlement | Button Click | Show generated settlement between members | Pass |
| TC06 | Settlement update | Pay amount = 200 | Status updated to “Completed” | Pass |

**8 Future Enhancement**

* + Mobile app (Android/iOS)
  + Payment gateway integration (UPI, PayPal)
  + Offline mode with data sync
  + AI-powered expense categorization
  + Multi-language support

**9 Bibliography/References**

React Documentation – [https://react.dev](https://react.dev/)

Node.js Docs – [https://nodejs.org](https://nodejs.org/)

MySQL Reference Manual – <https://dev.mysql.com/doc>

Bootstrap Docs – [https://getbootstrap.com](https://getbootstrap.com/)