EXPENSE SPLITTER

In an era where financial management is essential for both individuals and groups, the Expense Splitter project offers a comprehensive solution for tracking and managing expenses. This user-friendly application integrates expense tracking, data visualization, and database management, enabling users to streamline their financial activities.

Project report by: Manya Sharma

In Guidance (Instructor): Jyoti Patil

Submission Date: 20 March 2025

Github-Repo:
https://github.com/manya271001/ExpenseSplitter

Table of Content

Problem Definition and Objective

Problem Definition

Managing shared expenses efficiently is a common challenge for individuals and groups such as friends, families, and colleagues. Traditional methods, like manual calculations or spreadsheets, can be cumbersome, prone to errors, and lack real-time tracking. Without a structured system, users may struggle to track who owes whom, leading to financial mismanagement and potential disputes. Additionally, a lack of data visualization tools makes it difficult to analyse spending patterns and make informed financial decisions.

Objective

The Expense Splitter project aims to provide a seamless and efficient solution for tracking, managing, and splitting expenses. It simplifies financial management by integrating expense tracking, data visualization, and database management into a user-friendly application. Key goals include:

- Allowing users to input, categorize, and track expenses effortlessly.
- Providing clear and insightful data visualization to help users analyse their spending habits.
- Ensuring secure and structured data storage using an SQL database.
- Enhancing user experience with an intuitive UI .

- Supporting fair and transparent cost-sharing for groups, minimizing disputes.

Frontend & Backend Architecture

Overview of Chosen Technology Stack

The Expense Splitter application is developed using a modern technology stack to ensure efficiency, scalability, and a seamless user experience. The chosen technologies include:

- **Frontend:** React with Bootstrap for a responsive and visually appealing UI.
- **Backend:** ASP.NET Core Web API for handling business logic and API endpoints.
- **Database:** MS SQL Server for secure and structured data storage.

System Design Diagram

Below is a high-level system design diagram illustrating how different components interact

```
+--------------------+      +-------------------------+      +----------------------+

|   React Frontend | <--> | ASP.NET Core Web API | <--> | MS SQL Database |

+--------------------+      +-------------------------+      +----------------------+
```

• React Frontend

- Provides an interactive UI for managing expenses.
- Communicates with the backend via API calls.
- Uses Bootstrap for styling and responsiveness.

• ASP.NET Core Web API Backend

- Handles API requests and processes business logic.
- Exposes endpoints for expense management, user authentication, and settlements.
- Implements security measures like authentication and authorization.

• MS SQL Server Database

- Stores users, expenses, and settlement details.
- Ensures data consistency and integrity.
- Provides efficient querying and retrieval of financial records.

Component Breakdown & API Design

Key Components in the Expense Splitter Application

1. **App.js**
   - Defines all the routes and navigation structure using `react-router-dom`.
   - Separates pages with and without the layout (Navbar & Footer).
2. **Layout**
   - A wrapper component that includes a Navbar, Footer, and Outlet for nested routes.
3. **Registration & Login**
   - Handles user authentication and account creation.
   - Uses forms for user input and authentication logic.
4. **HeroSection**
   - The landing page of the application, providing an overview of the Expense Splitter.
5. **AboutSection & ContactPage**
   - About page explains the project, and ContactPage provides a way to reach support.
6. **Dashboard**
   - Central hub where users can see an overview of their expenses, groups, and settlements.
7. **Groups & AddGroup**
   - `Groups`: Displays all user groups.
   - `AddGroup`: Allows users to create a new group for expense sharing.
8. **InviteModal & Invitations**
   - `InviteModal`: Handles sending invitations to users for joining groups.
   - `Invitations`: Displays pending invitations users have received.
9. **ExpenseTracker**
   - Allows users to add and track expenses within their groups.
10. **ManageExpense**

- Displays detailed expense breakdown per group.

- Allows users to see who paid, how much to split, and who owes whom.

API Endpoints and Their Usage

User API (`/api/users`)

1. **`GET /api/users`**
   - Fetches all users from the database.
2. **`GET /api/users/{id}`**
   - Retrieves details of a specific user by their ID, including name, email, initial balance, and setup status.

☞ **Used for:** Fetching user information and managing user profiles.

Invitation API (`/api/invitations`)

1. **`POST /api/invitations/send`**
   - Sends an invitation to a user for joining a group after validation.
2. **`GET /api/invitations/pending/{userId}`**
   - Fetches all pending invitations for a user, including group and inviter details.
3. **`POST /api/invitations/respond`**
   - Allows users to accept or reject an invitation and updates the status accordingly.

☞ **Used for:** Handling group invitations and membership requests.

Group API (`/api/group`)

1. **`POST /api/group/createGroup`**
   - Creates a new group with specified members and stores group details in the database.
2. **`GET /api/group/userGroups/{userId}`**
   - Retrieves the number of groups a user is part of.
3. **`GET /api/group/userGroups/details/{userId}`**
   - Fetches detailed information about groups a user is part of.
4. **`GET /api/group/createdBy/{userId}`**
   - Retrieves all groups created by a specific user.
5. **`DELETE /api/group/leaveGroup/{groupId}/{userId}`**
   - Allows a user (except the creator) to leave a group.
6. **`DELETE /api/group/deleteGroup/{groupId}/{userId}`**
   - Deletes a group but only if the request is made by the creator.
7. **`GET /api/group/members/{groupId}`**
   - Fetches details of all members in a specified group.
8. **`GET /api/group/userGroups/memberCounts/{userId}`**

- Retrieves the number of members in all groups a user is part of.

9. **`GET /api/group/memberCount/{groupId}`**
    - Fetches the total number of members in a particular group.

☞ **Used for:** Managing group creation, deletion, membership, and fetching details of groups and members.

Authentication API (/api/auth)

1. **`POST /api/auth/register`**
    - Registers a new user, hashes the password, and stores user details in the database.
    - Generates a JWT token upon successful registration.
2. **`POST /api/auth/login`**
    - Authenticates a user by verifying email and password.
    - Returns a JWT token if credentials are valid.
3. **`GET /api/auth/profile`** *(Protected Route)*
    - Retrieves the profile details of the authenticated user.
4. **`GET /api/auth/user/{id}`**
    - Fetches a user's details by their ID.

☞ **Used for:** User authentication, registration, and profile management.

Expense API (`/api/expenses`)

1. **`POST /api/expenses/addExpense`**
    - Adds a new expense to a group after verifying the user's membership.
2. **`GET /api/expenses/group/{groupId}`**
    - Retrieves all expenses for a specific group.
3. **`GET /api/expenses/user/{userId}`**
    - Fetches all expenses made by a specific user.
4. **`GET /api/expenses/group/{groupId}/totalExpense`**
    - Calculates the total expense in a group and distributes the amount equally among members.
    - Determines who owes whom and generates a payment flow.
5. **`GET /api/expenses/user/{userId}/balances`**
    - Computes the total balance of a user across all groups.
    - Displays how much they owe or need to receive.

☞ **Used for:** Managing, tracking, and splitting expenses within groups.

Database Structure & Relationships

1. Overview

The Expense Splitter application uses **MS SQL Server** as the database, managed via **Entity Framework Core**. The database follows a **relational structure**, ensuring efficient data storage and retrieval.

2. Tables and Relationships

Table Name	Description
`New Users`	Stores user information (authentication, balance, groups).
`Groups`	Stores group details (name, creator, max members).
`UserGroups`	A junction table linking users and groups (many-to-many relationship).
`Expenses`	Tracks expenses in groups (who paid, how much, for which group).
`Invitations`	Stores group invitations (sent to users, pending/accepted/rejecte

3. Detailed Table Structure & Relationships

NewUsers Table (Users Table)

- **Primary Key:** `Id` (Unique user identifier).
- **Columns:**
  - `Id` (int, PK, Auto-increment)
  - `Name` (nvarchar)
  - `Email` (nvarchar, Unique)
  - `PasswordHash` (nvarchar)
  - `InitialBalance` (decimal)
  - `NumberOfGroups` (int)
  - `HasSetup` (bool)
- **Relationships:**
  - One-to-Many with `UserGroups` (A user can be in multiple groups).
  - One-to-Many with `Expenses` (A user can pay for multiple expenses).

🔹 *Groups Table (Group Details)*

- **Primary Key:** `Id` (Unique group identifier).
- **Columns:**
  - `Id` (int, PK, Auto-increment)
  - `Name` (nvarchar)
  - `CreatedBy` (int, FK → NewUsers.Id)
  - `MaxMembers` (int)
  - `TotalBalance` (decimal)
  - `IsActive` (bool)
  - `Description` (nvarchar)
- **Relationships:**
  - One-to-Many with `UserGroups` (A group has multiple users).
  - One-to-Many with `Expenses` (A group has multiple expenses).

🔹 *UserGroups Table (Many-to-Many Relationship)*

- **Primary Key:** Composite Key (`UserId`, `GroupId`).
- **Columns:**
  - `UserId` (int, FK → NewUsers.Id)
  - `GroupId` (int, FK → Groups.Id)
- **Purpose:**
  - This table **links users to groups** (a user can be in multiple groups, and a group can have multiple users).

🔹 *Expenses Table (Expense Tracking)*

- **Primary Key:** `Id` (Unique expense identifier).
- **Columns:**
  - `Id` (int, PK, Auto-increment)
  - `GroupId` (int, FK → Groups.Id)
  - `PaidBy` (int, FK → NewUsers.Id)
  - `Amount` (decimal)
  - `Category` (nvarchar)
  - `CreatedAt` (datetime)
- **Relationships:**
  - Many-to-One with `Groups` (Each expense belongs to a group).
  - Many-to-One with `NewUsers` (Each expense is paid by a user).

🔹 *Invitations Table (Group Invitations)*

- **Primary Key:** `Id` (Unique invitation identifier).
- **Columns:**
  - `Id` (int, PK, Auto-increment)
  - `GroupId` (int, FK → Groups.Id)
  - `InvitedUserId` (int, FK → NewUsers.Id)
  - `Status` (nvarchar, values: "Pending", "Accepted", "Rejected")

    - `CreatedAt` (datetime)
- **Relationships:**
    - Many-to-One with `Groups` (An invitation is linked to a group).
    - Many-to-One with `NewUsers` (Each invitation is sent to a user).

Entity-Relationship (ER) Diagram

```
+------------------+          +----------------+
|   NewUsers       |          |   Groups       |
+------------------+          +----------------+
| Id (PK)          | ◄────────► | CreatedBy (FK) |
| Name             |          | Name           |
| Email            |          | MaxMembers     |
| CreatedAt        |          | CreatedAt      |
| UpdatedAt        |          | UpdatedAt      |
+------------------+          +----------------+
        |                                |
        |                                |
+------------------+          +----------------+
|   UserGroups     | ◄────────► |   Expenses     |
+------------------+          +----------------+
| UserId (PK, FK)  |          | Id (PK)        |
| GroupId (PK, FK) |          | GroupId (FK)   |
| JoinedAt         |          | PaidBy (FK)    |
+------------------+          | Amount         |
                              | Description    |
                              | CreatedAt      |
                              +----------------+
                                        |
                                        |
                              +----------------+
                              |   Invitations  |
                              +----------------+
                              | Id (PK)        |
                              | GroupId (FK)   |
                              | InvitedUserId  |
                              | Status         |
                              | SentAt         |
                              | ExpiryDate     |
                              +----------------+
```

Flowchart representing the process flow of your **Expense Splitter application:**

```
Start
  |
  ▼
User Registers/Login
  |
  ▼
Home Screen
  ├──> Create Group
  |     |
  |     ▼
  |   Add Members
  |     |
  |     ▼
  |  Group Created ✓
  |
  ├──> Add Expense
  |     |
  |     ▼
  |   Select Group
  |     |
  |     ▼
```

```
│   Enter Amount, Payer, Split Details
│     │
│     ▼
│   Expense Added ✓
│
├──> Manage Expenses
│     │
│     ▼
│  View Detailed Breakdown
│     │
│     ├──> Who Paid?
│     │
│     ├──> Who Owes Whom?
│     │
│     ├──> Share Per Person
│     ▼
│  Settle Up
│     │
│     ▼
│  "Hurray! The settlement has been done" 🎉
│
▼
End
```

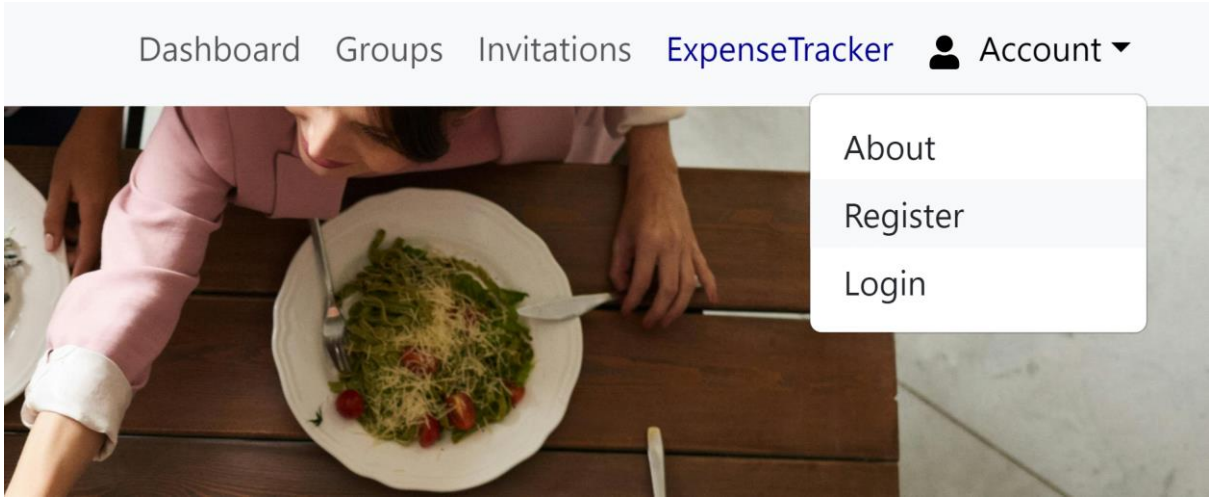
USER GUIDE

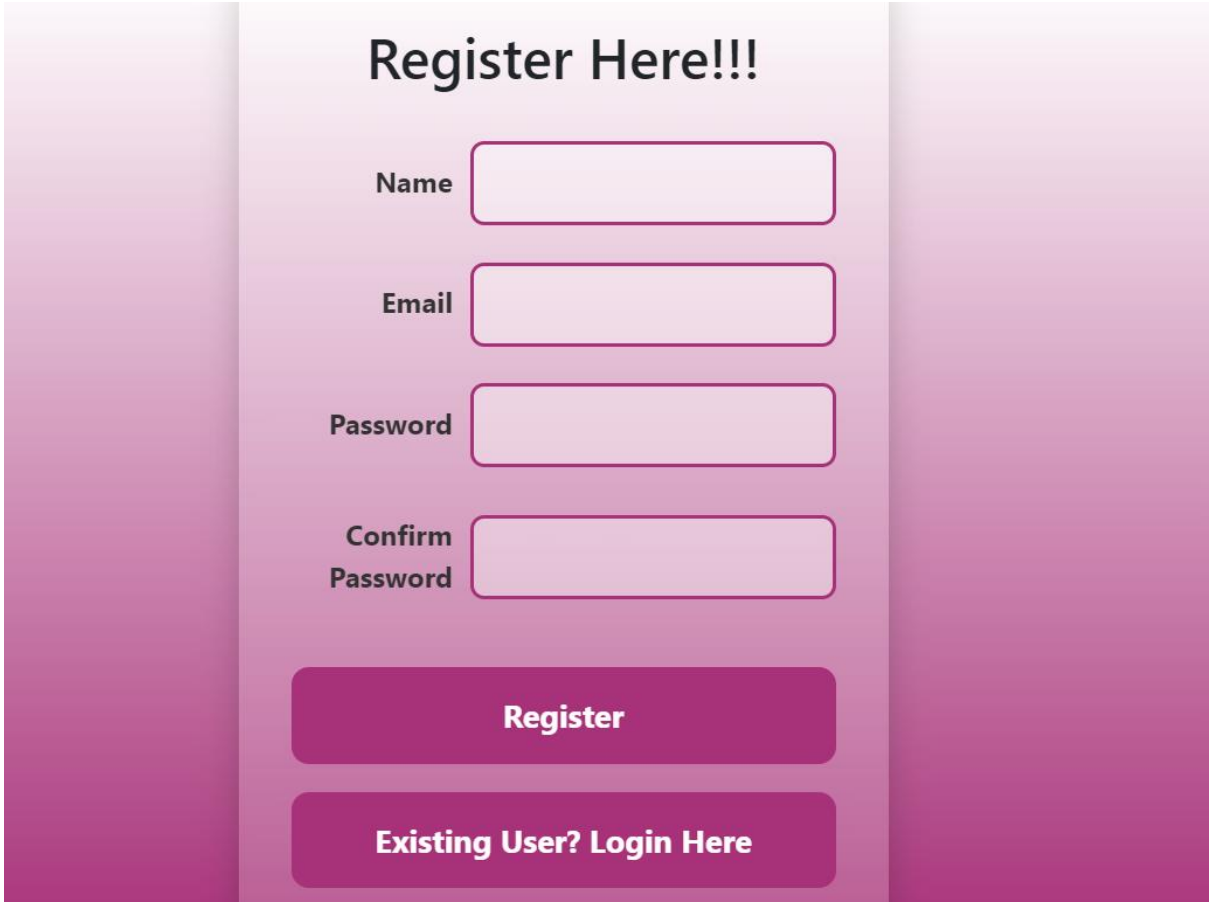
Step 1: User Registration

🔹 **Scenario:** A new user registers with their name and email.

Steps:

1. Open the application.
2. Click on **"Register"**.
3. Enter the following details:
   - Name
   - Email
   - Password
4. Click **"Submit"**.
5. If successful, the user is added to the **NewUsers** table.
6. If the email is already registered, show **"Email already exists. Please log in."**





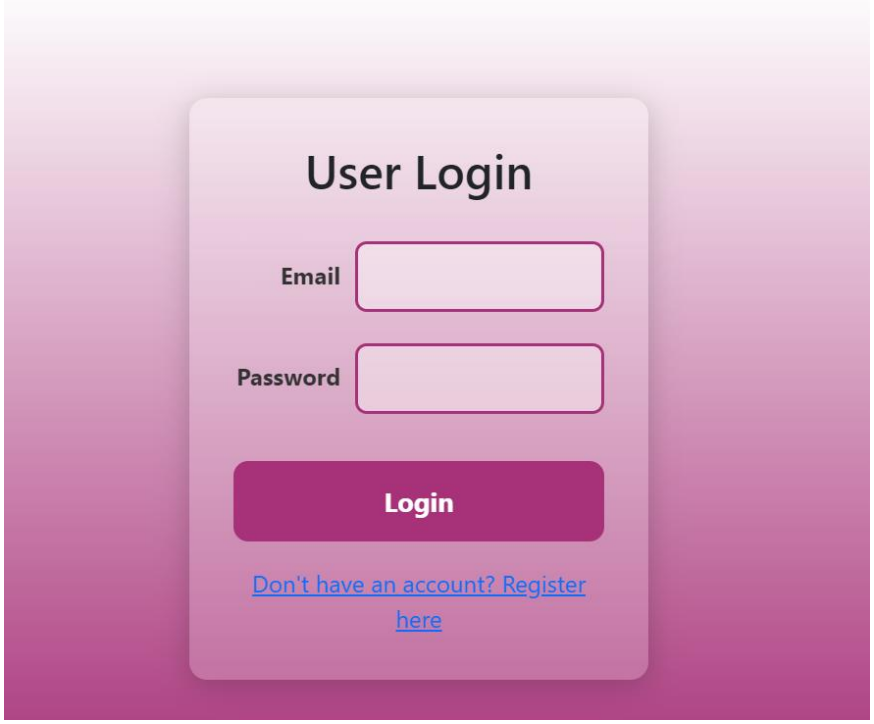
Step 2: User Login

🔹 **Scenario:** A registered user logs into their account.

Steps:

1. Click on **"Login"**.
2. Enter **Email** and **Password**.
3. Click **"Submit"**.
4. If credentials are correct, show **"Login successful!"** and redirect to the dashboard.

5. If incorrect, show **"Invalid email or password."**

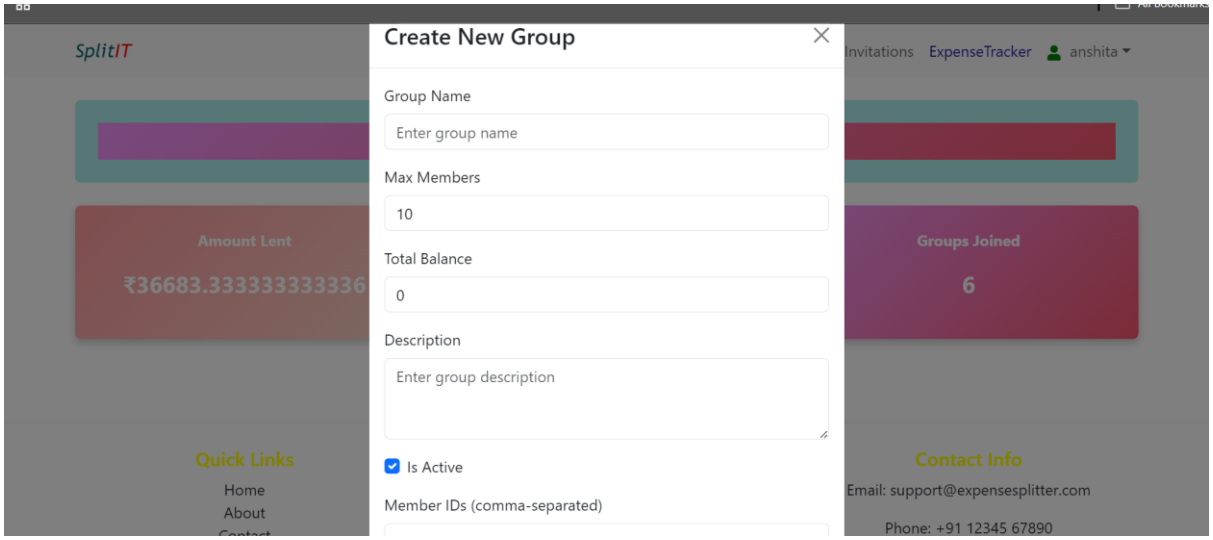


Step 3: Creating a Group

🔹 **Scenario:** A user creates a new group to manage shared expenses.

Steps:

1. Go to **"Dashboard"**.
2. Click on the **"Groups"** section.
3. Click **"Create New Group"**.
4. Enter:
   - **Group Name**
   - **Max Members**
5. Click **"Create"**.
6. The group is stored in the **Groups** table with **CreatedBy (FK)** linked to the user.
7. After clicking **"Create"**, you will be redirected back to the **Dashboard**, where the newly created group will be listed.

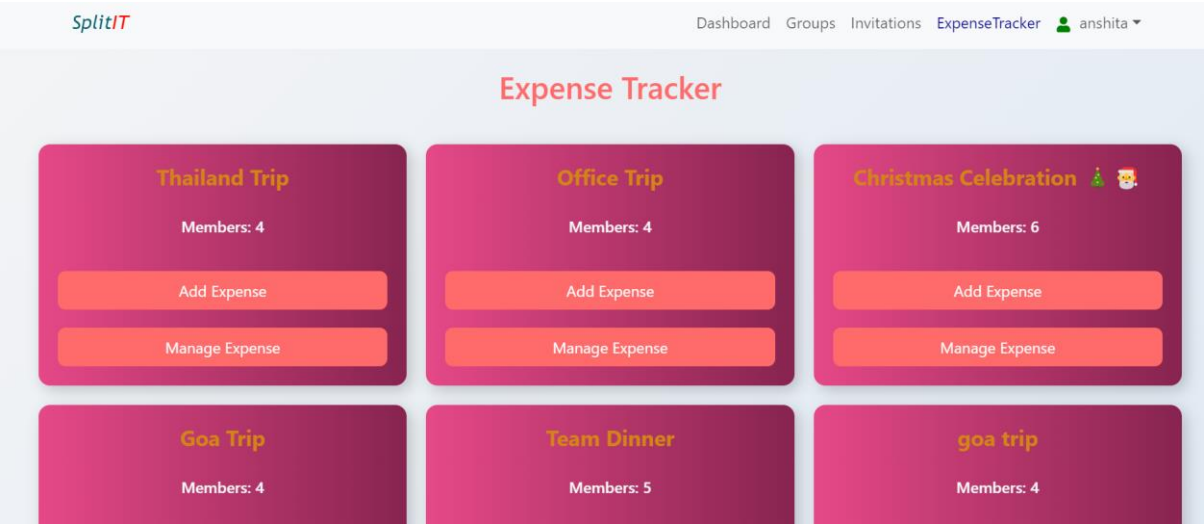


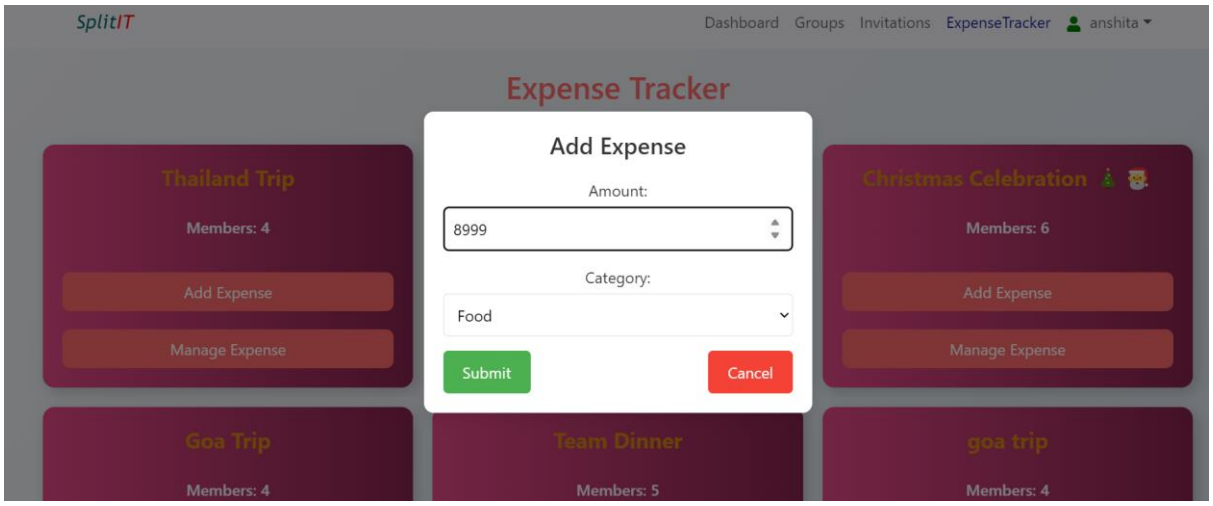
Step 5: Adding an Expense

🔹 **Scenario:** A user records an expense within a group.

Steps:

1. Go to **"Expense Tracker"** from the **Dashboard**.
2. You will see a list of all groups you are a part of.
3. Select the group where you want to add an expense.
4. Click **"Manage Expenses"** for that group.
5. Click **"Add Expense"**.
6. Enter the details:
   - **Amount**
   - **Paid By** (User who paid the amount)
   - **Select Group** (if not pre-selected)
7. Click **"Save"**.
8. The data is stored in the **Expenses** table with foreign keys linking to the **Group** and **PaidBy** user.
9. After saving, you can see and manage all recorded expenses within the selected group.





Step 5: Accepting or Declining an Invitation

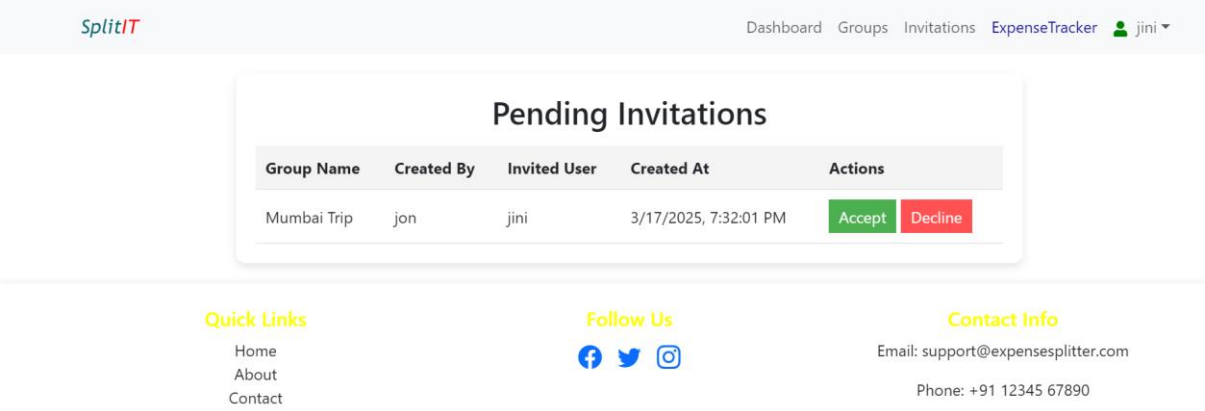
🔹 **Scenario:** A user receives an invitation to join a group and decides whether to accept or decline it.

Steps:

1. Go to the **Dashboard**.
2. Click on **"Invitations"** in the navigation bar.
3. You will see a list of **Pending Invitations**.
4. Each invitation displays:
    - **Group Name**
    - **Created By** (Who sent the invitation)

- **Invited User** (Your username)
- **Created At** (Time of invitation)
- **Actions**:
  - ✅ **Accept**: Click to join the group.
  - ❌ **Decline**: Click to reject the invitation.

5. After accepting, you will be added to the **UserGroups** table, linking you to the group.
6. You can now see and manage expenses within the group from the **Expense Tracker** section.

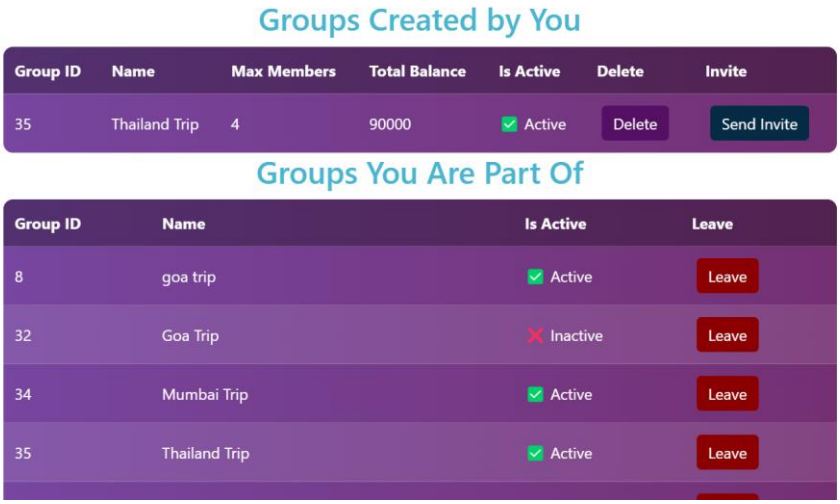


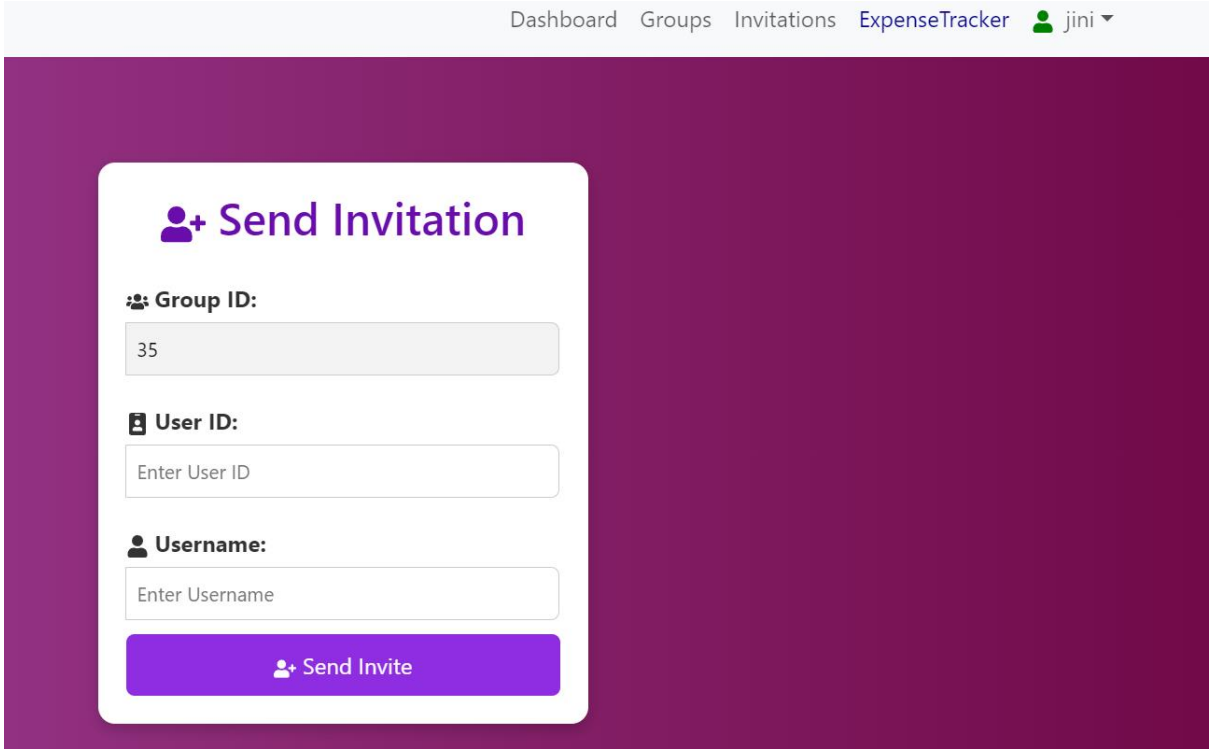
Step 5: Tracking Your Groups

🔹 **Scenario:** A user wants to view and manage the groups they are part of.

Steps:

1. Click on **"Groups"** in the navigation bar.
2. You will see a list of all groups you are a part of.
3. Each group displays:
   - **Group Name**
   - **Created By**
   - **Max Members**
   - **Your Role** (Creator/Member)
   - **Actions**:
     - 📌 **View Details**: See group expenses and members.
     - ➕ **Add Members** (If you are the group creator).





Step 6: Checking Who Owes Whom

🔹 **Scenario:** A user wants to see pending balances between group members.

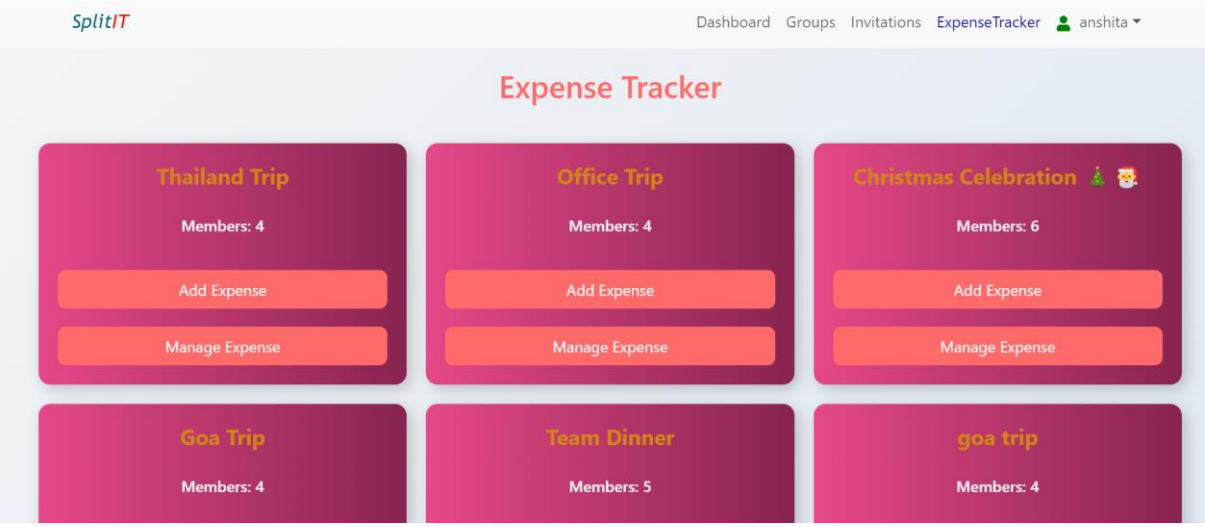
Steps:

1. **Go to the Dashboard.**
2. Click on **"ExpenseTracker"** in the navigation bar.

3. Select the group you want to check.
4. You will see a detailed expense table showing:
    - **Expense Name**
    - **Amount Paid**
    - **Paid By (User who paid)**
    - **Split Among (Users sharing the expense)**
    - **Each Person's Share**
    - **Pending Balances**

📌 **To check who owes whom:**

- The system automatically calculates and displays:
    - ✅ **Who has paid more**
    - ❌ **Who needs to pay back**
    - 🔄 **Pending settlements**



Click on manage expense to see

Paid By	Amount	Category	Date
jon	1000.000	Shopping	3/18/2025
	3000.000	Shopping	3/18/2025
	400.000	Other	3/18/2025
anshita	20000.000	Travel	3/18/2025
Hrishabh	10000.000	Travel	3/18/2025
	7000.000	Food	3/18/2025
	800.000	Rent	3/19/2025
	5600.000	Food	3/19/2025
	9000.000	Utilities	3/19/2025

Payment Flow Summary

User Name	Total Paid	Total to Receive	Total owned	Payment Flow
anshita	25933.333	25933.333	0.000	jon: ₹5066.667, jini: ₹9466.667, manya: ₹9466.667, neha: ₹1933.333
Hrishabh	7533.333	7533.333	0.000	neha: ₹7533.333
jon	0.000	0.000	5066.667	Pays No One
jini	0.000	0.000	9466.667	Pays No One
manya	0.000	0.000	9466.667	Pays No One
neha	0.000	0.000	9466.667	Pays No One

Step 7: Logging Out

🔹 **Scenario:** A user wants to securely log out from the application.

Steps:

1. Click on your **profile name (top-right corner).**
2. A dropdown will appear.
3. Click **"Logout"**.
4. You will be redirected to the **login page** for future access.

CONCLUSION

The **Expense Splitter** project effectively simplifies financial management for individuals and groups by providing **user authentication, group-based expense tracking, and detailed financial insights**. With a **React frontend**, an **ASP.NET Core Web API backend**, and **MS SQL for data storage**, it ensures a **seamless and efficient user experience**. The platform enables users to **track shared expenses, manage settlements, and visualize financial data**, promoting **transparency and ease of collaboration**. By streamlining expense tracking and simplifying settlements, the application makes managing group finances more **organized, fair, and accessible**.