|  |
| --- |
| **Expense Sharing App**  *“A simple app to split and track shared expenses among friends.”*  **Developed By: Gaurvi Paneri**  13 August 2025  BrainyBeam Placement Task-5 |

# **Table of Contents**

1. **Introduction** *Overview and purpose of the Expense Sharing App*
2. **Objective** *What app aims to achieve and why*
3. **Features** *Description of key functionalities*
4. **Technology Stack** *Tools and libraries used in the project*
5. **Application Architecture** *Explanation of frontend structure and data flow*
6. **Implementation Details** *Key code logic, components, and state management*
7. **Usage Instructions** *How to run and use the app locally*
8. **Challenges and Solutions** *Challenges that were faced along with their solutions*
9. **ScreenShots**

*Screenshots of the output*

1. **Conclusion** *Summary of the project and learnings.*
2. **References** *Resources and documentation referred to during development*

### **1. Introduction**

The Expense Sharing App is a simple yet effective web application designed to help users split and track shared expenses among a group. It provides an intuitive interface where members can be added, expenses recorded, and balances calculated automatically.

This implementation is **frontend-only** and uses **LocalStorage** to persist data in the browser. The app is ideal for friends, roommates, or colleagues who share expenses regularly and need a quick, offline-friendly solution.

### **2. Objectives**

The objectives of this project were:

* To develop a working expense tracker that splits costs equally among selected participants.
* To allow users to add and remove members dynamically.
* To display detailed expense history and real-time balances.
* To implement local data storage without the need for a backend server.

### **3. Features Implemented**

The key features of the Expense Sharing App include:

1. **Add / Remove Members** – Users can create and manage the group dynamically.
2. **Add Expenses** – Users can record expenses and split them equally among chosen participants.
3. **Expense History** – A log of all added expenses for reference.
4. **Balances View** – Displays who owes money and who is owed money.
5. **Clear All Data** – Option to reset all stored information instantly.
6. **Local Data Storage** – Data is stored in localStorage for persistence across sessions.

### **4. Tech Stack**

* **React with Vite** – Framework and build tool for fast frontend development.
* **JavaScript (ES6)** – Logic implementation for calculations and state management.
* **HTML & CSS** – Structure and styling for a user-friendly interface.
* **LocalStorage** – Browser storage to keep data persistent without a backend.

### **5. Application Architecture**

1. **Folder Structure**

**Expense\_Sharing\_App\_Code**

**├──src**

**│ ├── components**

**│ │ ├── MemberList.jsx**

**│ │ ├── ExpenseForm.jsx**

**│ │ ├── ExpenseList.jsx**

**│ │ └── BalanceSheet.jsx**

**│ ├── utils.jsx**

**│ ├── App.jsx**

**└── main.jsx**

**├──public**

**├──index.html**

**└──package.json**

1. **Main Components & Responsibilities**

* **App.jsx:** Root component managing state (members, expenses) and coordinating subcomponents.
* **MemberList:** Handles adding/removing group members.
* **ExpenseForm:** Lets users add new expenses.
* **ExpenseList:** Displays all added expenses.
* **BalanceSheet:** Calculates and shows who owes or is owed.

1. **Data Flow**

* State is held in App.jsx using React useState.
* Data persistence via localStorage.
* Child components receive data and callbacks via props.

1. **How components interact**

* User inputs in **MemberList** and **ExpenseForm** update the App state.
* Changes cause re-render and update the balances and expense lists.

### **6. Implementation Details**

* The application is structured into modular React components for members, expenses, and balances.
* When a new expense is added, the total amount is divided equally among the selected participants.
* Balances are recalculated every time an expense is added or removed.
* Data is stored in localStorage as JSON and retrieved when the app loads.
* Styling is applied for a clean layout and better user experience.

## **7. Usage Instructions**

### **Running the App Locally**

**Prerequisites:**

* Make sure you have **Node.js (v16 or higher)** installed on your system.
* A modern web browser (Chrome, Firefox, Edge, etc.).

1. **Installation:**

* Clone the project repository to your local machine:  
   **git clone React\_Task\_5\_GaurviPaneri**
* Navigate to the project folder:  
  **cd React\_Task\_5\_GaurviPaneri/code/Expense\_Sharing\_App\_Code**
* Install dependencies:  
  **npm install**

1. **Start the Development Server:**

* Run the app locally using Vite:  
  **npm run dev**
* Open the URL printed in the terminal (usually http://localhost:5173) in your browser.

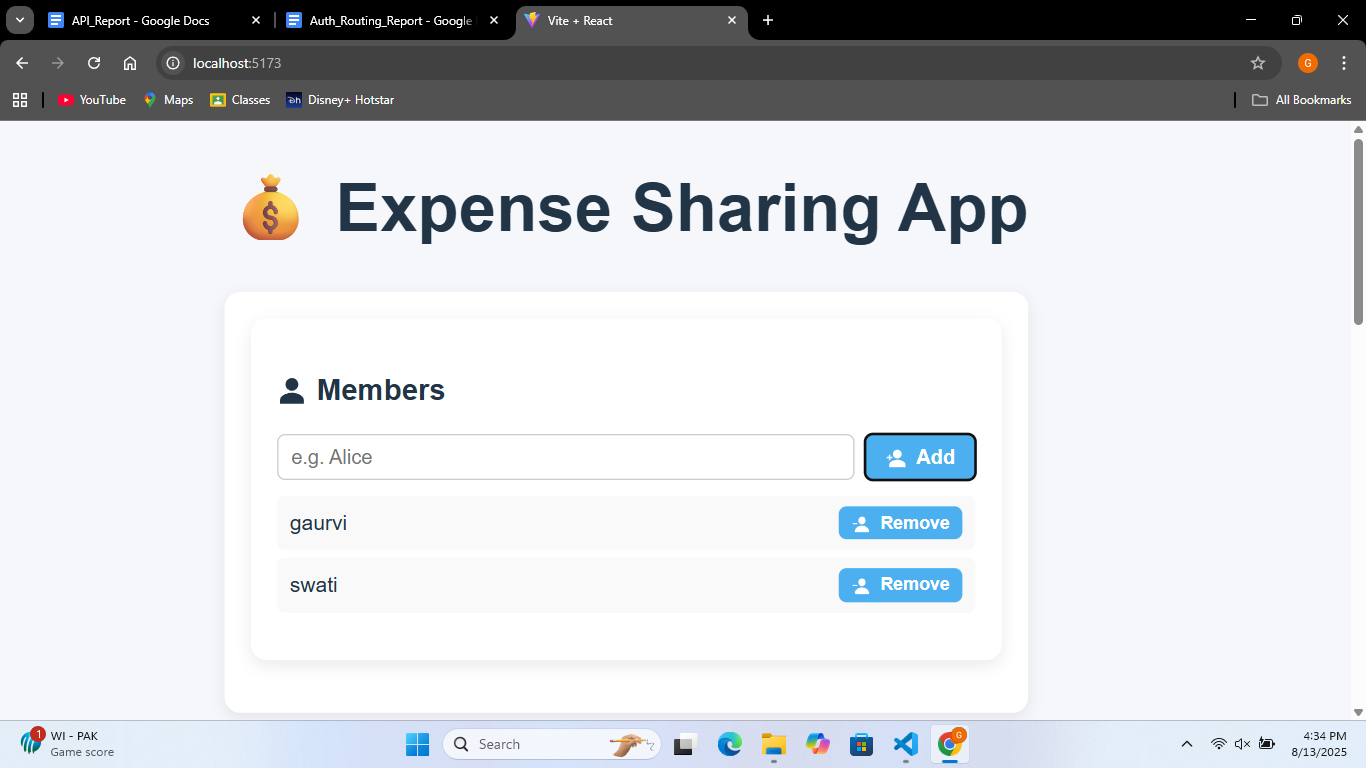
1. **Using the App:**

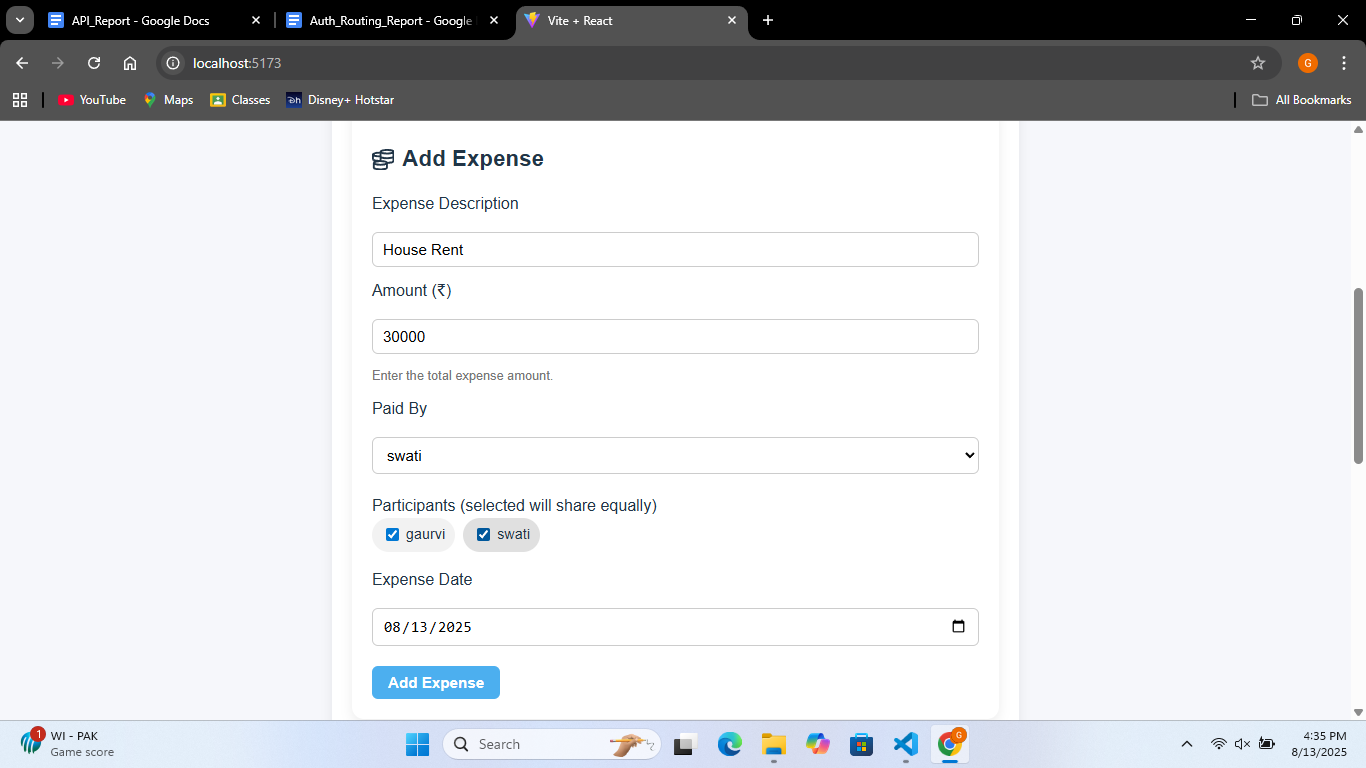
* **Add Members:** Enter member names in the “Add Member” section and click the add button.
* **Add Expenses:** Fill in the expense details—description, amount, who paid, and participants involved—and submit.
* **View Balances:** See the summary of who owes or is owed money in the balances section.
* **Manage Data:** Remove members or clear all data using the buttons provided.  
  All data is stored **locally in your browser**, so it persists even if you refresh or close the tab.
* **Clearing Data:**Use the “Clear All Data” button to reset members and expenses completely. This action cannot be undone.

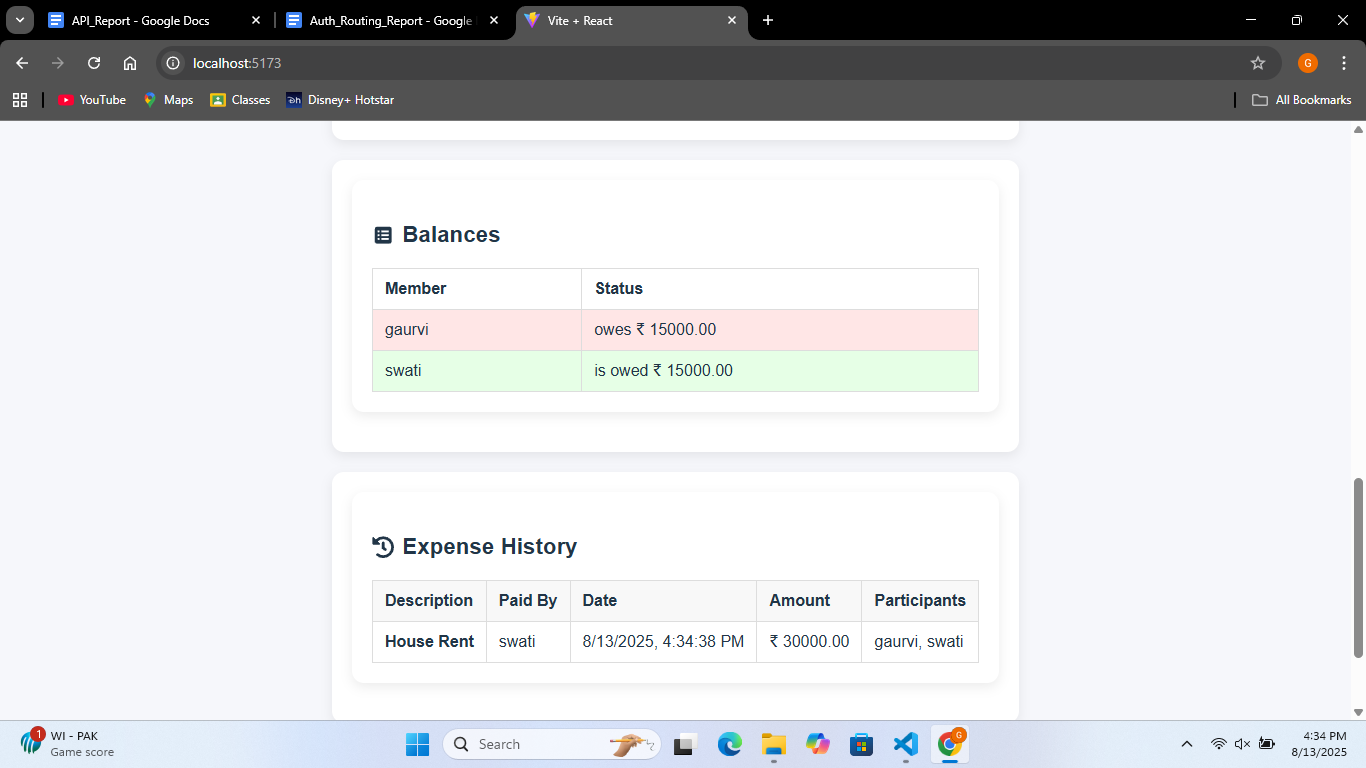
### **8. Challenges Faced & Solutions**

* Balances were miscalculated when members were removed.  
  **Solution:** Added a cleanup function to remove related expense entries before recalculating.
* Persisting data without a backend server.  
  **Solution:** Used browser **localStorage** to save and retrieve the group data in JSON format.
* Understanding the best way to split expenses among selected members.  
  **Solution:** Reviewed online documentation and examples to refine the calculation logic

**9. Screenshots**

****

****

****

### **9. Conclusion**

This project successfully delivers a functional and user-friendly expense sharing application that meets the stated objectives. Despite being frontend-only, the use of **localStorage** ensures data persistence without server-side integration. The app can be further enhanced in the future with backend support for multi-device synchronization and authentication.

### **10. References**

* [React Official Documentation](https://react.dev/)
* [Vite Official Documentation](https://vite.dev/guide/)
* [ChatGPT](https://openai.com/index/chatgpt/) by OpenAI – used for guidance and logic suggestions during development.