Split Payment App

Overview

In this assignment, you will build a web-based split payment application that allows users to track shared expenses and calculate how to settle balances within a group. This project simulates real-world financial technology applications and will demonstrate your ability to work with JavaScript, manage complex data structures, and create an intuitive user interface.

Learning Objectives

- Implement advanced JavaScript concepts, including ES6+ features
- Design and manipulate complex data structures
- Create an intuitive user interface with modern web technologies
- Handle financial calculations with precision
- Implement proper form validation and error handling
- Apply responsive design principles

Requirements

Core Functionality

1. **Group Management**
   - Create and name expense groups (e.g., "Vacation to Hawaii", "Apartment 4B", "Team Lunch")
   - Add/remove participants to/from groups
   - View all groups and their members

2. **Expense Tracking**
   - Add expenses with the following details:
     - Description
     - Amount
     - Date
     - Paid by (which participant)
     - Split method (equal, percentage, exact amounts)
     - Categories (optional: food, transportation, accommodation, etc.)
   - Edit or delete existing expenses
   - View expense history with filtering options

3. **Balance Calculation**

- Calculate what each person owes or is owed
- Generate a simplified settlement plan (minimum number of transactions to settle all debts)
- Display the balance summary for each participant

4. **User Interface**
   - Clean, responsive design that works on both desktop and mobile devices
   - Intuitive navigation between different sections
   - Clear visualization of expenses and balances

Technical Requirements

1. Frontend Development
   - Use vanilla JavaScript (ES6+) without frameworks like React, Angular, or Vue
   - Implement proper DOM manipulation for dynamic content
   - Style with CSS (preprocessors like SASS/LESS are optional)
   - Make the application responsive using media queries

2. Data Management
   - Store data using the browser's localStorage or sessionStorage
   - Implement proper data structures for efficient manipulation
   - Ensure data integrity across all operations

3. Code Quality
   - Follow JavaScript best practices and naming conventions
   - Include proper comments and documentation
   - Organize code in a modular, maintainable structure
   - Handle errors and edge cases gracefully

Submission Requirements

1. Complete source code with comments
2. README file with:
   - Setup instructions
   - Feature overview
   - Code structure explanation
   - Assumptions and limitations
   - Future improvements

Implementation Guidelines

1. Data Structure Suggestions

Example data structures

1. Group structure

```
const group = {
  id: "unique-id",
  name: "Trip to New York",
  members: [
    { id: "m1", name: "Alice" },
    { id: "m2", name: "Bob" },
    ...
  ],
  expenses: [
    {
      id: "e1",
      description: "Dinner at Restaurant",
      amount: 120.50,
      date: "2025-05-14",
      paidBy: "m1", (reference to member ID)
      splitMethod: "equal", "percentage", or "exact"
      splitDetails: {}, (depends on splitMethod)
      category: "food"
    },
    ...
  ]
};
```

2. Balance calculation example

```
const balances = [
  { member: "m1", balance: 45.20 }, // positive means others owe this person
  { member: "m2", balance: -20.30 }, // negative means this person owes others
  ...
];
```

3. Settlement plan example

```
const settlements = [
  { from: "m2", to: "m1", amount: 20.30 },
  ...
];
```

2. Suggested Project Structure

```
split-payment-app/
│
├── index.html
├── css/
│   ├── main.css
│   ├── responsive.css (You can ignore)
│   └── ...
│
├── js/
│   ├── app.js          # Main application initialization
│   ├── models/         # Data structures and business logic
│   │   ├── group.js
│   │   ├── expense.js
│   │   └── balance.js
│   │
│   ├── views/          # UI rendering logic
│   │   ├── groupView.js
│   │   ├── expenseView.js
│   │   └── balanceView.js
│   │
│   ├── controllers/    # Logic that connects models and views
│   │   ├── groupController.js
│   │   ├── expenseController.js
│   │   └── balanceController.js
│   │
│   └── utils/          # Helper functions
│       ├── storage.js
│       ├── calculator.js
│       └── validation.js
│
└── assets/             # Images, icons, etc.
    └── …
```

3. Key Algorithms to Implement

1. **Split Calculation Logic:**

   Implement algorithms for different splitting methods (equal, percentage, exact) and ensure they handle decimal precision correctly.

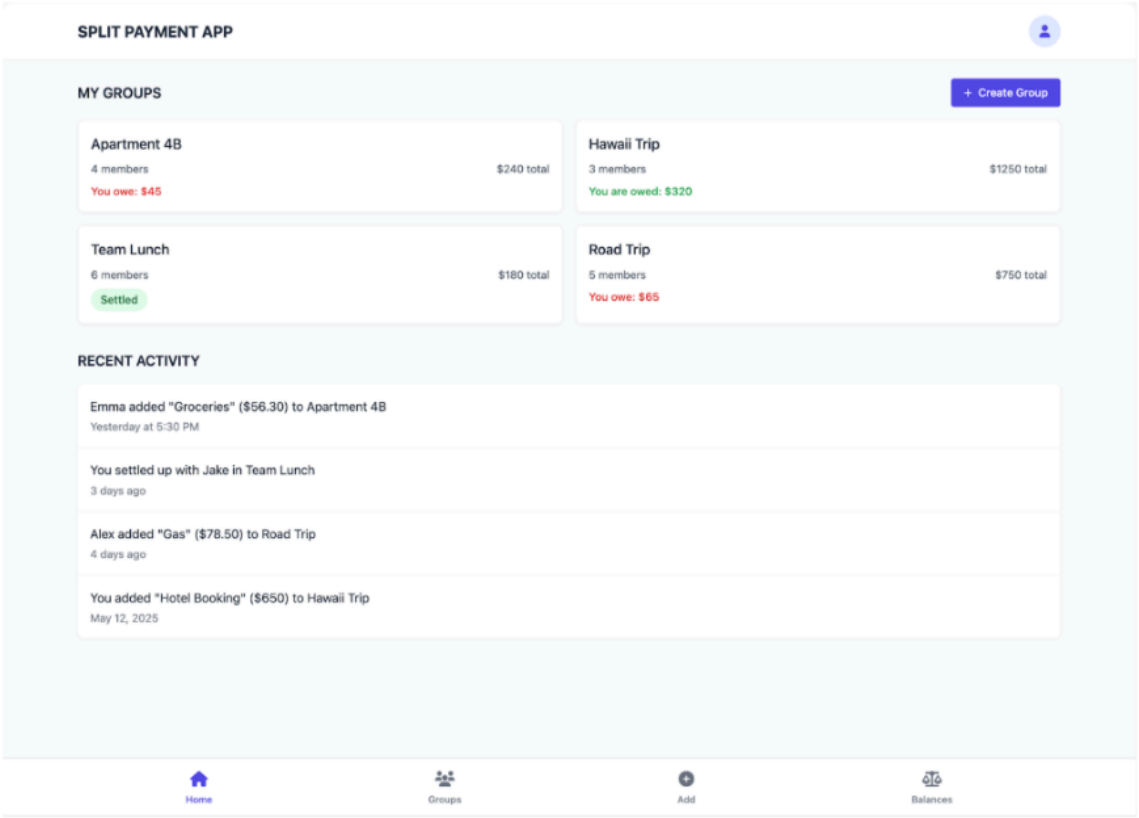
2. **Balance Resolution:**
   Create an algorithm that calculates the minimum number of transactions needed to settle all debts within a group.

3. **Data Persistence:**
   Implement saving and retrieving data from localStorage, including versioning if needed for future updates.

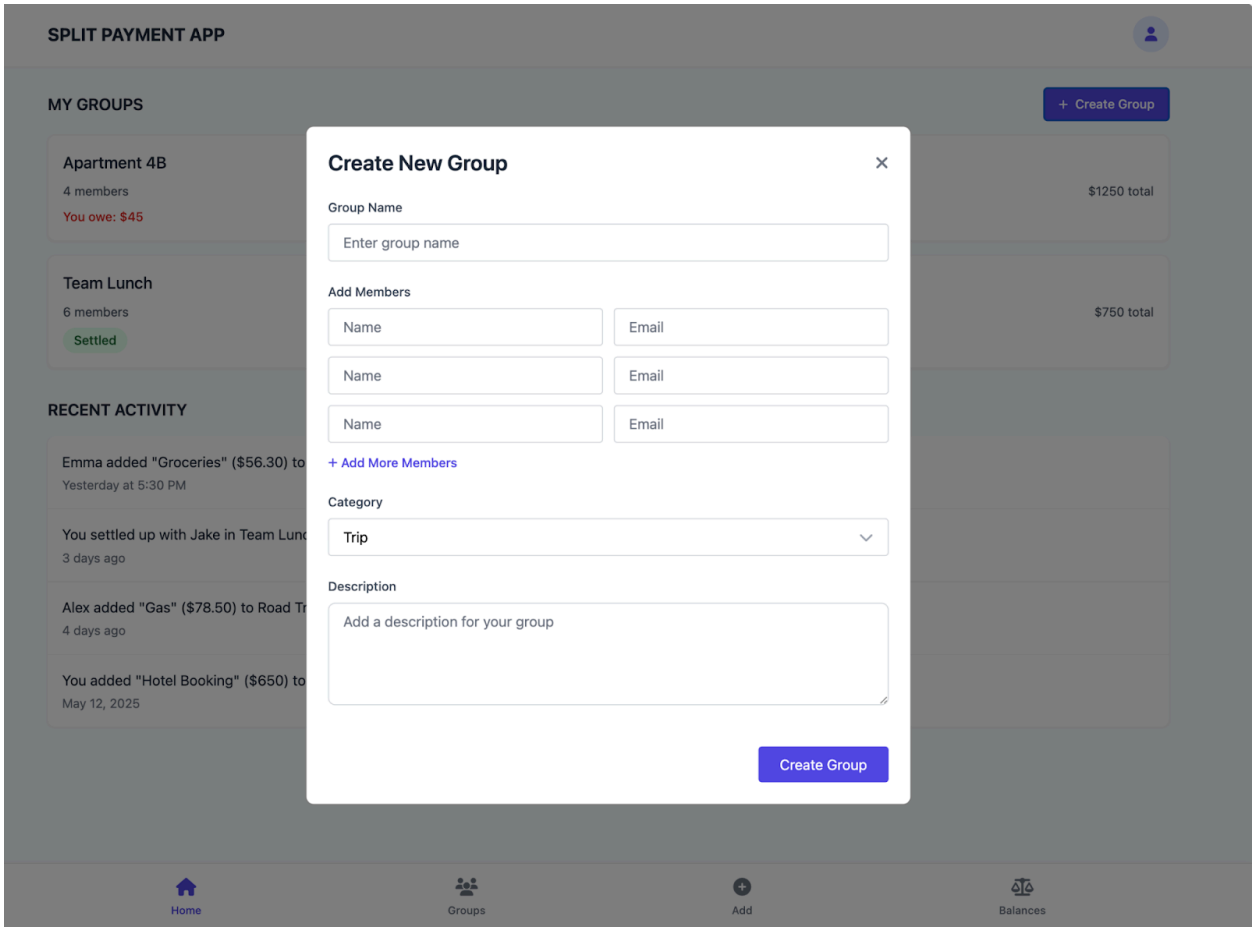
UX

1. Home page

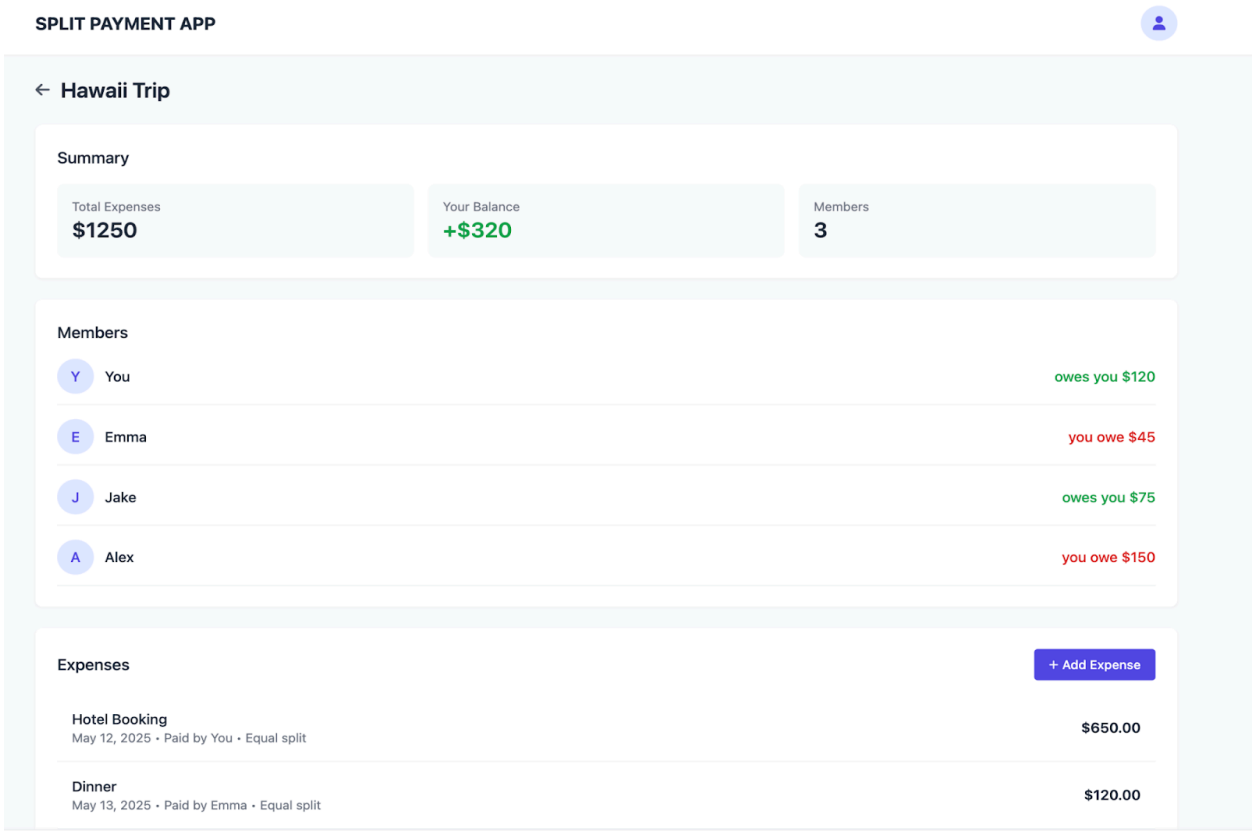
    1. Home page



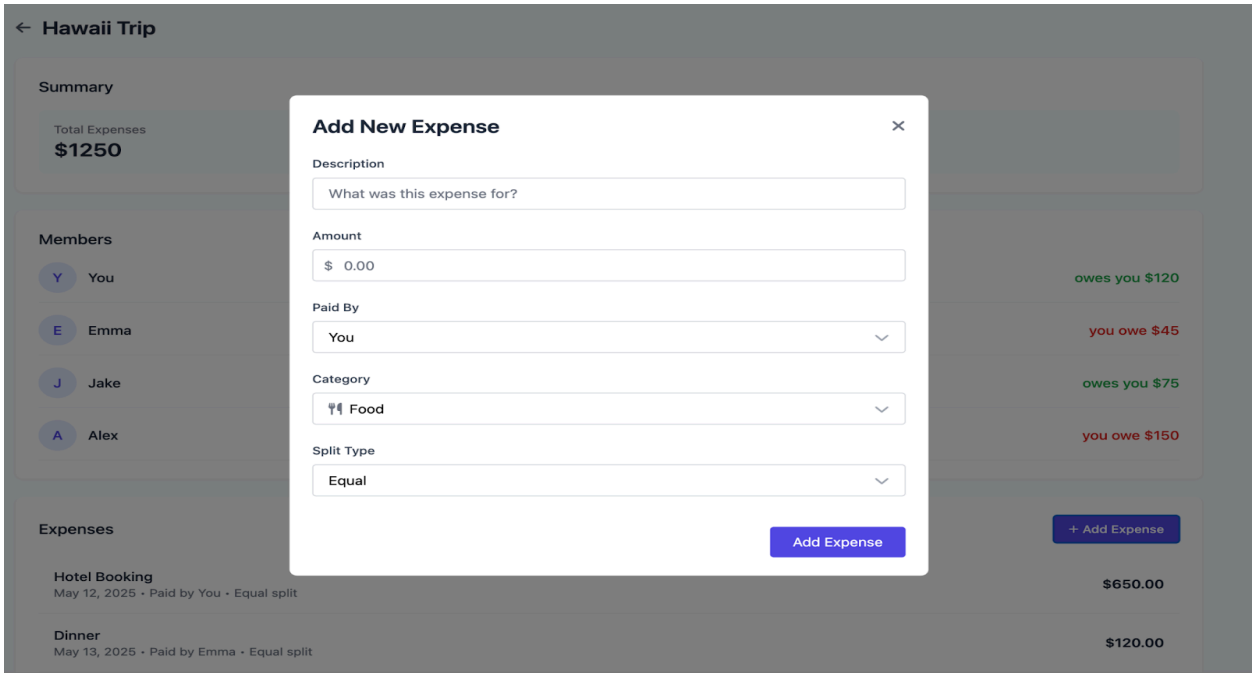
2. Create Group



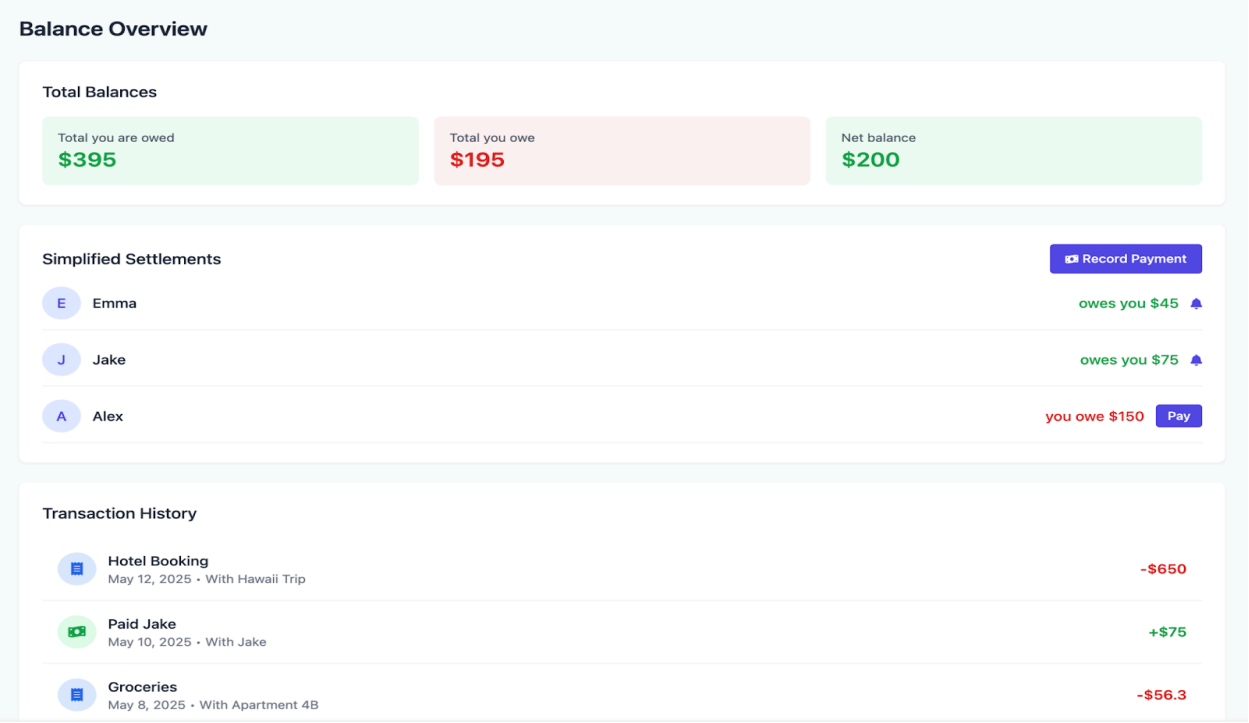
3. View Group



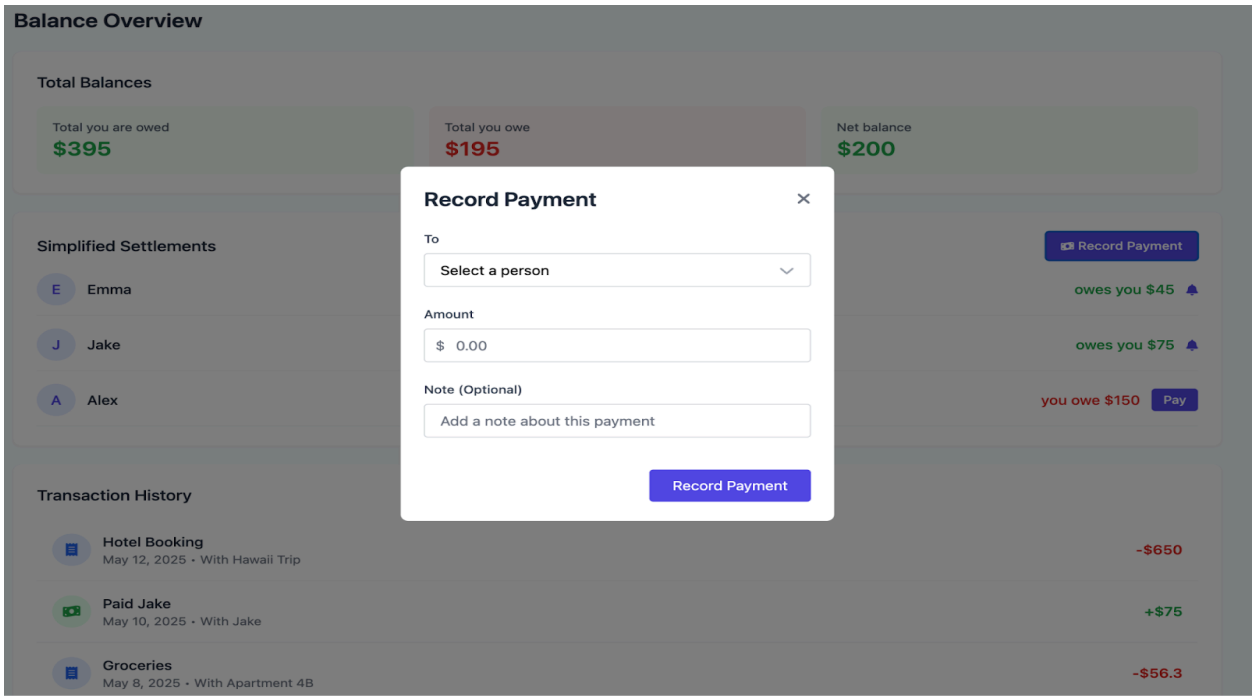
4. Add Expense



5. Balance Overview



6. Record payment



Good luck with your implementation! This project will give you valuable experience with JavaScript development and financial application logic that's highly relevant in today's tech industry.