Simple Expense Splitter Description:

Create a web application using React that helps users split expenses among a group of friends. The app will allow users to add friends, record expenses, and automatically calculate the amount each friend owes or is owed. This application is ideal for group trips, shared living situations, or any scenario where people need to split costs fairly and transparently.

Core Features

Add and Manage Friends:

- Allow users to add friends by entering their names.
- Display a list of added friends with options to edit or remove them.
- Ensure that each friend has a unique identifier.

Add and Manage Expenses:

- Allow users to add expenses by specifying the amount, description, and the person who paid the expense.
- Optionally allow users to add a date for each expense.
- Display a list of all added expenses with details such as the amount, payer, and participants.
- Include options to edit or delete expenses.

Expense Splitting Logic:

- Calculate the total expenses and display them to the user.
- Split expenses equally among the friends involved or allow customization for splitting based on specific criteria (e.g., percentage split, share per person).
- Show a summary of how much each friend owes or is owed, considering who paid for each expense.

Interactive User Interface:

- Create a dashboard that provides an overview of total expenses, the number of friends, and unsettled balances.
- Enable easy navigation between adding friends, adding expenses, and viewing summaries.

React Concepts

Components:

- **FriendList:** Displays the list of friends and provides forms for adding and editing friends.
- **ExpenseList:** Manages the list of expenses, including forms for adding, editing, and deleting expenses.
- **ExpenseSummary:** Displays the total expenses, per-person calculations, and the settlement summary.
- **Dashboard:** Provides an overview and serves as the main navigation hub of the application.

Services:

- **FriendService:** Handles the logic for managing friends, including adding, removing, and retrieving friend data.
- **ExpenseService:** Manages expense data, including adding, editing, deleting, and calculating the split amounts.
- **CalculationService:** Contains the logic for calculating total expenses, individual shares, and settlement amounts between friends.

Forms:

- Use controlled components to handle user input for adding friends and expenses, allowing for better state management and validation.

Routing and Navigation:

- Implement React Router to navigate between different sections of the application, such as the friend list, expense list, and summary view.

Note:

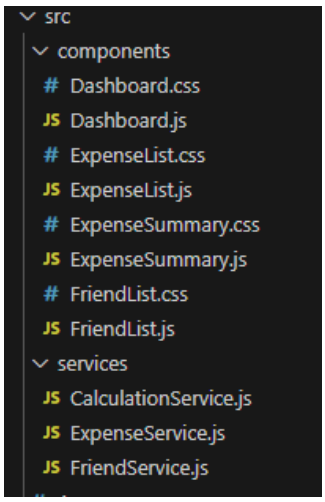
All the components needs to be created. Inside the `src/components` folder, learners have to create the following files with the names as given below:

- `FriendList.js`, `FriendList.css`
- `ExpenseList.js`, `ExpenseList.css`
- `ExpenseSummary.js`, `ExpenseSummary.css`
- `Dashboard.js`, `Dashboard.css`

As required, the service files are also to be created, and learners are supposed to create the below files only:

- `FriendService.js`, `ExpenseService.js`, `CalculationService.js`

Below is the structure of Components and service files it should look like this:



Ensure that CSS files are linked correctly and they provide proper styling for layout, visibility, and responsiveness. CSS files should be properly structured, and the JSX should be correctly formatted to display UI elements.

Following are the below rubrics to evaluate your project.

1. Functionality (30 points)
Core Features Implementation *(30 points)*

- **(10 points)** Users can add, edit, and delete friends successfully.
- **(10 points)** Users can add, edit, and delete expenses with correct details (amount, payer, participants).
- **(10 points)** Expenses are calculated correctly, showing how much each friend owes or is owed.

Expense Splitting Logic *(10 points)*

- **(5 points)** Expenses are split equally among participants, with an option for custom splits.
- **(5 points)** Summary correctly reflects the final amounts per person (who owes whom).

2. Code Quality (10 points)
Component Structure & Reusability *(5 points)*

- Proper separation of concerns—each component handles a single responsibility.
- Code is modular, with reusable components and services.

State Management & Data Handling *(5 points)*

- Uses React state/hooks effectively to manage application data.
- Services (FriendService.js, ExpenseService.js, CalculationService.js) are implemented properly for data logic separation.

3. UI & UX (8 points)
User Interface & Navigation *(4 points)*

- The UI is visually appealing and maintains a consistent design.
- The application is easy to navigate using React Router.

Forms & Input Handling *(4 points)*

- Input fields have validation (e.g., preventing empty names or negative amounts).
- Controlled components are used properly for managing form data.

4. Best Practices & Additional Features (2 marks)
Code Readability & Maintainability

- Code follows proper formatting and naming conventions.
- Comments should be added where necessary for clarity.
- Any extra functionality, you may add and customize CSS to view the pages.