Full-Stack Developer Take-Home Assignment

Expense Splitter Application

Overview

Build a full-stack expense splitting application where users can create groups, add expenses, and automatically calculate who owes whom. This assignment evaluates your ability to work with React, Node.js, and implement complex business logic.

Technology Stack

Required:

- **Frontend:** React with hooks
- **Backend:** Node.js with Express/Nestjs
- **Language:** TypeScript
- **Storage:** In-memory (arrays/objects) or SQLite

Optional (Bonus Points):

- TypeScript
- State management (Context API, Redux, Zustand)
- Styling library (Tailwind CSS, styled-components, Material-UI)
- Unit tests (Jest, React Testing Library, Supertest)

Part 1: Backend API (Node.js + Express)

Data Models

Implement the following data structures:

1. Group

- id (string/number)
- name (string, required)
- description (string, optional)
- members (array of member IDs)
- createdAt (timestamp)

2. Member

- id (string/number)
- name (string, required)
- email (string, optional)

3. Expense

- id (string/number)
- groupId (string/number, required)
- description (string, required)
- amount (number, required)
- paidBy (member ID, required)
- splitBetween (array of objects: { memberId, amount })
- splitType (enum: 'equal', 'percentage', 'exact')
- date (timestamp)

Required API Endpoints

Implement RESTful endpoints with proper error handling and validation:

Method	Endpoint	Description
POST	`/api/groups`	Create a new group
GET	`/api/groups`	Get all groups
GET	`/api/groups/:id`	Get group details with members and expenses
PUT	`/api/groups/:id`	Update group information
DELETE	`/api/groups/:id`	Delete a group
POST	`/api/groups/:id/members`	Add member to group
DELETE	`/api/groups/:groupId/members/:memberId`	Remove member from group
POST	`/api/expenses`	Create a new expense
GET	`/api/expenses`	Get expenses (with filtering by groupId)
PUT	`/api/expenses/:id`	Update expense
DELETE	`/api/expenses/:id`	Delete expense
GET	`/api/groups/:id/balances`	Get balance summary for group members
GET	`/api/groups/:id/settlements`	Get optimized settlement suggestions

Core Features to Implement

1. **Group Management:** Create, read, update, delete groups with member management
2. **Expense Tracking:** Add expenses with three split types: equal, percentage-based, or exact amounts
3. **Balance Calculation:** Calculate individual balances (who owes/is owed)

4. **Settlement Algorithm:** Optimize who should pay whom to minimize number of transactions
5. **Input Validation:** Validate all inputs (use Joi, express-validator, or similar)

Settlement Algorithm Details:

The settlement algorithm should minimize the number of transactions needed to settle all debts. For example:

- Alice owes $30, Bob is owed $20, Charlie is owed $10
- Optimal: Alice pays Bob $20, Alice pays Charlie $10 (2 transactions instead of potentially more)

Part 2: Frontend Application (React)

Required Pages/Views

1. **Groups List Page:** Display all groups, button to create new group
2. **Group Detail Page:** Show group info, members, expenses list, balances, and settlement suggestions
3. **Add Expense Form:** Form with validation to add expenses (support all three split types)
4. **Manage Members:** Add/remove members from a group

UI/UX Requirements

- Responsive design (mobile-friendly)
- Loading states during API calls
- Error handling with user-friendly messages
- Form validation with clear feedback
- Visual balance indicators (who owes, who is owed)
- Clear settlement suggestions display

Technical Requirements

- Use React Hooks (useState, useEffect, custom hooks)
- Proper component composition and reusability
- API integration using fetch or axios
- Client-side routing (React Router recommended)
- Proper error boundaries

Bonus Features (Optional)

Implementing any of these will earn extra credit:

- **Currency Support:** Allow multiple currencies with conversion
- **Expense Categories:** Categorize expenses (food, travel, utilities, etc.)
- **Data Visualization:** Charts showing spending patterns using Chart.js or Recharts
- **Export Functionality:** Export group expenses to CSV or PDF
- **Search & Filter:** Search expenses by description, filter by date range or member
- **Authentication:** Simple JWT-based authentication

- **Docker Setup:** Dockerfile and docker-compose.yml for easy deployment
- **Testing:** Unit and integration tests (Jest, React Testing Library, Supertest)

Evaluation Criteria

Criteria	Weight	What We Look For
Code Quality	**30%**	Clean, readable code with proper naming and structure
Functionality	**25%**	All core features work correctly, proper error handling
API Design	**15%**	RESTful design, appropriate status codes, validation
Frontend UX	**15%**	Intuitive interface, responsive design, good user feedback
Algorithm Implementation	**10%**	Correct settlement calculation and optimization
Documentation	**5%**	Clear README, code comments where needed

Submission Guidelines

1. **Code Repository:** Share a GitHub/GitLab repository link with clear commit history
2. **README:** Include setup instructions, API documentation, and any assumptions made
3. **Architecture Document:** Brief explanation of your design decisions and trade-offs (in README or separate doc)
4. **Demo Video (Optional):** Short video walkthrough of your application (2-3 minutes)

README Must Include:

- Prerequisites (Node version, npm/yarn, etc.)
- Installation steps
- How to run backend and frontend
- API endpoint documentation
- How to run tests (if implemented)
- Known limitations or future improvements

Example Scenarios to Test

Scenario 1: Weekend Trip

- Group: Weekend Trip with Alice, Bob, and Charlie
- Alice pays $120 for hotel (split equally)
- Bob pays $45 for gas (split equally)
- Charlie pays $60 for dinner (split equally)
- Expected: Each person should owe/be owed $75 total

Scenario 2: Unequal Split

- Group: Roommates with Alice, Bob, Charlie
- Alice pays $300 for rent
- Split: Alice 50%, Bob 30%, Charlie 20%
- Expected: Bob owes $90, Charlie owes $60

Post-Submission Interview Topics

After reviewing your submission, we will discuss:

5. **Architecture & Design:** Walk us through your code structure and design decisions
6. **Settlement Algorithm:** Explain your approach to minimizing transactions
7. **Scalability:** How would you modify this for production with 10,000 users?
8. **Testing Strategy:** What would you test and how?
9. **Security:** What security concerns exist and how would you address them?
10. **Feature Extension:** How would you add recurring expenses or payment reminders?
11. **Trade-offs:** What compromises did you make and why?

Important Notes

- **Code Quality Over Features:** We value clean, maintainable code over a feature-complete solution
- **Ask Questions:** If anything is unclear, please reach out to us
- **Document Assumptions:** If you make assumptions, document them in your README
- **Have Fun:** This is a chance to showcase your skills and creativity!

Good luck! We're excited to see what you build.