Split App - Backend Assignment

Problem Statement

Build a backend system that helps groups of people split expenses fairly and calculate who owes money to whom. Think of scenarios like roommates splitting utility bills, friends sharing dinner costs, or travel buddies managing trip expenses.

The system should handle adding expenses, tracking who paid for what, and automatically calculating settlements so everyone knows exactly how much they owe or are owed.

Refer to popular mobile apps like Splitwise and Google Pay Bills Split, etc.

Time Allocation: 2 Days

MUST HAVE Features (Core Requirements)

1. Expense Tracking

- Add new expenses with amount, description, and who paid
- People are automatically added when mentioned in expenses (no separate person creation)
- View all expenses with complete details
- Edit or delete existing expenses
- Option to select percentage/share/exact amount for an expense

2. Settlement Calculations

- Calculate how much each person has spent vs their fair share
- Determine who owes money and who should receive money
- Show simplified settlements (minimize number of transactions needed)
- Provide clear summary of who should pay whom and how much

3. Data Validation & Error Handling

- Validate all inputs (positive amounts, required fields, valid person names)
- Handle edge cases gracefully (empty expenses, invalid calculations)
- Return clear, helpful error messages
- Proper HTTP status codes for all scenarios

Expected API Endpoints

```
# Expense Management
GET    /expenses              # List all expenses
POST   /expenses              # Add new expense
PUT    /expenses/:id          # Update expense
DELETE /expenses/:id          # Delete expense

# Settlement Calculations
GET    /settlements           # Get current settlement summary
GET    /balances              # Show each person's balance (owes/owed)
GET    /people                # List all people (derived from expenses)
```

Example API Payloads:

```
// POST /expenses
{
  "amount": 60.00,
  "description": "Dinner at restaurant",
  "paid_by": "Shantanu"
}

// Response format
{
  "success": true,
  "data": {...},
  "message": "Expense added successfully"
}
```

🌟 OPTIONAL Features (Extra Credit)

1. Recurring Transaction

- Automatically track and split regular expenses like rent or subscriptions monthly/weekly, etc.

2. Expense Categories

- Assign categories to expenses (Food, Travel, Utilities, Entertainment, Other)
- Show spending breakdown by category

- Category-wise summaries and totals

3. Enhanced Analytics

- Monthly spending summaries
- Individual vs group spending patterns
- Most expensive categories or transactions

4. Simple Web Interface

- Basic HTML forms to add people and expenses
- Display current balances and settlements
- Simple dashboard showing key information

Deliverables & Testing Instructions

Public Postman Collection Required

You MUST provide a publicly accessible Postman collection that we can import and test immediately without any local setup.

Steps to Submit:

1. **Deploy your backend** to a free hosting service:
2. **Create Postman collection** with:
   - All required API endpoints
   - Sample requests with realistic test data
   - Pre-populated data that demonstrates the system working
   - Clear folder organization
3. **Export and share** your collection:
   - Export as JSON from Postman
   - Upload to GitHub Gist: https://gist.github.com/
   - Make the gist public and shareable
   - Include your deployed API base URL in collection
4. **Test data setup**:
   - Your APIs should work immediately when we import the collection
   - Pre-populate with sample people: Shantanu, Sanket, Om
   - Include sample expenses that show different scenarios
   - Settlement calculations should show realistic results

📁 **Expense Splitter APIs**
 📁 **Expense Management**

➢ **Add Expense - Dinner (₹600, paid by Shantanu)**
➢ **Add Expense - Groceries (₹450, paid by Sanket)**
➢ **Add Expense - Petrol (₹300, paid by Om)**
➢ **Add Expense - Movie Tickets (₹500, paid by Shantanu)**
➢ **Add Expense - Pizza (₹280, paid by Sanket)**
➢ **List All Expenses**
➢ **Update Expense - Change Petrol to ₹350**
➢ **Delete Expense - Remove Pizza**

📁 **Settlements & People**
➢ **Get All People (should show: Shantanu, Sanket, Om)**
➢ **Get Current Balances (after all transactions)**
➢ **Get Settlement Summary (optimized transactions)**

📁 **Edge Cases & Validation**
➢ **Add Expense - Invalid (negative amount)**
➢ **Add Expense - Invalid (empty description)**
➢ **Add Expense - Invalid (missing paid_by)**
➢ **Update Non-existent Expense**
➢ **Delete Non-existent Expense**
➢ **Get Balances - With No Expenses**

Complete Submission Checklist:

1. Code Repository

- [ ] Git repository with clear README
- [ ] Setup instructions for local development
- [ ] Clean, commented code
- [ ] Database schema/setup scripts

2. Deployed Demo

- [ ] Working backend deployed on free hosting
- [ ] All APIs accessible via public URL
- [ ] Database hosted online (not local SQLite)

3. Postman Collection

- [ ] Public GitHub Gist with collection JSON
- [ ] Collection uses deployed API base URL
- [ ] All endpoints have working examples
- [ ] Pre-populated with test data
- [ ] Clear documentation in request descriptions

4. Documentation

- [ ] README with API documentation
- [ ] Explanation of settlement calculation logic
- [ ] Known limitations or assumptions
- [ ] Brief demo video (optional but helpful)

Common Mistakes to Avoid

Deployment Issues:

- ❌ APIs only work locally
- ❌ Database requires local setup
- ❌ Postman collection has localhost URLs
- ❌ Missing environment variables in deployed version

Business Logic Issues:

- ❌ Incorrect settlement calculations (assuming equal split among all people)
- ❌ Not handling floating point precision in money
- ❌ Inconsistent person names across expenses
- ❌ Allowing negative amounts or invalid inputs

API Design Issues:

- ❌ Inconsistent response formats
- ❌ Poor error messages
- ❌ Missing input validation
- ❌ Incorrect HTTP status codes

Recommended Tech Stack

- **Backend**: Java (Spring Boot), Node.js, or Python
- **Database**: PostgreSQL (via Railway/Render) or MongoDB Atlas
- **Deployment**: Railway.app or Render.com
- **Testing**: Postman for API testing

Remember: Focus on getting core functionality working perfectly rather than adding many features. A simple system that works flawlessly is much better than a complex system with bugs!

Feel free to explore solutions using Generative AI tools—they can help you get unstuck, brainstorm ideas, or speed things up. Best of luck!