Design Document: Coffee Run

1. OVERVIEW

Coffee Run is a Flask-based web service designed to manage users, groups, and expenses related to coffee purchases. It provides functionality for:

1. User management
2. Group creation and management
3. Expense tracking
4. Balance calculations
5. Determining next payer

2. SYSTEM ARCHITECTURE

The application follows a client-server architecture:

- Frontend: React.js application
- Backend: Python Flask REST API
- Database: PostgreSQL

3. CORE FEATURES

- **User Management**
  - Create and view users
  - View user profiles with group memberships
- **Group Management**
  - Create groups with members
  - View group details with members and expenses
  - Add members to existing groups
- **Expense Tracking**
  - Record expenses with multiple items
  - Track who paid for what
  - View expense history
- **Balance Calculations**
  - Calculate net balances between members
  - View simplified and detailed balances
  - Determine who owes whom
- **Next Payer Determination**
  - Algorithm to suggest who should pay next based on current balances

4. DATABASE SCHEMA

- **users**: Stores user information
- **groups**: Stores group information
- **group_members:** Junction table for group membership
- **expenses**: Records expenses
- **expense_items:** Records individual items within expenses

HIGH-LEVEL DESIGN (HLD)

1. API ENDPOINTS

USER ROUTES

- GET /api/users - Get all users
- POST /api/users - Create new user
- GET /api/users/<user_id> - Get user details

GROUP ROUTES

- GET /api/groups - Get all groups
- POST /api/groups - Create new group
- GET /api/groups/<group_id> - Get group details
- POST /api/groups/<group_id>/members - Add member to group

EXPENSE ROUTES

- POST /api/expenses - Create new expense
- PATCH /api/expenses/<expense_id>/update-payer - Update expense payer

BALANCE ROUTES

- GET /api/groups/<group_id>/balances - Get simplified balances
- GET /api/groups/<group_id>/detailed-debts - Get detailed debts
- GET /api/groups/<group_id>/transaction-history - Get transaction history
- GET /api/groups/<group_id>/next-payer - Get next payer suggestion

2. DATA FLOW

1. Client makes HTTP request to Flask API
2. Flask processes request, interacts with PostgreSQL
3. Flask returns JSON response
4. Client renders data in React components

3. KEY ALGORITHMS

1. **Balance Calculation**
    - Aggregates all expenses and items
    - Calculates net amounts between all members
    - Simplifies mutual debts
2. **Next Payer Determination**
    - Identifies member with highest debt
    - Falls back to random selection if no debts exist