Engineering Design Assignment

Expense Sharing Application – Backend Design Report

1. Introduction

This project is a backend implementation of a simplified expense sharing application, inspired by Splitwise.
The system allows users to create groups, add shared expenses with different split strategies, track balances, and view simplified settlements.

The focus of this assignment is on:

- Clean backend design
- Correct business logic
- Data modeling
- Balance simplification
- Clear explanation of decisions

2. Problem Statement

Design a backend system that allows users to:

- Create groups
- Add shared expenses
- Track balances (who owes whom)
- Settle dues

Supported Split Types

1. Equal Split
2. Exact Amount Split
3. Percentage Split

Balance Tracking Requirements

- Track who owes whom
- Each user should be able to see:
    - How much others owe them
    - How much they owe
- Balances should be **simplified** to minimize transactions

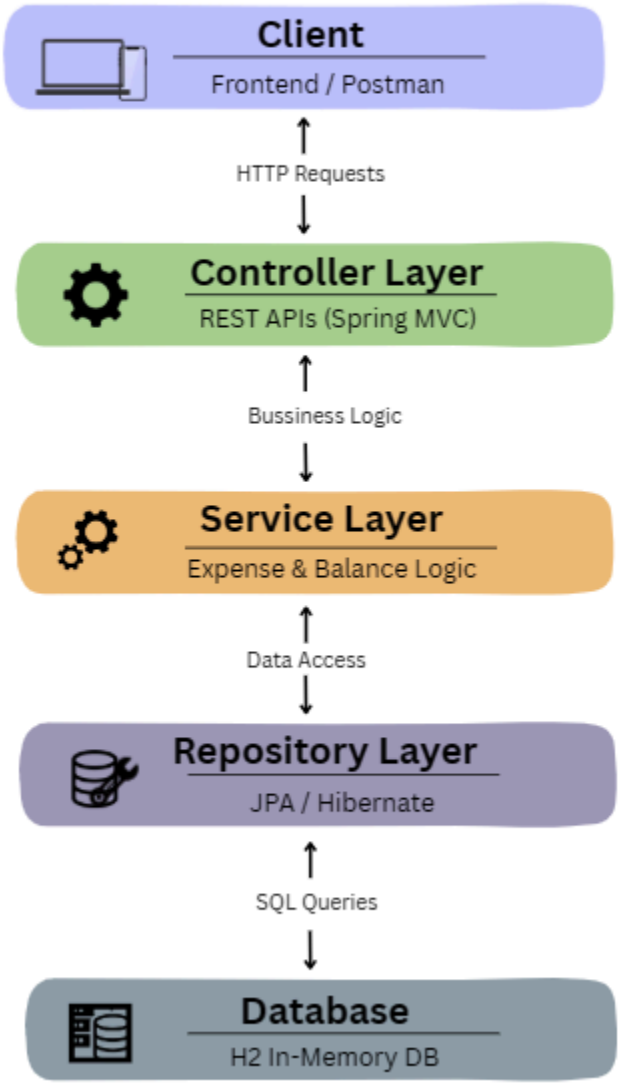
3. High-Level Architecture

The application follows a layered architecture, ensuring separation of concerns and maintainability.

Architecture Layers

1. **Controller Layer**
    - Exposes REST APIs Handles
    - HTTP requests/responses
2. **Service Layer**
    - Contains core business logic
    - Expense creation
    - Balance calculation and simplification
3. **Repository Layer**
    - Data access using Spring Data
    - JPAAbstracts database operations
4. **Database Layer**
    - H2 in-memory database
    - Automatically managed using Hibernate

Architecture



4. Entity Relationship Design (ERD)

Core Entities

User

- id (Long)
- name
- email

Group

- id
- name
- members (Many-to-Many with User)

Expense

- id
- description
- totalAmount
- splitType (EQUAL / EXACT / PERCENTAGE)
- paidBy (User)
- group (Group)
- settled (boolean)

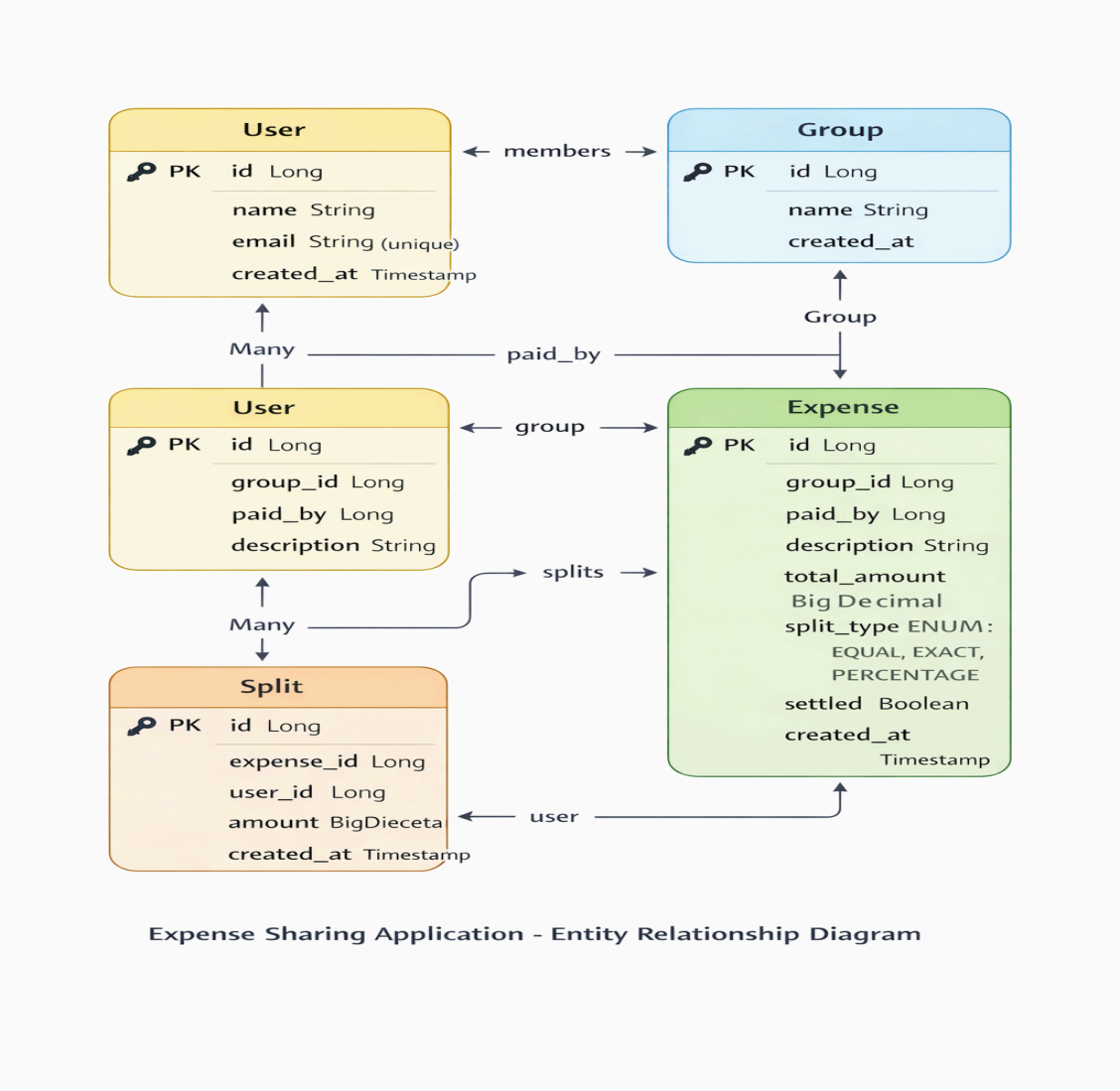
Split

- id
- expense (Many-to-One)
- user (Many-to-One)

- amount

Relationships

- A **Group** has many **Users**
- A **Group** has many **Expenses**
- An **Expense** has many **Splits**
- Each **Split** belongs to one **User**



Expense Sharing Application - Entity Relationship Diagram

This normalized structure avoids redundancy and supports accurate balance calculations.

5. Expense Handling Logic

Adding an Expense

When an expense is added:

1. The payer (`paidBy`) is identified
2. The total amount is split among users based on split type
3. Individual `Split` records are created
4. Expense and splits are persisted together

Supported Split Types

Equal Split

- Total amount divided equally among participants

Exact Split

- Each user pays a specified amount
- Validated so total matches expense amount

Percentage Split

- Each user pays a percentage of total
- Percentages sum to 100

6. Balance Calculation & Simplification

Net Balance Calculation

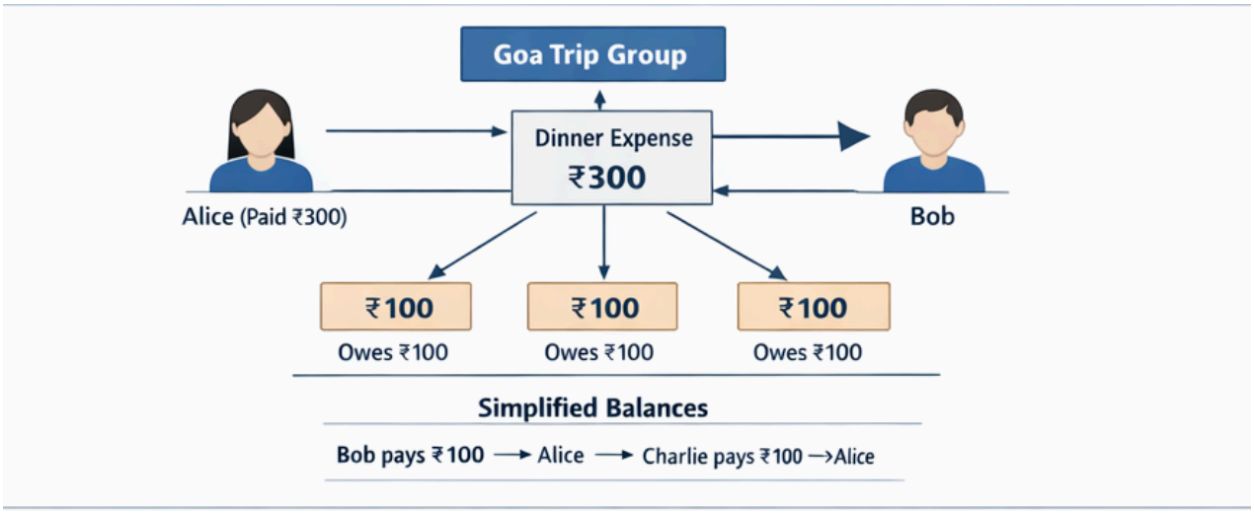
For each expense:

- Payer gets +totalAmount

- Each participant gets –their split amount

Net balance per user:

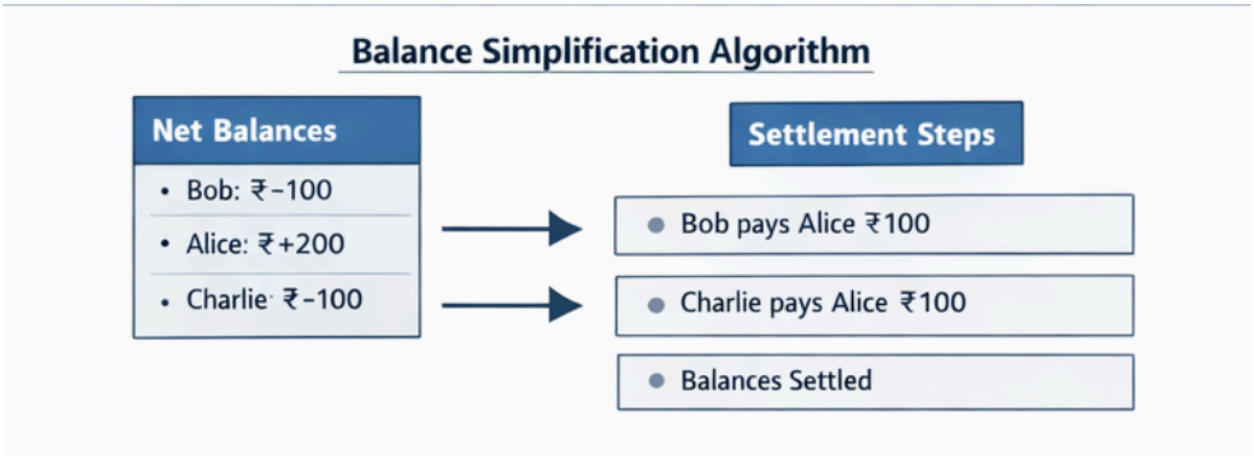
- Positive → user should receive money
- Negative → user owes money



Balance Simplification Algorithm

To minimize transactions:

1. Separate users into:
    - Creditors (positive balance)
    - Debtors (negative balance)
2. Match largest debtor with largest creditor
3. Transfer minimum possible amount
4. Update balances
5. Repeat until all balances are settled



This ensures minimum number of transactions.

7. API Design

User APIs

- **`POST /users`** → Create user
- **`GET /users`** → List users

Group APIs

- **`POST /groups`** → Create group
- **`GET /groups/{id}`** → Get group details

Expense APIs

- **`POST /expenses`** → Add expense
- **`GET /expenses/group/{groupId}`** → List group expenses

Balance APIs

- **`GET /balances/{groupId}`** → Get simplified settlement

8. Technologies Used

- **Java 17**
- **Spring Boot**
- **Spring Data JPA**
- **Hibernate**
- **H2 In-Memory Database**
- **Postman** (API testing)
- **Maven**

9. Testing Approach

The application was tested using Postman:

Testing Flow

1. Create users
2. Create group with users
3. Add expenses (Equal / Exact / Percentage)
4. Fetch balances for group
5. Verify simplified settlements

All APIs return expected responses and maintain consistency.

10. Assumptions & Limitations

Assumptions

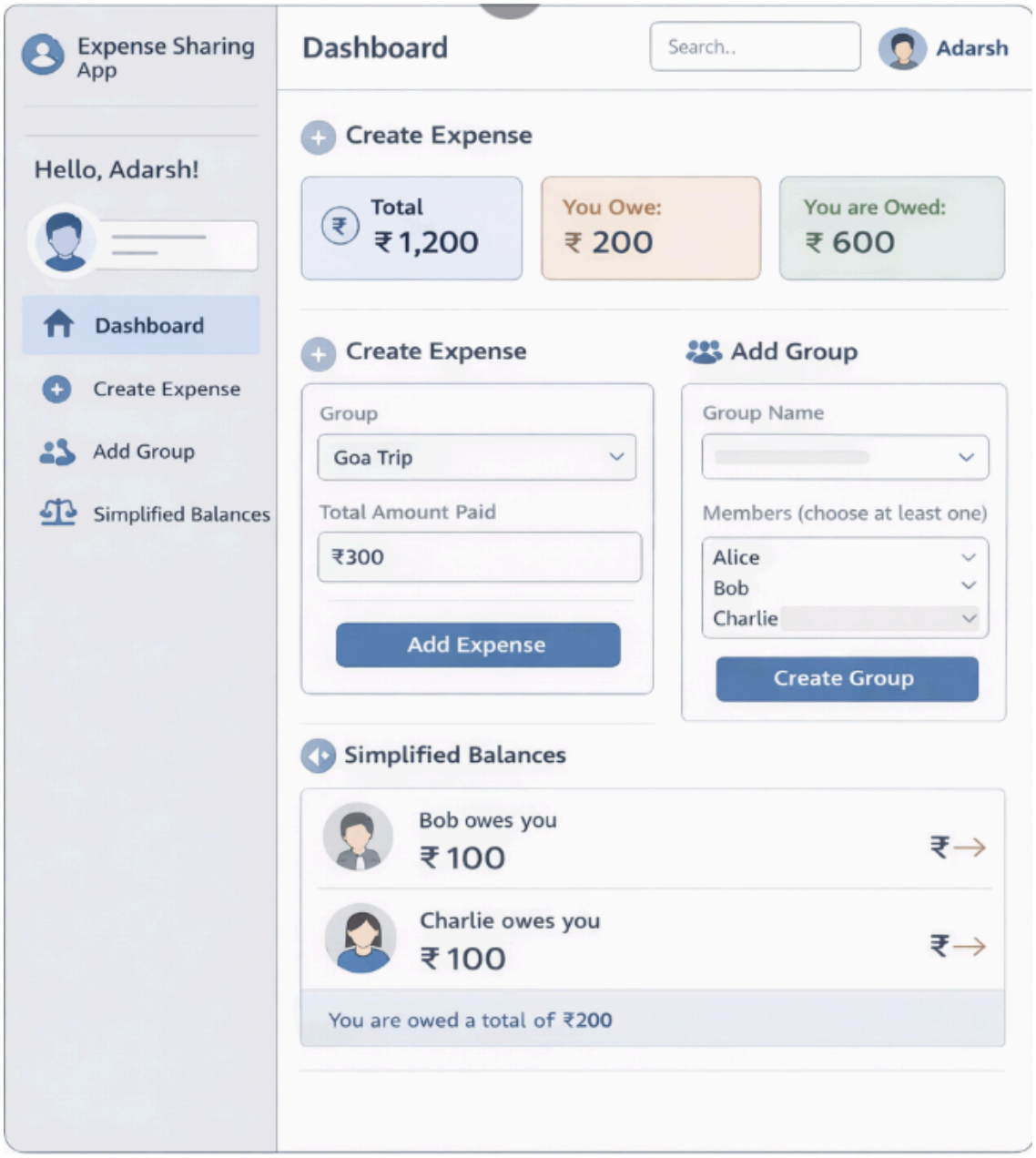
- Authentication is not required
- Users are pre-created before adding expenses

- All balances are settled within a group

Limitations

- No UI (backend-only)
- No persistent database (H2 used for simplicity)
- No activity logs or comments

Wireframe Design



Link of doc:

https://docs.google.com/document/d/15pzfegxe-D42m5VtBzsKOxRMKkCXlUTYLbhJSXCGHT8/edit?usp=sharing

SUBMITTED BY:

ADARSH GAURAV