# Splitwise - Design Document

## Problem Statement

Managing shared expenses among friends and groups leads to manual calculations, complex settlements, lack of transparency, and social friction over money disputes. Splitwise automates expense management, optimizes settlements using heap-based algorithms, and provides transparent tracking of group finances.

## Requirements & Features

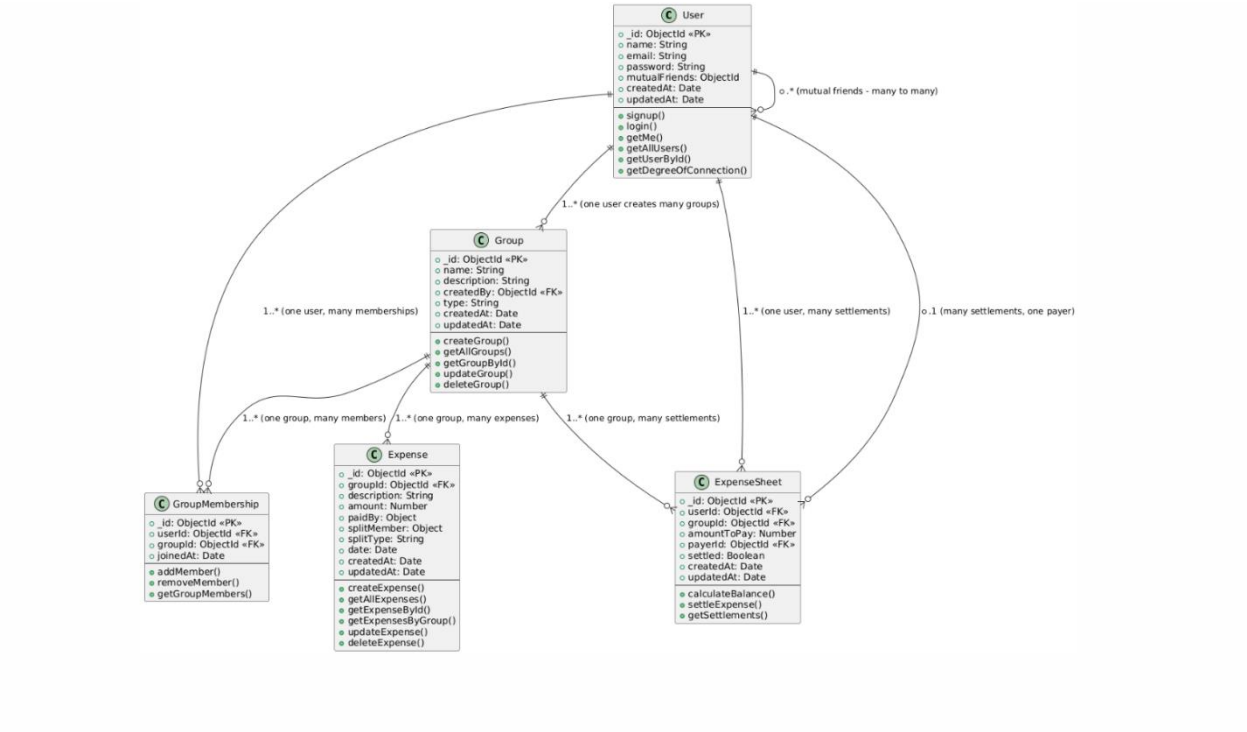
### Functional Requirements

- User registration, authentication, and profile management
- Create and manage expense groups with member addition/removal
- Add expenses with multiple payers and flexible split types (equal, custom)
- Calculate optimal settlements using heap algorithm to minimize transactions
- Track settlement status and generate recommendations
- Find degree of connection between users using BFS algorithm
- View expense history and user involvement tracking

### Non-Functional Requirements

- API response time < 500ms, support 1000+ concurrent users
- JWT-based authentication with bcrypt password hashing
- Responsive design for mobile and desktop
- 99.9% uptime with automated error handling
- Horizontal scaling capability and database optimization

## Class Diagram




## Cardinality Relationships

**User ↔ Group (1:Many):** One user can create multiple groups, but each group has exactly one creator. This establishes ownership and administrative control.

**User ↔ GroupMembership (1:Many):** One user can join multiple groups through separate membership records, enabling users to participate in various expense groups simultaneously.

**Group ↔ GroupMembership (1:Many):** One group can have multiple members, with each membership representing a unique user-group association with join timestamps.

**Group ↔ Expense (1:Many):** One group can contain multiple expenses, but each expense belongs to exactly one group, maintaining expense organization and context.

**User ↔ ExpenseSheet (1:Many):** One user can have multiple settlement records as both payer and payee, tracking all financial obligations across different groups and expenses.

**ExpenseSheet ↔ User (Many:1):** Multiple settlement records can reference the same payer, allowing one user to be responsible for settling multiple debts.

**User ↔ User (Many:Many):** Self-referencing relationship for mutual friends, enabling the BFS algorithm to find degrees of connection between users in the social network.

**Expense ↔ User (Many:Many):** Complex relationship where expenses can involve multiple users as both payers and split members, with amounts tracked in embedded arrays for flexible expense splitting scenarios.