SplitBill Assignment

26th November 2020
Version 0.2

Prepared For:
HUEx

Hashedin

Revision History

Version	Date	Author	Changes
0.1	11-02-2020	Ayush	Initial Version
0.2	26-11-2020	Aniket	Updated for HUEx

Objective

HashedIn University has been assigned a task to build a platform to make splitting bills easy. This document describes the business context of the application that needs to be developed for the same.

Business Context

Hashedin organises employee-engagement activities every week. As a part of these, it holds regular Football tournaments, poker etc among others.

Hashedin wants to make bill splitting easier for the employees. An application that helps the organisers and the group members keep track of shared expenses, so that bills (and members) get paid on time would go a long way in easing out this process.

To accomplish this, Hashedin University has been given this opportunity to come up with a platform which uses concrete logic helping people keep track of their expenses.

Project Plan

The delivery is expected to be high quality, iterative and showing progressive enhancements without waiting for a last-minute integration. There are daily checkpoints to keep up with the progress.

The milestones below show the must-have features and how they should be planned for delivery. The remaining time should be used to complete the functionality and prioritized based on the business context.

Milestones

- Everything is in-memory (Should be maintained in a list/map etc.).
- Unit test cases are **must** for each functionality you implement.
- If input for the function in invalid, your function should be able to communicate the same to the caller.

Milestone - 1	Models- Users, Groups, Bills (See milestone 2, 3 and 4 for requirements)
Milestone - 2	- Create functions to fetch, create, update and delete (soft delete) a user. - Users should have a unique identifier like email or phone no. apart from an integral primary key.
Milestone - 3	- Create functions to get/create bills.
Milestone - 4	- Create functions to get total balance for a user (user owes to other users or other users owe him). - Create functions to get individual balances (with respect to per user) for a user.

QUALITY FOCUS
Even though the client doesn't mention quality, the quality focus has to be there. Whatever we deliver has to be delivered with excellent quality. This is important for getting continued business and establishing the value HashedIn can deliver to them i.e. "High Quality delivered Quickly"

Functional Requirements

Bills

Users should be able to add bills by specifying the users among which bill is to be splitted or a group, the amount of the bill, what is this bill for, who paid the bill, when this expense occurred and other necessary fields.
In order to add a user to the bill, the user account should be created on the platform.

Settlements

Settlements are simply a way to clear dues on a bill. A user can settle a bill by paying a partial or full amount of the bill. All settlements for a bill should be recorded.

When getting details of a bill, it should get all the basic details of the bill in addition to this it should also get the amount owed by each person for this bill.

Balance

Total- If Alice owes Bob 200 for bill B1 and Rachel owes Alice 100, then total balance for Alice is -100.
Individual- If Alice owes Bob 200 for bill B1 and Rachel owes Alice 100, then individual balances for Alice will be-

- Bob: -200
- Rachel: 100

Split bill among users

Suppose a bill is to be split among: Alice, Bob and Rachel. Alice paid 300 for Cab to be split among all users, meaning 100 would be due by Bob and Rachel.
Bob pays 50 to Alice, this means he still owes 50 and Rachel still owes 100 to Alice.
Now Rachel pays 100 then Rachel owes 0 and Bob owes 50 to Alice.

Split bill in a group

Suppose the group Trip consists of 3 members: Alice, Bob and Rachel.
Alice paid 300 to be split among all members, meaning 100 would be due by Bob and Rachel. Bob pays 50. This means he still owes 50 and Rachel owes 100.

Delete Bill and Undo

Users should be able to delete a bill (partially settled and settled bills should not be deleted). In reality a bill should never be deleted but only marked as deleted so that it can be undone later if required. Once a bill is marked as deleted, its amount should not reflect in users total dues.
Ex-

- Bob owes Alice 100 for bill B1
- Bob added a bill B2 with Alice for 50
- Now Bob owes Alice 50
- Bob deleted bill B2
- Now Bob owes Alice 100
- Bob undos delete bill B2
- Now Bob owes Alice 50

Simplified Debt

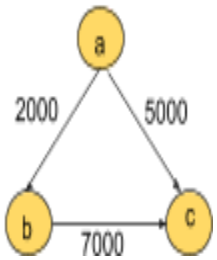
Simplify Debts is a feature of an application which can be used to restructure debt within a group. It makes it easier to pay people back by minimizing the total number of payments.
It should meet the following requirements-

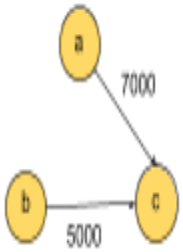
- Feature does not change the total amount that anyone owes.
- No one owes a person whom they didn't owe before.
- No one owes more money in total than they did before the simplification.

Example:

1. A owes B 200, B owes A 200. **Output:** A owes nothing to B and B owes nothing to A.
2. A owes B 2000, B owes C 7000, A owes C 5000. **Output**: A owes C 7000, B owes C 5000, A owes nothing to B.

:





3. A owes B 2000, B owes C 7000 **Output**: No simplification,A owes B 2000, B owes C 7000.

Summary Report

- Generate a summary report in Json format (How much the user owes and is owed) for each group user belongs to.
- Report should be generated for each user at a specific time. (For Example: Everyday at 12:30 am)

Non Functional Requirements

Best Practices

Verify your codebase. There shouldn’t be any Detekt errors.
Unit tests handling business cases should be written for all functionalities.

Evaluation Criteria

Merely covering the requirements would not be enough. There would be heavy focus on code quality , unit test cases and robustness of code in general. The evaluation criteria would be shared as the project progresses. At all times focus will be on non-functional, quality and following proper S/E processes. A production ready code quality is expected.