**Full Stack Web Application Project Final Report**

**Team Name:** iTeam

**Application Name:** SmartSplit

**Coordinator:** Ashwini Prabhu (G48389451)

**Members:** *Shuo Zhang(G20811169), Rahul Dhamnani (G37000752), Mruganka Londhe (G36912321), Yifei Shen(*G49720084*)*

**Source Code Github Repository URL**: https://github.com/ashwinipra/SmartSplit.git

**Table of Contents**

Application Summary 3

Architecture Report 3

Project Architecture 3

Client application: 3

Server: 3

Data Access: 4

Database: 4

Micro Service Architecture Implementation 4

Technology Report 4

MySQL 4

Java 5

Spring 5

Hibernate 5

JSP 5

Maven 6

Tradeoffs in Technologies 6

User Management: 6

API Document 6

Transaction: 6

User: 7

Testing Report 8

Development Report 8

What were the easiest parts of development process? And why? 8

What were the hardest parts of development process? And how did you deal with it? 8

If you could make different decisions regarding any aspect of this project, what would it be and why? 9

How did you manage work distribution among the group members? 9

What were the hardest parts of working in a group environment? And how did you deal with it? 9

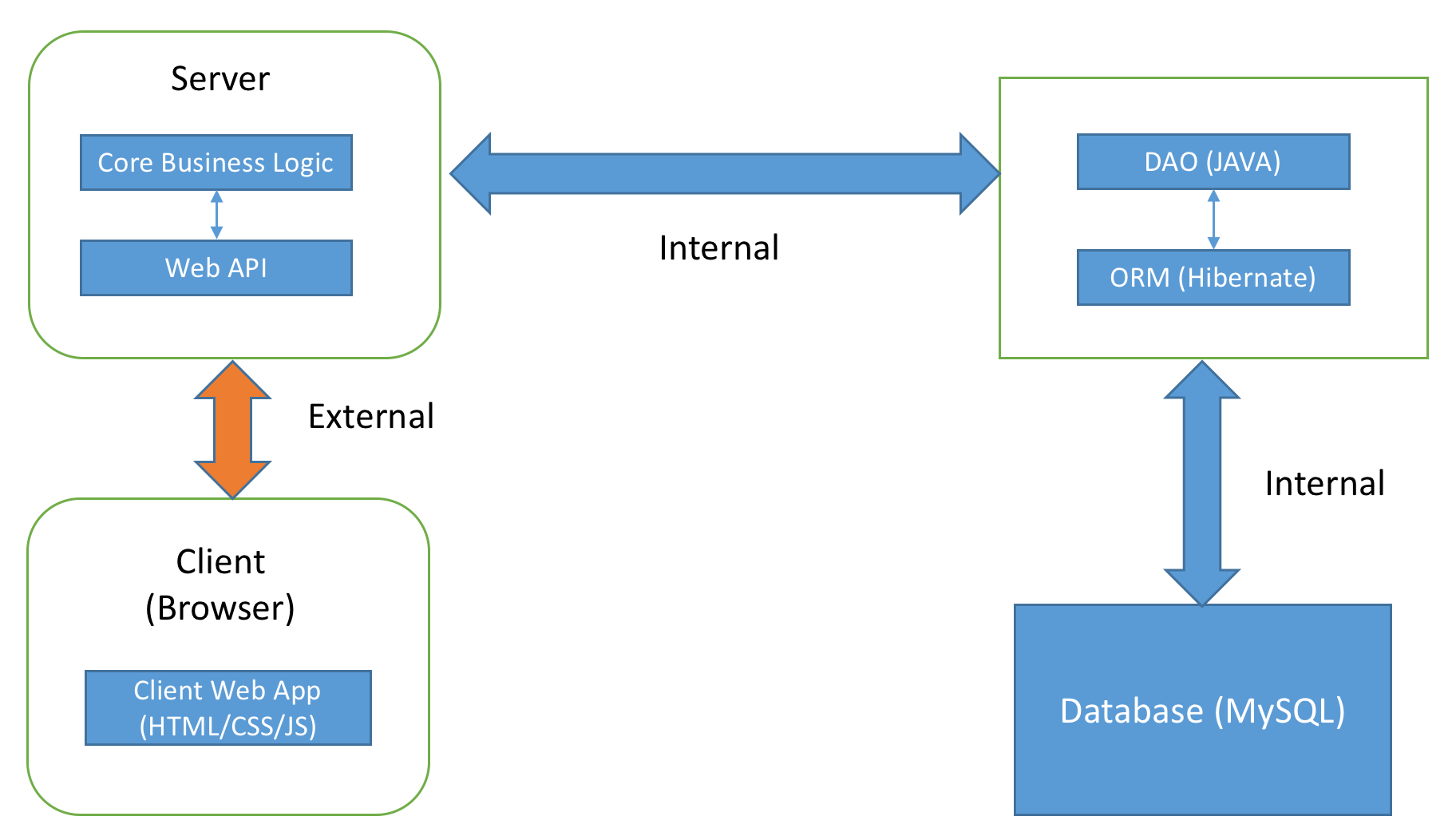
If you were to continue developing the application, what would be the next step? 9

# Application Summary

SmartSplit is a website that allows the user to split bills with roommates, friends and family. It provides an interactive graphical interface to keep track of the expenditure. It will also allow the user to update/delete the bill as well as to modify his/her information. User can view the details at the high level of his friends as well as at the granular level of individual transaction. Every user is required to create an account with full name, email ID and a password. When the user logs in with a valid email ID and password, a session is maintained until he/she logs out.

# Architecture Report

## Project Architecture



The application consists of four parts that are server, client application, DAO and Database.

### Client application:

Creates applicable requests for the server to obtain the user-desired results.

### Server:

It takes the request from clients, connects to the backend, gets results and sends an appropriate response back to the client. It has core business logic and web API layers.

#### Web API

This layer helps in building restful HTTP services very easily. It also provides well-formatted responses e.g. JSON.

#### Core Business Logic

This layer processes the raw data coming from the database and converts it into required output, such as transactions and user management. It is written in Java.

### Data Access:

It is the interface for storing and retrieving data. It connects server and the database.

#### Data Access Object (DAO)

It’s the interface for storing and retrieving the data from the database to the next layer. It’s written in Java.

#### Object Relational Mapper (ORM)

It’s an abstraction over the database interface, we have used Hibernate as our ORM, it is written in Java and excellent in performance.

### Database:

It is physical storage for data. At a particular point in time database represents the state of the application for that time.

## Micro Service Architecture Implementation

We wrapped DAO module and core business logic module as jar files which we imported into Web module. This kind of architecture helps in keeping a project as a suite of independently deployable, small, modular services. Moreover, using micro service architecture the overhead of testing, changing and maintenance decreases. Also, these modules are reusable into different project with similar business requirements.

# Technology Report

## MySQL

MySQL is an open source relational database management system developed by Oracle. First we chose to work with relational databases and then we selected to build our database using MySQL.

Reasons to choose:

* All group members have worked with relational database in the past and were comfortable in writing SQL queries.
* MySQL compared to other database is easy to install and easy to work with in JAVA.
* Relation database was selected because our database schema is much easy to implement in relational database compared to NoSQL and also none of the group members have worked with NoSQL database.

## Java

We have used Java for writing our core business logic as well as the DAO module. The web APIs are also written in Java.

Reasons to choose:

* Most of us know java very well.
* We decided that Object oriented approach would work best for our project.

## Spring

Spring is an application framework and inversion of control container for the Java platform.

Reasons to choose:

* Spring utilizes some of the well-known technologies, like ORM, logging framework, JSON Wrapper, which has been implemented in the project.

## Hibernate

Hibernate is an Object/Relational Mapping (ORM) framework, Hibernate is concerned with data persistence as it applies to relational databases. We choose Hibernate as our ORM.

Reasons to choose:

* Written in Java
* Removes overhead of database connections, handling exceptions, managing keys in database
* High performance
* Easy to deploy
* Makes codebase independent of Database

## JSP

JavaServer Pages (**JSP**) is a technology that helps software developers create dynamically generated web pages based on HTML, XML, or other document types. We have used JSP to make our frontend pages.

Reasons to choose:

* Since our codebase is written in Java we wanted scripting language that works with Java
* Easy to write frontend pages.
* Supports JSON objects.

## Maven

Apache Maven is a software project management and comprehension tool. We have used maven to build and manage our project.

Reasons to choose:

* Maven can easily manage all the dependencies in the project.
* Maven works well with Java Project.

## Tradeoffs in Technologies

* Python is a much easier language to implement the web module but we all decided to spend more time in architecture and full stack development instead of learning a new technology and using it.
* For database storage purpose we chose relational database. Relational databases are difficult to construct and maintain when we are dealing with unstructured data like JSON objects. NoSQL works best with such data but none of the team members had worked with NoSQL database and we would have wasted a lot of time in understanding the working and implementing NoSQL database compared to constructing a Relational database schema.
* We chose Spring over other simpler framework because the configuration is made very easy by Spring annotations.

## User Management:

Every user is required to create an account with full name, email ID and a password. When the user logs in with a valid email ID and password, a session is maintained until he/she logs out.

# API Document

## Transaction:

*show main page*

**Endpoint**: GET /mainPage

*show login page*

**Endpoint**: GET /login

*show logout page*

**Endpoint**: GET /logout

*show dashboard page*

**Endpoint**: GET /dashboard

*add transaction*

**Endpoint**: POST /dashboard/addt/username

**Sample Payload**:

{"addbillnew","userid"}

*showFriendsTransactions*

**Endpoint**: POST /dashboard/friend/username

**Sample Payload**:

{" friendsTransactions","username"}

*show a single transaction page*

**Endpoint**: POST /dashboard/friend/transaction/transactionid

**Sample Payload**:

{" "transaction", transid}

*update user*

**Endpoint**: POST /dashboard/ updateusr

**Sample Payload**:

{" updateuser"," usr"}

*update transaction*

**Endpoint**: POST /dashboard/updatetrans/transid

**Sample Payload**:

{" transaction "," transid "}

*settle transaction*

**Endpoint**: POST /dashboard/updatetrans/transid

**Sample Payload**:

{" transaction "," transid "}

## User:

*Show user login page*

**Endpoint**: GET /userlogin

*Register new user*

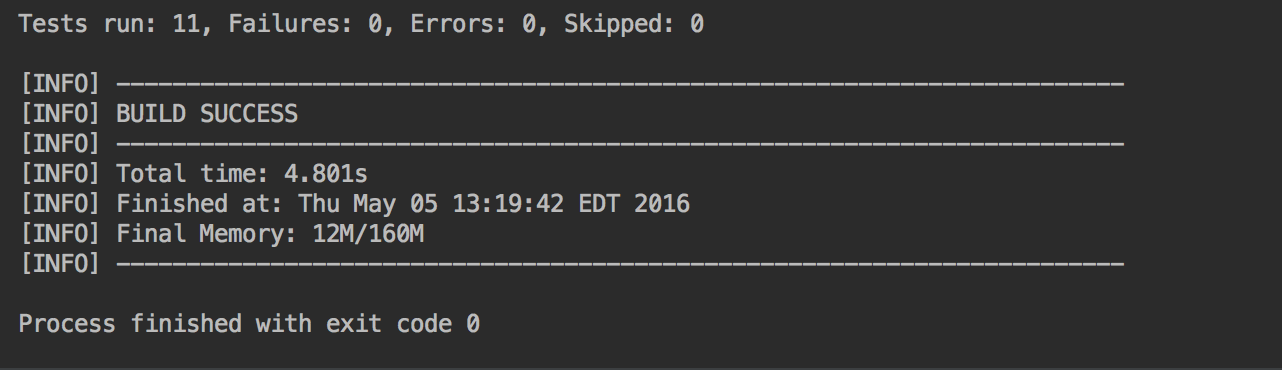
**Endpoint**: GET /reg

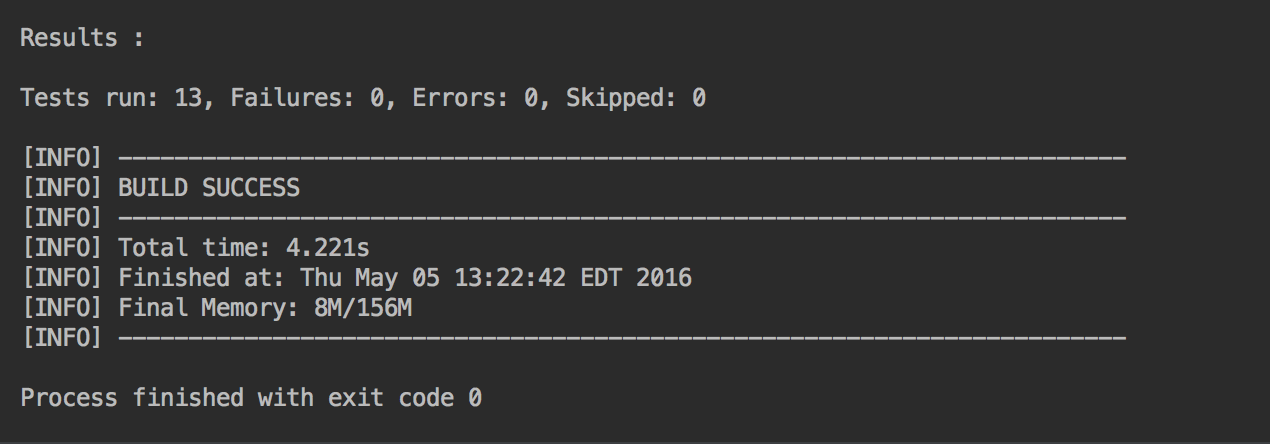
**Sample Payload**:

{" register","username"}

# Testing Report

Using JUnit Framework we have performed Unit and Integration testing. Most of the unit test cases are written in SmartSplitDAO. The integration testing is done in SmartSplitCore. In DAO module we have written altogether 11 test cases. In Core module we have written altogether 13 test cases.





# Development Report

## What were the easiest parts of development process? And why?

The easiest part of the development was developing the java code. Before we start coding, we had to set the developing environment, decide what was needed to be finished, decide what tools to use and learn how to use those tools. After all this preparation actual coding might be the easiest part.

## What were the hardest parts of development process? And how did you deal with it?

One of the hardest parts of the development process was deciding the technology to use. At the beginning of the project, we had couple of meetings to decide how the architecture of the project would be, which programming language we are going to use, which tools could be helpful and so on. We moved to sWe moved to MySQL from FireBase and from Angular JS/ Node.js to Spring + Hibernate. We took couple of week to discuss and decide all those things.

Another hard part was merging everyone’s code. Sometimes conflicts occured, sometimes there were compatibility issues. In this process, communication in the team is very important, we need to sit down and solve the issue together.

## If you could make different decisions regarding any aspect of this project, what would it be and why?

If we had more knowledge and time, we could have learnt and used node.js to develop this website. Node.Js is very popular these days because of its powerful feature. So many people work on the npm platform writing modules and plugins to make the development process easier.

We could have also used a NoSQL database in that case. We would have tried deploying the application on a real web server, instead of running on our local machine. For this purpose we could use one of our machines to work as a server and be accessed by the public internet or we could use Amazon AWS or Google Cloud.

## How did you manage work distribution among the group members?

We used Git which might be the most popular version control system, to manage and keep our project. The benefit is obvious; it can keep all of us on the same page. We also used email.

## What were the hardest parts of working in a group environment? And how did you deal with it?

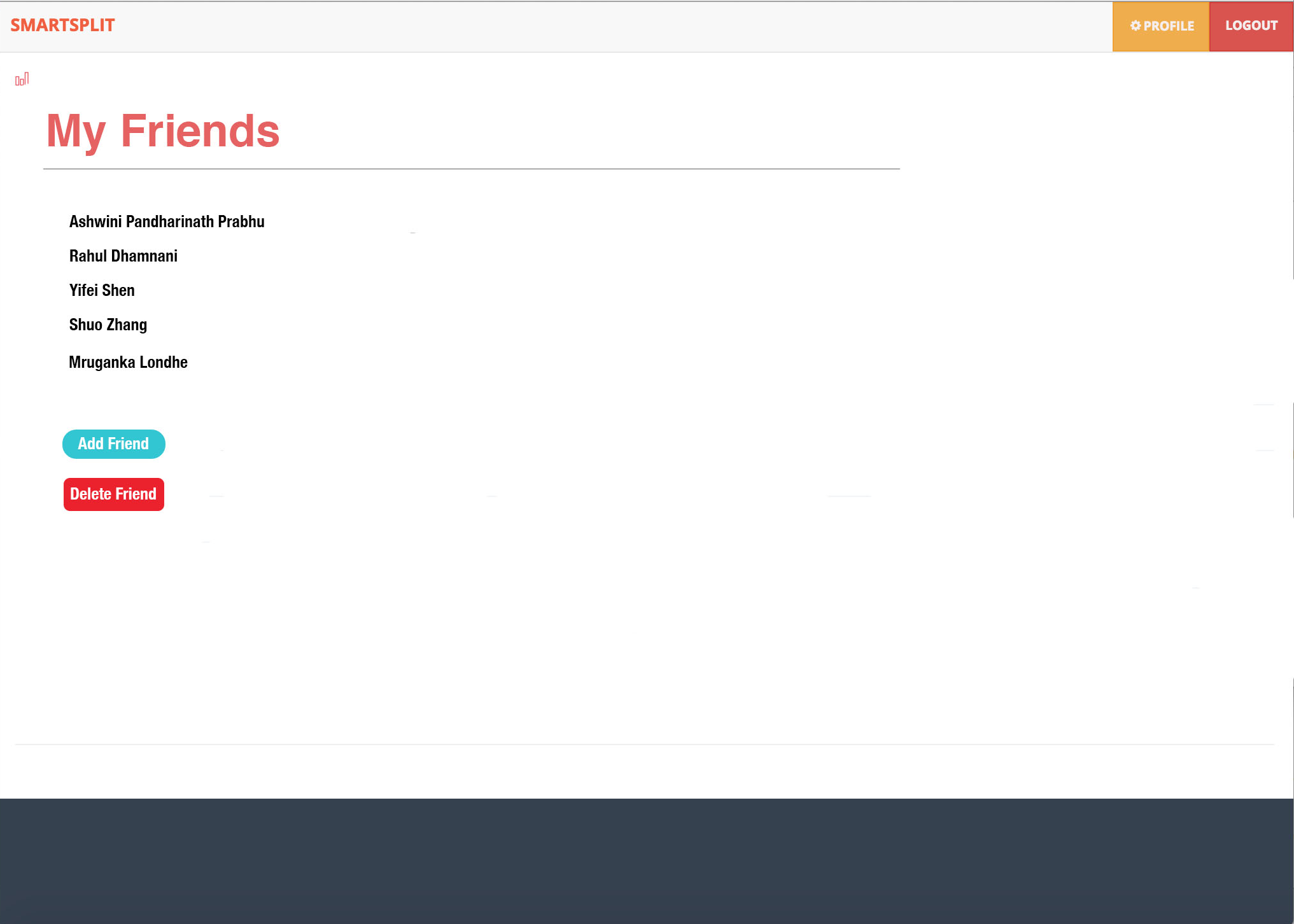
The hardest part is the development environment. For the project codes, we can share and work with Git without any trouble. However, for the development environment, like configuration, or data in the database, it is not possible to share these things. They need to be configured separately. When something changes, we need to notify everyone else in the group. Even though, it is still possible that something goes wrong.

## If you were to continue developing the application, what would be the next step?

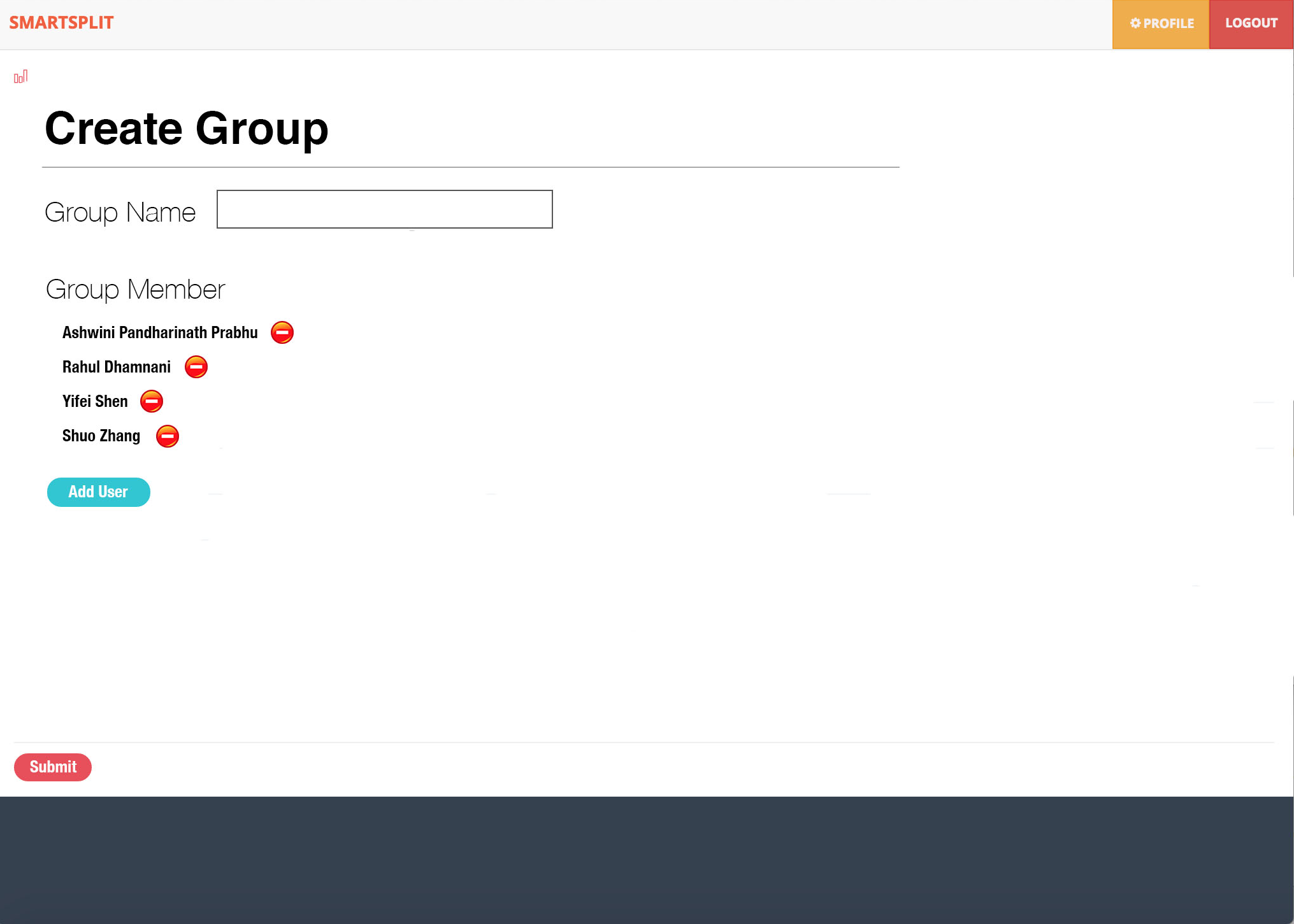
We would allow user to add people as friends for easy access. We would also allow users to create groups for sharing recurring expenses e.g. shared groceries. It will require the user to have people he wants to add in the friends’ list. We would allow users to login from Facebook and Twitter.

We are thinking of having multiple currency models. Currently we support only $. Also, we would provide a functionality to connect to the bank account or add a credit card to make the payment more easy and quick.

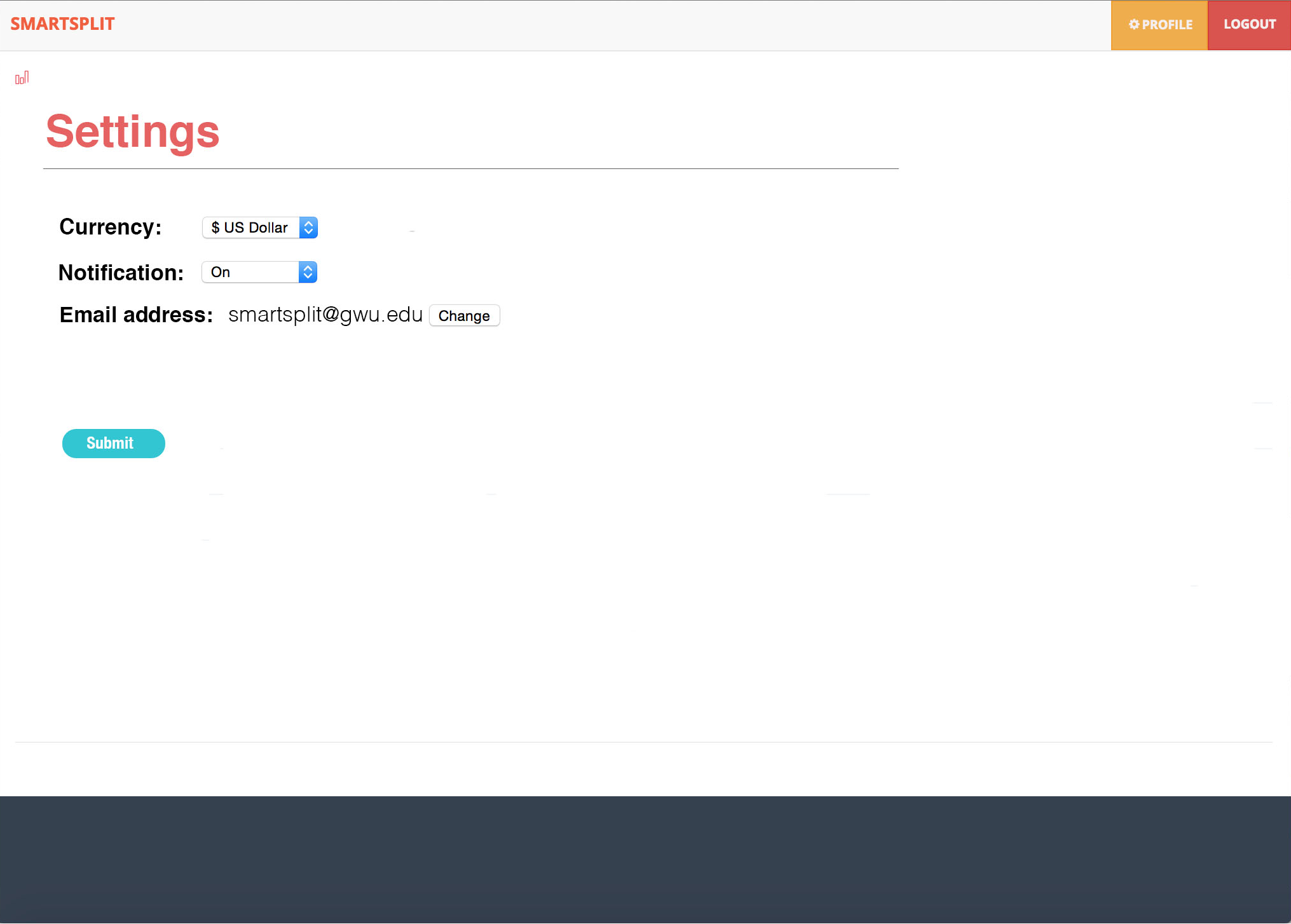
Following are the prototypes for some of the enhancements we have decided to add.



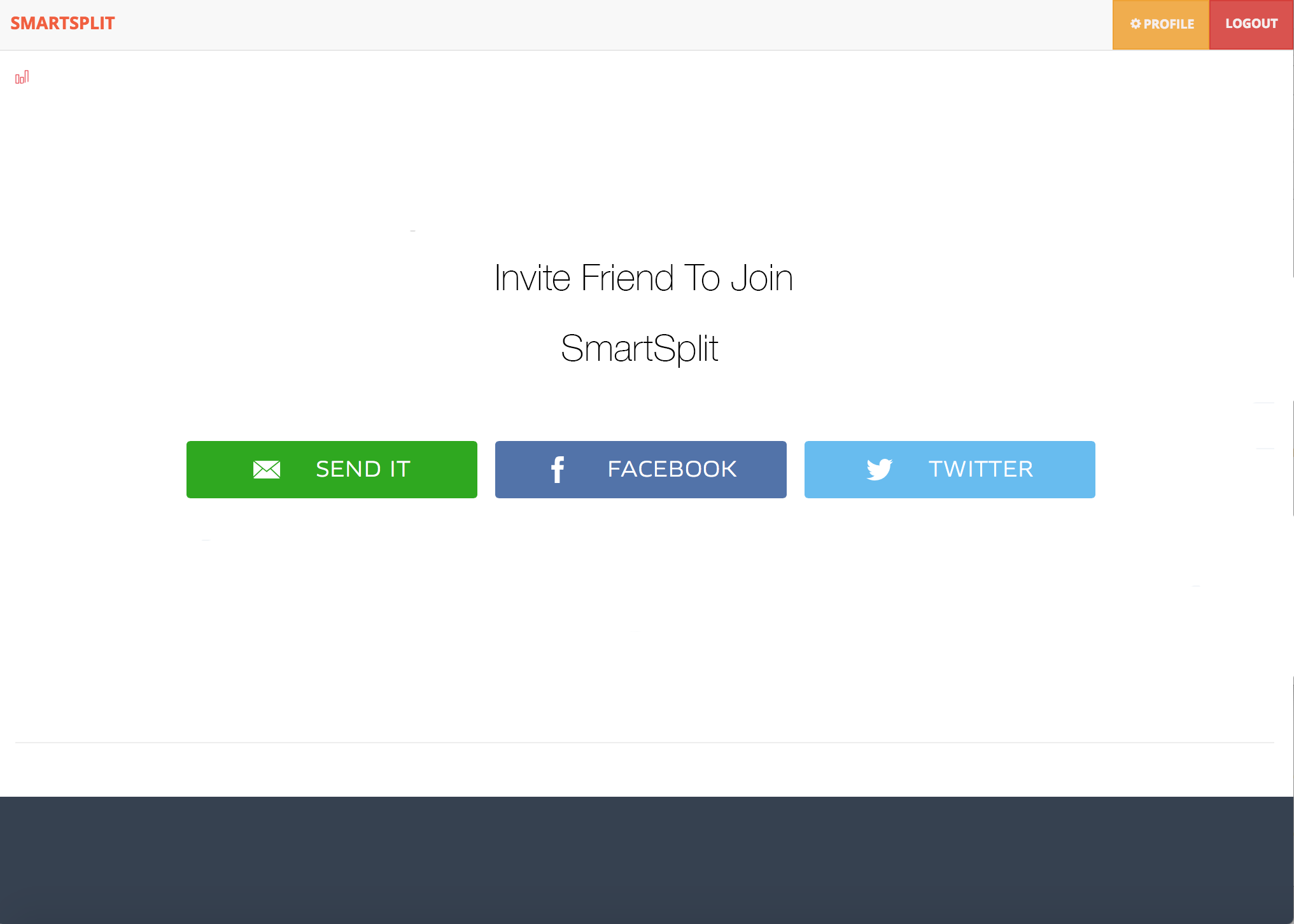
Add Friend Use case



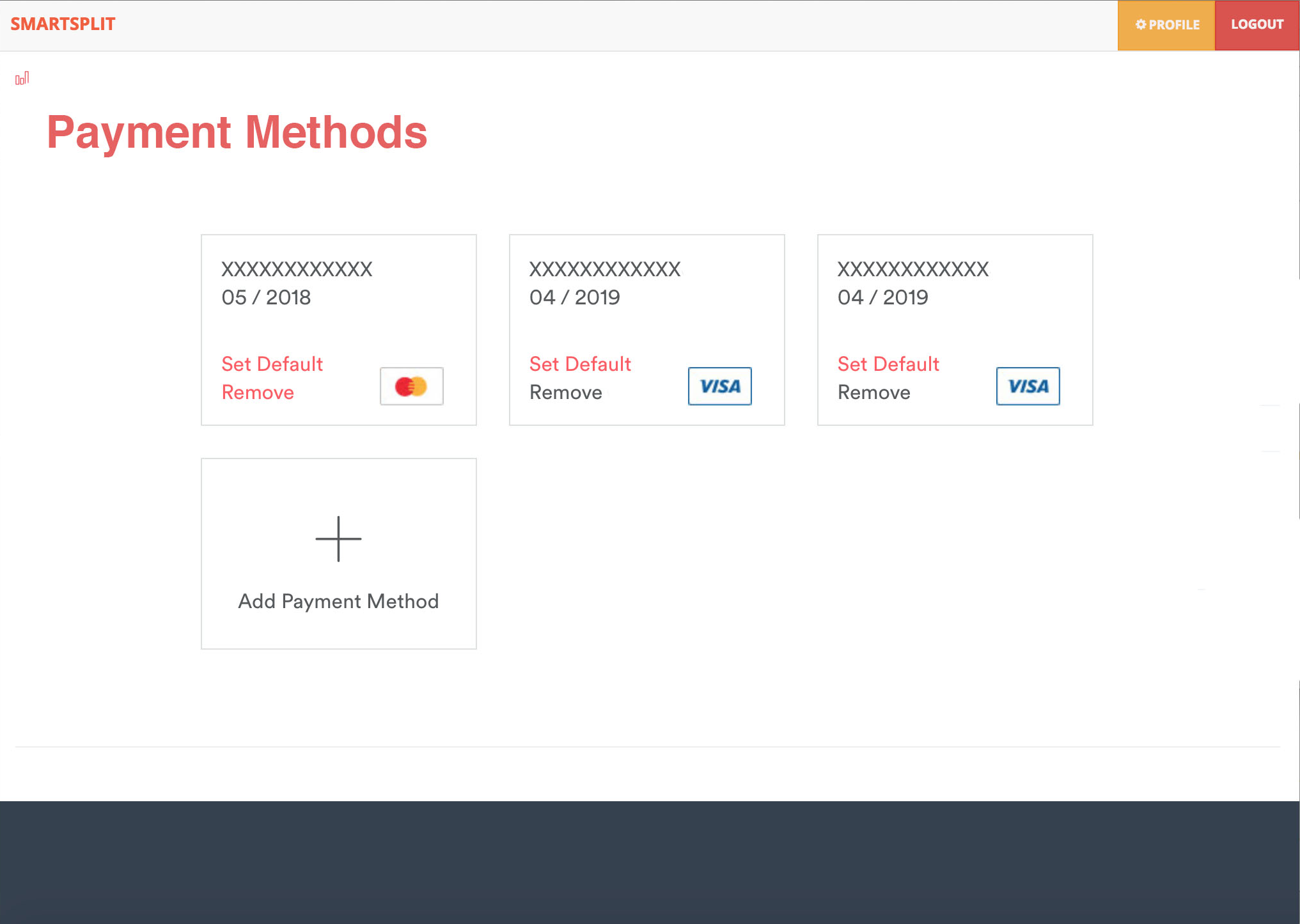
Create Group



Change Currency



Invite through other social media



Pay back using the application