**Functional Requirements Document of APT final project *Expense Splitter***

Chenguang Liu, Yanhan Zhang

Dec 2015

[Purpose](#h.ydylebhfpxto)

[Functional Components](#h.sgen4bv6encq)

[1. Expense Management Functionality](#h.1gstsckw2jh7)

[2. Task Planning Functionality](#h.j679ngxiogh0)

[3. Spending Analyzer](#h.n50ba9qwsn4)

[4. Payment Functionality](#h.mohaka9j3wap)

[5. Notepad Functionality](#h.ex3hjauoto0z)

[Front End Features](#h.hy684jnak1zg)

[Data Model](#h.n5gilwi30gd7)

# Purpose

*Expense Splitter* is an app aims to help people who live together to split their bills(e.g. utility) and living expenses (grocery/commodity costs) evenly and wisely. It also provides task planning, spending analyzing, and notepad tools to ease the users’ burden of their daily life.

# Functional Components

## 1. Expense Management Functionality

This component is in charge of expense tracking. An expense may consist of multiple spending items. Its main responsibility is letting users to create/update/remove expenses under their shared account(e.g. an apartment).

## 2. Task Planning Functionality

Users can use this component to create a shared task and complete a task. For example, a user can post a task saying he needs trash bags (and also specifies it). Later his roommate can complete this task and the amount of money spent will be split evenly among the related users.

## 3. Spending Analyzer

The spending analyzer gets activated when the status of shared expense in an account changes. Particularly, it may caused by someone completes a task or adds a new spending item within an expense.

## 4. Payment Functionality

Every month all the users under the same shared account need to settle down their own expenses at least once. This component takes the payment information (from which user to which user, and the amount of money) and uses the PayPal mobile API to transfer the money.

The user can also finish this process offline (pay by cash). The sender can create a cash payment and the receiver will need to acknowledge this payment.

## 5. Notepad Functionality

This component is like a mobile/web version of the old fridge notepads. However it differs from the paper ones in the sense that users can have the information of read receipts(i.e. When another user reads the note, everyone can know that he has read the message).

# Front End Features

1. ***User Registration***

A user uses his Google account to sign-in. The system should be able to create a local account which includes the followed information:

parameters: {user\_email, user\_nikename, profile\_image\_url, paypal\_account(optional)}

return null.

1. ***Create a Shared Account***

A user creates a shared account (apartment) and invites the participants to join.

parameters: {user\_email, account\_name, user\_emails[]}

return: {account\_id, account\_name}

1. ***Join a Shared Account***

A user joins a shared account.

parameters: {user\_email, account\_name}

return: success/fail

1. ***View a shared account***

A user can check the account to get information of the account, current spending of each participant, and how much this user owes to/by every other participant.

parameters: {user\_email, account\_id}

1. ***Create a new expense***

A user creates a new expense(of which only he can remove it). The items list will be initialized as an empty list. An expense has one **tag** attached.

parameter: {expense\_name, user\_email, account\_id, participant\_emails[], tag(optional), items\_list[]}

return: success{expense\_id}/ fail (participant or item information contain mistake)

1. ***Add participants(users) to an expense***

The expense creator can relate users under the same shared account to a specific expense.

parameters: {creator\_email, account\_id, participant\_emails[], expense\_id}

1. ***View an expense***

A participant is able to check an expense. The result contains the basic information about the expense and all the spending items under it.

parameters: {user\_email, expense\_id}

return: {expense\_id, expense\_name, participants\_email[], cost, items[]}

(Note: the items list should contain the complete item objects, and the cost is calculated from the costs of these items.)

1. ***Add an item to the expense***

A participant of a expense can add new spending items to an existing expense.

parameters: {user\_email, item\_name(str), cost(double), date}

return: status, item\_id

1. **Check the unsettled expenses (\*\*\* Current Expenses)**

Check all the unsettled expenses related to an user.

parameters: {user\_email, account\_id}

return: status, {amount\_to\_settle(money, double), expense\_list[]}

(Every expense contains all the item and their details)

1. **Check expenses history**

Check expense history for a period of time, returns the total money spent, the amount of money spent by him, and amount of money he owed to the others.

parameters: {user\_email, account\_id, start\_date, end\_date}

return: status, {}

1. **View spending trend**

A user can check the spending trend of a shared account.

parameters: {user\_email, account\_id, start\_date, end\_date} (default period: one month)

return: status, {total\_money\_spent(doube), details:{{tag1(str), money\_spent(double), percentage1}, {tag2(str), money\_spent(double), percentage2} ...}}

1. **Create a task**
2. **Complete a task**
3. **Create a note**
4. **Reply to a note**
5. **Check notepad**

# Data Model

1. Account

account\_id, account\_name, user\_email\_list(list of user\_emails), expense\_id\_list(list of expense\_ids), user\_spendings\_list(list of money(double) spent or paid by each user)

(user\_spendings\_list may not be even ever because not every expense is shared by all the participants)

1. Expense

expense\_id(id), expense\_name(str), account\_id(id), item\_list(list of SpedingItems), cost(double), is\_settled(boolean), user\_list[](list of user\_ids), amount\_to\_settle[](list of double, positive means he needs to pay to settle this expense)

1. SpendingItem

item\_id(id), item\_name, user\_id, cost, date

1. User

user\_id, user\_email, user\_nickname, user\_profile\_url, account\_id, paypal\_account

1. Payment

from\_user\_id, to\_user\_id, amount\_of\_money, date, acknowledged(boolean)

1. Task
2. Notepad
3. Note