**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. ***User Login:***

After the user passed the google authentication and presses the ‘enter’ button, it generates a login request to the server which returns request status, identifier-normal, the user information, shared account information, the amount of money owed by and owes to other users, and ongoing expenses(unsettled). If the user has not registered, it returns request status and an identifier-uninitialized. If the user has not joined any account, it returns request status and an identifier-.

Parameters: {user\_email}

Return values as described above.

1. ***Create an apartment (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 an apartment (Shared Account)***

A user joins a shared account.

parameters: {user\_email, account\_name}

return: success/fail

1. ***View an apartment (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**

Apartment members can create a task under a expense, picture and due date to be finished later.

1. **Complete a task**

Once an apartment members finish a task, the service will build a corresponding item in the same expense, and add it to the apartment.

1. **Create/edit a note**

Each apartment has a unique notepad, and apartment members can add note to this notepad. Each note is visible for all apartment members, however, only the note creator is allowed to edit it.

1. **Reply to a note**

Allow each apartment member reply to a note.

1. **Check notepad**

It can get all notes that is visible for the current users.

parameters: {user\_email, apt\_name}

return: status, {amount\_to\_settle(money, double), all\_notes\_list[author, descriptions, date]}

# 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
4. reply

# Scenario:

**1 Create a shared account() for a group of people:**

**step1: let each apartment create their own accounts**

Request: nick\_name, user\_email, bank\_account

HTTP POST: localhost:8090/createAccount

Response: status, user email

**step2: let one of the apartment member to create this shared account(apartment) with a roommates user list(not including the creator himself/herself):**

Request:

(required) apt\_name, user\_email, user\_email\_lst

(optional) apt\_photo

HTTP POST: localhost:8090/createAccount

Response: status, apt\_id

**step3: if you forgot to add some of your roommates to the apartment, or someone joins the apartment, you can add them one by one. You need to provide your email\_address and the new\_user email address. Also, the new user needs to be a registered user.**

Request:

apt\_name, user\_email, new\_user

HTTP POST: localhost:8090/createAccount

Response: status, apt\_id

**2 Plan an expense(create several tasks, and items):**

If you have already paid for sth, you can create an items under the expense, including information like total cost, sharer list and so on. Also, you can create a task which can be finished later by others.

**step1: created an expense under shared account**

Request:

expense\_name, user\_name, user\_email\_lst (not including the email of creator himself),

apt\_name

HTTP POST

Response:

status, expense\_id

**step2: create tasks:**

Request

required: expense\_id, user\_email, task\_name, user\_email\_lst, description

HTTP POST

optional: task\_photo

**step2: create items:**

Request:

required: expense\_name, buyer\_email, sharer\_email\_lst,total\_cost, item\_name

HTTP POST

optional: item\_photo

**3 Build, assign and finish a task**

Any apartment members can create a task under expense, and provide a list of users(not including himself/herself, he/she will be added to the list automaticall) who can be responsible for this task; this list of people are also the people who need to share the bill regarding this task. Then, any apartment member in this this can decide to do this task, and the person in charge can finish this task (the app will build an item automatically for you).

**step 1: create task:**

Request:

required: expense\_id user\_name task\_name user\_email\_lst description

optional: task\_photo

HTTP POST

Response: task\_id status

**step 2: Assign the task (any volunteer is willing to do this task)**

Request: task\_id, user\_email (person in charge)

HTTP GET

Response: status

**step3: Finish the task(the person in charge can finish the task once get work done), and he needs to provide the total\_cost for this task**

Request: task\_id, total\_cost

Response: status

**4 Check out**

Once the trip(expense) has been finished, there are two steps we need to do to checkout.

1. **checkout all/specific expense**

HTTP GET:

check all:

user\_name, apt\_name

check specific expense with expense name:

expense\_name, apt\_name,user\_name

response: status

1. **create payment for each user(the minimal number of transfer)**

one of the apartment member can request to calculate payment for all apartment members

HTTP GET

request:

apt\_name, user\_email

response: payment\_lst status

1. **Notes and comments**

Any apartment member can add notes to this shared account, and edit it later. All apartment members can reply to this note.

**1 Create a Note**

Any apartment can add note to the shared account(apartment)

HTTP POST

request: apt\_name, user\_email, description

response: note\_id, status

**2 Edit a Note**

The athor can edit his/her notes

HTTP POST

request:

note\_id, user\_email, new\_description

response: status

1. **Make a comment/Reply to a note**

Apartment members can reply to any notes under this shared account(apartment):

HTTP POST:

Request: user\_name, description, note\_id

Response: reply\_id, status