W4156 - Project Proposal

Project: Go Dutch
Project Team: Dutchers

Qianrui Zhao (qz2338)
Zhihao Li (zl2695)
Yushi Lu (yl3974)
Qian Zheng (qz2348)

Sep 24th, 2018
Revised: Oct 9th, 2018

Language: Python (OCR), Java (Android development), MySQL (database)
Platform: Android

1 Introduction

We would like to develop an Android mobile application named “Go Dutch”. Unlike many budget planning and expense recording applications on the current market, this application would be more focused on record, manage and maintain expense shared by multiple people, i.e. group bills. This application helps users to record expense by simply scanning receipts of purchase, and users are allowed to specify their own items with customized quantity. Besides, this application can automatically split common expense, including tips and taxes, among all involving consumers. If a shared purchase paid by one or multiple people, the money flow will be clear so that every involving consumer can realize how much they should pay for the payer(s) or received from others. Apart from this, group chat will be supported in this application. Once the sub-bills is worked out, invitations which invite consumers to check out those sub-bills will be automatically sent out to all group members as a gentle reminder to press for payment. We believe by using this application, users can be free from tedious and error-prone calculation process. As a result, making budget plans and maintaining expense-logging books will no longer be tedious and inefficient. Last but not least, some additional social functionalities, as listed below, would also be implemented to enhance the communication between users and the user experience as well.

The development process of this application involves two main parts: mobile side (Android with naive Java), and server side (OCR technology powered by Python). Additionally, we use MySQL to store data persistently (on server side). Due to the time limit, we may not be going to build these two parts from scratch, which means we may take advantage of some available open-source resources on Github or other third party platforms. Even so, we will try our best to keep the originality of this application.

2 User stories

1) Order
    a) **Creation:** As a lazy person who involves in group consumptions, I hope this app can recognize receipts by simply scanning so that users are free from the tedious typing process. My condition of satisfaction is that receipt in good condition (no destruction and rotation) should be recognized correctly. Once a receipt is accepted by this application, a corresponding order should be created. Each order should include the quantity and price of each item and the extra cost such as tax and tips, where the quantity of item should be modifiable for users to allocate their own items.
    b) **Tracking:** As a person who involves in many group consumptions, I want this application can be able to keep track of the payment process of each receipt by reminding receipt shared for unpaid purchase so that people will not be in the

embarrassing case where they totally forget to pay someone for the group consumption. My condition of satisfaction is that when receipt sharers complete their payment by external approaches (in cash or by card) they can mark this order as completed through this application, and the display of this order should have some status transition, e.g. the color turns to be gray. When everyone involved in an order completes the payment, the payer will receive a notification to ask for their confirmation. Once the payer confirms, the order can be closed down, otherwise, unpaid receipt sharers will constantly receive notifications to remind them.

c) **Display:** As a person who needs to review past orders, I hope not only the ongoing orders but also the past orders can be displayed in this application. My condition of satisfaction is that at least recent 10 orders should be display with style variations depending on their status (ongoing or completed).

2) Item allocation:
    a) **Countable items:** As a person involving a group purchase, I want this application can allow users to pick up their own countable items, for example, 3 out of 5 apples in this purchase belong to Amy, so that the expense of each person can be worked out easily. My condition of satisfaction is that users can be allowed to choose the number of items under the available amount. Once a receipt share has done and submit the allocation, the corresponding number of items should be deducted before showing to the next sharers. Once conflict happens, for example, Amy and Jane both specify 3 apples but the total is only 5, neither of allocations is accepted and they should do it again.
    b) **Uncountable items:** As a person involving a group purchase, I want this application can allow users to allocate their own uncountable item, such as flour, pizza and common fee like tax and tips. My condition of satisfaction is that the receipt payer can specify the number of shares so that users can specify how many shares they are assigned to. For instance, Amy and Jane bought a pizza divided by 3 pieces. Amy as the bill payer, specifies there are 3 pieces of this pizza. Amy picked up 2 of them and Jane picked up the rest. Once conflict happens, the same case as mentioned above is allowed.

3) Calculation:
    a) **Countable items:** As a receipt sharer, I want this app can figure out the expense of each person for their countable items so that everyone knows how much they should pay to the payer. My condition of satisfaction is shown in the following case. If Amy and I are in the same group and we have an unresolved receipt which was paid by Amy. This receipt includes an apple (2.5 dollars) I bought, a banana (1 dollar) Amy bought. This application should be able to work out I should pay 2.5 dollars to Amy.

b) **Uncountable items:** As a receipt sharer, I want this app to figure out the expense of each person for their uncountable item so that everyone knows how much they should pay to the payer. My condition of satisfaction clarifies in the following case. Amy, Jane and I bought a pizza, plus tax, ending up 9 dollars. This purchase is paid by Amy. After specifying the number of share we will take, the application works out Jane and I should pay 3 dollars each to Amy.

4) Additional functionalities
   a) **Multiple payers:** As a receipt sharer, I want this app can calculate the money flow between people when a purchase is paid by multiple persons so that we can get rid of this tedious and error-prone process of working this out manually. My condition of satisfaction is shown in the following case. Suppose there is a purchase involving 3 persons, Amy, Jane and I, where each of us bought 2, 3 and 1 banana(s) (1 dollar for each). After adding up 1.5 dollars tax to the total cost, the total expense of this purchase is 7.5 dollars paid by Amy (3 dollars) and Jane (4 dollars). I assume this application is capable of telling me that I should pay 0.5 dollars and 1.5 dollars to Amy and Jane respectively.
   b) Social features:
      i) As a social butterfly, I want to share my purchase with my friends online in real-time, so that the cost seems to be worthier. My conditions of satisfaction include posting the names and photos of the goods online.
      ii) As a social butterfly, I want to give my friends suggestions on extra-value goods or restaurants, so that I can help my friends to save meaningless costs. My conditions of satisfaction include sharing my ranks of the vendors online, and inquiring the best vendors of a specific category.

3 APIs:

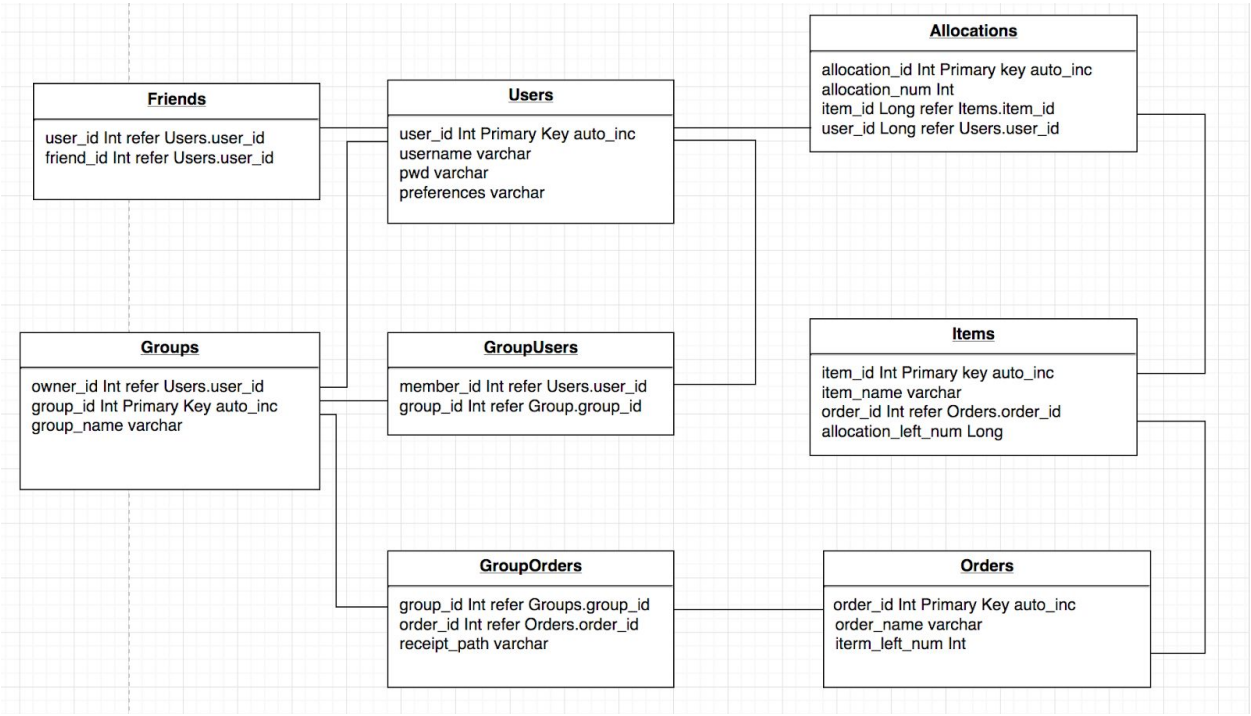
- **OCR: Tesseract**
  Tesseract is an open-source OCR API. It has Unicode (UTF-8) support, and can recognize more than 100 languages "out of the box".
  Tesseract supports various output formats: plain-text, hocr(html), pdf, tsv, invisible-text-only pdf.

- **Image Processing: OpenCV**
  OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license.

- **Chatting Groups**
  RivChat - Android chat app base on Firebase will be elaborated in Chapter 6 Prototype Design.

4 **Database** (draft overview)

- User table (User id [P], username, password, preferences)
- Friend table (User id [P], friend id)
- Group table (Group id [P], owner id)
- Group-User table (Group id, user id)
- Group-Order (Group id, order id, receipt image)
- Allocations (Allocation id [P], Item id, quantity, user id)
- Items (Item id [P], Order id, quantity)
- Orders (Order id [P], number)
  [P] for primary key



5 Acceptance Tests

Users					
User stories	**Roles**	**Explanation**	**Input**	**Output (Pass)**	**Output (Fail)**
2.1.a	Receipt payer	Receipt Importing	Capture a photo of the receipt or upload one from the phone	The entries on the receipt, together with their quantities and prices, are shown on the screen.	The app can't successfully recognize the entries or presents the imprecise results.

			gallery.		
2.1.a	Every User	Budget Management	Enter the budget management interface.	The statistics of recent income and expenses are shown in a diagram.	The statistics are shown incorrectly.
2.1.b	Receipt payer	Tracking	The receipt sharers pay the receipt payer by a certain **external approach** and tap the "confirm payment" button.	The transaction is shown as "completed" in the app and the receipt payer will receive a payment confirmation notification.	The transaction isn't shown as "completed" or the receipt payer doesn't receive the notification.
2.1.b	Receipt sharer	Tracking	One of receipt sharer has not paid to the payer for a long time.	A notification to remind this user should be pushed for a reasonable interval.	No notification appears or it shows up at an annoying pace.
2.1.c	Receipt Payer & Receipt Sharer	Receipt Management	Open the receipt management interface	Receipt history are shown in the interface ordered by a certain rule (time, status, etc.)	Not all receipts are shown or receipts are shown in incorrect order.
2.2	Receipt Payer & Receipt Sharer	Item Claiming	Tap the "Get 1" button several times at a certain entry.	The item is labeled as selected with a specified quantity, and the total price is updated regarding the new item.	The item can't be selected or the total price is incorrect.
2.2	Receipt Payer & Receipt Sharer	Item Claiming	#(User specifies) > # (Remaining)	A prompt should turn up and the action fails.	The action succeeds or the system crashes.
2.2.a	Receipt Payer & Receipt Sharer	Share Quantity Specifying (Countable)	Specify the amount of each countable item.	The quantities of each item can be specified, and are updated in the interface of receipt sharers. As mentioned in 2.2.a, Amy should be able to specify 3 apples, while the rest sharers can only specify 5 - 3 = 2 apples in total.	The quantities are mistakenly shown in the receipt sharers' interfaces. Amy is not able to collect a specified amount of apples, or the amount of all specified apples is larger than 5.

2.2.b	Receipt Payer	Share Quantity Specifying (Uncountable)	Specify the shares of uncountable items.	The number of shares of uncountable items is recalculated and shown correctly in receipt sharers' interfaces. As mentioned in 2.2.b, Amy (the payer) should be able to set up three shares of the pizza, and any one of us is able to collect one of the shares.	The uncountable item can't be divided by the number of shares, nor mistakenly shown. Amy is not able to set up the amount of shares of the pizza, or the rest of us can't claim ourselves shares.
2.3.a	Receipt Payer & Receipt Sharer	Calculation	Specify the amount of each countable item.	The application figures out the total expense for each person correctly. In the case mentioned in 2.3.a, $ 2.5 will be displayed in my receipt, while $1 in Amy's, indicating that I should pay $2.5 to Amy.	Either wrong results appear on Amy's or my receipt.
2.3.b	Receipt Payer & Receipt Sharer	Calculation	Specify the shares of uncountable items.	The application figures out the total expense for each person correctly. In the example mentioned in 2.3.b, $3 will be shown on each of our receipts.	Wrong results come out on any of our receipts.

Database				
Table	**Explanation**	**Input**	**Output (Pass)**	**Output (Fail)**
Allocation	Item quantity deduction	Item A with the specified quantity is assigned to one user	The specified quantity of item A is deducted from the total quantity.	Wrong quantity is deducted by, or deducted from the quantity of any other wrong item.
	Item quantity less than 0	Quantity (remaining) - Quantity (specified by user) < 0	An error prompt is shown and the quantity of this item remains unchanged.	The operation is allowed to carry out and the quantity of this item goes below zero.
	Item release	Item A with certain quantity	The specified quantity should be added to the	The quantity of A remains unchanged, or the specified

		assigned to a user now is released.	remaining quantity of item A.	quantity is added to the quantity of any other item.
Friend	Adding friends	User A adds User B as a friend	In the database, a corresponding entry specified by A and B are friend should be added.	The corresponding entry is not inserted, or a wrong entry is inserted.
Group	Creating groups	A user creates a group and adds specified members as group members.	A corresponding entry specified the group id and its creator is generated and inserted to the Group table.	The corresponding entry is not inserted, or a wrong entry is inserted.
Group-User	Creating groups	A user creates a group and adds specified members as group members.	Corresponding entries the relation between the group and group members are inserted to the Group-User table.	Nothing happens or wrong entries generated.
Order	Order creation	A receipt is scanned.	A receipt is assigned to a unique order id and entries with correctly specified each item and its quantity should be created in this table.	Nothing happens or wrong entries generated.
Allocation	Order creation		At the same time, the corresponding entries mentioned in the upper cell should appear in the Allocation table as well.	Nothing happens or wrong entries generated.
Group-Order	Order creation		An entry specified the group which the order belongs to and the order ID should be inserted in this table.	Nothing happens or wrong entries generated.

6 Prototype Design

6.1 Android Framework

Our work is based on this chatting application where we can take advantage of some functionalities with little modification but we still have to implement many functionalities to support the features of our application.

Features it already has:

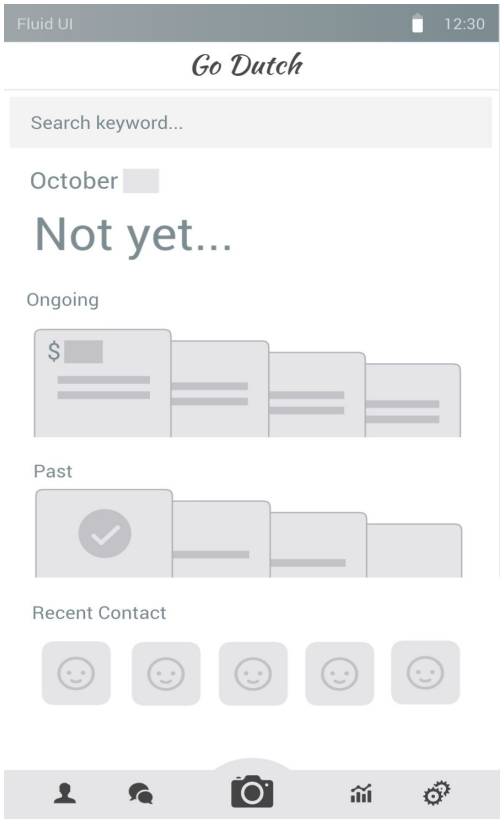
Feature description	Details
User registration, sign-in, and sign-out	1. Allow user register a new account with username and password. 2. Users must sign in with their registered username and password to use this app. 3. Connect to Firebase service to reset users' password. 4. Change avatar image, edit username.
Chat, and group chat	1. Add friends, remove friends, and send text messages. 2. Create and delete groups, and rename group name. 3. Add and remove group members.

Features it doesn't have or don't fit our requirements:

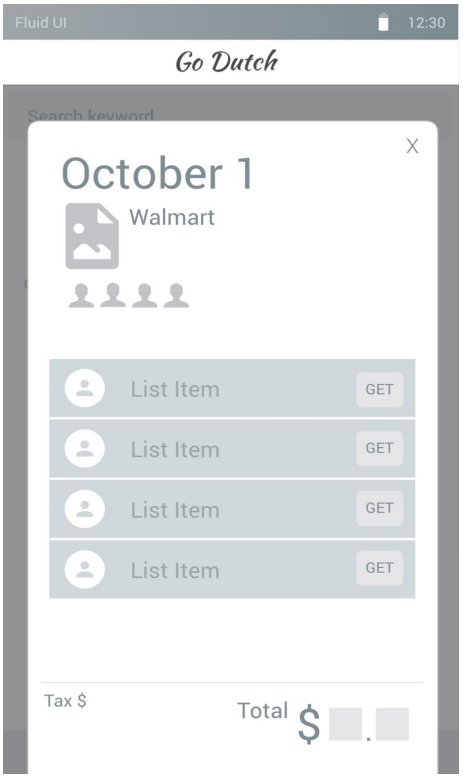
Problem description	Addition or modifications
Abnormal photos	1. Backend function: add image processing function in our server to process irregular images 2. Push notification to users if the server can't parse the image.
Item unclaimed	Push notifications if a user doesn't complete an order in an interval of 6 or 8 hours.
Insufficient database	1. To maintain the connections between user-and-user and user-and-order, we need extra tables such as User, Friend, Group, Group-User, and Group-Order. 2. We also need tables such as Items, Orders, and Allocations in our server to keep track of orders, users'

	items and other data to support features of our application (See section 4 for details). 3. Our database should be robust against problematic and erroneous user inputs.
Unsuitable layout	Redesign the UI for our features. (See section 6.2 for details)

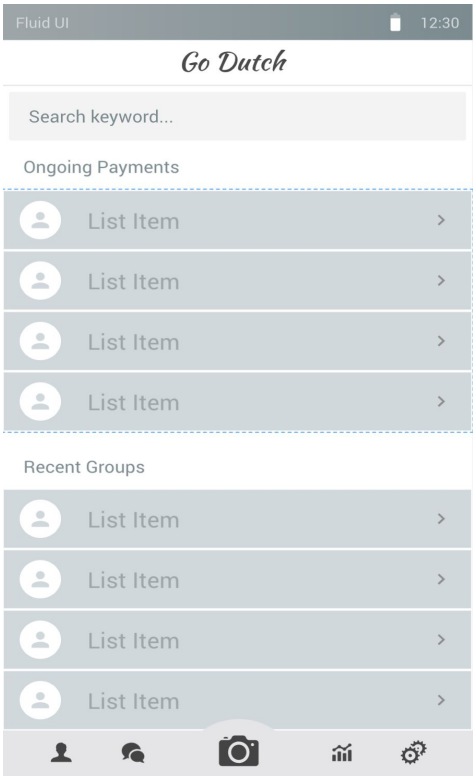
6.2 User Interface (preview)



Main Interface



Receipt Importing Interface



Group Chatting Interface

6.3 Interfaces provided by server-side

- Database

1. User registration

```
{
        Request: {
              Username: String
              Password: String
              Email: String
        }
        Respond: {
              Status: bool
              User id: Integer
        }
  }
```

2. Receipt Creation

```
{
      Request: {
```

```
            Receipt: {
                  Title: String
                  Total price: Double
                  Owner(user id): Integer
                  Items: [
                        Name: String
                        Quantity: Integer
                        Price: Double
]
            }
      }
      Response: {
            Status: bool
            Receipt id: Integer
      }
}
```

3. Receipt Deletion

```
{
      Request: {
            Receipt id: Integer
            User id: Integer (Check if is owner)
      }
      Response: {
            Status: bool
      }
}
```

4. Group Creation

```
{
      Request: {
            Users: [
                  User id: Integer
            ]
      }
      Response: {
            Status: bool
            Group id: Integer
      }
}
```

5. Group User Insertion

```
{
	Request: {
		Group id: Integer
		User id: Integer
	}
	Response: {
		Status: bool
	}
}
```

6. Group User Deletion

```
{
	Request: {
		Group id: Integer
		User id: Integer
	}
	Response: {
		Status: bool
	}
}
```

7. Receipt Request

```
{
	Request: {
		Receipt id: Integer
	}
	Response: {
		Status: bool
			Receipt: {
				Title: String
				Total price: Double
				Owner(user id): Integer
				Items: [
					Item id: Integer
					Name: String
					Quantity: Integer
					Price: Double
				]
```

```
                }
        }
}
```

8. Allocation Request

```
{
      Request: {
            Receipt id: Integer
            User id: Integer
            Item: {
                  Item id: Integer
                  Name :  String
                  Quantity: Integer
                  Price: Double
            }
      }
      Response: {
             Status: bool
      }
}
```

9. Allocation Cancel

```
{
      Request: {
            Receipt id: Integer
            User id: Integer
            Item: {
                  Item id: Integer
                  Name :  String
                  Quantity: Integer
                  Price: Double
            }
      }
      Response: {
            Status: bool
      }
}
```

10. Receipt Update

```
{
```

```
      Request: {
            receipt id: Integer
      }
      Response: {
             status: bool
      }
}
```

- OCR Interface

1. Receipt Recognition

```
{
      Request: {
            Image: image file
      }

      Respond: {
            Status: bool
            Receipt: {
                  Title: String
                  Total price: Double
                  Owner(user id): Integer
                  Items: [
                        Name: String
                        Quantity: Integer
                        Price: Double
                  ]
            }
      }
}
```