# W4156 Final Iteration Report

Team: Dutchers

## 0 Our team

**Team name:** Dutchers

**Team members:**

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

**IA:** Sara Samuel

## 1 Use Cases

### 1.1 User-related

#### 1.1.1 Signing Up

**Actor:** Anyone using our app

**Description:** For anyone who wants to have a try on our app, including the bill-payer or non-bill-payer, the first step is to sign up.

**Basic Flow:**

1. **Action:** User inputs his/her username and password, both of which should only contain lowercase and uppercase alphabetical characters, numbers and ~~special characters of '@' and '_', and a limited number of characters.~~ Special characters of '@', '.' and '_', and at most 30 characters, then presses submit button.
   **Response:** On pressing submit button, system will store the valid user information in the database and prompt hint for the user that he/she has successfully registered an account.

**Alternative Flow:**

1. **Action:** If the username already exists in the database, and a user register again.
   **Response:** On pressing submit button, system will prompt hint that the username already exists and can not register again.
2. **Action:** If the user try to conduct SQL Injection, like input the command "' or 1 =1  union drop table User --", which will delete the table User in our system.
   **Response:** System should sanitize the input before executing command. System should not believe user. After sanitizing, illegal input will be filtered and prompt corresponding hint. For example, after deleting the apostrophe, spaces, equality sign and hyphen of "' or 1 =1  union drop table User --", it

would be “or11uniondroptableUser”, which is a valid username since it is just a normal string.

### 1.1.2 Signing In

**Actor:** Any user with an account
**Description:** For any user who already has an account, he/she can login to his/her own account.
**Basic Flow:**

1. **Action:** User inputs his/her username and password.
   **Response:** System checks whether the combination of username and password is valid. If the combination is invalid, prompt hint “wrong username or password”. System should NOT prompt further hint for the user, just “wrong username or password” is enough. Otherwise, if the combination of username and password is valid, the user is successfully logged in.

**Alternative Flow:**

1. **Action:** User try to avoid login authentication using SQL Injection, like “‘ or 1 =1 --”.
   **Response:** Sanitizing user input before execution.

### 1.1.3 Searching and Adding Friend

**Actor:** Anyone who has logged in the app
**Description:** User has logged onto the app, and wish to find other users and add them.
**Basic Flow:**

1. **Action:** Type in the name of another user and press “search”.
   **Response:** If the target user is in our database, presented the user to the finder. Else, prompts hint that the target user is not in the database.
2. **Action:** Add the found user by pressing the “add” button
   **Response:** Pressing the button “add” to add the selected user, the system will relate the selected user and the current user as a friend.
3. **Action:** User wants to experience partial searching when finding friends, i.e., search ‘tom’ will get all the result like ‘tommm’, ‘ttttttttom’ and ‘totototototom’.
   **Response:** Implemented partial search, thus user inputs part of his/her friend’s name then gets all other users that contain that string as substring.

**Alternative Flow:**

1. **Action:** User try to avoid login authentication using SQL Injection, like “‘ or 1 =1 --”, or even try to delete the table in the database.
   **Response:** Sanitizing the input from user and prompting corresponding hint. For example, after deleting spaces, hyphen, equality sign and apostrophe in “‘ or 1 = 1 --”, it will be “or11”, which will be a valid username.

### 1.1.4 Creating Group

**Actor:** A bill-payer

**Description:** A bill-payer select some of his/her friends and want to create a group.

**Basic Flow:**

1. ~~**Action:** User selects some of his/her friends in the friend list, press the "create group" button.~~
   ~~**Response:** System will create a new group entry in the database. The group has group name, owner and members.~~
2. **Action:** User presses Create Group button, inputs the group name and selects some of his/her friend as group members. User can also create empty group by not selecting any of his/her friends.
   Response: System inserts group information into gGroup Table.

**Alternate Flow:**

1. **Action:** User may try to exploit group name since it's an input item. User may use techniques like SQL Injection, or XSS since it will send POST to server.
   **Response:** Group name should only contain alphabetical and numerical characters, and NO special characters allowed.

## 1.2 Receipt-related

### 1.2.1 Uploading receipts

**Actor:** A bill-payer

**Description:** The consumer has just paid for a bill in either a restaurant or a supermarket and needs to record or divide it with friends.

**Basic Flow:**

1. **Action:** After logging in, press the floating action button on the homepage
   **Response:** A list of buttons is unfolded by the pressing, involving creating a new receipt button and refreshing button.
2. **Action:** Press the "New Receipt" button
   **Response:** A dialog will be popped up to ask the user whether taking a photo of the receipt or selecting one from the gallery.
   a. **Action:** Take a new photo
      **Response:** Navigate to the camera page.
   b. **Action:** Select one from the gallery.
      **Response:** Navigate to the gallery page.
3. **Action:** Wait until "Go Dutch" finish processing the image.
   **Response:** Show a spinning circle with dark background while processing the image. After successfully loaded, a modal containing detailed information of the receipt (item names, item price, and the total price) will be shown as an overlay.

4. **Action:** Modify names of items if anything wrong in the detailed information form, ~~and add other bill-sharers from the friend list,~~
   **Response:** New names take place of previous names if changes are made.
5. **Action:** Choose a group of friends to share the receipt with in the receipt overlay.
   **Response**: The members of the group will be shown below the group icons, and the total price will be recalculated according to the new number of sharers. If no group is selected to share the receipt, there won't be a "Split" option for the receipt items.
6. **Action:** Specify ~~the number of shares of items~~ how items in this receipt split across sharers (either on average or completely owned by the payer) ~~and add other bill-sharers from the friend list~~
   **Response**: The total price change as the way of receipt being split
7. **Action:** Press the "Submit" button to submit the ~~form~~ receipt.
   **Response**:
   a. All the members in the selected group will receive a notification for this receipt to notify their payment
   b. This confirmed-by-payer receipt will be displayed as a historical record on the history pane and allows ~~users~~ payer to review and edit before its payment is confirmed by all sharers.
   c. All information of this receipt to the server to store in the database.
   d. After pressing the "Submit" button, the receipt overlay will become invisible ~~Make the overlay invisible~~

8. **Action:** Press the "Cancel" button to close down the receipt
   **Response**: The receipt overlay will be shut down but the receipt will be displayed in the receipt list which is editable by the payer afterward.

### 1.2.2 Paying for bills

**Actor:** A bill-sharer
**Description:** A bill-sharer receives a notification of a newly generated bill which waits to be paid to the bill payer(s). Once a receipt is shared by the bill-sharer, the receipt will be displayed in the receipt list of all involving users.

**Basic Flow:**

1. **Action:** Either press on the notification or select it from the receipt list can open receipts to see the details of receipts.
   **Response:** Data is fetched from the remote database, and an overlay will be shown to present the details of the selected receipt and let the bill-sharer specify their own items.

2. ~~**Action:** Select the number of shares of items specified by the bill-sharer on the overlay.~~
   ~~**Response:** The total price of items will be calculated and updated~~
3.
   a. **Action:** Press the "~~Confirm~~Pay" button shown at the bottom of the overlay to confirm the payment (Payment operates ~~offline~~ online) if a sharer agrees with this purchase and this way to split this receipt
   b. Action: Press the "Challenge" button to challenge this receipt or the way to split this receipt.

   **Response:**

   a.
      - The amount of unpaid money and the number of sharers who have not confirmed their payment shown in the receipt list should change as sharers confirm their payment.
      - If all sharers confirm their payment by pressing the "Pay" button, this receipt will be shown in the "past receipts" section and no longer to be revisable by the payer.
      - If all sharers confirm their payment, the payer will receive a notification to notify the completion of this receipt.
      - Once a receipt is done, the database will be updated

   b.
      - If challenges happen, the payer may update the way he/she split the receipt. If the update is saved, all existing confirmation is reset and shares who have confirmed their payment should do it again.

   ~~Request will be sent to the server and doing some check against the remaining amount of items stored in the database.~~
   - ~~If every item has the sufficient amount remaining, the confirmation is successful and a prompt will be shown up to inform the bill sharer. The database will be updated at the same time. Once the payment is confirmed by the bill sharer, the bill payer (creator) will receive a notification of the confirmation. Then the bill payer can double confirm that the payment has been done. After that, the receipt will be marked as "finished" in the interface, and the database will be updated in the meanwhile.~~
   - ~~If one of the items has not enough amount remaining, the confirmation is unsuccessful and a prompt will also appear to inform users. Nothing update in the database.~~

### 1.2.3 Modifying details of receipts

**Actor:** A bill-payer

**Description:** After ~~creating~~ a receipt is confirmed by the payer (generator), the receipt generator wants to edit some details of this receipt. ~~manually modify some information of the receipt.~~

**Basic Flow:**

1. **Action:** Press on a generated receipt in the list of receipts.
   **Response:** A receipt overlay ~~with revisable information~~ will be popped up.
2. **Action:**
   a. Modify the name ~~information~~ of items ~~the receipt and~~ pressed the "Save" button
   b. Modify the way to split the purchase and pressed the "Save" button
   **Response:**
   a. New names will take place of previous names from then on. No notification will send to sharers.
   b.
      - The recalculated total price and other modified information will be updated in the interface as the payer changes.
      - All changes will be saved after pressing "Save" button and will be displayed in the receipt afterward
      - Notifications will be sent to all sharers and all existing confirmation will be unchecked.
3. ~~**Action:** Press the "save" button.~~
   ~~**Response:** The overlay disappears and new data are sent to the database in the server.~~

## 2 Test Suite

Test scripts can be accessed via this readme file on our git repo.

### 2.1 User-related

#### 2.1.1 Signing Up and Signing In

The circumstances for signing up and signing in are similar, thus they will have similar valid and invalid equivalent partition(will point out their differences).

- **Valid Equivalent Partition:** The valid partition contains of username and password is that the username and password only contains lowercase and uppercase alphabetical characters, numbers and special characters of '@' and '_', and a limited number of characters special characters of '@' '_' and '.', and at most 30 characters.
- **Invalid Equivalent Partition:** There are some kinds of invalid partition:
  - Username or password containing Illegal characters like '#', '<' or '&'.
  - ~~Username or password is longer than 16 characters.~~
  - Username or password is longer than 30 characters.
  - (Signing In Only) Username does not exist.

    - (Signing Up Only) Username already exists.
    - Username or password contains commands that may cause security issue.
- **Boundary Condition:** There will be boundary case as the length of username or password increases from 0 to >16. If the length is 0, system should prompt NULL INPUT error. If the length is within 1-16 range, it's valid in length. If the length exceeds 16, system should prompt LONG INPUT error.

| Test Case | Input | Output |
|---|---|---|
| Valid EP* | Username: 'tom'<br>Password: 'mot' | Prompt hint "signing up successfully" or "log in successfully" |
| 1st invalid EP | Username: '<tom>'<br>Password: '<mot>' | (1) Deleting these invalid characters in the server before progressing.<br>Or<br>(2) Prompt hint "wrong username or password". |
| ~~2nd invalid EP~~ | ~~Username: 'ttttttttttttttttttttttttom'~~<br>~~Password: 'mmmmmmmmmot'~~ | ~~Prompt hint "wrong username or password"~~ |
| 2nd invalid EP | Username/Password: 'ttttttttttttttttttttttttttttttttttttttttttttttttom' | Prompt hint "wrong username or password" |
| 3rd invalid EP | Username: 'tom', but 'tom' does not exist in the database during signing in. | Prompt hint "wrong username or password". It's a bad idea to prompt hint like "Username does not exist", because attacker can test what usernames are in the database using this hint. |
| 4th invalid EP | Username: 'tom', but 'tom' is a registered username during signing up. | Promt hint "username exists". |
| 5th invalid EP | Username: "' or 1 =1 --"<br>Password: "' or 1 =1 --" | Sanitizing the input in the server. After sanitizing, the input would be "or11". Prompt corresponding hint for input "or11" |
| Boundary condition | Username: '' | Prompt hint "NULL INPUT"` |

| | | |
|---|---|---|
| ~~Boundary condition~~ | ~~Username: "thereare15words"~~ | ~~Valid input, no further hint~~ |
| Boundary condition | Username: "there_are_30_words_30_words @@@" | Valid input, no further hint |
| ~~Boundary condition~~ | ~~Username: "@thereare16words"~~ | ~~Valid input, no further hint~~ |
| Boundary condition | Username: "there_are_31_words_31_words @@@@" | Invalid input |
| ~~Boundary condition~~ | ~~Username: "@@thereare17words"~~ | ~~Invalid input, prompt hint "LONG INPUT"~~ |

*EP: Equivalent partition

### 2.1.2 Searching and Adding Friend

The valid partition of this subroutine is when the friend exists in the database. If the user has already added the friend, there will be no adding function. Otherwise, the user can choose to add the search result as friend.

- **Valid Equivalent Partition:**
  - ~~Find friend but not add him/her~~
  - ~~Find friend and already added him/her~~
  - Search friend's full name without adding him/her
  - Search friend's partial name without adding him/her
  - Search friend's full name and add him/her
- **Invalid Equivalent Partition:**
  - Friend not exists in the database
  - Friend name contains some commands that may cause security issue
- **Boundary Condition:** There will be no boundary condition in this case. Or the boundary case is similar to that in **2.1.1 Signing Up and Signing In** part, but it's meaningless to discuss it again.

| Test Case | Input | Output |
|---|---|---|
| ~~1st valid EP~~ | ~~Search Friend: 'tom' Press "Add" button~~ | ~~List 'tom' or all the user that contains 'tom', like 'tommy'(will implement it in the next iteration), and after pressing "Add", system will relate the user and "tom" as friend~~ |
| ~~2nd valid EP~~ | ~~Search Friend: 'tom'~~ | ~~List 'tom' or all the user that~~ |

| | | |
|---|---|---|
| | | ~~contains 'tom'.~~ |
| 1st valid EP | Search friend 'tommy', suppose there is a tommy in database | List 'tommy' |
| 2nd valid EP | Search friend 'tom' suppose there is a tommy in database | List 'tommy' |
| 3rd valid EP | Search friend 'tommy' and press adding button | List 'tommy' and add him as your friend |
| 1st invalid EP | Search Friend: 'tom' | Prompt hint "Friend not exists", Show Nothing |
| 2nd invalid EP | Search Friend: "' or 1 =1 --" | Sanitizing the input in the server. After sanitizing, the input would be "or11". Prompt corresponding hint for input "or11" |

**2.1.3 Creating groups**

In our design, every group name should be unique, which means users are only able to create a group if the group name has never been used before. When creating a new group, users are able to select members from the list of friends. Users can also choose to create an empty group and add more friends later.

- **Valid Equivalent Partition:**
  - Create an empty group with an unused name
  - Create a new group with an unused name
- **Invalid Equivalent Partition:**
  - Create a group -with a name which is used before
  - Create a group without a name
  - Create a group with group name longer than 16 characters
  - Create a group with group name containing special characters
- **Boundary Condition:** The input group string should be in length 1 to 100. A group with length 0 or > 100 is invalid.

| Test Case | Input | Output |
|---|---|---|
| 1st valid EP | Group name: "new group" Selected friend ids: [5, 19, 26] | The "new group" name has never been used before so the server create a new group, adding group owner and three members into this group. Return group id of the new |

| | | |
|---|---|---|
| | | group. |
| 2nd valid EP | Group name: “new group 2”<br>Selected friend ids: [] | The “new group” name has never been used before so the server create a new group. Add only group owner into this group. Return group id of the new group. |
| 1st invalid EP | Group name: “new group”<br>Selected friend ids: [] | The group name is used before, therefore, server return -1. |
| 2nd invalid EP | Group name: “”<br>Selected friend ids: [] | The group name is with length 0, server return -1. |
| 3rd invalid EP | Group name: ‘TomIsAGoodBoyAnyGirlLovesHim’ | System will trim the group name to 16 characters to avoid security issues |
| 4th invalid EP | Group name: ‘I_am_handsome’ | System will eliminate the underscore to a valid group name: ‘Iamhandsome’ |

**2.1.4 Adding more member into an existing group**

User are provided with a list of friends’ name, and able to pick 0 or more members from the list. Then click submit button to commit the change.

- **Valid Equivalent Partition:**
  - Select 0 person from the list and save the change
  - Select 1 person from the list and save the change
  - Select > 1 people from the list and save the change
  - Select more than 1 people from the list and some of them are already in the group
- **Invalid Equivalent Partition:**
  - A user is only able to choose a member from the given friend list.
- **Boundary Condition:** Selected number of people from 0 to the number of total friends.

| Test Case | Input | Output |
|---|---|---|
| 1st valid EP | Select 0 person from friend list. | “true”, add no member into the group. |
| 2nd valid EP | Select 1 person from friend list. | “true”, add the specified member into the group. |
| 3rd valid EP | Select > 1 people from friend | “true”, add all selected |

| | | |
|---|---|---|
| | list. | members into the group. |
| 4th valid EP | Select more than 1 people from friend list and some of them are already in the group. | “true”, add only selected members who are not in the group into the group. |

## 2.2 Receipt-related

### 2.2.1 UI component tests:

For the following use cases in this section, the Valid Equivalent Partition and Invalid Equivalent Partition of them are the same but for a different testing component. For brevity, a general template will be introduced first and customized testing component for each use case will be articulated in the following table.

- **Valid Equivalent Partition:** User presses the button specified in the *Testing component* column
- **Invalid Equivalent Partition:** User presses somewhere else rather than the intended button specified in the *Testing component* column

Secondly, the boundary condition is not applicable for those use case. In addition, the test schema for tests designed for these use cases includes testing if the intended event happens when users operate as expected and also testing if the event does not happen when users perform unexpectedly.

| # Use case | Testing component | The intended result when under VEP* |
|---|---|---|
| 1.2.1.1 | The floating action button on the homepage | A list of buttons is unfolded |
| 1.2.1.2 | “Create new receipt” button | A modal containing two buttons specified in 1.2.1.2 shows up |
| 1.2.1.2.1 | “Take a new photo” button | Navigation to the camera page allowing users to take a picture |
| 1.2.1.2.2 | “Choose from gallery” button | Navigation to the gallery page allowing users to choose a receipt from the gallery |
| 1.2.2.1 | ~~“Notification”~~ Receipt Modal | A modal where the details of the receipt will be shown up upon the receipt list |
| 1.2.2.1 | Select a receipt from the | An overlay where the details of the |

| | | |
|---|---|---|
| | receipt list | receipt will be shown up upon the receipt list |
| 1.2.2.3 | “Confirm” button | Either a *successful* prompt or an *unsuccessful* prompt will be displayed and the modal will be shut down |
| 1.2.2.3 | “Cancel” button | The modal will be shut down |

*VEP: Valid equivalent partition

### 2.2.2 Item claiming tests

Corresponding to the use cases, an overlaying modal will be popped up to let the user specify some items. The valid equivalent partition, as well as the invalid one, will be provided down below.

- **Valid Equivalent Partition:** Users press either the “Split” or “All” buttons ~~increase or decrease button~~ to specify their own items. ~~valid amounts of items, which are bounded by the boundary values set by the bill-payer, and then press the “confirm” button.~~
- **Invalid Equivalent Partition:** ~~Users would like to specify more items even the upper boundary has been reached or quit before any item is specified.~~

| # Use case | Test description | Boundary condition |
|---|---|---|
| 1.2.1.1 - 1.2.1.3 | UI rendering tests | |
| 1.2.1.4 | Test if the names of items are editable by the payer | An empty string (“”) is not allowed |
| 1.2.1.5 | Test if icons representing members of a group shown below the group icon | |
| 1.2.1.5 | Test if the total price displayed at the bottom changes as the number of members changes by selecting different groups | |
| 1.2.1.5 | Test if both “Split” and “All” button showed if a group is selected for each item | If no group selected, only “All” button is displayed for each item |
| 1.2.1.6 | Test if the total price changes as specifying items. | |

| | | |
|---|---|---|
| | 1. The total price should increase by the correct amount if the button is selected from “Split” to “All” of each item.<br>2. The total price should decrease by the correct amount if the button is selected from “Split” to “All” | |
| 1.2.1.7.a | Test if notification of this receipt generation is sent to all sharers | If no group is selected, no notification is sent. |
| 1.2.1.7.b | Test if this confirmed-by-payer receipt is displayed on the history pane | |
| 1.2.1.7.b | Test if this receipt is editable by the payer before all sharers pay | |
| 1.2.1.7.d | Test if the modal (receipt) is invisible after pressing the “Submit” button | |
| 1.2.1.8 | Test if the modal (receipt) is invisible after pressing the “Cancel” button | |
| 1.2.1.8 | Test if the receipt displayed on the history pane and is editable even though the payer did not confirm it | |

| # Use case | Test description | Boundary condition |
|---|---|---|
| 1.2.2.1 | Test if pressing on the notification can open the corresponding receipt modal with its detailed information (all items, total price etc.) | |
| 1.2.2.3 | Test if the modal is shut down after pressing the “Pay” button | |
| 1.2.2.3.a | Test if the amount of amount which has not been paid and the number of sharers who have confirmed their payment change shown on the receipt list after a sharer presses | |

| | | |
|---|---|---|
| | the “Pay” button | |
| 1.2.2.3.a | Test if the bill payer receives a notification to notify shares’ payment | |
| 1.2.2.3.a | Test if a receipt is shown in the “past receipts” section instead of “onging receipts” after all sharers confirm their payment | |
| 1.2.2.3.b | Test if a notification of challenge is sent to the payer | |
| 1.2.2.3.b | Test if a payer update the way to split receipt, all confirmation is unchecked and all relevant details (the amount of unpaid money etc.) are reset. | |

| # Use case | Test description | Boundary condition |
|---|---|---|
| 1.2.3.2.a | Test if new names of items shown in the place of the previous ones when the receipt is open after the modification | |
| 1.2.3.2.b | Test if total price updates with the change | An empty string (“”) is not allowed |
| 1.2.3.2.b | Test if all changes are saved | |
| 1.2.3.2.b | Test if notification sent to all sharers and same as 1.2.2.3.b | |

| ~~# Use case~~ | ~~Test description~~ | ~~Boundary condition~~ |
|---|---|---|
| ~~1.2.2.2~~ | ~~Test if pressing on the notification can open the corresponding receipt modal with its detailed information (all items, total price etc.)~~ | |

| | | |
|---|---|---|
| ~~1.2.2.3~~ | ~~Test if there are sufficient items left for each one specified by a user involving in a bill, a *successful* prompt should be displayed, and the number of corresponding items should be deducted at the same time.~~ | ~~The race condition may happen when multiple users involved since the total amount of items they allocate may exceed the total amount available. If it is the case, none of the requests from involved users would be executed, which means the database remains unchanged.~~ |
| ~~1.2.2.3~~ | ~~Test if there are insufficient items left for each one specified by a user involving in a bill, a *unsuccessful* prompt should be displayed, and the number of items involved should be retained.~~ | |