Second Iteration

Join Me: An Event Management Platform

Advanced Software Engineering Project Proposal

Team: MissingOne
Jiayi Li(jl4924), Chi Zhang(cz2465), Chengyun Yu(cy2468)

1. Revise your user stories to reflect what is now fully implemented. Expand each user story into a *use case*. Each use case should include at least title, actor(s), description, basic flow and alternate flows (when applicable).

User Case 1:

Title

Post Event and Management I

Actors

The event hosts of our system.

Description (User Story)

This user case mainly aims at testing post event logic of our system and also the group email & extract information functions in the post event logic.

User Story

"As an outdoor enthusiast, I want to invite my friends to go cycling in the Central Park on this weekend. However, all of my friends can't come because of some personal issues. In this case, I want to find someone else to cycle with me, so that I post my cycling route and schedule on Join Me.

My conditions of satisfaction of this case are that:

- I want my events to be attractive, so that I should be able to post not only plain words but also pictures or even videos on the billboard.
- I want to inform all attendees in time rather than post public messages on the platform, the platform should allow me to send emails to all attendees when I want to send some notices or updated information.
- I hope I can connect with each attendee personally. In Join Me, I can download all attendees' personal information in form of Excel file."

Flow of Events

1. The actor: Click "Sign in with Google" button on the welcome page.
2. The system: Open a link on the browser for users to login in the system with their Google account.
    a.
        i. The system: If the system finds this is the first time that a user uses our system, it will redirect to the sign up page.

ii. The actor: Input all his personal information needed on that page, such as the profile image, the real name, the nickname, the gender, the address, the email address,t he tags, the self description, etc and then click “save” button.
iii. The system: Store the personal information into the database and redirect to the Main page of our system.

b.

i. The system: If the system finds this is not the first time that a user uses our system, it will redirect to the main page directly.

3. The actor: Click the “Post Event” button on the Main page.
4. The system: Redirect to the Post Event page.
5. The actor: Input all the event information needed on that page, such as the title of the event, the brief description, the address the event will hold at, the city, the tags of the event, the event date, the register time period, event images, etc and click the “Save” button on the sign on page.
6. The system: Save the event information to the database and redirect to the Main page.
7. The actor: To manage the event his/her hosted, he/she can click the “See More” button in the “Event Hosted” block or click the Event Title button directly in that block if there’s a match.
8. The system: Redirect to the Host Event Display page, which will show the detailed information of the event as well as some management functions.
9. The actor: Input the content of the group email into the input box at the bottom of the page and click “send” button.
10. The system: Send the email to all the attendees of this event.
11. The actor: Click “Extract Information” button on the page.
12. The system: Output the all attendees’ personal information in the form of a csv. file.

User Case 2:

Title

Join Events

Actors

Users who wish to find events to join in.

Description

This user case covers our system’s ability to let users join their desired events.

User Story

“As a normal user, I want to find some interesting activities on weekends and make some new friends. So that, I open *Join Me* and try to find some events by typing some keywords.

My conditions of satisfaction of this case are that:

- I want to find the most related and popular events that match with my searching keywords.
- I want to check the profile of the organizer to have an overview of the host and connect with the host to get more detail information about the involving activity.
- I want to register in the event when I’m interested in it. “

Flow of Events

1. The actor clicks “Sign in with Google” button on the welcome page.
2. The system opens a link on the browser for users to login in the system with their Google account.
    a.
        i. The system finds this is the first time that a user uses our system, it will redirect to the sign up page.
        ii. The actor inputs all his personal information needed on that page, such as the profile image, the real name, the nickname, the gender, the address, the email address,t he tags, the self description, etc and then clicks “save” button.
        iii. The system stores the personal information into the database and redirect to the Main page of our system.
    b.
        i. The system finds this is not the first time that a user uses our system, it will redirect to the main page directly.
3. The system displays the main page.
    a.
        i. The user chooses constraints on the filter bar on top of the main page and clicks the “Search” button.
        ii. The system displays the top 15 events that satisfies the filter.
    b.
        i. The user does nothing.
        ii. The system displays the top 15 events that share the same tag with the user.
4. The user clicks on an event displayed on the main page.
5. The system displays the details of the event and the host on a new page.
6. The user clicks on the profile image of the host.
7. The system displays the information of the host on a new page.

8. The user enters a message in the input box and presses the “Contact Host” button.
9. The system sends a email containing the message to the host.
10. The user clicks on the “Join” button.
11. The system stores the user’s attendance of the event and displays his or her profile image in the “Attendee” section.

User Case 3:

Title

Post Event and Management II

Actors

The event hosts of our system.

Description (User Story)

This user case mainly aims at testing the kick function and individuals functions in post event logic.

User Story

“As a graduate student in Columbia, I want to find some other students who have registered in the same class as me to discuss some questions after class or the teammates for the group projects. So that I post my requirements on Join Me.

My conditions of satisfaction of this case are that:

- I want to find the students who have registered in the same class as me. As a result, I need to communicate with each attendee after I get their personal information. And I hope I can kick off the attendees who are not matched with my demand.”

Flow of Events

1. The actor: Click “Sign in with Google” button on the welcome page.
2. The system: Open a link on the browser for users to login in the system with their Google account.
    a.
        i. The system: If the system finds this is the first time that a user uses our system, it will redirect to the sign up page.
        ii. The actor: Input all his personal information needed on that page, such as the profile image, the real name, the nickname, the gender, the address, the email address,t he tags, the self description, etc and then click “save” button.
        iii. The system: Store the personal information into the database and redirect to the Main page of our system.
    b.

    i. The system: If the system finds this is not the first time that a user uses our system, it will redirect to the main page directly.
3. The actor: Click the “Post Event” button on the Main page.
4. The system: Redirect to the Post Event page.
5. The actor: Input all the event information needed on that page, such as the title of the event, the brief description, the address the event will hold at, the city, the tags of the event, the event date, the register time period, event images, etc and click the “Save” button on the sign on page.
6. The system: Save the event information to the database and redirect to the Main page.
7. The actor: To manage the event his/her hosted, he/she can click the “See More” button in the “Event Hosted” block or click the Event Title button directly in that block if there’s a match.
8. The system: Redirect to the Host Event Display page, which will show the detailed information of the event as well as some management functions.
9. The actor: Click the “Edit” button on the page.
10. The system: Redirect to the Host Event Edit page of the system, where the actor can edit the event information.
11. The actor: Click the profile images of the attendees in the “Attendee” block.
12. The system: Delete the clicked attendee and update the database.
13. The actor: Click the “Save” button on the page.
14. The system: Save all the changes on the page, update the database and redirect to the Host Event Display page.
15. The actor: Input the content of the group email into the input box at the bottom of the page and also input the target attendee’s nickname in the input box above. Then click the “send” button.
16. The system: Send the email to the target attendee.

2. For your test plan, explain the equivalence partitions and boundary conditions necessary to unit-test each of the major subroutines in your system (functions, procedures, methods, etc. excluding getters/setters and helpers), with references to your specific test case(s) addressing each equivalence partition and each boundary condition.

We have two major classes, named *UserController* and *EventController*, that would handle *user* object and *event* object separately. The major subroutines in these two classes are add and delete *user* instance and *event* instance.

UserController:

1. **Test add_user:**
    - Valid:
      Add a new user and check if return ‘SUCCESS’
    - Invalid:
      Add the same user again and check if return ‘DUPLICATE’

2. **Test delete_user:**
    - Valid:
      Add a new user and then delete it, check if return ‘SUCCESS’
    - Invalid:
      Delete the same user again, as there is no such user in the database, but the SQL command will execute correctly, check if return ‘SUCCESS’

3. **Test edit_user:**
    - Valid:
        a. Add a new user and then retrieve this user, change the nickname of this user, check if edit_user could handle it without throwing exception, then, retrieve the edited user and check if the nickname is changed correctly
        b. Change the feature location in the same way and check it
    - Invalid:
      Edit the user without change any feature, check if return ‘MISSING’

4. **Test retrieve_user:**
    - Valid:
      Add a new user and retrieve and check if return an User instance
    - Invalid:
      Retrieve an user with invalid information and check if return ‘MISSING’

EventController:

1. **Test add_event:**
    - Valid:
      Add an nonexistent event and check if return valid event id
    - Invalid:
      Add the event with the same features again, because the database set event id as auto-increase value, add the same event again will also return valid event id. Check if return valid event id

2. **Test retrieve_event:**
    - Valid:

        Retrieve a valid event_id like 1, and check if return an event instance
    - Invalid:

        Retrieve an invalid event_id -1, and check if return 'MISSING'

3. **Test edit_event:**
    - Valid:
        a. Add a new event, and change the feature image of it. Then, edit this event check if return 'SUCCESS'
        b. Change the feature image of that event to the original value, and edit it, check if return 'SUCCESS'
    - Invalid:

        Edit one event without changing any value of features, check if return 'MISSING'

3. describe how you measured the branch coverage achieved by your test suite.

In the branch coverage part we use the *coverage.py* library.
This tool can be installed by using

```
pip install coverage
```

To measure the branch coverage of my test unit, using the command

```
coverage run unit_test.py
```

Then we can extract the report by using either

```
coverage report
```

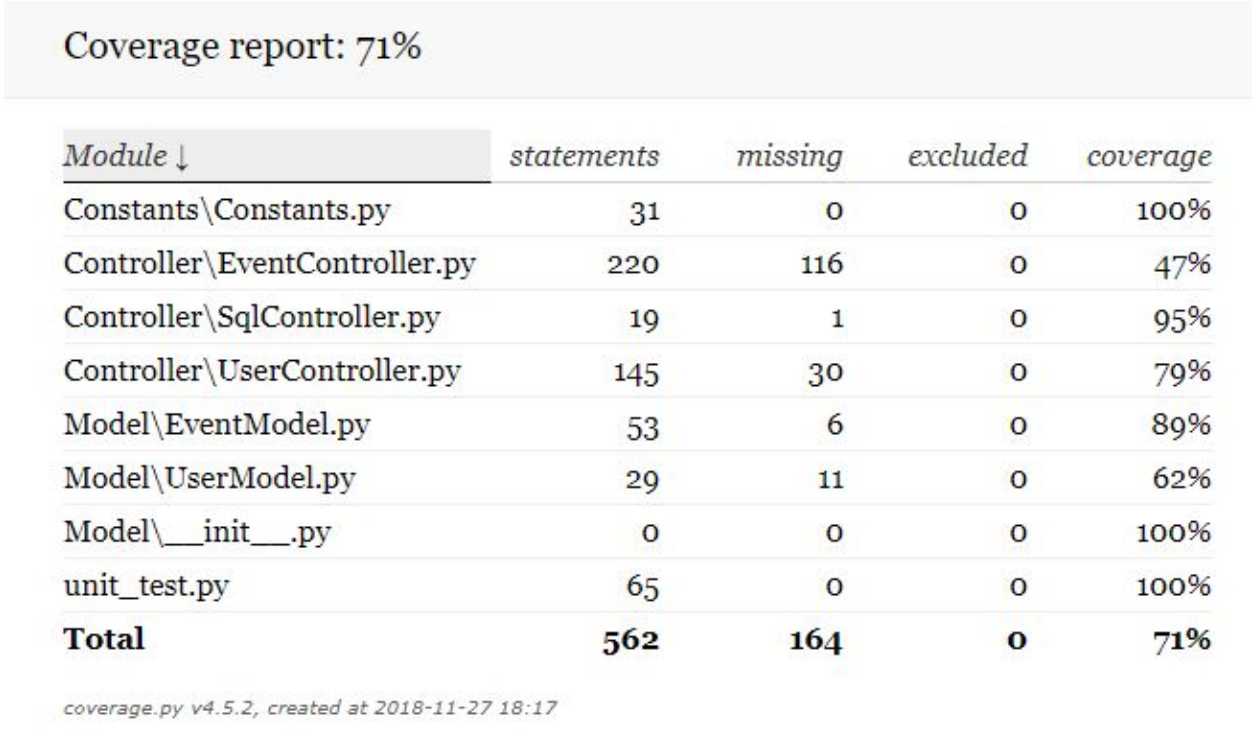
Which will print the report on the terminal windows:

```
Name                                Stmts   Miss  Cover
-------------------------------------------------------
Constants\Constants.py                 31      0   100%
Controller\EventController.py         220    116    47%
Controller\SqlController.py            19      1    95%
Controller\UserController.py          145     30    79%
Model\EventModel.py                    53      6    89%
Model\UserModel.py                     29     11    62%
Model\__init__.py                       0      0   100%
unit_test.py                           65      0   100%
-------------------------------------------------------
TOTAL                                 562    164    71%
```

Or

`coverage html`

Which will generate annotated HTML listings with coverage results:



Coverage report: 71%

Module ↓	statements	missing	excluded	coverage
Constants\Constants.py	31	0	0	100%
Controller\EventController.py	220	116	0	47%
Controller\SqlController.py	19	1	0	95%
Controller\UserController.py	145	30	0	79%
Model\EventModel.py	53	6	0	89%
Model\UserModel.py	29	11	0	62%
Model__init__.py	0	0	0	100%
unit_test.py	65	0	0	100%
Total	**562**	**164**	**0**	**71%**

coverage.py v4.5.2, created at 2018-11-27 18:17

The total branch coverage of our test suite is 71%, and we choose to generate HTML report in our post-commit CI process, all the report files are stored in the `./report` with the report of our unit test.

4. link to the github repository:

https://github.com/Debug1995/JoinMe