YesOK - T5: Second Iteration

Minxuan Gao (mg4115) Ruiguang Li (rl3090) Donghao Qiu (dq2146) Xinyan Zhang (xz2878)

Part 1:

User story 1
As an event organizer, I want to ***post events*** at a certain location during a certain time and get a sharable link (for different apps) so that people could see the event and join.
My conditions of satisfaction are:
1) I can log in
2) I can create an event and enter the event details.
3) I can get a link for the event that I have created.

User story 2
As an event organizer, I want to ***update or delete*** an event I have created so that I could let participants know the latest situation.
My conditions of satisfaction are:
1) I can log in
2) I have created an event
3) I can find the event detail page
4) I can delete the event or edit some details

User story 3
As a User I want to see all ***events nearby*** and filter the events based on distance, time and category so that I can pick an event I am interested in.
My conditions of satisfaction are:
1) I can share my location
2) I can see my events nearby
3) I can filter the events according to my needs

User story 4
As a User I want to see all events I have ***attended or organized*** and get sharable link so that I could refer them to my friends
My conditions of satisfaction are:
1) I can log in
2) I have attended or hosted at least one event.
3) I can see my past events

User story 5
As a User I want to save the events that I am interested in to the ***favourites list*** so that I can go back to them and decide whether I want to join these events.
My conditions of satisfaction are:
1) I can log in
2) I can save the event that I am interested in

User story 6
As a User, I can leave ***comments*** to the events that I am interested in.
My conditions of satisfaction are:

1) I can log in
2) I can comment on the events and see comments from other users.

User story 7

As a User, I can ***participate*** in an event and get a ***notification*** email.
My conditions of satisfaction are:

1) I can log in
2) I can register an event
3) I will receive a notification email containing the event link.

User story 8

As a User, I can see all the ***participants*** of an event so that I can know who is going to attend the same event with me.
My conditions of satisfaction are:

1) I can log in
2) I can see a participants list on the event detail page.

User story 9

As an event organizer, I can receive ***notifications*** related to the events I host.
My conditions of satisfaction are:

1) I can log in
2) I can create an event
3) I will receive notifications when other users comment or register my events.

Part 2:

Backend Test Plan:
Code can be found here:
https://github.com/ybbbby/ADV-SE-Collaboration/tree/master/backend/tests
Unit Test:
Comment:
create_comment()

- Description: create a comment for an event
- Test plan: Test whether or not the comment length is below 300, and test whether or not user and event exists
- Equivalence partitions (Test cases):
    - Valid: user and event exists, comment length good (test_add_comment_1)
    - Invalid:
        - user and event exists, comment length too long (test_add_comment_2)
        - user doesn’t exist (test_add_comment_3)
        - event doesn’t exist (test_add_comment_4)

get_comment_by_event()

- Description: get all comments of an event
- Test plan: Test whether or not the input event exists
- Equivalence partitions (Test cases):
    - Valid: event exists (test_get_comment_by_event_1)
    - Invalid: event doesn’t exist (test_get_comment_by_event_2)

delete_comment()

- Description: delete a specific comment given its id
- Test plan: Test whether or not the input comment exists
- Equivalence partitions (Test cases):
    - Valid: comment exists (test_delete_comment_1)
    - Invalid: comment doesn't exist (test_delete_comment_2)

Join:

create_join()

- Description: create a join representing a user joining an event
- Test plan: Test whether or not user and event exists
- Equivalence partitions (Test cases):
    - Valid: user and event exists (test_create_join_1)
    - Invalid:
        - user exists, event doesn't exist (test_create_join_2)
        - user doesn't exist, event exists (test_create_join_3)
        - user and event don't exist (test_create_join_4)

get_join_by_user()

- Description: get all joined (user, event) pair of a specific user
- Test plan: Test whether or not the user exists
- Equivalence partitions (Test cases):
    - Valid: user exists (test_get_join_by_user_1)
    - Invalid: user doesn't exist (test_get_join_by_user_2)

get_join_by_event()

- Description: get all joined (user, event) pair of a specific event
- Test plan: Test whether or not the event exists
- Equivalence partitions (Test cases):
    - Valid: event exists (test_get_join_by_event_1)
    - Invalid: event doesn't exist (test_get_join_by_event_2)

user_is_attend()

- Description: check whether or not a user joined an event
- Test plan: Test whether or not user and event exists
- Equivalence partitions (Test cases):
    - Valid: user and event exists (test_user_is_attend_1)
    - Invalid:
        - user exists, event doesn't exist (test_user_is_attend_2)
        - user doesn't exist, event exists (test_user_is_attend_3)
        - user and event don't exist (test_user_is_attend_4)

delete_join()

- Description: delete a join relation which is a (user, event) pair
- Test plan: Test whether or not user and event exists
- Equivalence partitions (Test cases):
    - Valid: user and event exists (test_delete_join_1)
    - Invalid:

        - user exists, event doesn’t exist (test_delete_join_2)
        - user doesn’t exist, event exists (test_delete_join_3)
        - user and event don’t exist (test_delete_join_4)

Like:
create_like()
- Description: create a like relation which is a (user, event) pair
- Test plan: Test whether or not user and event exists
- Equivalence partitions (Test cases):
    - Valid: user and event exists (test_create_like_1)
    - Invalid:
        - user exists, event doesn’t exist (test_create_like_2)
        - user doesn’t exist, event exists (test_create_like_3)
        - user and event don’t exist (test_create_like_4)

get_like_by_user()
- Description: get all liked (user, event) pair of a specific user
- Test plan: Test whether or not the user exists
- Equivalence partitions (Test cases):
    - Valid: user exists (test_get_like_by_user_1)
    - Invalid: user doesn’t exist (test_get_like_by_user_2)

get_like_by_event()
- Description: get all liked (user, event) pair of a specific event
- Test plan: Test whether or not the event exists
- Equivalence partitions (Test cases):
    - Valid: event exists (test_get_like_by_event_1)
    - Invalid: event doesn’t exist (test_get_like_by_event_2)

exist()
- Description: check whether or not like relation which is a (user, event) pair exists
- Test plan: Test whether or not user and event exists
- Equivalence partitions (Test cases):
    - Valid: user and event exists (test_exist_1)
    - Invalid:
        - user exists, event doesn’t exist (test_exist_2)
        - user doesn’t exist, event exists (test_exist_3)
        - user and event don’t exist (test_exist_4)

delete_like()
- Description: delete a like relation which is a (user, event) pair
- Test plan: Test whether or not user and event exists
- Equivalence partitions (Test cases):
    - Valid: user and event exists (test_delete_like_1)
    - Invalid:
        - user exists, event doesn’t exist (test_delete_like_2)
        - user doesn’t exist, event exists (test_delete_like_3)
        - user and event don’t exist (test_delete_like_4)

Events:
create_event()

- Description: create an event and return the event id.
- Equivalence partitions (Test cases):
    - Event name:
        - Valid: length of event name is equal or less than 50.
        - Boundary: length of event name is equal to 50.
        - Invalid: length of event name is larger than 50.
    - Address:
        - Valid: length of address is equal or less than 200.
        - Boundary: length of address is equal to 200.
        - Invalid: length of address is larger than 200.
    - Description:
        - Valid: length of description is equal or less than 600.
        - Boundary: length of description is equal to 600.
        - Invalid: length of description is larger than 600.
    - Image:
        - Valid: length of image is equal or less than 200.
        - Boundary: length of image is equal to 200.
        - Invalid: length of image is larger than 200.
    - Category
        - Valid: length of category is equal or less than 20.
        - Boundary: length of category is equal to 20.
        - Invalid: length of category is larger than 20.
    - User
        - Valid: the given user exists
        - Invalid: the given user does not exist in database
- Test plan:
    - Test1: create an event with all valid values (test_create_event_1)
    - Test2: create an event with all valid values except that the user is invalid (test_create_event_2)
    - Test3 Boundary: create an event with all valid values, the length of event name is 50 (test_create_event_3_boundary)
    - Test3: create an event with all valid values except that the length of event name is invalid (test_create_event_3)
    - Test4 Boundary: create an event with all valid values, the length of event address is 200 (test_create_event_4_boundary)
    - Test4: create an event with all valid values except that the length of event address is invalid (test_create_event_4)
    - Test5 Boundary: create an event with all valid values, the length of description is 600 (test_create_event_5_boundary)
    - Test5: create an event with all valid values except that the length of description is invalid (test_create_event_5)
    - Test6 Boundary: create an event with all valid values, the length of image path is 200 (test_create_event_6_boundary)
    - Test6: create an event with all valid values except that the length of image path is invalid (test_create_event_6)

        - Test7 Boundary: create an event with all valid values, the length of category is 20 (test_create_event_7_boundary)
        - Test7: create an event with all valid values except that the length of category is invalid (test_create_event_7)

delete_event()

- Description: Given an event id, delete the event from the database
- Equivalence partitions (Test cases):
    - Valid: the given event exists in the database
    - Invalid: the given event does not exist in the database
- Test plan:
    - Test1: delete a valid event (test_delete_event_1)
    - Test2: delete an invalid event (test_delete_event_2)

update_event()

- Description: Given an event id,and the columns and values to be updated, update the event information. Can only update description, time, (address, longitude, latitude) at one time.
- Equivalence partitions (Test cases):
    - Description:
        - Valid: length equal or less than 600
        - Invalid: length larger than 600
    - Address:
        - Valid: length equal or less than 200
        - Invalid: length larger than 200
- Test plan:
    - Test1: update an event with valid description value (test_update_event_1)
    - Test2: update an event with invalid description value (test_update_event_4)
    - Test3: update an event with a time value (test_update_event_2)
    - Test4: update an event with valid address value and longitude, latitude (test_update_event_3)
    - Test5: update an event with invalid address value and longitude, latitude (test_update_event_5)

get_event_id()

- Description: returns an event given event id
- Equivalence partitions (Test cases):
    - Valid: event id exist
    - Invalid: event id does not exist
- Test plan:
    - Test1: given an event id which does not exist in the database. (test_get_event_id_1)
    - Test2: given an event id which exists in the database. (test_get_event_id_2)

Get_all_event_created_by_user()

- Description: returns a list of events created by given user
- Equivalence partitions (Test cases):
    - User:
        - Valid: user exists

        - Invalid: user does not exist
    - Events created by user:
        - Valid1:user has not created any event
        - Valid2:user has created some events
- Test plan:
    - Test1: a user which exists in the database and he creates some (test_get_all_event_created_by_user_1)
    - Test2: a user which exists in the database and he hasn't created any events (test_get_all_event_created_by_user_2)
    - Test3: a user which does not exists in the database (test_get_all_event_created_by_user_3)

get_all_event_liked_by_user():

- Description: returns a list of events liked by given user
- Equivalence partitions (Test cases):
    - User:
        - Valid: user exists
        - Invalid: user does not exist
    - Events created by user:
        - Valid1:user has not liked any event
        - Valid2:user has liked some events
- Test plan:
    - Test1: a user which exists in the database and he likes some (test_get_all_event_liked_by_user_1)
    - Test2: a user which exists in the database and he hasn't liked any events (test_get_all_event_liked_by_user_2)
    - Test3: a user which does not exists in the database (test_get_all_event_liked_by_user_3)

get_all_history_event_by_user()

- Description: returns a list of events has attended by given user
- Equivalence partitions (Test cases):
    - User:
        - Valid: user exists
        - Invalid: user does not exist
    - Events created by user:
        - Valid1:user did not attend any events
        - Valid2:user has attended some events
- Test plan:
    - Test1: a user which exists in the database and he has attended some (test_get_all_history_event_by_user_1)
    - Test2: a user which exists in the database and he hasn't attend any events (test_get_all_history_event_by_user_2)
    - Test3: a user which does not exists in the database (test_get_all_history_event_by_user_3)

get_all_event_joined_by_user()

- Description: returns a list of events has joined by given user
- Equivalence partitions (Test cases):

        - User:
            - Valid: user exists
            - Invalid: user does not exist
        - Events created by user:
            - Valid1:user did not join any events
            - Valid2:user has joined some events
- Test plan:
    - Test1: a user which exists in the database and he has joined some (test_get_all_event_joined_by_user_1)
    - Test2: a user which exists in the database and he hasn’t join any events (test_get_all_event_joined_by_user_2)
    - Test3: a user which does not exists in the database (test_get_all_event_joined_by_user_3)

get_all_ongoing_event_by_user

- Description: returns a list of ongoing events has joined by given user
- Equivalence partitions (Test cases):
    - User:
        - Valid: user exists
        - Invalid: user does not exist
    - Events created by user:
        - Valid1:user did not join any events
        - Valid2:user has joined some events
- Test plan:
    - Test1: a user which exists in the database and he has joined some (test_get_all_ongoing_event_by_user_1)
    - Test2: a user which exists in the database and he hasn’t join any events (test_get_all_ongoing_event_by_user_2)
    - Test3: a user which does not exists in the database (test_get_all_ongoing_event_by_user_3)

get_nearby_events

- Description: given coordinates, returns a list of ongoing events sorted by distance
- Equivalence partitions (Test cases):
    - User:
        - Valid: user is provided
        - Valid: user is not provided
        - Invalid: user is provided but does not exists

- Test plan:
    - Test1: a user email is provided and the user exists (test_get_nearby_events_1)
    - Test2: a user email is not provided (test_get_nearby_events_2)
    - Test3 :a user email is provided but the user does not exist(test_get_nearby_events_3)

Get_all_ongoing_events

- Description: returns a list of ongoing events
- Equivalence partitions (Test cases):
    - User:

        - Valid: user is provided
        - Valid: user is not provided
        - Invalid: user is provided but does not exists
- Test plan:
    - Test1: a user email is provided and the user exists (test_get_all_ongoing_events_1)
    - Test2: a user email is not provided (test_get_all_ongoing_events_2)
    - Test3 :a user email is provided but the user does not exist(test_get_all_ongoing_events_3)

User

create_user()
- Description: create a user into the database
- Equivalence partitions (Test cases):
    - User:
        - Valid: user's name is equal or less than 25
        - Invalid: user's name is larger than 25
- Test plan:
    - Test1: create a user with valid name(test_create_user_1)
    - Test2: create a user with invalid name(test_create_user_2)

Get_user_by_email
- Description: return a user with the given email address
- Equivalence partitions (Test cases):
    - User:
        - Valid: an email address which exists in the database
        - Invalid: an email address which does not exist in the database
- Test plan:
    - Test1: get user using a valid email address(test_get_user_by_email_1)
    - Test2: get user using an invalid email address(test_get_user_by_email_2)

Get_attendees_by_event
- Description: return a list of users who attend the given event
- Equivalence partitions (Test cases):
    - Event:
        - Valid: event id exists in the database
        - Invalid: event id does not exist in the database
- Test plan:
    - Test1: get attendees using a valid event id(test_get_attendees_by_event_1)
    - Test2: get attendees using an invalid event id(test_get_attendees_by_event_2)

delete_user()
- Description: delete the given user from the database
- Equivalence partitions (Test cases):
    - Event:
        - Valid: user exists in the database
        - Invalid: user does not exist in the database
- Test plan:

    - Test1: delete a valid user(test_delete_user_1)
    - Test2: delete an invalid user(test_delete_user_2)

Note: When creating the event, we will assume the time of events are always valid(future time), for components in the frontend will check the validity of the time.

Integration Test:

Test 1:

- Description: test the event related APIs, like create, update and delete
- Test plan:
    - Visit the index page and login.
    - Create an event, check the event again to make sure that everything shown is the exact as the input.
    - Update the event information - updating its description, time, address, longitude and latitude.
    - Check the event information page again
    - Delete the page. Make sure that the deleted event will not show up.

Test 2:

- Description: test the event related APIs, like get nearby events
- Test plan:
    - Visit the index page and login.
    - Create several events
    - Call get_near_by API with no location information provided, check if three events show up.
    - Call get_near_by API with location information provided, check if three events show up and sorted by distance

Test 3:

- Description: test the comment related APIs, like create and delete comment
- Test plan:
    - Visit the index page and login.
    - Create an event
    - Make some comments under the event and check the comments show up
    - Delete some comments and refresh the page to see the deleted comments are gone.

Test 4:

- Description: test the join related APIs, get history events APIs and get attendees APIs
- Test plan:
    - Visit the index page and login.
    - Create an event in the past and join this event. Check the history events joined by the user and all the attendees of that event.
    - Leave the event and check again.

Test 5:

- Description: test the join related APIs and get ongoing events APIs
- Test plan:
    - Visit the index page and login.

    - Create an event in the future and join. Check the ongoing events joined by the user.
    - Leave the event and check again.

Test 6:

- Description: test the like related APIs
- Test plan:
- Visit the index page and login.
- Create an event and like it. Check all events liked by the user.
- Unlike it and check again.

Frontend Test Plan:
Frontend tests are mainly two parts: API tests and component tests.
For the API tests, we have tested all the APIs in our design (https://app.swaggerhub.com/apis-docs/yesOK/YesOK/1.0.0). Each API test has two equivalence partitions, the valid one returns the correct response, and the invalid one returns an error or exception. We tested them by mocking the corresponding request params and fetch calls. The corresponding testing code can be found here:
https://github.com/ybbbby/ADV-SE-Collaboration/tree/master/frontend/src/test/api
For the component tests, we mocked the props of each component and tested that they are rendering the right contents. Meanwhile, we tested that they are functioning properly by simulating user behaviours such as clicking, selecting, etc. The corresponding testing code can be found here:
https://github.com/ybbbby/ADV-SE-Collaboration/tree/master/frontend/src/test/components
https://github.com/ybbbby/ADV-SE-Collaboration/tree/master/frontend/src/test/pages

Part 3:

In general, we achieved 96% coverage and we have added Coveralls to our CI to generate coverage reports for each commit.

Backend:
We have achieved 94.61% coverage. Detailed report link: https://coveralls.io/jobs/71840990

Technical explanation:

- *Google auth* contains the OAuth steps for Google Login API. We find it hard to cover this part in our tests because one of the steps requiring a request to Google servers and the login step must be operated in a webpage. It's hard to handle it by code.
- Codes that are not covered in *app.py* are those handling exceptions. It's hard to simulate all kinds of exceptions.
- Codes that are not covered in *send_email.py* are because google email blocks travis.ci so that we can't test them on travis. However, we have tested them locally.
- Codes that are not covered in *database_connector.py* are because we use two separate databases for testing and developing. Thus the developing database won't be covered in testing.

Frontend:
We have achieved 100% coverage. Detailed report link: https://coveralls.io/jobs/71841015

Part 4:

Link to CI configuration:
https://github.com/ybbbby/ADV-SE-Collaboration/blob/master/.travis.yml
Link to an example CI report:
https://github.com/ybbbby/ADV-SE-Collaboration/runs/1513116766
You can see the CI log of each commit by clicking the check icon before each commit.