Assignment T5: Second Iteration

Team members:
Ping-Feng Lin, pl2730
Yuan-Hsi Lai, yl4305
Yen-Min Hsu, yh3328
Shao-Chi Wu, sw3525

Team name: TaiOne
IA: Asif Mallik

Part 1. User stories

s00 : As a live streamer , I want to show my exciting life in a cool way to my subscribers/audience so that I can be popular and probably earn some money.
My conditions of satisfaction are:

1. I can sign up with my own account.
2. There should be a web page showing relative information, e.g. how many people are watching.
3. I want to know the feedback from my viewer so that I can adjust what I'm going to do or respond to their comments in my streaming, creating a sense of interaction.
4. I must be able to save/keep the comments of my viewers.
5. I want to see myself on the screen

error/exception:

1. If the streamer cannot start building the streaming pipeline, it will show a prompt to tell him/her to wait for a few minutes and try again.

v00 : As a viewer, I want to watch quality streaming so that I will feel a little bit better about my miserable life.
My conditions of satisfaction are:

1. I want to sign up without much effort.
2. I want to know how much time in total I have spent watching.
3. I want to talk with the streamer as well as other audiences in the chat room

error/exception:

1. If a subscriber uses a browser our system does not support such as IE, there should be a message saying that "your browser does not support this, please change to firefox."
2. If a subscriber wants to leave, there should be a pop-up window showing for this subscriber to double check.

c00 : As a corporation , I want to put advertisements on the platform so that our image advertising strategy can work.
My conditions of satisfaction are:

1. I will need statistics such as the number of views by streamer, category. To be more specific, I need a statistic of a single category/streamer so that I can see the potential views of my advertisement.
2. This system should support advertising in the form of images.
3. I am capable of assigning ads to specific streamers who I think will maximize our company reputation.
4. I can also assign ads to specific categories that maximize our company reputation.

error/exception:

1. If a company has not specified that he is a company in the profile, he will not be able to access company advertisement features.

Part 2. Equivalence Partitioning & Boundary Conditions

Equivalence Partition

The first part is testing utils/db.py and utils/user.py.

Since these two files are mainly the management of databases, the results are generally landed into two partitions: we get our data or we get None (according to the input). This is the methodology from which I design valid and invalid equivalence partition testing.

Function	Valid	Invalid
utils/db.py		
get_db()	When db is in the project.	When db is not in the project.
close_db()	When db is in the project.	When db is not in the project.
init_db()	When schema is in the project.	When schema is not in the project.
init_db_command()	This is not really testable by itself, and it has been reported as a bug. https://github.com/pytest-dev/pytest/issues/5532 This is part of Flask fundamentals, so it should be okay. (I still leave the code of how I am going to test it in the comment.)	
get_streaming()	When table streaming has data.	When table streaming has no data.
create_stream(id_, name, category)	Create stream in streaming. When we provide all the info.	Create stream in streaming. When we miss a “not null” key.
update_stream(id_, m3u8_URL)	Update an existing row in streaming.	Update a non-existing row in streaming.
get_streaming_m3u8(id_)	Fetch an existing row in streaming.	Fetch a non-existing row in streaming.
get_current_watching(id_)	Fetch an existing row in streaming.	Fetch a non-existing row in streaming.

update_current_watching (id_, increase=True)	Increase the existing current_watching number by 1 in streaming.	Increase the non-existing current_watching number by 1 in streaming.
	Decrease the existing current_watching number by 1 in streaming.	Decrease the non-existing current_watching number by 1 in streaming.
delete_stream(id_)	Delete an existing row in streaming.	Delete a non-existing row in streaming.
insert_watch_history (watcher, streamer, UUID)	Create a row in streaming.	Create a row in streaming missing streamer Info.
		Create a row in streaming with wrong streamer Info.
update_watch_history(UUID)	Update a row in watch_history.	Update a row in watch_history with the wrong UUID.
get_watch_history(watcher)	Fetch an existing row in watch_history.	Fetch a non-existing row in watch_history.
	Fetch an existing row in watch_history where the watching has not ended.	
get_analytics_category_all()	Fetch all data from watch_history. Data is existing.	Fetch all data from watch_history. Data is empty.
get_analytics_category(category)	Fetch an existing row in watch_history.	Fetch a non-existing row in watch_history.
get_analytics_user_all()	Fetch all data from watch_history. Data is existing.	Fetch all data from watch_history. Data is empty.
get_analytics_user(userid)	Fetch an existing row in watch_history.	Fetch a non-existing row in watch_history.
update_user (id_, name=None, email=None, profile_pic=None, usertype=None)	Update an existing row in user.	Update a non-existing row in user.
		Missing a “not null” key.
insert_audience_comment (watcher, streamer, message)	Create a row in streaming in audience_comment. Just simple insertion, it almost never affects our system.	
get_audience_comment(streamer)	Fetch an existing row in audience_comment.	Fetch a non-existing row in audience_comment.

<table>
<tr><td>insert_advertise
(streamer, category, valid_until, image)</td><td>Create a row in advertise.
Giving all info.</td><td>Create a row in advertise.
Missing streamer info.</td></tr>
<tr><td rowspan="4">get_advertise(streamer = None)</td><td rowspan="2">Fetch an existing row in advertise.</td><td>Fetch an existing row in advertise.
An advertisement is expired, there isn’t any advertisement to use.</td></tr>
<tr><td>Fetch an existing row in advertise.
An advertisement is expired, there are some advertisements to use. But not the same category.</td></tr>
<tr><td rowspan="2">Fetch an existing row in advertise.
An advertisement is expired, but there is another advertisement to use.</td><td>Fetch an existing row in advertise.
An advertisement is expired, there are not any advertisements to use.
But there are advertisements from other streamers that are in the same category.</td></tr>
<tr><td>Fetch an existing row in advertise.
Advertise is empty.</td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td colspan="3">utils/user.py</td></tr>
<tr><td>get(user_id)</td><td>Fetch an existing row in user.</td><td>Fetch a non-existing row in user.</td></tr>
<tr><td>create(id_, name, email, profile_pic, usertype)</td><td>When we provide all the info.</td><td>When we miss a “not null” key.</td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td></tr>
</table>

The second part is testing app.py. If not specified, the tests related to a function will be test_<function name> in test_app.py

Many invalid use cases are handled by Flask’s login_required decorator, so we don’t need to worry about guest users accessing login-required endpoints.

function	Valid equivalence partition(s)
	Invalid equivalence partition(s)
app.py	
index()	1. Users reach endpoint "/" before signing in. 2. Users reach endpoint "/" after signing in.
callback()	Users get redirected by google authorization endpoint, to our "/login/callback" endpoint with valid authorization code. Users who get redirected to "/" if they have signed up before, otherwise they get a page to configure profiles. (test_callback, test_callback_exist)
	Users who reach this with an expired authorization code or whose emails are not verified will receive a status code 400. (test_callback_invalid)
newuser()	Users who have given their consent to OAuth can sign up using this endpoint.
	Users who have not given their consent to OAuth will not be logged in when accessing this endpoint.
stream()	1. Sreamers access this endpoint to start streaming but the AWS stacks don't exist. (test_stream) 2. Streamers access this endpoint to start steaming and the AWS stacks already exist. (test_streamExist) 3. Streamers access this endpoint to clear AWS stacks. (test_streamDelete) 4. Streams reach this endpoint to prepare to start streaming. (test_stream)
	Streamers access this endpoint but we fail to start creating a streaming pipeline. (test_streamFail)
stream_describe_stack()	Streamers access this endpoint after starting AWS stacks construction. (test_stream_describe_stack, test_stream_describe_in_progress,

	test_stream_delete)
	Streamers access this endpoint but they have not started construction of AWS stacks yet. (test_stream_yet)
stream_show_video_player()	Streamers access this endpoint after AWS stacks construction is completed. (test_stream_show_video_player, test_stream_show_video_player_in_progress)
	Streamers access this endpoint before AWS stacks construction is completed. (test_stream_show_video_player_yet)
view_id(id_)	Viewers go to this endpoint to watch streaming video of a specific streamer.
	Viewers go to this endpoint to watch streaming video of a specific streamer, but the streamer is not streaming.
company(type_)	Company users reach this endpoint to see statistics of streaming.
	Company users select an non-existing category.
thanks()	Company users reach this endpoint to post an ad image.
	Company users forget to upload an image.

Explanation for functions lacking equivalence classes:

Every @login_required function has a equivalence class in common, which is unauthorized users accessing them. These cases are handled by the function with the decorator @login_manager.unauthorized_handler decorator, in our case it is unauthorized().
load_user(user_id): a helper function
login(): redirect users to google authorization endpoint
login_t(): login function just for testing usage
logout(): just a line of code
get_google_provider_cfg(): getter function
profile(): login required
editprofile(): login required
payment(): a toy payment page

allowed_file(filename): getter function
Boundary Condition

In our system, we limit the name length of user names as below (from "/newuser").

During boundary condition testing, we try name lengths 49, 50, 51, and we expect that the invaild operations (name length too long) will be handled and will not affect my data integrity in the database.

```
if len(name) > 50:
    return render_template("newuser.html", message="Full name too long", userid=uid, fullname=name,
                           email=email, profile_pic=profile_pic)
```

And it works!

Frontend

For the frontend, we are mainly rendering html with javascript written inside it. The first thought is to use Jest to test the Javascript part. After some research on Jest, we found that it requires a separate js file and it also requires javascript functions to be tested. However, our Javascripts are mostly non separable from html since we are sending variables to both js and html; also, most of our javascripts are functionless. Moreover, one library we are using is d3js, but in the Jest test, we couldn't find a way to import it and perform the jest test.

We are all new to the front end and this course does not focus on instructions on the front end, so we can only gather JS code from different sources and tutorials to make the system function as we expected. If we use a more common front-end structure such as angular, vue, or react, there would be plenty of tools to test our code. I think this is how we can improve next time. Hope that points won't be taken away from frontend testing.

Part 3. Branch Coverage

The coverage report (generated by PyCharm) for the backend is included in report/iteration2/backend in our repo. Please download the directory and open index.html. Please ignore /Applications/PyCharm.app/Contents/plugins/python/helpers/pycharm/* files mentioned in the report. Only the items shown in the following picture are relevant.

https://github.com/Jefflin413/ASE-team-project/tree/master/report/iteration2/backend

/Users/hym/PycharmProjects/ASE-team-project/app.py	273	3	0	62	3	98%
/Users/hym/PycharmProjects/ASE-team-project/test_app.py	731	2	0	6	1	99%
/Users/hym/PycharmProjects/ASE-team-project/utils/db.py	219	0	0	86	0	100%
/Users/hym/PycharmProjects/ASE-team-project/utils/user.py	22	0	0	4	0	100%

Although we are unable to write unit tests for the front end as mentioned above, we managed to use chrome to test the coverage of each html and its javascript for our project. We managed to achieve and average 90.43% coverage over all webpages. The screenshots are inside the frontend folder under test.

Part 4. Continuous Integration

https://github.com/Jefflin413/ASE-team-project/blob/master/.travis.yml

https://travis-ci.com/github/Jefflin413/ASE-team-project/builds