Team17 Final Report

shallWeGame: SNS for Gamers

I. Abstract

1. Introduction

shallWeGame provides a neat and efficient platform for gamers to share daily game routines. Users can freely show their identities as an active playing gamer by posting their game lives. Also, they can look for other gamers to play with, see posts about games they like, or that their followers have posted.

2. Differential point

Our service is similar to Instagram in the form of a microblog. Users can share posts with short texts and images. Also, shallWeGame borrows the concept of 'follow' in Instagram, so that users form and reinforce social relationships through posts and communication. We expand the use of this functionality for gamers to play games together. If users follow other users, they not only see posts of followers but also invite them to the chatroom.

shallWeGame is alike Inven as SNS for gamers. However, shallWeGame limits the form of posts to show posts in a catchy way. Also, unlike Inven, shallWeGame provides private space for individuals such that they can express their identity as gamers and share their own gaming lives. Also, users can find other users to play with among their followers or even globally.

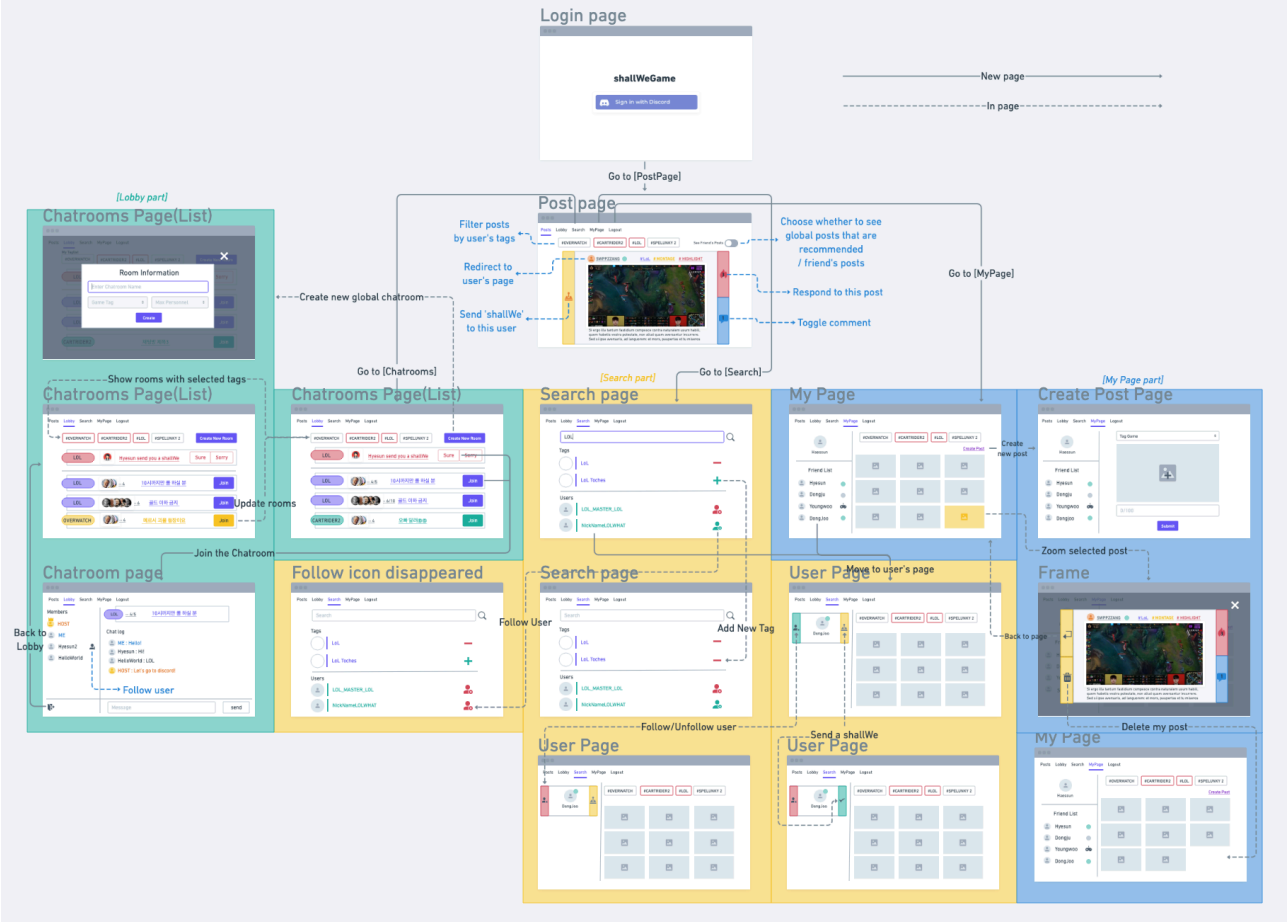
To sum up, shallWeGame is distinct from the existing services by the following properties: shallWeGame provides individual spaces for gamers to post their own gaming lives. shallWeGame provides gamers to find users to play certain games among their followers or globally. In shallWeGame, users can propose other users to play games with through posts they like.

3. Key feature

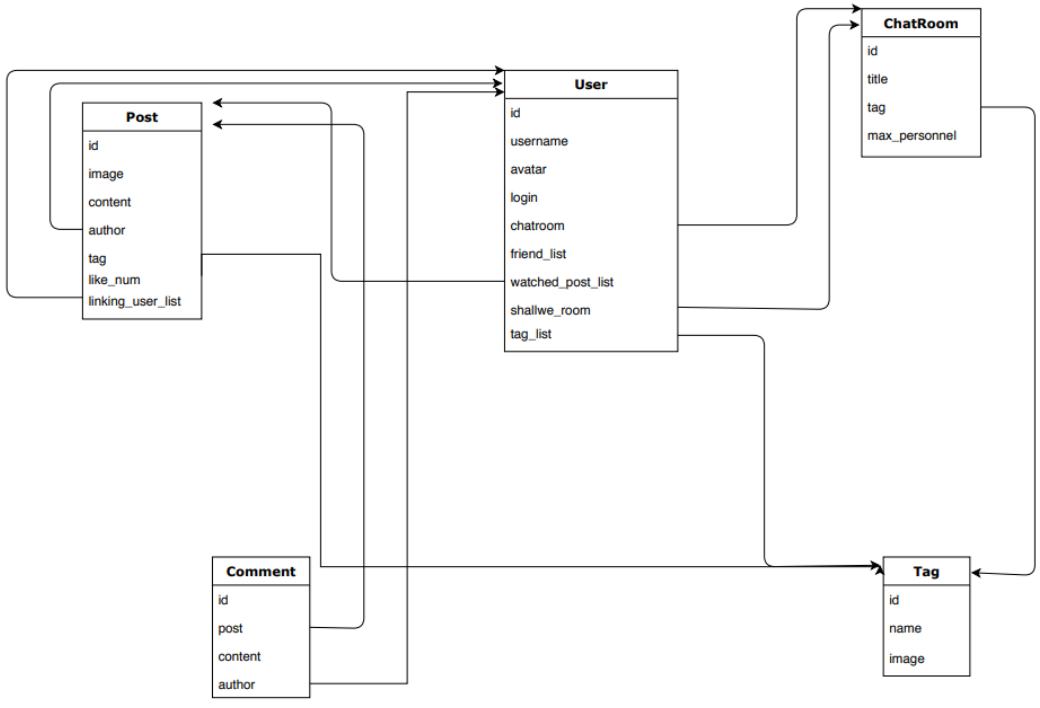
In shallWeGame, users can send 'shallWe' privately to whom they hope to play with through posts. We expect our service to be a platform that users actively communicate with other gamers on specific games they like, and easily accessible to those who are interested in games.

II. System Architecture

1. Bird's eye view



2. Model



3. View

3.1. Login Part

1. **Login Page (‘/login‘)**

- If the user clicks ‘Login with Discord’ button, login is handled by Discord API.
- If the user is new, get user info(‘username’ and ‘avatar’) from Discord account.

3.2. Post Part

2. **Posts Page (‘/post‘)**

- Post has ‘shallWe’, ’like’, ‘comment’ buttons.
  - If the user is in a room, ‘shallWe’ button is disabled.
  - If the user clicks ‘shallWe’ button, the author of the post receives shallWe.
  - If the user clicks ‘like’ button, the number of likes of the post increments by 1.
  - If the user clicks ‘comment’ button, shows Comment.
  - If the user clicks ‘avatar’ of the author, redirect to UserPage of the author.

3. **Comment**

- Show comments of certain post.
- Get ‘comment’ as user input.
- If user clicks ‘submit’, comment is saved.

3.3. Lobby Part

4. **Lobby Page (‘/lobby’)**

- Show shallWe’s, and rooms with chosen game tags
- If the user clicks ‘Create New Room’ button, redirect to CreateRoom page.
- The list of shallWe’s are shown above the list of rooms.
  - ShallWe has ‘Sure’ and ‘Sorry’ button.
    - If the user clicks ‘Sure’ button, redirect to the room.
    - If the user clicks ‘Sorry’ button, the shallWe disappears.
  - Room has ‘Join’ button.
    - If the user clicks ‘Join’ button, redirect to the room.
- Create New Room ('/RoomInfo')
- Get ‘chatroom name’, ‘tag’, ‘max personnel’ as user input.
- If the user clicks ‘Create’ button, new chatroom is created and redirect to the chatroom.
- Chatroom Page (‘/lobby/:id’)
- Members are listed in left.
  - If the user clicks ‘follow’ button of a member, he/she is added as friend.
- Members can chat in this page.

3.4. Search Part

5. **Search Page ('/search')**

- Search games or users
- If the user clicks ‘search’ button, show list of searched games and users by getting ‘searched text’ as user input.
  - Searched game has ‘delete’ button if the user has corresponding tag, and has ‘add’ button if not.
  - Searched user has ‘unfollow’ button if he/she is a friend of the user, and has ’follow’ button if not.
  - If the user clicks ‘avatar’ of searched user, redirect to UserPage of searched user.

6. **User Page (‘/user/:id’)**

- Show his/her avatar with introduction and posts with chosen game tags.
- If the user clicks a post, shows ViewSelectedPost (redirect).
- If the user clicks ‘follow’ button, he/she is added to the user’s friend.
- If the user clicks ‘shallWe’ button, shallWe is sent to him/her and redirect to chatroom.

7. **ViewSelectedPost in User Page ('/post/:id')**

- Show details of certain post
- Function as same as post in Posts page.

3.5. MyPage Part

8. **My Page (‘/user/:id’)**

- Show my avatar, friend list, and my posts with chosen game tags.
- If the user clicks a post, shows ViewSelectedPost (redirect).
- If the user clicks ‘Create Post’ button, redirect to CreatePost page.

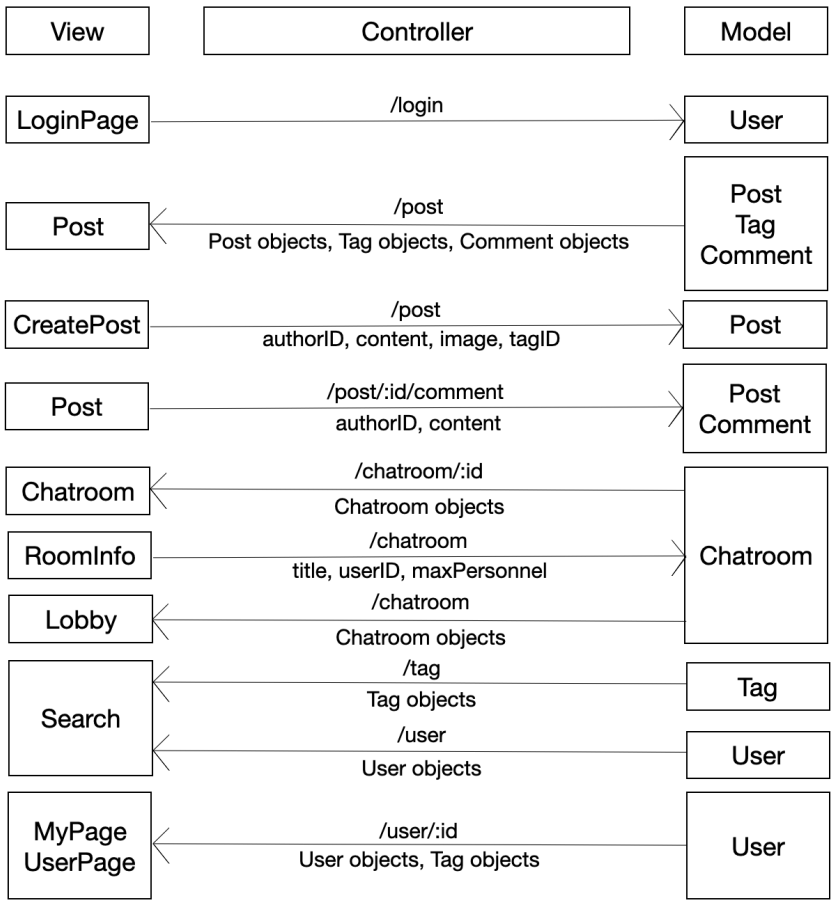
9. **ViewSelectedPost from My Page ('/post/:id')**

- Show details of certain post
- Function similarly as post in Posts page, but without ‘shallWe’ button.

10. **CreatePost Page (‘/user/:id/create’)**

- Get game tag, image, and text as user inputs.
- If the user clicks ‘submit’ button, new post is created.

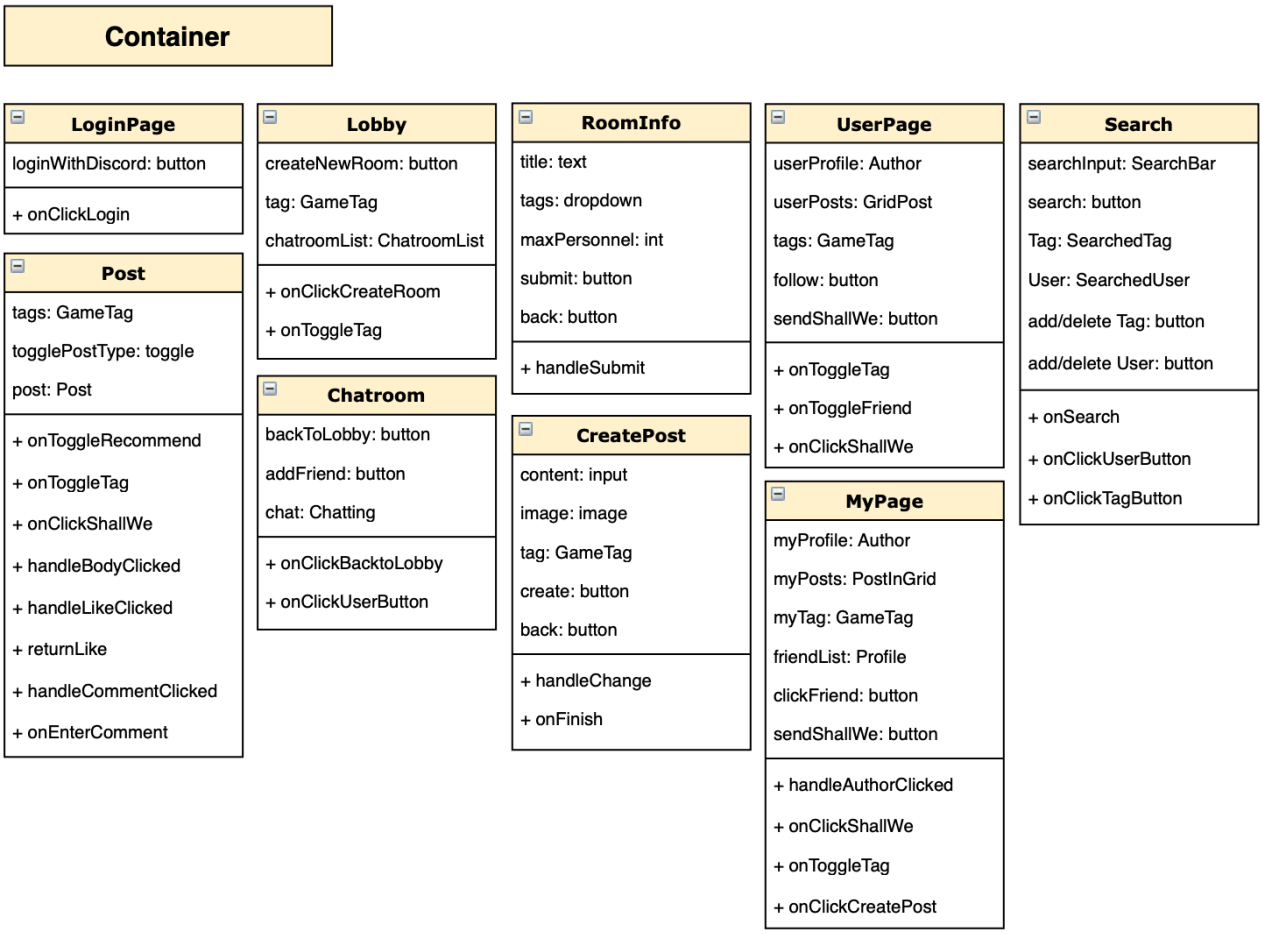
4. Controller



III. Design Details

1. Frontend Design

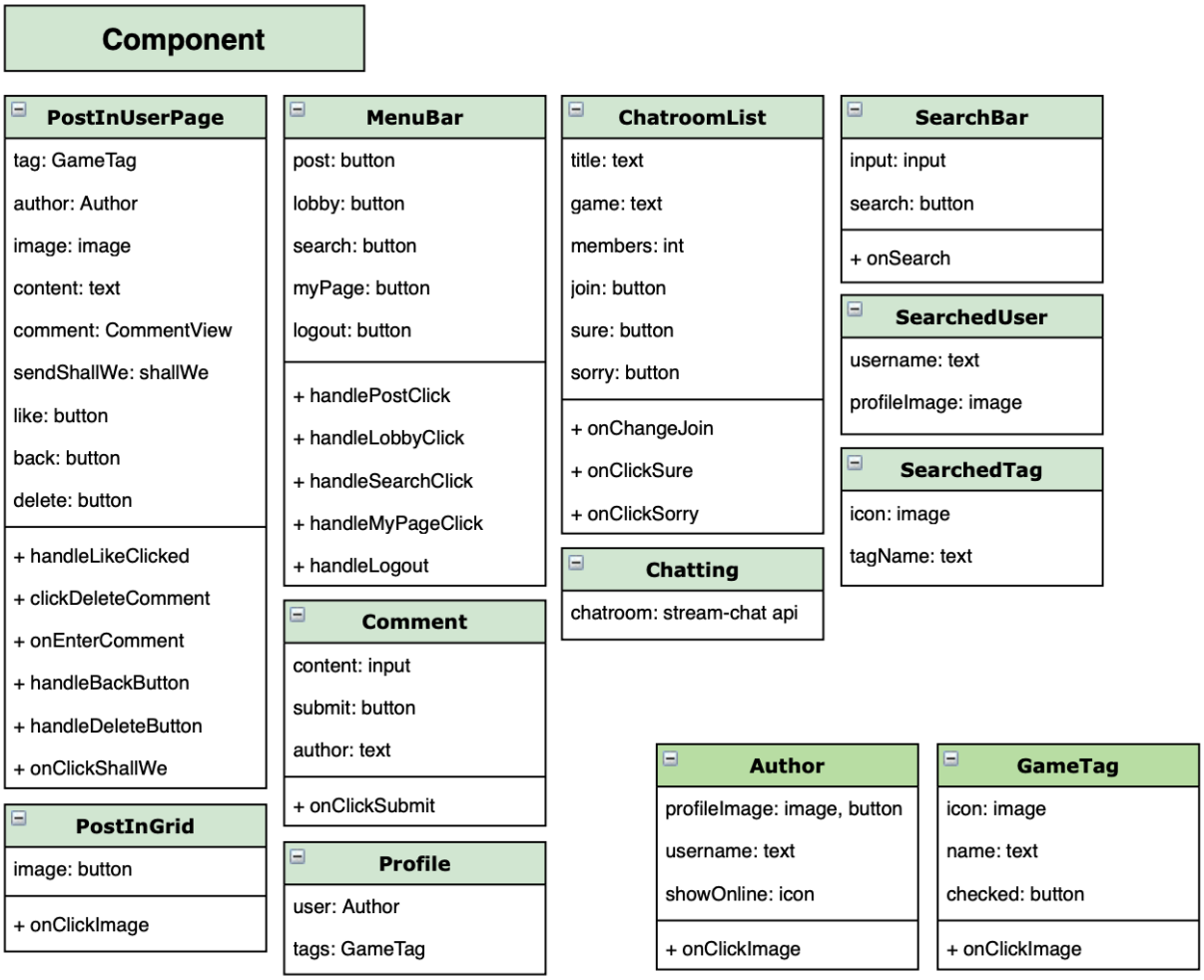
1.1. Containers

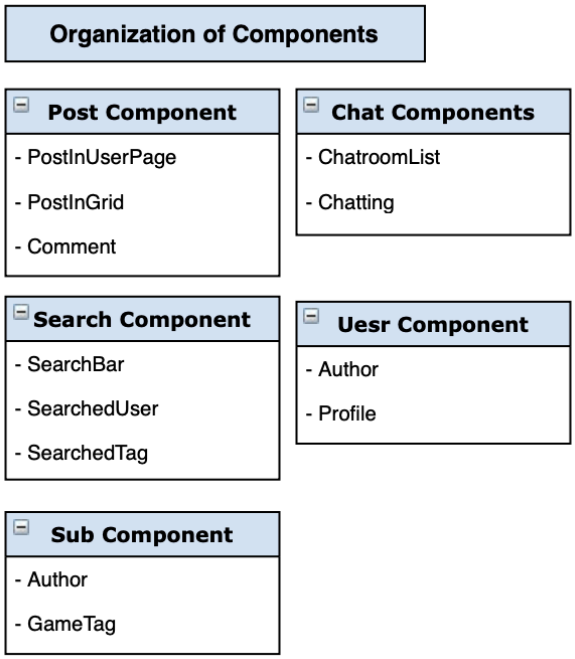


1. **Login**
   - onClickLogin : Redirect to ‘Discord’ and authenticate user
2. **MyPage**
   - onClickPost : Show all the details of selected post (redirect)
   - onClickCreatePost : Redirect to CreatePost page
   - onToggleTag : Toggle the selected tag (display posts filtered by selected tags)
   - onClickShallWe : Send shallWe to friend and redirect to chatroom
3. **Post**
   - onToggleRecommend : Toggle whether to watch recommended global posts or only friend’s posts
   - onToggleTag : Filter the posts to only watch the post with selected tags
   - onClickShallWe : Send shallWe to other user and redirect to chatroom
   - handleBodyClicked : Click the image to watch it bigger. View log collected and sent to backend
   - handleCommentClicked : Watch comment of the post
   - onEnterComment : Write new comment and submit

4. **Search**
   - onSearch(keyword : string) : Search and show the result of users and tags objects with the keyword in their name
   - onClickTagButton : Add or Delete selected tag from taglist of user
   - onClickUserButton : Add or Delete selected user from userlist of user
5. **Lobby**
   - onClickCreateRoom : Redirect to RoomInfo to create new room
   - onToggleTag : Toggle the clicked tag. Chatrooms with selected tags are shown to the user
6. **CreatePost**
   - onClickCreatePost(image: image, content: content, tags: int) : Create new post and redirects to MyPage.
7. **UserPage**
   - onToggleTag : Toggle the clicked tag. Posts with selected tags are shown to the user
   - onToggleFriend : Add or Delete selected user from userlist of user
   - onClickShallWe : Send shallWe to other user and redirect to chatroom
8. **Chatroom**
   - onClickBacktoLobby : Redirected to lobby
   - onClickUserButton : Add or Delete selected user from userlist of user
9. **RoomInfo**
   - handleSubmit : Create new chatroom and redirect to it

1.2. Components



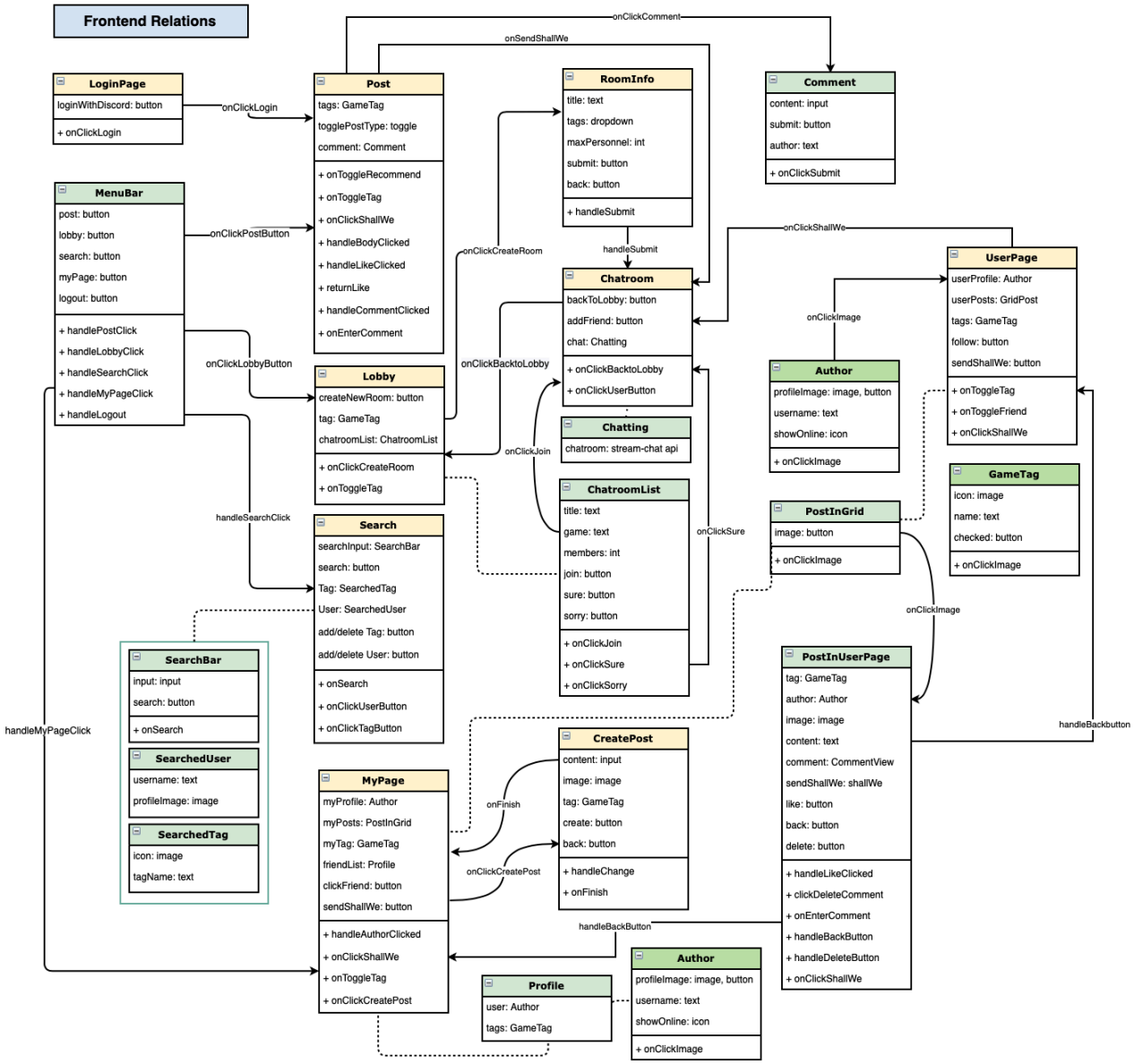


1. **MenuBar**
   - handlePostClick : Redirect to Post page
   - handleLobbyClick : Redirect to Lobby page
   - handleSearchClick : Redirect to Search page
   - handleMyPageClick : Redirect to MyPage
   - handleLogout : Logout and Redirect to Login page

2. **PostInUserPage**
   - handleLikeClicked : Increase or Decrease number of ‘like’ of the post when clicked
   - clickDeleteComment : Delete comment
   - onEnterComment : Create new comment when submitted
   - handleBackButton : Redirect to MyPage or UserPage
   - handleDeleteButton : Delete post
   - onClickShallWe : Send shallWe to friend or user
     and redirect to chatroom
3. **PostInGrid**
   - onClickImage : Redirect to watch specification of the post
4. **Comment**
   - onClickSubmit : Create new comment
5. **Author**
   - onClickImage : Redirect to UserPage
6. **GameTag**
   - onClickImage : Change the status of the clicked tag (selected / not selected)
7. **Profile**
   - consists of 2 components, Author and GameTag
8. **ChatroomList**
   - onClickJoin : Redirect to selected chatroom
   - onClickSure : Accept shallWe and redirect to chatroom
   - onClickSorry : Reject shallWe and the chatroom disappears (no redirection)

9. **Chatting**
    - component to show chat service with stream-chat api
10. **SearchBar**
    - onSearch(keyword : string) : Search and show the result of users and tags objects with the keyword in their name
11. **SearchedUser**
    - components to show searched user with profile image and name
12. **SearchedTag**
    - components to show searched tag with image icon and name of the game

1.3. Relations



2. Backend Design

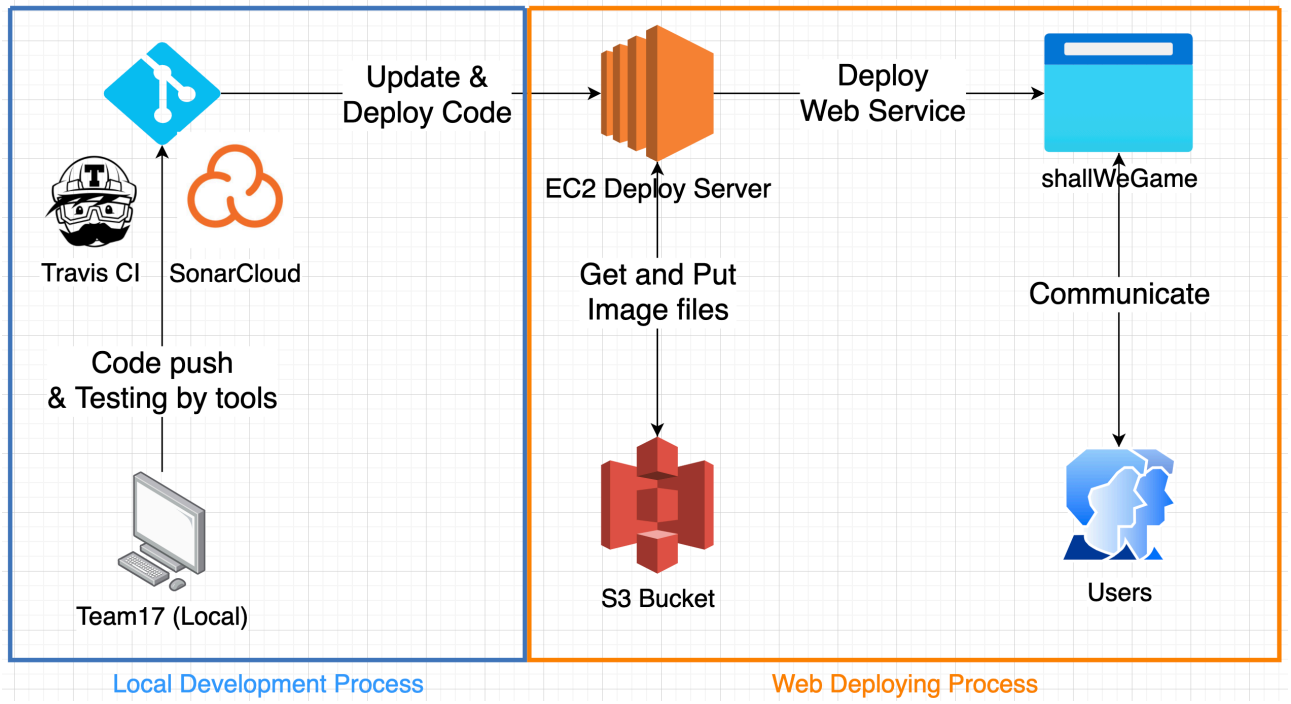
2.1. RESTful API

(prefix : '/api')

Model	API	GET	POST	PUT	DELETE
DiscordUser	login/	Redirect to discord login page	X	X	X
	login/redirect/	Redirect to shallWeGame after authorized by discord	X	X	X
	logout/	Log out	X	X	X
	currentUser/	Get current user	X	X	X
	user/	Get all users	X	X	X
	user/:id/	Get specific user	X	Change user info	X
Post	post/	Get all posts	Create new post	X	X
	post/:id	Get specific post	X	Edit post info	Delete specific post
	recommend/	Get recommended posts (ML)	X	X	X
Comment	post/:id/ comment/	Get all comments	Create new comment	X	X
	comment/:id/	Get specific comment	X	X	Delete specific comment
Tag	tag/	Get all tags	X	X	X
	tag/:id/	Get specific tag	X	X	X
Chatroom	chatroom/	Get all chatroom	Create new chatroom	X	X
	chatroom/:id/	Get specific chatroom	X	Edit chatroom info	Delete chatroom
	token/	Get token	X	X	X

IV. Implementation

1. Process Overall

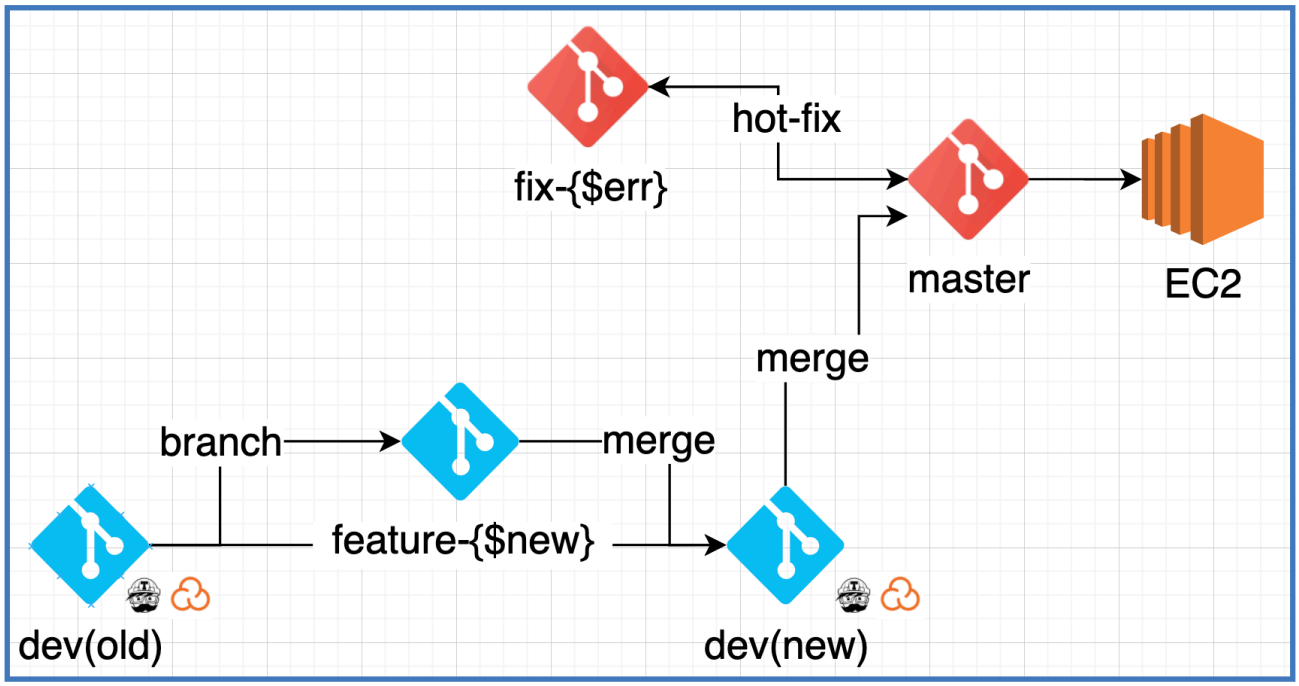


shallWeGame is implemented with Python3.7 for backend, React for frontend. Django is used as web framework. For front-end testing, ESLint is utilized as it is default option for it. About back-end testing, PyLint is utilized.

After testing in local environment, we manage our code by using Git as VCS and utilizing services such as TravisCI and SonarCloud, we were able to distribute our service with less errors. For deploying environments, AWS EC2 server takes a role to store deploying code, static files and to run web server. Considering specifications such as CPU and RAM and network bandwidth, the M5 model was selected. AWS S3 bucket were also utilized to handle massive image files with by storing and responding. S3 is authorized by deploying server(EC2).

We will focus on each of the left and right processes of the above-mentioned photos, describe the details and methodology our team has been used.

2. Local Development



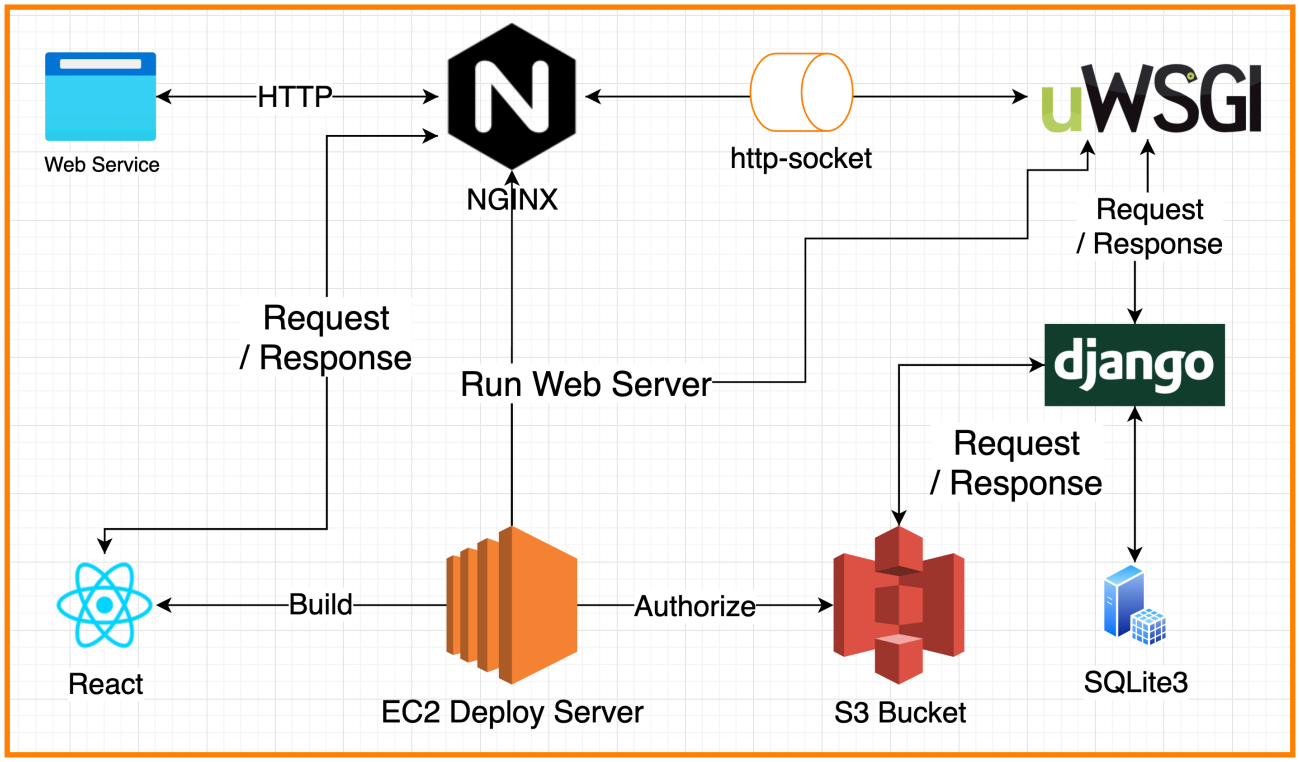
As developing as a team, we realized the need for a stable git-branch-strategy. By introducing different versions of the strategy and repeating trials and errors, we found the most effective branch strategy in our team's development environment and tried to keep it.

Standard branch that serves as the basis for developing a new feature is [dev]. To preserve [dev] branch's 24/7 stability while steadily introducing new features, we set CI tool's target branch to this. After testing with local environment and CI tools, we start developing a new feature by branching [feature-A] from [dev]. There can be multiple [feature] branches at the same time, but it must be through a tightly set internal rule in order to be merged toward [dev]. In conclusion, the expensive safety maintenance cost of dev branch ensures two major advantages. First, [dev] is always ready to send PR into [master] branch. Secondly, starting to develop a new feature is always no need to wait.

As mentioned earlier, [dev] should be easy to be cloned and operated in the local environment as it will have to go through frequent testing to maintain its stability. However, [master] branch should remain suitable for web server-driven environments. For that reason, our team's development priorities does not utilize the automatic build and deployment feature of CI tool. Therefore, it becomes clear that codes in [dev] branch that is running in the local environment and the codes in [master] that would be run in deploying server environment should have different contents of some "config" files represented by the proxy settings. As such, we performed merge action into [master] from [dev] with the command 'git checkout -p --{$FILE_WITH_LOCATION}', because, this procedure should always target specific files, not the entire repo.

By the result of this regulation, we could easily clone the [master]'s code into EC2 server. Considering the fact that git behavior on EC2 servers is somewhat limited and risky, we believe that the cost-benefit of previous investments is significant.

3. Web Deploying



The Deployment of the service is achieved by running NGINX and uWSGI after building a static code on an EC2 deploy server to build a static file.

When NGINX get HTTP request, it respond with static files including HTML, CSS, JS which are generated by the 'build' command performed by deploying EC2 server from its containing React sources. When the request includes API request, React sends internal request via NGINX toward backend. As Django itself cannot receives forwarded request, uWSGI acts as a translator. Active communication between NGINX and uWSGI is via http-socket communication. We can expect some speed improvement effect than the method that is done by basic command we dealt within the class.

4. Machine Learning

4.1. Objective

Our goal in Machine Learning feature is to recommend posts to user. We used crawled data from ‘Inven’ to pre-train the model and calculate similarity with input posts. We used Doc2Vec in gensim for pre-trained model

4.1. Datasets

We crawled data from ‘lol.inven.co.kr', ‘hs.inven.co.kr', ‘maple.inven.co.kr' which is the community for game ‘League of Legends’, ‘Hearth Stone’, ‘Maple Story’. Crawled data includes content(text) and images(1 image maximum for each post). We trained the model only with contents.

Number of Crawled posts from each Inven. (total 3279 posts)
- League of Legends : 980
- Hearth Stone : 1421
- Maple Story : 868

4.2. Preprocessing

1) Remove special characters, html tags and unnecessary whitespaces and dots
2) Tokenize the content by whitespace
3) Phrase (konlpy) the tokenized data by bigram
4) Make TaggedDocument (gensim) with bigram tokens

4.3. Train

- Made 3 models, one for each game
- Used gensim.models.doc2vec

```
model = doc2vec.Doc2Vec(window=10, workers=4, alpha=0.025, min_alpha=0.025, epochs=20, min_count=20, hs=1)
model.build_vocab(docs)
for epoch in range(100):
    model.train(docs, epochs=model.epochs, total_examples=model.corpus_count)
    model.alpha -= 0.002
    model.min_alpha = model.alpha
model.save(f'/Users/admin/Desktop/5-1/소개원실/ML/preprocessor/model_{game}_')
```

1) Set parameters (initial learning rate : 0.025)
2) Build vocabulary in model with TaggedDocuments
3) Train : epoch 100
- Decrease learning rate(alpha) by 0.002 for each epoch for better optimization

4.4. Recommend (implemented in backend)

- Input posts for pre-trained model are,
1) Written posts by user
2) Like-ed posts by user
3) Viewed posts by user (posts that user clicked in PostPage)

- If there is no posts to refer, randomly chosen the posts will be shown.

[Process steps for recommendation]
1) Preprocess the input posts
2) Vectorize preprocessed posts (model.infer_vector(input_data))
3) Feed the vectors in model to get the most similar posts (by its' unique id)

[Sample Results] : tuple(post_id, similarity)

- Hearth Stone

```
Most similar posts for post_id 775 are :
[(835, 0.4796614646911621), (1586, 0.39039379358291626), (2805, 0.33548930287361145), (2409, 0.33462244272232056)
, (1064, 0.3326381742954254), (2621, 0.31461983919143677), (524, 0.3106387257575989), (1587, 0.3041708767414093),
 (397, 0.2992543578147888), (1561, 0.29839780926704407)]
```

- Maple Story

```
Most similar posts for post_id 921 are :
[(1482, 0.8776152729988098), (1246, 0.8321461081504822), (1477, 0.7906079888343811), (1251, 0.6913303136825562),
(960, 0.6631165742874146), (1664, 0.6604825258255005), (1228, 0.6146957278251648), (955, 0.5931888222694397), (17
14, 0.590952455997467), (1677, 0.5671077966690063)]
```

- League of Legends

```
Most similar posts for post_id 775 are :
[(835, 0.4796614646911621), (524, 0.3106387257575989), (397, 0.2992543578147888), (1403, 0.2961180508136749), (49
0, 0.2851440906524658), (1307, 0.283713698387146), (297, 0.2759314775466919), (1203, 0.2726977467536926), (1296,
0.25297555327415466), (229, 0.25108468532562256)]
```

V. Difficulties

1. Discord API

- We planned to use Discord API in order to implement the feature of creating new discord chatroom url and upload it to chatroom in our service. Premise for the feature is that all users in shallWeGame chatroom should have discord account, so we made the login and authorization process to bypass discord login page.

1.1. Login Process (implemented)

1) Send request to discord and get ‘code’

2) Prepare headers that include discord developer’s client id, client secret and redirect uri

3) Post headers with code to discord Oauth

4) If user accepts to login to shallWeGame, get response that contains user information

1.2. Create Discord Chatroom (not implemented)

1) Create discord bot with discord account

2) Implement macro functions

- From this part, we found it difficult to implement because macro function for discord bot is triggered with ‘manual input’, but what we need was automatic creating chatroom service. Also, discord bot can only create chatroom with account that the bot belongs to, which means that all chatrooms are only available to be created in ‘one’ discord acoount.

2. ANT Design

- To reduce the time invested in css and focus on implementing core features, we introduce ANT Design library. We learned that using libraries involves both convenience and constraint. We would like to describe what I felt using this library and the trial and error of general css application together.
- It helps to implement css comfortably to a certain level, but if detailed adjustment is needed, it is often less flexible than the design that was written directly, as if it was stacked one by one, causing difficulties.

1.1. Convenience sacrifices flexibility

1) It helps to implement css comfortably to a certain level, but if detailed adjustment is needed, it is often less flexible than the design that was written directly, as if it was stacked one by one, causing difficulties.

1.2. Flex

1) It was unfamiliar at first to set up the main axis and to operate all logic based on it.

3. StreamChat API

- We used Stream-Chat API to implement real-time chatting service for users to communicate actively. This is also important for our main feature, shallWe. Stream-chat API provides appropriate UI for various circumstances and provides proprer pre-implemented chat services. Still, We found it difficult to customize the situations of using those features and save the chat-client, user_info, chat_logs and etc..
  But for result, we were able to implement chat service that includes,
  1) user can make new chatroom
  2) user can decide the title of chatroom
  3) users can communicate real-time
  3) user can send text, images, emoticons and also files

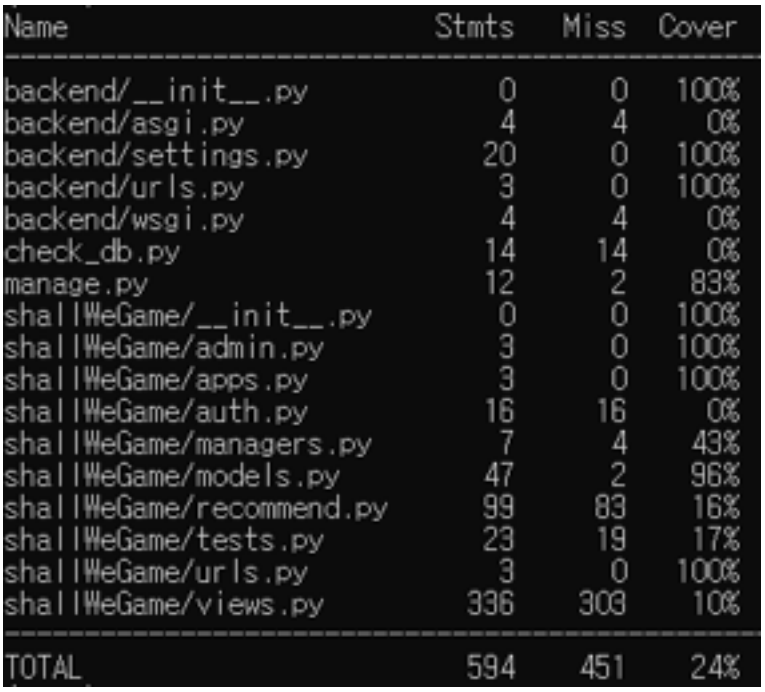
VI. Test

1. Front-end: Jest

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	83.03	68.38	79.23	83.83	
components/Author	100	83.33	100	100	
Author.js	100	83.33	100	100	71
components/ChatroomList	78	66.67	83.33	78	
ChatroomList.js	78	66.67	83.33	78	... 9,85,86,94,131
components/Chatting	100	100	100	100	
Chatting.js	100	100	100	100	
components/Comment	91.3	81.82	100	95.24	
Comment.js	91.3	81.82	100	95.24	84
components/GameTag	100	100	100	100	
GameTag.js	100	100	100	100	
components/PostInGrid	72.41	43.75	65.22	73.21	
PostInGrid.js	84	66.67	63.64	83.33	58,77,81,83
UserPagePostInGrid.js	63.64	30	66.67	65.63	... 33,43,44,59,78
components/PostInUserPage	56.82	36.67	37.5	58.02	
PostInUserPage.js	56.82	36.67	37.5	58.02	... 43,447,449,451
components/Profile	100	100	100	100	
Profile.js	100	100	100	100	
components/SearchBar	100	100	100	100	
SearchBar.js	100	100	100	100	
components/SearchedTag	100	87.5	100	100	
SearchedTag.js	100	87.5	100	100	38
components/SearchedUser	100	100	100	100	
SearchedUser.js	100	100	100	100	
components/SideBar	76.19	50	70	76.19	
SideBar.js	76.19	50	70	76.19	34,35,41,85,87
containers/Chatroom	75	69.57	69.57	75.86	
Chatroom.js	75	69.57	69.57	75.86	... 91,217,244,246
containers/CreatePost	30.56	4.17	25	29.41	
CreatePost.js	30.56	4.17	25	29.41	... ,59,60,138,140
containers/Lobby	86.27	77.27	88.89	87.76	
Lobby.js	86.27	77.27	88.89	87.76	... ,33,34,151,157
containers/Login	100	100	100	100	
Login.js	100	100	100	100	
containers/LoginPage	100	100	100	100	
LoginPage.js	100	100	100	100	
containers/MyPage	88.41	62.96	88.89	88.06	
MyPage.js	88.41	62.96	88.89	88.06	... 66,181,217,339
containers/Post	71.33	55	63.27	75.36	
Post.js	71.33	55	63.27	75.36	... 37,538,644,646
containers/RoomInfo	95.24	83.33	90	95.24	
RoomInfo.js	95.24	83.33	90	95.24	99
containers/Search	100	95.65	100	100	
Search.js	100	95.65	100	100	51
containers/UserPage	84.38	83.78	76.19	84.13	
UserPage.js	84.38	83.78	76.19	84.13	... 50,151,184,278
store	100	100	100	100	
store.js	100	100	100	100	
store/actions	93.23	83.33	93.07	92.95	
actionTypes.js	100	100	100	100	
chatroom.js	93.24	83.33	91.43	93.06	14,15,60,159,166
comment.js	94.12	83.33	100	93.75	13,14
index.js	0	0	0	0	
post.js	96.08	83.33	100	95.92	13,14
tag.js	90.48	83.33	100	89.47	13,14
user.js	87.5	83.33	82.61	86.96	... 96,103,104,106
store/reducers	93.42	81.48	100	93.33	
chatroom.js	88	76.47	100	88	51,52,54
comment.js	100	77.78	100	100	14,20
post.js	94.12	85.71	100	94.12	20
tag.js	100	100	100	100	
user.js	92.86	81.82	100	92.86	16
test-utils	100	100	100	100	
mocks.js	100	100	100	100	

We use Jest in frontend for unit test. Overall, we covered 83.03% lines of the code. The components related to image grid view are low in branch coverage as we couldn't handle react-photo-gallery, which is an external library. Also, testing feature of uploading image was quite challenging, which is seen in CreatePost component.

2. Backend: Django-test



Name	Stmts	Miss	Cover
backend/__init__.py	0	0	100%
backend/asgi.py	4	4	0%
backend/settings.py	20	0	100%
backend/urls.py	3	0	100%
backend/wsgi.py	4	4	0%
check_db.py	14	14	0%
manage.py	12	2	83%
shallWeGame/__init__.py	0	0	100%
shallWeGame/admin.py	3	0	100%
shallWeGame/apps.py	3	0	100%
shallWeGame/auth.py	16	16	0%
shallWeGame/managers.py	7	4	43%
shallWeGame/models.py	47	2	96%
shallWeGame/recommend.py	99	83	16%
shallWeGame/tests.py	23	19	17%
shallWeGame/urls.py	3	0	100%
shallWeGame/views.py	336	309	10%
TOTAL	594	451	24%

We failed to mock discord api including authentication request, so, all kinds of backend testing with user authentication(e. g. user information change, to make post) was failed. We couldn’t find documents about discord modules to help testing.