Open Source Report
Phase 3

For phase 3, we use the Django framework, redis server and MySQL database.

Django Framework:
For phase 3, we focus on the chat page which allows a user to add other users and chat with one of them, and also the profile page which allows a user to change the profile photo.

1.For chat page, we create the urls in Chat/urls.
(Source code: https://github.com/Rexhgx/CSE312/blob/develop/Chat/urls.py)
For profile page, we create the urls in Profile/urls.
(Source code: https://github.com/Rexhgx/CSE312/blob/develop/Profile/urls.py)

2.After getting the URL from the client, the server will first check if the associated function is callable.

(Source code: https://github.com/Rexhgx/CSE312/blob/develop/reports/phase%201/conf)

3.Then, the server will check the URL pattern is valid and call the associated function inside Chat/views.py and Profile/views.py.
(Source code: https://github.com/Rexhgx/CSE312/blob/develop/reports/phase%201/resolvers)

4.Inside the functions, if we need to access the database data, Django server will use QuerySet to store data to and read data from database. Also, the server can filter and order the data by QuerySet.

(Source code: https://github.com/Rexhgx/CSE312/blob/develop/reports/phase%202/query)

5.After getting the data, we use "render" function and pass data as one argument to get the specific template.

(Source code: https://github.com/Rexhgx/CSE312/blob/develop/reports/phase%201/loader)

6. After getting the template, the server is going to send the response to the client.
(Source code: https://github.com/Rexhgx/CSE312/blob/develop/reports/phase%201/response)

Authentication:

When a user sign up successfully, the server will store the username and the salted and hashed password into the database. When a user sign in successfully, the server will check if the username and the raw password provided are correct or not.

(Source code: https://github.com/Rexhgx/CSE312/blob/develop/reports/phase%203/hasher)

Once the user has signed in successfully, the server will set a cookie for users to auto sign in when visits the website again. Also, when the user tries to skip log in without already log in and use the website, the server will find the cookie value is not valid and redirect the page to the sign in page.

(Source code: https://github.com/Rexhgx/CSE312/blob/develop/reports/phase%201/response)

File Upload:

In the profile page, we allow users to change their profile photos by uploading an image type of file. When a user upload a file to the server, the server will get the file from "request.FILES" and assign it to the user's photo attribute which the type is FileField. Then the server will store it to the public place.

(Source code: https://github.com/Rexhgx/CSE312/blob/develop/reports/phase%203/fields)

Redis:
Inside Project/settings.py file, we use a channel layer that uses Redis as its backing store. A channel layer is a kind of communication system allows multiple consumer instances to talk with each other. I will explain why we use the channel layer for our project later. Once we run the 'docker-compose up -d' command, there will be a redis image created in a container called 'redis', and then we are going to connect our project to the redis server by port 6379.
(Source code: https://github.com/Rexhgx/CSE312/blob/develop/Project/settings.py)
(Source code: https://github.com/Rexhgx/CSE312/blob/develop/reports/phase%203/core)

WebSocket:

In the chat page, we implement the WebSocket by using the "Channels" library for two users communicate with each other. A channel is like a mailbox where messages can be sent to and every consumer instance provided by Channels has an automatically generated unique channel name, and so can be communicated with via a channel layer which is like the post office.

1.When the user enters the chat page, there will one request send from the JavaScript file by path "ws://localhost/chat/<room>". And then the server will implement the WebSocket handshake and switch the server to WebSocket. In order to prevent other users seeing the messages, we create a “room” for the two users. When messages are sent to this room, only the two users in this room are available to see the messages.

(Source code: https://github.com/Rexhgx/CSE312/blob/develop/reports/phase%202/websocket)

2.After sending the path to the server, the server is going to call a function “re_path” and pass the path and associate class as arguments to check if the path and associate class are valid or not.
(Source code: https://github.com/Rexhgx/CSE312/blob/develop/reports/phase%202/functools)

3.The associate class will receive messages from WebSocket and send messages to the front end.

(Source code: https://github.com/Rexhgx/CSE312/blob/develop/Chat/consumers.py)

Database:
Inside Project/settings.py file, we set MySQL database by giving the ‘ENGINE’, ‘NAME’, ‘USER’, ‘PASSWORD’, ‘HOST’ and ‘PORT’ values. For our project, the ‘HOST’ value is ‘db’ and the ‘PORT’ is 3306. Once we run the ‘docker-compose up -d’ command, there will be a MySQL database running in a container called ‘db’, and then we are going to connect our project to the database by port 3306.
(Source code: https://github.com/Rexhgx/CSE312/blob/develop/Project/settings.py)

Database Models:
One advantage for using Django framework is that we do not need to spending to construct database models manually. All we need to do is creating models in Django and run “python manage.py makemigrations” for checking any update in the Django models and “python manage.py migrate” for updating the database models.

(Source code: https://github.com/Rexhgx/CSE312/blob/develop/reports/phase%203/migration)