DOSP Project 4 Part-II
Twitter Clone

Divyasri Naraharisetti UFID: 7195 5711
Varsha Vulli UFID: 7548 8252

1 Description

This project aims to create a WebSocket interface for the part I implementation using the WebSharper web framework. This implies that even while the Erlang implementation (Part I) is allowed for client-server implementation using AKKA messaging, and we now need to build and utilize a suitable WebSocket interface. The following specifications must be executed:

1. All messages and their responses must be represented using a JSON-based API (including the errors).

2. To implement the WebSocket interface, you must completely rewrite some of your engine's code using WebSharper.

3. To use WebSockets, you must rewrite some of your client's code.

2 Implementation

Additionally, we need to try to implement the following features utilizing this service.

1. Setup or Register the account.

2. Send the tweets.

3. We need to subscribe to the user's tweet.

4. Retweet to share an intriguing tweet you found via another method with your followers.

5. Enabling searching for tweets that a user has subscribed to, have certain hashtags, or reference them (by mentions)

6. Provide the mentioned categories of tweets live if the user is connected (without querying)

In our video, we demonstrate all the aforementioned features, including new client user registration, login, tweeting, retweeting, and using hashtags and mentions. checking out the delete and logout features as well.

3 Execution

Client Implementation Details

1. client.erl and server.erl: This file contains data about a single client participant, which corresponds to a single Twitter user. Here, the data was stored using ETS. A method for storing Erlang data types in memory is offered by the ETS (Erlang Term Storage) module. ETS tables, which offer a number of actions for adding, removing, and searching up data, are an effective way to retrieve and store information in Erlang. ETS tables provide effective lookup operations depending on the key of the information stored and may be accessed by numerous processes at once. The information kept in ETS tables will be erased when the Erlang execution system is turned off since they are not durable. Certain portions of the server and client code have been modified by using WebSocket as mentioned below.

2. WebSocket Interface:

   (a) Firstly, using WebSockets, the client part has been redesigned.

   (b) Then, we have to use the web-socket interface to launch the Twitter server first.

   (c) We open a fresh or new terminals for each user to create an account, and then separate WebSocket connections are established for each and every user.

   (d) When a user client registers, the information is then stored in the ETS database and may subsequently be matched whenever the person is logged into the Twitter.

   (e) Nextly, each client user formed in the same sequence requests the server after constructing the simulation test case using all the querying apps permitted and defined in the aforementioned section.

   (f) Now, the client user then uses WebSockets to transmit the message back to the server through a gen-tcp connection.

(g) So, before sending the message to the server, it is then encoded into the JSON format. Later it is then processes the request and encodes the response, the server then delivers the message to the client user.

(h) The client user then decodes the response and publishes the pertinent messages for each registered user in the live chat window. The same thing happens when you choose to tweet, retweet, or subscribe to users on the Twitter clone that was deployed.

(i) The user-provided tweets and subscribers are encoded into the JSON format for each of the aforementioned parameters, and then the server's reply and any problems are likely to be encoded in the same JSON format before it is then being sent from the server to the client's window using the same WebSocket Interface Connection.

(j) When the option to query Subscribed user tweets, my mentions, or hashtag mentions is selected, the client encrypts this message in the JSON format, which the server will subsequently decode this format and then it extracts.

(k) Then this JSON message is parsed to retrieve this option, and the query's related API is then called. One general API handles all of the querying, searching for the specified term among all the tweets in the present or available ETS database table.

(l) Our modifications to the word to be searched that are dependent on the query from the request we got in the JSON file, which is formatted as (i) my mentions = @currentUserName (ii) hashtag Search = hastag word (iii) tweet Search = tweet to be searched.

(m) If a match is discovered, then the associated user clients UserName-Tweet Posted and any mistakes are also added are populated to the string. The search is conducted by looking up each user using the ets:lookup function, extracting the related tweets posted, and searching the phrase.

(n) Next, this very same websocket connection that was created when the user was registered is then used to send the response from the server to the JSON file. The response is automatically decrypted and then shown in each related user's live chat window.

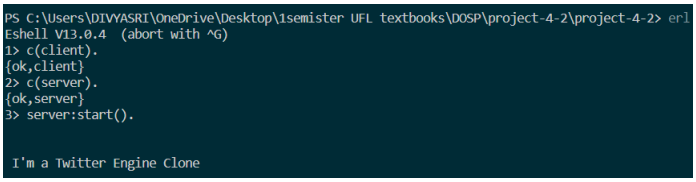
(o) The foremost registration ETS database table maintains process state and is used for disconnect and reconnect logic.

4 Output Solution

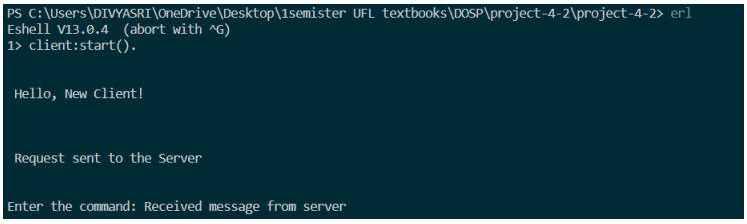
Running the Code

1. Firstly, open the command prompt with the server and client files. Then compile both the files by using **c(server)** and **c(client)** commands.

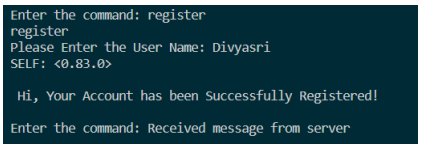
2. Next, start the Twitter Engine for starting the server using **server:start().** as shown below.

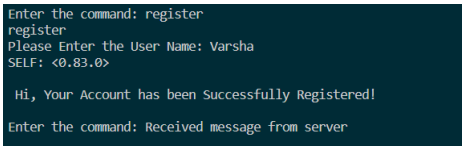


3. We have to then start two client terminals using **client:start().** as shown below.

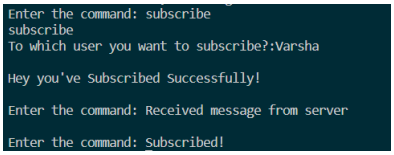


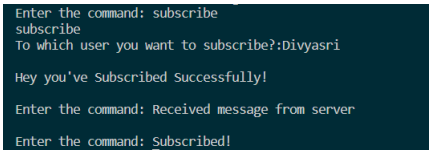
4. Now we have to register two different users using **register** option in the client terminal and the replies from the server are formatted into the JSON format.





5. Then one user can try to subscribe to the other user using the **subscribe** option and vice versa.





6. Next to type a new tweet use the **tweet** option from one of the client server. And In the other user sever we can observe the tweet which has been tweeted.

```
Enter the command: tweet
tweet
Let's Tweet?:Hi Varsha #dosp #twitterClone @Varsha

Success, Tweet has been Sent!

Enter the command: Received message from server

Enter the command: Your tweet is sent!

Enter the command: Received message from server

Enter the command: Server processed tweet
```

```
Enter the command: You have Tweeted!
Divyasri:Hi Varsha #dosp #twitterClone @Varsha
```

7. To demonstrate that all calculations take place at the server end, that connections are made using web sockets, and that messages are passed using JSON format, the following illustrates the JSON format printed at the server end.

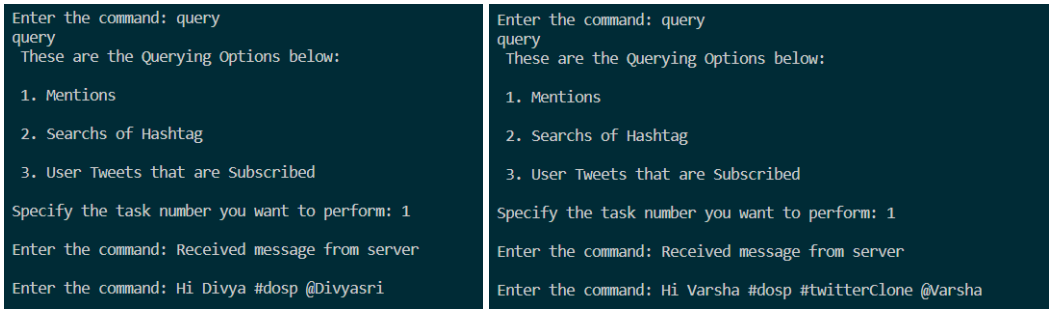
```
DATA: [<<"subscribe">>,<<"Varsha">>,<<"Divyasri\n">>]



TYPE: "subscribe"

 ["Varsha"]

Output after subscribing: [{"Divyasri",
                            [{"followers",["Varsha"]},{"tweets",[]}]}]
Do Receive



Output after tweeting: [{"Divyasri",
                         [{"followers",["Varsha"]},
                          {"tweets",
                           ["Hi Varsha #dosp #twitterClone @Varsha\n"]}]}]
Client to send: "Varsha"

Remaining List: []

Client Socket: #Port<0.6>

Sorry, No followers!
Send message!
Send message!
```

8. Using the **retweet** option from the second client user we try to retweet the first clients tweet.

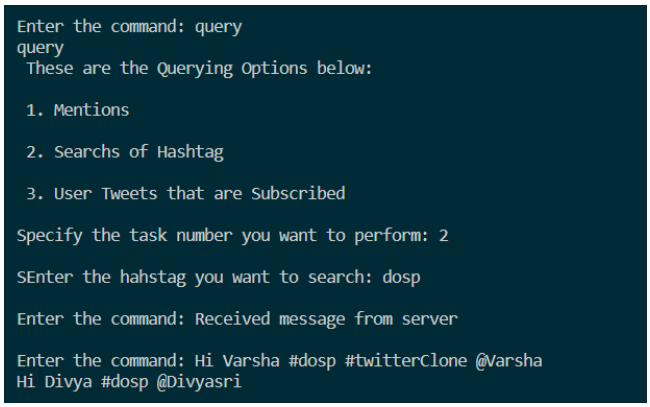
```
Enter the command: retweet
retweet
Enter the User Name whose tweet you want to re-post: Divyasri

Now, Enter the Tweet you want to re-post: Received message from server

Now, Enter the Tweet you want to re-post: Hi Varsha #dosp #twitterClone @Varsha

Now, Enter the Tweet you want to re-post:
```

9. If we choose the **query**option it then shows three different option:

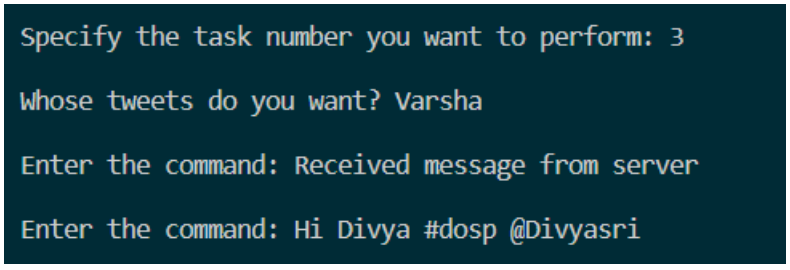
   (a) When the option 1 is chosen, The tweets that reference the current user are presented as follows



(b) When option 2 is chosen, you only need to enter the term without the hashtag to allow the option to query based on the hashtag.



(c) When the option 3 is chosen, The servers get a request to look up the tweets of the subscribed-to user right now.



10. Lastly, We can logout from both the client user servers.

Output Result Graphs

A graph is drawn showing that how long it will take for simulation to run relative to the number of subscribers for both Message Passing and WebSocket

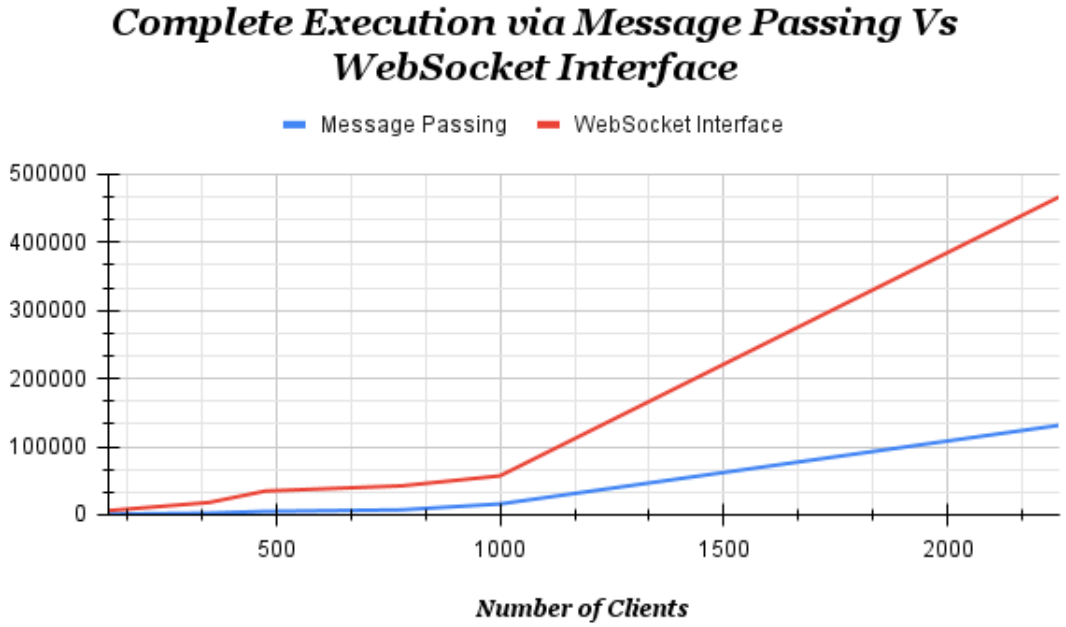
Interface. Since all of the components of our client-server architecture are distributed, we are certain that more clients may be created with enough CPU resources. The maximum no. of Users Simulated are: **10000**. The following values and combined graphs represent the outcomes we have derived:

1. Performance using Erlang Message Passing when Max Subscribers(and max tweets/account) = Number of Clients (0% disconnectClients)

Number of Clients	Max Subscribers	Time to Tweet, Retweet	Query Tweets Subscribed To	Query Tweets by Hashtag	Query Tweets by Mention	Query All Relevant Tweets	Complete Execution
125	122	46.23	126.83	39.52	16.83	7.42	887
350	340	130.1	559.94	180.582	300.154	28.972	2362
475	474	235.18	2313.92	323.418	397.68	149.386	5012
780	778	318.39	3527.74	636.11	472.44	238.12	7270
1000	999	1251.75	6620.81	1297.74	730.56	196.789	15898
2250	2245	5946.56	54364.68	7396.04	19759.375	2693.684	131320

2. Performance using WebSocket Interface when Max Subscribers(and max tweets/account) = Number of Clients (0% disconnectClients)

Number of Clients	Max Subscribers	Time to Tweet, Retweet	Query Tweets Subscribed To	Query Tweets by Hashtag	Query Tweets by Mention	Query All Relevant Tweets	Complete Execution (WebSocket Interface)
125	122	240.78	1326.48	395.94	85.87	59.76	6129
350	340	265.18	17356.73	2147.32	1111.77	313.21	18025
475	474	1269.63	21659.68	9420.345	9158.92	5325.03	34786
780	778	2569.28	46090.93	19560.89	5672.02	4669.21	42452
1000	999	4589.75	75070.48	13856.66	11994.47	1742.33	57125
2250	2245	8652.997	110089.695	16982.95	19759.375	1698.684	466510

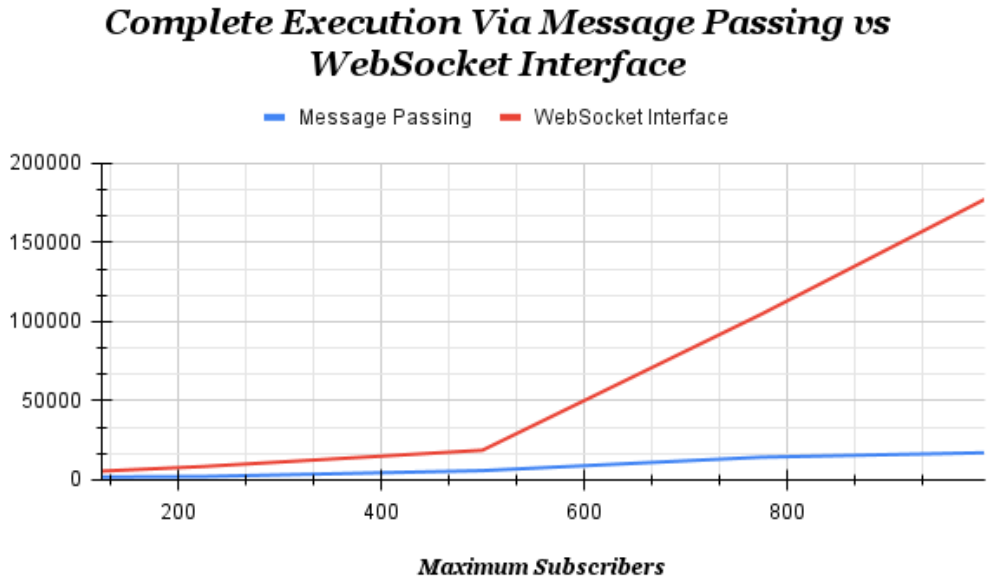


3. Performance using Erlang Message Passing for different number of Max Subscribers / Max Tweets per Account (0% disconnectClients)

Number of Clients	Max Subscribers / Max Tweets per Account	Time to Tweet, Retweet	Query Tweets Subscribed To	Query Tweets by Hashtag	Query Tweets by Mention	Query All Relevant Tweets	Complete Execution (WebSocket Interface)
1250	125	36.68	197.66	368.3	336.36	176.32	1836
1250	225	168.85	406.91	402.88	355.75	193.684	2152
1250	500	428.54	1798.34	1456.19	441.33	226.89	5849
1250	775	1660.92	7166.45	3241.22	1898.51	253.87	14351
1250	995	1536.22	11666.365	3756.47	2105.64	298.5	17155

4. Performance using WebSocket Interface for different number of Max Subscribers / Max Tweets per Account (0% disconnectClients)

Number of Clients	Max Subscribers / Max Tweets per Account	Time to Tweet, Retweet	Query Tweets Subscribed To	Query Tweets by Hashtag	Query Tweets by Mention	Query All Relevant Tweets	Complete Execution (WebSocket Interface)
1250	125	242.78	270.98	1124.45	621.03	218.51	5549
1250	225	265.18	396.25	1452.98	659.22	197.97	8461
1250	500	1266.63	2895.15	1845.02	958.62	589.93	18722
1250	775	2566.28	19568.7	5219.65	5158.97	2263.81	104699
1250	995	4592.75	40542.39	18452.08	13289.12	7031.84	177238



Below is the link for the Project video : `https://youtu.be/nYItXIvy2so`