

# DOS Project 4 Part 2

Saptarshi Chakraborty, UFID - 8857-1418  
Vagisha Tyagi, UFID - 0428-9808

Demo video link -

<https://youtu.be/nGsJXSBv0jQ>

## 1. Documents submitted

- 1) Code for the project in zip format
- 2) Readme file in pdf format
- 3) Demo Video youtube link

## 2. Aim and scope

Use Phoenix web framework to implement a WebSocket interface for the part 1 of project 4, i.e. using the same architecture of the engine implemented in the previous assignment, we are creating an websocket interface to communicate with the client. In this case, the client is the web UI developed using Javascript and HTML instead of the console based client used in the previous assignment.

We have designed the following functionalities for our project -

- 1) Register users in the system with login API
- 2) JSON based API to send and receive messages between server and client
- 3) Search the tweets
- 4) Online and offline tweeting
- 5) Retweet

## 3. Instruction for running the project in terminal

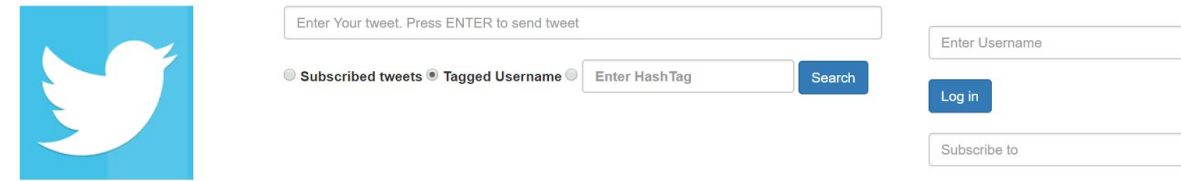
- 1) Unzip the **project4part2.zip** file.
- 2) Open terminal and navigate to the directory containing the unzipped folder '**project4part2**'
- 3) Type the following commands -
  - a) **cd hello/** [go to the hello directory]
  - b) **mix deps.get**
  - c) **cd assets/ and do npm install**
  - d) **cd ..** [come back to the hello directory]

e) **mix phoenix.server**

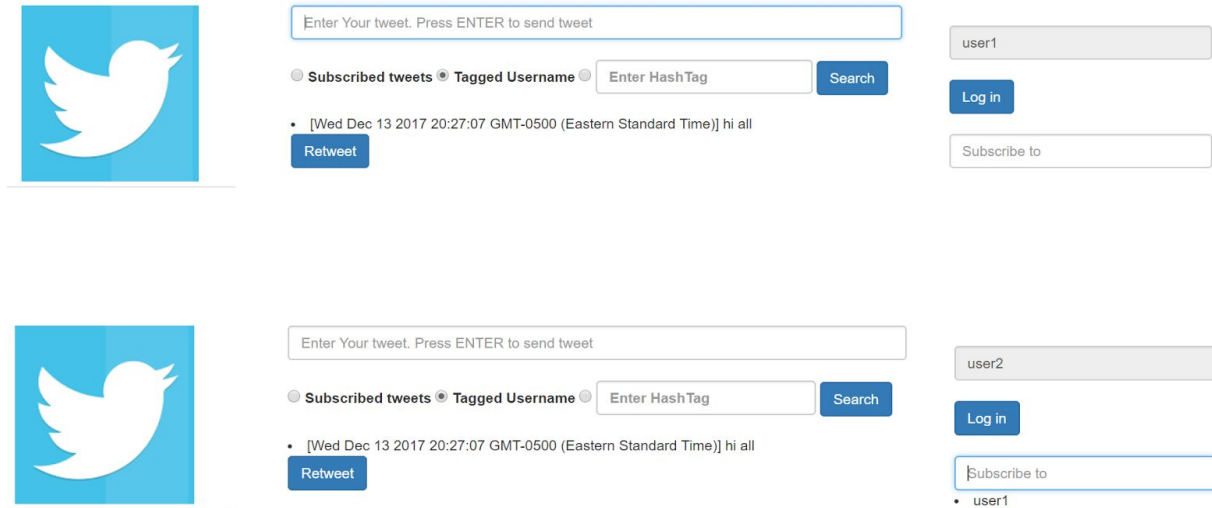
4) Run the above commands and type **localhost:4000** in the browser to open the UI.

#### 4. Instruction for running the project in Browser

- 1) After opening the browser, type the url and enter username and click on Login button. The text box will be greyed out.

A screenshot of a web application interface designed to look like Twitter. On the left is a blue square with a white bird icon. To its right is a text input field with the placeholder text "Enter Your tweet. Press ENTER to send tweet". Below this is a row of radio buttons: "Subscribed tweets" (selected), "Tagged Username", and "Enter HashTag". To the right of these is a "Search" button. Further right is a text input field for "Enter Username" and a "Log in" button. At the bottom right is a "Subscribe to" text input field.

- 2) If the user wants to subscribe to any other user, who is logged in, then go to the Subscribe to textbox and type the username of that user and press **Enter**. The subscriber list will be displayed below the **Subscribe to** textbox. User cannot enter its own name in the text box.
- 3) For sending the tweets, enter the tweet and press **Enter**. The message should be displayed to the user who tweets the message as well its subscribers. The following are the message structure expected while sending the tweets -
  - *"Hi all"* -> Message with no username and no hashtag
  - *"@user2 hi all"* -> Message with username and no hashtag
  - *"hi all #sample"* -> Message with no username but with hashtag
  - *"@user2 hi all #sample"* -> Message with username and hashtag
  - Username tag should be mentioned only at the beginning of the tweet and the next part of the message should be separated by a single whitespace.
  - Hashtag should be at the end of the tweet and a single whitespace should be between the message and the '#' character.



- 4) User needs to select a radio-button for searching tweets.
- 5) For searching all tweets, that it has subscribed to he/she needs to select the **Subscribed tweets** option and click on the **Search** button.
- 6) For searching all tweets tagged with its own username, user needs to select the **Tagged Username** option and click on the **Search** button.
- 7) For searching based on hashtags, enter the **hashtag value** in the hashtag text box and select the radio button, and click on the **Search** button. Only the search term should be entered and it should not contain the **#** letter.
- 8) All the search results will be displayed on the user performing the search query.
- 9) For Retweet, the user needs to click on the **Retweet** button and it will be send to its subscribers.

## 5. Implementation details

- 1) Every request response in this project is passed over the web-socket channels and every such request is associated with a message identifier, so that the server as well as client can identify what type of message is being sent over the web socket channel.
- 2) The backend architecture uses ETS as the in-memory data storage to store data, which are initialized on server start.
- 3) The request response uses a JSON based message passing data structure to communicate all messages as well as errors, e.g, - **{“new\_msg”:”hello all”}**, or errors such as **{“error”:”cannot subscribe himself”}**
- 4) The data storage is temporary so, on server restart the data can not be retrieved.

- 5) Online offline feature uses a different data structure to store the messages corresponding to the users, and this cache is cleared once the user logs in to the system after logging out.
- 6) For searching, the response of search results are also sent in JSON format, such as - `{ "results": ["msg1", "msg2"] }`

## 6. Performance result

We have used the following metrics to test the performance of our system -

- Scalability - This aspect tests the number of users which can be simulated in our simulator. We have tested with maximum 50 users.
- Tweet Rate - This determines the number of tweets that are sent to the server from the client. On average, we have tested with maximum around 20 tweets per second. As we have not automated this part, so we can only achieve up to this number, but the engine can handle more number of tweets per second.
- Request Rate - This metric shows the number of tweets received in the client, i.e. if a user has 10 followers, and it sends 1 tweet, then total 10 requests are performed in the server. In our project, the maximum we have tested is with almost 000 requests per seconds, i.e. with 50 clients and 20 tweets per seconds, when all of them are subscribed to each other.
- Retweet Rate - For our project, we have tested the retweet functionality with almost 10 retweets per second.
- Searching - We have performed search with a data set of almost 500 tweets to test the scalability of the search feature with all the three search criteria.
- Comparison - Compared to the previous assignment, the number of tweets and request rate per second is decreased as the previous assignment used simple message passing technique to communicate between the server and the client. But, here the overhead of maintaining a socket channel to communicate between server-client and as well as parsing the event types to take appropriate action reduces the performance.