

Distributed Operating System Principles (COP5615)

Project-4 Part II

Group Members

JAYACHANDRA GUNTURU (UFID: 56276000)

SATYA NAGA AKHILESH IRRINKI (UFID: 64730546)

Youtube link for the project demo: <https://www.youtube.com/watch?v=2N82bmm0s68>

1. Twitter-Engine Extension:

In project 4 part-1 we have implemented the Twitter engine and twitter simulator. For the second part of the project, we have implemented WebSocket interface on the first part. We have also implemented JSON based API representing the messages and replies exchanged between users.

2. Considerations:

Should design a JSON based API that represents all messages and their replies (including errors). Also, must re-write parts of the twitter engine accordingly using an F# Web Framework to implement the WebSocket interface. Finally, WebSocket should be implemented on the client side.

3. Implementation:

Implemented a minimalistic user interface through which the user can perform the actions like Tweet, Retweet, Hash Tagging, mentioning other users, subscribing to other users, searching for tweets with required hashtag and searching for tweets in which a user is mentioned. We have implemented a client server architecture. The two parts are implemented as below,

I. Server:

- The server exposes API endpoints on which the client can send messages (actions) to the server.
- The server receives requests in JSON format and gives responses in return as JSON format.

- The server returns response in the following format,

```
{  
    userName: string  
    service: string  
    message: string  
    code: string  
}
```

- UserName is the name of the user that receives the response.
- Service is the type of service (it can be tweet, retweet, follow, hashtag, mention, query, register, login) that has been handled.
- Message is the response to each of the above services. The response for each of the messages is as follows,
 - Tweet: The text of the tweet is in the message. It is returned to the user's subscribers. If a user mentions any other user, then it receives the mentioned user.
 - Retweet: It is the same as tweet except retweet text is added in front of the message.
 - Query: Searches for the tweets with given username or hashtag and returns the tweets if found. If not found it returns "No users found" or "No hashtags found" with an error code.
 - Follow: Searches for the given user to subscribe for. If the user is not found it returns "No users found". If found it returns the message subscribed to the given user.
 - Login: If the password is not correct or if the entered user has not been registered it returns "Invalid username / password". If the credentials are correct. It returns the message "User logged in successfully".
 - Register: If the user has already registered it returns user has been already registered. If the user is not registered it returns the message "User successfully registered".

II. Client

- The client consists of user interface through which actions are performed.
- In order to perform any action, the users has to first register and then login.

- The client requests are in the following format,

```
{
  userName: string
  value: string
  reqType: string
}
```

- The userName is the name of the user that is performing the action.
- The value for various actions would be as follows,
 - Register: The value would be the password of the user trying to register.
 - Login: The value would be the password of the user trying to login.
 - Tweet: The value would be the content of the tweet.
 - Retweet: The value would be the content of the retweet.
 - Subscribe/Follow: The value would be the name of the user subscribing to.
 - Query: The value would be the name of the user or the hashtag to be searched for.
- The reqType is the type of the request made by the user. It can be any of the above actions.
- The logged in user would be getting tweets from the user subscribed to.
- The user would also be notified if the user has been mentioned by any other user.
- After performing the desired actions, the user can logout by clicking the logout button.

4. Implementation of Web Sockets:

- We have implemented the websockets for the functionality of tweets.
- In real world scenario any number of users can have any number of subscribers.
- Therefore, if we use API requests each time an user does a tweet there would be very high number of requests. Also, the tweet has to be sent to every user in its subscribers list.
- That would be very costly as well as time taking. Websockets help us in this situation by continuously maintaining a connection to the server.
- The messages can be exchanged freely without performing requests each time a tweet is performed.
- Websockets are popularly known for their usage in live streaming, online games and many other crucial areas.
- That is the reason we have implemented the Websockets interface for the functionality of tweets.
- As soon as the user successfully logs in, a websocket connection is opened with the server.

5. Dependencies

- dotnet core sdk with version 6 and above.
- Suave library of version 2.6.1 and above.
- Akka actor model is to be installed.
- Package Fsharp.Json is to be installed.
- Akka.Fsharp package is to be installed.

6. Commands to run the program:

- Unzip the folder and move to the directory where the Program.fs is present
- Run the following command,

`dotnet run`

- The application will start running on the port 8080. It can be accessed by url "localhost:8080" in any browser.