DOS Project 4 – Twitter Clone using Elixir

Project Members:

Bhaasita Desiraju (UFID 4081-1191)

Nidhi Sharma (UFID 6843-1215)

Problem Description:

The goal of this project is to implement a Twitter Clone in Elixir. This has been achieved by creating Genserver actors where one actor acts as the Server and the remaining actors are clients. The clients send tweets, send tweets that contain hashtags and mention other users as well. The client sends a registration request to the server and the server registers the user by inserting the client and its PID in the user Register.

The functionalities implemented in this project are:

1. **The user can register an account and delete the account.:**
   This is achieved by the server registering a user after receiving a request. The server inserts the user in a user register. If the user sends a delete request, the server deletes the user's account, and the user is removed from the user register along with its tweets and retweets removed from their respective registers (ETS tables)
2. **The user can send tweets that can have hashtags and mentions of other users:**
   Every user that is registered can send tweets. This is achieved by selecting a tweet from a pre-defined set of tweets and the tweeting it. The tweets are added to a list of tweets that is linked to that user and stored in a tweet register. Similarly, the user can send tweets that contain hashtags, which are stored in a hashtag register and also mention other users in their tweets that are stored in the mentions register.
3. **The user can subscribe to another user**.
   A user can follow another user. For this, we add the following user in their following register as well as update that user's follower register.
4. **Re-tweets by a user:**
   The user can retweet from the list of tweets of the users that they follow. A tweet is fetched from the list of tweets and retweeted.
5. **Query tweets subscribed to, tweets with specific hashtags, tweets in which the user is mentioned:**
   A user can query tweets of a user that they are subscribed to. This is achieved by fetching the followed users tweet list and displaying them one by one. Also the tweets can be queried by mentions in which all the tweets a user is mentioned in is displayed. The tweets can also be queried by hashtags in which all the tweets in which a particular hashtag is mentioned is displayed.
6. **If the user is connected, deliver the above types of tweets live (without querying):**
   If a user is active, that is if it exists in the user register, then its tweets are displayed live which is shown as live view in our implementation.

Implementation Details:

Proj4.ex:
This elixir file is the entry point and hosts the main method. Our implementation takes two parameters, number of users and number of requests per user.
numOfUsers: the number of users to simulate (e.g.: 100)
numOfRequests: the maximum number of tweets a Twitter account can send (e.g. :10)
The maximum number of users that can be deleted is taken as 10 percent of total users.
Also, the maximum number of users that can be added as users' followers is taken as fifty percent of the total users. The function to display the simulation time is added in this file where the simulation time is calculated after all functionalities of tweeting, tweeting with hashtags and mentions and all other functions have completed.

Server.ex:
The server will be started automatically when the simulation is started without it having to be stared explicitly. This file contains the code for Twitter Server implementation that is responsible for processing all the tweets and the connection and disconnection as well as deletion of users. The engine directly corresponds with the databases that are implemented using ETS tables to handle users, their followers, their tweets, their queries for the hashtags, mentions and their followers' tweets, etc. The Client sends a request to the server from a user to tweet or to follow a user or register or delete a user account, which is processed by the server. It communicates directly with clients to display the results of a users' query for hashtags or mentions or tweets, it processes the users request to retweet and follow a user as well. Inside the server maintaining process states is done by the userRegister ETS table that maintains track of all users, that is Genserver actors.
The server maintains a register for every functionality namely, userRegister, tweetsRegister, hashtagsRegister, mentionsRegister, followingRegister, followersRegister, deletedUsers and disconnectedUsers.

Client.ex:
This file consists of the data concerning a single Genserver actor that stands for a single twitter user. This includes the information of its userPID and username that is in the format of "User1" which is passed to the server passed to it upon initiation and it also communicates everything to the server by means of a ServerPID. The Client part, that is, the user makes requests to the server to register itself, tweet, which may also contain hashtags and mentions of other user, follow a user, query for a hashtag, query for the tweets in which it is mentioned, query for tweets of user it is subscribed to and get a live view of its tweets when it is active or online.

BONUS PART

Zipf distribution –
As per the requirements we have simulated a Zipf distribution on the number of followers a user has. The Zipf distribution was implemented by adding followers to every user depending upon the Zipf constant which is calculated using the Zipf distribution function which states that the frequency of every element is inversely proportional to its rank meaning that the rank of a user if inversely proportional to its number of followers. The rank one user will have the highest number of followers and the last ranked user will have the least number of followers. We calculated the number of followers for a user using the formula:

$$\boldsymbol{Zipf\ Constant} = \left(\frac{\mathbf{1}}{\mathbf{1}} + \frac{\mathbf{1}}{\mathbf{2}} + \frac{\mathbf{1}}{\mathbf{3}} + \frac{\mathbf{1}}{\mathbf{4}} \ldots\ldots\ldots.\frac{\mathbf{1}}{\boldsymbol{n}}\right)^{-1} * \boldsymbol{Total\ Number\ of\ Users}$$

Then we determine the number of Zipf followers by $\frac{\boldsymbol{Zipf\ Constant}}{\boldsymbol{User\ rank}\ (\boldsymbol{i.e. User\ number\ in\ our\ case})}$ and add these followers to the followers list of all user accordingly. The following list of every user is updated accordingly. The number of tweets of users having more number of followers is increased proportionally.

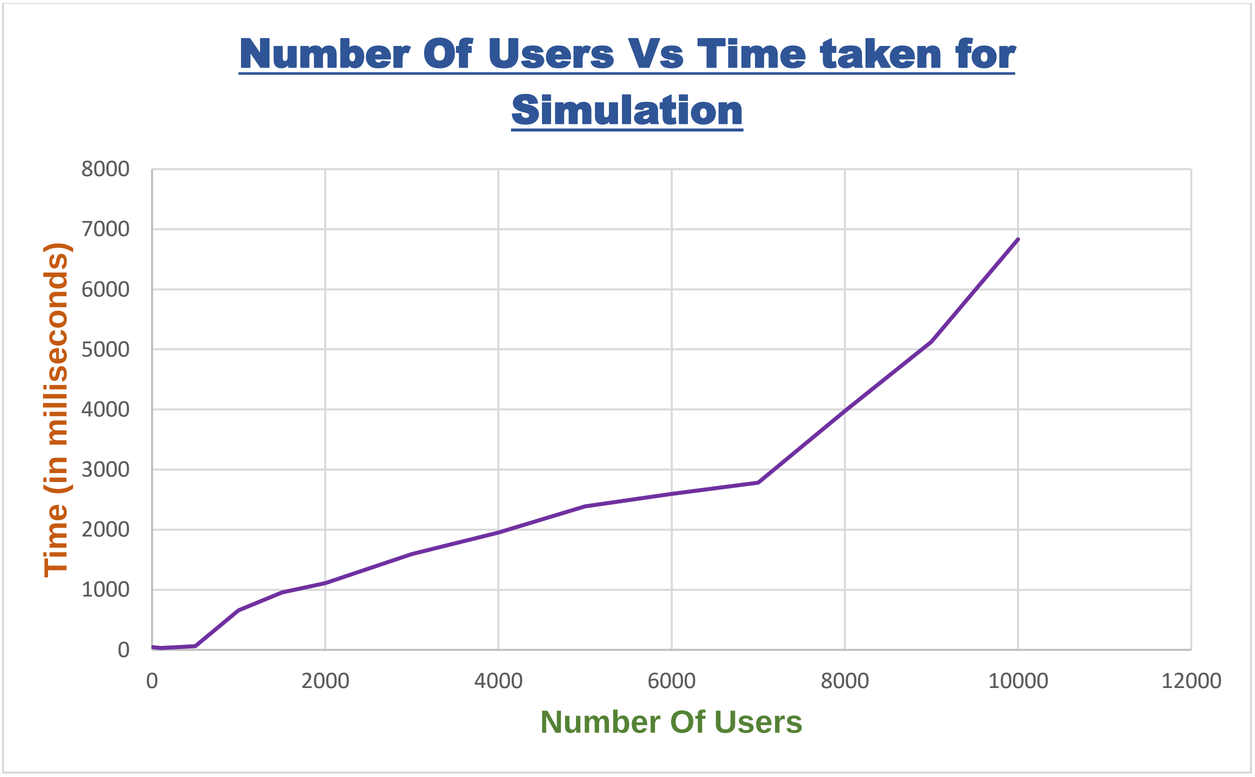
Thus, the client having the maximum number of tweets had more followers and hence it had more retweets as well. For users having more followers, the tweets by them had to be increased so that the Zipf Distribution can be achieved. Thus, through this Zipf implementation, we were able to imply the concept that, the more a user is popular, the more number of followers it would get and that would also result in their tweets being retweeted more number of times as compared to a less popular user. This is inline with the concept of Zipf law which can be applied to twitter in the sense that the more followers a user has, the more followers it would get. That is, more attracts more and less attracts less.

Periods of live connection and disconnection for users –
The second part of the bonus was to simulate periods of live connection, that is, a few users were disconnected at a given point of time and a few were simultaneously reconnected. This was achieved by selecting several users randomly from the list of users that are currently active and disconnecting them. For disconnection we inserted 'nil' in place of the userPID of the user that is to be disconnected, in the user Register. We stored the username and userPID in another register named as disconnected users, for the purpose of reconnection. The number of clients disconnected at a point were less than or equal to the 'Maximum number of disconnected users' which in our case was 10 percent of the total users. When reconnecting a user, that users' userPID was fetched from the disconnected user register and reinserted into the user Register and then deleted from the disconnected users' register.

Performance Analysis:

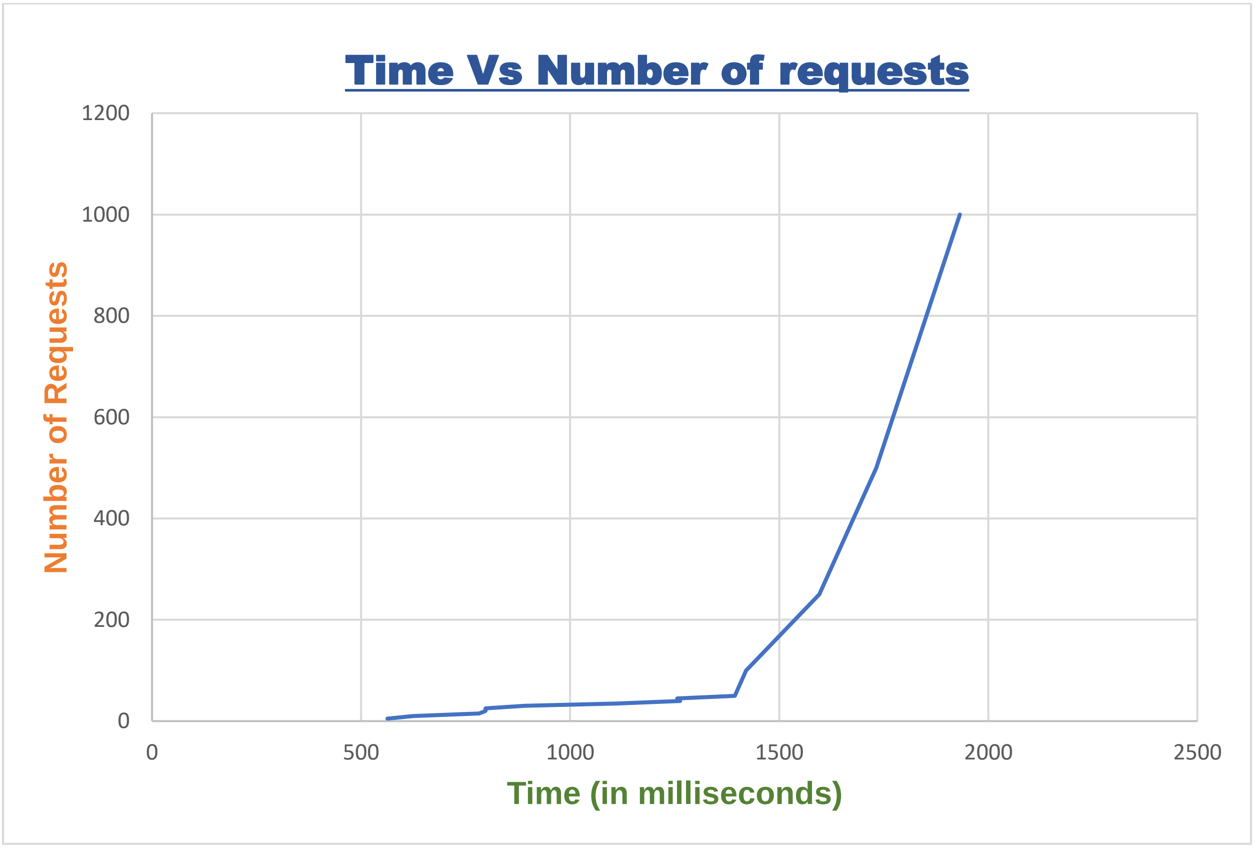
1. The time taken for the Twitter simulation to complete, is plotted against the total number of Users keeping the maximum number of requests per user constant to get the graph of Number of Users versus the total time taken. As observed from the graph, the time taken for the simulation to complete is uniformly increasing as we go on increasing the total number of users. Our implementation is successfully running for 10000 users and 50 requests per user.

Num of users	Time (in ms)	Num of requests
10	16	5
100	31	5
500	62	5
1000	656	5
1500	953	5
2000	1109	5
3000	1594	5
4000	1953	5
5000	2390	5
6000	2593	5
7000	2781	5
8000	3973	5
9000	5123	5
10000	6832	5



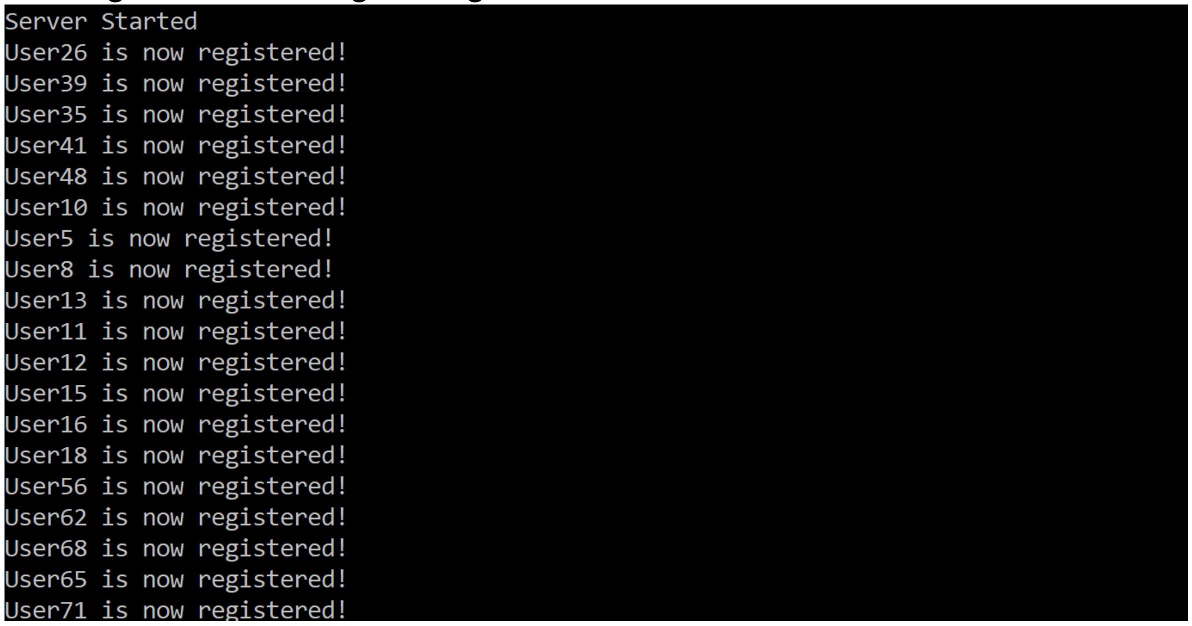
2. The time taken for the Twitter simulation to complete, is also plotted against the total number of requests made by every user, keeping the number of users constant to get the graph of total time versus the Number of Requests per Users. As observed from the graph, the time taken for the simulation to complete is uniformly increasing as we go on increasing the number of requests per user.

Num of users	Time (in ms)	Num of requests
1000	563	5
1000	625	10
1000	782	15
1000	798	20
1000	798	25
1000	891	30
1000	1109	35
1000	1263	40
1000	1256	45
1000	1394	50
1000	1421	100
1000	1596	250
1000	1732	500
1000	1932	1000



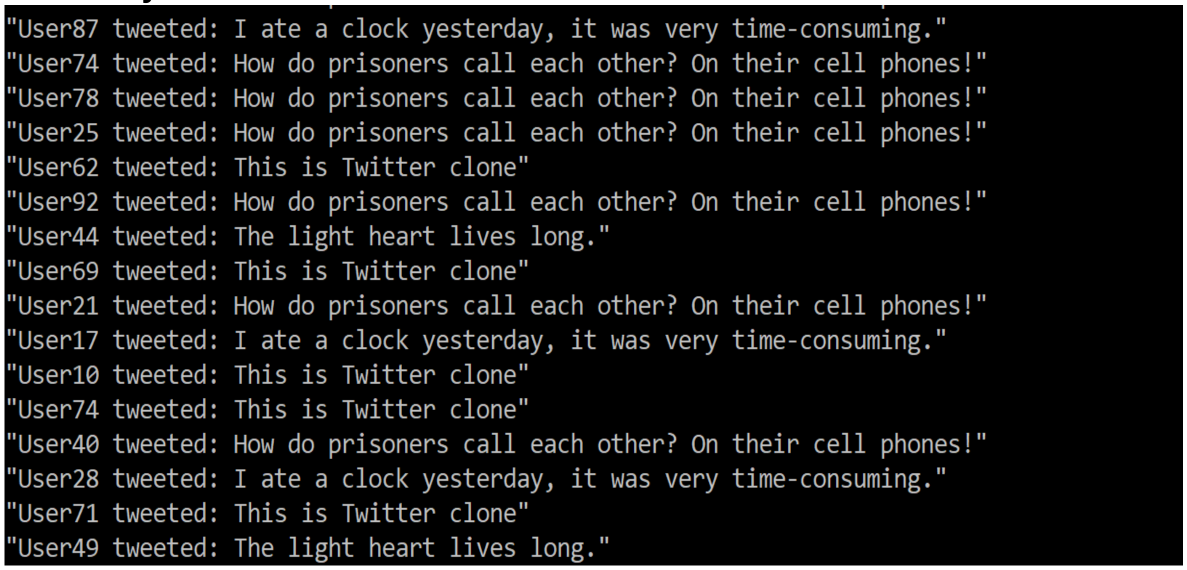
Working Project Snapshots depicting all functionalities:

1. **Starting Server and Registering Users:**



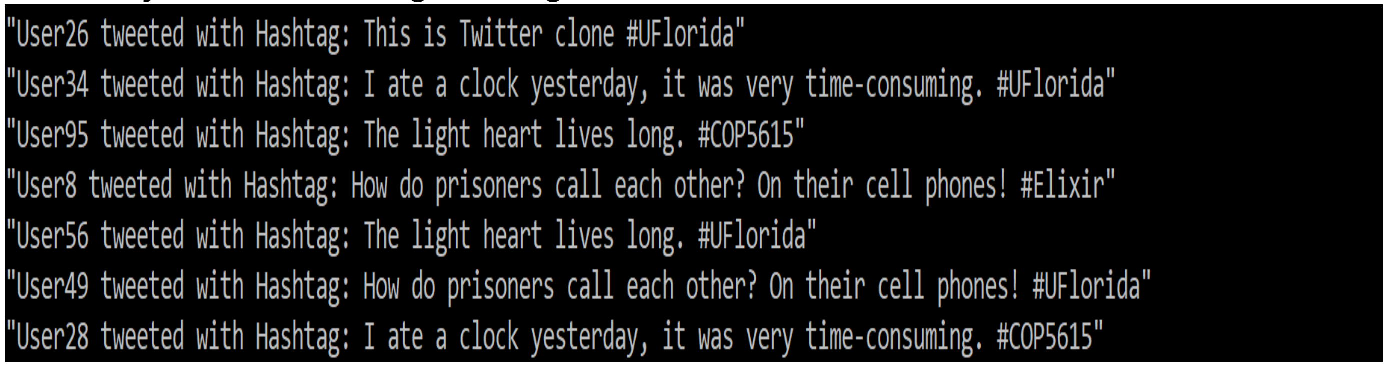
```
Server Started
User26 is now registered!
User39 is now registered!
User35 is now registered!
User41 is now registered!
User48 is now registered!
User10 is now registered!
User5 is now registered!
User8 is now registered!
User13 is now registered!
User11 is now registered!
User12 is now registered!
User15 is now registered!
User16 is now registered!
User18 is now registered!
User56 is now registered!
User62 is now registered!
User68 is now registered!
User65 is now registered!
User71 is now registered!
```

2. **Tweets by Users:**



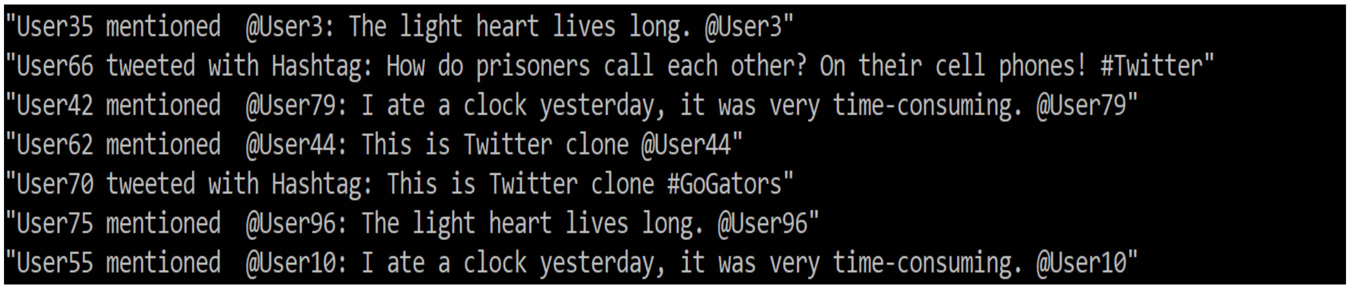
```
"User87 tweeted: I ate a clock yesterday, it was very time-consuming."
"User74 tweeted: How do prisoners call each other? On their cell phones!"
"User78 tweeted: How do prisoners call each other? On their cell phones!"
"User25 tweeted: How do prisoners call each other? On their cell phones!"
"User62 tweeted: This is Twitter clone"
"User92 tweeted: How do prisoners call each other? On their cell phones!"
"User44 tweeted: The light heart lives long."
"User69 tweeted: This is Twitter clone"
"User21 tweeted: How do prisoners call each other? On their cell phones!"
"User17 tweeted: I ate a clock yesterday, it was very time-consuming."
"User10 tweeted: This is Twitter clone"
"User74 tweeted: This is Twitter clone"
"User40 tweeted: How do prisoners call each other? On their cell phones!"
"User28 tweeted: I ate a clock yesterday, it was very time-consuming."
"User71 tweeted: This is Twitter clone"
"User49 tweeted: The light heart lives long."
```

3. **Tweets by Users containing Hashtags:**



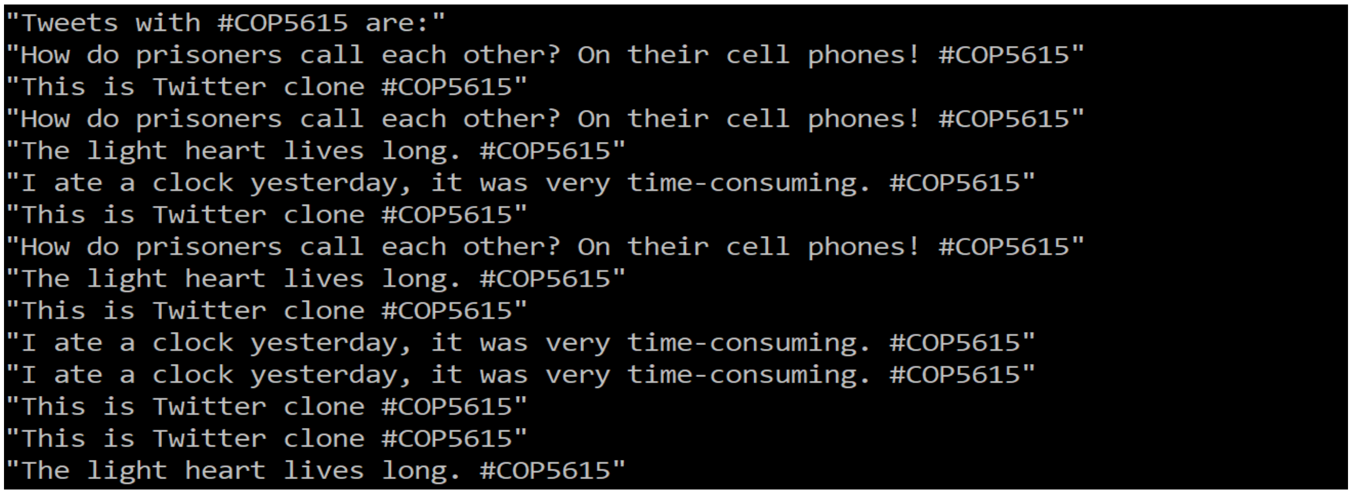
```
"User26 tweeted with Hashtag: This is Twitter clone #UFlorida"
"User34 tweeted with Hashtag: I ate a clock yesterday, it was very time-consuming. #UFlorida"
"User95 tweeted with Hashtag: The light heart lives long. #COP5615"
"User8 tweeted with Hashtag: How do prisoners call each other? On their cell phones! #Elixir"
"User56 tweeted with Hashtag: The light heart lives long. #UFlorida"
"User49 tweeted with Hashtag: How do prisoners call each other? On their cell phones! #UFlorida"
"User28 tweeted with Hashtag: I ate a clock yesterday, it was very time-consuming. #COP5615"
```

4. Tweets by Users with mentions of other users:



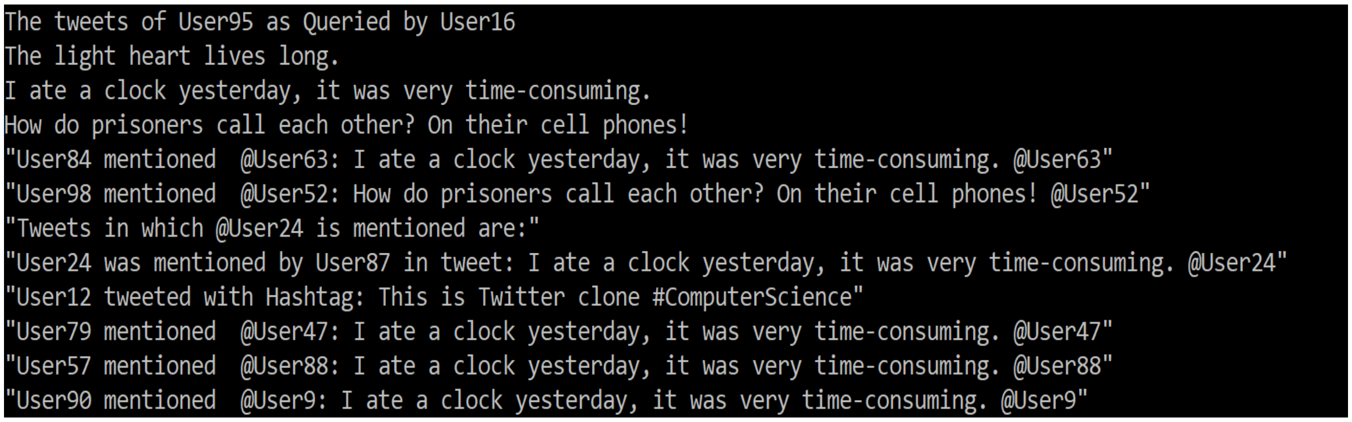
```
"User35 mentioned  @User3: The light heart lives long. @User3"
"User66 tweeted with Hashtag: How do prisoners call each other? On their cell phones! #Twitter"
"User42 mentioned  @User79: I ate a clock yesterday, it was very time-consuming. @User79"
"User62 mentioned  @User44: This is Twitter clone @User44"
"User70 tweeted with Hashtag: This is Twitter clone #GoGators"
"User75 mentioned  @User96: The light heart lives long. @User96"
"User55 mentioned  @User10: I ate a clock yesterday, it was very time-consuming. @User10"
```

5. Fetch Tweets by Querying for a Hashtag:



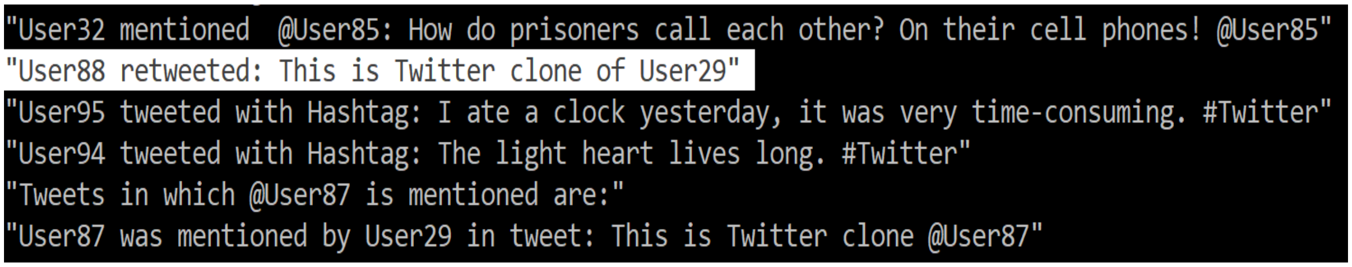
```
"Tweets with #COP5615 are:"
"How do prisoners call each other? On their cell phones! #COP5615"
"This is Twitter clone #COP5615"
"How do prisoners call each other? On their cell phones! #COP5615"
"The light heart lives long. #COP5615"
"I ate a clock yesterday, it was very time-consuming. #COP5615"
"This is Twitter clone #COP5615"
"How do prisoners call each other? On their cell phones! #COP5615"
"The light heart lives long. #COP5615"
"This is Twitter clone #COP5615"
"I ate a clock yesterday, it was very time-consuming. #COP5615"
"I ate a clock yesterday, it was very time-consuming. #COP5615"
"This is Twitter clone #COP5615"
"This is Twitter clone #COP5615"
"The light heart lives long. #COP5615"
```

6. Fetch Tweets of a User that it follows:



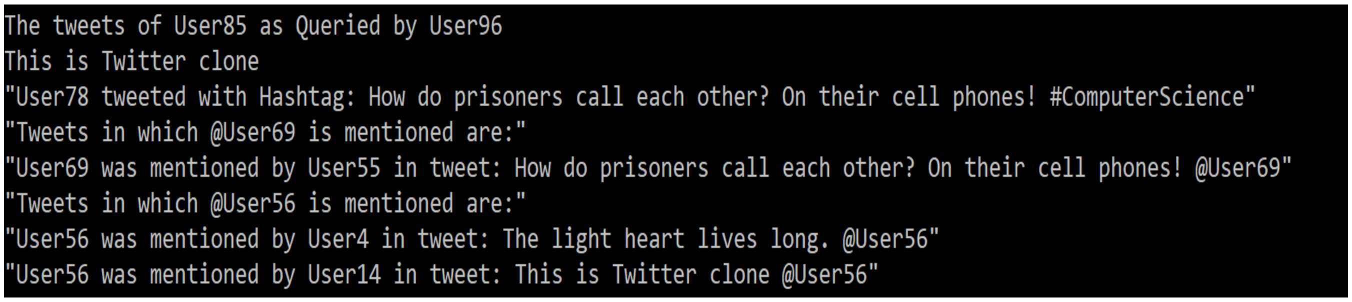
```
The tweets of User95 as Queried by User16
The light heart lives long.
I ate a clock yesterday, it was very time-consuming.
How do prisoners call each other? On their cell phones!
"User84 mentioned  @User63: I ate a clock yesterday, it was very time-consuming. @User63"
"User98 mentioned  @User52: How do prisoners call each other? On their cell phones! @User52"
"Tweets in which @User24 is mentioned are:"
"User24 was mentioned by User87 in tweet: I ate a clock yesterday, it was very time-consuming. @User24"
"User12 tweeted with Hashtag: This is Twitter clone #ComputerScience"
"User79 mentioned  @User47: I ate a clock yesterday, it was very time-consuming. @User47"
"User57 mentioned  @User88: I ate a clock yesterday, it was very time-consuming. @User88"
"User90 mentioned  @User9: I ate a clock yesterday, it was very time-consuming. @User9"
```

7. Retweet by a User:



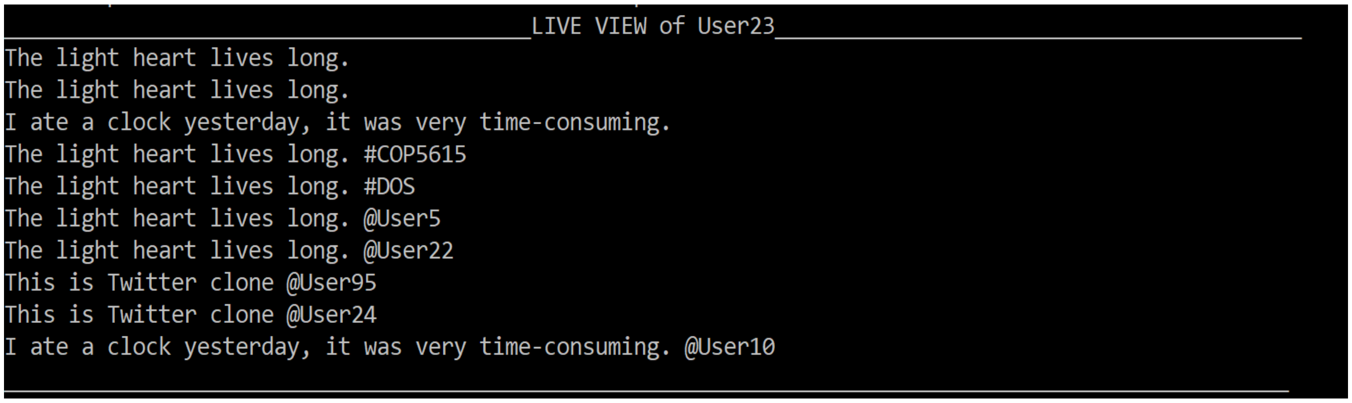
```
"User32 mentioned  @User85: How do prisoners call each other? On their cell phones! @User85"
"User88 retweeted: This is Twitter clone of User29"
"User95 tweeted with Hashtag: I ate a clock yesterday, it was very time-consuming. #Twitter"
"User94 tweeted with Hashtag: The light heart lives long. #Twitter"
"Tweets in which @User87 is mentioned are:"
"User87 was mentioned by User29 in tweet: This is Twitter clone @User87"
```

8. Fetch Tweets in which a User is mentioned:



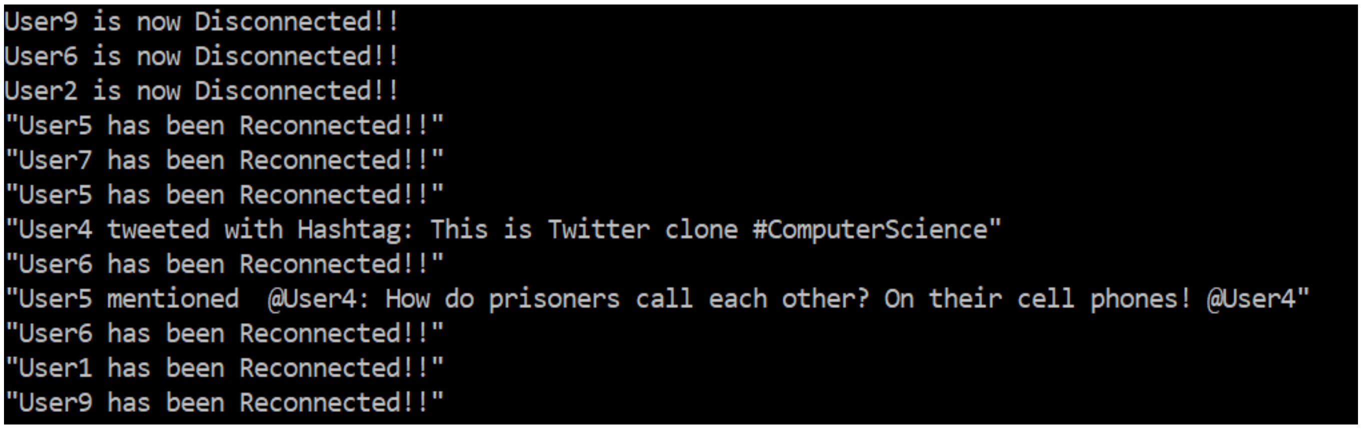
```
The tweets of User85 as Queried by User96
This is Twitter clone
"User78 tweeted with Hashtag: How do prisoners call each other? On their cell phones! #ComputerScience"
"Tweets in which @User69 is mentioned are:"
"User69 was mentioned by User55 in tweet: How do prisoners call each other? On their cell phones! @User69"
"Tweets in which @User56 is mentioned are:"
"User56 was mentioned by User4 in tweet: The light heart lives long. @User56"
"User56 was mentioned by User14 in tweet: This is Twitter clone @User56"
```

9. Live View of Tweets of a User without Querying:



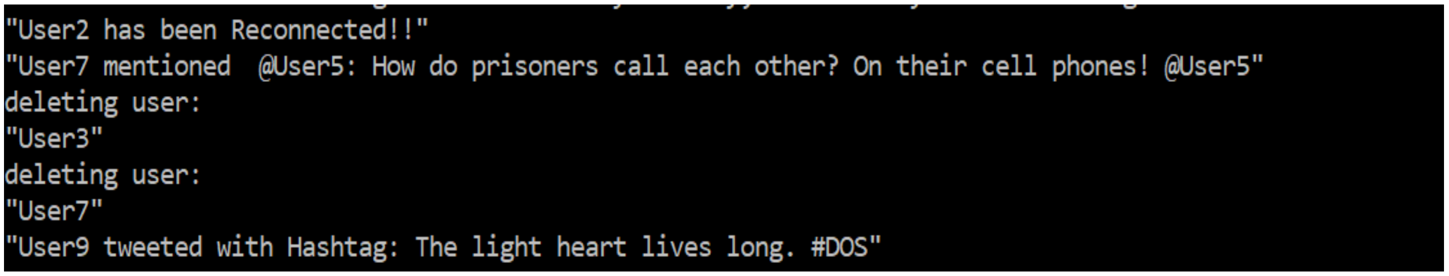
```
________________________________________LIVE VIEW of User23________________________________________
The light heart lives long.
The light heart lives long.
I ate a clock yesterday, it was very time-consuming.
The light heart lives long. #COP5615
The light heart lives long. #DOS
The light heart lives long. @User5
The light heart lives long. @User22
This is Twitter clone @User95
This is Twitter clone @User24
I ate a clock yesterday, it was very time-consuming. @User10

___________________________________________________________________________________________________
```

10. Disconnection and Reconnection of Users:



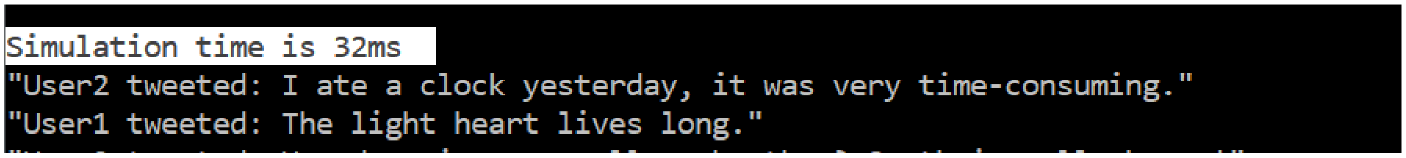
```
User9 is now Disconnected!!
User6 is now Disconnected!!
User2 is now Disconnected!!
"User5 has been Reconnected!!"
"User7 has been Reconnected!!"
"User5 has been Reconnected!!"
"User4 tweeted with Hashtag: This is Twitter clone #ComputerScience"
"User6 has been Reconnected!!"
"User5 mentioned  @User4: How do prisoners call each other? On their cell phones! @User4"
"User6 has been Reconnected!!"
"User1 has been Reconnected!!"
"User9 has been Reconnected!!"
```

11. Deletion of a User:



```
"User2 has been Reconnected!!"
"User7 mentioned  @User5: How do prisoners call each other? On their cell phones! @User5"
deleting user:
"User3"
deleting user:
"User7"
"User9 tweeted with Hashtag: The light heart lives long. #DOS"
```

12. Simulation time of the Twitter Implementation:



```
Simulation time is 32ms
"User2 tweeted: I ate a clock yesterday, it was very time-consuming."
"User1 tweeted: The light heart lives long."
```