PROJECT 4 PART 1: TWITTER CLONE

HOW TO RUN THE PROGRAM

- Save the file on your system as project4.erl
- The latest version of Erlang/OTP 25.1 is recommended.
- Run the erl command on your system to ensure it is installed correctly.
- In line 16 of the code, make sure server_node() is set to the name of the current computer. Using the convention

```
messenger + @ + USER's COMPUTER ADDRESS
```

- Move to the directory where the program is saved.
- Open 3 terminals. In one run **erl -sname messenger -setcookie atom**. In the other two run **erl -sname name -setcookie atom** where name is set to whatever you want, i.e. a and b.
- In each terminal run the command **c(project4).**
- In the messenger terminal, run **project4:start_server().** Now you are ready to use the twitter functions: register user accounts, log in users, tweets, retweets, subscriptions, and live tweet delivery. The client part (send/receive tweets) and the engine (distribute tweets) are in separate processes.
- We use multiple independent client processes that simulate thousands of clients and a single-engine process. This can be simulated with the simulator() function.

README REQUIREMENTS AND ANALYSIS

- What is working
  We created a Twitter-cloning engine with a client simulator and all the functionality of the back end of a Twitter application. The clients sending and receiving tweets and the server distributing tweets are separate processes that run in parallel. With the one engine, the program simulates thousands of clients successfully. The performance is detailed below.

- What is the largest network you managed to deal with?

Number of users	**Number of subscribers**	**Number of tweets**	**Runtime (microseconds)**
1500	**1250**	**1000**	**786149700**

We stopped at these limits because for this combination the system had reached its limit.

Description

Actor Model

The actors are users generated in the terminal and stored in a list in the engine. When they log in, this list is pulled and the password they registered, which is secured using SHA-256 encryption, is compared with the hashed password they entered to log in. If it matches, the user is able to send tweets, retweet, subscribe to users and hashtags, query the engine for tweets with particular hashtags or mentions, and receive the tweets they have subscribed to live.

Input

- Number of users (N) to designate how many clients are being generated and simulated in the engine. Below we give the performance overhead and limitations on the number of clients that can be generated.

Functions

- Register Account
  - This is done with the **reg(Username, "Password")** function where the new user is added to the User_List generated in **server(User_List)** and called upon in **start_server()**.

Server Output with the users logged on.

```
{ok,project4}
(messenger@DESKTOP-KHTE3K0)2> project4:start_server().
true
(messenger@DESKTOP-KHTE3K0)3> #{shivam => []}(messenger@DESKTOP-KHTE3K0)3> #{caroline => [],shivam => []}
(messenger@DESKTOP-KHTE3K0)3>
```

User1: Shivam registers with the encrypted password (sha256) stored for authentication.

```
{ok,project4}
(shivam@DESKTOP-KHTE3K0)2> project4:reg(shivam,"1234").
 "03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4"{register,<0.85.0>,"reg",shivam,
        "03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4"}
(shivam@DESKTOP-KHTE3K0)3>
```

User2: Caroline registers with the encrypted password (sha256) stored for authentication.

```
{ok,project4}
(caroline@DESKTOP-KHTE3K0)2> project4:reg(caroline,"5678").
 "f8638b979b2f4f793ddb6dbd197e0ee25a7a6ea32b0ae22f5e3c5d119d839e75"{register,<0.85.0>,"reg",caroline,
        "f8638b979b2f4f793ddb6dbd197e0ee25a7a6ea32b0ae22f5e3c5d119d839e75"}
(caroline@DESKTOP-KHTE3K0)3>
```

- The user can then login using **logon(Username, "Password")** function, which first checks that the user exists and then checks that the hash of the password they gave matches the registered password (both encrypted using SHA-256). Additionally the user can log off at any time using **logoff().**

User1: Shivam logon after authentication logged_on message is displayed.

```
(shivam@DESKTOP-KHTE3K0)3> project4:logon(shivam,"1234").
#{

  shivam =>
      "03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4"}true
(shivam@DESKTOP-KHTE3K0)4> logged_on
(shivam@DESKTOP-KHTE3K0)4> shivam
(shivam@DESKTOP-KHTE3K0)4>
```

User2: Caroline logon after authentication logged_on message is displayed.

```
(caroline@DESKTOP-KHTE3K0)3> project4:logon(caroline,"5678").
#{caroline =>
      "f8638b979b2f4f793ddb6dbd197e0ee25a7a6ea32b0ae22f5e3c5d119d839e75",
  shivam =>
      "03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4"}true
logged_on
(caroline@DESKTOP-KHTE3K0)4> caroline
(caroline@DESKTOP-KHTE3K0)4>
```

User1: Shivam if someone tries to logon again throws the message already logged on

```
(shivam@DESKTOP-KHTE3K0)6> project4:logon(shivam,"1234").
already_logged_on
(shivam@DESKTOP-KHTE3K0)7>
```

User1: Shivam logoff()

```
(shivam@DESKTOP-KHTE3K0)7> project4:logoff().
logoff
(shivam@DESKTOP-KHTE3K0)8>
```

- Subscribe
  - This is done using the **follow(ToName)** function which first checks that the given username exists and then adds the current user to the username's list of followers.

User2: Caroline follows User1: Shivam

User2: Caroline Follows User1: Shivam

```
(caroline@DESKTOP-KHTE3K0)4> project4:follow(shivam).
ok
(caroline@DESKTOP-KHTE3K0)5> sent
(caroline@DESKTOP-KHTE3K0)5> caroline
(caroline@DESKTOP-KHTE3K0)5>
```

Server Output: Updated Map that User2 now Follows User1

```
{ok,project4}
(messenger@DESKTOP-KHTE3K0)2> project4:start_server().
true
(messenger@DESKTOP-KHTE3K0)3> #{shivam => []}(messenger@DESKTOP-KHTE3K0)3> #{caroline => [],shivam => []} messen
ger@DESKTOP-KHTE3K0)3>
```

User1: Shivam is notified that User2: Caroline follows

```
(shivam@DESKTOP-KHTE3K0)4> #{caroline => [],shivam => []}(shivam@DESKTOP-KHTE3K0)4> shivam
(shivam@DESKTOP-KHTE3K0)4>
```

- Send Tweet
    - This is done using the **tweet(Message)** function and updates all users following this user. Users can also mention other users using the **mention()** function. This additionally updates the list of tweets from this user which can be queried in the future.

      User1: Shivam Tweets the text:Hi Hashtag:#Hello and Mentions:@caroline which is sent to all the followers

```
(shivam@DESKTOP-KHTE3K0)4>
project4:tweet("Hi #Hello @caroline").
shivam
#{caroline => [],shivam => ["Hi #Hello @caroline"]}Elem caroline
ok
(shivam@DESKTOP-KHTE3K0)5> sent
(shivam@DESKTOP-KHTE3K0)5> shivam
(shivam@DESKTOP-KHTE3K0)5>
```

      Server Output: Different process id's for both users for **distributed programming**

```
-KHTE3K0)3> list is now: [{<13450.104.0>,caroline},{<13448.104.0>,shivam}]
(messenger@DESKTOP-KHTE3K0)3>
```

      User2: Caroline Who follows User1:Shivam is notified of the tweet

```
(caroline@DESKTOP-KHTE3K0)5>        from shivam: "Hi #Hello @caroline"
(caroline@DESKTOP-KHTE3K0)5> caroline
(caroline@DESKTOP-KHTE3K0)5>
```

- Re-tweet
    - This is done using the **retweet()** function and updates the user's list of tweets as well as the followers of this user with the new tweet.
      User2: Caroline retweets that which appears with string Re: appended in front of tweet.
      Then it shows User2:Caroline's Feed Activity with her latest tweet sent and the tweet she got from User1:Shivam

```
(caroline@DESKTOP-KHTE3K0)5>project4:retweet().
caroline
"Re:Hi #Hello @caroline" -> caroline -> #{caroline => "Hi #Hello @caroline",
                                          shivam => []}caroline
#{caroline => ["Re:Hi #Hello @caroline"],shivam => ["Hi #Hello @caroline"]}ok
(caroline@DESKTOP-KHTE3K0)6>
```

      Server: Has the stored list of tweets from each user which would be displayed just in case one user logs off

```
(messenger@DESKTOP-KHTE3K0)3> #{caroline => ["Re:Hi #Hello @caroline"],shivam => ["Hi #Hello @caroline"]
}(messenger@DESKTOP-KHTE3K0)3>
```

- Mention
    - This is done using the **mention()** function which first checks that the given username exists among the tweets and then displays all the tweets in which the user is mentioned who queried.

      User2: Caroline queries all her mentions
      User2: Caroline was mentioned 2 tweets one was the tweet from User1: Shivam and the retweet() from User2: Caroline.

```
(caroline@DESKTOP-KHTE3K0)6> project4:mention().
caroline
Result: "Hi #Hello @caroline"
Result: "Re:Hi #Hello @caroline"
ok
(caroline@DESKTOP-KHTE3K0)7>
```

- Query Tweets for Hashtags
    - This is done in **search(Query)** and allows users to search by hashtag or substring of a hashtag and returns tweets that match the criteria.
      User1: Queries or searches for '#Hello' and gets 2 tweets one was the tweet from User1: Shivam and the retweet() from User2: Caroline.

```
(shivam@DESKTOP-KHTE3K0)5>
project4:search("#Hello").
Result: "Hi #Hello @caroline"
Result: "Re:Hi #Hello @caroline"
ok
(shivam@DESKTOP-KHTE3K0)6>
```

- Deliver live to connected users
    - This is done in the **message(ToName, Message)** function, where the lists of users and list of messages associated with the user who tweeted are updated. **server_transfer(From, To, Message, User_List)** carries this out on the network and sends the live message to the other user who is connected.
      User2: Caroline wants to send a message to User1: Shivam and gets confirmation of sent.

```
(caroline@DESKTOP-KHTE3K0)7> project4:message(shivam,"It's a lovely day").
ok
(caroline@DESKTOP-KHTE3K0)8> sent
(caroline@DESKTOP-KHTE3K0)8> caroline
(caroline@DESKTOP-KHTE3K0)8>
```

      Server enables the transfer by using the process id's of both the users

```
list is now: [{<13450.104.0>,caroline},{<13448.104.0>,shivam}]
```

      User1: Shivam receives the message.

```
(shivam@DESKTOP-KHTE3K0)6> Message from caroline: "It's a lovely day"
(shivam@DESKTOP-KHTE3K0)6> shivam
(shivam@DESKTOP-KHTE3K0)6>
```

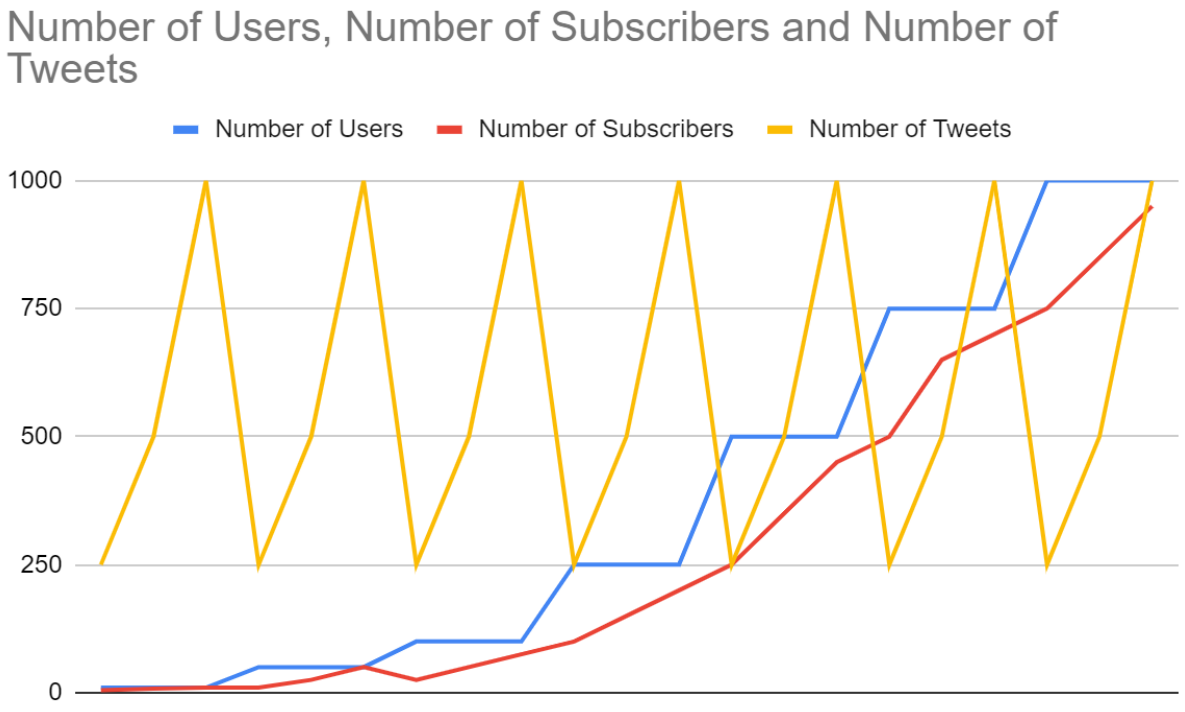
Termination of the Program
The program can be terminated at any time, as any or all users can log on or off. The engine can be terminated by exiting the server terminal.

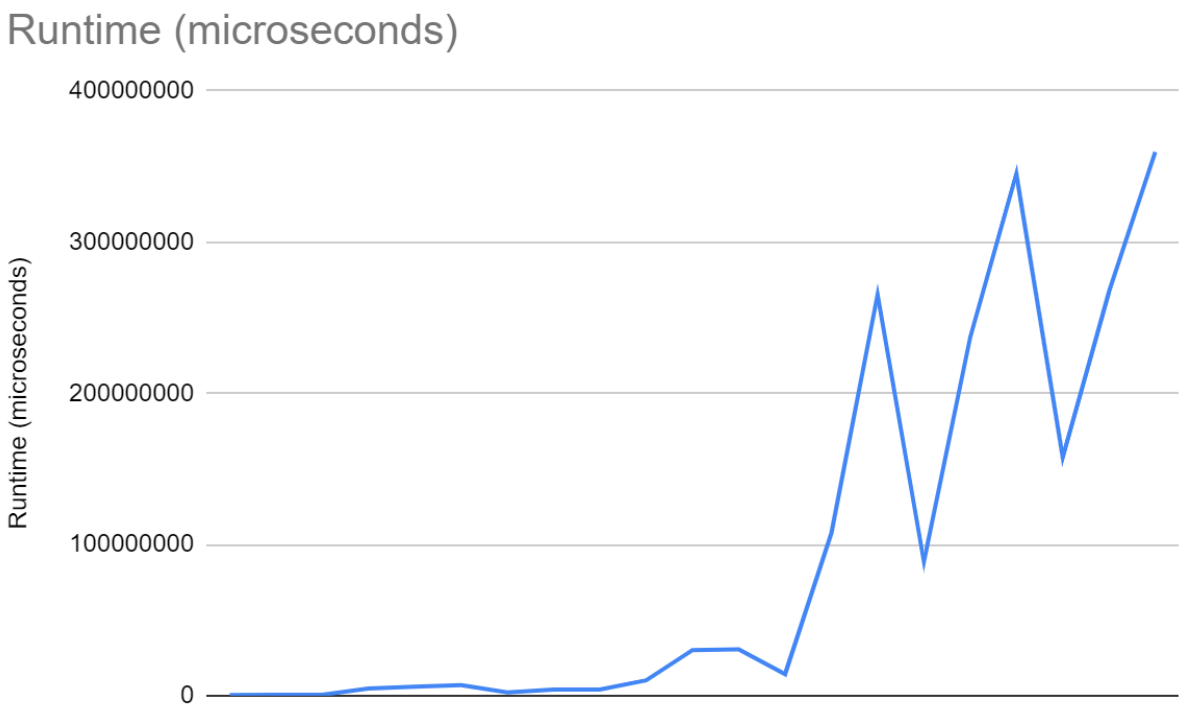
Simulator Testing
The maximum number of users we were successfully able to simulate was about 1000. Beyond this, the server cannot allocate any more memory. It returns an error “Crash dump is being written to: erl_crash.dump...done.”

Results
According to Number of Nodes

Number of Users	Number of Subscribers	Number of Tweets	Runtime (microseconds)
10	5	250	368000
10	8	500	538000
10	10	1000	679000
50	10	250	4792000
50	25	500	5961000
50	50	1000	7071000
100	25	250	2235000
100	50	500	4101000
100	75	1000	4150000
250	100	250	10182000
250	150	500	30216000
250	200	1000	30750000
500	250	250	14240000
500	350	500	107474000
500	450	1000	265152000

750	500	250	88745000
750	650	500	237127000
750	700	1000	345251000
1000	750	250	157669000
1000	850	500	267177000
1000	950	1000	359542300





Maximum values

Number of users	Number of subscribers	Number of tweets	Runtime (microseconds)
1500	**1250**	**1000**	**786149700**

Observations

- As number of tweets increases the runtime increases due to pressure on server.
- As number of Users and subscribers increase there is steady increase but not steep slope on graph.
- If number of users and subscribers are close the runtime becomes less.