# Twitter Project Part one – A simulated Engine

## Distributed Systems Project 4.1

### Team Member:

BHANU PRAKASH REDDY PALAGATI: 2623-3738; Akshith Sagar: 8788-8206

### Project Statement:

A Twitter clone should be implemented which supports *registering, login, subscribing, tweet, retweet, direct messages, query, logout* features. A simulator should be implemented, and it should perform all the Twitter operations and record the performance for performance analysis.

### How to Run Me:

This F# project has three files - twitterEngine.fsx, simulator.fsx, and dataDefnitions.fs.

First, you have to run twitterEngine.fsx because this acts as a server.

"*dotnet fsi twitterEngine.fsx*" this will run the server

Then you have to run the simulator with the number of users as an argument

"*dotnet fsi simulator.fsx 1000*" this will run the simulator with 1000 users.

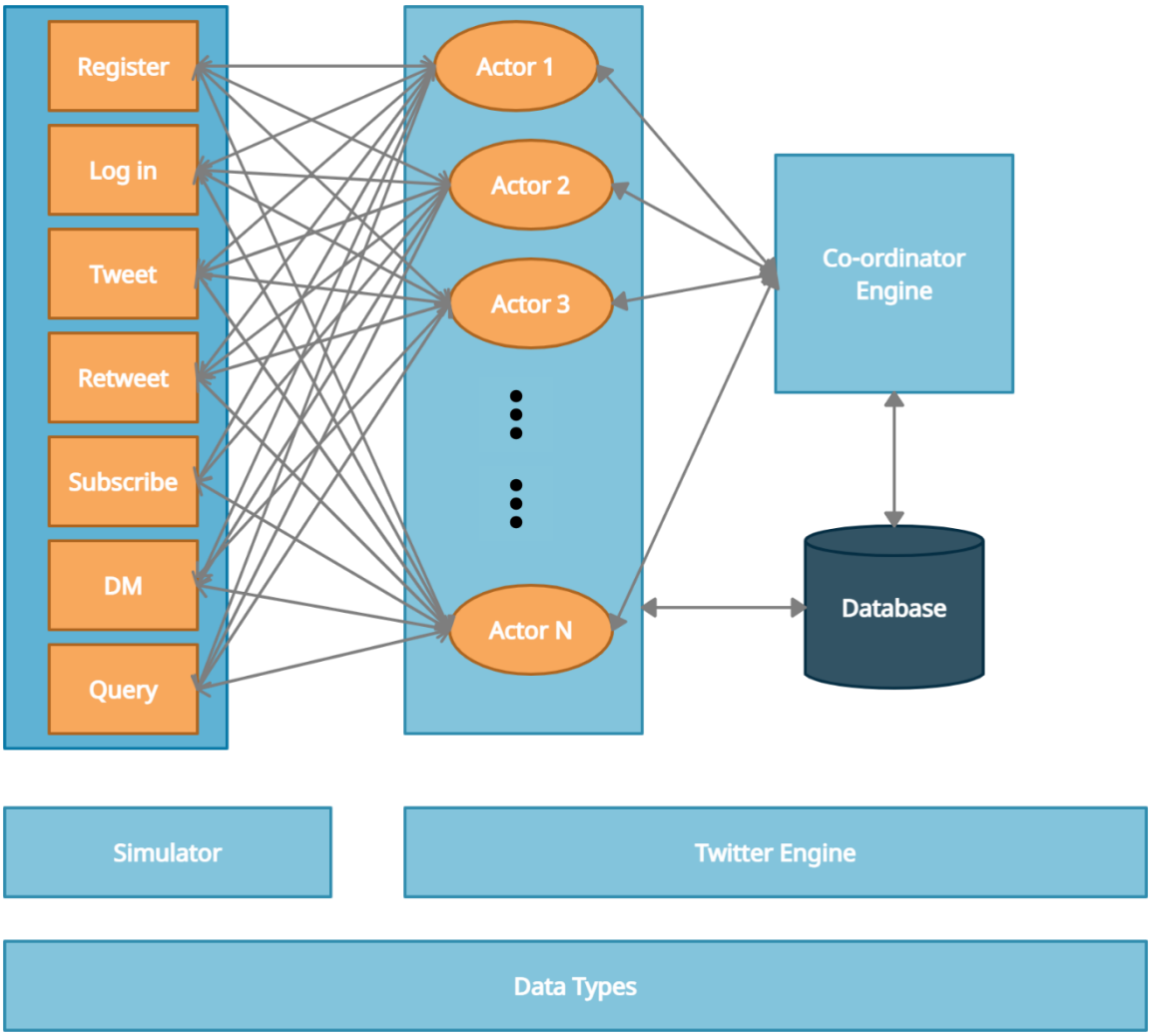
twitterCore instance running

```
E:\UF\DOS\Projects\Twitter4.1>dotnet fsi twitterEngine.fsx
C:\Users\bhanu\AppData\Local\Temp\nuget\17936--ef8dd635-5045-41aa-bd3f-1993df3db964\Project.fspro
j : warning NU1605: Detected package downgrade: FSharp.Core from 6.0.1 to 5.0.0. Reference the pa
ckage directly from the project to select a different version.
C:\Users\bhanu\AppData\Local\Temp\nuget\17936--ef8dd635-5045-41aa-bd3f-1993df3db964\Project.fspro
j : warning NU1605:  Project -> Akka.FSharp 1.4.28 -> FSharp.Core (>= 6.0.1)
C:\Users\bhanu\AppData\Local\Temp\nuget\17936--ef8dd635-5045-41aa-bd3f-1993df3db964\Project.fspro
j : warning NU1605:  Project -> FSharp.Core (>= 5.0.0)
C:\Users\bhanu\AppData\Local\Temp\nuget\17936--ef8dd635-5045-41aa-bd3f-1993df3db964\Project.fspro
j : warning NU1605: Detected package downgrade: FSharp.Core from 6.0.1 to 5.0.0. Reference the pa
ckage directly from the project to select a different version.
C:\Users\bhanu\AppData\Local\Temp\nuget\17936--ef8dd635-5045-41aa-bd3f-1993df3db964\Project.fspro
j : warning NU1605:  Project -> Akka.FSharp 1.4.28 -> FSharp.Core (>= 6.0.1)
C:\Users\bhanu\AppData\Local\Temp\nuget\17936--ef8dd635-5045-41aa-bd3f-1993df3db964\Project.fspro
j : warning NU1605:  Project -> FSharp.Core (>= 5.0.0)
[INFO][28-11-2021 23:01:10][Thread 0001][remoting (akka://remote-system)] Starting remoting
[INFO][28-11-2021 23:01:10][Thread 0001][remoting (akka://remote-system)] Remoting started; liste
ning on addresses : [akka.tcp://remote-system@192.168.0.97:9009]
[INFO][28-11-2021 23:01:10][Thread 0001][remoting (akka://remote-system)] Remoting now listens on
 addresses: [akka.tcp://remote-system@192.168.0.97:9009]
```

Simulator instance running with the response of the 30,000 users

```
[INFO][28-11-2021 23:01:30][Thread 0001][remoting (akka://twitterEngine)] Starting remoting
[INFO][28-11-2021 23:01:30][Thread 0001][remoting (akka://twitterEngine)] Remoting started; li
tening on addresses : [akka.tcp://twitterEngine@192.168.0.97:9010]
[INFO][28-11-2021 23:01:30][Thread 0001][remoting (akka://twitterEngine)] Remoting now listens
on addresses: [akka.tcp://twitterEngine@192.168.0.97:9010]
Performed the register in 985.7821 ms
Performed the login in 766.7263 ms
Performed the subscription in 1644.95 ms
Performed 75217 tweets in 3356.5534 ms
Performed Retweets in 6201.8597 ms
Performed the DMS in 695.111 ms
Performed the query in 119967.9294 ms
Performed the logout in 812.0256 ms
All actions were performed in 134479.8472 ms
```

## Engine and Simulator Interaction Architecture:




## How Twitter Simulator Implemented:

After starting the twitterEngine we will start the simulator while doing so we mention the number of users we want to simulate in this execution. Upon getting this information the main actor in the simulator requests the twitterEngine to generate as many user actors as specified by the users.

Once the actors were generated the main actor starts the simulation process and asks the coordinator to register each user. Upon getting the register request the user actor will store the username and hashed password as properties. This helps us to authenticate if the user is revisiting. Once the registration is done login will be performed by all the users by the simulator actor. For this username and hashed password will be sent as parameters for the user actors these will be compared with the username and password that is stored while registering the account.

After login simulator will start subscribing to different users. Each time there is a subscription request a list will be maintained to store who all are we are following, similarly each time we subscribe to a user we will let that user know that we are following him. So, we have both followers and the following list. Whenever we make a new tweet, we will notify all the users that are following us. Similarly, we can expect a notification if any of the users that are on the subscribers' list make a tweet. To simulate

a fair real-world subscription, a Zipf distribution library is used. Please use this link to get more information about Zipf distribution https://en.wikipedia.org/wiki/Zipf%27s_law. In a nutshell, rather than giving the same number of subscribers to everyone we will use assign subscribers with a probabilistic model.

Once we have followers and the following information the main simulator actor will start tweeting. Since we are creating a simulation, we will auto-generate tweets. So, we will randomly pick a number between 1 to 5 and assign those many hashtags and mentions randomly. For hashtags, we have a common hashtags list from this we will pick hashtags randomly. For mentions, we will randomly pick some users from available users. Then a message will be constructed with the entire tweet, mentions, hashtags, tweetedby, and this message will be registered with a tweetId, this acts as a primary to store in the database. Whenever a user receives a tweet, it will be propagated to the coordinator with all mentions and subscription lists and then the coordinator will send those tweets to those users as notifications.

Once the tweets were sent the simulator will randomly pick some tweets and resend these as retweets. If a retweet was made, then the engine will not notify the mentioned users again. However, it will notify all the followers of the retweeted user and call it a retweet.

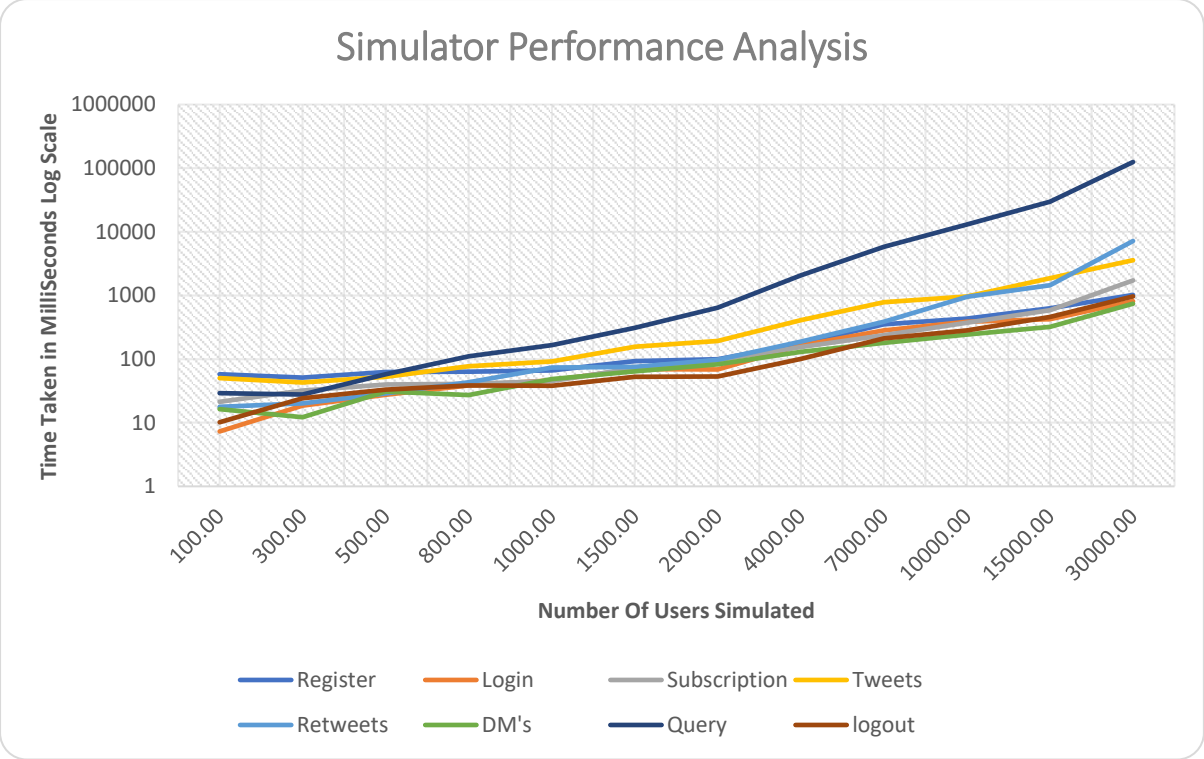
Then the simulator will send a message from one user to a randomly picked user this service is called a Direct message. Then the simulator will randomly perform a query based on mention or hashtag by all the users and check the response from those queries. After this the simulator is done with all the required simulations so, it will perform the final simulation that is making the user log out of the system.

Before starting each simulation, a timer will be started whenever an engine performs its action successfully it will inform the sender that the action is a success. Once all the request in a simulation responds with a success, we will call that the operation is successful and measure the time taken by all the users for that action. Then the simulator will continue with the same process for all the simulations. Before starting the first simulation another timer will be started and it will end after performing all the simulations successfully. This will give us the total time taken for the simulation. These timers allow us to perform analysis on the run time of various tasks.

## Performance Analysis:

This table depicts the time taken by each operation for the given users in ***milliseconds***.

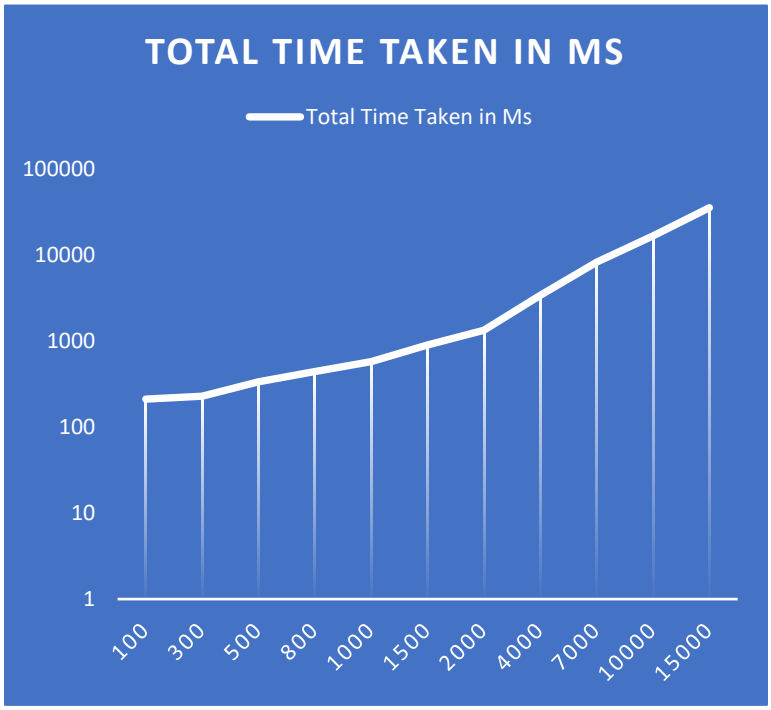
| Users | Register | Login | Subscription | Tweets | Retweets | DM's | Query | logout |
|---|---|---|---|---|---|---|---|---|
| 100 | 58.0356 | 7.2948 | 21.3223 | 50.491 | 17.7733 | 16.3238 | 29.178 | 10.1685 |
| 300 | 51.1596 | 18.4496 | 31.8891 | 42.86 | 20.1378 | 12.1973 | 27.7266 | 24.104 |
| 500 | 62.7622 | 27.5753 | 39.6588 | 52.4288 | 28.8571 | 31.038 | 57.9063 | 33.238 |
| 800 | 63.3537 | 38.6381 | 40.9983 | 76.8569 | 43.0113 | 27.2393 | 110.0633 | 38.3637 |
| 1000 | 66.4462 | 46.1559 | 45.2541 | 91.5452 | 73.6211 | 48.7638 | 165.9988 | 38.1253 |
| 1500 | 92.3043 | 69.4254 | 72.0072 | 157.1324 | 75.1607 | 64.4019 | 311.3844 | 52.377 |
| 2000 | 99.3395 | 68.8677 | 96.582 | 193.2088 | 97.0083 | 82.9837 | 642.2028 | 53.228 |
| 4000 | 173.2977 | 163.3014 | 155.0122 | 410.9715 | 188.8394 | 130.3488 | 2071.988 | 100.1023 |
| 7000 | 351.0098 | 281.7944 | 237.438 | 779.7531 | 386.6611 | 179.6463 | 5822.4 | 210.5671 |
| 10000 | 434.1036 | 384.2937 | 374.0536 | 960.6726 | 955.972 | 241.7039 | 13002.1 | 281.3114 |
| 15000 | 627.4798 | 422.1877 | 584.8529 | 1864.811 | 1438.548 | 322.1843 | 29553.05 | 461.2571 |
| 30000 | 1018.34 | 811.8216 | 1720.719 | 3577.659 | 7163.723 | 741.9882 | 124959.2 | 965.4268 |




## Total Time Performance Analysis:

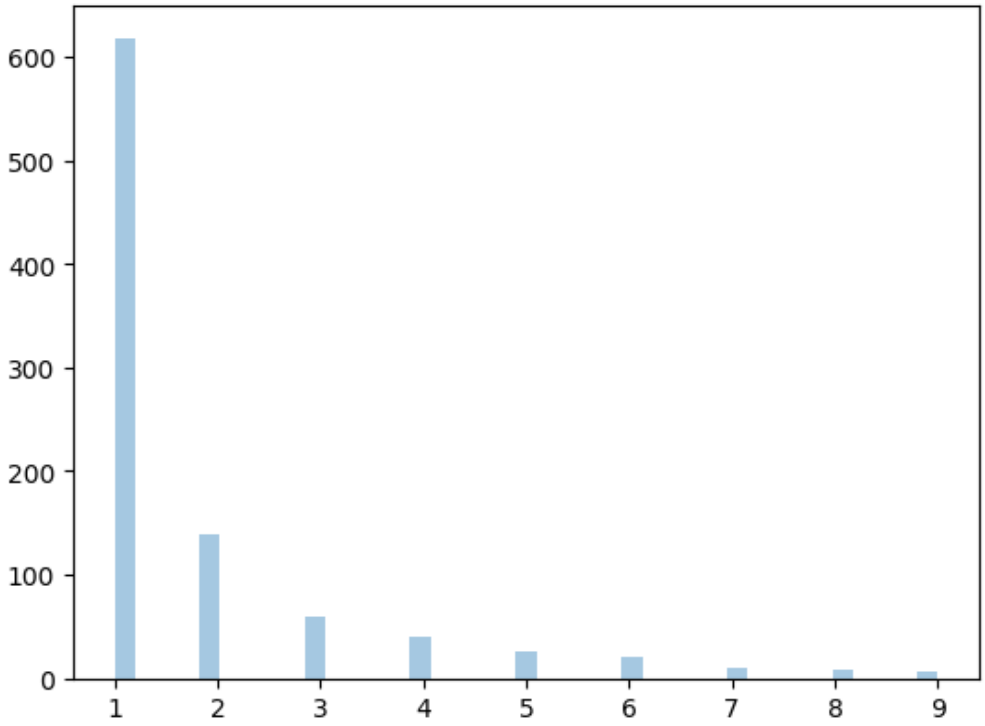
This table depicts the time taken by all the operations in aggregation for the given users in ***milliseconds***.

| Users | Total Time Taken in Ms |
|---|---|
| 100 | 210.5873 |
| 300 | 228.524 |
| 500 | 333.4645 |
| 800 | 438.5246 |
| 1000 | 575.9104 |
| 1500 | 894.1933 |
| 2000 | 1333.4208 |
| 4000 | 3393.8613 |
| 7000 | 8249.2697 |
| 10000 | 16634.2132 |
| 15000 | 35274.3693 |
| 30000 | 140958.8284 |




The above graph shows the graph between the number of users and the total time taken by the engine to perform all the operations. The time is recorded in milliseconds and the y-axis is on a log scale but the x-axis is on a normal scale.

## Sample Graph for Zipf Distribution:




## What are all Achieved In this Project:

- Register
- Login
- Tweet with mentions and hashtags
- ReTweet using tweetId
- Tweet Notifications to Subscribers and mentioned users
- DM
- Query on message, mentions, and hashtags
- Subscription
- Logout
- Zipf Distribution for Subscribers and Tweets
- Separate processes for the client and the engine and connected through AKKA remote facility.
- Simulated 30,000 users

## Bonus Achievement:

Stored password hash instead of raw passwords using SHA256. This adds the cryptographic flavor to the simulation which is a bonus part of the project.