

Readme

Team:

Tao Zhang UF-ID:7636-6624

Haowen Chen UF-ID: 8141-1485

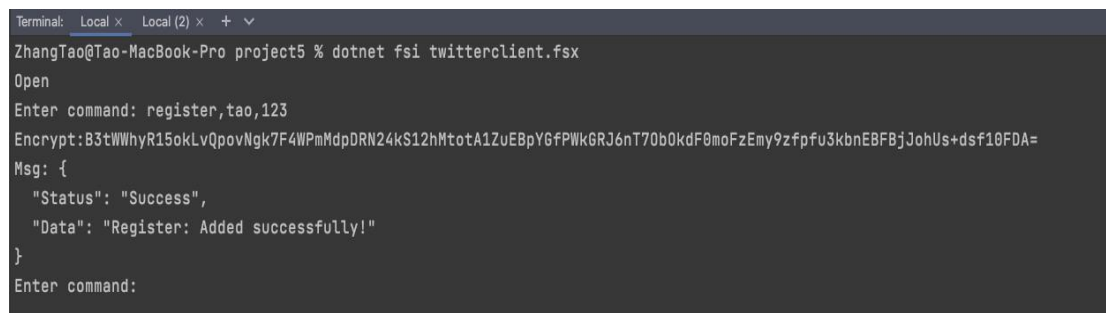
YouTube:<https://youtu.be/i893DZNs4hw>

Project Description:

In this project we have used Suave web framework to implement a WebSocket interface to our part I implementation, the twitter clone we have implemented supports register an account, login using username and password, send tweets, subscribe to user's tweets, do retweets of user's tweets, and deliver all these functions live without querying for logged in users, we have designed a JSON based API that represents all messages and their replies(including errors).

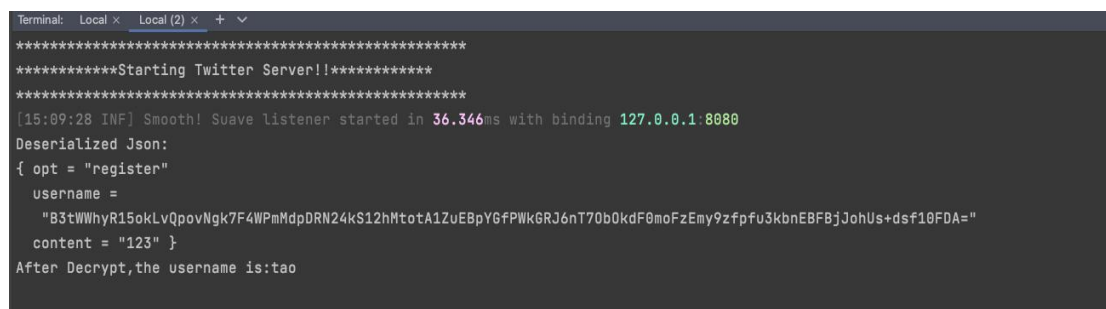
Bonus(256-bit ElipticCurve):

We used the 256-bit ElipticCurve algorithm to encrypt the registered user name on the client side, and then deserialized and decrypted it on the server side to get the user name. The result is shown in the figure below.



```
Terminal: Local x Local (2) x + v
ZhangTao@Tao-MacBook-Pro project5 % dotnet fsi twitterclient.fsx
Open
Enter command: register,tao,123
Encrypt:B3tWWHyR15okLvQpovNgk7F4WPmMdpDRN24kS12hMtota1ZuEBpY6fPWkGRJ6nT70b0kdF0moFzEmy9zfpu3kbnEBFBjJohUs+dsf10FDA=
Msg: {
  "Status": "Success",
  "Data": "Register: Added successfully!"
}
Enter command:
```

Client



```
Terminal: Local x Local (2) x + v
*****Starting Twitter Server!*****
[15:09:28 INF] Smooth! Suave listener started in 36.346ms with binding 127.0.0.1:8080
Deserialized Json:
{ opt = "register"
  username =
    "B3tWWHyR15okLvQpovNgk7F4WPmMdpDRN24kS12hMtota1ZuEBpY6fPWkGRJ6nT70b0kdF0moFzEmy9zfpu3kbnEBFBjJohUs+dsf10FDA="
  content = "123" }
After Decrypt,the username is:tao
```

Server

How to run:

references.fsx This is a required document.

AseEncryption.fsx:This is the **bonus** part, encrypt the registered user name.

Firstly open Server:dotnet fsi --langversion:preview Server.fsx

Secondly open Client:dotnet fsi --langversion:preview Client.fsx

N:Number of users

Implementation:

Server:

1. WebSocket:

The server is based on the Suave web framework and once it is triggered it will start the server on localhost:8080 and will be waiting to get connection requests from WebSocket clients. Once the connection is made it has the WebSocket and context details. For each WebSocket, we will be creating an actor and as the socket receives the message is forwarded to its actor and the actor reads the received JSON object and based on the request URL it deserializes the body into corresponding type record and based on the request URL it also forwards the deserialized object to corresponding actors which in turn performs the requested task and sends the response back to WebSocket actor.

2. actor:

- (1) register_actor: This Actor is responsible to register username and password.
- (2) login_actor: This Actor is responsible to log in with a registered username and password.
- (3) sendTweet_actor: This Actor is responsible to send tweets to other users.
- (4) subscribe_actor: This Actor is responsible to subscribe other users.
- (5) retweet_actor: This Actor is responsible to retweet other users tweet.
- (6) querySubscribe_actor: This Actor is responsible to query about what users subscribe to a user.
- (7) queryHashtag_actor: This Actor is responsible to query special Hashtag.
- (8) queryMention_actor: This Actor is responsible to query what content the user mentioned.

Client:

(1) **Client WebSocket:** In the client, we are using ClientWebSocket class for dotnet. The client initially creates a web socket connection to the server URL. Once the socket is created, we take input from the console and serialize it and send it to the server along with this task there will be another task running parallelly on the client-side for receiving the response. As it receives the responses it deserializes them and formats and prints the response on the console.

(2) Zipf_subscribe: Each client is ranked from 1-N where N is the number of users. Each client will make $1/\text{rank}$ number of tweets per millisecond to the server. number of subscribes, simulate a Zipf distribution.



(4)connect and disconnect: By default, all users are connected. There is a 10% probability that they will not be connected. The tags and mentions of users who are not connected will not exist, and other users who subscribe to him will not see him online.

Server:

Client:

```
Open
Enter command: register,chen,123
Msg: {
  "Status": "Success",
  "Data": "Register: Added successfully!"
}
```

```
Enter command: register,chen,1234
Msg: {
  "Status": "Error",
  "Data": "Register: Username already exists!"
}
```

```
Enter command: login,chen,123
Msg: {
  "Status": "Success",
  "Data": "success,login successfully"
}
```

```
Enter command: login,cheng,123
Msg: {
  "Status": "Error",
  "Data": "you do not have a account, plz sign up"
}
```

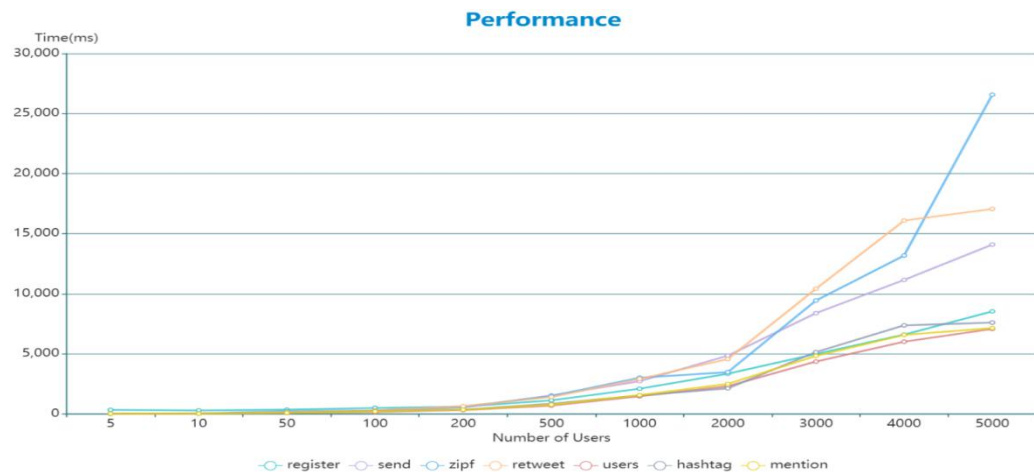
```
Enter command: login,chen,1234
Msg: {
  "Status": "Error",
  "Data": "username or password is not correctness"
}
```

```
Enter command: send,Sup bro #yolo @tao
Msg: {
  "Status": "Success",
  "Data": "Sent:Sup bro #yolo @tao "
}
Enter command: subscribe,chen,tao
Msg: {
  "Status": "Success",
  "Data": "tao now is following chen"
}
Enter command: retweet,tao,good
Msg: {
  "Status": "Success",
  "Data": "reTweet:tao taoSup bro #yolo @tao "
}
Enter command: retweet,chen,sup
Msg: {
  "Status": "Error",
  "Data": "this tweet maybe not exist"
}
Enter command: querysubscribe,tao
Msg: {
  "Status": "Success",
  "Data": "chen "
}
Enter command: query#, #yolo
Msg: {
  "Status": "Success",
  "Data": "Sup bro #yolo @tao \nSup bro #yolo @tao "
}
Enter command: query@, @tao
Msg: {
  "Status": "Success",
  "Data": "Sup bro #yolo @tao \nSup bro #yolo @tao "
}
Enter command: exit
baowenchen@haowendeMacBook-Pro: Proj 4-2 %
```

```
[14:45:46 INF] WebSocket disconnected: ConnectionError "Connection reset by peer"
```

Performance:

users	register	send	zipf	retweet	subscribe	hashtag	mention
5	329.3377	18.2400	12.2065	15.5313	6.3305	8.9978	14.3804
10	282.0952	33.1688	27.6950	32.7091	19.4049	15.0906	17.6283
50	343.0404	136.3510	213.9374	142.1209	78.2160	75.9960	79.1146
100	491.9659	270.3742	298.7337	294.5574	147.3760	230.2821	196.8646
200	598.8412	528.3827	594.8456	634.3038	308.4170	342.1422	340.9901
500	1124.9816	1529.5842	1490.7523	1397.0603	685.2494	837.6207	766.9667
1000	2089.2842	2716.4888	3010.5645	2905.7171	1459.2135	1531.3520	1559.0295
2000	3339.0786	4843.0530	3469.6742	4556.9809	2297.7820	2116.2172	2493.1995
3000	4987.1042	8383.3438	9437.7915	10412.5130	4351.1768	5143.2575	4822.3235
4000	6600.2778	11153.1924	13180.6067	16095.5663	6007.4597	7367.5819	6575.7702
5000	8530.7216	14097.9365	26572.4532	17060.1862	7070.4842	7596.6053	7140.3560



What is the largest network you managed to deal with:

The biggest number of users we tested is 5000.