

## **Project 4 Part 2-TWITTER SIMULATOR AND ENGINE**

NAME Jyotik Parikshya

UFID 0577-1395

NAME Daye Kim

UFID 8971-8937

### **Specifications:**

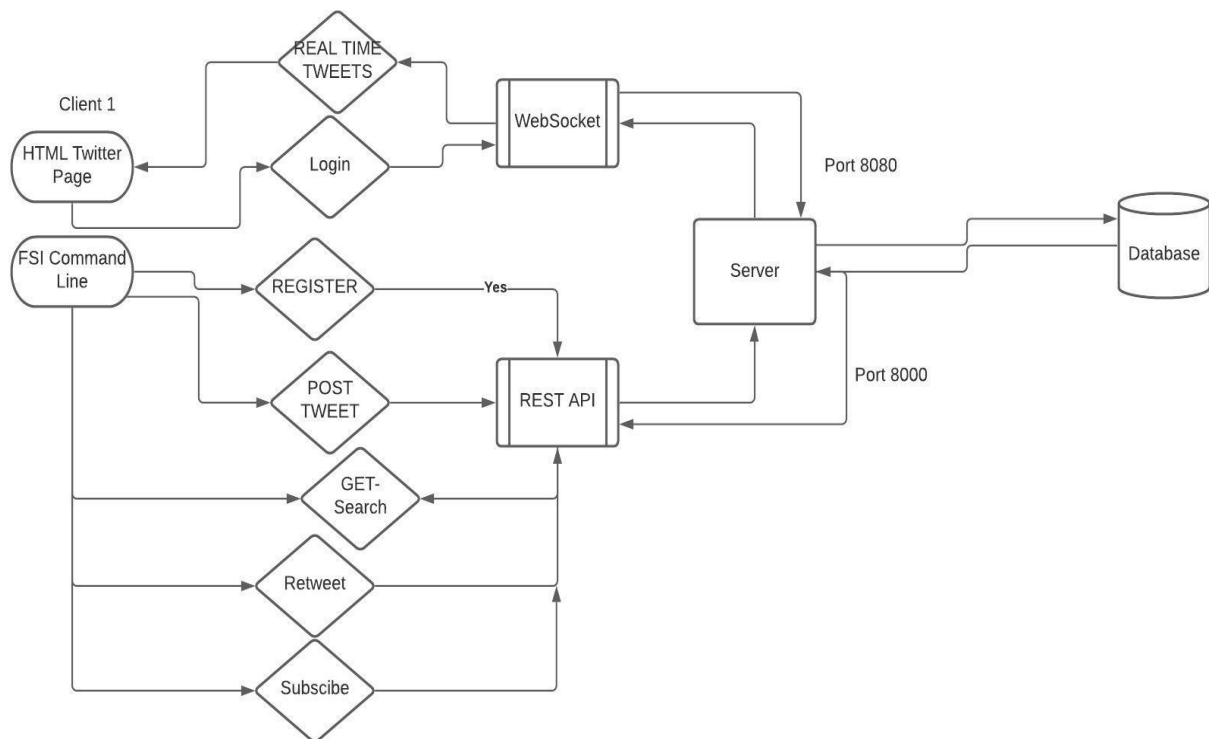
#### **Client**

- Each client has been simulated with 2 integrated interfaces.
- First is “The TWITTER WALL” which is a HTML page showing real time tweets received by the user through its subscribers. There is also an option of logging in on the interface.
- Second is “FSI Interface which is used for Register, Tweet, Retweet, Subscribe, Search by Mention, Search by Hashtag, Search tweets by subscribers on command prompt.
- Multiple Clients have been simulated and both the interfaces are integrated working for the same server on different async units.
- This has been done to show clear difference between functionality of REST API and Web Sockets.

#### **Server**

- Server represents a Twitter engine
- Server has two of its ports active. One for web sockets and another for REST API.
- Two separate Actors are responsible for running each of these processes.
- A JSON based API was implemented for both REST and WebSocket.

**Below is the Overview of the architecture of the Program:**



### **Communication Protocols:**

This program runs on REST API and WebSocket API, and Client-Server connections are based on these two API according to the situations. It runs with Suave which is a simple web development F# library providing a lightweight web server and a set of combinators to manipulate route flow and task composition. Also, HTTPFs Library is used to implement REST APIs.

### **JSON BASED API:**

User (Client) registers (sign up) and creates Tweet and subscribes other users on REST API. While server and users are connected, Server (Engine) stores the user's information and tweet with user's name. Each tweet consists of user who posts the tweet, message, mention, and hashtag topic.

## 1) REST Interfaces

A) [GET] search/mention/{id}	B) [GET] subscribes/{id}
Input: String Id	Input: String Id
Output: List of Tweets (or Empty JSON if no result)	Output: List of Subscriber Ids and subscribed ids(Strings) in JSON format
<pre>namespace MyWebApi  open System  [&lt;CLIMutable&gt;] type Tweet =     { From: string       Mention: string       Hashtag: string       Message: string       id:String     }</pre>	<pre>namespace MyWebApi  open System  [&lt;CLIMutable&gt;] type Subscribe =     { subscriberId: string       subscribee: string     }</pre>

C) search/hashtag/{id}	D) search/subscribes/{id}
Input: String Id (Hashtag to search)	Input: String Id (Fetches all the subscribers of this id and returns all Tweets by all those subscribers)
Output: List of Tweets	Output: List of Tweets
<pre>namespace MyWebApi  open System  [&lt;CLIMutable&gt;] type Tweet =     { From: string       Mention: string       Hashtag: string       Message: string       id:String     }</pre>	<pre>namespace MyWebApi  open System  [&lt;CLIMutable&gt;] type Tweet =     { From: string       Mention: string       Hashtag: string       Message: string       id:String     }</pre>

E) [POST] subscribe	F) [POST] tweet
Input: Subscribe JSON Object	Input: Subscribe JSON Object
Output: Boolean Success or Fail	Output: Boolean Success or Fail
<pre> namespace MyWebApi  open System  [&lt;CLIMutable&gt;] type Subscribe =     { subscriberId: string       subscribee: string     } </pre>	<pre> namespace MyWebApi  open System  [&lt;CLIMutable&gt;] type Tweet =     { From: string       Mention: string       Hashtag: string       Message: string       id:String     } </pre>

G)[POST] retweet	H) [POST]registration
Input: ReTweet JSON Object (From: Who is retweeting? Id: Tweet Id from the particular Tweet Object )	Input: User JSON Object
Output: Boolean- Success or Fail	Output: Boolean- Success or Fail
<pre> namespace MyWebApi  open System  [&lt;CLIMutable&gt;] type ReTweet =     { From:String       id:String     } </pre>	<pre> namespace MyWebApi  open System  [&lt;CLIMutable&gt;] type User =     { Id: string       Fname: string       Lname: string       Password: string     } </pre>

## 2)Web Socket API

A) Login	B) Tweet Read
Input: String Login ID  Output: String "Logged in" or "Logged in Failed"	After Login, WebSocket API sends new subscribed tweet every 5 seconds to the client while they are logged in.  Output: List of Tweet JSON Object  <pre>namespace MyWebApi open System  [&lt;CLIMutable&gt;] type Tweet =     { From: string       Mention: string       Hashtag: string       Message: string       id:String     }</pre>

**On Error:** If there are no Query Results then the program returns empty string and is handled on the client side.

### Client UI and Command Line:

#### 1) Command Line:

Users write input on the command prompt according to the request. Command Line shows the options of actions/functions the user wants to operate.

**A) Register** - [User ID: String, First name: String, Last name: String, Password: String]

This is for registration and this registered information is stored in engine. The user ID is the key which other users subscribes to and the user ID is used for searching to fetch the tweets.

**B) Tweet** - [From: String, Mention: String, Hashtag: String, Message: String, ID: String]

Each element is the key to search tweets. The element 'FROM' is the user ID who creates the tweets and the element 'ID' is the distinct key for each tweet.

**C) Subscribe** - [SubscriberID : String, Subscribee : String]

SubscribetID is the user Id and subscribee is the other user's Id to be subscribed.

**D) Retweet** - [From: String, Id: String]

In this program, the user can choose the tweet id to retweet, and this retweeted tweet messages will be stored in the user's tweet list. This retweeted tweet will be shown with original tweets from the same user to the other uses interface

**E) Search** - There are three elements to search the tweets [UserId, Hashtag, Mention]

When the user select UsesrId for searching keyword, the command prompt will show the subscribed IDs by the currently logged in user.

## 2) Browser user Interface

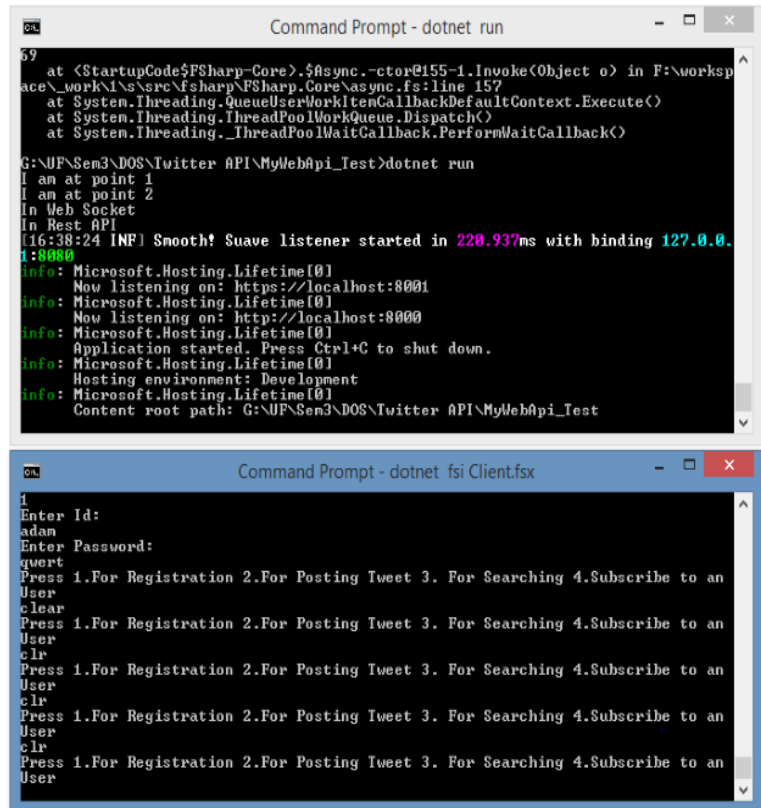
**A) log in** – The user can log in after writing the user id on the input box and click the 'log in' button. This Id is sent to the engine by 'POST' and the engine checks If there is registration data matching with the Id.

**B) Tweet** - The user successes to log in, the browser shows all tweets of other users who this user subscribed to. These tweets are updated in real-time. When other users who the currently logged in on WebSocket subscribed to makes new tweet on the command prompt, the new tweet will be shown on the browser (WebSocket Interface)

## User Interfaces:

### My Twitter Page

Login ID:



```
Command Prompt - dotnet run
69
at <StartupCode$FSharp-Core>.$Async.-ctor@155-1.Invoke(Object o) in F:\worksp
ace\work\1\src\FSharp\FSharp.Core\Async.fs:line 157
at System.Threading.QueueUserWorkItemCallbackDefaultContext.Execute()
at System.Threading.ThreadPoolWorkQueue.Dispatch()
at System.Threading._ThreadPoolWaitCallback.PerformWaitCallback()

G:\JF\Sen3\DOS\Twitter API\MyWebApi_Test>dotnet run
! an at point 1
! an at point 2
In Web Socket
In Rest API
[16:38:24 INF] Smooth! Suave listener started in 220.937ms with binding 127.0.0.
1:8000
Info: Microsoft.Hosting.Lifetime[0]
Now listening on: https://localhost:8001
Info: Microsoft.Hosting.Lifetime[0]
Now listening on: http://localhost:8000
Info: Microsoft.Hosting.Lifetime[0]
Application started. Press Ctrl+C to shut down.
Info: Microsoft.Hosting.Lifetime[0]
Hosting environment: Development
Info: Microsoft.Hosting.Lifetime[0]
Content root path: G:\JF\Sen3\DOS\Twitter API\MyWebApi_Test

Command Prompt - dotnet fsi Client.fsx
1
Enter Id:
adan
Enter Password:
quert
Press 1.For Registration 2.For Posting Tweet 3. For Searching 4.Subscribe to an
User
clear
Press 1.For Registration 2.For Posting Tweet 3. For Searching 4.Subscribe to an
User
clr
Press 1.For Registration 2.For Posting Tweet 3. For Searching 4.Subscribe to an
User
clr
Press 1.For Registration 2.For Posting Tweet 3. For Searching 4.Subscribe to an
User
clr
Press 1.For Registration 2.For Posting Tweet 3. For Searching 4.Subscribe to an
User
```

There are two ports on the server and one port is for REST API on CMD interface and the other port is connected for WebSocket with browser UI. On the Webpage, the user can log in/out and see all tweets from other users who the user is subscribing to. The top prompt window is showing the server state. The bottom CMD window provides the interface where the user registers and creates tweets and searches tweets with keywords. When the user creates new tweets on CMD window, the new tweets will be delivered on the server and sent to the web page on web socket.

- **REST API User Interface**

```

C:\ Command Prompt - dotnet fsi Client.fsx
Press 1.For Registration 2.For Posting Tweet 3. For Searching 4.Subscribe to an
User
1
Enter Id:
tester
Enter Password:
qwerty
Press 1.For Registration 2.For Posting Tweet 3. For Searching 4.Subscribe to an
User
4
Enter Your Id:
tester
Enter Your Id You want to subscribe:
adam
Now you are subscribing to: adam
Press 1.For Registration 2.For Posting Tweet 3. For Searching 4.Subscribe to an
User
3
Press 1. For Search by Mention 2. Search by HashTag 3.Search Tweets of Subscribe
rs

```

The REST API client initiate with four options: 1. Registration 2. Posting tweets 3. Searching tweets 4. Subscribe to other users. The above picture shows the following process when the user registers. When user enter id and password, the data is stored in engine (server). After registration, the user can user other options with the user id which is the special key. On this example picture shows the process the user subscribes to other users. User ID “TESTER” is subscribing User ID “adam”. The next picture will show how the user whose id is “TESTER” can utilize the WebSocket API on the browser Interface.

- **WebSocket API User Interface.**

## My Twitter Page

Login ID:

CONNECTED

SENT: tester

From: adam

Message: this is the first message

Mentions: the first

From: adam

Message: the second

Mentions: the second mentions

The image displays two sequential screenshots of a Windows Command Prompt window running a .NET Core application. The title bar of the window is "Command Prompt - dotnet fsi Client:fsx".

**Top Screenshot:** The application is in its initial state. It prompts the user to "Enter your ID:". The user enters "adam". It then prompts for "Enter Message:". The user enters "the second". Next, it prompts for "Enter Mentions:". The user enters "the second mentions". Then, it prompts for "Enter HashTag:". The user enters "the testing". Finally, it prompts for "Tweet Posted". The user presses "1", which registers the user. The application then displays a menu: "Press 1.For Registration 2.For Posting Tweet 3. For Searching 4.Subscribe to an User".

**Bottom Screenshot:** The application is now in a search loop. It repeatedly prompts "Searching For tester" and "query string is adam". The user consistently enters "2" for "Size of tweet", which results in the application displaying "Size of tweet 2" for each search iteration.



**How to run:**

Folder structure
<b>Proj4_part2</b> <ul style="list-style-type: none"><li>- http-rest-client</li><li>- Twitter_API</li><li>- Index.html</li></ul>

- 1) [client] Move to the directory "Proj4\_part2/ http-rest-client"  
In CMD: dotnet fsi Client.fsx
- 2) [server] Move to the directory "Proj4\_part2/Twitter\_API"  
On different CMD: dotnet run
- 3) [WebPage] index.HTML  
Open in Chrome and login through User Id