README

Nayanika Ghosh – 4191 4976

Disha Nayar – 6199 1035

Steps to Run:

1. Install the following nuget packages –
    a. Suave
    b. Newtonsoft.Json
    c. Fsharpx.Collections
    d. Akka
    e. Akka.Fsharp

    Things to note to work with Suave :

    On Windows - Start your Visual Studio in Administrator mode
    On Mac – Give write permission to FSharp. Mostly located in /usr/local/share/dotnet/sdk/<version>/

2. Run command - dotnet fsi /langversion:preview WebServer.fsx
3. The command in 1, starts the websocket server at - *127.0.0.1:8082.* Go to *127.0.0.1:8082/index* to use the application
4. Check the demo for steps to check each functionality in detail.

Overview:

Our project submission consists of 4 .fsx file, one html file and a mp4 video file. Below is an overview of each of the files.

1) Engine.fsx - We have developed a twitter engine, in F# using AKKA actor model, with the below user facing functionality:
    a. Register
    b. Login
    c. Logout
    d. Tweet
    e. Retweet
    f. Search Hashtag
    g. Search Mention
    h. Subscribe to a user
    i. Unsubscribe to a user
2) WebServer.fsx – This file has the websocket interface that serves as a medium of communication between the engine and the client. It also has implementation of webpart that facilitates exchange of request and response following the REST protocol.

3) Index.html – We have written a javascript based front end(client) that communicates with the server for each of the functionality . The client sends a json request to the websocket which is deserialized and passed to the engine. The engine then serializes its json response and sends it back.
4) JSON api.fsx - We have implemented a JSON request and response data structure that each of our API use to serialize and deserialize their messages
5) TwitterCommon.fsx – This file defines the type of messages that are sent to the server from the websocket.
6) Video.mp4 – This video gives a walk through of our code and a small demo.

Implementation

JSON based APIs

To make our request-response structure generic, we have defined two JSON types

1. Request
    a. This type of messages are sent from the client to the websocket to request data or to perform an action. The declaration is as below.
    Request {
        func: string;
        username: string;
        input: string
    }
    b. func parameter defines the request type(Register, Login, Tweet, etc). Based on the request, input parameter is either a tweet, hashtag or mention to query or user to subscribe or unsubscribe.
2. Response
    a. This type of message is the response sent from the server. The declaration is as below.
    Response{
        func: string;
        username: string;
        success: string;
        tweet: string;
        hashtag: string;
        mention: string;
        subscribe: string;
        unsubscribe: string;
        results: string list;
        feed: string list list
        error: string
    }
    b. The response type remains the same for all the requests. We populate the relevant fields and leave the others as blank.

c. The response includes a parameter success which is false when an error is encountered and true otherwise. When success is false, the server also sends back the error response.

Websocket interface

We have used Suave framework to implement our websocket interface in F#. The websocket receives the request in json format from the client. The JSON object is first deserialized and depending on the request type a message is sent to the engine server.

Every user is given its own websocket connection to interact with the server. Hence every time a new connection is established, we store the websocket along with a username in a map. This map helps the engine to send a response(ex: Feed) to the client without it requesting for it. The websocket connection is closed on logout or page refresh.

Client using websocket

Each user has a websocket connection opened with the websocket listening at the server end. This connection session is maintained from Login to Logout. We have defined functions on each of the event on websocket like onOpen, onClose, onMessage, onError, and more. All the messages for different user facing functionality are sent over this websocket. Since websocket is used for two-way communication, the response is also received over the same websocket per user.

Twitter Engine

We have implemented a twitter engine using AKKA actor model. The websocket at the server end routes the client request to the engine based on the requested function. The engine has the main implementation where we are interacting with in-memory datatables.