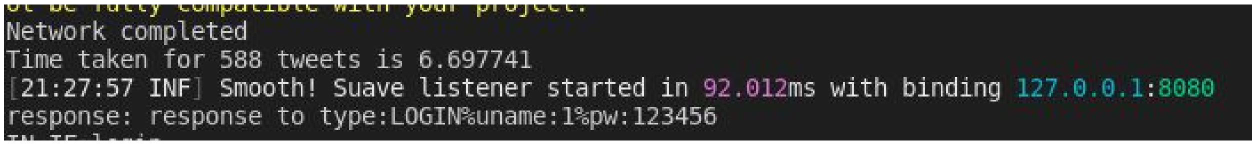
Team Members:

- Aditi Page (UFID: 0427-9968)
- Kartik Rode (UFID: 8971-1791)

Video Link: https://youtu.be/Kk2TeAN4YM4

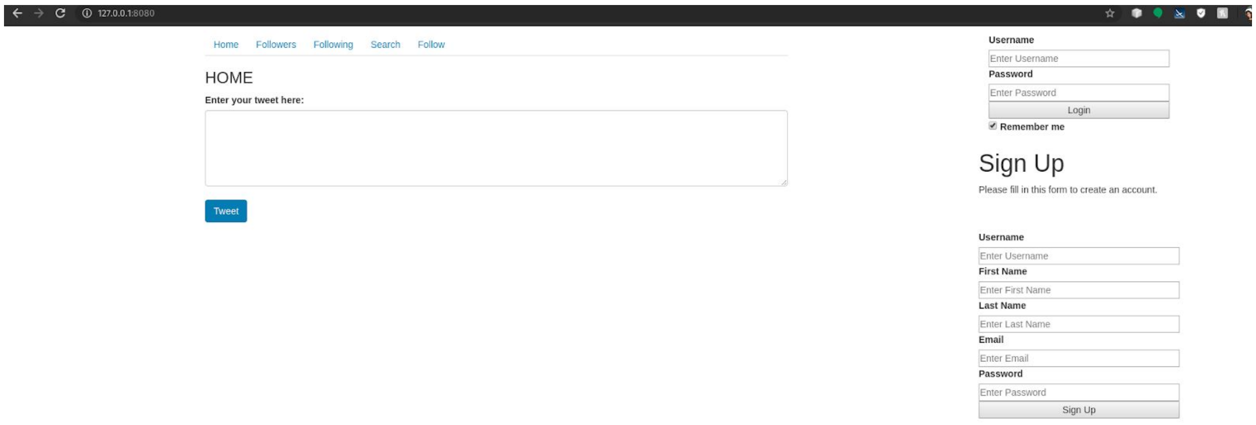
Instructions to run the file:

1. Unzip the file
2. Go to the project folder
3. Run the project using:
    a. dotnet run



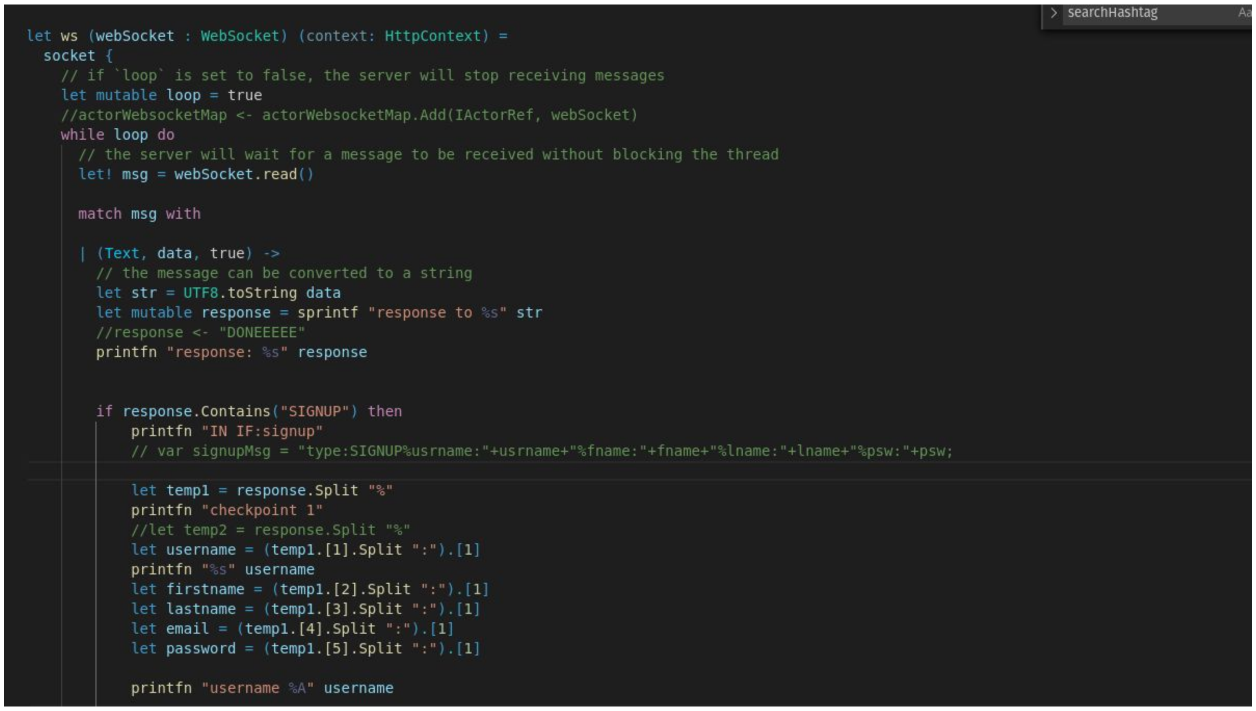
```
ot be fully compatible with your project.
Network completed
Time taken for 588 tweets is 6.697741
[21:27:57 INF] Smooth! Suave listener started in 92.012ms with binding 127.0.0.1:8080
response: response to type:LOGIN%uname:1%pw:123456
```

We are using Suave.IO for using websockets. After going to the above URL, we have the following web page



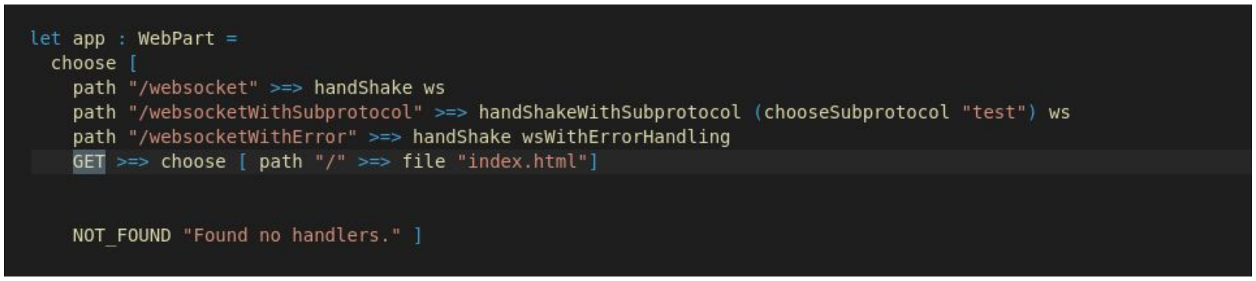
On the web page, users can register and login on the system. (screenshot above)

In the screenshot below, the code snippet for receiving responses from the front-end through websockets is shown. These responses are then converted from byte arrays into string and depending on their types (“SIGNUP, “LOGIN”, “SEARCH:, “FOLLOW”, etc.), they are further processed in the backed for sending the responses back to the front-end using websockets.



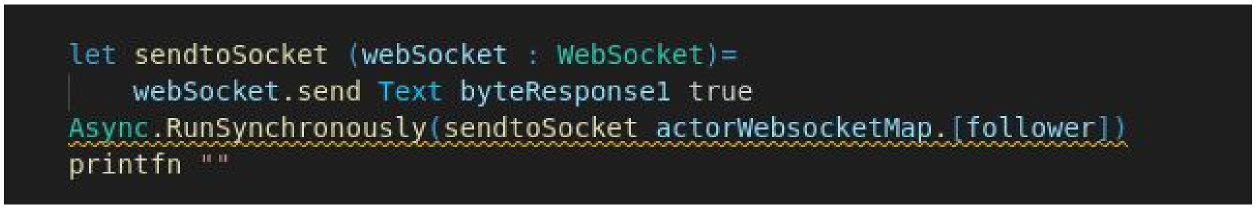
```
let ws (webSocket : WebSocket) (context: HttpContext) =
  socket {
    // if `loop` is set to false, the server will stop receiving messages
    let mutable loop = true
    //actorWebsocketMap <- actorWebsocketMap.Add(IActorRef, webSocket)
    while loop do
      // the server will wait for a message to be received without blocking the thread
      let! msg = webSocket.read()

      match msg with

      | (Text, data, true) ->
        // the message can be converted to a string
        let str = UTF8.toString data
        let mutable response = sprintf "response to %s" str
        //response <- "DONEEEEE"
        printfn "response: %s" response


        if response.Contains("SIGNUP") then
            printfn "IN IF:signup"
            // var signupMsg = "type:SIGNUP%usrname:"+usrname+"%fname:"+fname+"%lname:"+lname+"%psw:"+psw;

            let temp1 = response.Split "%"
            printfn "checkpoint 1"
            //let temp2 = response.Split "%"
            let username = (temp1.[1].Split ":").[1]
            printfn "%s" username
            let firstname = (temp1.[2].Split ":").[1]
            let lastname = (temp1.[3].Split ":").[1]
            let email = (temp1.[4].Split ":").[1]
            let password = (temp1.[5].Split ":").[1]

            printfn "username %A" username
```

In the screenshot below, the code snippet for creating end points is shown.
Code referred from:
https://github.com/SuaveIO/suave/tree/master/examples/WebSocket



```
let app : WebPart =
  choose [
    path "/websocket" >=> handShake ws
    path "/websocketWithSubprotocol" >=> handShakeWithSubprotocol (chooseSubprotocol "test") ws
    path "/websocketWithError" >=> handShake wsWithErrorHandling
    GET >=> choose [ path "/" >=> file "index.html"]


    NOT_FOUND "Found no handlers." ]
```

The values requested from the front-end functionalities are sent from the backend using the following snippet that involves sending a message converted to byteResponse via the active actor’s websocket.



```
let sendtoSocket (webSocket : WebSocket)=
    webSocket.send Text byteResponse1 true
Async.RunSynchronously(sendtoSocket actorWebsocketMap.[follower])
printfn ""
```