README File[Report]:

- ## Project Team
  - Shromana Kumar
  - Sangpil Youm

- ## Youtube link
  - https://www.youtube.com/watch?v=rdQBSAdkUgw

- ## What is working
  - Twitter Clone with WebSocket interface.
  - Cowboy WebSocket implemented here
  - By using Cowboy WebSocket, JSON based API was designed.
  - Re-write Project4 part 1 code with cowboy
  - Re-write client parts corresponding server using WebSocket

- ## How it works

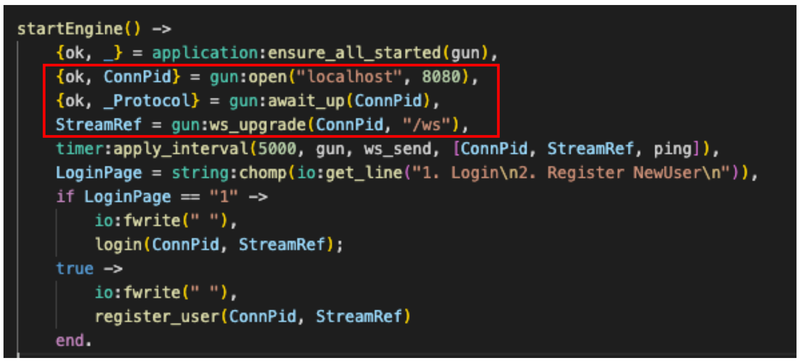
File Structure

```
Server [directory: twitter_sever/src]
        twitter_handler.erl
        twitter_server_app.erl
        twitter_server_sup.erl

Client [directory: twitter_clone/src]
        twitter_client.erl
```

Client File : twitter_client.erl

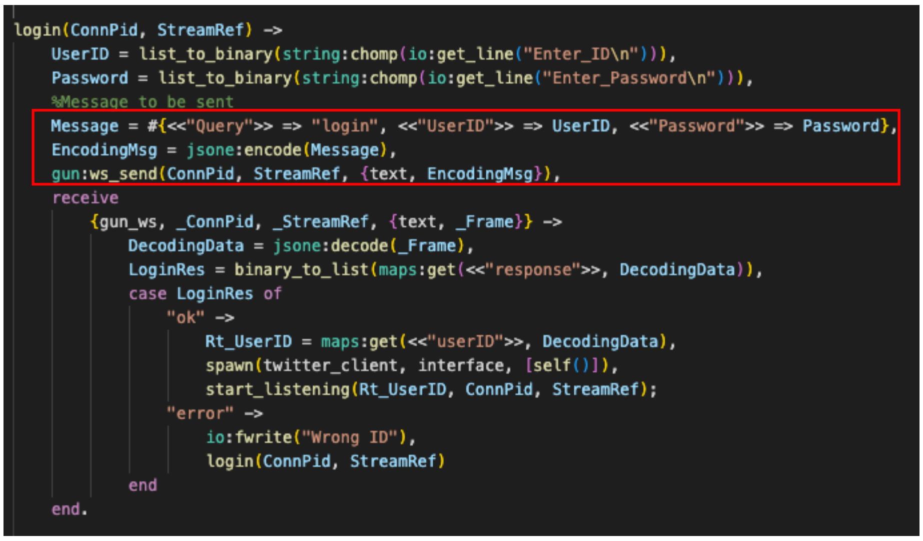
By using gun, the client connect to server



```
startEngine() ->
    {ok, _} = application:ensure_all_started(gun),
    {ok, ConnPid} = gun:open("localhost", 8080),
    {ok, _Protocol} = gun:await_up(ConnPid),
    StreamRef = gun:ws_upgrade(ConnPid, "/ws"),
    timer:apply_interval(5000, gun, ws_send, [ConnPid, StreamRef, ping]),
    LoginPage = string:chomp(io:get_line("1. Login\n2. Register NewUser\n")),
    if LoginPage == "1" ->
        io:fwrite(" "),
        login(ConnPid, StreamRef);
    true ->
        io:fwrite(" "),
        register_user(ConnPid, StreamRef)
    end.
```

Login

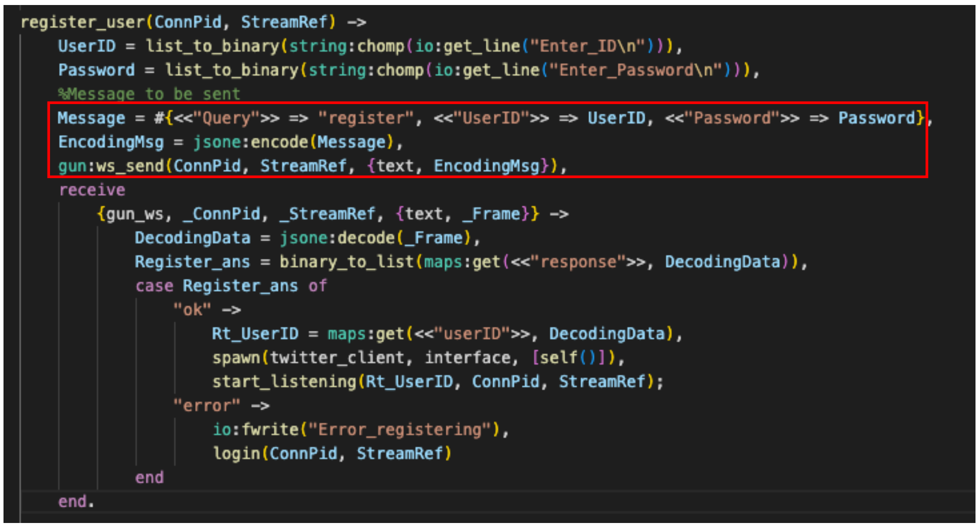
Message includes User ID and Password and encoding in JSON using jsone erlang library. Receive from server and decode the message that is from server.



```
login(ConnPid, StreamRef) ->
    UserID = list_to_binary(string:chomp(io:get_line("Enter_ID\n"))),
    Password = list_to_binary(string:chomp(io:get_line("Enter_Password\n"))),
    %Message to be sent
    Message = #{<<"Query">> => "login", <<"UserID">> => UserID, <<"Password">> => Password},
    EncodingMsg = jsone:encode(Message),
    gun:ws_send(ConnPid, StreamRef, {text, EncodingMsg}),
    receive
        {gun_ws, _ConnPid, _StreamRef, {text, _Frame}} ->
            DecodingData = jsone:decode(_Frame),
            LoginRes = binary_to_list(maps:get(<<"response">>, DecodingData)),
            case LoginRes of
                "ok" ->
                    Rt_UserID = maps:get(<<"userID">>, DecodingData),
                    spawn(twitter_client, interface, [self()]),
                    start_listening(Rt_UserID, ConnPid, StreamRef);
                "error" ->
                    io:fwrite("Wrong ID"),
                    login(ConnPid, StreamRef)
            end
    end.
```

Register_user

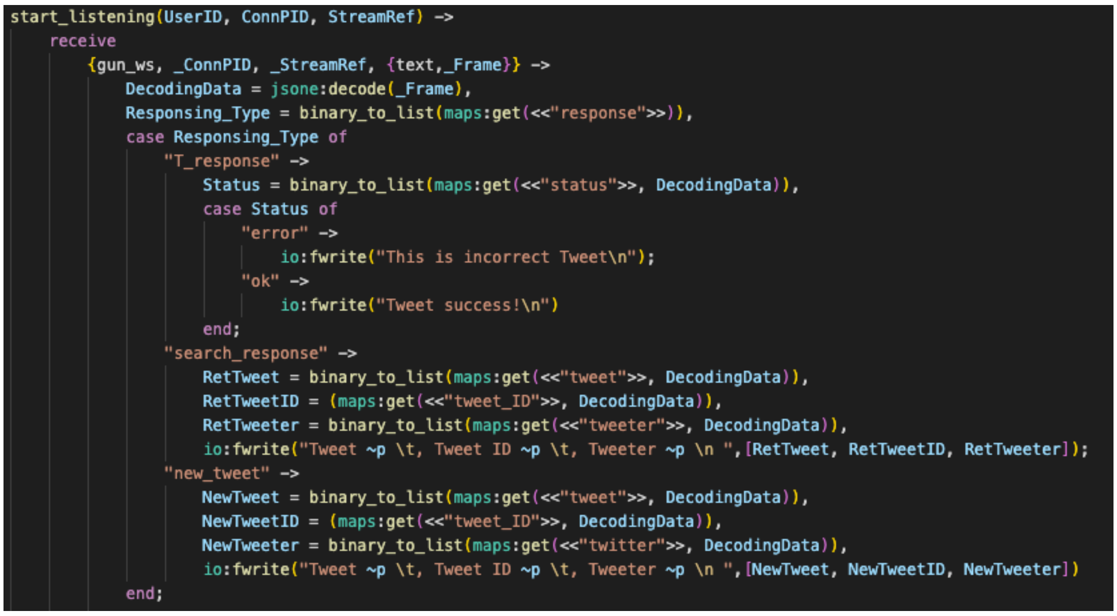
Message includes User ID and Password and encoding in JSON using jsone erlang library. Receive from server and decode the message that is from server.



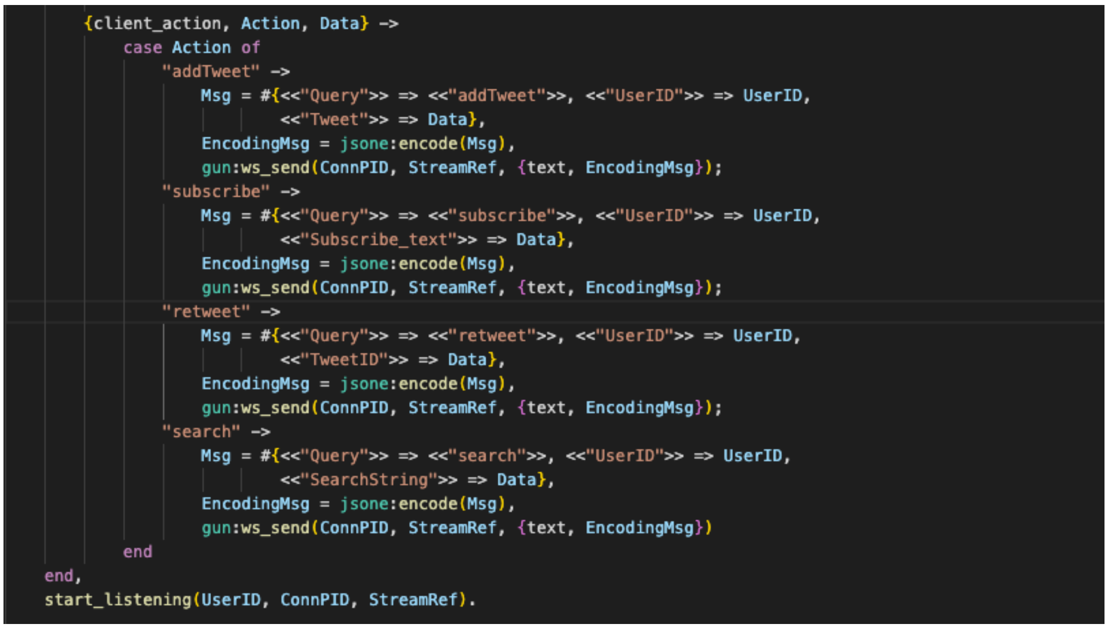
```erlang
register_user(ConnPid, StreamRef) ->
    UserID = list_to_binary(string:chomp(io:get_line("Enter_ID\n"))),
    Password = list_to_binary(string:chomp(io:get_line("Enter_Password\n"))),
    %Message to be sent
    Message = #{<<"Query">> => "register", <<"UserID">> => UserID, <<"Password">> => Password},
    EncodingMsg = jsone:encode(Message),
    gun:ws_send(ConnPid, StreamRef, {text, EncodingMsg}),
    receive
        {gun_ws, _ConnPid, _StreamRef, {text, _Frame}} ->
            DecodingData = jsone:decode(_Frame),
            Register_ans = binary_to_list(maps:get(<<"response">>, DecodingData)),
            case Register_ans of
                "ok" ->
                    Rt_UserID = maps:get(<<"userID">>, DecodingData),
                    spawn(twitter_client, interface, [self()]),
                    start_listening(Rt_UserID, ConnPid, StreamRef);
                "error" ->
                    io:fwrite("Error_registering"),
                    login(ConnPid, StreamRef)
            end
    end.
```

Start Listening

From Server, receive message and works function on “gun_ws”. Receiving with “client_action”, it implements actions that is operated (i.e., addTweet, subscription,retweet, and search).



```erlang
start_listening(UserID, ConnPID, StreamRef) ->
    receive
        {gun_ws, _ConnPID, _StreamRef, {text,_Frame}} ->
            DecodingData = jsone:decode(_Frame),
            Responsing_Type = binary_to_list(maps:get(<<"response">>)),
            case Responsing_Type of
                "T_response" ->
                    Status = binary_to_list(maps:get(<<"status">>, DecodingData)),
                    case Status of
                        "error" ->
                            io:fwrite("This is incorrect Tweet\n");
                        "ok" ->
                            io:fwrite("Tweet success!\n")
                    end;
                "search_response" ->
                    RetTweet = binary_to_list(maps:get(<<"tweet">>, DecodingData)),
                    RetTweetID = (maps:get(<<"tweet_ID">>, DecodingData)),
                    RetTweeter = binary_to_list(maps:get(<<"tweeter">>, DecodingData)),
                    io:fwrite("Tweet ~p \t, Tweet ID ~p \t, Tweeter ~p \n ",[RetTweet, RetTweetID, RetTweeter]);
                "new_tweet" ->
                    NewTweet = binary_to_list(maps:get(<<"tweet">>, DecodingData)),
                    NewTweetID = (maps:get(<<"tweet_ID">>, DecodingData)),
                    NewTweeter = binary_to_list(maps:get(<<"twitter">>, DecodingData)),
                    io:fwrite("Tweet ~p \t, Tweet ID ~p \t, Tweeter ~p \n ",[NewTweet, NewTweetID, NewTweeter])
            end;
```



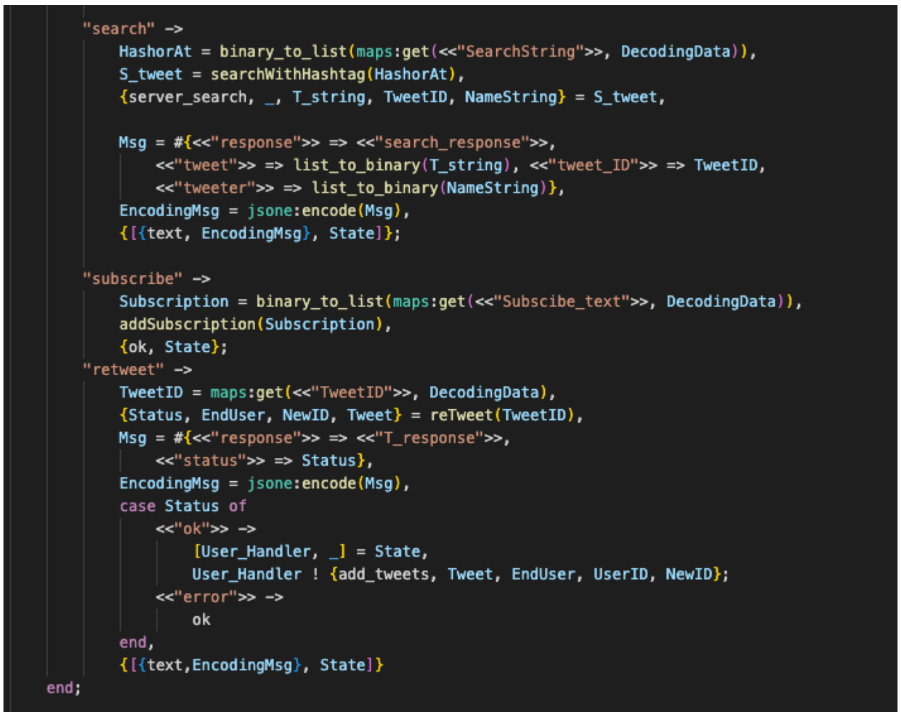
```erlang
    {client_action, Action, Data} ->
        case Action of
            "addTweet" ->
                Msg = #{<<"Query">> => <<"addTweet">>, <<"UserID">> => UserID,
                        <<"Tweet">> => Data},
                EncodingMsg = jsone:encode(Msg),
                gun:ws_send(ConnPID, StreamRef, {text, EncodingMsg});
            "subscribe" ->
                Msg = #{<<"Query">> => <<"subscribe">>, <<"UserID">> => UserID,
                        <<"Subscribe_text">> => Data},
                EncodingMsg = jsone:encode(Msg),
                gun:ws_send(ConnPID, StreamRef, {text, EncodingMsg});
            "retweet" ->
                Msg = #{<<"Query">> => <<"retweet">>, <<"UserID">> => UserID,
                        <<"TweetID">> => Data},
                EncodingMsg = jsone:encode(Msg),
                gun:ws_send(ConnPID, StreamRef, {text, EncodingMsg});
            "search" ->
                Msg = #{<<"Query">> => <<"search">>, <<"UserID">> => UserID,
                        <<"SearchString">> => Data},
                EncodingMsg = jsone:encode(Msg),
                gun:ws_send(ConnPID, StreamRef, {text, EncodingMsg})
        end
  end,
  start_listening(UserID, ConnPID, StreamRef).
```

Server File

: twitter_server_app.erl, twitter_handler.erl, twitter_server_sup.erl

websocket_handler

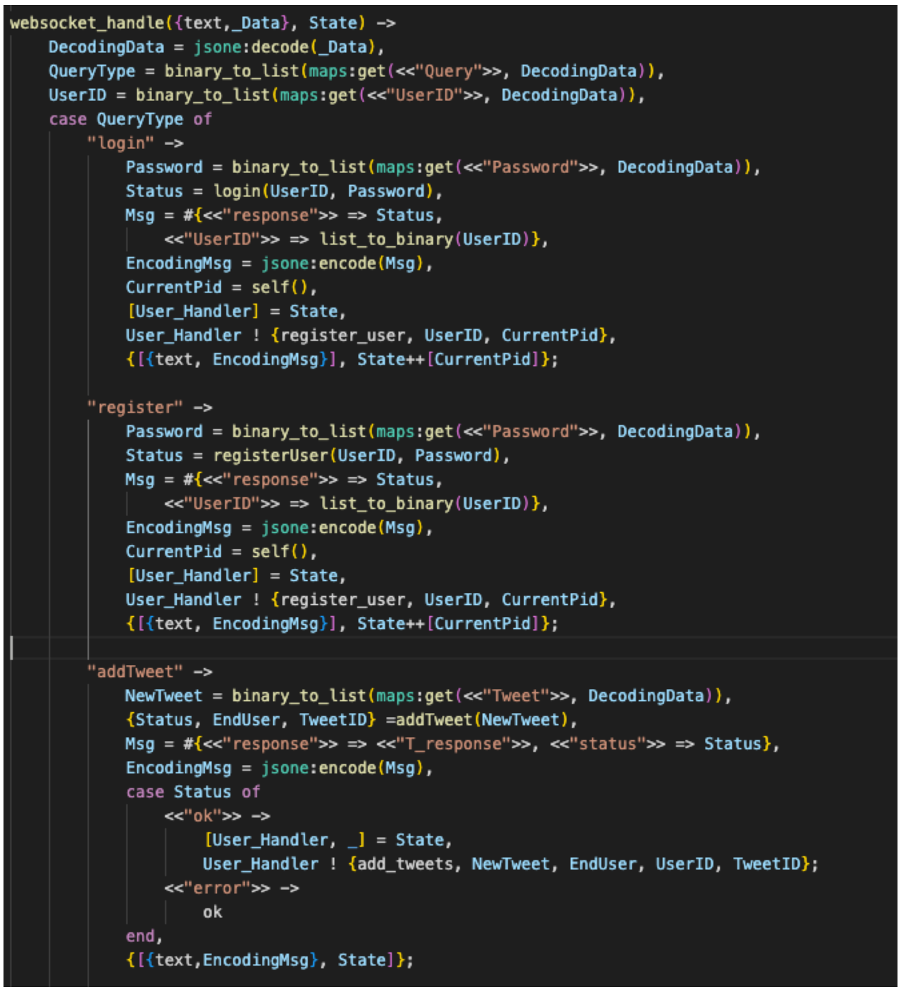
In websocket_handler, receive JSON data from client and decode this data using jsone library



```erlang
        "search" ->
            HashorAt = binary_to_list(maps:get(<<"SearchString">>, DecodingData)),
            S_tweet = searchWithHashtag(HashorAt),
            {server_search, _, T_string, TweetID, NameString} = S_tweet,

            Msg = #{<<"response">> => <<"search_response">>,
                <<"tweet">> => list_to_binary(T_string), <<"tweet_ID">> => TweetID,
                <<"tweeter">> => list_to_binary(NameString)},
            EncodingMsg = jsone:encode(Msg),
            {[{text, EncodingMsg}, State]};

        "subscribe" ->
            Subscription = binary_to_list(maps:get(<<"Subscibe_text">>, DecodingData)),
            addSubscription(Subscription),
            {ok, State};
        "retweet" ->
            TweetID = maps:get(<<"TweetID">>, DecodingData),
            {Status, EndUser, NewID, Tweet} = reTweet(TweetID),
            Msg = #{<<"response">> => <<"T_response">>,
                <<"status">> => Status},
            EncodingMsg = jsone:encode(Msg),
            case Status of
                <<"ok">> ->
                    [User_Handler, _] = State,
                    User_Handler ! {add_tweets, Tweet, EndUser, UserID, NewID};
                <<"error">> ->
                    ok
            end,
            {[{text,EncodingMsg}, State]}
    end;
```



```erlang
websocket_handle({text,_Data}, State) ->
    DecodingData = jsone:decode(_Data),
    QueryType = binary_to_list(maps:get(<<"Query">>, DecodingData)),
    UserID = binary_to_list(maps:get(<<"UserID">>, DecodingData)),
    case QueryType of
        "login" ->
            Password = binary_to_list(maps:get(<<"Password">>, DecodingData)),
            Status = login(UserID, Password),
            Msg = #{<<"response">> => Status,
                <<"UserID">> => list_to_binary(UserID)},
            EncodingMsg = jsone:encode(Msg),
            CurrentPid = self(),
            [User_Handler] = State,
            User_Handler ! {register_user, UserID, CurrentPid},
            {[{text, EncodingMsg}], State++[CurrentPid]};

        "register" ->
            Password = binary_to_list(maps:get(<<"Password">>, DecodingData)),
            Status = registerUser(UserID, Password),
            Msg = #{<<"response">> => Status,
                <<"UserID">> => list_to_binary(UserID)},
            EncodingMsg = jsone:encode(Msg),
            CurrentPid = self(),
            [User_Handler] = State,
            User_Handler ! {register_user, UserID, CurrentPid},
            {[{text, EncodingMsg}], State++[CurrentPid]};

        "addTweet" ->
            NewTweet = binary_to_list(maps:get(<<"Tweet">>, DecodingData)),
            {Status, EndUser, TweetID} =addTweet(NewTweet),
            Msg = #{<<"response">> => <<"T_response">>, <<"status">> => Status},
            EncodingMsg = jsone:encode(Msg),
            case Status of
                <<"ok">> ->
                    [User_Handler, _] = State,
                    User_Handler ! {add_tweets, NewTweet, EndUser, UserID, TweetID};
                <<"error">> ->
                    ok
            end,
            {[{text,EncodingMsg}, State]};
```