Project 4 Part II

Debarshi Mitra UFID: 3381-3136

Aisharjya Sarkar UFID: 4495-5999

The youtube link for the video: https://youtu.be/4GM6bs4u3k8

How to run the code –

The following steps needs to be performed to run the code:

1. Unzip the zipped file named twitterengine.zip and open it through an IDE (we used Visual Studio Code).

2. run the command `mix deps.get`

3. `cd twitterengine`

4. run the command `mix ecto.create`

5. Next to run server: `mix phx.server`

6. Open in a browser: http://localhost:4000

Next the highlighted portions in the video are explained in depth here:

Implementation of WebSocket interface using Phoenix Engine:

In the file endpoint.ex which marks the entry-point and contains the init() method as well. Here we define the socket which loads the corresponding UserSocket file. This UserSocket is the user_socket.ex file under twitterengine_web shown in the next screenshot.

```
defmodule TwitterengineWeb.Endpoint do
  use Phoenix.Endpoint, otp_app: :twitterengine

  socket "/socket", TwitterengineWeb.UserSocket

  # Serve at "/" the static files from "priv/static" directory.
  #
  # You should set gzip to true if you are running phoenix.digest
  # when deploying your static files in production.
  plug Plug.Static,
    at: "/", from: :twitterengine, gzip: false,
    only: ~w(css fonts images js favicon.ico robots.txt)

  # Code reloading can be explicitly enabled under the
  # :code_reloader configuration of your endpoint.
  if code_reloading? do
    socket "/phoenix/live_reload/socket", Phoenix.LiveReloader.Socket
    plug Phoenix.LiveReloader
    plug Phoenix.CodeReloader
  end

  plug Plug.RequestId
  plug Plug.Logger
```

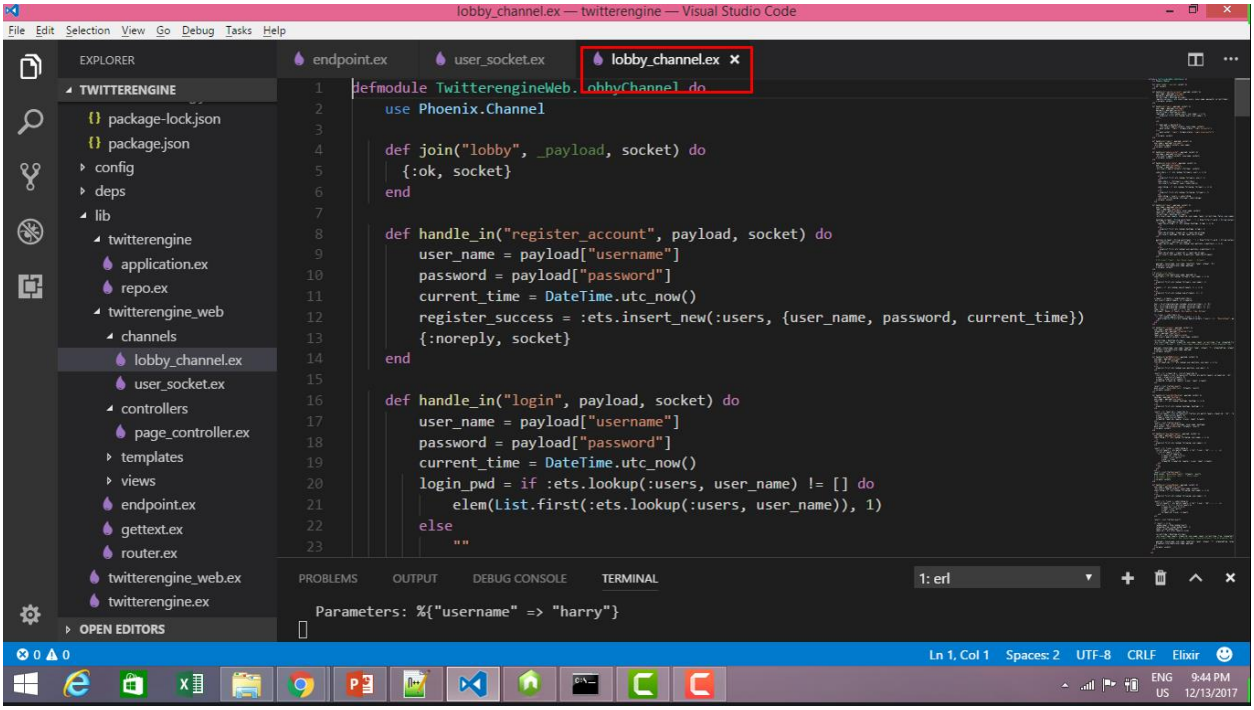
```
Parameters: %{"username" => "harry"}
```

In user_socket.ex, all the channels are defined with a string dentifier and corresponding channel file name. Here we have used only one channel named “lobby” and the corresponding channel as LobbyChannel. The connect method beneath returns the entire socket with all the channel and WebSocket information.

```
defmodule TwitterengineWeb.UserSocket do
  use Phoenix.Socket

  ## Channels
  channel "lobby", TwitterengineWeb.LobbyChannel

  ## Transports
  transport :websocket, Phoenix.Transports.WebSocket
  # transport :longpoll, Phoenix.Transports.LongPoll

  # Socket params are passed from the client and can
  # be used to verify and authenticate a user. After
  # verification, you can put default assigns into
  # the socket that will be set for all channels, ie
  #
  #     {:ok, assign(socket, :user_id, verified_user_id)}
  #
  # To deny connection, return `:error`.
  #
  # See `Phoenix.Token` documentation for examples in
  # performing token verification on connect.
  def connect(_params, socket) do
    {:ok, socket}
```

```
Parameters: %{"username" => "harry"}
```

This is the corresponding channel file as defined in the user_socket.ex file against identifier “lobby”. This file acts like the Genserver which handles all the functional logic by handle_in method calls where each method is for a specific functionality.



Client to use WebSockets:

Once a browser is loaded from the client side, the file app.js is loaded which imports the socket as shown below.

app.js — twitterengine — Visual Studio Code

```
// Brunch automatically concatenates all files in your
// watched paths. Those paths can be configured at
// config.paths.watched in "brunch-config.js".
//
// However, those files will only be executed if
// explicitly imported. The only exception are files
// in vendor, which are never wrapped in imports and
// therefore are always executed.

// Import dependencies
//
// If you no longer want to use a dependency, remember
// to also remove its path from "config.paths.watched".
import "phoenix_html"

// Import local files
//
// Local files can be imported directly using relative
// paths "./socket" or full ones "web/static/js/socket".

import socket from "./socket"
```

```
Parameters: %{"username" => "kelly"}
```

Next, new socket connection is invoked in socket.js as highlighted below.

socket.js — twitterengine — Visual Studio Code

```
// NOTE: The contents of this file will only be executed if
// you uncomment its entry in "assets/js/app.js".

// To use Phoenix channels, the first step is to import Socket
// and connect at the socket path in "lib/web/endpoint.ex":
import {Socket} from "phoenix"

let socket = new Socket("/socket", {params: {token: window.userToken}})

// When you connect, you'll often need to authenticate the client.
// For example, imagine you have an authentication plug, `MyAuth`,
// which authenticates the session and assigns a `:current_user`.
// If the current user exists you can assign the user's token in
// the connection for use in the layout.
//
// In your "lib/web/router.ex":
//
//     pipeline :browser do
//       ...
//       plug MyAuth
//       plug :put_user_token
//     end
//
```

```
[info] Sent 200 in 0ms
```

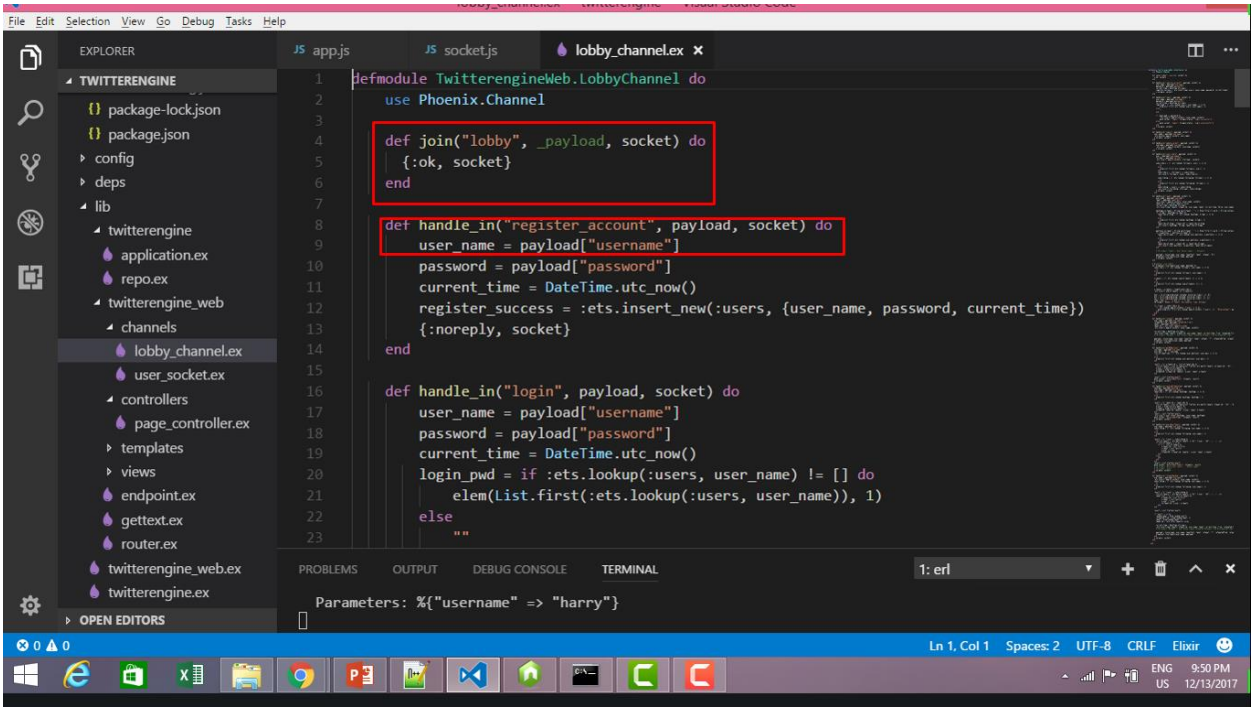
Next, the socket.connect() is invoked and a new channel is defined with the defined string identifier “lobby” as shown before. Also, based on an event invoked by the user browser, say button click, the

corresponding method is invoked in the channel by channel.push() method. The first parameter is the identifier which matches the handle_in method in the channel. The parameter are passed in JSON format by attribute: value pair and send to the channel as payload.

```
// from connect if you don't care about authentication.

socket.connect()

// Now that you are connected, you can join channels with a topic:
let channel = socket.channel("lobby", {});

$(document).ready(function() { channel.push('update_socket', { username: userID });
});

if(document.getElementById("signup"))        // use this if you are using id to check
{
  let newUsername = $('#newUsername');
  let newPassword    = $('#newPassword');
  //window.location = "http://localhost:4000/dashboard.html?temp=" + newUsername;

  document.getElementById("signup").onclick = function() {

    //console.log(window.location.href);
  channel.push('register_account', { username: newUsername.val(), password: newPassword.val()
  alert("Account created successfully!");

  //location.href="home.html"
```

```
Parameters: %{"username" => "harry"}
```

The “register_account” identitifer of channel.push() method invokes the handle_in(“register_account”, payload, socket) method in the channel. The socket joins the channel by the join method again by the identifier (here “lobby”) and the entire socket information is returned. The internal data is transferred by JSON format in Phoenix framework via the parameter named _payload.



All the other functionalities are performed in similar way.