DOS Project 4 Part 2:

TEAM MEMBERS:

Name: Poornima Kumar
UFID: 5684-4925

Name: Harika Bukkapattanam
UFID: 3683-6895

PROGRAM EXECUTION:

1. **Running the project:**

   Running the server:
   iex -S mix phx.server

   Running the clients:

   To test each client manually, run the following commands in another terminal:

   Execution steps:

   iex -S mix

   1. Registering users:
      Open different terminals or on the same terminal, create multiple users by executing the following commands multiple times with different client_name.
      Client.register(client_name)
      Eg. Tweeter.SocketClient.register("Client1")

   2. Subscribing to other users:
      Client.subscribe(your_name, name_of_client_to_subscribe)
      Eg. Tweeter.SocketClient.subscribe("Client1", "Client2")

   3. Sending tweets:
      Client.send_tweet(name_of_the_client_sending_tweet, tweet)
      Eg. Tweeter.SocketClient.send_tweet("Client1", "#DOS project #part1 @Client2")

   4. Retweet:
      For retweeting, the user must pass the tweet_id which he wants to retweet. This tweet_id can be retrieved from the console, where the tweet has been displayed along with the tweet_id.
      Client.retweet(name_of_the_client_retweeting, tweet_id)
      Eg. Tweeter.SocketClient.retweet("Client1", 3)

   5. Disconnecting a user:

Client.stop(name_of_client_to_be_disconnected)
Eg. Tweeter.SocketClient.stop("Client1")

6. Starting the disconnected client again:
   Client.start_link(name_of_disconnected_client)
   Eg. Tweeter.SocketClient.start_link("Client1")
   Make sure to pass the name correctly to start the disconnected client

7. Searching "my mentions":
   A client can query the tweets in which he has been mentioned
   Client.search_my_mentions(name_of_the_client)
   Eg. Tweeter.SocketClient.search_my_mentions("Client1")

8. Searching for tweets with specific hashtags:
   Client.search_hashtag(name_of_the_client, hashtag)
   Eg. Tweeter.SocketClient.search_hashtag("Client1", "#DOS")

9. Querying the tweets subscribed to:
   Client.get_user_tweets(name_of_the_client)
   Eg. Tweeter.SocketClient.get_user_tweets("Client1)

2. Running the project with the Simulator

Running the server:
iex -S mix phx.server

Running the Simulator in another terminal:
iex -S mix
Project4Part2Simulator.start(num_of_clients)
Eg. Project4Part2Simulator.start(10)

IMPLEMENTATION:

The project is a phoenix application. We have created one channel having multiple topics. Phoenix holds a single connection to the server and multiplexes other channel sockets over that one connection. All the messages and the replies are JSON based. Socket handlers are used by phoenix to authenticate and identify a socket connection which can be used by the channel. Both the client and the server communicate using web sockets. The Phoenix framework has the capability to handle the Elixir-to-JSON encoding on the Server side.

Modules:

Project4Part2Simulator.ex

This is a simulator file. It simulates Zipf Distribution and all the functionalities like registering, subscribing, send tweet, retweet, disconnect, connect, searching for hashtags and mentions.

Tweeter.ex

The Tweeter.ex module is a Phoenix application which acts as a supervisor to the endpoint connections.

On running the application, it creates :ets tables as mentioned below which are accessible and maintained by the Twitter engine (ie. Tweeter.ex module):
List of :ets tables:

:ets Table Name	Key	Value	Description
tweets	tweet_id	[tweet, hashtags, mentions, sender]	Maintains a list of tweets and its original sender.
subscribers	subscribed_to_process_name	subscribed_from_process_name	Maintains a list of all the process that has subscribed to the current process
following	subscribed_to_process_name	subscribed_from_process_name	Maintains a list of all the processes which the current process is following.
Hashtags	hashtag	tweet_id	Associates the hashtags with the tweets. The table is used for querying tweets based on hashtags.
My_mentions	mentioned_process_name	tweet_id	Maintains a list of all the processes who have been mentioned in different tweets.
User_tweets	process_name	sent_tweet_id	Maintains a list of all the tweets by the current user.
User_offline_tweets	process_name	received_offline_tweet_id	Stores all the tweets received by a client which is disconnected. The table is used to display all the tweets that were tweeted by its subscribers when the client was disconnected.

Tweeter.SocketClient module:

Each client is a process which make the API calls to perform a specific action. The different tasks are performed by making the following API calls:

Method Name	Description
register(name)	Used to register the client when it tries to connect for the first time.
start_link(name)	Restarts the client which was disconnected.
stop(name)	Disconnects the client
subscribe(name, process_name)	Current client can subscribe to any other client by specifying the name of the other client.
send_tweet(name, tweet)	Used to send tweets to all the subscribers.
search_hashtag(name, hash_tag)	Used to query the tweets having a specific hashtag.
search_my_mentions(name)	Used to query the tweets in which the current user is mentioned.
get_user_tweets(name)	Used to query all the tweets posted by the current user
get_user_retweets(name)	Queries all the tweets that has been retweeted by the current user.
retweet(name, tweet_id)	Used to retweet a tweet which the current user has received from its subscriber.

TweeterWeb.TweeterWebChannel module:

This module is the server and acts as the engine which distributes the tweets. All the client requests like registration, sending tweet, retweet, disconnection, reconnection and querying of tweets, hashtags and my_mentions, are handled by this module. The channel handles the events received from the client and behave in a similar way as the controller. There are many handle_in methods defined to handle different events like subscribe, send_tweet, retweet, search hashtags, search mymentions, retweets and loading the tweets missed by the client when a subscriber tweets.

We have made used of the Phoenix.Channels.GenSocketClient behavior which supports the websocket communication protocol. A client can join one or many topics on the same socket. Once a client joins a topic, join and push messages can be used messages to the server. All these functions make use of the transport information as an argument.

NOTES:

1. Please run mix deps.get to install all the dependencies and then execute mix deps.compile