DOS PROJECT 4.1

UFID : 6485-1063
UFID : 4996-9899

GOAL : Twitter-like engine

IMPLEMENTATION

PROJECT FLOW

- The project has a RequestHandler which is the main GenServer and has ‘n’ ClientServer GenServers for ‘n’ users.
- The RequestHandler initializes the tables using Elixir’s ETS(Erlang Term Storage) functionality.
- We have created the following tables:
    - User table -- stores user id, pid of each user.
    - Tweets table -- stores user_id, tweet id, tweet message, and original tweet id(is set if a tweet is retweeted)
    - Subscribers table -- stores subscriber, subscribed to
    - Hashtags table -- stores hashtags, tweet id(in which hashtag was present)
    - Mentions table -- stores mentions user id, tweet id(in which they were mentioned)
- Request Handler takes care of registration, login, storing tweets, querying tweets and other utility functions.
- ClientServer calls RequestHandler to add its subscribers, storing tweets, querying tweets by subscribers.
- We have 30 registered users and 20 logged in users who are sending 15 tweets each.

FUNCTIONALITIES IMPLEMENTED

- Account Registration
    - RequestHandler can register ‘n’ users by making a synchronous call.We made this functionality as synchronous(handle_call) as we needed to ensure that the users have been registered in order to maintain stability on other operations on the user.
- Account Login
    - We can make ‘x’ users login by making a call to RequestHandler and setting pid field in the user table to update alive users
- Sending Tweet
    - Clients(Users) can send tweets which will then by stored by RequestHandler under tweets table. We made this functionality as synchronous(handle_call) as we needed to ensure whether the tweets are being stored correctly for each user or not.
- Storing hashtags

    - Each tweet message is checked for hashtags by a utility function. We made this functionality as asynchronous(handle_cast) so that every user is able to access the function as it is an independent functionality.
- Storing mentions
    - Each tweet is checked for mentions, where mentions is the user id of the existing users by a utility function. We made this functionality as asynchronous(handle_cast) so that every user is able to access the function as it is an independent functionality.
- Managing subscribers
    - We have made 2 subscribers for each user ensuring that everyone has at least one subscriber. These 2 subscribers are next three user id's in the table. Each client(user) calls GenServer to store its subscribers, thus keeping track of it. We made this functionality as synchronous(handle_call) as we needed to ensure that the next in line task that is based on the subscriber is handled properly.
- Retweets
    - We have maintained a field "original_tweet_id" in :tweets table, which gets filled if it's a retweet. This can be maintained across treats by recurring backwards until a original tweet_id is found nil.
- Querying
    - Subscribed_to: queried the subscribers table via client server, executed by engine
    - Get particular hash_tag : queried the :hashtags table via client server, executed by engine
    - Get all mentions : queried the :subscribers table via client server, executed by engine
- Send messages to alive users
    - First got all subscribers and filtered out the live nodes and multicasted the message, which gets stored in the state of the node.
- Logout users
    - We log out users by setting their pid to nil in the user table which corresponds to their status of whether they are alive(logged in) or not.
- Delete account
    - We deleted a users account using ETS's delete and delete_match from user table, tweets table, subscriber table, mentions(if any).

EXUNIT TESTING

- ➢ **Test Case #1**
    - Firstly, we are initializing all the 5 tables we have created. Then asserting whether these tables have been created.
- ➢ **Test Case #2**
    - For the testing purpose, we have registered 30 users and asserting whether all of them have been registered by checking the user table entries.
- ➢ **Test Case #3**
    - We then login 20 users and assert if all 20 have actually logged in and their data is saved in the ETS table.
- ➢ **Test Case #4**

- All the logged in users are made to send 15 tweets and asserting if the tweets have been stored or not.

- **Test Case #5**
  - All users have at least 1 subscriber. We have made 2 subscribers for each user and asserting whether each users have 2 subscribers or not.

- **Test Case #6/#7**
  - While sending tweets, we are mentioning some hashtags. We are asserting whether these hashtags have been stored in the hashtag table or not.
- **Test Case #8**
  - While sending tweets, every user(client) is mentioning some user. We are asserting whether these mentions have been stored in the mentions table or not.
- **Test Case #9**
  - We have created registered and logged in 2 users, making one the subscriber of the other and asserting if the subscriber can retweet the tweet it got from the one it is subscribed to.
- **Test Case #10**
  - Asserting if the user can query for specific hashtags by ClientServer making a call to RequestHandler
- **Test Case #11**
  - Asserting if the user can query its mentions. They can check for mentions' tweets via tweet id which is used to assert the test case.
- **Test Case #12**
  - Asserting if all the live users got the notification of tweets posted by the users they have subscribed to. The notification is checked by validating a field in the state(where we are storing the tweet id in the notification)
- **Test Case #13**
  - Asserting if user has successfully logged out by setting its alive field(pid in user table) as nil.
- **Test Case #14**
  - Asserting if the account has successfully been deleted from user table, tweets table, subscriber table, mentions(if any).

PERFORMANCE ANALYSIS

The maximum users we tried to register for were **8000** with **3000** alive(logged in) users and they could send 1 tweet. Apart from this, we tried sending **15 tweets** each for **500 logged** in users with 1000 registered. And **1200 logged** in users with each sending **3 tweets** and 2000 registration. Apart from this, the program can be scaled so as to allow much more users to send many more tweets by simply increasing the sleep timer value. Following is the performance table indicating the total time taken for simulation(sending tweets) along with total registered users and total logged in users who are tweeting. It is based on number of tweets.

NUMBER OF REGISTERED USERS	**NUMBER OF LOGGED IN USERS**	**NUMBER OF TWEETS**	**SIMULATION TIME (seconds)**
500	300	5	18.2
500	300	10	36.0
500	300	15	37.0
1000	500	5	33.0
1000	500	10	32.6
1000	**500**	**15**	**31.2**
1500	1000	3	39.3
2000	**1200**	**3**	**49.5**
3000	2000	1	30.4
5000	2700	1	44.6
8000	**3000**	**1**	**57.9**