1. **Group members**

   Group members: Ziqi Liu 9364-4917
   Enyang Wang 6494-1124

2. **How to run**

   Steps: 1. Enter root folder (project 4.1).
   2. Type commend as: mix test

3. **What is working**

   The goal of our project is to implement a Twitter-like engine and use actor model to simulate several clients and a single-engine process.

   We implement the engine and client by using actor model (GenServer) and several functionalities.

4. **Implementation**

   We implement 8 functionalities, which are *register, login, delete, send tweet, subscribe, re-tweet, query and logout*. Besides, we use ETS table as our data storage.

   - ETS table:

     User: [user_id (string), user_pwd (string), user_ip (pid), connect (Boolean)]
     Subscribe: [user_id (string), user_id2 (string)]
     Tweet: [tweet_id (integer), publisher (string), content (string), retweet_id (integer)]
     Mention: [tweet_id (integer), user_id (string)]
     HashTags: [hashtag (string), tweet_id (integer)]

   - Functionalities:

     Register (*user_id, password*):
     Used to create an account in the ETS data storage. If there is already an account in the data storage with same user_id, the function will return false, else it will return true.

     Login (*user_id, password*):
     Used to login an account with the user_id and password. If system finds an account with same user_id and password exist in data storage, it will return true, else it will return false.

     Delete (*user_id, password*):
     Used to delete an account from the data storage. If the user_id and

password of the account is equal to the user_id and password which stored in the ETS, it will be deleted from the ETS data storage and function will return true, else the function will return false and don't change the ETS.

Send_tweet (*user_id, content, mention, hashtags*):
Used to publish a tweet to the account which follows the publisher and the account which the publisher mentions in the content. Words start with '#' and end with a space will be recognized as hashtags. Words start with '@' and end with a space will be recognized as mention.

Subscribe (*user_id, follow*):
If user *user_id* subscribe user *follow*, user *user_id* will receive all the tweets that user *follow* sends.

Re_tweet (*user_id, content, mention, hashTags, retweetID*):
Users can re-tweet other users' tweet while adding new *content*. Words start with "#" and end with a space will be recognized as "*hashTags*", which make the tweet searchable by "query" function. Words start with "@" end with a space will be recognized as "*mention*", which will notify the user with the mentions name. *retweetID* indicates the tweet_id of original tweet.

Query (*type*, *content*):
Users can search for tweets with specific hashTags, tweets subscribed to and tweets in which the user is mentioned.

5. Test case

Register:

```
.register test
user table(before register): []
user table(after register): [{"A", "a", #PID<0.165.0>, 0}]
```

Process <0.165.0> registers an account with user name "A" and password "a". The last element represents the login status of user, now it is "offline".
{user_id, password, ip, status}

Login0:

```
.login test 0
user table(before login): [{"A", "a", #PID<0.145.0>, 0}]
login table(after login): [{"A", "a", #PID<0.145.0>, 1}]
```

Process<0.145.0> login as user "A" with password "a". The last variable indicated the status. 1 means "online", 0 means "offline".
{user_id, password, ip, status}

Login1:

```
.login test 1
user A sub_tweet: [[0, "B", "abc", nil], [1, "C", "def", nil]]
user A men_tweet: [[2, "D", "lzq", nil]]
```

User A subscribed user B and C. User D sent a tweet mentioned user A. User A receive 3 tweets when he logins.
[tweet_id, publisher, tweet_content, re_tweetID]

Subscribe:

```
.subscribe test
subscribe table(before subscribe): []
subscribe table(after subscribe): [{"B", "A"}]
```

User B subscribes user A.

Send:

```
.send test
tweet table(after send): [{0, "A", "im happy today #everyday @D ", nil}]
hashTags table: [{"everyday", 0}]
mention table: [{0, "D"}]
user B: [[0, "A", "im happy today #everyday @D ", nil]]
user C: [[0, "A", "im happy today #everyday @D ", nil]]
user D: [[0, "A", "im happy today #everyday @D ", nil]]
```

User A send a tweet "im happy today" with hashtag "everyday" mentions user D. Since user B, C subscribed user A, they could receive the tweet that A sent. User D can also get the tweet since user A mentions D in the tweet (@D). The last element in tweet is "nil", which means this is the original tweet instead of a repost one.

Logout:

```
logout test
user table(before logout): [{"A", "a", #PID<0.142.0>, 1}]
user table(after logout): [{"A", "a", #PID<0.142.0>, 0}]
```

The last attribute in the user table represents the login status of an

account, 1 means online and 0 means offline.

Query:

```
.query test
query 'user A': [
  [1, "A", "im sad today #everyday ", nil],
  [0, "A", "im happy today #everyday  ", nil],
  [2, "A", "hello world @D ", nil]
]
query '#everyday': [
  [0, "A", "im happy today #everyday  ", nil],
  [1, "A", "im sad today #everyday ", nil]
]
query '@D': [[2, "A", "hello world @D ", nil]]
```

'Query user A' means querying all tweets which user A published before. Word start with '#' and end with a space will be recognized as hashtags. 'Query hashtag' means querying all tweets that contain that hashtag. Word start with '@' and end with a space will be recognized as mention. 'Query mention' means querying all tweets that contain that mention.

Delete:

```
.delete test
user table(before delete): [{"A", "a", #PID<0.172.0>, 1}]
user table(after delete): []
```

In the test case, there is one account in the data storage before do the deletion and the data storage are empty after deletion.

Retweet:

```
.retweet test
before retweet: [{0, "A", "im happy today #everyday @D ", nil}]
after retweet: [{1, "B", "im happy too", 0}, {0, "A", "im happy today #everyday @D ", nil}]
```

The last attribute in the retweet table means the tweet number of another tweet which retweeted by this tweet. In the test case, user B retweets a tweet of user A. The last attribute of the new tweet of user B will contain the tweet number of user A, which is 0.