

Mini-Tweet Project Report

Parth Daga (18110042) | Rahul Gupta(18110136) | S. Ganesh(18110147)

a. Design Document:

Our major aim has been to provide the user a similar experience as and when they use Twitter. In order to achieve that, we have approached the designing of the interface from an aspect of expected features and functionality.

We had decided to divide the GUI into 3 core windows: the login/register page, the entry page (to fill details and connect to the server) and the homepage of the registered user. In order to efficiently manage design aspects and to keep it as a separate task when compared to feature implementation, we decided to employ the Client/Server model, with TCP as the transport layer protocol. Socket programming in python has been employed due to the ease of operation and presence of necessary, advanced libraries.

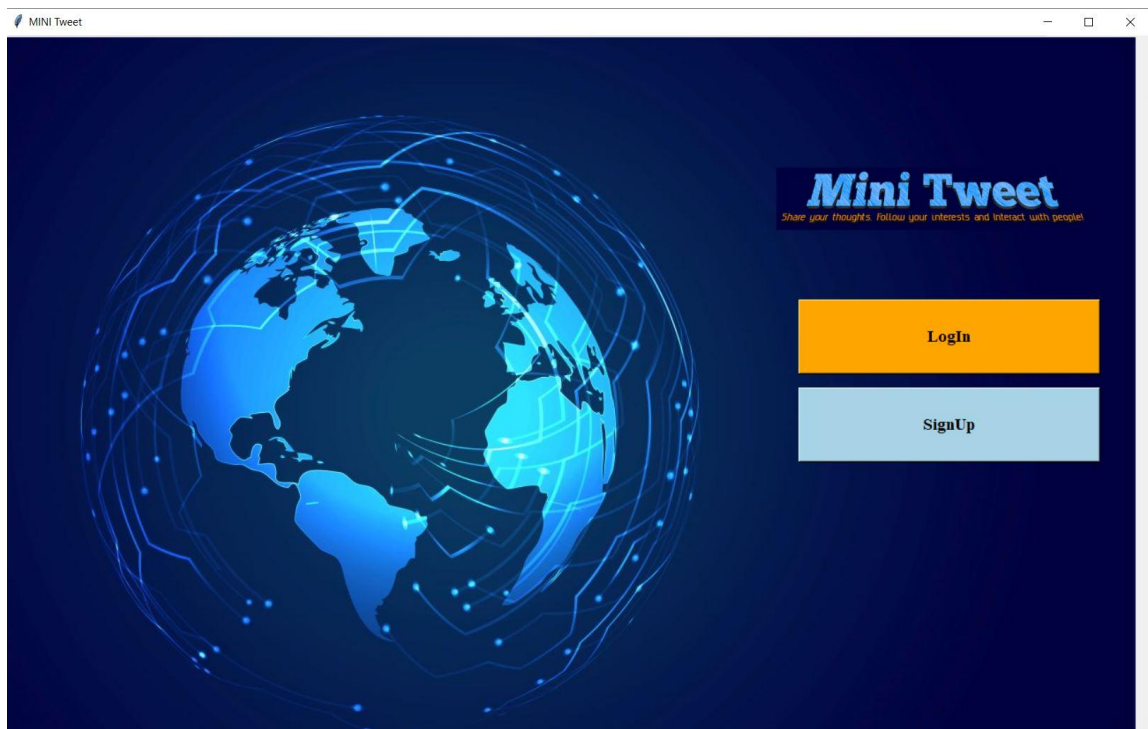
Now, in order to implement a wide range of features like developing an attractive register page, generating necessary dialog/warning windows along with the in-app features that are present in the homepage of a social networking platform, we decided to use Tkinter module in Python. It is a simple, versatile and a powerful interface design tool that caters to needs of building an advanced interface.

Core design considerations:

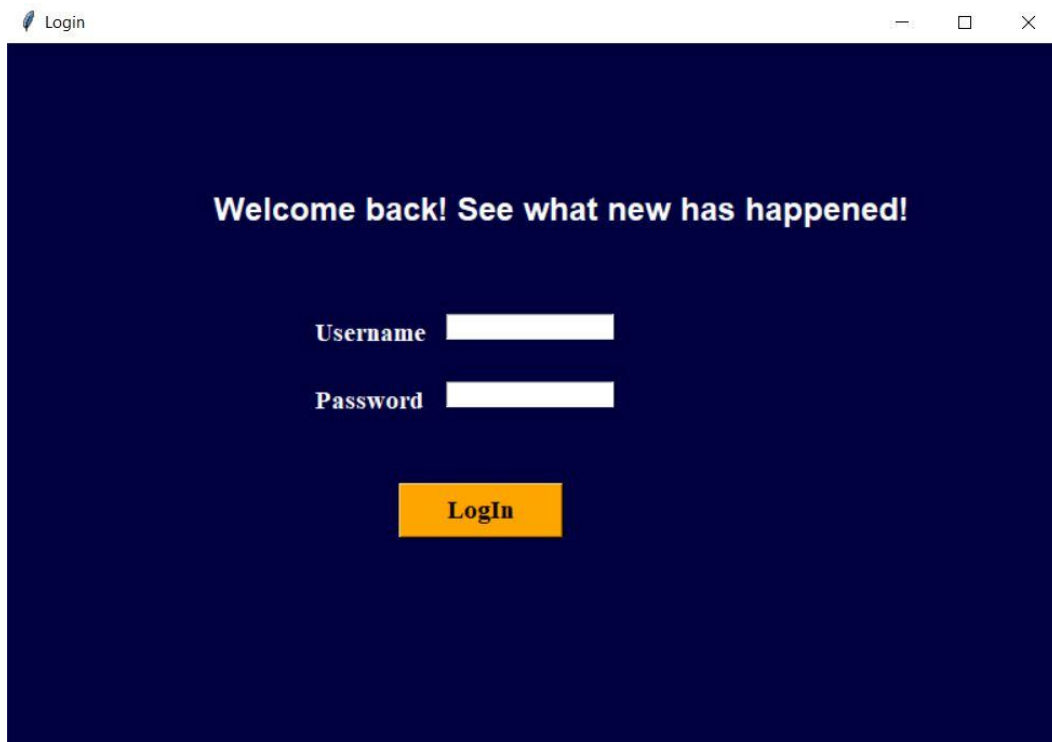
- All of the interface designs are created at the client side. This includes a main login/signup window, which controls all the succeeding windows that the user will encounter until he/she logs out from the system.
- The design of registering/logging in page is done keeping in mind all the possible cases that the user might enter, which will be responded back by a reply/ dialog box message from the server.
- Once logged in, the user will be exposed to the main/home page, where all the features that are part of the platform are present as a whole.
- It consists of the following sections: User details info, search tab, tweeting area, suggestions to follow existing users, check latest uploaded tweets, check the trend list, active user list and many more.

- All of this have been placed as separate blocks placed in a well-thought out manner, as one would witness in a usual networking platform. These are then connected to the functions on the server side at backend, which map efficiently to transfer data.
- Modularity and the appropriate data types have been used wherever possible to conveniently achieve the desired functionality.
- Client side has been kept as stateless, as it just transfers the command to the server and acquires the related information that needs to be provided to the user.
- In terms of network design, we have utilised a Multi-threading server that can accommodate several clients requesting the server simultaneously. We have implemented this through a fat-tree topology, in Mininet. This has been discussed later as part of the features incorporated section.

Opening Window



Login Window



A screenshot of a login window titled "Login Window". The window has a dark blue background. At the top left, there is a small icon and the text "Login". At the top right, there are three window control buttons (minimize, maximize, close). The main content area is dark blue and contains the following elements:

- A white text message: "Welcome back! See what new has happened!"
- A label "Username" followed by a white text input field.
- A label "Password" followed by a white text input field.
- A yellow button with the text "LogIn" in black.

Signup Page

SignUp

Welcome, enter the details to create your account!

Username

Password

Full Name

SignUp

Home Page

Minnie Twitter

Parth Daga
@parth
1 tweets
0 following
0 followers

#LOL **Hashtag**

Search

@biswa: Laughter is best medicine #LOL
0 likes **Like**

Trending
#corona_scare
#INDvAUS
#Elections2024
#LOL
Nothing to show

Welcome to Mini Tweet

Log Out

Suggestions For You
S. Ganesh **Follow** **Remove**
Rahul Gupta **Follow** **Remove**
Sagar Shah **Follow** **Remove**

You are following
No more following
No more following
No more following
No more following
No more following

Online **Refresh**
No more online
No more online
No more online
No more online
No more online

Add Hashtag **Tweet**

New Tweets

@parth: Stay safe #corona_scare
0 likes **Like**

Chat

Send

Get Messages

1. No more message
2. No more message
3. No more message
4. No more message

Minimum system requirements:

- OS: Windows 7 or above,

Mac OS X 10.11 or higher, 64 bit

Linux, RHEL 6/7, 64 bit

- X86 64-bit CPU (Intel/AMD architecture)
- 4 GB RAM
- 5 GB free disk space

List of modules, commands and components along with their functionality

→ Implemented Data structures are mentioned below where they are implemented.

In general, we only used lists, nested lists, dictionaries for all the implementations.

Server side:

Modules used:

- Socket
- Os
- Sys
- Threading
- Rsa
- Pickle
- Cryptography
- Hashlib
- Time
- Socketserver

ClientThread(Thread)

Its associated functions are as follows:

- `def __init__(self, ip, port, clientSocket):`

→ Initiating a new thread for a new user. Defined inside a class.

- `def run(self):`

→ Actual functions that handle all the client-server interaction.

Following are the functions within this function:

- `def list2string(list):`

→ Helper function that convert a list into a string separated with '#'
for smooth transmission over TCP

- `def list2string_not_Hashtag(list):`

→ Helper function that convert a list into a string separated with '*'
for smooth transmission over TCP

- `def follow(user_follow, i):`
- `def remove(i):`
- `def Tweet(username, finalTweet, hashtags, i):`

Now, the key functionality of the server is determined in a loop value and performs operations based on the state of the user. We have used conditionals and logical data types to perform the same inside a "while" loop. Following is the detailed list:

- `if state1 == '1':`

→ This if else statement checks if the input password is correct or not

1. `if username not in dct:`
2. `else:`

- `else:`

1. `if username not in dct:`
2. `else:`

- `if (check == password):`

1. `if`
`(self.clientSocket.recv(1024).decode('utf-`
`8') == "Ok"):`
- 2.

Now there are several operations performed again in a loop once the user is logged in.

→ All the below state is the client trying to contact the server and get the necessary information back to it.

→ The purposes of all the functions are self-explanatory and the nomenclature for both the client and server side are chosen to be the same wherever possible to avoid unnecessary confusion.

→ Once the user logs in, the client side send a unique message with a String that can be Like, Dislike, followOne or more. The server decodes and checks the message and sees what the client wants and replies accordingly.

- `if (state == 'followOne' or state == 'followTwo' or state == 'followThree') :`
- `elif (state == "refresh_followers") :`
- `elif (state == 'removeOne' or state == 'removeTwo' or state == 'removeThree') :`
- `elif (state == 'Tweet') :`
- `elif (state == 'refresh_tweets') :`
- `elif (state == 'Like') :`
- `elif (state == 'Dislike') :`
- `elif (state == 'Refresh_likes') :`
- `elif (state == "Tweets?") :`
- `elif (state == 'Existing Tweets') :`
- `elif (state == "searchUser") :`
- `elif (state == "searchHashtag") :`
- `elif (state == "follow") :`
- `elif (state == "unFollow") :`
- `elif (state == "followingGUI") :`
- `elif (state == "trending") :`
- `elif (state == "logOut") :`
- `elif (state == "online") :`

And if the user fails to login correctly with proper credentials, we provide the message to the user in the same case through an else statement.

Data Structures Used :-

- We have also made a list of clients stored on the server side using a list as follows: `threads = []`, on which we perform operations such as `append()`, `join()`. All the data are stored in a nested list named `Data` in the server.
- Multiple list data types are used for several functionalities like online users, tweet databases and also for the messaging implementation. The actual logic for how they are used can be seen from the code itself.
- Also, we have used the dictionary data type to store the information related to the registered users as well as provide room for adding more users. This dictionary is used to check whether the input password matches or not.
- Also, a dictionary is used to maintain the count of any particular hashtags, to easily get the most trending tweets.

Client side:

Modules used:

- Socket
- Os
- Sys
- Threading
- Rsa
- Pickle
- Cryptography
- Hashlib
- Time
- Socketserver
- tkinter, tkinter.messagebox, tkinter.font
- PIL

List of functions used:

- `def LogInWindow() :`

1. `def LogIn() :`

→ Letting the user to log in

→ Below features can only be used once the user is logged in

- `def logOut() :`

→ A feature to logout.

- `def followingGUI() :`

- → Created a section to show who you are following

- `def trending() :`

→ Created a section to show which tweets are trending

- `def get_online() :`

→ Created a section to show who is currently online

- `def send() :`

→ Initiating a chat or sending a message to someone

- `def get() :`

→ Get the three most recent messages you have received

- `def search() :`

→ Search option based on username(for users) and hashtags(for tweets)

```
if searchState == 'Username':
```

1. `def follow() :`

→ Option to follow a searched user

2. `def unFollow() :`

→ Option to unfollow if already followed

```
elif searchState == 'Hashtag' :
```

→ Get 5 most recent tweets with that hashtag

```
1. def refresh_likes1(finalTweet):
```

→ Refresh like for the first search tweet

```
2. def refresh_likes2(finalTweet):
```

→ Refresh like for the second search tweet

```
3. def refresh_likes3(finalTweet):
```

→ Refresh like for the third search tweet

```
4. def refresh_likes4(finalTweet):
```

→ Refresh like for the fourth search tweet

```
5. def refresh_likes5(finalTweet):
```

→ Refresh like for the fifth search tweet

```
6. def likeOne(templ1):
```

-> Increase like of the first searched tweet by one

```
7. def likeTwo(templ2):
```

-> Increase like of the second searched tweet by one

```
8. def likeThree(templ3):
```

-> Increase like of the third searched tweet by one

```
9. def likeFour(templ4):
```

-> Increase like of the fourth searched tweet by one

```
10. def likeFive(templ5):
```

-> Increase like of the fifth searched tweet by one

```
11. def dislikeOne(templ1):
```

-> Decrease like of the first searched tweet by one

```
12. def dislikeTwo(templ2):
```

-> Decrease like of the second searched tweet by one

```
13. def dislikeThree(templ3):
```

-> Decrease like of the third searched tweet by one

```
14. def dislikeFour(templ4):
```

-> Decrease like of the fourth searched tweet by one

```
15. def dislikeFive(templ5):
```

-> Decrease like of the fifth searched tweet by one

```
16. def toggle_1():
```

-> Toggle between like and unlike

```
17. def toggle_2():
```

-> Toggle between like and unlike

```
18. def toggle_3():
```

-> Toggle between like and unlike

```
19. def toggle_4():
```

-> Toggle between like and unlike

```
20. def toggle_5():
```

-> Toggle between like and unlike

- `def refreshWindowsearch():`

- `def refresh_followers():`

-> Refresh the followers

- `def up_Tweet():`

--> Support function for tweeting

- `def removeOne():`

-> Remove the first suggestions from suggestion for you

window

- `def removeTwo() :`
 -> Remove the second suggestions from suggestion for you window
- `def removeThree() :`
 -> Remove the third suggestions from suggestion for you window
- `def refresh_followers() :`
- `def followOne() :`
 -> Show the first suggestions in suggestion for you window
- `def followTwo() :`
 -> Show the second suggestions in suggestion for you window
- `def followThree() :`
 -> Show the third suggestions in suggestion for you window
- `def tweet() :`
 -> Actual function for tweeting
- `def newTweet() :`
 -> Supporting function to tweet
- `def gettweets() :`
 -> Get/Refresh the tweets
 - `def refresh_likes1(finalTweet) :`
 - `def refresh_likes2(finalTweet) :`
 - `def refresh_likes3(finalTweet) :`
 - `def refresh_likes4(finalTweet) :`
 - `def refresh_likes5(finalTweet) :`
 - `def likeOne(templ1) :`

- `def likeTwo(templ2):`
 - `def likeThree(templ3):`
 - `def likeFour(templ4):`
 - `def likeFive(templ5):`
 - `def dislikeOne(templ1):`
 - `def dislikeTwo(templ2):`
 - `def dislikeTwo(templ3):`
 - `def dislikeTwo(templ4):`
 - `def dislikeTwo(templ5):`
 - `def toggle_1():`
 - `def toggle_2():`
 - `def toggle_3():`
 - `def toggle_4():`
 - `def toggle_5():`
 - `def refreshWindowtweets():`
-
- `def SignUpWindow():`
 -> Forms the signup window
 1. `def SignUp():`
 -> Sends data to server required for signing up

b) Features Checklist

- **Basic Features:**

- a. **Any Client/user should be able to register and set up an account with Mini-Tweet and log in.**

→ A separate Window for login and signup has been implemented. The login feature only checks if the input password is correct with the password used to create the account. The signup checks if the input username is taken and creates a page for a unique new username.

- b. **Should be able to log in, get the updates, and Logout.**

→ After logging in, the user can see the previous tweets, count of followers, following and tweet, and much more information and log out when he/she wishes to end the session.

- c. **Should be able to search registered users, follow/unfollow any users, and control add/delete followers.**

→ Searching any registered user with the username gives the option to follow the username if not already followed, and the option to remove/unfollow if already followed. The suggestion window also lets you follow the rest of the users.

- d. **Support Users to post tweets, and categorize the tweets with specific hashtags.**

→ A separate window is provided to post any tweets with any number of hashtags which are all separated with a space between them and starts with '#'. The posted tweet will appear just below in the tweets section.

- **Advanced Features:**

- e. **Hashtags: Allow users to search and display tweets under specific hashtags. Show the Top 5 Trending hashtags.**

→ The search query has a toggle between next to it which asks if you want to search for a username or a hashtag. If the state of the toggle is a hashtag, the search will output a maximum of 5 most recent tweets with an option to like it. Below the list of searches, there is a list of trending hashtags displayed.

- f. **Should be able to determine the list of active/online followers and initiate a chat session with any follower.**

→ The bottom right part of the windows displays a maximum of 5 users that are online right now. A refresh button is also there to refresh the list of online users. Only those users appear in the list which you have followed

- g. **Retweet: Support users to use other users' tweets and post the retweets.**

→ You can see the tweets of other users directly in the tweet section. Each tweet starts with @username to see how the owner of that tweet. You could just type the tweet in the tweet box to post it. But as of now, there is no retweet option to explicitly post the tweets.

h. Scale to a concurrent server that can handle several client requests.

→ The entire server is implemented on a threaded server that can handle multiple clients at the same time. A limit of 100 connections is given to the server to now, which could be increased at any time. A separate mininet topology has already been implemented as a level 4 fat-tree topology with a maximum of 16 users at a time.

NOTE: To use the mininet topology just change the server host from 'localhost' to '10.0.0.17' and use host on node 17 and multiple clients on any node from 1 to 16.

- **Supporting a minimal set of security features.**

- i. Users should be able to authenticate with the server before trying to access any of the features.**

→ Any user can not see any tweet or use any feature unless he/she has logged in to his/her account using the correct password. Also before sending private messages or passwords we are encrypting and decrypting the messages and passwords as well.

- j. When a user is prompted for a login password, the user input for the password should be obscured/masked.**

→ The password is masked before inputting in both the login and signup window.

- **Other Bonus features (Grace points):**

- k. Twitter Page: You should be able to pin some tweets so that users twitter page can show those pinned tweets when someone visits the user's Twitter page.**

→ We could not make a separate Twitter page as of now due to time constraints. But we do have some ideas on how to implement it.

- l. Group Chats with all or a subset of followers.**

→ We can not implement a separate way to make a window and let all the users connect and text with each other. Still, users are free to text each other. If texting multiple users at once, as @user1 @user2 @user3 counts as group chat, we could have easily implemented it but we believe an ideal group chat is where all users can see who messages to everyone, thus we choose to skip this part.

- m. Good attractive GUI is always a bonus! -**

→ We have developed an intuitive, easy to navigate, and attractive GUI. This has also been done keeping in mind the diversity of functionality implemented within the platform. Hope you like it :)

A note on Encryption feature used:

The project uses the RSA systems of encryption generating the two keys, private and the public key. The keys are randomly generated whenever a client tries to connect with the threaded server. We are also using the fernet module to randomly generate the key for the implementation of the encryption and decryption on both the server and the client. The transfer of the fernet keys are masked by the library 'hashlib' and using the module 'sha256'. Once the fernet keys are transferred to each other it is used to encrypt the data on the client side and decrypt on the server side whenever there is a communication of critical data using socket programming.

NOTE :- Dependencies and Deployment are mentioned in the Readme file present in the compressed archive.