מטלת סוף מודול פייתון

מגיש: שון אורלוב

I chose the first task: building a server chat that allows multiple connections, the clients shouldn't necessarily be in the same network as the server. If a client sends a message it gets broadcasted to everyone else. When connecting to the server the client goes through a verification process: the server asks for a username and a password in order to see if the client is permitted to enter the server or not.

Solving part 1:

In order to solve the task, I needed to write a script for the server that would do the next things:

1. listen for incoming connections (also being accessible to connections from other LANs) and accept them.
2. Have stored information about clients that can enter the server and if the client enters a wrong username and password or a username of a client that is currently logged in (already existing client) then he's not permitted to enter the server and gets disconnected (after a few attempts).
3. The server will have to be constantly listening for messages and if there is a message then send it to every client except for the sender of the message.
4. The server would also need to send a prompt that will include an explanation of how to use the server and how to disconnect from the server if wanted by the client (e.g type c.exit to quit the server).
5. The server would need to have a block of code that would be responsible for empty reads from the client and if there are 3 or more empty reads (no data) then the server will assume that the client disconnected and close his connection.

Writing the script:

When writing the code I needed to import 3 modules for the server script:

1. Socket - create a socket with a default of IPV4 and TCP proto for the server and bind it with ip 0.0.0.0 (accepting all connection) and random port like 12345 (not a well known port) and then accept connections.
2. Select - using select to iterate through a list of sockets (including the server socket). If the current readable socket is the server then accept new connections but if it's a client socket then listen for incoming messages and broadcasts to everyone else.
3. ngrok - I used ngrok to get a public URL that will allow clients from other networks to establish connection with the server.

Here is a link for my server script in GitHub:
https://gist.github.com/shon687/b74322627a991efa040cbaa306768d4e

Solving part 2:

The second script is for the client.
The client script should do the next things:

1. Connecting to the server even if the client is located in a different network.
2. Ask the user for a username and a password and then send it to the server to get verified.
3. After getting verified successfully, entering the messaging phase: constantly listen for incoming messages and get an input from the user and send it as a message to the server.

Writing the script:

When writing the code I needed to import 3 modules for the client script:

1. Socket - using socket to create a client socket with a default of IPV4 and TCP proto, then connect to the host IP and Port.
2. Requests - using requests to send an HTTP GET request to ipify (a free API service that returns my public IP address. the service responds with a JSON object that contains my IP address) to get my public IP and send it to the server.
3. Threading - the threading module has the “Thread” function. This function accepts another function that I created and lets it run in the background without preventing the upcoming next lines from happening. The main thread would be a loop that is constantly asking for a user input and sending it back to the server as a message. Threading gives me the option of doing two tasks at once. The client script should always be asking for an input from the user (this is in the messaging phase after the verification part) and at the same time listen for data (messages) from the server in order to get the messages of other clients.
   Note: this module really helps due to the fact that if I were to write the script without it, the task wouldn’t have been possible. Asking for a user input (means that at that moment I’m not receiving any data from the server) if there is anything that the server would like to send the client it simply wouldn’t be possible for the client to receive it because the client script is currently stuck asking for an input. Threading creates sort of a different stream where the code (function) of receiving messages from the server would run and not come before or after the user’s input and only then be executed.

Here is a link for my client script in GitHub:
https://gist.github.com/shon687/d484a0c04d9f23b41574208db0936f37