A simple hybrid (Login server) Network which makes use of the cherrypy module.

Python server

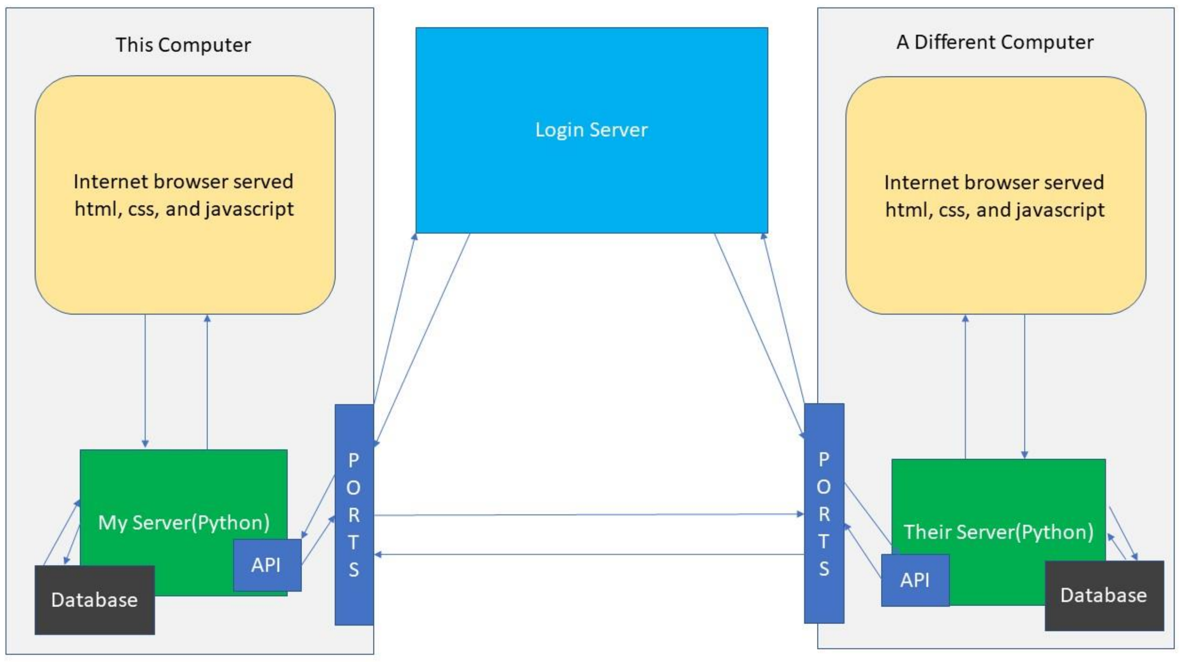
COMPSYS 302 Project

Henry Shen

General Overview

We were required to develop a social media network for use within a company. The client was concerned about the possibility of electronic surveillance being conducted to gain access to confidential information. To reduce the risk of this occurring, we proposed the Hybrid (Login server) network which would communicate between users via peer-to-peer methods. The developed system is capable of logging into the server, given that you have supplied the correct credentials, and viewing a list of people who are also using this network. The user can modify their profile details and retrieve other users' profiles. Messages and files can be sent between online users.

Top-Level View of the System



Development Issues

When developing the file sending feature, I created a file browser that selects a file to send to the recipient. However, I ran into a road block which slowed down development a bit. When this file was submitted as an argument to the sendFile method, I did not know what type of object it was, or how to deal with it. Research online and experiments I did on my own, did not yield any useful results and I was lost on what to do. So, I discussed it with my peers and I was able to get an explanation on what it was and what I could do with it. With this knowledge, I could implement a file sending method that worked with file browsing.

During the early stages of development, the python server would only report to the login server when the user logged in. This caused the user to be logged out after a short period of time, which was very inconvenient when testing. I needed to have the server automatically report to the login server while a user was logged in. To do this, I made use of threading which would report to the login server every 30 seconds. This thread would not close properly on application exit, which meant it would not close the terminal unless forced to terminate. To fix this issue, I set its daemon attribute to True, and from then it could close without a problem.

Peer to Peer Methods

Peer to peer methods allow nodes to communicate with each other without going through a main server first. This means that there is no need for a dedicated server, as each node is both a server and a client. This also helps with network traffic, because the nodes are communicating directly. If a node has problems, it will not affect communication between other nodes, unlike when the main server goes down.

Importance of the Protocol

The protocol was necessary so that each application would be able to communicate properly. Each developer would follow the guidelines from the protocol, so that arguments and API names would be consistent. Without it, the different applications would only be able to communicate with other students who discussed the project together. The manager brought all of us together to discuss what was necessary to have in the protocol, specifically which features we thought would be important to have in this application.

Why Should We Use Python?

Python was suitable for the project because it has the module called cherrypy which allows developers to build web applications easily and in less time. Due to the time constraints for this project, and the fact that we needed to learn everything about web development while we were developing the system, meant that cherrypy was the perfect tool to help us along. Also, Python is a high-level programming script language which has some very powerful operations and functionalities in its standard library. A program written in Java or C++ may take many times more lines of code than if you were to develop it in python. This promotes readability of code and makes it much easier to understand for someone new to the language.

Descriptions of the Methods

Signin(username, password, location)

Before reporting to the login server, this method will hash the password using SHA256 hashing algorithm. Then it will use the location argument to determine which IP needs to be reported to the login server. If the login server responds with the code “0”, we know that the user has successfully logged in, and they can be redirected to their profile. A thread will start which continuously reports to the login server every 30 seconds. If the login server responds with anything other than “0”, the user will be redirected back to the login screen and an appropriate message will be displayed.

Logoff(username, password)

This method will log off the current user, given that the credentials are correct. The thread will be stopped, and all sessions will be expired.

getProfile()

This is an API which is called by other nodes to retrieve a profile from this node. The profile information is put into a dictionary, and then JSON encoded before returning it to the other node.

writeInfo(name=None, position=None, description=None, picture=None, location=None)

This method is used to save the logged in user’s information which they had input in the profile editing page. The user information is stored into the database. The picture argument only accepts the .jpeg and .png image extensions.

saveInfo(UPI)

This method is used to retrieve and save other users’ profile details to the local database. The other node will return a JSON object which contains the user’s information. The JSON object is decoded and the data is saved to the database. If this was successful, the user will be redirected to the other user’s profile page, otherwise, they will be sent to the error page.

sendMessage(message, destination)

This method is used to send a message to another node. The message and all other relevant arguments are put into a dictionary, and JSON encoded before being sent to the other node. To send a message, we call the recipient’s receiveMessage API and include the JSON encoded data. If the other node returns the code “0”, the message was successfully sent and the message will be saved to the local database. If it fails to send, the user will be redirected to the error page which will display an appropriate message.

receiveMessage()

This is an API which is called by other nodes that want to send this node a message. The message data is given as a JSON object which needs to be decoded. Once it has been decoded, it is saved to the local database. Finally, it returns “0” to the caller

sendFile(destination, fData)

This method will send the chosen file to the recipient. The input file data is saved locally to the working directory, so that it can be displayed in the messages later. The file is encoded in base64 and then put into a dictionary with all other relevant arguments. This dictionary is JSON encoded and the recipient’s receiveFile API is called including this JSON object. If the response code is “0”, then the file was sent successfully and the user is redirected to the messaging page. Otherwise, they are sent to an error page with an appropriate message.

receiveFile()

This is an API which is called by other nodes that want to send this node a file. The file is stored in the JSON object which is sent to this node, and it needs to be decoded from base64. Once it has been decoded, it is saved locally to the working directory to be displayed in the messages later. Finally, it returns “0” to the caller.

ping()

This is an API which is called before any other requests to another node. It simply returns “0” to tell the caller that this node is still active and is ready for communication.

Suggested improvements

- Message notifications. The user currently has no way of knowing if they have been messaged by someone unless they were to check the python terminal. For a messaging application, it is not very useful if the users do not reply as soon as possible.
- Offline messaging. The user currently cannot contact an offline user. Often, you will find that the user you want to talk to is not always online at the same time as you. If you had a very important message to relay to them, it would be very inconvenient if you had to wait until they logged on.
- Refreshing Messages. The user needs to manually refresh the messaging page to see new messages from the other user. This will slow down the conversation between users, and also making it quite annoying to use.