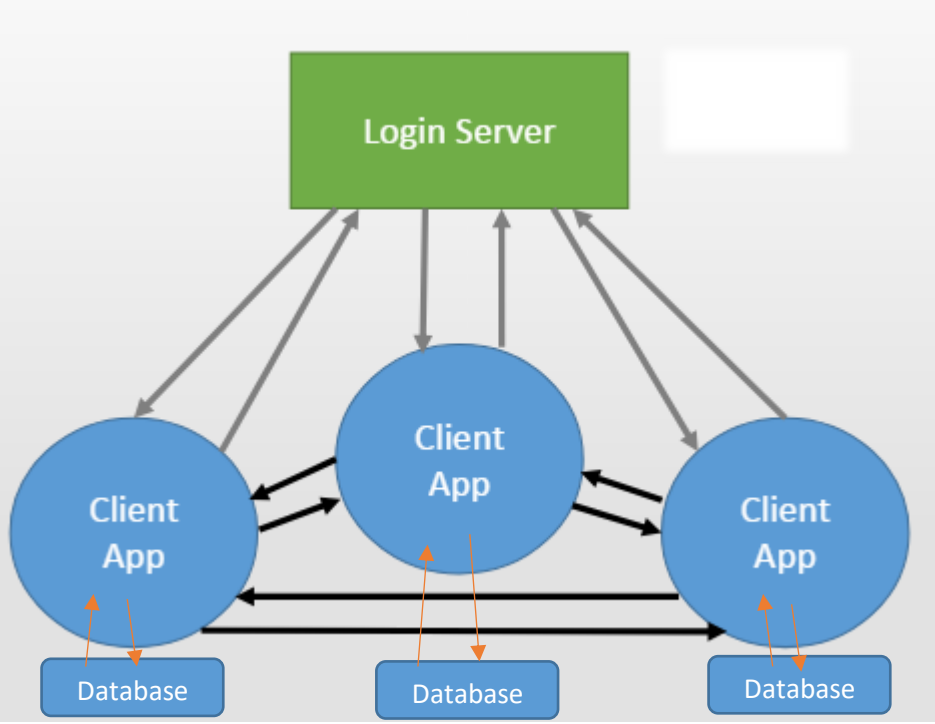
Compsys 302 – Python Project – Final Report

BY SAVI MOHAN UPI: SMOH944

How the developed system has met the client's requirements

The developed application has met the client's requirements. The application allows a user to log into the system. A user using the application can send and receive both messages and files (images, audio, video, PDF etc) from other users in the network. The user can view sent and received files and messages via the user interface. The user is also able to see who is currently online in the network and is able to view each online user's profile including their own. The user is also able to modify their own profile page.

Top Level View of the System



There are 3 main components, the Client App, the databases and the login server. The login server keeps track of logged in users and their ip addresses. The server will return this data if requested. The client apps use the data from the login server to communicate with each other through their ports. The client apps constantly read and write data to their individual databases.

Significant issues during development

A few significant issues were encountered and overcome during the development of this application.

One significant issue that arose during development was that a number of functions that needed to be carried out were slowing down the application by preventing other functions from carrying out their tasks. In order to solve this issue, I placed the time consuming functions such as the retrieveMessages function and Offline File/Message sending functions in their own threads, so that their execution didn't slow down the rest of the program.

Another significant issue that was encountered during the development was the database getting locked and being inaccessible due to multiple threads trying to access it at the same time. At first I considered getting rid of threading in the application all together, but I realised that a number of functions in the application needed to run on their own thread so as to stop them from 'bogging down' the rest of the application. In order to solve this problem, I ensured that every function that wrote or read from the database, opened up a database connection for the least amount of time possible. For example, instead of opening a database connection before a large for-loop and closing the connection after the for loop had finished executing, I placed the code for opening and closing the database connection inside the for-loop so as to prevent the database connection from being open continuously. Thereby allowing multi-thread access to the database.

Features that improve functionality of the system

On top of the basic functions of the system such as being able to send and receive files and messages, being able to view your and other users' profiles, and being able to edit your own profile, the application has a number of features that improve its functionality.

For example, the application makes good use of databases to quickly store and retrieve information about messages/files that are sent and received, user profiles, user login details and user request tallies for rate limiting purposes. The application also has embedded video and audio players for displaying sent and received audio and video files. The application also allows the current logged in user to set their user status to 'Online', 'Offline', 'Away', 'Idle' or 'Do Not Disturb'. The application is also able to request and display this user status data from other users on the network. On top of being able to log a user out of the server by pressing the sign out button, the application will also log out the current user automatically on application exit.

The application gives users the option to send and receive messages in a markdown format, so as to allow for the communication of text formatting.

Multiple user sessions can be simultaneously supported by the application.

With regards to security, the application implements rate limiting to prevent the user from getting spammed or to protect them from a potential Denial of Service (DoS) attack. Also with regards to security, the application makes use of session data so that even if multiple user sessions are occurring on the same application, the users won't be able to see each other's confidential data such as their username or hashed Password.

The application has a user-friendly and easy to navigate user interface. Navigation between pages is easy due to a menu of buttons at the top of every page. The messaging page of the application is styled in a similar format as existing apps such as Facebook messenger, thereby making it easier to understand and use. The user interface is also compatible cross-browser (Chrome & Firefox), so users don't need to restrict themselves to one browser to use the application properly.

Another feature that improves functionality is Offline Messaging. If the logged in user wants to send a message to a destination user who is currently offline, they can send that message

to other nodes in the network, so that even after the logged in user goes offline, when the destination user logs in to the network they can request these nodes for the offline message that the previously logged in user sent.

The application is also designed to fail graciously when interacting with sub-standard clients in the network so that the application doesn't crash and can keep functioning properly.

Peer-to-Peer methods and their suitability for this application

Peer-to-Peer methods have a number of advantages that made them suitable for this application. They are generally quite secure since there is no central server that could become compromised. Peer-to-Peer methods are more reliable than having a central/login server as central dependency is eliminated. This means that the failure of one node doesn't impact the functioning of other nodes. Whereas in the cases of the central/login server network models, if the server goes down then the whole network goes down. Although Peer-to-Peer methods don't work as well in large networks, since our network will handle less than a 100 people, the scaling issue doesn't matter.

The Protocol and its suitability for the system

I thought that the protocol that was developed was very suitable for the system. I think it was good that both the login and application protocols allowed for different 'tiers' of communication, i.e. encrypted vs non-encrypted communications. This means that users with different encryption/hashing capabilities can still communicate with each other. I also thought that it is was good that a number of the application protocol APIs were optional, so that people could choose the APIs that they wanted to do.

Process of developing the protocol and its suitability

The process of developing the protocol initially started with class discussions, followed by more concrete prototype proposals from various groups in the class. This was then followed by more class discussion and debate about what kind of protocol and network model we wanted to do. After this a formal application and login server protocol documents were made. These formal documents were then incrementally updated following class discussions and changes in the requirements. Although the changing protocol documentation did pose a challenge, I also thought that it was more representative of what its like to work on a project in a company where the project requirements often change quickly.

Suitability of the suggested tools for the application

CherryPy provided a very suitable web application framework to develop on. By abstracting away the HTTP protocols, it made developing a web application much easier. CherryPy manages sessions for logins, which makes it a lot easier to support multiple user sessions on the same application.

Python was a very suitable language to write the server backend in. I was able to write functions (especially computation functions) in significantly fewer lines that I would have in other languages. Also the large number of in-built python libraries such as urllib and urllib2 made web development much easier.

Future development

There are a number of avenues for future development. In order to improve the security of the application further, highly secure encryption standards such as AES could be implemented to make it even harder for the application's communications to be deciphered. Also, the integrity of the messages sent between applications can be verified with a high level of certainty by using advanced hashing standards such as scrypt and bcrypt. Furthermore, in order to protect user accounts that may have weak passwords, or may have had their login credentials leaked or stolen, 2-factor authentication can be used to provide an additional layer of security. Also, to further improve local data security, locally stored files and databases could be encrypted.

Group messaging is another possible future improvement to allow for discussion and conversation between more than just two people at a time.

Another possible future development is the implementation of a fall-back peer to peer networking system, which will add redundancy to the network in case of a Login server outage.

A search function for profile pages and messages would be another worthwhile feature for future development as it would cut down on time spent trying to find a certain user or message.