CS302 P2P Social Network Report

Kenny Li

1.0 Introduction

LiChat is a social network platform catered to the CEO of a company, having the hopes of reducing the risk of unwanted monitoring of their communications. This network will allow its users (company employees) to directly and securely log on to the system, to send and receive messages and files between each other. The user will be able to log on to the system, and once their login has been authenticated, they are able to view the other employees who are also logged in and their profiles.

2.0 Project Requirements

A few core requirements were set by the client. These include:

- Allows a user to log into the system
- The system is capable of automatically find other users on other computers
- The user can create and maintain a simple profile page
- The user can send and receive messages, images, audio and PDF files.

LiChat has incorporated all these core requirements into the finished system. The opening page greets the user with a login screen and once the user is logged in, they will be greeted with a page displaying their profile. While on this page, the user can also view all the other users on the login server and has the option to search for other user profiles, navigate to the messages page and log out. When on the messages page, the user can start sending messages and files to other users.

3.0 System Overview

As shown in figure 1, this system uses multiple components together including Python, HTML, CSS and JavaScript. The main component used is Python, as it allows the client to interact with the front end and the database. Python communicates with other nodes and hosts a server that others may access. It is also used to receive requests from other nodes and communicates with the login server. On the front end, HTML, CSS and JavaScript are used. HTML and CSS allow the user to interact with the system when accessing the website while JavaScript constantly refreshes data displayed to the user. When passing information to HTML from Python, the external library Jinja2 was used.

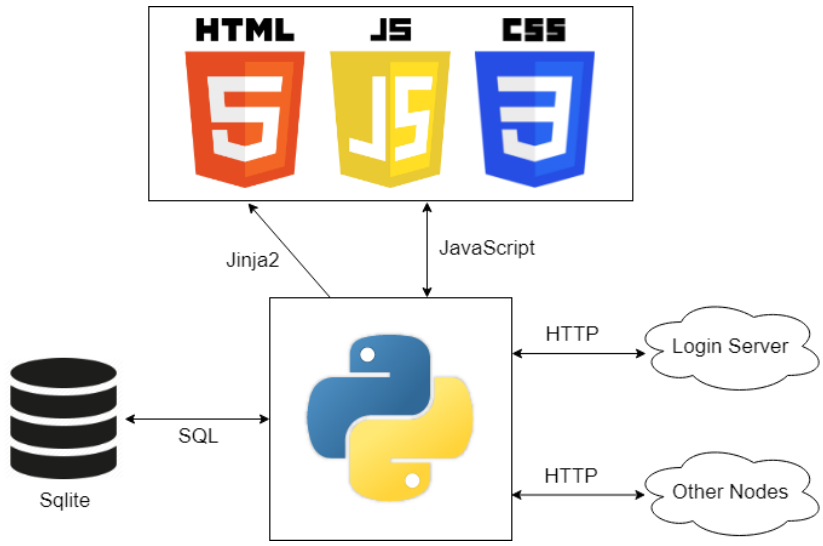

Figure 1: Top level view of how the system works

4.0 Significant Challenges

While working on LiChat, many challenges arose. These included handling errors and integrating the different components used in the system.

4.1 Error Handling

When working on the project, one of the most tedious issues was error handling. Initially, many HTTP and URL errors occurred, and these were relatively easy to handle. Many errors also arose when testing with other users. This proved difficult as the other users were still developing their system, therefore finding the direct source of the error was extremely tedious. There were also many errors that were very hard to spot. An example of this was using functions in HTML with Jinja2 where the HTML containing a Jinja2 function was commented out. The only way of figuring out the error in the code was to observe the console through the browser.

4.2 Integration

When starting to work on the front end of the system, a lot of research and learning was done. With the research done, many options to implement the front end came up. Something that proved extremely difficult was to think about how to pass information from the Python code to show on HTML. This continued to prove difficult when it was required to extract the information from the database. To work through these challenges, extensive testing and trial and error was done. Also, once the Jinja2 library was learned and utilised, communicating between Python and the front end become much simpler.

5.0 Features

In addition to the required specifications, LiChat also adds more functionality to the social network platform. These extra features include:

- Automatic refreshing of messages
- Use of SqLite3 database to store user details, messages and profiles
- Displays confirmation of message receipts
- Logs any errors that may occur when the system is running
- Support for Unicode
- Logs out from the server when the system shuts down
- Rate limits the system to protect against DDOS attacks
- Uses threading to communicate with the login server regularly
- Uses good page templating with Jinja2
- Has a nice and simple user interface
- Has the capability to run multiple users simultaneously
- Fails gracefully when interacting with substandard clients
- Defends against HTML, script and SQL injections

6.0 Discussion

6.1 Peer-to-Peer Methods

During the initial discussion, the central server, hybrid and pure peer-to-peer models were analysed. The chosen model was the hybrid server using a login server to only store user credentials. The rest of the system was proposed to be peer-to-peer. Having the peer-to-peer aspect of the system meant that the communication between clients is more secure, as data is sent directly to the receiver's IP. Another advantage of the peer-to-peer model is that it is more robust. This meant that even if one node failed, the whole system would not be negatively impacted. Overall, the hybrid server model was chosen, as it is more secure and robust while being a more cost-effective option.

6.2 Protocol Suitability

When developing the protocol in the initial teams, many people were confused as the terminology was completely new to them. However, after discussing, learning and analysing the chosen server model, the protocols became clearer to make.

The final draft of the protocol did not differ too much from the initial one. All the API's were laid out clearly and were relatively easy to follow. They provided sufficient functionality to reach the client's minimum requirements while having the capability and flexibility to implement even more features.

6.3 Tools and Development

Overall, Python proved to be the main tool used throughout this project. It was used to handle HTTP requests with the web server framework CherryPy. CherryPy also proved to be a very powerful tool

and was core to the development of the system. It gave a great deal of flexibility when it came to implementing APIs and serving content.

On the front end, Jinja2 was used to pass information into HTML. It was chosen as it provides consistent tag syntax, HTML escaping and template inheritance. HTML escaping is extremely useful when sanitising unwanted messages and scripts especially in the context of a social network platform. Template inheritance is also very helpful when it comes to passing dynamic data. The Bootstrap CSS library was also utilised for the login form, as it provided a nice and clean layout. JavaScript was also used but proved to be a bit confusing at first. Due to this, minimal JavaScript was used and the main use of it in this system was to make parts of the page load in certain ways when it loads.

6.4 Future Improvements

Although the current system is functional to the client's requirements, much more can be done to improve it. The first are to have more features to improve functionality and security. These features include:

- Use of high standard encryption to improve security to the system and database
- Two factor authentication to improve security to the user
- Deleting and clearing messages
- Group conversations
- Global message board
- Fall back peer-to-peer networking if the login server is to go down
- Improved front end using more JavaScript to call necessary functions

Another improvement that could be made is the way messages are loaded onto the page. Currently, JavaScript is used to load the entire history of messages stored on the database for the selected sender and receiver. This starts to become a larger issue once the message history becomes larger. A way of implementing this in the future would be to only load the new messages that have been added to the database.

7.0 Conclusion

Overall, LiChat had met all the requirements set out by the client while having added functionality. Many different tools were used during the development of the system, all of which were being learnt for the first time during this project. Although many improvements can still be made to the system, the current system provides a clean user interface and is able to be used efficiently within a small-scale environment.