302 Python Report

Rohan D'Souza

rdso323



# Abstract

The aim of this project was to build a peer-peer network to increase the security of the company by preventing rival companies from accessing and stealing private data. The main programming language used was python, but parts of HTML, CSS and JavaScript were required along the way. The project was made to be challenging as most students had little to no experience in building a server. However, after hours of research and testing, a working and functioning interface was created.

# Meeting Requirements

I believe the system has checked all the requirements to be suitable for use for the client. The service allows communication between peers using a login server. It is secure, ensuring only select members (employees) have access to the system by requiring them to use their provided username and password. All online users are displayed on the main menu to make certain each employee knows who they can interact with. Users can message one another to inform that person of an important task that needs to be completed. Files up to 5Mb in size can also be sent; this includes images, videos, docs and many more. To make the system more personal, each user has a bio which they can edit to their pleasing. Some of the parameters include “Location”,” Position” and “Name”

# Notable Features

There were several features that were implemented to help with the functionality of the system. One of the highlight features include a simple GUI. It was designed with simplicity in mind to ensure users can see the information they desire and can get quick access to any tools at their disposal. All the main features can be accessed through the main menu.

Threading was another essential feature which helped improve the system tremendously. Previously, users would get logged out of their session within 5 minutes regardless of their current activity. This led to several errors occurring, as users would continue sending messages and files once offline as they believed they were still online. Threading solved this issue by allowing the user to report back to the Login Server every 30 seconds. This ensured they stay logged in until they physically clicked on the Log off button or cut the connection between the server and themselves.

With the use of JavaScript, auto refreshing was implemented. This meant every 20 seconds the user would get an updated list with all the online members, thus preventing the hassle of manually refreshing the page. The decision was made to only implement this feature in the main screen as on other pages the user could get cut-off while carrying out a certain action; for example, typing a message.

The use of databases ensures all data is securely backed away and can be accessed and retrieved at any given time. This played an essential role in providing the data for the message, file and profile pages.

# Top-Level View

The model below is a simple representation of how the system behaves. The python code stores data when necessary in the database and can extract from it when required as well. Communication between the user and the Login server happens frequently to ensure the user stays logged in. The python code on its own can display very little, which is why HTML and CSS is used to help present all the information through a web browser to the viewer. When it comes to sharing files and messages, users can straight interact with one another.

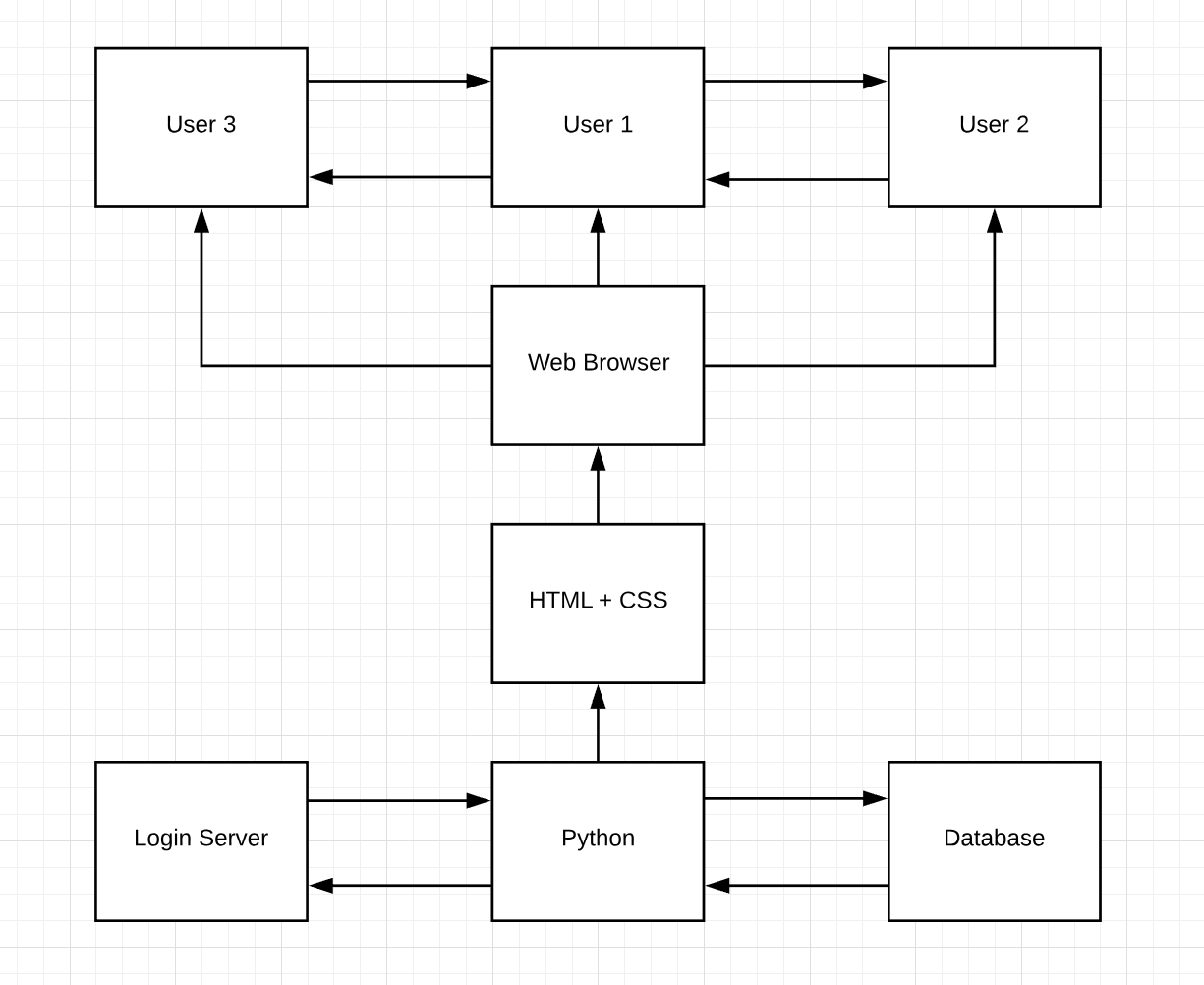


Figure 1:Top-Level Diagram

# Challenges Faced

Several issues needed to be overcome to reach the final product. These ranged from small syntax errors to an inability in retrieving certain users from the database. The hardest challenges included inputting users in to a database and figuring out how to integrate variables from python into html.

Databases were something that was relatively new. Understanding the syntax exactly was quite tricky at first and figuring out how to store the details I wanted into the right column was the biggest dilemma. However, once a basic understanding was attained, it was quite easy to manipulate the database to do exactly what was required. Good database usage was key to keep track of received and sent content.

The other major problem was being able to display lists and strings in python in html. The easy solution was to create the html in the same page as the python code, but this was bad coding practice and I decided it was best to move it the code into separate files. To overcome this issue, jinja was needed. Jinja is a template engine which allows variables to be passed from python into HTML. The alternative was JavaScript, but I personally felt jinja was far easier to use and thus a much better choice to go with.

# Peer-To-Peer Overview

Peer-to-Peer methods involve two computers being able to share data with one another. The three main types of networks include pure peer-to-peer, login server and central server with each one having its own perks and downsides. A pure peer-to-peer server was what was envisioned as it required no server to run and offered the most efficient communication between members, but in the end a vote was made to pick the server. The forefront reason we wanted to use peer-to-peer methods was to ensure data could be securely sent between users. This was to make certain we complied with the requirements that were set out by the client.

# Protocol Overview

The protocol went through careful consideration to achieve a balance between functionality and ease. It manages to keep things relatively simple but still allows for all necessary content to be sent and delivered. In the end we collectively decided the best option was to settle on the login server. The login server maintains a high level of security whilst requiring little bandwidth and providing ease of access. It works as a hybrid server, allowing users to login through the login server but then message by straight peer-to-peer means. The main downsides included the costs of the server and the need for users to be online when sending messages, however these in the end were outweighed by the benefits

# Tools

The main software used was python, an interpreted high-level programming language. Python was a good choice as it allowed for readable code that was easy to create (requiring less lines) in comparison to other languages. It also had Cherrypy integrated within, which was a key framework we needed for this project. Python is an interpreter meaning there was no need to compile. This was useful as continuous testing could take place without much delay.

Other software used included HTML, CSS, jinja2 and a little JavaScript. HTML and CSS were essential to make the user interface simple to navigate and access. Jinja2 helped allow content to be easily displayed where required whilst JavaScript helped provide timed background refreshes.

For saving and backing up code we used bitbucket. This was an essential application as it allowed us to always have a backup stored in case files get lost or deleted. As you can commit and pull code from any device, it enabled me to switch between operating systems, as well as between the university Desktop and my personal laptop.

# Future Improvements

Despite the extension provided, a lack of time was the main culprit behind not completing some of the more demanding tasks. If time wasn’t an issue, there were several features that would likely have been implemented to drastically improve the system. The biggest ones would be 2-factor authentication, offline messaging and encryption.

With security as one of the main factors to consider in this project, 2-factor authentication was a very beneficial feature to implement. Passwords can easily be discovered by outside users; however, 2-factor prevents this from being an issue by sending a code to your phone which needs to be entered during login. As the code is different each time, there is no danger of anyone ever discovering it.

Offline messaging would be another massive edition in improving functionality. The chance of two users being online at the same time is low as each employee will likely have their own meeting and presentations to attend throughout the day. With offline messaging, users can send messages at any time without the worry of the receiver not seeing it.

Encryption was the feature I felt most disappointed to not have in the final implementation. Encryption would ensure that if someone did hack into our database they wouldn’t be able to intercept and read messages and files. With security as a top priority, this should have been one of the features that I made an absolute necessity to get done.

As the aim of the project was to provide a functional back-end, the amount of effort put into the front-end wasn’t nearly as much. This resulted in the server lacking aesthetically. With more time and effort, a much cleaner and functional user interface could be provided. Some of the main improvements would include a chat box for each individual user oppose to having all the messages and files being displayed on a single page. Another would be having the option to get too any page from wherever in the UI. Currently, only the main page can access all of the other pages.