Python Report

# Abstract

The aim of this project was to build a peer-peer network to increase the security of the company by ensuring rival companies couldn’t access and steal private data. The main programming language python java 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 python. However, after researching and trying different implementations we were eventually able to make a working interface.

# Meeting Requirements

I believe the system has checked all the requirements to be suitable for use in the company. 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 username and password. All online users are displayed on the main menu snsuring 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 upto 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 at their pleasing.

Almost all data is backed up to databases ensuring that nothing is accidentally deleted and all records can be tracked back to.

# Notable Features

The highlight features include a simple GUI. It was designed with simplicity in mind to ensure users can see the information they need to see and can get quick access to any tools at their disposal. All of the main features can be accessed through the main menu. The ability to send emojis meant users can occasionaly send informal messages to fellow peers and the auto refreshing page means users are always updated on who is online at any given time. The use of databases ensures al data is securely backed away and can be accessed at any given time.

# Top-Level View

# Challenges Faced

Several issues needed to be overcome to reach the final product. These ranged from small syntax errors to an inabolty to retrieve certain users from the database. The hardest challenges included putting users intoa database and figuring out how to integrate variables from python into html.

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

The other major problolem 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 into seperate files. To overcome this issue, jinj was needed. Jinja is a template engine which allows variables to to be passed from python into html. The alternative was javascript, but I personally felt jinja was far easier 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 it’s own perks and downsides. A pure peer-to-peer server was what was envisioned but a vote was made to pick the server in the end. The main reason we wanted to use peer-to-peer methods was to ensure data could be sent between users securely. This ensured we kept with the requiremnets that were set out.

# 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 bandwith and providing ease of access. The main downsides included the costs of the server and the need for users to be online when sending messages, but these in the ned were outweighed by the benefits

# Tools

The main software used was python, a interpreted high-level programming language. Python was a good choice as it allowed for readable code that was easier to code (requiring less lines). Python is an interpreter meaning there was no need to compile. This was useful as continuous testing could be done 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 easily displayed where required.

# Future Improvements

Despite the extension provided, a lack of time was the main culprit behind not completing some of the more complex tasks. If time wasn’t an issue, some of the more impressive features such as multiple sessions and two factor authentications may have been implemented. As the 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 an aesthetic look. With more time and effort a much cleaner and functional user interface could be provided.