COMPSYS302 – PYTHON PROJECT

# INTRODUCTION

Talkie Chat is an executive chatting system that allows authenticated users to chat with each other through a client over a secure network. In addition to the users being able to send messages to other clients, they can also send files of other types, such as audio, video and images. Talkie also provides the users with the ability to schedule events with other available users. Communication to be network is aided by CherryPy, a simple web framework. Python 2.7 has been used to implement the back-end framework and establish communication with the server, other clients as well as the local database. The database is a SQLite3 database that can be connected to, created, updated and queried for data via dedicated Python libraries. For front end development, JavaScript (jQuery), HTML and CSS have been used to maintain, update and create the web pages.

# CLIENT SPECIFICATION

The client has specified that a level of security is required for the system ensuring the integrity of sensitive information. The underlying expectation of the client is for a system to be create that:

1. Allows the user to log into a system that automatically interacts with the server.
2. Finds other clients the user can interact with securely.
3. Allows for the creation and maintenance of a simple profile page.
4. Enables the user to send messages, audio, video, images and PDFs to other online users

Talkie Chat has covered the client’s requirements in a working system. Upon logging in successfully to the client, the user will be able to view their profile and navigate using the side bar to the area of the program they intend to use. These features include updating their profile page, viewing other profile pages as well as creating and managing the events on their agenda. Communication of text messages, audio, video and images have been tested and verified as working with multiple other systems on the same network. The user can type text or select a file to send to others. JavaScript has been used to actively fetch new network data to display currently active users.

# SYSTEM OVERVIEW

Overall, the system incorporates four components, with the Python section being the key component in the development of the system. Firstly, the Python software enables the client to interact with the database as well as the front end of the system. The front end has a simple design comprising of two parts; the HTML and CSS as well as the JavaScript. The HTML and CSS allows for the user to interact with the system using the dedicated website. JavaScript facilitates the animation of the website as well its communication with the Python software.

## FILE STRUCTURE

Within in the Python system, the code had been divided into four files. One such file, *databaseFunctions*, is dedicated to the communication between the Python software and the databases. By blending SQLite3 as well as the Python SQLite library, the Python software can connect to, execute, fetch and commit updates and inserts into the tables within the database.

To communicate with the web service, in particular the log in server, the Python URLLIB and URLLIB2 libraries were used to request other servers for information. These requests are kept separately in the *externalComm* file. This file contains all requests to other addresses. Once other functions have retrieved and formatted the information to be sent properly, an *externalComm* function will be called to have the packet sent to the IP and port last associated with the receiver.

Internal interactions and processes may also occur in the system. These actions are called in the *internalComm* file. The purpose of this file to abstract away several processes from the main server file in order to maintain good readable code. Currently the *internalComm* file contains the least amount of functions, however further developments which will be discussed further in the report will expand on the purposes of this dedicated file.

The main python file *server\_py* contains all the functions which are exposed for the JavaScript, HTML and other clients to call. As this require the tag *@cherrypy.expose* they are required to be in the same file as the CherryPy engine block where the sessions are stored and engine is started.

Any HTML or JavaScript file has been placed into the *public* folder. In total, there are three HTML files used to display the entire website. One of these is used to display the sign-in form; *index.html*. The other two, *messages.html* and *profile.html*, are used to display the message data and the profile and event data, respectively. To change and refresh the contents of the divs in the HTML, *main.js* and *profile.js* are used. File uploading occurs using the simpleUpload.js library which contains a set of functions which allows for files to be more easily transferred from a network to be stored in the local directory. Within the public folder is also the static and the downloads folders which are used to store the *main.css* file as well as the downloaded images, files, audio and video sent to and from other clients.

## USER INTERACTION

When the user first starts of the server using the command line and opens the main page, the user will be redirected to the log in page. The page prompts the user to input their credentials as well as their location. This is then authenticated with the external login server. Once authenticated, the client will start a Python thread which will continually call the login server APIs that report information about the user as well as get a list of users who are currently online. The contents of the response will be used to update the tables containing user data.

After successfully logging in the user will be shown their profile. The side navigation bar will allow them to navigate to the other features of the system. By clicking on messages, the user will able to see all the users they are currently able to connect with. As this is for executive within a company to communicate with each other, it has been designed for optimum security whereby, two users must be online together to communicate. Therefore, messages between two people can only be seen when both parties are online.

# SIGNIFICANT ISSUES

Several issues surfaced during the production of the system, including integration between languages and communicating between clients. Contributing to these issues was the underlying issue of learning four new languages while simultaneously parsing data between them so they can operate their desired function.

## INTEGRATION

Each programming language used had some desired function that it needed to fulfil for the system to function. To get information from the database to the web interface to be displayed, the data was required to be extracted from the database, formatted in python, parsed through to JavaScript where it was formatted for HTML. Several additional functions were required to gather this information. To overcome this issue, several functions needed to be written to introduce modularity into the system and avoid repetition of code. Supplementing this, additional testing features such as try excepts and print statements were placed at vulnerable points to deduce the nature of the issue. Print statements aided in understanding the type of data being transferred and allowed for code to be created that either bypassed or changed or follow through the following processes.

CSS proved more difficult to debug due to the nature of the language. This meant that the console was used make quick changes or fixes to the CSS which were later transferred to the actual file. The same testing method was used to debug the JavaScript.

## CONNECTIVITY

Having worked on the remote Linux environment for majority of the project, there was a lot of difficulty with connecting with other clients. This was simply because of the way the internet was configured within the development setting. In addition to this, sending the information packets correctly was also something that needed to be managed throughout the system. This was making sure that all the parameters within the JSON object had all the required keys which were spelt correctly to the standard of the report. To reduce the amount of confusion in the system, conversion to a JSON object was done before calling the transmission function associated to that data. Catching the exception raised by the program also allowed for a detailed assessment of the error as well as where it is occurring. Although ideally, using return statements from the other clients would generally suffice, this heavily relies on the other client having implemented their code correctly.

# FEATURES OF SYSTEM

Building upon the previously discussed the system contained several additional features which allows the user to more actively connect with other clients. These include features within the code development as well as features in the user interface.

## MULTIPLE SESSION SUPPORT

Talkie Chat can support multiple users logged into the same client on different computers. This means that in a company setting, Talkie Chat can support all the users in the company without the use of different clients. This system uses both sessions data as well as data stored within the database ensure all users are automatically logged off if the server going down.

## MODULARITY OF SYSTEM

To reduce the number of lines used within the files, code reused was emphasised during development. This meant that functions were designed to perform a function well and return a stable and recognised result upon its call. Although this needs improvement the basis of this has been established and allows continuation in development.

## JAVASCRIPT USE

JavaScript was used to automatically refresh the message contents as well as the list of active users in which the user can message. It was also used to refresh the profile and the forms displayed depending of the button clicked. Communication between the JavaScript and HTML was important for this system to occur. This reduced the number of HTML files required for the user interface.

## EVENT CO-ORDINATION SYSTEM

The event co-ordination system allows for the user to create, store and respond to event invitation for within the application. A form similar to the form for editing profiles is displayed that allows the user to add events is displayed for the user to determine the appropriate fields for sending. Upon form submission, JavaScript is used to call functions within the Python application to store the event data in the appropriate table and send that event request to the other person via their own APIs. In the user interface, two categories are displayed; *Upcoming Events* and *My Events*, which detailed events which have the user as the guest and then the user as the host respectively. Under the *Upcoming Events* header, the user is then able to respond to the host their status; going, maybe and not going. This is then returned to the host to acknowledge the user’s attendance to their event.

# SUITABILITY FOR APPLICATION – PEER-TO-PEER NETWORK

The peer-to-peer network is suitable for this application due to its secure nature. As the client intends to use the messaging application to transmit sensitive data it is important the application be both robust in its ability to find and acknowledge peers. Peer-to-peer applications send data directly from one IP to another resulting in a more secure method of transmitting and receiving from clients. Having the login server to aid finding peers also helps to initialise the system. It is suitable for our application because the network required is within the company. This means that clients would specifically know who each other are, therefore a login server seems appropriate for solving the client zero issue that arises in a peer to peer application.

# PROTOCOL DISCUSSION

## SUITABILITY

The protocol provides guidelines for how to implement the API calls to the login server as well as to other client nodes. Since many developers will be making the same type of client and all the clients will need to be able to communicate with each other, a protocol is necessary to ensure that all communication is managed and controlled. It also ensure that a standardised method is used through the systems.

## DEVELOPMENT

While developing the protocol it for difficult to understand how the system was supposed to function. Seeing as all the terminology was new and unexplored, a lot of trust was put into place when others who knew more were voicing their opinions. This led to a lot of discussion once functions were actually written, where small changes were made to add more sense or logic. Elements of the network data transfer process have been diluted and therefore it took quite a while to determine the way the data would be transferred and how it would be received. This was attributed by the fact that the networking problem had been abstracted away by the CherryPy engine and therefore several layers of the internet protocol have been removed to ease the transfer of data.

CherryPy is a simple web server framework that responds to HTTP requests and abstracts away the difficult elements in internet communication. This gave a portal for the website elements; HTML and JavaScript to be able to communicate with the Python server. Due to this, major communication difficulties have been abstracted away. HTML and CSS have been used to develop the front end of the website. These used in conjunction with the JavaScript enables the website to be responsive to the changes in the Python code. Python libraries have also enabled communication with the databases where all the information is stored. Overall the tools used were able to work well with each other once the method of communication was established.

# FUTURE IMPROVEMENTS

Several aspects of the clients brief have been inherently met through the project, however to further satisfy the client’s needs, elements of data security, network security will need to be addressed. As this system is a prototype system that is modular, encryption functions may be easily implemented alongside the current system. These will be added to the *internalComm* file where all modification to the data before sending will occur. These will include different levels of security and the ability to communicate with substandard clients. Eventually the system will be implemented so that the databases will store encrypted messages and data to hide sensitive information from attackers. Two factor authentication can also added to ensure the system is secure from attackers targeting front end entrance.

The ability to communicate offline to other clients will also be a feature to be implemented. That will ensure that all messages will be able to be sent and received by all clients. Although this is trusting that the other clients will have the ability to also receive offline messages. The ability to also receive reports from other clients while the network is offline will also be a feature to be implemented that will allow for the user to try and communicate with the last known IP and port of all users. This feature will enable the clients to be able enter the network of existing users while the login server is down.

Front end development will also need to be polished to ensure the user is able to easily use the website in a coherent way. Notification and other informative tools should also added to ensure the system is more responsive to received information. Tool for markdown and other formatting systems for the text editor can also be added to allow more user customisation for the messages.

Introducing templating engines such as Jinja will also remove code from the Python and JavaScript elements to ensure a mode modulated and standard system.

# APPENDICES

## APPENDIX A – SYSTEM DIAGRAM

