

## Contents

### Analysis

<b>Problem Identification</b>	<b>5</b>
Stakeholders	5
Why is it suited to a computational solution	6
Computational methods that the problem lends itself to	7
<b>Interview/Survey</b>	<b>9</b>
Questions and response analysis	9
Teacher Interview	13
<b>Research</b>	<b>14</b>
Existing similar solutions	14
Features of the proposed solution	16
Stakeholders opinions regarding the initial concept	22
<b>Requirements</b>	<b>23</b>
Software and hardware requirements	23
Stakeholder requirements	24
<b>Success criteria</b>	<b>26</b>

### Design

<b>User interface design</b>	<b>28</b>
Symbols	28
Primary user interface	30
Settings user interface	31
Log-in user interface	32
Stakeholder input regarding the proposed interface	33
<b>Algorithms</b>	<b>34</b>
Client-side	35
Server-side	37
<b>Algorithms in more detail</b>	<b>38</b>
Client-side	38
Server-side	41
How the modules will link	42
<b>Sampling Investigation</b>	<b>43</b>
Lossless Compression techniques	43
Lossy compression techniques	44
Dictionary encoding	44
<b>Script Communication Codes</b>	<b>45</b>
Codes table	45
<b>Database design</b>	<b>46</b>
UML diagram	46

## Computer Science NEA project - Chat application

<b>Pseudocode functions and modules</b>	<b>47</b>
Logging-In	47
Connect	47
Update Application	47
Run Length Encoding (RLE)	48
Dictionary Encoding	48
Lossy Compression	49
Compression Function	49
Asymmetric Encryption/Decryption	50
Sending messages	51
Hashing algorithm	51
Receive message	52
Dictionary Decoding	52
Run Length Decoding	53
Decompress	53
Report	53
<b>Server Side</b>	<b>54</b>
Create Client/client thread/create connection Class	54
Connect	54
Send_message	55
Compress	55
Receive message	55
Hashing algoirthm	55
<b>Testing the algoirthms</b>	<b>56</b>
Explanation and justification of this process	56
<b>Inputs and outputs</b>	<b>57</b>
Input output table	58
<b>Key Variables</b>	<b>58</b>
<b>Validation</b>	<b>59</b>
File explorer	59
Buttons	59
Text Boxes	59
Drop down menus	59
<b>Testing checklist</b>	<b>60</b>
High Level checklist	60
<b>Iterative unit testing</b>	<b>61</b>
Trace table template	62
<b>Performance testing</b>	<b>62</b>

# Computer Science NEA project - Chat application

## Development and testing

<b>Server-Side</b> .....	<b>63</b>
Database.....	63
Functions.....	66
Encryption.....	66
Hashing.....	67
Send_message.....	67
Decompression.....	65
Dictionary decode.....	65
Receive_message.....	66
Compression.....	70
Declaration of variables and the creation of important connections.....	71
Instantiating client connections.....	72
Authentication.....	73
ThreadC class.....	74
Methods.....	75
Run method.....	75
Update.....	76
Update_message.....	74
Forward_message.....	77
Report.....	77
<b>Server-side script</b> .....	<b>78</b>
<b>Client-Side</b> .....	<b>79</b>
Functions.....	79
Encryption.....	80
Hashing.....	80
Receive_message.....	81
Send_message.....	82
Decompression.....	83
Run Length Decoding.....	84
Dictionary Decoding.....	84
Dictionary Encoding.....	85
Compression.....	86
Run Length Encode.....	86
Lossy Compression.....	87
Login.....	87
Connect.....	87
Script.....	88
Constants.....	88
Key Variables.....	88
Graphical User Interface .....	89
App Creation and Log-In window .....	89
Log-In Button.....	90
Address selection.....	92
Send Message Button.....	93

## Computer Science NEA project - Chat application

File selection.....	94
Report button.....	95
Settings button.....	96
<b>Client-side script.....</b>	<b>98</b>
<b>Script Testing.....</b>	<b>98</b>
<b>Project Summary .....</b>	<b>109</b>

# Analysis

## Problem Identification

Chat applications are used daily around the world, as of late February 2022 alone in the UK 57.6 million people regularly used social media showing a profound need for a chat application. There are already many chat applications however these are not useful for students and lack privacy, especially with new legislation that has been enacted.

At the moment there is no way to effortlessly send large files and data without it being blocked or compressed greatly. Membership is usually available for purchase that offers benefits such as no file size limit however these files are still greatly compressed and the file size is only raised slightly to send large files.

Currently, if people wish to transfer large files without much compression a different application is required such as DropBox. There is no application that allows users to socialise and send large files for free with little compression and effort.

Students typically lack money unable to afford expensive memberships required to transfer their school work, a necessity within education. Many lessons are now held remotely as technology becomes more prevalent within the classroom.

Files can be printed and exchanged physically, however, this is time-consuming, expensive, bad for the environment, and requires printer ink which can be an expensive commodity. The problem is amenable to a computational approach as a computer and network would be required to solve this problem. The solution will require data to be sent across a network to another computer that requires networking and computer hardware.

## Stakeholders

The main stakeholders for this project are students as the issues discussed plague students, however, some teachers struggle with these problems too. Homework assigned online is typically a large file that can not experience any lossy compression so students can read the assignment. This software will be open to a wide range of users as many users send large files such as office workers.

Students send data frequently which opens up many social media platforms to online bullying. Online bullying can affect teachers as the problem can not be easily resolved disrupting students. I will be implementing a report feature into the application that will allow students to flag messages.

Students of all areas of education require document transfer frequently so this application will be used by a large demographic of students.

The application will be free, but due to student's concerns regarding online privacy and security, asymmetric encryption will be used.

Jude Easton is a college student who studies graphic design requiring him to transfer large images and virtual models for his project. These mustn't be lossy compressed. Other students are in a similar position such as Harry Taylor who studies architecture requiring him to transfer large files such as floor plans.

**Jude Easton** - "I study graphic design as part of my course I am required to create multiple designs, these designs must be sent to my teacher. Currently when I send my designs I have to use multiple applications to message the teacher and send my designs without lossy compression. If my designs

## Computer Science NEA project - Chat application

are compressed with lossy compression they will be ruined therefore I require an application that allows me to choose the compression method to be used."

**Harry Taylor** - "As one of my subjects I study Architecture, this requires me to send floor plans and work with other students. I am required to send over designs and converse with them currently to do this I have to use multiple applications as I can not risk my floorplans and designs being compressed via lossy compression. If my work is compressed with lossy compression I will lose essential detail confusing team members and lowering my grade!"

**Group of students** - I have chosen a sample of year 12 students studying differing subjects to represent the typical students who would use this application. The general consensus was that this application would be necessary as they require more freedom in choosing compression techniques for certain subjects. Those who did not need different compression techniques wanted a more secure application than those currently available.

### Why is it suited to a computational solution?

The solution is suited to a computational approach as to solve the problem data will be required to be sent on a network from a computer. The software will run on a computer and will not be able to be used without a computer. The program will take the message, break it into packets and send them to the correct IP and MAC addresses; this can not be done without a computer.

### Computational methods that the problem lends itself to

#### Problem recognition:

The problem requires me to send data safely to other users, however, the underlying problem is sending encrypted and compressed data. I will have to decide what compression technique to use, I will break down how each compression algorithm works and break down the different encryption methods. Different file formats will need to be accounted for and the compression method that correlates best. To solve this I will allow the user to choose their compression technique if two methods can be used for that particular file type.

I will need to break down the program's GUI as each part of the GUI will have a specific purpose requiring a thorough design to efficiently produce an easy-to-use Graphical User Interface that offers all functionality required to solve the problem and meet my client's requirements. The program will need to store data and process data by applying different functions and algorithms to data.

## Computer Science NEA project - Chat application

### Problem decomposition

Using abstraction I have broken down the problem into a series of steps. This is still a very simplistic approach to the solution.

1. Use the GUI to take inputs, address book,(MAC, IP addresses)data being sent, and compression technique
2. Create a stable connection
3. Apply the compression technique being used
4. Apply the encryption method being used
5. Send the data
6. Display the "Received" confirmation message
7. Check for messages received if so display them(loop frequently throughout processes)

Once these tasks are completed the message will have been sent and the problem solved. I have decided to use a GUI to gather inputs as this is more intuitive e.g the address book.

The system has all the inputs required quickly due to the GUI allowing the CPU to process other instructions rather than wait for each input. This improves the time efficiency of the program, improving its Big O notation and allowing the system to be run on a computer with limited resources.

## Computer Science NEA project - Chat application

### Divide and conquer

By employing abstraction and the divide and conquer computational thinking methods I can easily break the problem down into intuitive modules. I will split the program into a series of steps so that the program can be designed modularly making development much easier as there are clear deliverables. I will be breaking my problem down into small parts(modules) then programming these separately this makes the application easier to understand, program and design. An example of divide and conquer within my application is the breaking up of the user interface into separate functions.

### Thinking ahead

I will employ this computational thinking method to ensure that my code is efficient. This is necessary as data will need to be sent quickly and accurately. To employ this I will be thoroughly planning my project to ensure the most efficient high-quality solution is achieved. I can think ahead for each module planning it thoroughly making the problem easy to understand ensuring all user requirements are fulfilled. I will need to think ahead when choosing what data to pass into the different functions and algorithms

### Abstraction

Abstraction is the act of removing unnecessary data allowing me to make use of the divide and conquer approach making the system easy to develop and of high quality. The Problem is very large therefore removing unnecessary data such as the user interface initially, will allow me to break the problem down into manageable simple parts. The compression methods are an example of this, compressing data is a complex problem however unnecessary data can be abstracted such as the hundreds of file types that do not need support in this application. Removing unnecessary data types makes the compression methods easier to program, understand and design.

### Logical thinking

This is the method of solving a problem by understanding the problem thoroughly through research. The appropriate solution is chosen through this research, addressing facts and logic. This method is suited to this problem because I will need to compress data based on the data type and compression methods specified. Logical thinking will enable me to solve the problem of choosing compression techniques based on data type as I will have to research and understand the problem thoroughly to choose the most appropriate compression technique for that data type.



## **Computer Science NEA project - Chat application**

### **Interview/Survey**

To gain insight into the current problems posed by social media I have conducted a survey with a sample of twelve students.

This survey was conducted on college students of varying ages studying a wide range of subjects. I have identified what problems students are facing and on what platforms these issues reside. I can now research these platforms identifying the areas of improvement.

I have also interviewed a teacher asking them similar questions to identify any concerns surrounding networking that they had.

### **Questions and response analysis**

The questions that were given out as part of the survey were as follows:

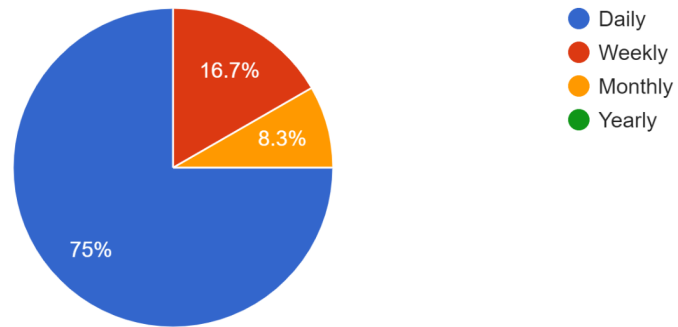
1. How often do you send data online?
2. How often do you send documents online?
3. Are you worried about privacy when sending data online?
4. Which platform do you use the most for messaging?
5. Do you often message other students?

## Computer Science NEA project - Chat application

**Questions one and two** established if a chat application was necessary and if file size limits were a problem.

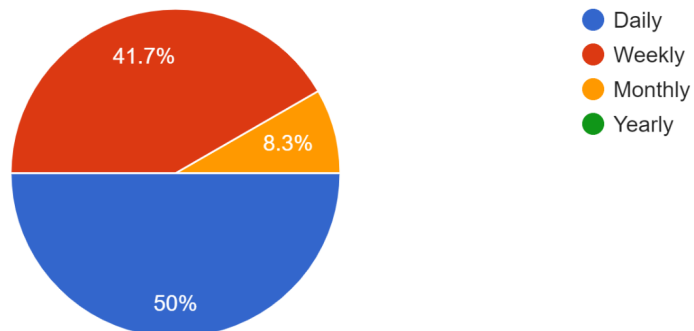
How often do you send data online?

12 responses



How often do you send documents online?

12 responses



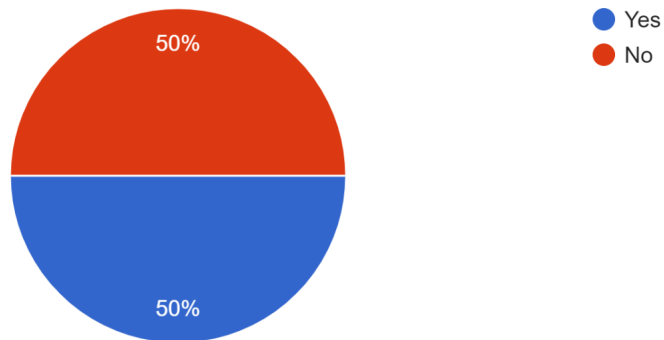
Clearly, from these results, a chat application is necessary as 50% of users send documents online showing many students require a chat application that offers users the ability to send and receive files/documents of all sizes.

## Computer Science NEA project - Chat application

**Question three** was written to attain information regarding how safe people felt online. This would help me decide if a stronger encryption method was necessary than the one in use on many applications today.

Are you worried about privacy when sending data online?

12 responses



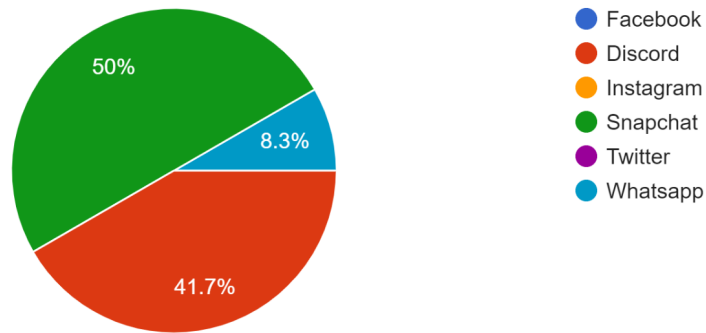
The responses to this question are 50/50 showing that a stronger encryption algorithm is necessary. These results may be slightly off as many of the students may not be aware of the current legislation and ethical problems regarding online security/privacy.

## Computer Science NEA project - Chat application

**Questions four and five** determined what platforms were in use the most and if students were making the most use of this chat application. These questions determine the demographic I am aiming this application towards and the user requirements surrounding this.

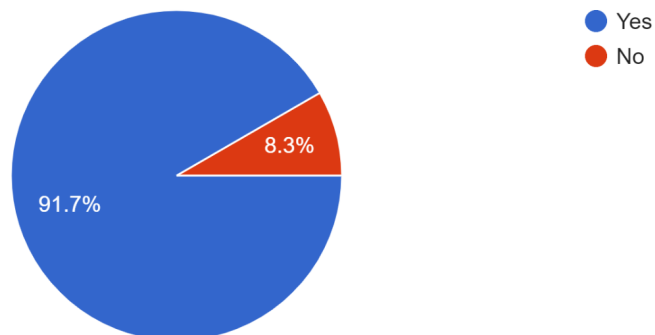
Which platform do you use the most for messaging?

12 responses



Do you often message other students?

12 responses



These results show that Discord and Snapchat are the most popular social media applications and the most popular for transferring files/documents. During my research, I will look into these applications and identify areas of improvement. I have also discovered from these results that 91.7% of users frequently message other students. This is incredibly high, showing that this application is necessary.

Overall the results show that there is a necessity for a chat application with limited compression, strong encryption, and unlimited file transfer size.

## Computer Science NEA project - Chat application

### Teacher Interview

To gain insight into the problems surrounding safeguarding and ethical concerns online, I posed some questions to a teacher. I approached Mrs Wood due to her extensive experience in teaching at all ages; this would help me to get the best insight into safeguarding and ethical problems surrounding online social media platforms. The questions I asked and responses given were as follows:

1. What are the main problems regarding social media platforms for students?

**“Online safety in regards to bullying and plagiarism, homework and school work is often shared amongst students”**

2. How can social media platforms help mitigate the problem?

**“Social media platforms could introduce an anonymous report feature to mitigate online bullying and improve online safety, a certificate of authenticity could be sent as metadata with the file to stop plagiarism”**

3. Are social media platforms useful for students?

**“Social media platforms can be useful for students however they can be just as harmful, a useful social media platform would be safe and secure”**

4. With the increasing prevalence of technology within classrooms and the emergence of remote classes, how else could documents/files be shared?

**“Documents must be transferred online as sharing documents with students by hand is time-consuming and expensive, sometimes even impossible as we have observed in the last few years, currently there are many problems associated with the current platforms used”**

**Questions one and two** were intended to identify students' problems from a teacher's perspective. The responses show that the primary issues are safeguarding and security concerns. I will be looking into implementing a report feature into the application as suggested by Mrs Wood to solve this problem.

**Questions three and four** were aimed at the feasibility of this application. These responses concluded that this application can be helpful for students and teachers if the correct precautions are implemented.

## Computer Science NEA project - Chat application

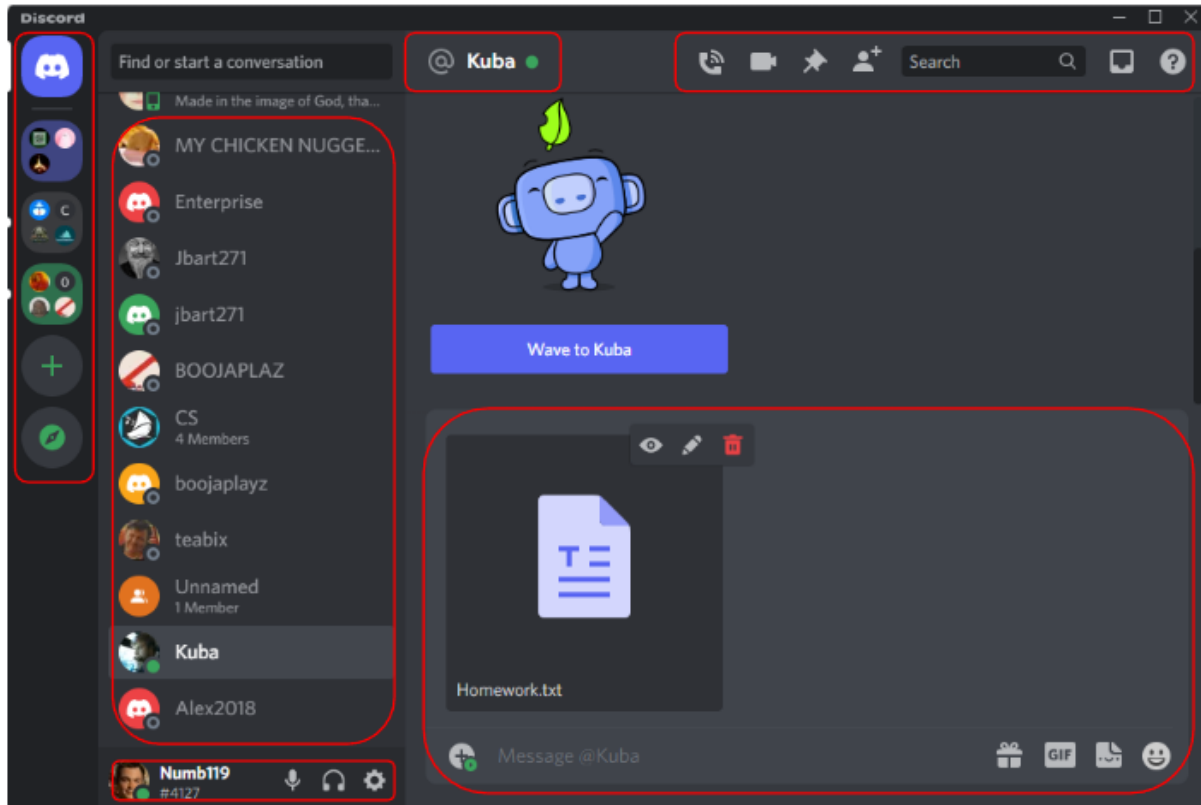
### Research

I am now going to research existing solutions identifying key aspects of these solutions. I will be looking at how the solutions work and the improvements that can be made to them. Looking at these solutions will ensure my application is successful as these current solutions are already incredibly successful and used by students.

### Existing similar solutions

Discord:

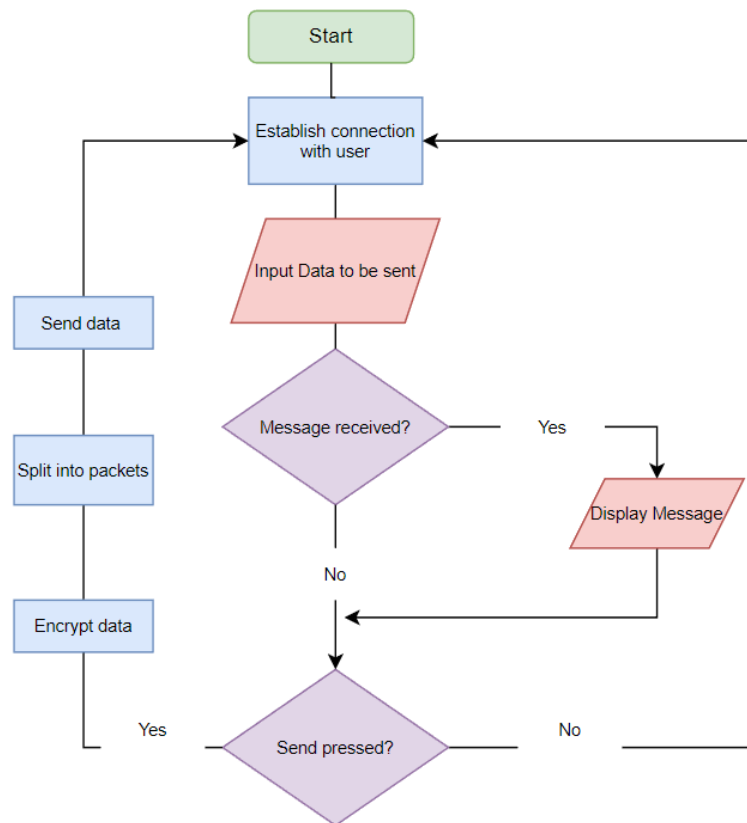
### Interface



I have identified key features of the discord application to analyse them and determine what can be improved and what can be implemented into my application. This analysis will allow me to develop a better solution that is in line with user requirements.

Currently Discord has **150 million users**, therefore, analysing this application will give me a very clear understanding of what to include within my application.

## Computer Science NEA project - Chat application



As identified by the red areas Discord has a very **intuitive** user-friendly graphical interface. This GUI makes it easy for anyone to use the application, this will be important to my application as it is designed to be able to be used by many students therefore a **neurodiverse** application will be a necessity.

The application follows other social media standards to achieve an intuitive user interface, the addresses are placed on the left making it easy to read, a user will identify the addresses before the other less important parts of the user interface. The UI is designed to be discovered in the order that is undertaken when sending a message, this is similar across many social media/messaging applications like Facebook.

The application's User interface makes use of **contrasting colours** to draw the user's attention to key areas of the application such as the address book and direct messaging section.

The user interface uses **cultural colours** to create a different experience compared with other social media platforms. The colour scheme used is different shades of blue-grey these colours symbolise a sense of separation and dreariness however also adds a level of professionalism and sophistication to the design.

This colour scheme has been chosen to show users that discord is a gaming platform and an escape from daily stresses it is different and separated from other applications that use opposite colour schemes such as Facebook, blue-gray also gives an impression of professionalism advertising Discord's functionality to be used as a professional environment and casual environment.

I will be identifying cultural colours that can be used in my application to improve the functionality of the application and appeal to a larger audience the application will have to be neuro and **culturally diverse** due to the wide range of students and teachers using the application.

## Computer Science NEA project - Chat application

### Functionality

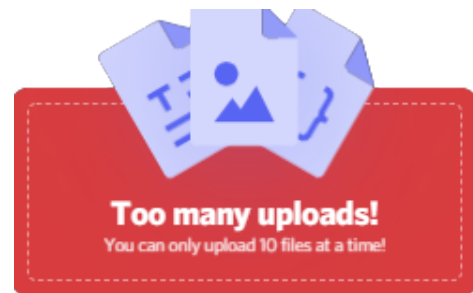
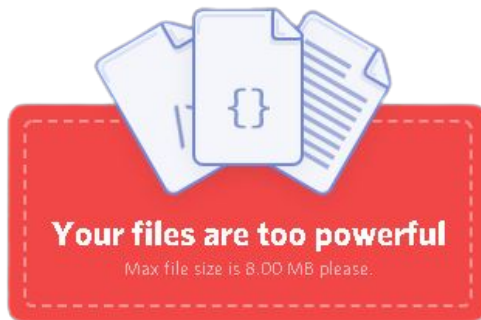
Discord is a very **functional** application it allows users to communicate through a wide range of innovative features, discord makes use of **server storage** allowing users to access their accounts and message data.

Some of Discord's features include:

- **Servers**(Servers are identified on the far left of the application they are spaces where many users can gather, they are a fully customisable environment allowing users to create the perfect space for their group)
- **Direct Messages**(As seen on the right of the screenshot discord provides a private direct messaging service this allows users to communicate in confidence sharing files and messages amongst each other, unfortunately there is a lack of support for bullying as there are no report features )
- **Video streaming/calls**(Discord allows users to privately call each other this allows users to communicate privately and allow the other user to view their screen through streaming)
- **Channels**(Channels are an essential part of Discord they allow users to communicate either by voice or text within servers these channels can be made private by adjusting user permissions making them a great space for communication amongst many members. Servers can cause a problem as users with malicious intent can easily spread content to many users)
- **Groups**(These are private areas where a small group of users can communicate it is used when users do not require a server but want to communicate with each other)
- **Developer tools**(These tools allow users to create addons/modifications to their discord application improving the functionality of the application by enabling in-depth customisation)
- **Server Bots**( Discord supports the open source community by allowing users to create bots these act as users however can behave based on code such as music playing improving discords functionality further however Discord bots can cause security risks due to the lack of management)



## Computer Science NEA project - Chat application



Discord is **very functional** it is an ideal application to communicate on, however Discord lacks key safety features such as a report feature and strong encryption for messages a safety concern identified from the survey conducted.

Discord lacks functionality regarding file sharing as there is a small space cap of 8.00MB which can lead to problems when trying to send data. High-definition images on average are much larger than **8.00MB** making file transfer an impossible task unless sending very small compressed files.

Files are **compressed automatically** using **lossy compression** techniques this is detrimental to some student's homework as many students require the transfer of high-resolution images.

Discord imposes a file limit of ten files at a time which requires users to **upload files individually**, a **time consuming** process.

Discord uses a significantly large amount of resources due to its **poor optimisation** deterring users with fewer resources to use the application as it can take up a large amount of processing time slowing their computers down significantly.

Below is an example of Discord taking a large amount of resources compared to other applications significantly slowing down the user's Computer;

File Options View

Processes						
Performance App history Startup Users Details Services						
Name	Status	76% CPU	79% Memory	4% Disk	0% Network	Power usage
> Discord (32 bit) (12)		61.3%	245.3 MB	0.2 MB/s	0.1 Mbps	Very high
System		4.7%	0 MB	1.0 MB/s	0 Mbps	Very low
> Among Us (32 bit)		3.0%	16.3 MB	0.1 MB/s	0 Mbps	Very low

## Computer Science NEA project - Chat application

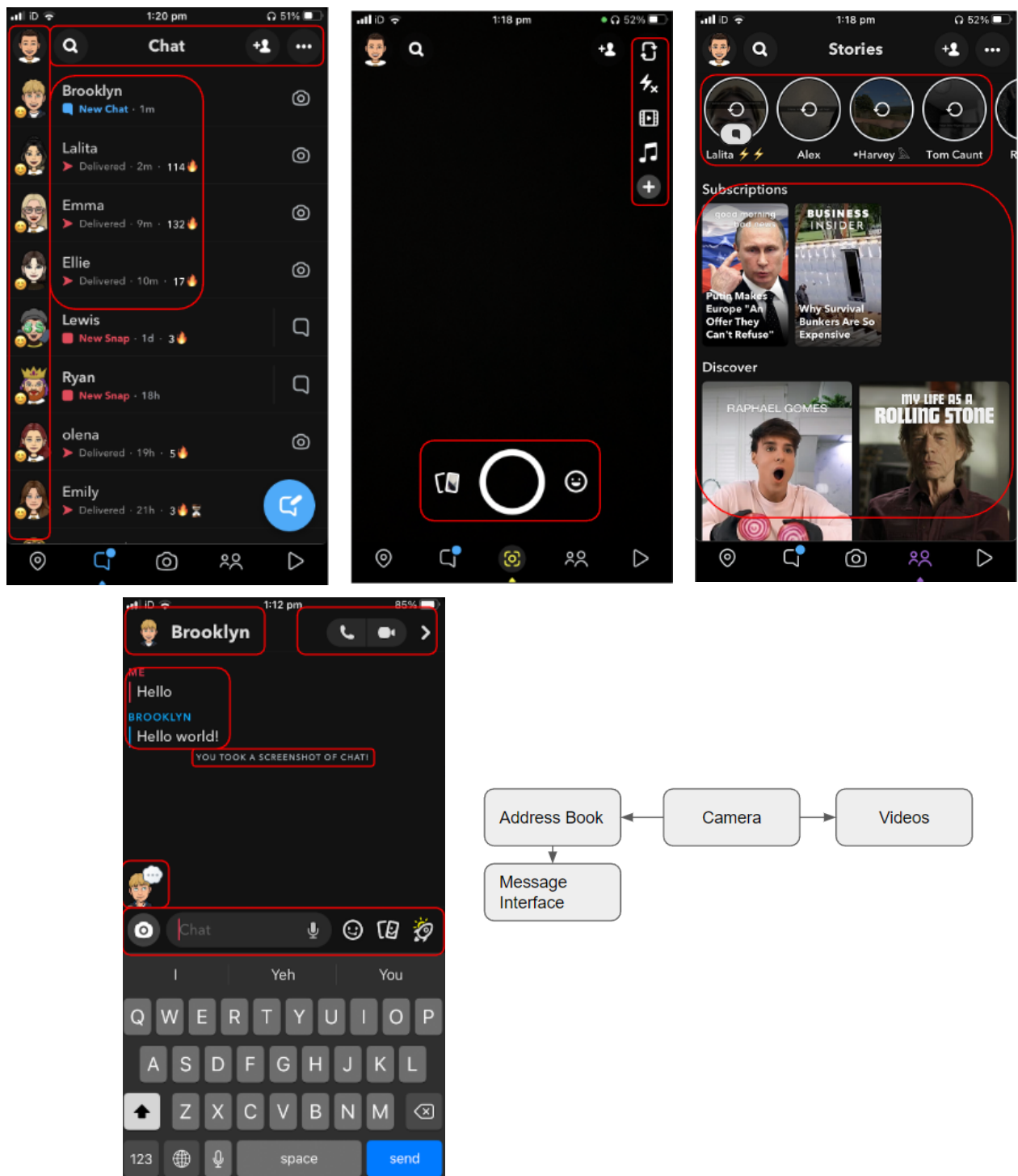
### What I have learned from this analysis

I will be implementing similar features to my application however adding essential changes that I have discussed above the features I am now going to implement are:

- Cultural Colours
- Left to right (important to least important) to make a neurodiverse intuitive interface that can be easily navigated requiring little effort by the user
- I am going to look at implementing a group feature to allow classes and small groups to communicate/share resources
- After learning how Discord handles reports I will be implementing an easily accessible report feature
- My application will be optimised to run on a large variety of systems I will achieve this by programming the application efficiently increasing the application's memory and time efficiency
- I am going to be looking at the possible implementation of accounts to facilitate remote access allowing users to access their messages and address book anywhere this will also allow messages to be received while others are offline
- I am going to allow large files to be sent using a compression technique chosen by the sender this gives users greater freedom and allows them to transfer files vital for school work without data loss

## Computer Science NEA project - Chat application

### Snapchat: Interface



I have outlined key areas of the Snapchat application. This application is a mobile application however the survey found this to be one of the top applications students use for communication, therefore I am going to be analysing this application identifying features that can be implemented into my application.

## Computer Science NEA project - Chat application

Snapchat makes use of many interfaces to reduce clutter and create an intuitive user interface. Users navigate the user interface by swiping to the different Graphical User Interfaces. The application contains many innovative features not seen in similar applications some of these are below;

- An **in-built camera** into the app allows easy navigation as the user does not need to leave the app.
- **Image editing**, filters using augmented reality, emojis, etc.
- The **stories** feature allows users and companies to publish videos and photos for users to watch, subscription feature includes allowing users to subscribe to certain stories.
- Users can share their **location** with friends.
- Notification on message received, message typing, and screenshots/saved images in chat.
- **Streaks**, displays how many days you have contacted a user without a twenty-four-hour silence.
- **Birthday reminder** if users have their birthday connected to their account users will be notified when it is their birthday.
- Great control over what you send, messages and photos can be unsent by users, by default Snapchat remove messages and photos after a twenty-four-hour period.
- **End-to-end encryption** for photos sent however messages are not encrypted.
- **Neurodiverse GUI**.

Snapchat is not ideal for everyday use as it **lacks functionality**. Snapchat can not be used for multiple purposes due to it only being available on mobile devices, the app only allows images to be shared however these are greatly compressed using **lossy compression** making this application not ideal for students to communicate on.

Snapchat **lacks security**; only images are encrypted, something that I will be fixing in my implementation as fifty percent of the survey participants were concerned about encryption. I will allow users to choose encryption methods, encrypting all messages sent.

Snapchat lacks privacy users locations and account details are shared with others in their address book by default. I will not be implementing this level of detail as many survey participants did not feel secure online, sharing user location will create a security risk.

Snapchat is **not very inclusive to all**, it uses block colours making the interface neurodiverse and culturally sensitive however some users may also struggle to navigate the interface due to the lack of information and buttons.

Users are expected to swipe, something that is **not universal** for other chat applications which could create confusion, I will be aiming to make my application inclusive to all users.

## Computer Science NEA project - Chat application

### What I have learned from this analysis

Snapchat lacks many essential features to fully solve this problem however I will be implementing some of Snapchat's features into my application due to its popularity. Snapchat features I will be looking at implementing are:

- **Allow unsending messages** however I will have to look at the feasibility of this as a database will be required to store accounts and messages
- **Notifications** for messages received and sent
- **End-to-end encryption** however for all messages not just files
- Control over how messages are compressed to avoid the lossy compression of files important for schoolwork such as artwork
- If implementing the **unsending of messages** I will look at implementing 24 hours for message saving to take the load of the server and allow users to load chats quicker
- Snapchat lacks a **report feature** so I will be implementing this allowing messages to be flagged and reported for review
- **Voice memos** these messages are an integral part of Snapchat I will be looking at the implementation of this

### Features of the proposed solution:

#### The initial concept of my solution considering this research:

My software will present the user with a **simple intuitive Graphical User Interface** upon launch, this interface will allow users to choose whom they chat with upon choosing this the application will output the user's past messages. Messages will be stored locally to take the load off the server however I can get around this by using a **decentralised network** allowing me to implement more features such as the unsending of messages. This will require many users so is not applicable to this solution however for an application designed to be used across schools at all levels of education this approach would be feasible due to the larger user base.

The user will be able to choose from **different compression techniques and encryption methods** giving the user greater freedom in how their data is transferred.

Users will be able to **select files for transfer regardless of size** and have access to a **report feature** notifying an admin/trusted personnel of the problem, the user's username and UID will be flagged and stored to be dealt with at a later date.

To facilitate many users, each user upon login will be able to choose a username which will be **assigned a unique identifier** by the server. Once the connection is made to the **static IP** server from the user's computer the server will check if messages have been received and relay those back to the application to be displayed in message history the message will then be deleted from the server. The server will also update the address book for every new user who has connected to the server.

## Computer Science NEA project - Chat application

### Limitations of my solution:

Users will not be able to un-send messages from one another due to the **lack of server support**, messages will be **stored locally** this could be a **security issue** so messages will be **encrypted within storage**. This application will not have enough users to make use of **decentralised server techniques** at first and server **resources will be limited**.

Users will not be able to be identified undermining the report feature this can, however, be mitigated by asking users to use their name; however this is still open to abuse. If a decentralised server technique or a server with more resources was being used I would implement a **larger database to store more data allowing easy and quick identification of users**.

### Stakeholders opinions regarding the initial concept:

**Jude Easton** - "This is a great solution and meets all of my user requirements, graphic design consistently requires me to send large files of varying types so the solution will have to be open to a wide range of file types as I can not risk losing data as this would affect my coursework grade"

**Harry Taylor** - "This is great for architecture and general communication I like how I can remotely discuss my work with other students and send them the work without changing the application, I am also happy about the security of the application as this is a big concern for me when transferring data online"

**Group of year twelve students** - The general consensus of the group was that the solution **met their requirements** however the application may be open to abuse so a report feature is a good implementation however this report feature **needs to be robust** allowing **reports to be made 24/7**, many of the group also revealed that they lack a system with a surplus of resources so the application must be optimised.

## Computer Science NEA project - Chat application

### Requirements

I will now be researching the requirements for the solution ensuring the application will be used as designed meeting all user requirements. This section will make sure the application is designed to the correct specification and usable by the target audience.

### Software and hardware requirements

#### Hardware

- **A system able to run the software**  
*The system the application is running on will need to meet the required specifications of the operating system in use as this typically leaves resources available for other applications. The application will be optimised requiring few resources, Windows required system specification is a dual-core 1 GHz processor and 4 Gb of RAM more than enough for this application.*
- **The correct peripherals**  
*The system will need to have a mouse, keyboard, and visual output device.*
- **Network connection**  
*A stable network connection will be required to use the application as intended, message history will be able to be viewed while offline however no messages will be able to be sent/received while lacking a network connection. A NIC(Network Interface Card) will be required and the facilities to connect to a network that is connected to the applications server.*

#### Software

- **Operating system**  
*The computer will need an operating system that supports python to run the program as intended, some of the Operating systems that support Python are; Windows, macOS, and Linux.*
- **Python Interpreter**  
*Python can not be compiled into machine code due to it being a dynamic language it can only be compiled into bytecode so an interpreter will be required to run the application.*
- **Python Libraries**  
*The user will need the correct libraries, these will be MySQL, GUIZERO, Sockett, ,Pillow and the random library however these libraries are built into python except GUIZERO and MySQL this was a conscious decision to reduce download size..*

## Computer Science NEA project - Chat application

### Stakeholder requirements

Through a thorough analysis of the survey conducted and meetings with stakeholders, I have determined a list of stakeholder requirements for the application's functionality and interface design. I have explained how I am going to implement these requirements below:

### Interface

- **Intuitive**

- *To create an intuitive interface, I will make sure that the user interface is familiar to users I will achieve this by creating similarities to other social media platforms and chat applications.*
- *The interface will be read from left to right naturally. Due to this I will implement the order that is undertaken to send a message left to right across the GUI with addresses being to the far left, clearly showing the user how to use the application and create a fluid interface speeding up the process.*
- *I will be implementing block colours into the interface to identify important information and features of the application these colours will be carefully chosen based on familiarity and their meaning within cultures.*
- *Symbols will be essential for quick and easy use as these can be interpreted quickly, symbols will have to be familiar to the user and easy to understand.*

- **Culturally appropriate**

- *Colours will have to be carefully chosen to send the correct message to the user, many users of different backgrounds will be making use of this application as it will be used within schools so a culturally appropriate colour scheme is necessary. Currently, the stakeholders want the application to represent familiarity, friendliness, and separation from other social media and chat applications to show the user that the application is not just a chat application but designed to help students by offering bespoke functionality something which similar applications lack.*
- *Symbols will have to be chosen carefully to be interpreted as intended across different cultures.*

- **Responsive**

- *To maintain consistency the application will have to be responsive adjusting to different screen sizes and hardware effortlessly without losing any quality. The application must not be more difficult or easier to use on different systems, due to the target market of my application many users will be using the application across several systems such as school computers.*

- **Offers features custom to students**

- *Users will be able to select the teacher's section of the User interface to quickly navigate the interface.*
- *Classes will be able to be made to manage and organise users in their address book.*
- *Pre-written messages will be able to be sent from a separate menu like an emoji menu. Users will be able to select messages commonly sent between students such as "Can anyone help with this homework?".*



## Computer Science NEA project - Chat application

### Functionality

- **Able to send messages and files without a cap**
  - *To send large files and messages the algorithm used to split the data into packets and send them will have to be very efficient. UDP could be used instead of TCP due to the transfer speed however data loss is a risk which my stakeholders can not have.*
  - *A compression technique will have to be chosen to transfer these files quickly.*
  - *Users will be able to select files from their system and send them just like a message*
- **Choose message compression and encryption technique**
  - *A list of lossless compression techniques will be displayed for the user to select one, these methods will have to be optimised and efficient as lossless compression techniques can take large amounts of processing time. Processing server-side will be better as the server can multitask allowing the user to send other messages while the file just sent is being compressed and sent server-side creating the illusion that there is no wait to send large files.*
  - *Encryption methods will have to be programmed these will have to be strong and efficient algorithms. To test the encryption methods they will undertake rigorous testing using a powerful system to determine the security of the method.*
- **An easy and quick method of contacting people**
  - *I will be creating an address book this will be retrieved from the server, it will be a list of all users that have a UID, username, and IP associated with their account only the username will be displayed to the user in the application. The client will connect to the server, once connected the server will check that the details transferred are correct such as checking the IP has not changed, the server will then be able to transfer that users inbox this makes sending messages easier as the recipient does not need to be online. Messages will only be stored temporarily on a database within the server due to the lack of server resources if a more powerful server with a large space was being used messages could be saved allowing users to access them remotely.*
- **See message history**
  - *Due to the limited server resources to achieve message history, I will be storing the message history locally rather than server-side, it will be updated upon connection to the server.*
- **Report Feature**
  - *Due to stakeholder concerns a report feature will be implemented to allow users to flag messages notifying a moderator, any teachers on the platform will be a moderator by default. Regarding the report's content, the user will be banned or suspended from the platform.*
  - *Users will have to enter their student ID to create an account this will deter users from sending harmful messages as the users will be able to be identified easily.*

## Computer Science NEA project - Chat application

### Success Criteria

This is the criteria I will be following to ensure that I covered all of my user requirements the criteria will help me to stay in line with these requirements and identify my deadlines, below I have listed the success criteria explaining how I will be evidencing these:

- **Connection with the server can be maintained**
  - *I will evidence this by taking a screenshot of the server traffic and comparing that to the application's current tasks and IP.*
  - **Reason** - *If a connection can not be maintained with the server the application will not be able to be used as intended, and the problem will not have been solved.*
- **Messages including files can be sent and received regardless of size**
  - *I will display this by displaying the server traffic upon transmission of the files and making a copy of the recipient's chat history before and after transmission. I will do this for many different file sizes.*
  - **Reason** - *A problem identified by the stakeholders from the survey was that they could not send large files as some were blocked due to their size, to solve this I will ensure the application allows the transfer of all file sizes.*
- **Users can send messages via usernames/accounts**
  - *I will take screenshots of the server traffic along with the sender and recipient message history.*
  - **Reason** - *It will be hard for users to remember users IPs therefore it is important to use usernames so the application is intuitive as it will be used in a school setting.*
- **Users can choose their compression methods**
  - *To evidence this, I will display the data received by the server to show that the message has been compressed after choosing various compression techniques.*
  - **Reason** - *I have identified from stakeholders that a problem many students face is the lack of choice surrounding compression methods as this effected the quality of their work.*
- **The application must have a responsive intuitive GUI**
  - *The user interface will be displayed with the user's opinions regarding the interface.*
  - **Reason** - *The application is designed to be used by schools therefore it is important the application is accessible to all.*
- **Users can see their message history**
  - *I will display the message history of the sender and recipient.*
  - **Reason** - *Users will struggle to use the application if they can not access previous messages such as homework(files) sent in the past.*
- **The address book should be easy to navigate**
  - *I will display the address book and get users' opinions on this.*
  - **Reason** - *The application will be used in schools by students of varying abilities therefore it is important that the application is accessible to all.*

## Computer Science NEA project - Chat application

- **The application is easily ran**

- *I will run the application using virtual machines with restricted resources and monitor how the application performs making notes of hardware usage.*

- **Reason** - Stakeholders and users will have systems of varying power therefore it is important that it can run on machines of limited resources, this also enables me to accurately list hardware requirements.

- **The user is notified when a message is sent/received**

- *I will display server traffic and message history.*

- **Reason** - A user will need to be notified of a message otherwise they may not respond to a message.

- **Users can send pre-written messages aimed at students**

- *I will screenshot the sender's and recipient's message history.*

- **Reason** - This will enable users to quickly respond to each other with ease.

- **The application will have a report feature**

- *The report feature will send a message to a moderator. I will be displaying this message showing that the report feature works as intended.*

- **Reason** - It is important that the application is safe as it will be used within school settings.

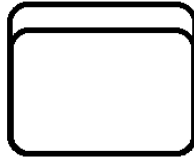
### Design

#### User interface design

I am now going to be looking in depth at the design of the application to ensure it is designed in a way that is efficient, easy to develop and meets all of my user requirements.

#### Symbols

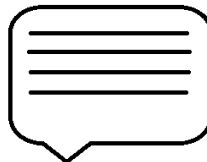
##### File upload:



This symbol represents a file it has been designed to look similar to the windows and other Operating systems file explorers such as Linux, for comparison the windows file icon looks like this:



##### Quick messages:

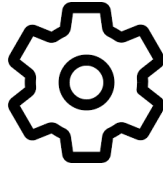


This symbol will identify the **pre-written messages** UI allowing students to access messages aimed at their needs. The symbol is designed to look similar to many social platforms and applications for example Microsoft team's and Facebook's messaging symbols:



## Computer Science NEA project - Chat application

### Settings:



This symbol will represent the **settings** user interface this symbol must be intuitive as it is important all users can access the settings user interface. I have designed this symbol to look similar to many other application's settings symbols such as Discord and the Windows Operating system these symbols are:

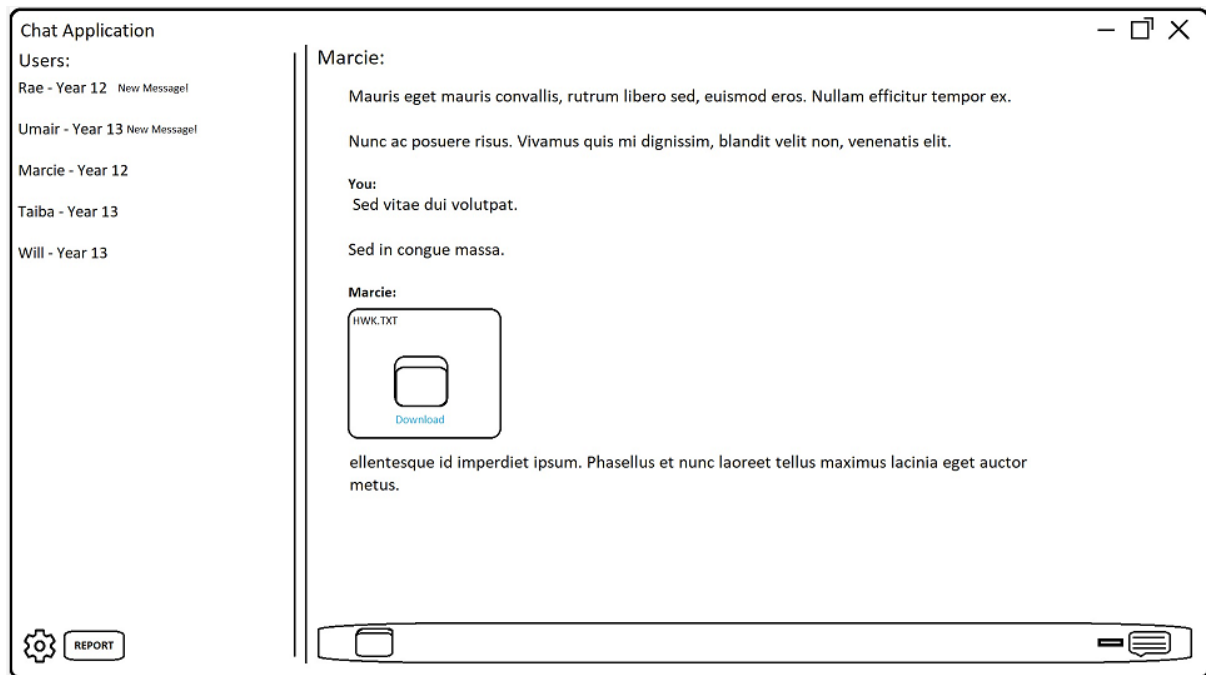


All of these symbols have carefully been chosen to avoid confusion creating an intuitive user interface across cultures. This has been achieved by creating similarities between these symbols and other widely used symbols such as the Windows operating system symbols.

Windows is a worldwide operating system. It is the dominant operating system with 1.4 billion users actively using Windows 10 worldwide every day therefore creating similarities to this operating system ensures users know what they represent.

## Computer Science NEA project - Chat application

### Primary user interface



I have created a very simple diagram of what I intend for the main user interface, this interface has been designed to be **simpilistic and familiar** by creating **similarities** between this interface and other similar solutions such as Discord.

The interface **reads left to right** creating a **fluid intuitive user interface** as users will understand the interface quickly and easily. The interface will link to other interfaces through the buttons provided these buttons have been specifcally chosen to avoid confusion ,the symbols are **universal** and have **similar meanings across cultures** they are used frequently in many other applications, for example, the file symbol is incredibly similar to the Windows operating system file explorer application.

I have not included colours however I am aiming to **accent** the user's address book on the left-hand side using a lighter colour and then using a **dark-grey** colour to create a **comfortable experience** for users sending messages. This will be the section that is used the most and users will not want to be distracted by a bright colour.

To the left of the interface the user can choose from a list of users to send messages to this will be stored on the server and updated when connected.

I have displayed **message history** to the right of the interface using **Lorem Ipsum** to simulate what the interface would look like after use. Users can write their message below the message history including files, or use a pre-written message.

## Computer Science NEA project - Chat application

### Settings user interface

Chat Application

**Settings:**

Compression Technique:

Lossy

Lossless

Account:

Username - Jude1234

Year - 12

Password - L.....oP1

Encryption Technique:

DES

AES

RSA

Save and exit

Above is another simple user interface diagram , I intend my user interface to be similar to this.

Currently there is a **lack of detail** I will be adding information regarding each setting and adding more options. Some users may not be aware of the effects of certain settings therefore adding more information is vital to creating an easy to use application.

I have only implemented a few settings as this will **decrease the need for error handling** as some users may not understand the settings sending a file with an incorrect compression technique causing errors for the sender and recipient.

The settings User interface will allow users to change their **encryption, account, and compression settings** users will also be able to change their password therefore I will be implementing a button to contact a server administrator to do this securely. If the application was to gain more users this process **could become automated** making use of **two-factor authentication** however due to server resources and the feature not being one of my stakeholders requests I will not be implementing this during the design phase however this could be implemented during **perfective maintenance**.

## Computer Science NEA project - Chat application

### Log-in user interface

The diagram shows a window titled "Chat Application" with standard window controls (minimize, maximize, close) in the top right corner. Centered within the window is a "Log-in" form. The form has a title "Log-in" at the top. Below the title are two input fields: the first is labeled "Username:" and the second is labeled "Password:". At the bottom of the form is a button labeled "Log-in".

Due to the nature of the application a sign-up user interface is not necessary as users will be given their accounts improving the security of the application. From the survey we know that only **8.3%** of participants messaged outside of school so proving the feature is not necessary.

I have designed this user interface to be intuitive which has been achieved by creating a familiar user interface similar to many applications and identifying the different parts of the user interface using text.

To **highlight** the key areas of this interface I will be outlining the log-in section with a **blue** outline and then accenting this section with a **dark grey** background.



## Computer Science NEA project - Chat application

### Stakeholder input regarding the proposed interface

**Jude Easton** - "I am very happy with the current user interface design as I am familiar with the layout and colour scheme of the interface. I am however concerned about the lack of detail within the settings menu I am not familiar with some of these terms so this lack of information will confuse me and others"

**Harry Taylor** - "This user Interface is perfect for me, it is intuitive and allows me to take control of the data I am sending using the settings menu, by splitting the interfaces the application is much cleaner and simplistic allowing me to really focus on my project"

**Group of year twelve students** - After discussing this user interface with the same group of students earlier, people who will be using this application. I have determined that the group consensus was that the User interface was perfect apart from the lack of detail in the settings menu leading to a confusing experience.

I will be adding more detail to the settings menu to solve this issue, below is a refined design based on this feedback.

I have added more data to the settings menu and simplified it by removing the complex terms, automatically choosing specific settings based on the environment. I have also added a privacy feature to the account setting stopping people from seeing the users password as this application will be used in schools where monitors are easily seen. I have also implemented a red star on important required settings to create a more intuitive user experience.

Chat Application

My Account:

Username: User124

Year: 12

Password: L---P5

Encryption Technique:

This is an optional setting however diabsling it will cause security risks, your emssages will be bale to be seen by others if disabled

Encryption

ON

OFF

Compression Technique: ★

Please choose a compression tehcnique this will determine if data it lost upon transmission howeveer this settigns iwll affect transmission speeds

Compression

Lossy

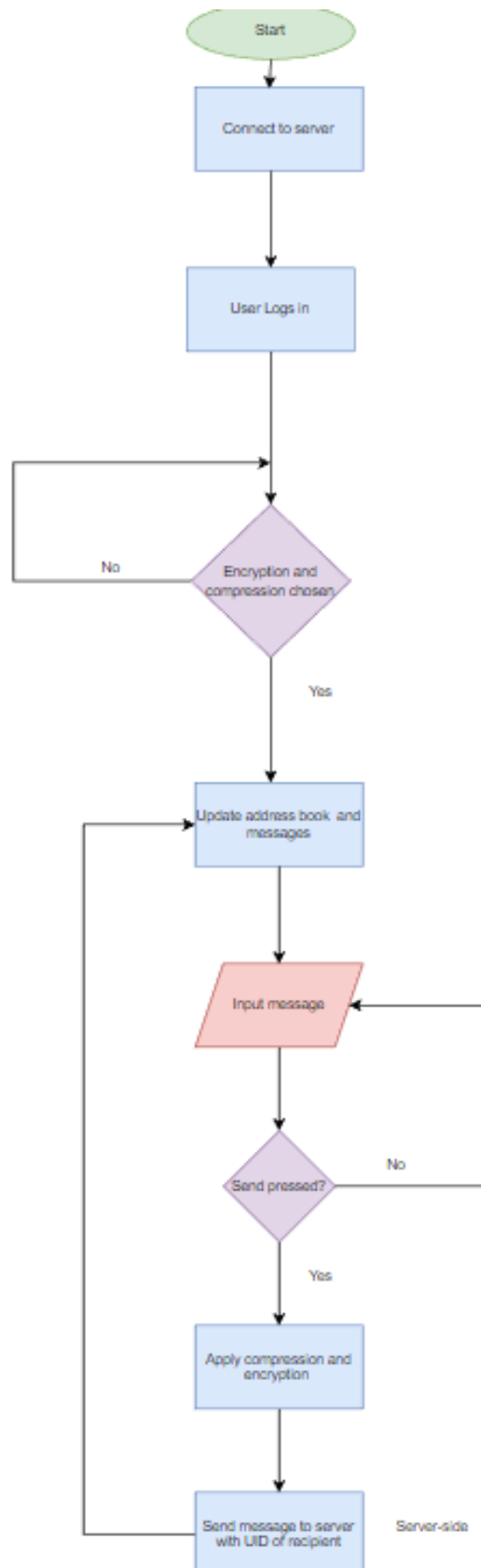
Lossless

Save and exit

## Computer Science NEA project - Chat application

### Algorithms

The hardest aspect of this application is determining the algorithm used to send messages, connect to the server and receive the messages, some of this will be server-side and some client-side. The algorithm will have to be carefully designed, below I have created an abstract flow-chart to represent the steps a user will undertake when sending a message:



## Computer Science NEA project - Chat application

I will be programming the application using modular programming splitting the flow chart above into modules that will make the program easier to read, allowing for easy maintenance and development.

Programming the application modularly will ensure that I am in line with user requirements and keeping within deadlines. After each module is programmed I will be looking back at this and testing it to ensure the program is functioning as intended by comparing the module to the success criteria.

Below I have listed the different modules that I will be breaking the problem down into and the different processes these modules entail, this is a very simplistic abstraction of the solution:

### Client-side

#### Logging In

- *Take input from the user*
- *Send input to server applying a hashing algorithm*
- *Compare input to the database on the server server will respond with a Boolean value*
- *Change variable allowing the user to continue*

#### Connecting to server

- *Use the in-built socket library function to connect using the servers IP/Hostname and port*
- *Sever connect*
- *The server sends back a confirmation message*
- *Check if connected(message received) if not keep trying*

#### Update application

- *Send a message to the server requesting an up-to-date address book and inbox*
- *Messages sent will be simple codes so the server can efficiently respond. By having a low time complexity per request the server will be bale to process other requests quickly*
- *The server will respond by sending the data, the decryption algorithm will need to be applied at this stage*
- *Update the user interface with the new data received*

#### Compress the message

- *Apply an appropriate compression algorithm(function) this will be stored in a dictionary stored alongside the data type there will be two dictionaries one for lossy and one for lossless compression which dictionaries are accessed will be determined by the user's input within the settings GUI*

## Computer Science NEA project - Chat application

### Encrypt the message

- *The data will be encrypted using an encryption algorithm*

### Transfer message to the server

- *Split the data into packets applying the necessary data*
- *Using an in-built socket library function the data will be sent using the servers IP and Port*

### Decrypt message

- *Apply the decryption algorithm*

### Decompress message

- *Apply the decompression algorithm specified in the transmission metadata*

### Hashing function

- *Hash data is passed into the function making sure the data is considerably smaller and different when returned, this function will be tested to ensure there are limited hash collisions.*

### Receive message

- *This function will be called to receive and decode a message this will be achieved by calling other functions such as the decryption and decompression functions. This function will return a dictionary storing the metadata and message*
- *The metadata will be separated from the message before the other functions are applied to the message as the metadata will not be encrypted or compressed due to its size and lack of importance*

## Computer Science NEA project - Chat application

### Server-side

#### Create client connection/client thread/create connection

- *I will program this using object-oriented programming creating "user" instances of the object. User instances will hold the Client's IP and Port, the server will create an instance of this object when a connection has been made via the in-built socket function .listen()*

#### Send Data

- *The data will be passed on to the recipient specified in the transmission metadata this will be done by making use of the in-built socket library functions using the IP that is linked to the recipient in the address database*

#### Update application

- *If the message metadata refers to this function from the function library using the send data function the application will send an up-to-date version of the address book stored in its database*
- *Check inbox and send any messages for the UID of the client then delete these messages once a confirmation message has been received from the client*

#### Deconstruct message/Receive data

- *The program will be listening for data received, I will use the select function from the in-built socket library to maintain a connection with multiple users*
- *The message will be decoded and searched for the code specifying the purpose of the data this will be compared to a dictionary and a function will be returned for example C103 could mean Update Application so this function would be called passing the current client UID into the function*

#### Hashing function

- *Hash data is passed into the function making sure the data is considerably smaller and different when returned, this function will be tested to ensure there are limited hash collisions*

## Computer Science NEA project - Chat application

### Algorithms in more detail

I am now going to break the problem down further this makes the application **easier to program** and understand by other programmers it ensures a structure is followed and the application has **clear deliverables**.

### Client-side

#### Logging in

I will be using a GUIZERO user interface to create a Login interface this will be a **simplistic interface**, it will take the users password and username relaying this data to the server using the **send function**. Authentication data will be passed into the hashing function the return of this will be passed as an argument with the messages **metadata** code into the send function to be relayed to the server.

Upon receiving the data the server will break the message up identifying the metadata and **returning a Boolean value** that my program will be waiting for.

#### Connecting to server

To connect to the server the client application will be making use **in-built functions** part of this is the **socket library**. I will pass into these functions the IP and port the server is on, to achieve this the server will have a static IP. The client will listen for data by calling the **receive message** function, once a confirmation message has been received the loop will be exited.

#### Update Application

Once a connection has been established the client will send an **information request** to the server using the **send message function**.

The application will listen for data using the **receive data** function, once the data has been received the loop will be exited and the application will be updated assigning relevant variables to the data received. Subsequently the GUIZERO user interface will be changed giving the appearance that the user's inbox and address book has been updated.

## Computer Science NEA project - Chat application

### Compression function

This algorithm will be represented as a function the function will take in the data to be compressed and then choose an appropriate compression technique based on the user's settings. The program will select a **lossy or lossless dictionary** retrieving the correct compression technique for the data type being compressed, below I have listed the compression techniques that will be programmed:

#### Lossless

- Run Length Encoding  
*This compression technique works by recording the order of pixels of a certain colour, for example, an image may be represented as 4B 2W 5R this is much smaller than storing the binary representation of the pixel.*
- Dictionary Encoding  
*Dictionary encoding is used on text. It records one instance of each word/character creating a dictionary then the program stores the number of these items in the text document for example "Fair is foul, and foul is fair." would be represented as 2 Fair, 2 is, 1 and etc*

#### Lossy

- *Used on images and recordings this algorithm removes data that is not noticeable such as frequencies not in the typical human hearing range and colours not on the human spectrum.*

I will by default be applying **lossy compression** to messages. **Lossless compression** techniques will always be applied to text as **text can not be compressed via lossy compression** as the file will not be readable after transmission.

### Encryption

Data will be encrypted after the necessary compression techniques have been applied to the message.

There is only a need for one function as regardless of file type the data type will be the same after compression this increases the efficiency of the program **reducing its time and memory space complexity**. The function will work by using **asymmetric encryption** with the server any data sent will be encrypted using a combined encryption key.

The function will already have a copy of the client's **public** and **private key** this is safer as no key transfer is necessary removing the chance of it being cracked this **combined encryption key** will be stored encrypted on each application however the key could become compromised if the application is decrypted so a strong encryption method will be used for this.

#### Transfer message to the server

This will be a function taking the message, **compressing it, encrypting it, and attaching any necessary metadata** which will be passed into this function. The function will **split the data into packets** and send it to the server IP and port using an in-built function.

The client will keep trying to send this data until a confirmation message is received.

## Computer Science NEA project - Chat application

### Message Decryption

To decrypt a server message the **private server key** will be used and the **recipient public key**, there is no need to authenticate the message as the keys will be **predetermined and stored locally** there is no way a hacker could pretend to be a client of the server due to this encryption method.

### Decompress Message

This function will read the message **metadata** to determine if **lossy or lossless compression** has been used. If lossless compression has been used the program will determine the type of compression and decompress it.

Below I have listed how the message will be decompressed for the different compression techniques:

### Run Length Encoding

- *The matrix sent will be read and the image reconstructed by copying the pixel pattern listed the matrix will be read left to right.*

### Dictionary Encoding

- *The dictionary sent with the data will be read and the text will be reconstructed; this will also work via index from low to high.*

### Hashing function

This function will scan through the text applying a caesar cipher then converting these to values based on the **ASCII table** this will then be reduced to a **three-digit number** choosing the digits from the **first, middle and last value** of the changed data. This algorithm may be updated later and altered depending on the outcome of testing and research.

### Receive Message

Using in-built functions I will receive incoming data to do this I will listen on the port for data being received, once received the data will be **decoded** and **held in a buffer** any missing packets or data will be requested. Once all packets have been received the **metadata will be pulled from the end of the message**.

Once the message has been decoded and the metadata has been removed the **data will be decoded and decompressed if necessary**.

The data is **decrypted, decompressed** and returned if its metadata corresponds to the value passed in; this is much more efficient than authenticating the metadata for each time it is used and memory space is saved due to the reduction of code.

### Dictionary Decoding

When a dictionary encoded message is received this function will be applied it will read through the dictionary received with the data rebuilding the text by looping through the array swapping each value for its corresponding character/word in the dictionary.

### Run Length Decoding (RLD)

Upon receiving a Run Length Encoded image this function will be called taking the **image size** and **mode** as parameters this will then be used to create a plain black image of the same dimensions and then loop through this images **pixel map** changing the **RGB pixel value** to match the transmission.

### Decompression

This function will serve as a **DLL(Dynamic Link Library)** linking functions. Its output will be the message decompressed this function will determine how data is decompressed.



## Computer Science NEA project - Chat application

### Server-side

#### Create Client/client thread/create connection

The server will be listening on its port for new connections once a connection has been made a **client thread** will be created using a class. This will allow me to create **multiple connections** efficiently, each user will have an **IP, UID, and port** stored in their object/thread. This will be required so that data can be received, sent and data can be looked for in the server's database using the user's unique identifier(UID).

#### Send Data

This module will be programmed similarly to the client version as they don't differ in required functionality.

#### Update Application

If a message's metadata specifies the update application abbreviation then this module will **search the server's database** for any information linked to the user's UID and send an updated address book to the user. This module will be programmed as either a method in the client class or a function.

MYSQL is the library that will be used to access the server's database.

#### Receive Message

This function will work similarly to the client's version however may be a method in the **client class**. If the message's **metadata** refers to logging in the hashed value will be compared with the **password table** within the database that will be storing the hashed versions of passwords. The username will be compared with the user database.

This algorithm will send a confirmation message once data has been received.

#### Hashing Function

This will have to be the same algorithm applied to other data to avoid **data corruption**.

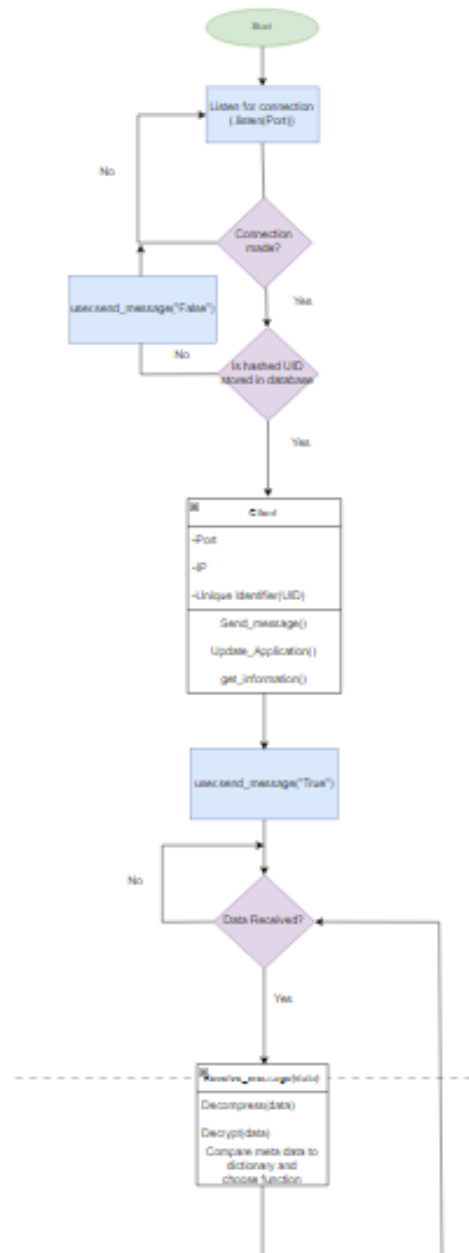
# Computer Science NEA project - Chat application

How the modules will link

## Server-Side



## Client-side



[Full Flowchart](#)

## Computer Science NEA project - Chat application

### Sampling Investigation

This algorithm will need to convert a large range of data types into the same data type taking less memory and doing it in a short amount of time. I will choose typical data types that are used for transferring data the most and investigate how the algorithm will break this down so the data can be passed into the encryption algorithm.



This is a floor plan that has been created by Harry Taylor, a stakeholder in this application. I will be showing how the algorithm will compress this image using lossy and lossless compression techniques.

#### Lossless compression techniques:

Run Length Encoding

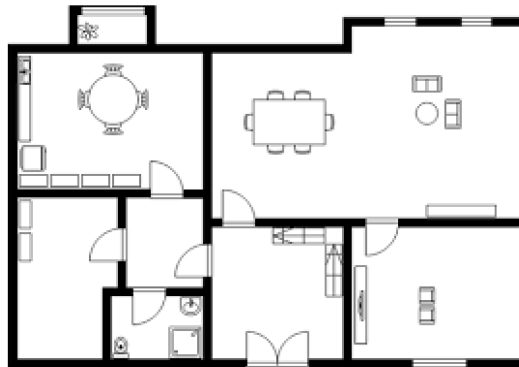


The floor plan has been pixelated and the pixels counted in reference to their colour ,the output produced for the first couple rows of pixels would look similar to this:

```
pixels = [{"15W", "10B"},  
          [{"15W", "1B", "10W", "1B"},  
          [{"10B", "2W", "10B", "11W"}]]
```

## Computer Science NEA project - Chat application

### Lossy compression techniques:



This image has been significantly compressed to show how the algorithm will work, the algorithm will scan through the image comparing each pixel's **RGB value** to a predetermined limit. If the RGB value exceeds this then the pixel will be removed from the image as this value will represent what pixels are noticeable and what is not, for example, a pixel may lie outside the **human colour spectrum** so can not be viewed anyway.

Once this has been achieved **Run Length encoding** will take place producing a similar output as before however taking up **less memory and processor time due to the fewer pixels**.

### Dictionary Encoding:

As dictionary encoding will only be called upon for text files I have gathered a separate sample this sample is an extract from a year thirteen's English Language coursework, this compression method will always be called when sending text files regardless of the user's settings as lossy compression should not be used on text as it would not reduce the file size and may cause data corruption it would also take up all of the bandwidth slowing the network down. Below is the extract and how it has been compressed:

#### Section B – Essay Question

'Even if he had stayed within the controlling order of Venice, Othello's tragic downfall was inevitable'. To what extent do you agree with this view?

Shakespeare's tragedy 'Othello', is indeed somewhat revolutionary. It contradicts the claim Aristotle made on how tragedies should take place at one time and in one place, however 'Othello' does not adhere to this. As a result, many critics have pondered over the importance of Cyprus, as a scene for a tragedy, often arguing that even if Othello had resided within the Venetian State, his downfall was inevitable. In terms of tragedy, this most certainly seems like a statement that is, if not wholly, then at least partially, true. Very little of the tragedy that Othello faces is actually explicitly linked to Cyprus, for when he arrived there is no war he must fight in as the Turks had been drowned. In Act Three, we even see Othello surveying the town for fortifications. Therefore, by choosing to construct 'Othello' as a tragedy that takes place over two locations, Shakespeare is defying the notion that place and time matter. What Shakespeare is giving importance to is not setting, but society and emotions. Human nature is where the real tragedy lies, and this presides over all places.

```
wordsdict = {
  1: "if",
  2: "even",
  3: "had",
  4: "he",
  5: "stayed",
  6: " "
}
numb_occ = [2,6,1,6,4,6,3]
```

## Computer Science NEA project - Chat application

Each word has been put into a **dictionary** and **assigned a key** this key represents the word(item) and is displayed in the **numb\_occ** array, this array represents the order of the text deciphering this would give us the first couple words of the extract, "even if he had", this text gives a length of 14 characters while the compressed version only has a length of 7 characters the size of the text has been **reduced by 50%**.

### Script Communication Codes

To work as intended the client and server will need to communicate with each other specifying what to do with data this can be achieved by including **metadata** in transmissions.

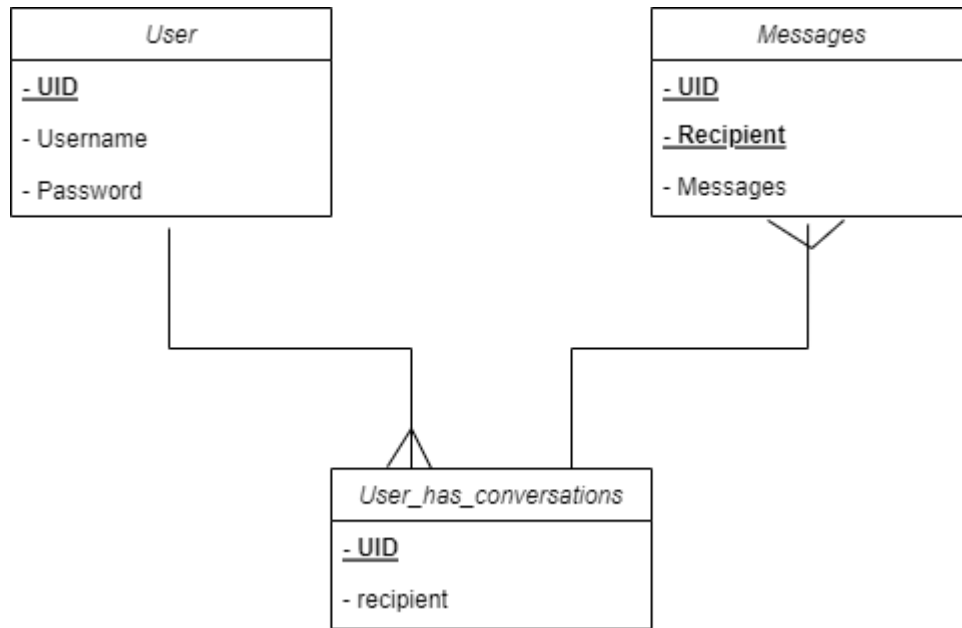
Below is a table specifying the meaning of each code. These codes have been chosen as they are minimal in size taking up little bandwidth improving the data transfer speed and have a low chance of collisions which could be detrimental to the program.

Code	Use	Explanation
#1	Update	Split data, using user ID to respond with database contents.
#2	Authenticate	Compare data with the database, username, and password.
#3	Connected	Return True if received.
#4	Compressed	Respond accordingly by checking other metadata to choose what decompression software is necessary.
#5	Encrypted	Data is encrypted and makes use of the decryption function.
#6	Dictionary Encoded	Data is Dictionary encoded decompress using the appropriate decompressor.
#7	Run Length Encoding (RLE)	Data is Run Length encoded decompress using the appropriate decompressor.
#8	Forward (Send data to IP)/message	Use UID storing data in the corresponding inbox.
#9	Received	Used as a confirmation for data receipt.
#10	Report	Used so the server knows to store the information as a report.
#11	Close the socket	Sent when the user exits the application stops the application crashing due to an open socket. More efficient too as the program is not waiting to receive data.
#1A	Update messages for recipient	Query Database, reformat message and send to client

## Computer Science NEA project - Chat application

### Database design

Due to the nature of my problem I will be using a database that will allow me to store lots of data corresponding to users in an organised manner. A database makes it easy to access and change this data. The entity relationship diagram below is a representation of the database I am going to be using. It is designed with three tables and connected via keys underlined.



I am going to need to fill the user database with the whitelisted users. The database will include the hashed values of users password, usernames and the non hashed unique identifiers, below is the table of **passwords,UIDs and usernames** that will be inputted into the table.

<u>UID</u>	username	password
Brooklyn@Limbert127	Brooklyn.Limbert	@4pSEu
Harry@Taylor456	Harry.Taylor	V5^uiW
Jude@Easton342	Jude.Easton	cg8N=#
Nat@Gray111	Nat.Gray	Sk6ol!

<u>UID</u>	Hashed username	Hashed password
Brooklyn@Limbert127	717	161
Harry@Taylor456	527	206
Jude@Easton342	440	96
Nat@Gray111	292	125

## Computer Science NEA project - Chat application

### Key Pseudocode functions and modules

#### Client Side:

##### Logging-In:

```
Function log-in(username,password):  
    send_message([hash(username),hash(password)],'#2')  
    return receive_message(8081,"#2")
```

This function works by comparing the user's hashed username and password to the hashed database within the server. This is very secure as there is no way to revert the hash function if intercepted data is secure.

Once the data has been sent with the hashed value and the correct metadata the server will respond, the "receive\_message" function will wait for data with corresponding metadata and return the message. The log-in functions return will be handled by the main program.

##### Connect:

```
Function connect():  
    socket.connect((IP, 8081))  
    socket.sendall('#3')  
    return (receive_message(8081, "#3"))
```

This function will be called to connect to the server this is done by sending the correct metadata and awaiting a reply the boolean value replied will be returned to the main script.

##### Update Application:

```
Function update():  
    send_message([UID], '#1')  
    data = receive_message(8081, "#1")  
    add_book = data[0] #Address Book Array within matrix  
    inbox = data[1] #Inbox Array within matrix
```

To update the application this function will be called. The function works by sending a message to the server with the unique identifier of the user, the function will then await the response from the server. Once a reply has been received the message will be broken up and assigned to the correct variables subsequently updating the information being displayed.

## Computer Science NEA project - Chat application

### Run Length Encoding (RLE):

```
Function RLE (image_path):
    rgbs = []
    counter = 0
    im = Image.open(image_path)
    for row in range(im.size[0]):
        for col in range(im.size[1]):
            rgbs.append(str(im.getpixel((row,col))))
    return (dict_encode(' '.join(rgbs),False))
```

This function scans through the image specified by the “image\_path” parameter; this is done via two for loops that act as coordinates. The pixel is chosen accordingly and its RGB values temporarily stored, this is then encoded via the “dict\_encode” function and returned.

### Dictionary Encoding:

```
Function dict_encode(data,file):
    dict = {}
    text = []
    counter = 0
    if file:
        data = open(data)
        for line in data:
            text.append("/n")
            for x in (line).split():
                print(x)
                text.append(x)
        return dict_encode(' '.join(text),False)
    else:
        data = data.split()
        data = list(map(lambda x: x.replace("/n","\n"),data))
        for i in data:
            if not(i in dict.keys()):
                dict[i] = counter
                data[data.index(i)] = dict.get(i)
                counter = counter + 1
            else:
                data[data.index(i)] = dict.get(i)
        return [dict,data]
```

This function encodes data first by determining if the data passed in is a file or not if it is, the file is opened and converted into an array that is then passed into the function as a string rather than a file.

I have designed the function this way to make the program more memory efficient as less code is required.

If the data is a file the data is split and turned into an array, a new line character “\n” is added to the end of each line so the file is reconstructed correctly.



## Computer Science NEA project - Chat application

### Lossy Compression:

```
Function loss_comp(image_path):
    im = Image.open('original.png')
    rgbs = []
    counter = 0
    for row in range(im.size[0]):    # for every col:
        for col in range(im.size[1]):# For every row
            rgbs.append(im.getpixel((row-1,col)))
    return (dict_encode(rgbs),[im.size,im.mode])
```

This function is similar to the RLE function however it stretches each pixel in the x direction creating a lower resolution image reducing its size significantly as many RGB values are now the same.

The function will return the image compressed via lossy compression with the correct metadata; image mode and size. Metadata is essential for image reconstruction after transmission. This method of compression creates unnoticeable changes as only one-pixel colour is copied compared to stretching the pixel more.

This method is different to the design however it is more efficient taking up less memory while achieving the same result, therefore I have programmed the compression technique in this way.

### Compression Function:

```
Function compress(data,technique,image,file):
    if technique == 'lossless':
        if image:
            return RLE(data)
        else:
            return dict_encode(data,file)
    elif technique == 'lossy':
        return loss_comp(data)
```

This function works similarly to a DLL(Dynamic Link Library) ; it calls the compression functions in correspondence to the type of compression and data.

## Computer Science NEA project - Chat application

### Asymmetric Encryption/Decryption:

```
Fucntion caesur(plaintext,shift):
    encrypted = []
    while True:
        encrypted = []
        for i in plaintext:
            encrypted.append(chr(ord(chr(ord(i)+shift))+shift))
        encrypted = "".join(encrypted)
    return encrypted

Function key_A_PUB(data):
    data = (list(map(lambda x: caesur(x,5),data)))
    return data

Function key_B_PRIV(data):
    data = (list(map(lambda x: caesur(x,-5),data)))
    return data

Function key_C_PUB(data):
    data = (list(map(lambda x: caesur(x,2),data)))
    return data

Function key_D_PRIV(data):
    data = (list(map(lambda x: caesur(x,-2),data)))
    return data
```

Data is encrypted using a **caesar cipher** by taking away a shift value from the Unicode of the character and converting this back to a character, this is then joined to the rest of the text. This is done twice to add extra security however a caesar cipher is not a very secure encryption method so it is paired with asymmetric encryption.

Asymmetric encryption is achieved by creating different keys with different shift values. I have used a caesar cipher algorithm as it is memory efficient and has a low time complexity allowing data to be quickly encoded and decoded while taking up little memory space.

## Computer Science NEA project - Chat application

### Sending messages:

```
Function send_message(data,meta):  
    if connect() and timeout<5:  
        data.encode()  
        meta.encode()  
        socket_client.sendto([data,meta](IP,8081))  
        timeout = 0  
        return True  
    else:  
        timeout = timeout+1  
        send_message(data,meta)
```

When called this function will take messages metadata and data as parameters with this it will encode them, storing them in an array and sending them to the server to be processed.

The data has been put in an array so metadata can be easily obtained upon arrival, the "encode()" function breaks the data down to bytes so it is ready to be transferred. I have ensured that the user is connected to the server before attempting to send data as this could cause unexpected results. I have implemented a countdown timer making the function attempt to connect four times.

### Hashing algorithm:

```
Function hash(value):  
    hash_val = 0  
    hashed = []  
    for i in value[::-1]:  
        hashed.append((ord(i)) + 10)  
    for i in hashed:  
        hash_val = i + hash_val - 66  
    return hash_val
```

This algorithm works as a hashing algorithm. It is a simple hashing algorithm however it is sufficient for this scenario as it reduces the size of data significantly producing the same output for the same characters.

The algorithm splits the data up into characters and assigns a value to each character based on its Unicode value. The algorithm adds a number to this and scrambles the value further by adding more data to it.

## Computer Science NEA project - Chat application

Receive message:

```
Function receive_message(port,meta):
    if connect():
        data = client_socket.recv(port)
        data = decode(data)
        if data[1] == meta:
            send_message(True, '#9')
            return decompress(data[0][0],data[0][1],data[0][1][0],data[0][1][1])
```

This function will be called when the application is requesting data. The function listens on the port passed into the function, data is received then checked to see if its metadata corresponds to the "meta" parameter if it does the message is returned and a confirmation message is sent to the server.

I will be ensuring all data sent is in the form represented in the decompression section. This function is going to need to be adapted to ensure that all data is received and any lost is requested. I will be looking at the implementation of a buffer during the programming and testing section of the project.

### Dictionary Decoding

```
Function dict_decode(data,file):
    dict = data[0]
    text = data[1]
    counter = 0
    if file:
        text = dict_decode(data,False)
        f = open("temp1.txt","a")
        f.writelines(text)
        f.close()
    else:
        for i in text:
            for x in list(dict.keys()):
                if i == dict[x]:
                    text[text.index(i)] = x
        return " ".join(text)
```

To decode dictionary encoded data the algorithm first splits the parameters up into specific parts. The algorithm will determine if the data is a file, if it is the data is read and decoded using function recurrence. Data is passed as a string rather than file during recurrence, the output of this will then be written to the file. The open method is "a" to handle errors as the file may not have been created yet.

## Computer Science NEA project - Chat application

### Run Length Decoding

```
Function RLD (metadata):  
    rgbs = []  
    new = Image.new( metadata[1], metadata[0], "white")  
    for row in range(new.size[0]):  
        for col in range(new.size[1]):  
            new.putpixel((row,col),int(rgbs[counter]))  
            counter = counter + 1  
    new.save('temp.png')
```

This function will be called by the compression function, it will be used to reconstruct the image from an array of RGB pixel values. X and Y coordinates are created using two for loops, a black image is also created.

The black image created is to the same dimensions of the image being reconstructed, each pixel is changed in the pattern they were recorded during run length encoding. The image created has the same image mode and image dimensions as the image encoded. The image created is a duplicate of the one encoded; it will be saved to the 'temp.png' storage location.

### Decompress

```
Function decompress(data,technique,image,file):  
    if technique == 'lossless':  
        if image:  
            return RLD(data)  
        else:  
            return dict_decode(data,file)  
    elif technique == 'lossy':  
        return data
```

This function is similar to the compression function however it is doing the opposite, calling the decode function based on its parameters.

### Report

```
Function report(uid,report,uid_recipient):  
    send_message(compress([uid,report,uid_recipient],comp_technique,False,False),"#10")
```

When reporting a user this function will be called it will send the data inputted to the server and stored in the server report database with the users uid and user recipient uid.

## Computer Science NEA project - Chat application

### Server Side:

#### Create Client/client thread/create connection class

<i>ThreadC</i>
-clientsocket:integer
-clientaddress:string
-ClientID:string
+send_message(self,recipient,message)
+client_info(self)
+update(self)

```
class ThreadC(threading.Thread):
    def __init__(self,clientsocket,clientID,clientIP,clientName):
        threading.Thread.__init__(self)
        self.client_socket = clientsocket
        self.client_ip = clientIP
        self.UID = clientID
        self.name = clientName

    def send_message(self,recipient,message):
        cursor = db.cursor()
        cursor.execute(f"UPDATE users SET inbox = {[message,self.name]} WHERE uid = {recipient}")
        db.commit()

    def client_info(self):
        return [self.client_socket,self.client_ip,self.UID,self.name]

    def update(self):
        cursor = db.cursor()
        cursor.execute(f"SELECT {self.UID} FROM users")
        data = cursor.fetchall()
        return send_message(data,"#1",self.client_ip,self.client_socket)
```

To create client connections I have taken an object oriented approach as each client will be created each having different attributes however they will have similarities as all users have similar functionality.

Creating a class and using instances of this for clients is more efficient as less memory is taken up due to the smaller amount of programming required, this approach also ensures that all clients can connect as a limit does not need to be predetermined.

I began to create this class by creating a UML diagram, I then programmed the class using this diagram. The class works by assigning specific attributes to each client such as their address, port and UID(unique identifier). When the "send\_message" method is called the program will use MySQL to store the data passed in the users database inbox. The data in the inbox will then be retrieved when the program updates, this saves memory and time as the program has less processing to do as the program only needs to send data from one method the "update" method.

### Connect

```
Function connect(IP,port):
    socket.connect((IP,port))
    socket.sendall('#3')
    return receive_message(8081, "#3")
```

This function works the same as the client-side script however due to the IP and Port changing per client I have added these as parameters so that an object can pass in its attributes as arguments allowing the program to connect to that client.

## Computer Science NEA project - Chat application

### Send\_message

```
Function send_message(data,meta,IP,port):  
    if connect(IP,port) and timeout<5:  
        data = data.encode()  
        meta = meta.encode()  
        socket.sendto([data,meta],(IP,8081))  
        timeout = 0  
        return True  
    else:  
        timeout = timeout+1  
        send_message(data,meta,IP,port)
```

This function is similar to the client-side version however takes the IP and port in as parameters as clients will be using different ports and IPs.

### Compress

```
Function compress(data,technique,image,file):  
    if technique == 'lossless':  
        if image:  
            return RLE(data)  
        else:  
            return dict_encode(data,file)  
    elif technique == 'lossy':  
        return loss_comp(data)
```

The compression function works the same as the client version.

### Receive message

```
Function receive_message(port,meta,IP,port):  
    if connect():  
        data = client_socket.recv(port)  
        data = decode(data)  
        if data[1] == meta:  
            send_message(True,'#9',IP,port)  
            return decompress(data[0][0],data[0][1],data[0][1][0],data[0][1][1])
```

This function is similar to the client-side version however has a change in arguments for the “send\_message” function.

### Hashing algorithm

```
Function hash(value):  
    hash_val = 0  
    hashed = []  
    for i in value[::-1]:  
        hashed.append((ord(i)) + 10)  
    for i in hashed:  
        hash_val = i + hash_val - 66  
    return hash_val
```

This function is identical to the client-side version.

## Computer Science NEA project - Chat application

### Testing the algorithms

To ensure the program works correctly no matter how it is used **rigorous testing is necessary**. Testing ensures that all data is **error handled** and any problems that occur in the program can be handled without the program crashing or unintended actions being produced as this could cause security concerns as well as usability issues.

To test the program, I will be using **trace tables** to perform **validation, verification, functional and non-functional tests**. I will be spending a large amount of time testing the program later in the application development until then I will be performing basic tests as the application is tested. This is known as **iterative testing**. I will then be performing **basic unit tests**. After the application has been designed I will document the testing that will be performed on the application to ensure no areas are missed.

### Explanation and justification of this process

Due to the complex nature of the application the problem requires lots of computational thinking, I have employed the use of **divide and conquer, thinking ahead, abstraction and logical thinking** through this approach to the problem. I have displayed this through the breakdown of the problem into modules. These modules have been designed to work with each other, they have been designed to be **efficient** taking little memory space and processing power. Problems have been broken down further with some modules/functions employing the use of another function to achieve the desired result.

I have created **flowcharts** and **UML diagrams** to plan the modules, working out how the module will communicate with the rest of the program. I have employed the use of **abstraction, logical thinking and thinking ahead**.

I have designed the modules using a mix of **pseudocode and python**. I am going to be programming the application in **Python**. Due to the decomposition of the problem using pseudocode the development section will be fast and efficient, allowing me to focus on the testing of the application and ensuring the program meets the **success criteria**.



## Computer Science NEA project - Chat application

### Inputs and expected outputs

Input	Process	Output
Username	Hash and compare with server database.	Boolean value.
Password	Hash and compare with server database.	Boolean value.
Compression technique	If statement used to branch program based on compression technique.	Compressed output, array.
Encryption technique	If statement branches program to specific encryption algorithm.	Encrypted output, array.
Exit	Send Metadata, close socket then application	Closing of application
Report	Users input their report and the report is logged (sent to the server and stored).	No output.
Settings	Settings user interface is opened upon mouse click on the symbol.	Settings user interface loaded.
File choice	User chooses a file and the file is uploaded into temporary program storage.	Operating systems file explorer is opened allowing file selection.
Save file	Files in temporary storage are saved.	No output.
Send message	Call function and send message to the server, send data inputted into the user interface.	Displays messages in the recipient message history.
Recipient choice	User chooses the recipient of the message, this in turn displays an inbox from that user requesting this from the server.	Displays message history between user and recipient.

## Computer Science NEA project - Chat application

### Key Variables

I have listed the key variables as part of the **pseudocode algorithm** and listed some **global key variables** below. Every variable is essential to the program's functionality making it difficult to display key variables, most have been separately displayed through the pseudocode algorithms above.

The most important key variables are below these are key variables that the program could not function without:

#### Server-Side

Variable Name	Data Type	Function
nextm_buffer	String	Store any data received after the terminator has been received.
SPORT	Integer constant	Stores the port the application is running on.
SIP	String constant	The Ip the server is binding the port to.
BUFF_SIZE	Integer constant	Stores the buffer size/packet size.
db	Object	Used to store a database object enables the program to access the database.
s	Object	The socket that the IP and port has been binded to.

#### Client-Side

Variable Name	Data Type	Function
SIP	String constant	Server IP to be connected to.
SPORT	Integer constant	Server port to be connected to.
UID	String	Users log-in details removes the possibility of an SQL injection attack.
nextm_buffer	String	Stores and data received after the terminator has been received.
comp_technique	String	The compression technique being used.
address_book	List	Users that can be messaged.
s	Object	The socket being used.
messages	Dictionary	Messages stored per user to be displayed in the GUI.
app	Object	Object used to create the GUI.

## Computer Science NEA project - Chat application

### Validation

All data inputted is going to need to be **validated or error handled** to avoid unexpected results. I am going to ensure inputs are error handled so that **only valid data can be inputted**. I will be making sure all data is universal by making the majority of functions **output the same data type**, therefore a function will only need to take in an **array** ensuring the program runs as designed. The methods I am going to be using to ensure data is inputted correctly are listed below:

#### File explorer

This function will be ran when the file explorer button is pressed it will display the system's file explorer allowing a user to import a file into the program. To ensure that the file imported is of a valid data type the file explorer will check the file type when the file is selected to be imported. This ensures that the wrong data is not inputted as this would produce undesired results due to the compression algorithms.

#### Buttons

Buttons are another solution to data validation as the user can only input one type of data, a boolean value of either True or False. Buttons will be linked to a function calling this function when the boolean value is True(the button has been pressed).

#### Text Boxes

Text boxes ensure that data is a string allowing me to predict the data type and design the program around this ensuring the program works correctly as all data is of the same type.

#### Drop down menus

Menus like this ensure that the user only selects usable options this ensures the program will work correctly as I can program the application to process these without error handling before they are selected as I know the user will not be choosing any other values as it is not possible.

## Computer Science NEA project - Chat application

### Testing checklist

What to test	Improvements
Check GUIs meet user requirements and design	
Compression and encryption methods	
Save and Exit button	
Address book	
Download button	
Settings button	
Report button and interface	
File explorer button and file import	
Dropdown quick message menu	
Message history	
Sending and receiving message	
Log-in text boxes	
Log-in authentication	

This list will ensure that the program works correctly and no sections have been missed, all sections will be tested thoroughly by following this list. For each section of the list I will be creating **truth tables** and choosing a testing technique.

For each section tested I will identify improvements that can be made; these improvements will then be carried out depending on the importance of the improvements when compared with the user requirements and the allocated maintenance time within the application development.

## Computer Science NEA project - Chat application

### Iterative unit testing

To ensure the program works as intended when the modules are joined together I will be testing each individual module as I program them. The result of this will allow me to make adaptations to the module ensuring it works with other modules. If iterative testing is not carried out it will be hard to identify problems within the final design.

Input	Expected Output	Valid?
Normal Data(Expected data type and data)	True	True
Extreme Data(Valid data but upper and lower limit of data)	True	True
Erroneous Data(Erroneous data)	False	False
Normal Data(Expected data type and data)	True	True
Extreme Data(Valid data but upper and lower limit of data)	True	True
Normal Data(Expected data type and data)	True	True
Extreme Data(Valid data but upper and lower limit of data)	True	True
Erroneous Data(Erroneous data)	False	False

I will be using this trace table as a template to iteratively test units the table contains each type of data more than once to ensure that the program is working correctly. Using this table ensures that each module is thoroughly tested and the table will be applicable to all units.

## Computer Science NEA project - Chat application

The application is GUI based therefore test data will be interaction with elements of the GUI and the output will be the response from the program.

There is only one part of the application where specific test data will be chosen, this is the sending of messages, below I have created a trace table with specific test data. The test data has been chosen to test certain aspects of the application, test data will be of either extreme erroneous or normal data.

Test data	Data type	Reason
12345678910	Integer	To test how the program handles large integers.
!#\$%)	String	Test the program's formatting, how will complex characters be handled?
True	Boolean	Will a boolean value interfere with database querying.
*	Character	Will a special single character be able to be sent?
How are you today?	String	A typical message.
19762.123123	Floating point	Unique message will a floating point value be able to be stored and retrieved from the database.

### Performance Testing

To ensure the application satisfies the hardware requirements listed I will be conducting performance testing. The application will be run with a resource monitor; the resources used will then be recorded and compared with similar applications. Performance testing will ensure the application can run on all systems that meet the recommended hardware requirements listed earlier.

I will also be looking at calculating the Big O of the modules and algorithms used within the application to ensure the application is as efficient as possible.

### Development and testing

Due to the nature of the application I will begin this development phase by programming all of the functions that have previously been programmed in pseudocode. I have chosen to program the application this way as many different parts of the script require many different functions. If all functions are not programmed first parts of the program can not be tested causing problems later in development.

I have created two different files one with the server-side script and the other a client-side script. To begin development I converted the pseudocode functions to python code.

I programmed the server-side script first, setting up the database and programming the functions developed using pseudocode.

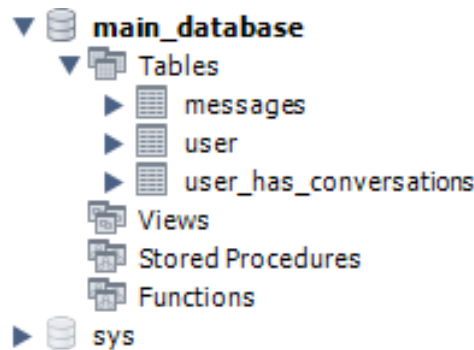
### Server-Side

#### Database

I have created the database using the **MySQL Database Workbench GUI**, I used this software as it ensured that the database was **secure,normalised** and the server-side script could access the database as intended.

As well as using the GUI features of the database I also used **SQL(structured query language)** to create, input and access the database.

The database structure is below:



## Computer Science NEA project - Chat application

### Messages

	UID	recipient	messages
▶	Brooklyn@Limbert127	Harry@Taylor456	{'hello': 0}/,
	Brooklyn@Limbert127	Jude@Easton342	{'hey': 0}/,
	Brooklyn@Limbert127	Nat@Gray111	{#, #, #{da
	Harry@Taylor456	Brooklyn@Limbert127	NULL
	Harry@Taylor456	Jude@Easton342	NULL
	Harry@Taylor456	Nat@Gray111	NULL
	Jude@Easton342	Brooklyn@Limbert127	NULL
	Jude@Easton342	Harry@Taylor456	NULL
	Jude@Easton342	Nat@Gray111	NULL
	Nat@Gray111	Brooklyn@Limbert127	NULL
	Nat@Gray111	Harry@Taylor456	NULL
	Nat@Gray111	Jude@Easton342	NULL
✱	NULL	NULL	NULL

Table: **messages**

Columns:

<u>UID</u>	varchar(45) PK
<u>recipient</u>	varchar(45) PK
messages	longtext

This table was created to store conversations that have been made for each user in the system, the UID and the recipient fields combined uniquely identify data within the application, therefore UID and recipient fields create a composite key. The composite key is used to identify specific conversations(records).

Due to the size of UID and Recipient I have created the table so that UID and recipient is only storing a **VARCHAR of 45** characters. I am not expecting a larger UID as they are assigned by an administrator to whitelisted users.

I have chosen **longtext** as the variable type for the messages field as this can store up to **4GB** of data. I am not expecting more message data than this at the current deployment level, however if the application was deployed to a larger audience, I would create more tables to store message data and change the application so some message data is stored locally.



## Computer Science NEA project - Chat application

### User

	UID	username	password
▶	Brooklyn@Limbert127	717	161
	Harry@Taylor456	527	206
	Jude@Easton342	440	96
	Nat@Gray111	292	125
★	NULL	NULL	NULL

Table: **user**

#### Columns:

<u>UID</u>	varchar(255)
	PK
username	int
password	int

This table's **primary key is UID**, the table is used for **log-in** and **authentication** purposes hence the **Integer** variable type for username and password as these store the hash value of log in details. Storing hashed data is much safer compared to storing the actual data as no transmission of sensitive data is required, this method also speeds up item searching.

### User\_has\_conversations

	UID	recipient
▶	Brooklyn@Limbert127	Harry@Taylor456 Jude@Easton342 Nat@Gray...
	Harry@Taylor456	Brooklyn@Limbert127 Jude@Easton342 Nat@G...
	Jude@Easton342	Brooklyn@Limbert127 Harry@Taylor456 Nat@G...
	Nat@Gray111	Brooklyn@Limbert127 Harry@Taylor456 Jude@...
▲	NULL	NULL

Table:

#### **user\_has\_conversations**

#### Columns:

<u>UID</u>	varchar(255)
	PK
recipient	varchar(255)

To identify conversations between users I have created this table that stores conversations users have with each other this can be used to select messages from conversations between these users. Upon a larger deployment of the application I would find a way of representing this data in another form as the varchar limit has been set to 255 characters this will not be large enough for a larger user-base.

## Computer Science NEA project - Chat application

### Functions

Some algorithms have significantly changed from the pseudocode algorithms to create more efficient solutions.

#### Encryption:

```
5  #Encryption Functions -----
6
7  def caesar(plaintext,shift):
8      encrypted = []
9      while True:
10         for i in plaintext:
11             encrypted.append(chr(ord(chr(ord(i)+shift))+shift))
12
13         encrypted = "".join(encrypted)
14         return encrypted
15
16  def key_A_PUB(data):
17      data = (list(map(lambda x: caesar(x,5),data)))
18      return data
19
20  def key_D_PRIV(data):
21      data = (list(map(lambda x: caesar(x,-2),data)))
22      return data
```

```
176 | | | image_data = (key_D_PRIV(key_A_PUB(image_data)))
```

This function works the same as the pseudocode function, to achieve encryption data needs to be formatted as a list data structure.

The encryption technique used is asymmetric encryption this is achieved using two sets of keys each key containing a unique **caesar cipher shift value**. The encryption technique is not very strong as it is using the caesar cipher encryption method.

As mentioned this encryption method is not very secure, however due to it being **asymmetric** a hacker would have to access the **source code** to gain the shift values. I could implement a random shift for each message however due to time limitations and my clients interests I will not be implementing this at the current deployment level. The caesar cipher algorithm ensures that data is encrypted quickly, a necessity for smooth user experience especially when sending large files and images.

The encryption technique is applied using the two asymmetric encryption keys (programmed as a composite function) they are applied when sending messages to ensure the correct data format is passed into the caesar cipher.

Data it is split before being inputted in the function formatting it as a list, I have insured that all messages when sent and received are of the same format and joined using "##,##" allowing me to split the message up and encrypt/decrypt the message ensuring universal formatting.

To decrypt the message I will use the keys in a different order (programmed as a composite function).

## Computer Science NEA project - Chat application

### Hashing:

```
24 def hash(value):
25     hash_val = 0
26     hashed = []
27     for i in value[::-1]:
28         hashed.append((ord(i)) + 10)
29
30     for i in hashed:
31         hash_val = i + hash_val-66
32
33     return hash_val
```

To compare the hashed values with hashed values stored in the database this function is used. The function will be applied to received data so it can be compared with hashed values in the database.

A hash value is generated by taking in the data as a list reversing this, then adding 10 to each unicode value for each character in the data. The values produced are then looped through adding their position to the hash and taking sixty six away from them.

This method ensures that all hashed values are smaller than the original input and all messages can be hashed.

### Send\_message:

```
def send_message(data,meta,image,file,comp_technique):
    print("Sending",data)
    if image == True or image == "True":
        data = data.split(";;")
        image_meta = data[1]
        image_data = data[0]
        image_data = (key_D_PRIV(key_A_PUB(image_data)))
        to_be_sent = (f"{image_data};#{image_meta}###{meta}###{comp_technique}###{image}###{file}///n").encode()
        clientsock.sendto(to_be_sent,(clientaddress[0],1024))
    else:
        data = "##,##".join((key_D_PRIV(key_A_PUB(data.split("##,##")))))
        to_be_sent = (f"{data}###{meta}###{comp_technique}###{image}###{file}///n").encode()
        clientsock.sendto(to_be_sent, (clientaddress[0],1024))
```

The send message function is very different to the pseudocode algorithm. Data is passed into the function and processed, its format is changed and sent to the client to be processed by the receive message function.

The function starts by checking if the data passed in is an image if so it is split up using the expression used to join images “;;”. The essential parts of the image are stored in image\_data and image\_meta they are then decoded and sent to the client using the client address. The images data is sent in packet sizes of **1024 Bytes**.

If the data is not an image the data is encrypted and joined by “##,##” it is then sent to the client address in **1024 Byte** sized packets.

## Computer Science NEA project - Chat application

### Decompression:

```
def decompress(data, technique, image, file):  
    if image == True or image == "True":          #Images always RLE even if it has been lossy compressed before  
        decoded = dict_decode(data[0], False)  
        metadata = [decoded[2], decoded[1]]  
        return RLD(decoded, metadata)  
  
    if technique != None and technique != "None": #Apply the correct decompression method, text will always be dict_encoded  
        return dict_decode(data, file)  
  
    else:  
        return data
```

The decompression function works as a mediator between the data and the different decompression functions, they are applied to data based on the type of data passed into the function.

The function identifies if the data is an image if so it is split into necessary parts and passed into the run length encoding function. If the data doesn't represent an image the dictionary decoding function is applied to the data however if the data hasn't been compressed at all, the function will return the data passed in.

### Dictionary decode:

```
150 def dict_decode(data, file):  
151     counter = 0  
152     print("being decoded", data, file)  
153     if file == True or file == "True":  
154         text = dict_decode(data, False)  
155         print(text)  
156         text = list(map(lambda x: x+" ", text))  
157         print(text)  
158         f = open("temp1.txt", "w")  
159         f.writelines(text)  
160         f.close()  
161     else:  
162         data = data.split("/", "/")  
163         dict = ast.literal_eval(data[0])  
164         text = ast.literal_eval(data[1])  
165         for i in text:  
166             for x in list(dict.keys()):  
167                 if i == dict[x]:  
168                     text[text.index(i)] = x  
169     return text
```

The dictionary decode function works similarly to the pseudocode algorithm, the function works by taking the parameters file and data to determine how to decode the data.

The data is split up into its part using `ast.literal_eval()` which converts string lists into lists, this is done as the data is sent as a string which is quicker to send. The data is checked to see if it is a file, if true the function is reapplied to the data as it is text only this is a recursive approach to the problem creating a more efficient program.

Once the data has been decoded it is saved into the text file 'temp1.txt'. I have opened the file using the write method "w". If the data type is not a file the data and dictionary is looped through and the text is changed to the corresponding dictionary value the data is then returned.

## Computer Science NEA project - Chat application

### Receive\_message:

```
40 def receive_message(meta):
41     global nextm_buffer
42     buffer = nextm_buffer          #Adding previous received data to buffer
43     data = ""
44     terminator_pos = -1           #By default no terminator in transmission
45
46     while terminator_pos == -1: #If equal to -1 ///n must exist
47
48         buffer = buffer + (clientsock.recv(1024).decode())
49         terminator_pos = buffer.find("///n")
50         if terminator_pos != -1:
51
52             nextm_buffer = buffer[terminator_pos + 4:]
53
54         data = buffer[:terminator_pos]
55         buffer = buffer[terminator_pos + 1:]
56
57         #Temporarily store any data received after terminator
58
59     data = data.split("##,##")      #Split message into parts
60     if data[-2] == "True" or data[-2] == True:      #Check if image received
61
62         image_info = data[0].split(";#;")           #Split image section
63         image_meta = image_info[1]                  #Store corresponding image data
64         image_data = image_info[0]                  #Store corresponding image data
65         image_data = ast.literal_eval(image_data)    #Convert Data into correct form
66         image_meta = ast.literal_eval(image_meta)    #Convert Data into correct form
67         image_data = key_D_PRIV(key_A_PUB(image_data)) #Decrypt
68         image_data = "".join(image_data)
69         act_data = f"{image_data};#{image_meta}"     #Store formatted data
70
71     else:
72         act_data = key_D_PRIV(key_A_PUB(data[0].split()))[0]
73
74     if data[1] == meta:
75         return decompress(act_data,data[2],data[3],data[4])
76
77     elif data[1] == "#8":
78         return act_data,data[1],data[2],data[3],data[4]
79
80     elif meta == None:
81         return decompress(act_data,data[2],data[3],data[4]),data[1]
```

The receive message function has changed significantly from the pseudocode algorithm, I have implemented a buffer as data was being lost, ending up in different transmissions due to the lack of a buffer. The program did not know when a message had been fully received, cutting transmissions early, producing errors.

The buffer works by using a terminator position; this is a variable that stores the location of the terminator characters identifying the end of a transmission/message, the buffer will loop through and keep receiving data until the terminator position is in the transmission.

Once the data has been received it is split(formatted into a list), if statements are applied to the data to check if it is an image and check the messages metadata.

The program will call this function and pass in the metadata of the messages it wants to receive; this is errorhandling and a security method ensuring only expected data is received. The data is split up and decrypted using the encryption keys it is then decompressed and returned. If the data is an image the data is split up differently depending on the image format.

## Computer Science NEA project - Chat application

### Compression:

```
135 def dict_encode(data,file,image):
136     dict = {}
137     text = []
138     counter = 0
139
140     if file:
141
142         data = open(data)
143         for line in data:
144             text.append("/n")
145             for x in (line).split():
146                 text.append(x)
147         return dict_encode(" ".join(text),False)
148
149     else:
150
151         if not image:
152             data = data.split()
153             data = list(map(lambda x: x.replace("/n","\n"),data))
154
155         for i in data:
156
157             if not(i in dict.keys()):
158                 dict[i] = counter
159                 data[data.index(i)] = dict.get(i)
160                 counter = counter + 1
161
162             else:
163
164                 data[data.index(i)] = dict.get(i)
165
166         return f"{dict}/{}/{data}"
```

I have programmed the server-side application in a way that foregoes the need for many of the compression functions through the use of function recurrence, this improves the program's time-space complexity. This function is called to compress text and text files, the function will process the data depending on the type of data.

If the data is a file it will be opened and read line by line adding it to a list with a line terminator at the end of each line this will be represented as a string and encoded like a normal piece of text.

If the data is not an image it is split and the line breaks are replaced, a for loop is then used to go through each item stored in data. Each value is added to a separate dictionary variable, the original text index is replaced by a counter value. The counter value represents the character the program will continue to loop through checking if the item/word is represented; if it is not it will be added to the dictionary, each item/word will be represented by a value.

The function returns a string joined by "/",/ this unique set of characters are used to identify a dictionary encoded message and can be used to split the data up that is specific to this function.

## Computer Science NEA project - Chat application

### Declaration of variables and the creation of important connections

After programming the functions listed above I began to program the application following the flowcharts and diagrams created. I began the next section of development declaring essential variables and connections with the database.

```
294 #Key Global Variables -----
295
296 nextm_buffer = ""
297 recipient = ""
298
299 #Constants -----
300
301 SPORT = 8081
302 SIP = "0.0.0.0"
303 BUFF_SIZE = 4096
304
305
306
307 #Database, Server Connection -----
308
309 db = mysql.connector.connect(
310     host="localhost",
311     user="root",
312     password="root",
313     database="main_database"
314 )
315
316 mycursor = db.cursor()
317 #Creating client connection
318 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
319 s.bind((SIP,SPORT))
320 print("Server Started, waiting for client connection...")
```

To allow messages of all sizes to be received I have implemented a buffer, however sometimes more information is received than required. If 10 characters are received and the transmission is missing two, the receive method is called again and 8 characters from the next transmission are received. A global buffer is required to store this data so it can be added to the start of the next transmission/buffer.

The recipient variable improves efficiency, instead of the client application requiring the transmission of the recipient of messages the client only needs to send this data once when that client's message window is opened. A global variable is required so that the recipient variable can be used throughout the entire application.

Constants are represented as capitalised variables as they are a python limitation, using constants ensures data is not changed, otherwise this would create security vulnerabilities and cause undesired outputs. The constants store the server's address and socket that the program will be bound to.

Programming the application this way makes it easy for a developer to locate this information and edit it if required.

I have also created a database connection using "mysql.connector.connect()" . I have binded this connection to "main\_database" ensuring only the correct data is fetched. I have also initialised a database cursor that is used to access,edit and fetch data from the database.

## Computer Science NEA project - Chat application

### Instantiating client connections

```
326 while True:
327
328     s.listen(5)
329     clientsock, clientaddress = s.accept()
330     print(receive_message("#3"))
331     send_message("True", "#3", False, False, None)
332     #Client now conencted Now needs to be authenticated
333     data = receive_message("#2")
334     data = data.split()
335     query = """SELECT username,password FROM main_database.user WHERE UID = %s"""
336     mycursor.execute(query,tuple(map(str, data[1].split())))
337     queryresult = mycursor.fetchall()
338
339     if (queryresult[0][0] + queryresult[0][1]) == int(data[0]): #Check if username nad password right
340         send_message("True", "#2", False, False, None) #User is now logged in and authenticated
341         new_thread = ThreadC(clientsock,clientaddress,data[1])
342         new_thread.start()
```

This is the start of the script, it is used to create new connections. The while loop will run until the program is stopped, it checks for connections and instantiates new objects(threads) using the ThreadC class with the new connection.



## Computer Science NEA project - Chat application

### Authentication

```
328 | s.listen(5)
329 | clientsock, clientaddress = s.accept()
330 | print(receive_message("#3"))
331 | send_message("True", "#3", False, False, None)
332 | #Client now conencted Now needs to be authenticated
333 | data = receive_message("#2")
334 | data = data.split()
335 | query = """SELECT username,password FROM main_database.user WHERE UID = %s"""
336 | mycursor.execute(query,tuple(map(str, data[1].split())))
337 | queryresult = mycursor.fetchall()
338 |
339 | if (queryresult[0][0] + queryresult[0][1]) == int(data[0]): #Check if username nad password right
340 |     send_message("True", "#2", False, False, None) #User is now logged in and authenticated
341 |     new_thread = ThreadC(clientsock,clientaddress,data[1])
342 |     new_thread.start()
```

Upon receiving a connection the program needs to authenticate the connection to ensure that the client is a whitelisted user. Without this the program would not run as intended as messages are stored per user within the database.

This module starts by listening for connections, this is currently limited to 5 simultaneous connections due to server resources and the small deployment of the application; this can be easily increased when the application is deployed to a larger audience.

Once a connection is made the client application will send a confirmation of connection, this message is received using the `receive_message` function, the confirmation should have the metadata "#3". The message is received and the authentication process is started.

The authentication algorithm begins by receiving the hashed username+password and the client's unique ID. This data is split up into separate parts, the database is then queried using this data to authenticate the user. If the data stored within the database matches the values received a True boolean value will be sent represented as a string back to the client, if the data is not correct no message will be sent.

Using the data received a new thread is created using the `ThreadC` class, this thread is then started allowing multiple users at once.

## Computer Science NEA project - Chat application

### ThreadC class

```
189 class ThreadC(threading.Thread):
190     def __init__(self, clientsocket, clientaddress, clientID):
191         threading.Thread.__init__(self)
192         self.client_socket = clientsocket
193         self.client_address = clientaddress
194         self.UID = clientID
195         print(f"{self.UID} has connected!")
```

The class is used to create objects (instances of the ThreadC class) the class will create cpu threads enabling multiple simultaneous users. I have programmed the application in this way as less code is required, each client can use the same class and there is no forward developer planning involved, this class allows multiple users/clients.

The class is completely isolated and private to the rest of the script, each method is private and can only be ran inside the script but not accessed or ran outside the class; this is the same for the object's attributes too. Calling methods of the class outside the class would create security vulnerabilities and cause undesired outputs.

I have created the class based on the design created using the UML class diagram created in the design phase.

## Computer Science NEA project - Chat application

### Methods

#### Run Method

```
199     def run(self):
200         global queryresult,db,recipient,address_book
201         print("running")
202         while True:
203             address_book = []
204             data = receive_message(None)
205             if data[1] == "#1":                #Update address
206                 self.update()
207
208             elif data[1] == "#1A":            #Update messages
209                 self.update_messages(data)
210
211             elif data[1] == "#10":            #Report
212                 self.report(data)
213
214             elif data[1] == "#8":            #Forward message
215                 self.forward_message(data)
216
217             elif data[1] == "#11":            #Exit
218                 clientsock.close()
219                 break
```

This method is run when a new thread is created. A forever loop is used to receive data and process it based on its metadata. This is done using an elif statement calling the correct methods for that metadata.

The elif statement follows the metadata table created in the development section and processes the information correspondingly.

#### Update

```
224     def update(self):
225         global address_book
226         mycursor.execute("""SELECT UID FROM main_database.user""")
227         queryresult = (list(filter(lambda x: x[0] != self.UID, mycursor.fetchall())))
228
229         for i in queryresult:
230             address_book.append(i[0])
231
232         send_message(dict_encode(" ".join(address_book),False,False),"#1",False,False,"lossless")
```

This method is called when the client requests an address book update. The update method works by querying the database and fetching the UIDs stored within the database. This is then formatted and looped through adding each UID to an address book, this address book is then sent with the corresponding metadata and formatting.

## Computer Science NEA project - Chat application

### Update\_message

```
236 def update_messages(self,data):
237     global queryresult,recipient,messages
238
239     query = """SELECT messages FROM main_database.messages WHERE UID = %s AND recipient = %s"""
240     mycursor.execute(query,(self.UID,data[0][0]))
241     queryresult = mycursor.fetchall()
242     recipient = data[0][0]
243
244     if queryresult[0][0] != None:
245
246         print(queryresult)
247         messages = queryresult[0][0].split("#,#,#")
248         send_message(str(len(messages)),"#1A",False,False,None)           #Allow client to receive all messages
249
250         for i in messages:                                               #Looping through inbox for recipient
251             message = i.split("#*#")                                     #split messages into parts
252             print(message)
253             print(message[0],message[1],message[3],message[4],message[2])
254             send_message(message[0],message[1],message[3],message[4],message[2]) #Send message as received
255
256     else:
257         send_message(" ", "#8",None,False,False)
258         queryresult = [(None,)]
```

This method is responsible for fetching the messages stored in the clients inbox from the database and sending them to the client. The method achieves this by first querying the database using the UID of the client and the recipient, this query returns the conversation between the client and recipient specified. Messages are stored in a particular format so have to be reformatted after being fetched.

Once the messages have been fetched they are reformatted and the recipient variable is updated to the recipient received. A recipient variable avoids the need to send the recipient with every message sent, speeding up the client's application and decreasing the load on the server. Once the data is fetched it is checked to see if any data has been fetched if not an empty string will be sent and the queryresult variable updated to None.

If the query result contains messages it is split up and reformatted into a list, the length of this list is sent to the client so the client can use a for loop to receive all of the messages being sent. A for loop is then used to send each message using the send\_message function passing in a different part of the message as received before being stored in the database.

Updating a message this way is much more efficient as less code is required to check the type of message. The message will be forwarded and no extra data will be added improving the big O of the program.

## Computer Science NEA project - Chat application

### Forward\_message

```
262 | def forward_message(self,data):
263 |     global queryresult
264 |     global recipient
265 |     comp_data = []
266 |     for i in data:
267 |         comp_data.append(i)
268 |
269 |     comp_data = "###".join(comp_data)
270 |     query = "UPDATE main_database.messages SET messages = %s WHERE (UID = %s and recipient = %s)"
271 |
272 |     if queryresult[0][0] == None:
273 |         queryresult = comp_data
274 |
275 |     else:
276 |         queryresult = (queryresult[0][0]+"###"+comp_data)
277 |
278 |     mycursor.execute(query,(queryresult,self.UID,recipient))
279 |     query = "UPDATE main_database.messages SET messages = %s WHERE (UID = %s and recipient = %s)"
280 |     mycursor.execute(query,(queryresult,recipient,self.UID))
281 |     db.commit()
```

This method is used to store messages being sent, the method will reformat the message and store it in the conversation table between the client and recipient.

To achieve this the method will loop through the data received adding it to the comp\_data list using the append function, this will then be made into a string. Once data has been formatted an if statement will be used to check if the query result from the last query (which will always be the updating of messages) is equal to None. If the last query result is equal to None there is no data stored within the clients inbox therefore, the data just received is going to be the first data, not query result+comp\_data therefore, queryresult is changed to equal the value stored in comp\_data.

If the query result is not equal to None (there is data stored in the inbox) this will need to be added before the new data is added ensuring no data is lost this is done by joining the two variables using "###".

The data stored in the query result is combined with the object's(clients) UID and recipient receiving the message that is taken in by the query and the database is updated with the information.

### Report

```
285 | def report(self,data):
286 |     reports = open("report.txt", "a")
287 |     reports.write(f"{data[0][0]}, recipient:{data[0][1]},    Author:{self.UID}\n")
288 |     reports.close()
```

The report method is used to save reports to a text file on the server, the method formats the report correctly and writes it to the file in a readable format.

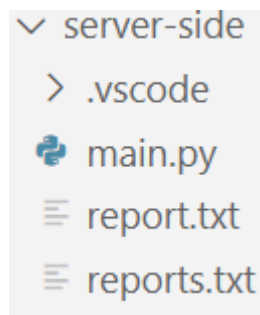
```
server-side > report.txt
1 Copied my homework, recipient:Harry@Taylor456,    Author:Brooklyn@Limberty127
```

The method opens the document first using the append mode so that data is only added to the end and no data is overwritten. The write command is used and a formatted string is written the file is then closed.

## Computer Science NEA project - Chat application

### Server-side script

The server Side script is now fully programmed and all modules above are joined together, working as designed. The server-side script only requires 11 Kilobytes of storage allowing me to use the script on almost any sized server, below I have listed the other files that are part of the server-side folder.



[Full Server Application](#)

## Computer Science NEA project - Chat application

### Client-Side

I have also begun development on this script by writing the functions designed in the design section. Many of the functions differ from those designed and written using pseudocode.

The client-side and server-side applications share some of the same functions such as the caesar cipher function, therefore some functions are not explained thoroughly. This information can be found in the server-side script explanation.

### Functions

#### Encryption:

```
7   #Encryption Functions -----
8
9   def caesur(plaintext,shift):
10      encrypted = []
11
12      while True:
13          encrypted = []
14
15          for i in plaintext:
16              encrypted.append(chr(ord(chr(ord(i))+shift))+shift))
17
18          encrypted = "".join(encrypted)
19
20      return encrypted
21
22  def key_B_PRIV(data):
23      data = (list(map(lambda x: caesur(x,-5),data)))
24      return data
25
26
27
28  def key_C_PUB(data):
29      data = (list(map(lambda x: caesur(x,2),data)))
30      return data

```

```
200      data = (key_B_PRIV(key_C_PUB(data[0])))
```

These functions are very similar to the server-side equivalents. The key pair is different due to asymmetric encryption however the only difference between these keys is the different shift values used.

The key functions work by looping through the input that is passed in by value. This should be formatted as an array, it is then looped through using a map and lambda function. Each value in the list will be caesar ciphered. The value of the caesar cipher array will be stored in the data variable and returned.

To encrypt and decrypt data these keys are combined this method is known as asymmetric encryption.

## Computer Science NEA project - Chat application

### Hashing:

```
34 def hash(value):
35     hash_val = 0
36     hashed = []
37
38     for i in value[::-1]:
39         hashed.append((ord(i)) + 10)
40
41     for i in hashed:
42         hash_val = i + hash_val-66
43
44     return hash_val
```

This function is a duplicate of the server-side version; it hashes the value, in the same way, to ensure that the result can be duplicated on the server side; this is essential for database querying.



## Computer Science NEA project - Chat application

### Receive message:

```
50 def receive_message(meta):
51     global nextm_buffer
52     buffer = nextm_buffer           #Add data received to buffer
53     data = ""
54     terminator_pos = -1             #Is a terminator in the buffer
55     print(terminator_pos)
56
57     while terminator_pos == -1:      #If equal to -1 ///n must exist
58
59         buffer = buffer + (s.recv(1024).decode())
60         terminator_pos = buffer.find("///n")
61         if terminator_pos != -1:
62             nextm_buffer = buffer[terminator_pos + 4:]
63         data = buffer[:terminator_pos]
64         buffer = buffer[terminator_pos + 1:]
65
66     data = data.split("##,##")      #Split message into parts
67
68     if data[-2] == "True" or data[-2] == True:    #Check if image received
69
70         image_info = data[0].split(";#;")         #Split image section
71         image_meta = image_info[1]                #Store corresponding image data
72         image_data = image_info[0]                #Store corresponding image data
73         image_data = ast.literal_eval(image_data) #Convert Data into correct form
74         image_meta = ast.literal_eval(image_meta) #Convert Data into correct form
75         image_data = key_B_PRIV(key_C_PUB(image_data)) #Decrypt
76         image_data = "".join(image_data)
77         act_data = image_data, image_meta
78
79     else:
80
81         act_data = key_B_PRIV(key_C_PUB(data[0].split()))[0]
82
83     print(act_data, "Actual Data")
84
85     if data[1] == meta:
86
87         return decompress(act_data, data[2], data[3], data[4])
88
89     elif meta == None:
90
91         return decompress(act_data, data[2], data[3], data[4])
```

The receive message function is identical to the server-side version however this function decrypts data using the combined decryption keys B and C. The function has different if statements as it is looking for different metadata.

based on the branch chosen, decided by the if statement. The function will receive the message through the buffer and split it into parts processing this data

The data is processed the same as the server-side version, however the function uses an elif statement to check if the data has a meta value (has a specific purpose). If the metadata is not equal to the parameter then the metadata parameter is checked to see if it is equal to None, if it is the data received will be returned.

## Computer Science NEA project - Chat application

### Send message:

```
def send_message(data, meta, image, file, comp_technique):  
    if image:  
        imagedata = data[1]  
        data = (key_B_PRIV(key_C_PUB(data[0])))  
        message = (f"{data};#{imagedata}###{meta}###{comp_technique}###{image}###{file}///n").encode()  
        s.sendto(message, (SIP, SPORT))  
  
    else:  
        data = "##,##".join((key_B_PRIV(key_C_PUB(data.split("##,##")))))  
        s.sendto((f"{data}###{meta}###{comp_technique}###{image}###{file}///n").encode(), (SIP, SPORT))
```

The send message function works similarly to the server-side equivalent, however data is sent to different addresses and joined together using different unique characters: "##,##".

The function begins by using an if statement to determine the type of data being sent and the formatting required. If the data is an image it will be split up into key parts, encrypted and sent joined with the unique characters, there is a different set of characters separating data and image data as the server-side application can split the message like normal messages, this is more efficient as less memory is required for the extra code that would be required without this technique.

If the data passed into the function is not an image it is encrypted and then sent to the address stored as constants SIP and SPORT after being formatted and joined to get however the unique characters "##,##".

## Computer Science NEA project - Chat application

### Decompression:

```

97  def decompress(data, technique, image, file):
98      if image == True or image=="True":          #Images always RLE
99
100         decoded = dict_decode(data[0], False)
101         metadata = data[1]
102         print(metadata)
103         return RLD(decoded, metadata)
104
105
106     if technique != None and technique != "None": #Text always dict_encoded
107
108         print(technique)
109         return dict_decode(data, file)
110
111     else:
112         return data
```

The decompression function is used to decompress messages, data is formatted and decompressed based on data type and metadata.

The decompression function makes use of if statements to choose the branch the program should take. The first if statement determines if the data is an image, if the data is an image it will be split and passed into the dictioanry\_decode function to be decoded.

The data that hasn't been encoded is not passed into the dict\_decode function but passed into the Run Length Decode function with the decoded data to decode the message and store the image. Regardless of the type of image compression used ,all image compression algorithms have been designed to always be encoded with Run Length encoding after compression. This technique saves memory as only one function is required to decode images regardless of compression technique.

The second if statement function makes sure data is encoded. The program now knows that the data is text, if the data has not been encoded it is returned as received. If the data has been compressed it is decoded using the dict\_deocde function and the file parameter, the output of this is returned.

## Computer Science NEA project - Chat application

### Run Length Decoding:

```
116 def RLD(rgbs,metadata):
117
118     counter = 0
119     print(metadata[1],metadata[0])
120     new = Image.new(metadata[1],metadata[0],"white") #metadata[1] = im.mode and metadata[2] = im.size
121
122     for row in range(new.size[0]):           #Loop Column
123         for col in range(new.size[1]):       #Loop Row
124
125             new.putpixel((row,col),(rgbs[counter]))
126             counter = counter + 1
127
128     new.save("Ctemp.png")
129     new.show()
```

The run length decoding function takes in the rgb values of each pixel(rgbs) and the images metadata(image size and image mode).

The function creates a new blank white image to the same dimensions as listed in the image metadata, the image is then traversed using two for loops representing the image graphically. Each pixel is edited and the rgb value of the original image is applied to the new blank image. Once the "rgbs" values have been looped through and applied the new image is saved in the file location 'temp.png'.

The method of run length decoding is very efficient for different image sizes and modes.

### Dictionary Decoding:

```
133 def dict_decode(data,file):
134     if file == True or file == "True":
135
136         text = dict_decode(data,False)
137         print(text)
138         text = list(map(lambda x: x+" ",text))
139         print(text)
140         f = open("temp1.txt","w")
141         f.writelines(text)
142         f.close()
143
144     else:
145
146         data = data.split("/,/")
147         dict = ast.literal_eval(data[0])
148         text = ast.literal_eval(data[1])
149
150         for i in text:
151             for x in list(dict.keys()):
152
153                 if i == dict[x]:
154                     text[text.index(i)] = x
155
156         return text
```

This function is a duplicate of the server-side version.

## Computer Science NEA project - Chat application

### Dictionary Encoding:

```
158 def dict_encode(data,file,image):
159     dict = {}
160     text = []
161     counter = 0
162
163     if file:
164         data = open(data)
165         for line in data:
166             text.append("/n")
167             for x in (line).split():
168                 text.append(x)
169         return dict_encode(" ".join(text),False,False)
170
171     else:
172         if not image:
173             data = data.split()
174             data = list(map(lambda x: x.replace("/n","\n"),data))
175
176         for i in data:
177
178             if not(i in dict.keys()):
179                 dict[i] = counter
180                 data[data.index(i)] = dict.get(i)
181                 counter = counter + 1
182
183             else:
184                 data[data.index(i)] = dict.get(i)
185
186         return f"{dict}/{data}"
```

This function is identical to the server-side version.

## Computer Science NEA project - Chat application

### Compression:

```
204 def compress(data, technique, image, file):
205
206     if technique == "lossless":
207         if image:
208             return RLE(data)
209         else:
210             return dict_encode(data, file, image)
211
212     elif technique == "lossy":
213
214         if image:
215             return loss_comp(data)
216
217         else:
218             return dict_encode(data, file, image)
```

Like the decompression function this function acts as a mediator between the different compression methods encrypting data based on the technique specified by parameter. This function works by employing if statements to apply different compression techniques based on the technique passed into the function.

### Run Length Encode:

```
222 def RLE(image_path):
223
224     rgbs = []
225     im = Image.open(image_path)
226
227     for row in range(im.size[0]):
228         for col in range(im.size[1]):
229
230             rgbs.append((im.getpixel((row, col))))
231
232     return dict_encode(rgbs, False, True), [im.size, im.mode]
```

This function is used to compress images, representing them as a list of RGB values; this list is then compressed with the dict\_encode function compressing the image further, decreasing transmission time.

The function takes the file path of the image as a parameter, this is then opened using the Image library method "Image.open". Two for loops are used to create an axis/grid to represent the image. they are made to loop through the value of the image's dimensions. As both loops run the pixel in that position of the image is stored and added to the rgbs list. Once all RGB values have been stored in the rgbs variable they are dictionary decoded and returned.

## Computer Science NEA project - Chat application

### Lossy Compression:

```
252 def loss_comp(image_path):
253     im = Image.open(image_path)
254     rgbs = []
255
256     for row in range(im.size[0]):
257         for col in range(im.size[1]):
258             rgbs.append(im.getpixel((row-1,col)))
259
260
261     return dict_encode(rgbs,False,True),[im.size,im.mode]
```

This function is very similar to the run length encoding function, however this function will reduce the quality of the image. The function chooses the last pixel in the row for each pixel lupus through. The rgbs value of each pixel in the positions chosen are stored in the rgbs value list.

### Login:

```
268 def login(username,password):
269     send_message((f"{hash(username)+hash(password)} {UID}"),"#2",False,False,None)
270     return receive_message("#2")
```

This function is called when the user attempts to log-in it is used to authenticate a user. The function will hash the username and password passed into the function and send these added together with the UID of the client. The server will process the request and return a boolean value this function will return the value that is received.

### Connect:

```
286 def connect():
287     try:
288         s.connect((SIP,8081))
289         s.sendall(("None##,###3##,##None##,##False##,##False##,##None##,###//n").encode())
290         return (receive_message("#3"))
291     except:
292         pass
```

This function is called to create a connection with the server, the function will connect to the server using its static IP and port, the function will then send the adequate metadata.

If the server has received the metadata sent it will respond and this data is returned by calling the receive\_message function. If an error is raised by the connect method the except statement will handle it and end the function allowing the user to re enter their details.

## Computer Science NEA project - Chat application

### Script

The rest of the script is heavily user interface centred calling functions based on interface inputs.

#### Constants:

```
425     #Constants -----
426
427
428
429     SIP = "127.0.0.1"
430     SPORT = 8081
431     UID = "Brooklyn@Limbert127"
```

I have used constants for this data so the data is easily changed, these variables can not be tampered with. If changed these variables can cause a security vulnerability as key information can be sent to an unintended address.

To bypass SQL injection attacks I have used a constant for the UID as this will be the data that is used to query the database. A constant will remove the possibility of an SQL attack as the user's input is not used to query the database. If these constants change it could crash the server and application therefore constants are most applicable.

#### Key Variables:

```
439     nextm_buffer = "" #Store data after terminator
440     comp_technique = "lossy"
441     address_book = ["Jude@Easton342", "Nat@Gray111"]
442     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
443     messages = {
444         "Jude@Easton342": ["Please can I have some help with this work", "temp.png", "Yes!"],
445         "Nat@Gray111": ["When is the HWK due for maths?", "03/11/22 I think"]
446     }
```

These are the application's key variables, used throughout the application; without these the program could not function as designed.

The buffer variable will store the data received after the terminator is received ensuring no data is lost after receiving it. The comp\_technique variable is used to store the compression method and to decide the compression method to be used. The address\_book list will store the address book that is used to update the graphical user interface. The messages dictionary stored the messages for each user in the address book.

The socket is bound to the variable s and is used to send and receive data.



## Computer Science NEA project - Chat application

### Graphical User Interface

```
2 from guizero import App, Text, TextBox, PushButton, Window, Box, ListBox, Combo
```

To implement a graphical user interface I have used the GUIZERO library implementing many of its features into my application. The application will perform actions when a button in the GUI is pressed this will call a function and the input will be processed.

I have created the interface using multiple windows connected to the main App window, programming it this way was easier and more efficient as widgets did not need to be destroyed and remade every time a new interface is opened. This improves the space time complexity of the application.

#### App Creation and Log-In window:

```
454 app = App(title="Chat Application")
455 app.when_closed = exit
456 heading = Text(app, text="Log-in")
457 username = TextBox(app)
458 password = TextBox(app, hide_text=True)
459 login_button = PushButton(app,command=logincommand, text="Log-in")
460 app.display()

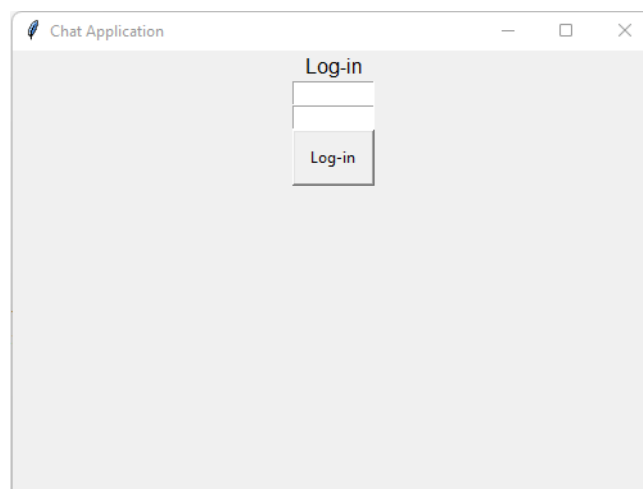
377 def exit():
378     try:
379         send_message((f"None "),"#11",False,False,None)
380     except:
381         pass
382     app.destroy()
```

Above is the code used to create the main application window. All windows will branch from this window making it easier to manage the GUI as only the main window needs to be closed to stop the entire application running.

The app has the title "chat application", when it is closed the exit function is ran to close the socket on the server side this is done by sending the meta data "#11".

The Text, TextBox and PushButton methods are used to create the basic log-in page, the login-button has the command logincommand. This command will authenticate the inputs in the two textboxes and create the main application window.

After all commands have been run the app.display method is called to display the app.



## Computer Science NEA project - Chat application

```
312 def lbuttoncommand():
313     global mainwin,messages
314
315     if connect() == "True":
316         if login(username.value,password.value) == "True":
317             global address_list_gui,address_book,messages_gui
318             global messages,message
319             heading.value = "Authenticated"
320
321             #GUI main window design
322
323             mainwin = Window(app,title = "Chat Application")
324
325             address_box = Box(mainwin,align="left",height = "fill")
326             address_list_gui = ListBox(address_box, items=address_book,command = updatemessages)
327
328             buttons_box = Box(address_box, width="fill",height ="150")
329             PushButton(buttons_box, text="Settings", width="fill", align= "top", command = settings)
330             PushButton(buttons_box, text="Report",width="fill", align="bottom", command=report)
331             PushButton(buttons_box, text="File",width="fill", align="bottom",command = uploadfile)
332             message_box = Box(mainwin, width="fill", align="bottom",height="30", border=True)
333             PushButton(message_box, text="Send", align="right",command=send_butcommand)
334             message = TextBox(message_box,width = "fill", height="fill", align="left")
335
336             recipient_box = Box(mainwin, width="fill", height = "150", align="top")
337             recipient = Text(recipient_box,text="Recipient:",align="left")
338
339             messages_box = Box(mainwin,width = "fill", height="150",align="top")
340             messages_gui = ListBox(messages_box,items = messages,width = "fill",height="fill",scrollbar=True)
341
342             #Updating address book
343
344             send_message((f"None "),"#1",False,False,None)
345             address_book = receive_message("#1")
346             address_list_gui.clear()
347
348             for i in address_book:
349
350                 address_list_gui.append(i)
351                 messages[i] = []
```

### Log-In button:

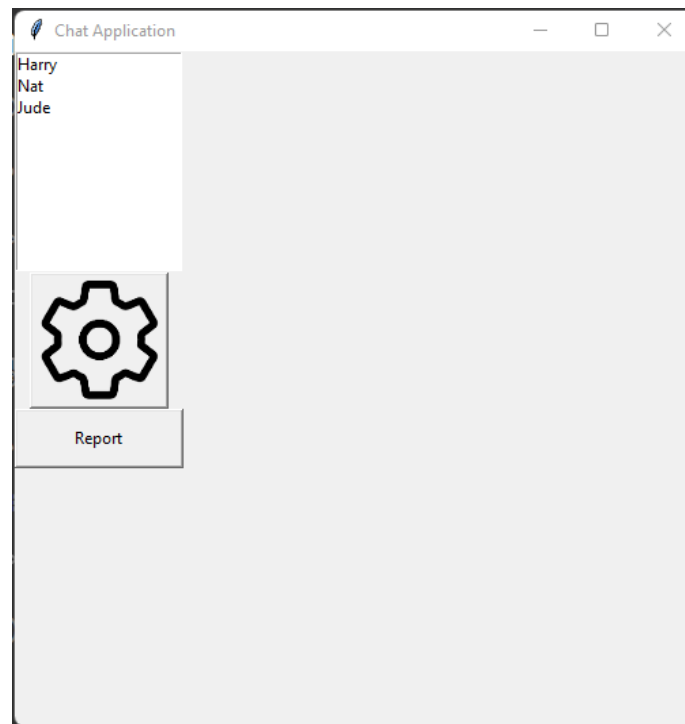
This function is called when the log-in button is pressed, the function will connect and authenticate the user the main application will then be loaded and displayed.

The function uses an if statement to branch the application based on the value returned from the connect function, if the value is true then the program will attempt to authenticate the user. If the value is not true then the function will cease to run and the user will have to try to log-in again.

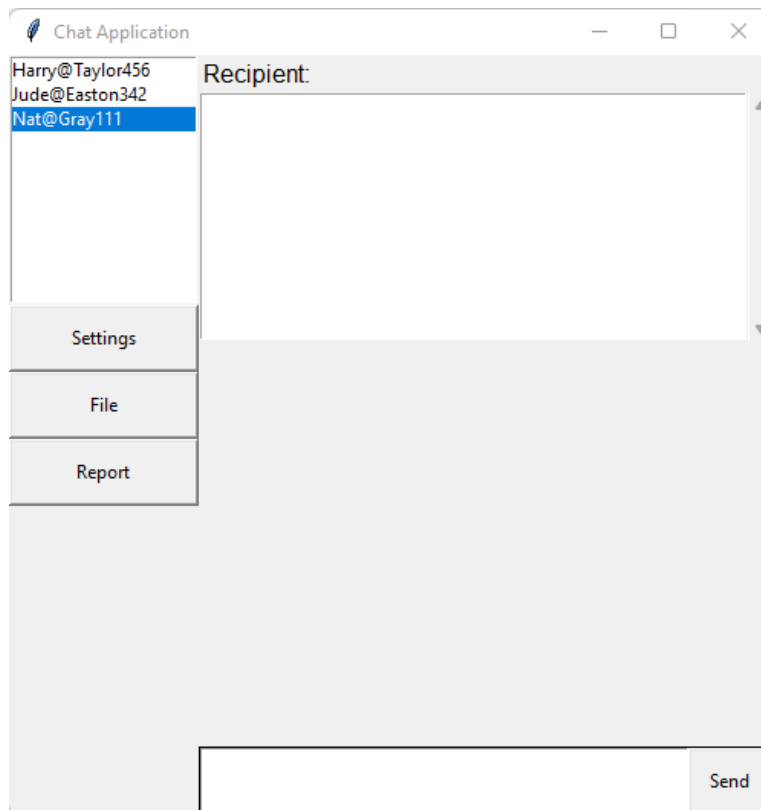
Once the user has been authenticated the main window will be created and built using guizero methods.

Once the window has been created the address book will be updated, a message with the update metadata will be sent and the response stored in the address book variable , the gui will then be cleared and a for loop will be used to add the received data (Users). An item and key will be created for each address used.

## Computer Science NEA project - Chat application



Due to Guizeros limitations, I was not able to implement the symbols as designed in the design phase. If I was deploying the application to a larger audience I would implement a different library to allow the implementation of symbols. The implementation of a different library is not possible at the moment due to time constraints and it not being a necessity for my client and target audience.



The graphical user interface has been designed to follow the designs made in the last section however due to the limitations as described I have not been able to implement symbols instead I have used labelled buttons.

## Computer Science NEA project - Chat application

### Address Selection:

```
379 def updatemessages():
380
381     global messages,address_book,recipient
382     recipient = address_list_gui.value
383     messages_gui.clear()
384     #Updating messages corresponding with server and updating item list
385     send_message(compress(recipient,comp_technique,False,False),"#1A",False,False,comp_technique)
386     message_number = receive_message("#1A")
387
388     if message_number != None: #error handling, what if no messages to be sent??
389
390         for i in range(int(message_number)):
391             message = receive_message(None) #Message data received
392             message_history = messages[recipient]
393             message_history.append(message)
394             messages[recipient] = message_history
395             messages_gui.append(message)
```

When a user is selected from the address book the user's conversation will be displayed in the message text box. This is done via the update message function that is called when a user is selected in the Listbox (address book).

The function will store the user-selected recipient and clear the messages in the text box a message will then be sent containing the compressed recipient, compression technique and the metadata "#1A". The server will process the message and query the database using the UID and recipient, the message number received from the server will be returned by the receive\_message function.

If the query returns no data (no data has been sent to the recipient before) None will be returned and displayed the function will then end.

If data is received then the message number received will not be None and the function will continue to process the data, a for loop will run with the integer version of the message\_Number received.

The for loop will run a receive function call every loop receiving a message one at a time this message will be stored in the message\_history list and added to the GUI textbox.

## Computer Science NEA project - Chat application

### Send message button:

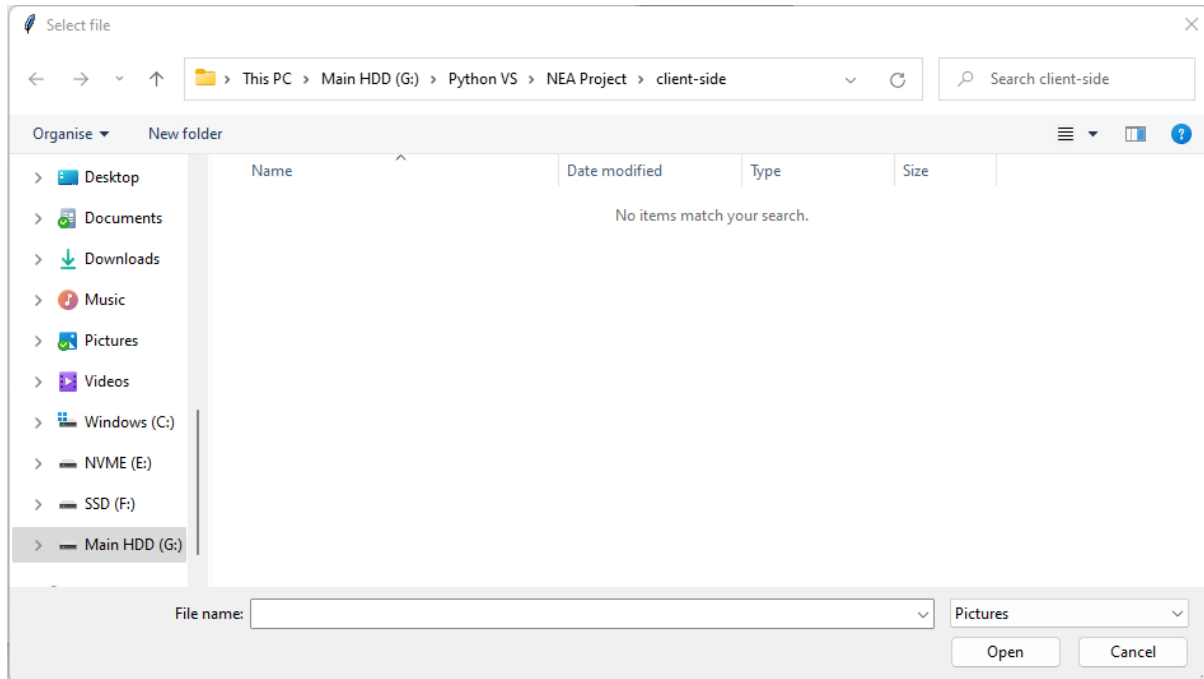
```
399 def send_butcommand():
400
401     global address_book
402     global messages
403     send_message(compress(message.value, comp_technique, False, False), "#8", False, False, comp_technique)
404     messages_gui.append(message.value)
405     message_history = messages[recipient]
406     message_history.append(message.value)
407     messages[recipient] = message_history
408     message.clear()
```

This function will be called when the send button is pressed, the value in the send textbox will be sent to the server and the corresponding variables updated with the new message, the message box will then be cleared.

To achieve this the function begins by sending the value of the message box compressed with the correct metadata. Once the message has been sent it is added to the GUI message text box and the message history is updated along with the messages dictionary for the particular recipient it was sent to. Once the message has been processed the message box will be cleared and the function will end.

## Computer Science NEA project - Chat application

### File selection:



```
418 def uploadfile():
419
420     filepath = app.select_file(filetypes=[["Pictures", "*.png"], ["Pictures", "*.jpeg"], ["Text documents", "*.txt"]])
421     if filepath.find("*.txt") < 0:
422         img = Image.open(filepath)
423         img = img.save('Ctemp.png')
424         filepath = 'Ctemp.png'
425         send_message(compress(filepath, comp_technique, True, False), "#8", True, False, comp_technique)
426     else:
427         send_message(compress(filepath, comp_technique, False, True), "#8", False, True, comp_technique)
```

Upon clicking the file button a file selection window will be opened allowing the user to select a file from their computer as long as it is of the correct format, txt, jpeg or png, these as specified by the select\_file method. The select\_file method is called in this function. This function is called when the user selects the file button.

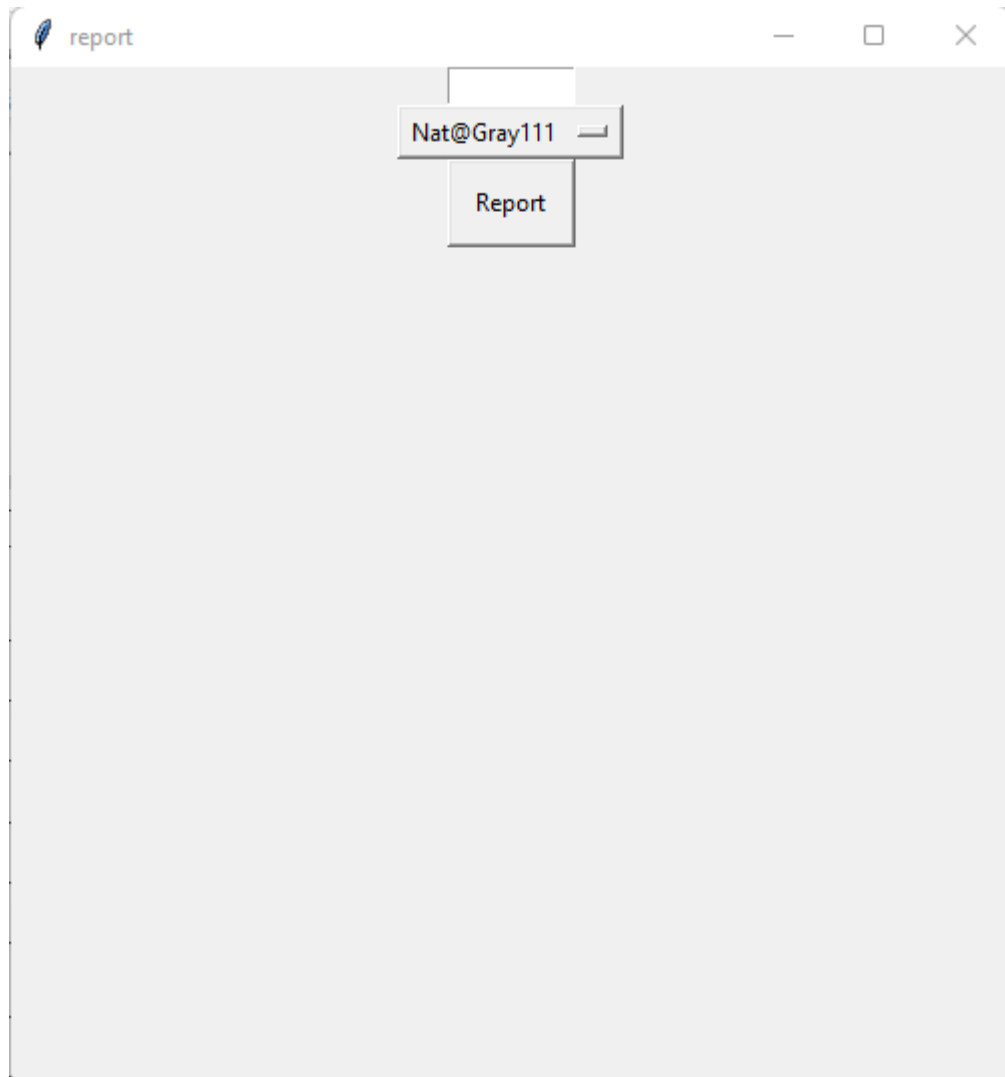
The file path is stored and an if statement is used to determine how to process the file based on its data type.

If the file is a text document the find method will be less than 0, if this is true then a message will be sent with the compressed file and its corresponding metadata.

If the file isn't a text document but an image it will be opened and saved to "Ctemp.png" the file path will be changed to this and a message will be sent with the compressed image and corresponding metadata.

## Computer Science NEA project - Chat application

Report button:



```
296 def report():
297     global username_recip, reportt
298     reportwin = Window(app, title="report")
299     reportt = TextBox(reportwin)
300     username_recip = Combo(reportwin, options = address_book)
301     PushButton(reportwin, command= rbutton, text="Report")
```

The report function will be called when the report button is pressed and this window is displayed, the window contains a drop-down menu and text box to select whom to place a report on and what the report is.

When the report button in this window is pressed, the function below will be called.

## Computer Science NEA project - Chat application

```
305 def rbutton():
306     send_message(compress(f"{reportt.value} {username_recip.value}", comp_technique, False, False), "#10", False, False, comp_technique)
307     mainwin.focus()
```

This function will send the value of the dropdown box and textbox to the server with the corresponding metadata it will then refocus the main window. By default the dropdown menu value will be the first user in the address book so a report can not be made on no one as this would cause errors.

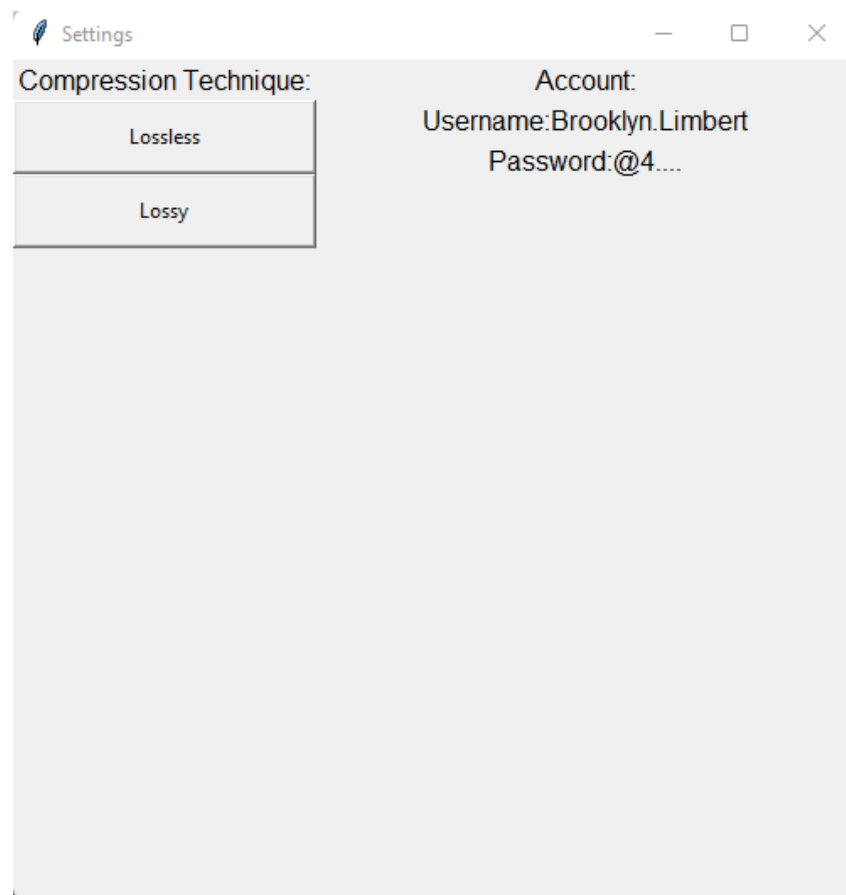
### Settings button:

```
360 def settings():
361
362     global password, username
363     settingswin = Window(app, title="Settings")
364     settingsbox = Box(settingswin, align = "left", height = "fill")
365     Text(settingsbox, text = "Compression Technique:")
366     PushButton(settingsbox, width="fill", text = "Lossless", command = lossless)
367     PushButton(settingsbox, width="fill", text = "Lossy", command = lossy)
368
369
370     compbox = Box(settingsbox, align = "right", height = "fill")
371     Text(settingswin, text = "Account:", align = "top")
372     Text(settingswin, text = f"Username:{username.value}")
373     Text(settingswin, text = f"Password:{password.value[0:2]}....")
```

This function is called when the settings button is pressed it will create the settings window. There are only two buttons in the settings window, they are used to choose a compression technique.



## Computer Science NEA project - Chat application



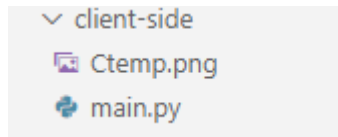
After the settings window has loaded it looks like this, on the right the user's details are displayed with the compression techniques to choose from on the left of the window.

```
352 def lossless():
353     global comp_technique
354     comp_technique = "lossless"
355
356 def lossy():
357     global comp_technique
358     comp_technique = "lossy"
```

These are the functions that are called depending on what button is pressed, the function will change the global variable `comp_technique` to the compression technique chosen.

## Computer Science NEA project - Chat application

### Full Script



### [Full Script](#)

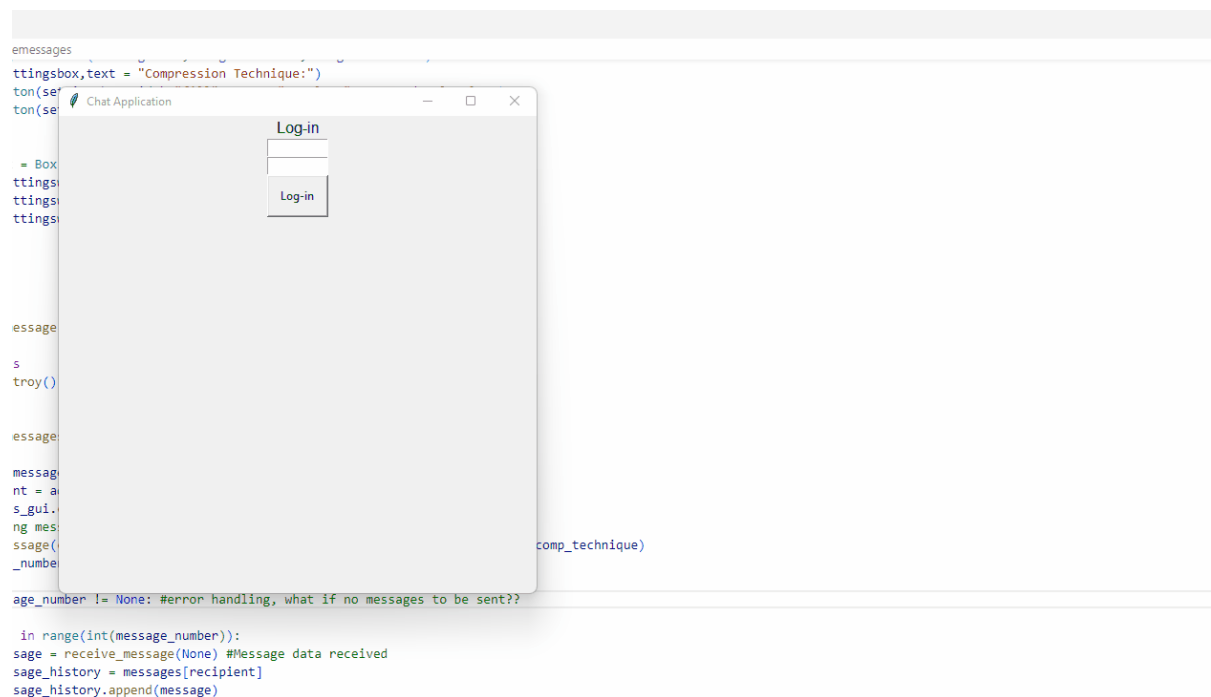
I have uploaded the full client-side script, it can be accessed via the link above.

### Script Testing

To test the application I will be using the trace table designed in the design phase of the project, only the client side will be tested as this will also test the server.

If a feature does not work on the client side then the server side will be at fault as the client relies on the server side to process requests.

Below is a video showing the GUI and the most important features being tested:



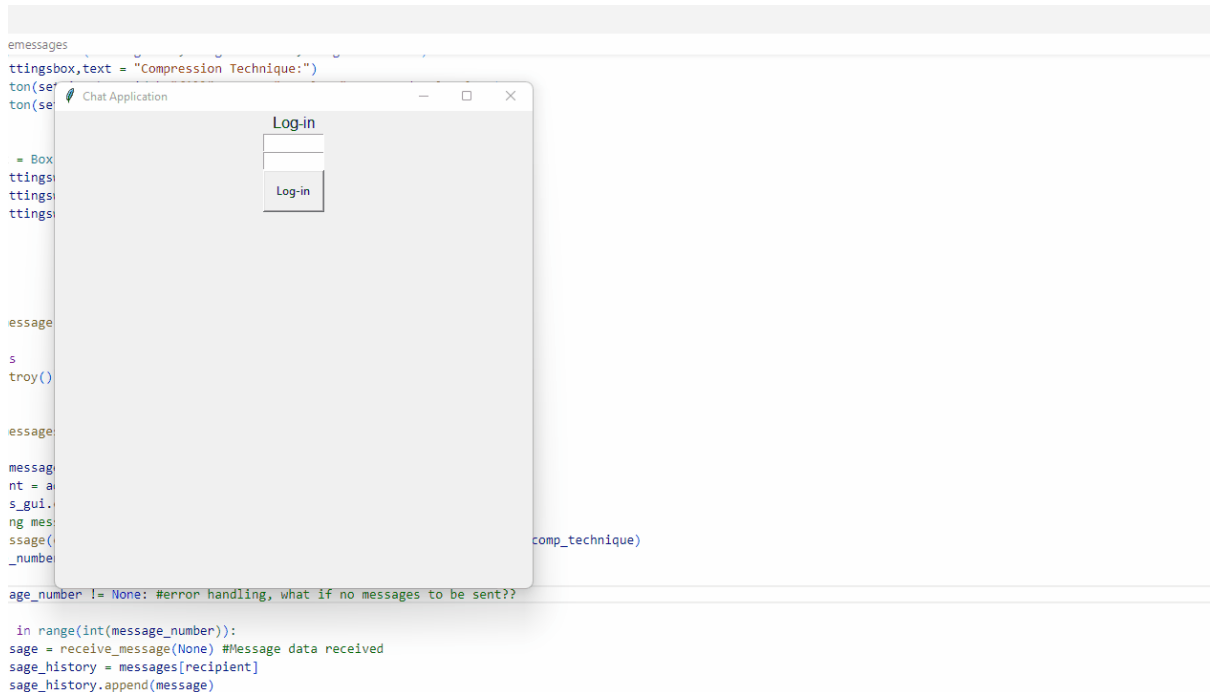
## Computer Science NEA project - Chat application

### Success criteria met?

I am now going to go through the success criteria ensuring that it is met. I will evidence each part of the success criteria tests to show the program has met this area of the criteria.

- **Connection with the server can be maintained**






Below is a video showing requests being sent and processed without error at random for a prolonged time this shows that a server connection can be maintained.



## Computer Science NEA project - Chat application

- **Messages including files can be sent and received regardless of size**

To test this I will be using the test data below to test the application and record its response, the test data has been chosen to vary in data type and size.

 Animal Farm CH1	15/12/2022 12:16	Text Document	15 KB
 Homework.txt	15/12/2022 12:14	Text Document	1 KB
 Original3	18/02/2022 12:58	PNG File	22 KB
 Original1	20/09/2019 21:27	JPG File	6 KB
 Original2	17/01/2021 14:35	JPG File	7 KB

All Files above were successfully formatted and sent to the server; they were then received upon message refreshing and stored in the temporary client file locations where they could be moved to permanent storage.

The file sizes were the same as lossless compression was used to conduct this test.

After conducting the test using lossless compression I used lossy compression to send the test data, due to the compression technique used the file transfer was much faster when it came to the images. The text documents did not vary in size after being received as lossless compression is enacted upon them no matter the compression technique chosen by the user as lossy compression would corrupt the file.

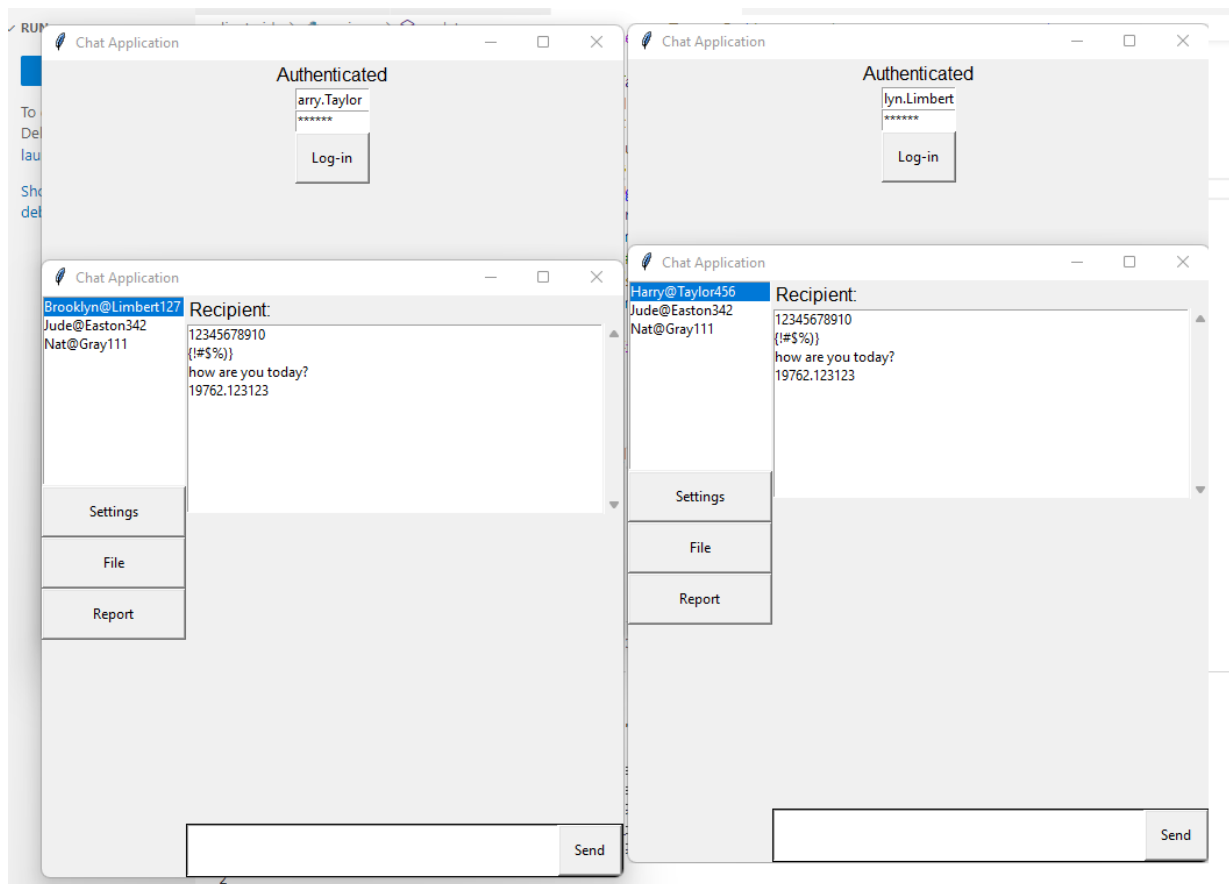
The results of this test show that regardless of file size it can be sent and received. The lossless compression technique however is not very fast however this can be expected for lossless compression using run length encoding on large images, a problem that my client and target audience do not care about. Upon larger application deployment I would create a separate library for the lossless compression technique using the programming language C as this is a more efficient language.

## Computer Science NEA project - Chat application

- Users can send messages via usernames/accounts

Test data	Data type	Reason
12345678910	Integer	To test how the program handles large integers.
!#\$%)	String	Test the program's formatting, how will it handle complex characters.
True	Boolean	Will a boolean value interfere with database querying.
*	Character	Will a special single character be able to be sent?
How are you today?	String	A typical message.
19762.123123	Floating point	Unique message will a floating point value be able to be stored and retrieved from the database.

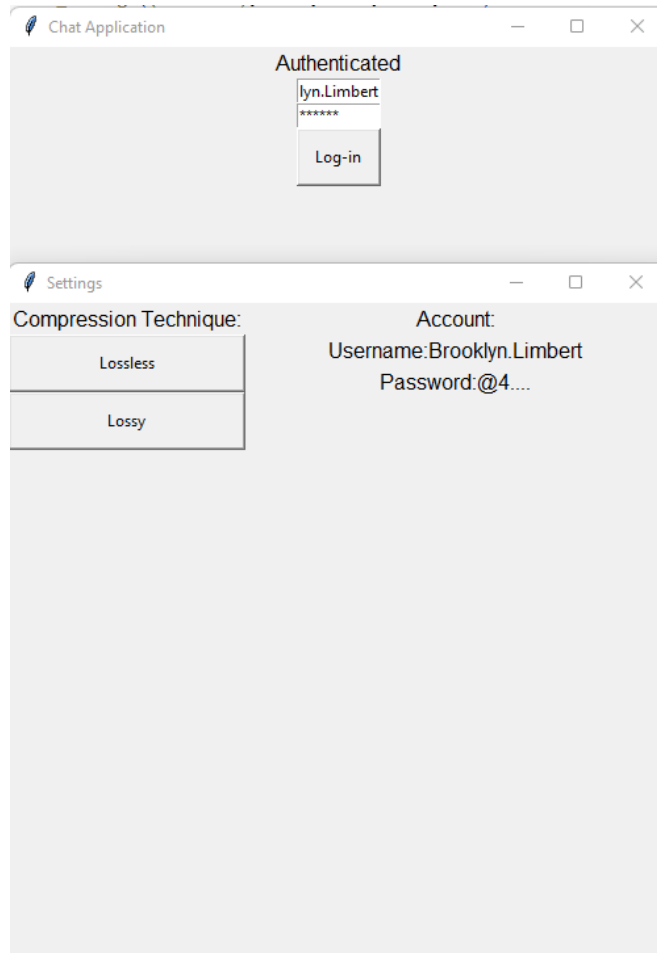
I have created a test data table above. This is the data I will be using to test that messages can be received and sent from multiple accounts simultaneously. The program should handle this fine as the program converts all inputs into strings ensuring all data inputs are of the same type.



The program produced the expected outputs without any problems for both instances of the program.

## Computer Science NEA project - Chat application

- Users can choose their encryption and compression techniques



When opening the settings part of the application the user is presented with this window allowing the user to choose between compression techniques and see their account information but not change the encryption technique in use.

I have not implemented the encryption technique selection due to security concerns, this application will be predominantly used within schools and is designed for closed networks where security and safety is of utmost importance. If a user selects a weak encryption method or none at all this could put the network at risk therefore I have not implemented this feature.

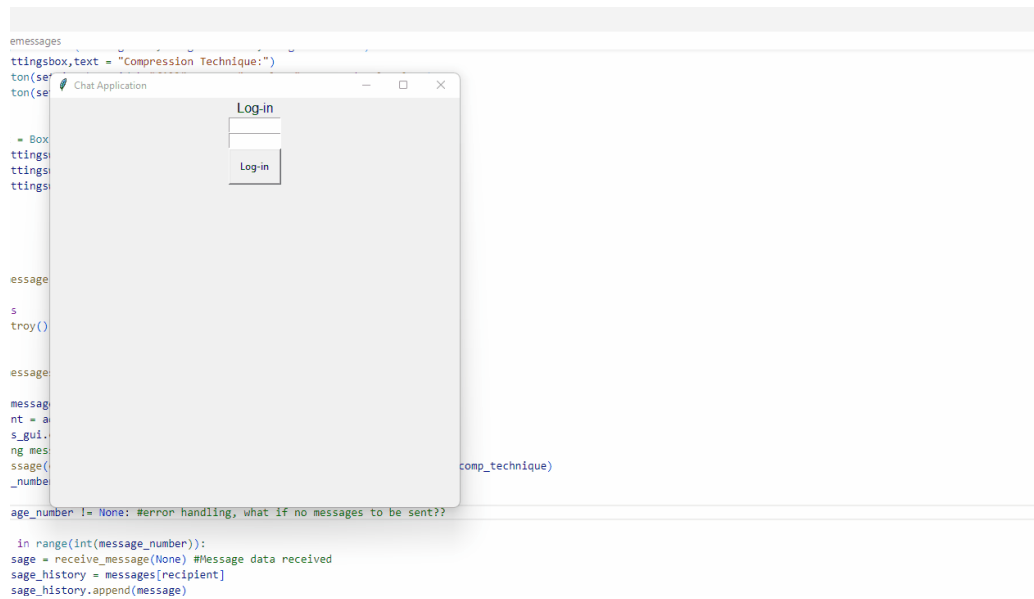
### Discussion with stakeholders:

After discussing the change with stakeholders they agreed that it was not a necessity however the encryption technique being used must be efficient, fast and secure.

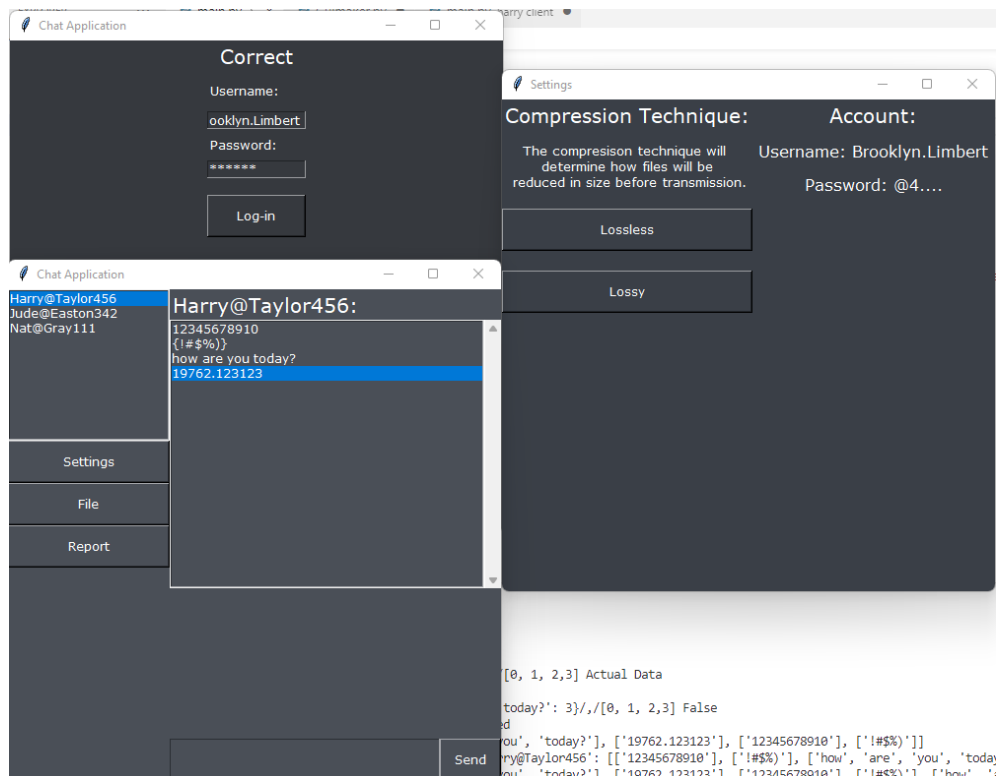
The encryption technique being used is fast and efficient however it currently lacks a sophisticated level of security something which I will be adapting upon larger deployment of the application, however for its current use it is secure.

## Computer Science NEA project - Chat application

- The application must have a responsive intuitive GUI



As seen by the video representation of the algorithm the application is intuitive and responsive however the gui may be hard to see due to the colour palette used therefore I will be implementing a different colour palette to meet this user requirement.



I have used shades of grey and white to accent key parts of the application. This will help people with poor eyesight as it is easy to identify parts of the application because of the different shades of colour, bold writing, accented colours and the padding that has been implemented.

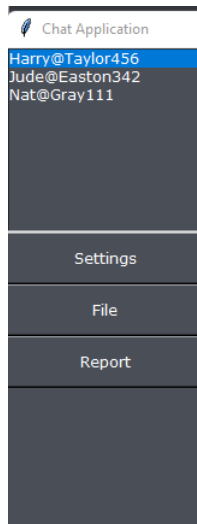
## Computer Science NEA project - Chat application

- **Users can see their message history**

As seen by the previous tests message history can be displayed even after logging off due to the database that is being used.

No matter the data type previous messages can be accessed and displayed as all data is stored in the database in the same format.

- **The address book should be easy to navigate**



The address book has been designed to be easy to navigate and use, addresses will be highlighted if selected with an easy to see accented colour compared to the applications colour palette.

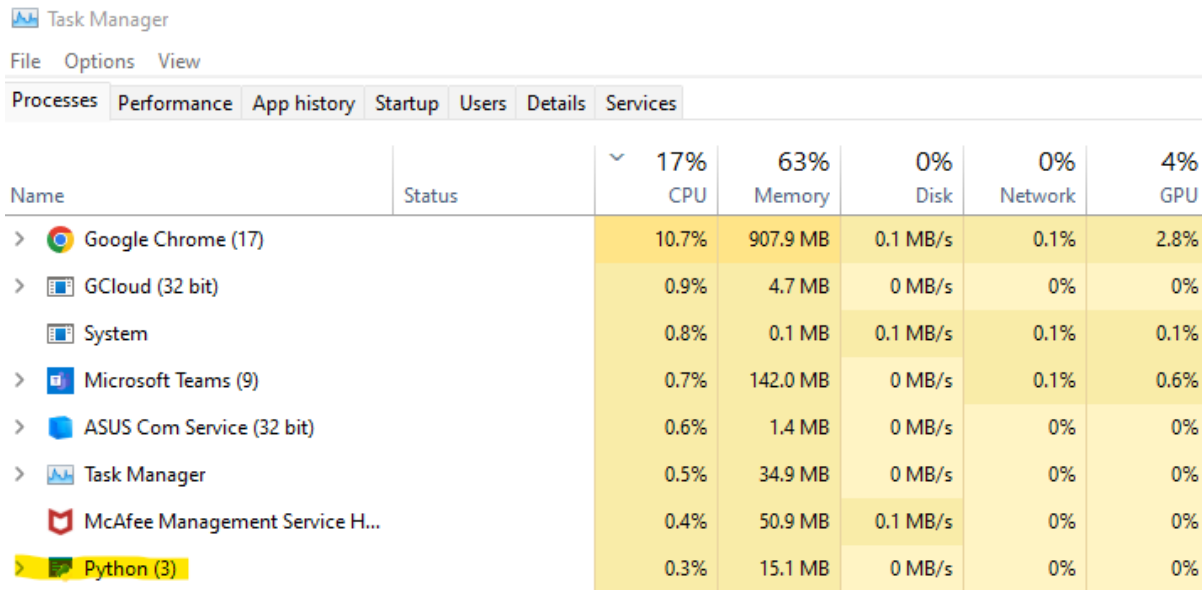
### **Stakeholder discussion:**

The general consensus of the stakeholders after using the application was that the address book was easy to navigate and use even for those with poor eyesight who required glasses.



## Computer Science NEA project - Chat application

- The application is easily ran



The screenshot shows the Windows Task Manager Performance tab. The top bar indicates 'Task Manager' and 'Performance'. Below the tabs, a table displays system resource usage. The table has columns for Name, Status, CPU, Memory, Disk, Network, and GPU. The data is as follows:

Name	Status	CPU	Memory	Disk	Network	GPU
> Google Chrome (17)		10.7%	907.9 MB	0.1 MB/s	0.1%	2.8%
> GCloud (32 bit)		0.9%	4.7 MB	0 MB/s	0%	0%
System		0.8%	0.1 MB	0.1 MB/s	0.1%	0.1%
> Microsoft Teams (9)		0.7%	142.0 MB	0 MB/s	0.1%	0.6%
> ASUS Com Service (32 bit)		0.6%	1.4 MB	0 MB/s	0%	0%
> Task Manager		0.5%	34.9 MB	0 MB/s	0%	0%
McAfee Management Service H...		0.4%	50.9 MB	0.1 MB/s	0%	0%
> Python (3)		0.3%	15.1 MB	0 MB/s	0%	0%

To display the resources used I have taken a screenshot of the windows resource monitor while using the application. This image represents the most resources the application took up while using it for a prolonged period of time. Due to the way the application has been designed, limited processes take place without user input, this is very efficient as the program will not take system resources while not in use.

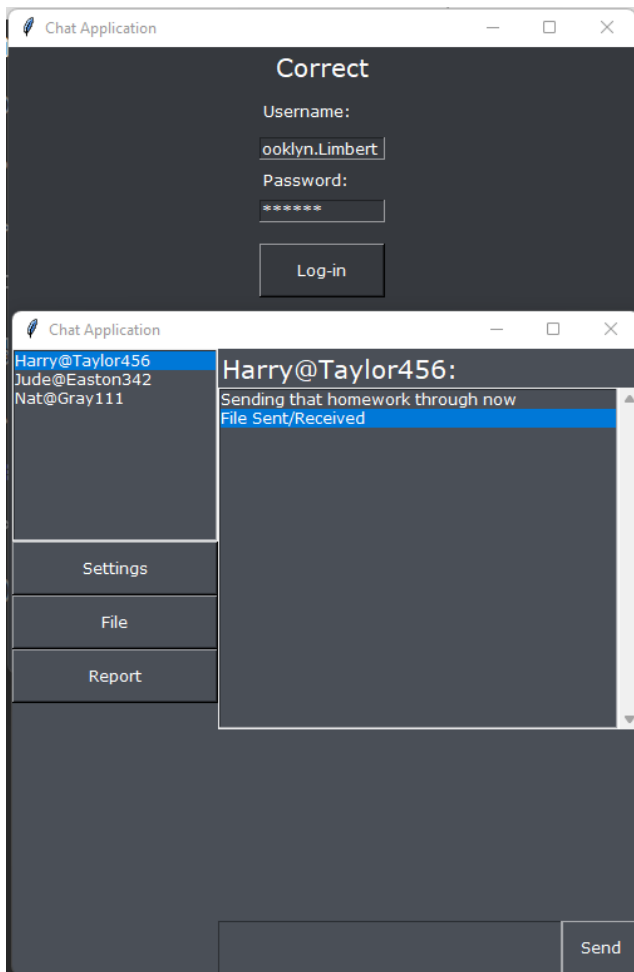
Utilisation	Speed	Base speed:	3.20 GHz
16%	3.42 GHz	Sockets:	1
Processes	Threads	Cores:	8
346	6583	Logical processors:	16
Handles		Virtualisation:	Enabled
Up time		L1 cache:	768 KB
1:22:28:59		L2 cache:	4.0 MB
		L3 cache:	16.0 MB

The system that the application is being run on has a 3.2 Ghz 8 core CPU the application is only taking up 0.3% of this processor's power therefore the application should run on all systems able to run windows 10 and 11.

## Computer Science NEA project - Chat application

- The user is notified when a message is sent/received

The application currently displays messages when they have been sent and received however this does not apply to files and images, I will be implementing a notification when files are sent and received.



```
103 def decompress(data, technique, image, file):
104
105
106     if image == "True" or file == "True":
107         messages_gui.append("File Sent/Received")
---
```

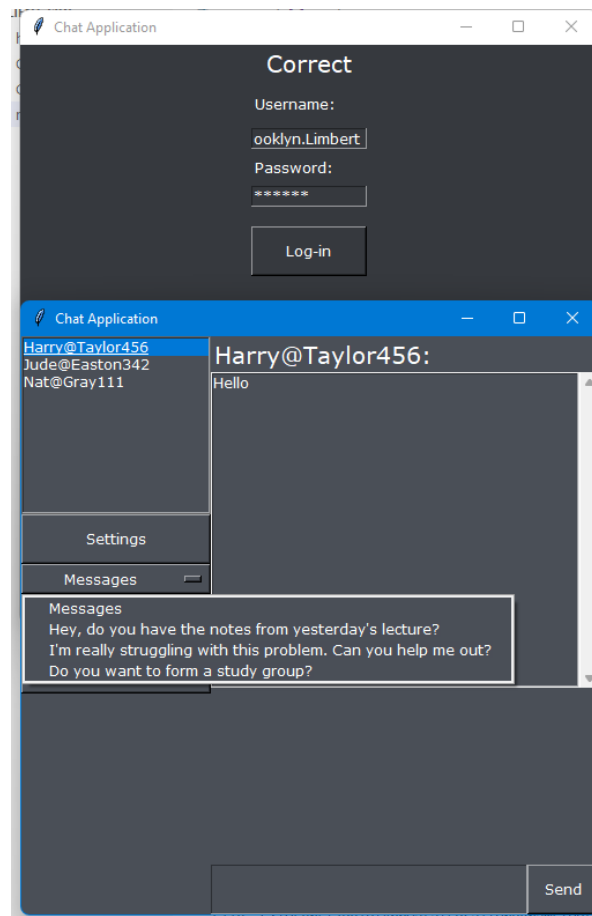
```
443 def uploadfile():
444     global message_gui
445     filepath = app.select_file(filetypes=[("Pictures", "*.png"), ("Pictures", "*.jpg"), ("Text documents", "*.txt")])
446     if filepath.find("*.txt") < 0:
447         img = Image.open(filepath)
448         img = img.save('Ctemp.png')
449         filepath = 'Ctemp.png'
450         send_message(compress(filepath, comp_technique, True, False), "#8", True, False, comp_technique)
451     else:
452         send_message(compress(filepath, comp_technique, False, True), "#8", False, True, comp_technique)
453
454     messages_gui.append("File Sent/Received")
```

This feature has been implemented by adding a notification to the listbox when receiving and sending files, if an image is received it will be displayed using the operating system's default image viewer.

## Computer Science NEA project - Chat application

- **Users can send pre-written messages aimed at students**

This feature has not been implemented yet due to time constraints and the implementation of more important features however it will now be implemented.



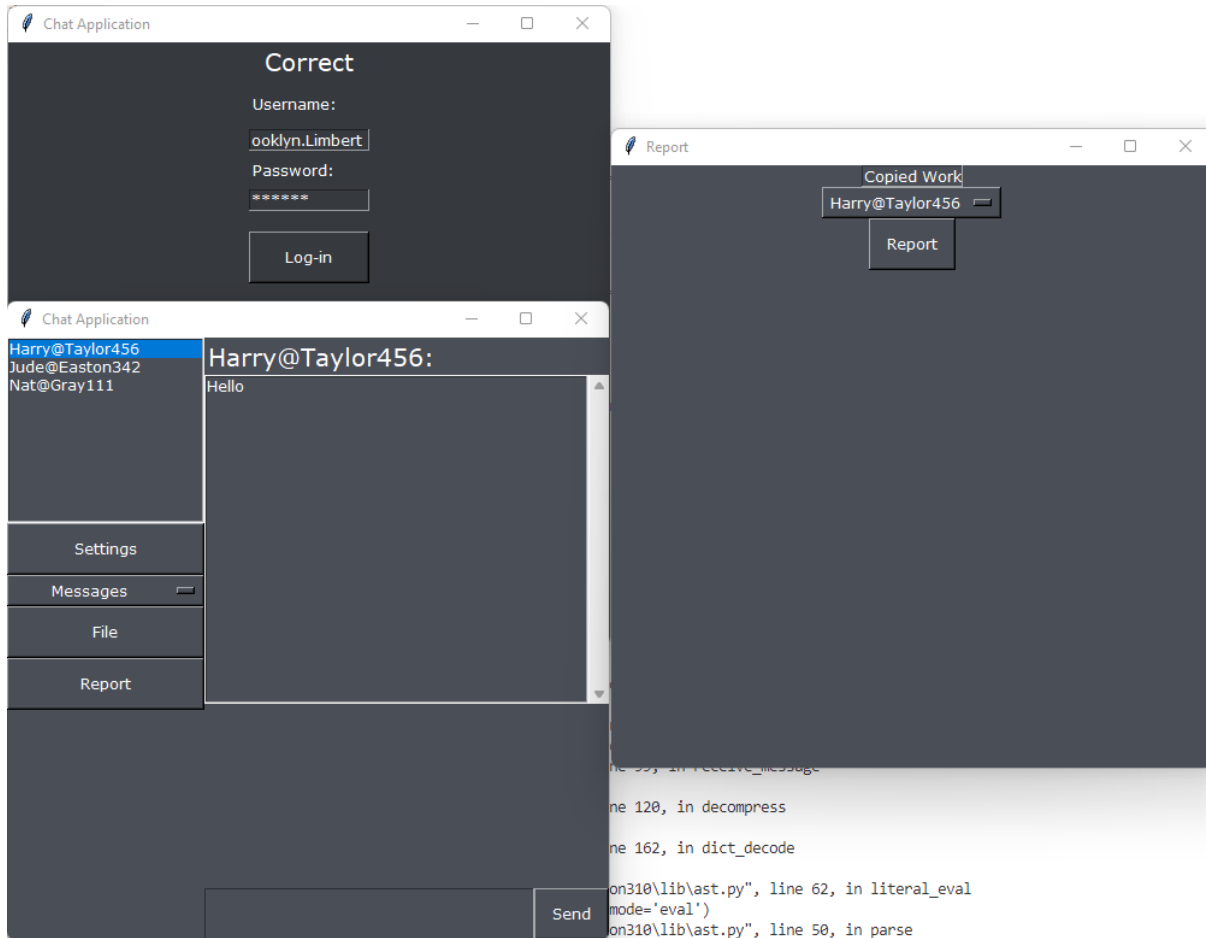
```
369 def quickmessage():
370     global quickmessages,address_book,messages
371     send_message(compress(quickmessages.value,comp_technique,False,False),"#8",False,False,comp_technique)
372     messages_gui.append(quickmessages.value)
373     message_history = messages[recipient]
374     message_history.append(quickmessages.value)
375     messages[recipient] = message_history
376     quickmessages.select_default()
```

```
338 | | | quickmessages = Combo(buttons_box,options = ["Messages","I
```

To implement this I have used the GUIZERO library to create a drop down menu that when selected sends whatever message has been chosen then refreshes the list.

## Computer Science NEA project - Chat application

- The application will have a report feature



The application does have a report feature, it is a minimalistic gui however it is easy to use and navigate. The report button will send the server the information inputted and this will be stored for an administrator to view.

The report feature makes use of the same functions tested earlier, specifically the send message function however different meta data is used therefore a trace table is not required.

[Full test video](#)

### Application feedback:

**Jude Easton** - "I am pleased with the application it meets all of my needs however I am still concerned about security however it will not prohibit me from using the application due to the current limited deployment of the application within closed communities"

**Harry Taylor** - "I can finally share my work without losing data. This application meets my needs, the application will boost my productivity greatly!"

**Group of year twelve students** - The group were happy with the final application and look forward to using it within school settings.

## **Project Summary**

The project has successfully met all user requirements and all users are happy with the application. If the application was to be deployed to a larger audience and development time was increased I would implement more features that would be decided by a larger user requirements study.

### **Client-side script**

[Full Client Application](#)

### **Server-side script**

[Full Server Application](#)

### **Full test video**

[Full test video](#)