**Client-Server Secure Messaging Platform**

**Computer Science A Level Non-Exam Assessment**

**Sam Poirier**

**10 October 2018**

# Contents

[Contents 1](#_Toc525574954)

[1 Analysis 2](#_Toc525574955)

[1.1 Problem Description 2](#_Toc525574956)

[1.2 Research 2](#_Toc525574957)

[1.2.1 Interview 2](#_Toc525574958)

[1.2.2 Questionnaire 2](#_Toc525574959)

[1.2.3 Existing Solutions 2](#_Toc525574960)

[1.3 Project Background 3](#_Toc525574961)

[1.3.1 Internet Relay Chat 3](#_Toc525574962)

[1.3.2 Discord 4](#_Toc525574963)

[1.3.3 Facebook Messenger 5](#_Toc525574964)

[1.4 Questionnaire 6](#_Toc525574965)

[1.4.1 Questions 6](#_Toc525574968)

[1.4.2 Responses 6](#_Toc525574969)

[1.4.3 Analysis 11](#_Toc525574970)

[1.5 Interview 11](#_Toc525574971)

[1.5.1 Questions 11](#_Toc525574972)

[1.5.2 Responses 11](#_Toc525574973)

[1.5.3 Analysis 11](#_Toc525574974)

[1.6 Objectives 12](#_Toc525574975)

[1.6.1 Client 12](#_Toc525574976)

[1.6.2 Server 13](#_Toc525574977)

# 1 Analysis

## Problem Description

I will be creating a secure Client-Server messaging platform, and a chat-bot to run on the server in order to help users with tasks. My client is Mr Ovayolu, a Computer Science teacher who wishes to set up a secure platform for the teachers to discuss work with each other. The focus will be on a secure design, hosted by the client to ensure they know exactly where their messages/data is stored and that it is not being sold to companies by the owners of the software. The end users will be teachers which may not be very ‘tech-literate’; the client must be easy to set up and use.

## 1.2 Research

### 1.2.1 Interview

It is important to interview the client in order to establish the key features that they require for the program to include, as well as what features are less important not needed. I will work from the responses the client gives in order to accurately tailor the software to their requirements.

### 1.2.2 Questionnaire

I will develop a questionnaire to determine the demand from the general public in order to tailor the software to be desirable to users other than the initial client, as long as the general demand does not contradict the client’s needs. I will use google forms to create the questionnaire.

### 1.2.3 Existing Solutions

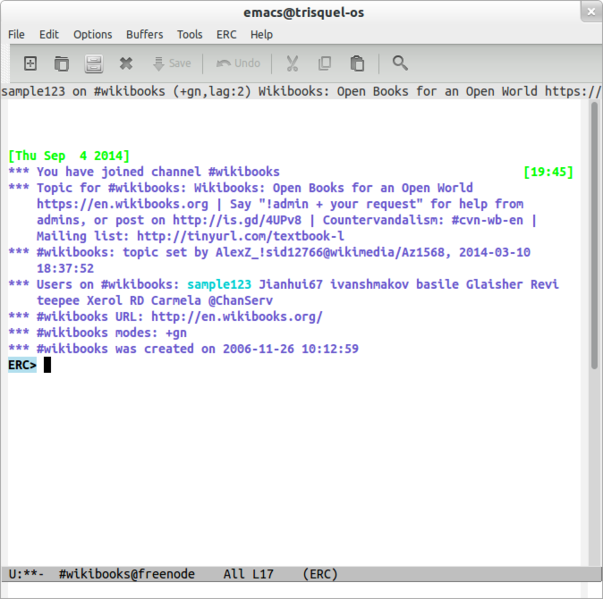
There are currently a number of existing pieces of software that fulfil similar functions, which I will analyse to determine crucial aspects that should be included in my project, but also determine flaws which I should make sure not to include in the software.

## 1.3 Project Background

Instant Messaging platforms have existed pretty much as long as computers have, originally (in the 1960s) acting as a way of communicating between users on multi-user operating systems. They were also originally used for notification systems like printing. A good example of possibly the first instance of this software is from CTSS (Compatible Time-Sharing System), an operating system developed by MIT which was the first to demonstrate any sort of messaging, and was a precursor to email.

Hashing has also existed as a concept for a long time, with the term originating from its non-technical meaning (to "chop" or "make a mess" out of something), and was first coined in the 1960s. For this project, we will be specifically looking at Cryptographic Hashing which usually consists of a mathematical algorithm that maps data of arbitrary size to a bit string (a hash) and is designed to be a one-way function. The purpose of this function is to obscure a user’s personal data so that it is secure.

### 1.3.1 Internet Relay Chat

The first IM platform to run over the internet was Internet Relay Chat (IRC), which was made in 1988. By 1989 IRC had spread across the world with over 40 servers worldwide.

*Figure 1.1: An IRC Client called ‘ERC’ which runs within the text editor Emacs.*

**Pros**

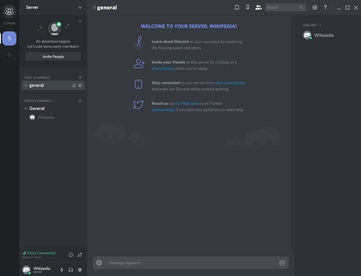
* Client-server model – the client’s GUI or functionality can be upgraded without touching the server allowing for all sorts of clients to be compatible with each other.
* Can be secure if set up correctly.

**Cons**

* Complicated to set up servers correctly and securely.
* Some clients are complicated to use and set up.

### 1.3.2 Discord

Released in May 2015, Discord is a multiplatform Instant Messaging and voice-over-internet Protocol (VoIP) application. It runs on pretty much every platform available and is free to use.



*Figure 1.2: The Discord desktop application after creating a new server.*

**Pros**

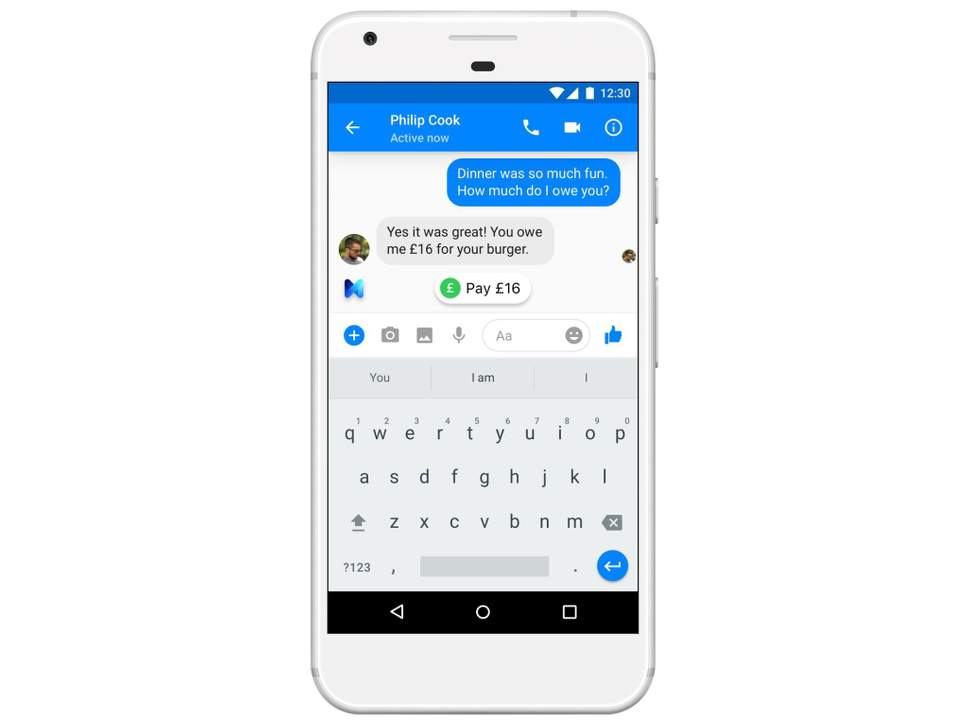
* Extremely easy to create new accounts and servers – all implementation is backend.
* Servers are secure – only invited users can join.
* Lots of useful features – image embedding, voice calling, direct messages etc.

**Cons**

* No personally hosted servers – all data is controlled and stored by Discord Inc. meaning that there is little a user is not in control of its security or status.
* Written using Electron – a framework with a famously high memory footprint on the PCs running it.

### 1.3.3 Facebook Messenger

Facebook Messenger (commonly known as Messenger) is an IM platform designed for mobile devices and originally developed as Facebook Chat, which was then released as a standalone client, separate from Facebook.



*Figure 1.3: Facebook Messenger running on an Android device.*

**Pros**

* Easy to use and setup.
* Easy to connect to new users.
* Lots of features, including sending money, voice chat, sharing photos etc.

**Cons**

* Very little privacy or control over data – it is all owned by Facebook, a company which is notorious for abuse of its user’s personal data.
* User interface is not designed for large groups of people or high rate of messages being sent.

### 1.3.4 MD5 Hashing Algorithm

### 1.3.4 SHA-2 Hashing Algorithm

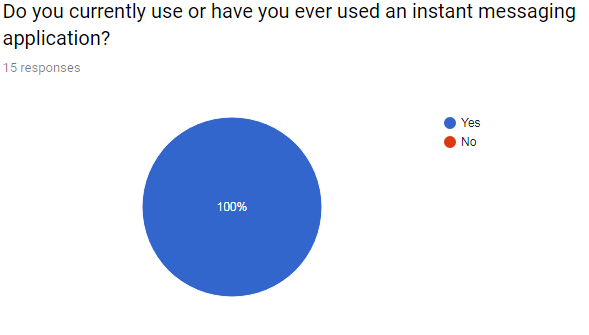
## 1.4 Questionnaire (<https://goo.gl/forms/pDIhsGHZyYHRz6J23>)

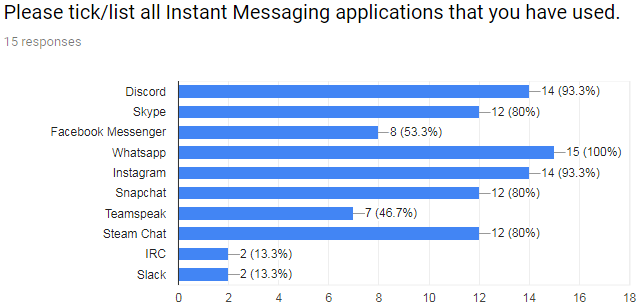


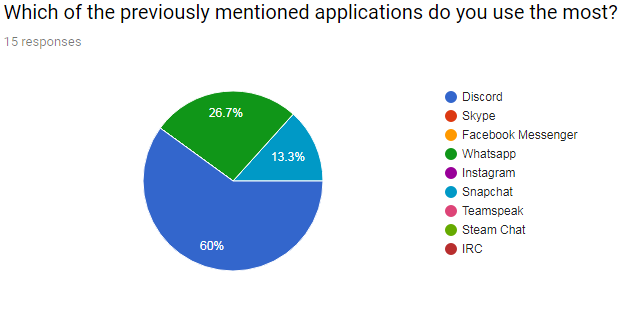
### 1.4.1 Questions

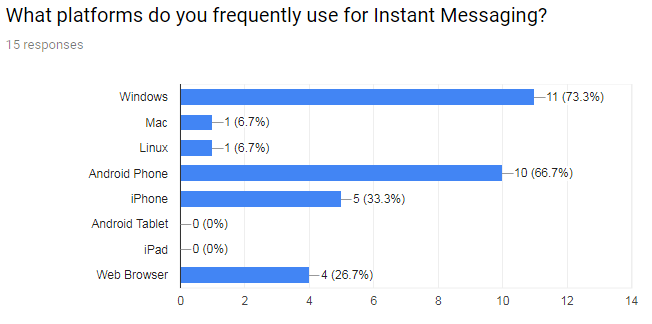
1. Do you currently use or have you ever used an instant messaging application?
   1. If no, is there a particular reason for this?
   2. Otherwise, continue
2. What Instant Messaging Applications have you used?
3. Which of the previously mentioned applications do you use the most?
4. What platforms do you frequently use for instant messaging?
5. Which of the previously mentioned platforms do you use the most?
6. On a scale of 1-5, how important are these features to you?
   1. Data Security
   2. Secure Logins
   3. Viewable Message History
   4. Transparency about what happens to your data/where it is stored
   5. Easy to understand User Interface
   6. User Interface Customisation
   7. Direct Messaging
   8. Group Chats
   9. Multimedia Support (image/video embeds, text formatting etc)
7. Are there any other features that are important to you?

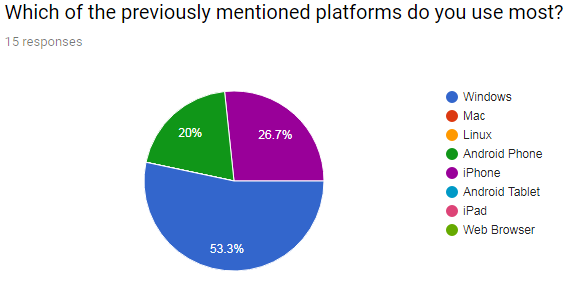
### 1.4.2 Responses

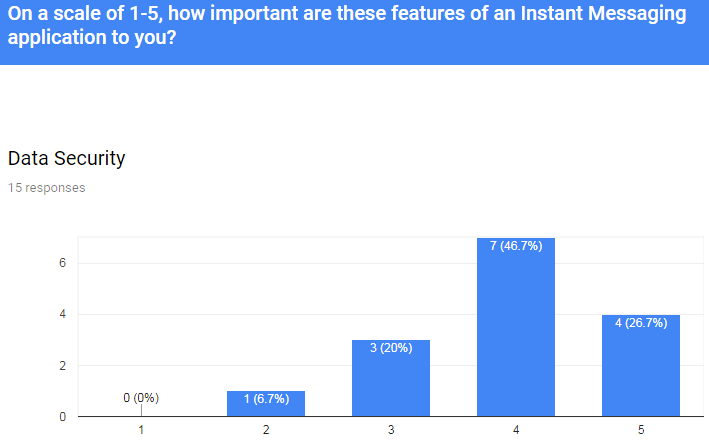


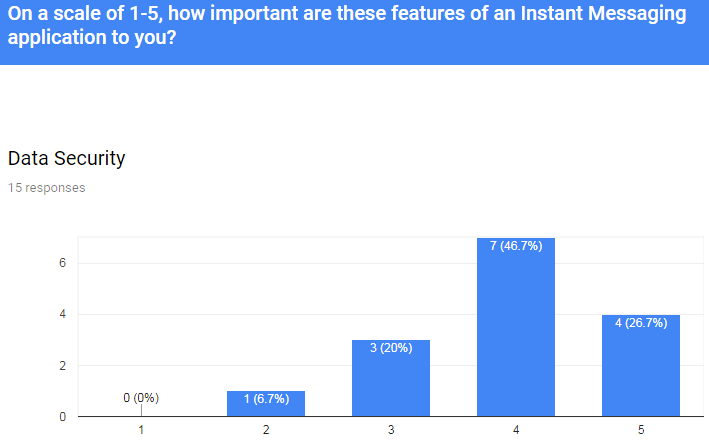


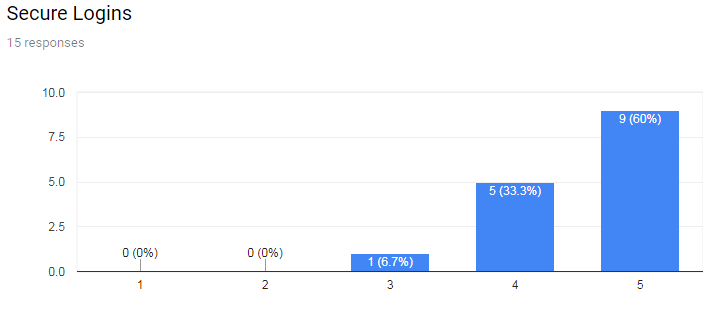


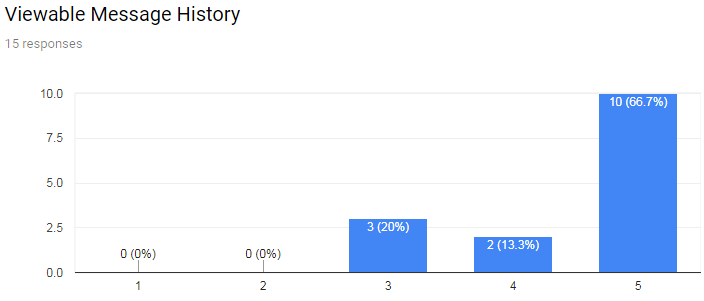


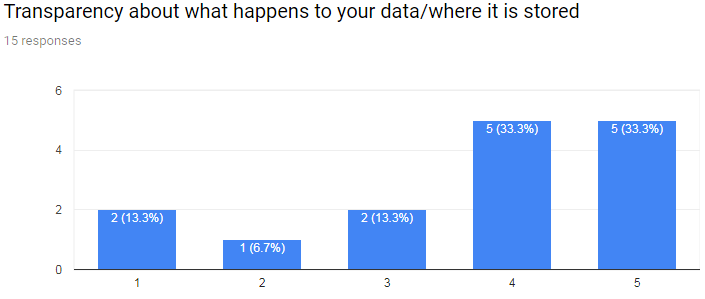


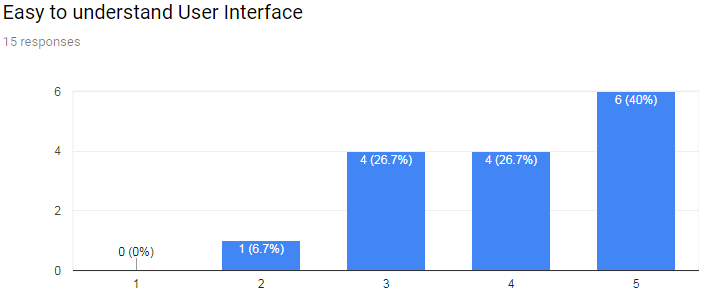


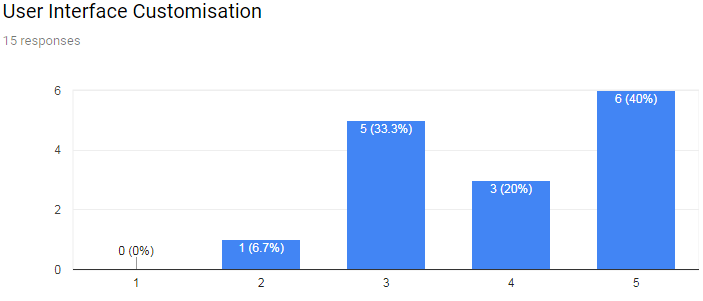


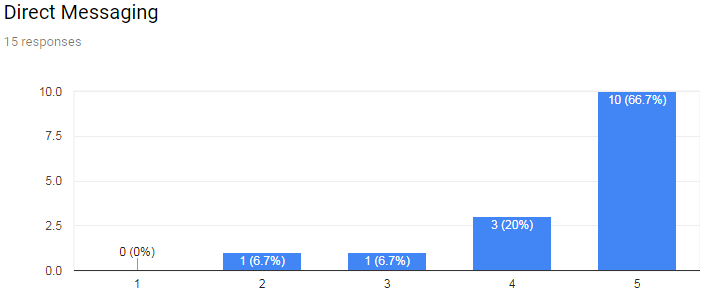


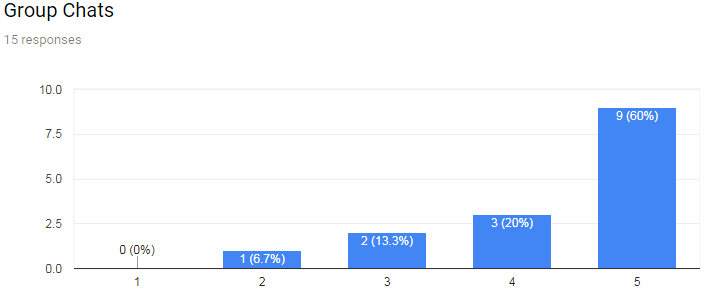


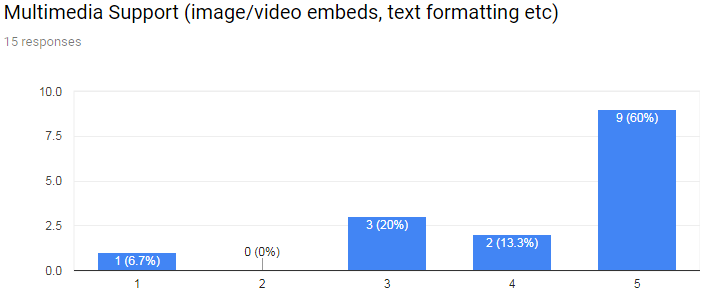


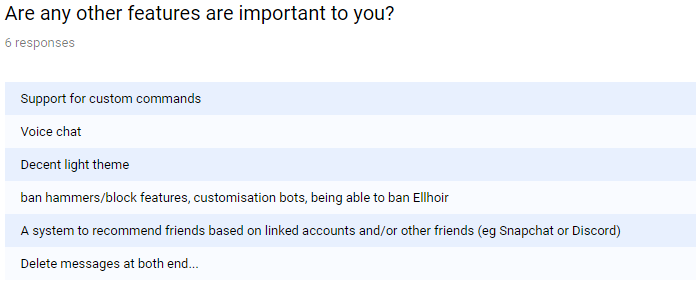












### 1.4.3 Analysis

## 1.5 Interview

### 1.5.1 Questions

1. What features of an instant messaging platform are most important for you? (For example, data security, simple UI, etc)
2. How often will the users use the platform?
3. How many users should the platform support? Concurrent or total?
4. How much message history should users be able to see?

### 1.5.2 Responses

### 1.5.3 Analysis

## 1.6 Objectives

### 1.6.1 Client

1. Login Screen - UI
   1. Input for username and password
   2. Login button
   3. Register User button
   4. Error label to display login issues (wrong password, etc)
2. Register User Screen - UI
   1. Input for username, password and confirm password
   2. Register button
   3. Error label to display register issues (user already exists, etc)
3. Main Window - UI
   1. Box to contain sent messages
   2. Input for message to send
   3. Button to send message
   4. Label displaying logged in username
   5. Box to contain online users - sorted alphabetically
4. Connecting to Server
   1. Initiates connection
   2. Send create user request (optional)
   3. Shares login details
   4. Waits for the message from server they are correct
   5. Logs user into main window
   6. Displays previous x messages which were sent before connection
5. Sending Messages
   1. Clears input box
   2. Checks validity
      1. No SQL injections, HTML markup or invalid characters
   3. Checks for commands (/ping, /help etc)
   4. Packages up message with user info
   5. Sends packet to server
6. Receiving Messages
   1. Formatted correctly: [date][user][message]
   2. Supports formatting (bold, italics etc)
   3. Displays new messages chronologically at the bottom of the UI element

### 1.6.2 Server

1. Connecting clients
   1. Can connect to multiple clients at once
   2. Accepts new connections if valid while simultaneously dealing with messages from already logged in users
   3. Registers new users (optional)
   4. Connects users if their login credentials are valid
   5. Sends the new client the previous x messages, allowing them to ‘catch up’
   6. Sends a message to inform all clients a new user has joined
2. Receiving Messages
   1. Checks if they’re a command and if so follows through
   2. Inserts time received information
   3. Broadcasts message to all connected clients
3. Disconnecting Clients
   1. If socket is dropped, tell all clients the user has disconnected
   2. Forget all data about that user’s session as it’s no longer needed
4. Storing Data
   1. All data should be stored in a relational database
   2. User passwords should be hashed
   3. All messages should be stored between server sessions

## 1.7 Program Structure

The application will be divided into three main sections:

* The Sever
  + One to many
  + Heavyweight; as much logic as possible should be located on the server
  + Includes database
  + Simultaneously holds connections with lots of clients – multithreaded
* The Client
  + Many to one
  + Lightweight; as little logic as possible should be located on the client
* Shared Library
  + Lots of functions/classes should be shared between the client and server as they need to be consistent, for example the data packet classes or the hashing algorithms.