Client-Server Messaging Platform

AQA Computer Science A Level Non-Exam Assessment

Sam Poirier

01 February 2019

Contents

[1 Analysis 4](#_Toc536793365)

[1.1 Analysis Plan 4](#_Toc536793366)

[1.1.1 Interview 4](#_Toc536793367)

[1.1.2 Questionnaire 4](#_Toc536793368)

[1.1.3 Project Background 4](#_Toc536793369)

[1.1.4 Objectives 4](#_Toc536793370)

[1.1.5 Solutions 4](#_Toc536793371)

[1.2 Interview 5](#_Toc536793372)

[Questions 5](#_Toc536793373)

[Responses 5](#_Toc536793374)

[Analysis 5](#_Toc536793375)

[1.3 Questionnaire 6](#_Toc536793376)

[Questions/Responses 6](#_Toc536793377)

[Analysis 6](#_Toc536793378)

[1.4 Project Background 9](#_Toc536793379)

[1.4.1 Specific Client Needs 9](#_Toc536793380)

[1.4.2 Existing Solutions 10](#_Toc536793381)

[1.5 System Objectives 14](#_Toc536793382)

[1.6 Solutions 15](#_Toc536793383)

[1.6.1 Framework 15](#_Toc536793384)

[1.6.2 GUI Framework 15](#_Toc536793385)

[1.6.3 Integrated Development Environments 16](#_Toc536793386)

[1.6.4 Database Formats 16](#_Toc536793387)

[1.6.5 Solution Decisions 16](#_Toc536793388)

[2 Documented Design 17](#_Toc536793389)

[2.1 Database Structure & SQL Statements 17](#_Toc536793390)

[2.1.1 Entity Relationships and Attributes 17](#_Toc536793391)

[2.1.2 Normalised Data Structures 17](#_Toc536793392)

[2.1.3 Data Dictionaries 17](#_Toc536793393)

[2.1.4 SQL Statements 18](#_Toc536793394)

[2.1.5 Data Flow Diagram 19](#_Toc536793395)

[2.2 System Design 20](#_Toc536793396)

[2.2.1 IPSO Chart 20](#_Toc536793397)

[2.2.2 Top-down diagrams 21](#_Toc536793398)

[2.2.3 Client Flowchart 23](#_Toc536793399)

[2.2.4 Server Flowchart 26](#_Toc536793400)

[2.3 Data Structures & Dictionary 28](#_Toc536793401)

[2.3.1 Circular Queue 28](#_Toc536793402)

[2.3.2 Client Data Dictionary 28](#_Toc536793403)

[2.3.3 Server Data Dictionary 28](#_Toc536793404)

[2.4 Class Design 29](#_Toc536793405)

[2.4.1 Client Classes 29](#_Toc536793406)

[2.4.2 Server Classes 30](#_Toc536793407)

[2.4.3 Generic Classes 30](#_Toc536793408)

[2.5 User Interface Design 31](#_Toc536793409)

[2.6 Algorithms 34](#_Toc536793410)

[2.6.1 Circular Queue 34](#_Toc536793411)

[2.6.2 Hashing Algorithm 35](#_Toc536793412)

[2.6.3 Integer Merge Sort 36](#_Toc536793413)

[2.6.4 String Merge Sort 37](#_Toc536793414)

[3 Technical Solution 38](#_Toc536793415)

[3.1 Server 38](#_Toc536793416)

[3.1.1 main.py 38](#_Toc536793417)

[3.1.2 logger.py 45](#_Toc536793418)

[3.1.3 database.py 45](#_Toc536793419)

[3.1.4 config.json 50](#_Toc536793420)

[3.2 Client 50](#_Toc536793421)

[3.2.1 main.py 50](#_Toc536793422)

[3.2.2 adminSettings.ui 63](#_Toc536793423)

[3.2.3 login.ui 66](#_Toc536793424)

[3.2.4 mainWindow.ui 68](#_Toc536793425)

[3.2.5 message.ui 73](#_Toc536793426)

[3.2.6 messageOptions.ui 76](#_Toc536793427)

[3.2.7 register.ui 79](#_Toc536793428)

[3.2.8 config.json 81](#_Toc536793429)

[3.3 Libs 81](#_Toc536793430)

[3.3.1 configManager.py 81](#_Toc536793431)

[3.3.1 packets.py 82](#_Toc536793432)

[3.3.1 photonUtilities.py 85](#_Toc536793433)

# Analysis

## Analysis Plan

### Interview

It is important to interview the client, Mr Ovayolu; a secondary school teacher, 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.

### 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.

### Project Background

Within the Project Background I will detail the specific Client needs, in order to fully flesh out the purpose of the software. I will also analyse a number of existing pieces of software that fulfil similar solutions 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

### Objectives

Under the objectives section, I will detail the exact objectives the project must meet, along with the specific criteria of each objective. This will allow me to ensure that the final product meets with all the client expectations.

### Solutions

Here I will detail the technical aspects I plan to use in order to develop the software, including programming languages, Integrated Development Environments, general database rules (if applicable) and other technical Solutions.

## Interview

|  |  |  |
| --- | --- | --- |
| Questions | Responses | Analysis |
| What features of an instant messaging platform are most important for you? | • Fast and secure communication between students by 1:1 or group chat  • Easy to understand, clean but functional user interface  • Be able to run in all Windows versions  • View chat history  • Group chat and direct messaging  • Reply to a particular message, both in a group chat and in a private chat.  • Forward the messages to a particular group chat or an individual.  • Get alert when new messages arrived.  • View list of connected users in the school.  • Report button for inappropriate, bullying, threatening messages… | The software is being designed specifically for this user. It is important that as many of these are implemented as possible, and ideally all of them are. |
| How often will the users use the platform? | Break and lunch time | The software will mostly be used during the day, so there will be time available for potential server maintenance; there isn’t a need for 100% uptime. |
| How many users should the platform support? | Concurrent around 100, total around 2000 | The program must support a large quantity of users – some stress/load testing will be needed to ensure the load can be managed successfully. |
| How much message history should users be able to see? | 3 Months or 100 messages. Ideally, user should be able to set this in settings. | The client wants a configurable number of messages to be readable, meaning the server should likely store all messages then send the client the amount they have requested. |

## Questionnaire

|  |  |
| --- | --- |
| Questions/Responses | Analysis |
|  | Everyone who answered the survey has used an IM application at some point, which means that the feedback received is not necessarily representative of all the users, so should not be treated as an absolute representation of the userbase. |
|  | WhatsApp is the most widely used IM application, followed closely by Discord and Instagram. The least widely used applications are IRC and Slack. All of these extremes should be investigated in more detail in order to replicate their success and avoid their mistakes. |
|  | Interestingly enough, although WhatsApp was the most widely used, Discord was in fact the one that was used the most overall, and snapchat which was thoroughly in the middle of the widely used, was one of the only three chosen as most used. This shows that the most widely used applications are not necessarily the best ones. |
|  | Windows and Android Phone are by far the most widely used, and so for the best coverage of users, are the platforms that would be the best to target. |
|  | Again, Windows is by far the most used platform and is therefore the clear frontrunner in terms of OS to develop for. |
|  | 73.4% of those surveyed indicated that Data Security is important for them. This is clearly an important feature. |
|  | 93.3% of users indicated that secure logins where an important feature for them so this also is a crucial aspect of the project. |
|  | Again, most users agree that having a viewable message history is an important feature of an IM application. |
|  | Only just over half of the responses indicated that they cared about transparency regarding their personal data and what was done with it. |
|  | Most people agreed that an easy to understand/use user interface was a good feature. It could be worth developing a few different layouts to receive feedback about. |
|  | A large number of those surveyed indicated that a customisable UI would be wanted. More detail might be needed into what exact customisations are wanted. |
|  | Most users agreed that Direct Messaging would be an important feature. It was also requested by the Client so is a high priority. |
|  | Group chats were also indicated a feature wanted by the client as well as the survey candidates. This means that it should definitely be implemented in the product. |
|  | A fairly mixed response from the survey, but this was also requested by the client so should be included regardless. |
|  | Some of these feature requests are more in the scope of this project than others. These were individually requested and so are low priority. |

## 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.

### Specific Client Needs

I will be creating a secure Client-Server messaging platform. My client is Mr Ovayolu, a Computer Science teacher who wishes to set up a secure platform for the teachers and students 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. This is important as a large amount of the data on the system will belong to minors, so the school has extra responsibility to protect them. The end users will be teachers or students who may not be very ‘tech-literate’; the application must be easy to set up and use.

### Existing Solutions

#### 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.

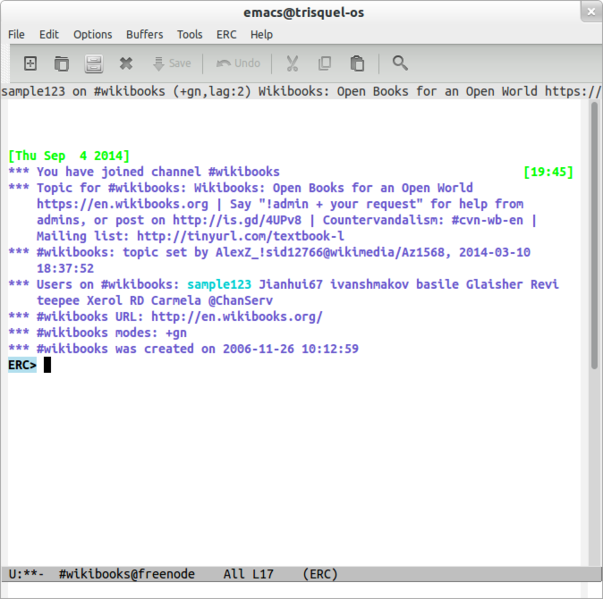


Figure 1.1: An IRC Client called ‘ERC’ which runs within the text editor Emacs.

|  |  |
| --- | --- |
| Pros | Cons |
| 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. | Complicated to set up servers correctly and securely. |
| Can be secure if set up correctly. | Some clients are complicated to use and set up. |
|  | Little support for multimedia formats, reporting messages, file transfer etc. |

#### 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.

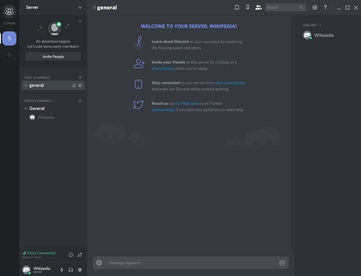


Figure 1.2: The Discord desktop application after creating a new server.

|  |  |
| --- | --- |
| Pros | Cons |
| Extremely easy to create new accounts and servers – all implementation is abstracted away. | No personally hosted servers – all data is controlled and stored by Discord Inc. meaning that a user is not in control of it’s data’s security or status. |
| Servers are secure in the sense that only invited users can join. | Written using Electron – a framework with a famously high memory footprint on the PCs running it, so may not work well on older machines. |
| Lots of support of extra features, for example VOIP, image embeds, direct messages, file transfer. |  |

#### WhatsApp

WhatsApp is a free Instant Messaging, Voice and Video Chat client. It was founded in 2009 and is owned by Facebook. It is supported on many platforms including most smart phones.

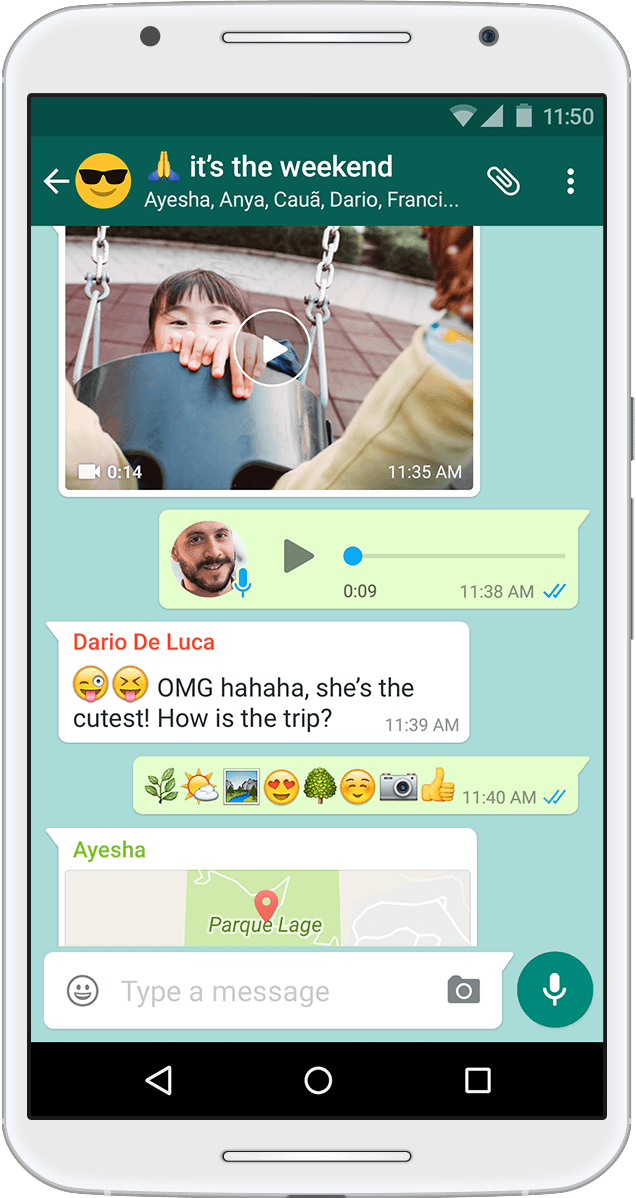


Figure 1.3: WhatsApp running on an Android Device

|  |  |
| --- | --- |
| Pros | Cons |
| Easy to use and set up. | Very little privacy or control over data – it is owned by Facebook, a company which is notorious for abuse of its users’ personal data. |
| Easy to connect to new ‘servers’ or group chats. | User interface is not designed for large groups of people or high rate of messages being sent. |
| Lots of features including voice chat, file sharing, direct messages etc. |  |

#### MD5 Hashing Algorithm

MD5 is one in a series of algorithms designed by Professor Ronald Rivest of MIT in 1991. It was designed as a cryptographic hashing function for use in encryption.

|  |  |
| --- | --- |
| Pros | Cons |
| Implementations exist already in many different programming languages already, in addition to pseudocode for the algorithm being widely available. | Suffers from extensive vulnerabilities, including a high chance of collisions (when two or more different input string produce the same output hash string) rendering it obsolete for encryption usage. |

#### 1.4.2.5 SHA-2 Hashing Algorithm

SHA-2 (Secure Hash Algorithm 2) is a set of cryptographic hash functions designed by the United States National Security Agency (NSA) in 2001. It utilises a Merkle–Damgård structure.

|  |  |
| --- | --- |
| Pros | Cons |
| When implemented correctly is virtually uncrackable. | Low support on older systems, for example Windows XP. This conflicts with the Client’s requests of running on all OSes |

## System Objectives

|  |  |  |
| --- | --- | --- |
| No | Objective | Performance Criteria |
| 1 | Login Screen User Interface must contain:   * Input for username and password * Login Button * Register Button * Error Label to display login issues | All of these UI elements must exist in an organised and easy to read layout. |
| 2 | Register User Screen UI must contain:   * Input for username, password and password confirmation * Register (submit) button * Error label to display register issues | All of these UI elements must exist in an organised and easy to read layout. |
| 3 | Main Window UI must contain:   * Box to contain message history * Input box for sending messages * Button to send message * General server information; who’s connected, who you’re logged in as etc | All of these UI elements must exist in an organised and easy to read layout. |
| 4 | Connecting to the server should be secure. | The initial handshake between client and server should be secure; ie no ‘fake’ clients attempting to connect should be accepted. |
| 5 | Client/Server Communication should be secure. | All packets to and from the server should be secure so that anyone intercepting them cannot read the information. |
| 6 | All messages and personal data should be stored securely. | The database should be encrypted; the whole thing in a two-way encryption unique to the server, and the passwords in a one-way encryption too. |
| 7 | User interface should be easy to use and read. | The UI should be intuitive to read, use and understand. |
| 8 | Client and server should be easy to set up and run. | The client and server should require minimal setup to get into full working/running order. |
| 9 | The server must be able to handle lots of users both consecutively and in general. | The server must be able to support 100 concurrently connected users and 2000 total registered users. |
| 10 | There must be a way for users to configure settings for the client. | The clients should be configurable via a settings menu which is saved between sessions. |
| 11 | The application must support most requested features by the client. | As many features requested by the client as possible should be implemented into the program. |

## Solutions

### Framework

|  |  |  |
| --- | --- | --- |
| Language | Pros | Cons |
| Python | Lightweight  Simple Syntax  Can be developed on school computers  Multiplatform  Decent documentation | Interpreted therefore slow and not memory efficient |
| C# | Powerful  Personally preferred language  Lots of good documentation  Very easy to debug | Low multiplatform support  Cannot be developed on school machines |
| JavaScript (via Electron) | Lightweight  Forgiving syntax  Multiplatform | Extremely loosely typed – easy to make bugs  Hard to debug |

### GUI Framework

|  |  |  |  |
| --- | --- | --- | --- |
| Language | Framework | Pros | Cons |
| Python | tKinter | Shipped stock with python, no extra installation | Extremely limited in terms of functionality  Hard to use |
| PyQt | Support for advanced GUI configuration  Easy to use | C++ Backend, some errors are hard to catch and result in silent crashes |
| C# | Winforms | Easy to use  Stock with C# | Lots of auto-generated code or lots of manual setup |
| JavaScript | Electron (with node.js, jQuery and npm) | Frontend developed in HTML/CSS so the GUI is very customisable. | Very memory inefficient  Uses lots of other people’s modules/code |

### Integrated Development Environments

|  |  |  |
| --- | --- | --- |
| IDE | Pros | Cons |
| Visual Studio | Advanced Debugging tools  Good at identifying programming errors before compiling/running  Easy to use | ‘Heavy’ piece of software – lots of overhead |
| Visual Studio Code | Advanced Debugging tools  Good at identifying programming errors before compiling/running  Easy to use  Lightweight | Less features than Visual Studio |
| IDLE | Comes stock with most python installations  Very lightweight  Simple Debugging tools | Feature lacking; little auto completion or pre-run bug finding |
| Notepad++ | Extremely lightweight | Very few features  Cannot compile many languages |

### Database Formats

|  |  |  |
| --- | --- | --- |
| Format | Pros | Cons |
| SQL | Very powerful  Lots of supported datatypes  Can create databases with complex structures | Complicated to set up  ‘Heavy’ – not very lightweight at all |
| SQLite | Lightweight  Reasonable amount of datatypes  Support for reasonably complicated structures | Less features than SQLite |
| MongoDB | Unique Solution; structure is based upon a tree rather than a table | Feature lacking; little auto completion or pre-run bug finding |

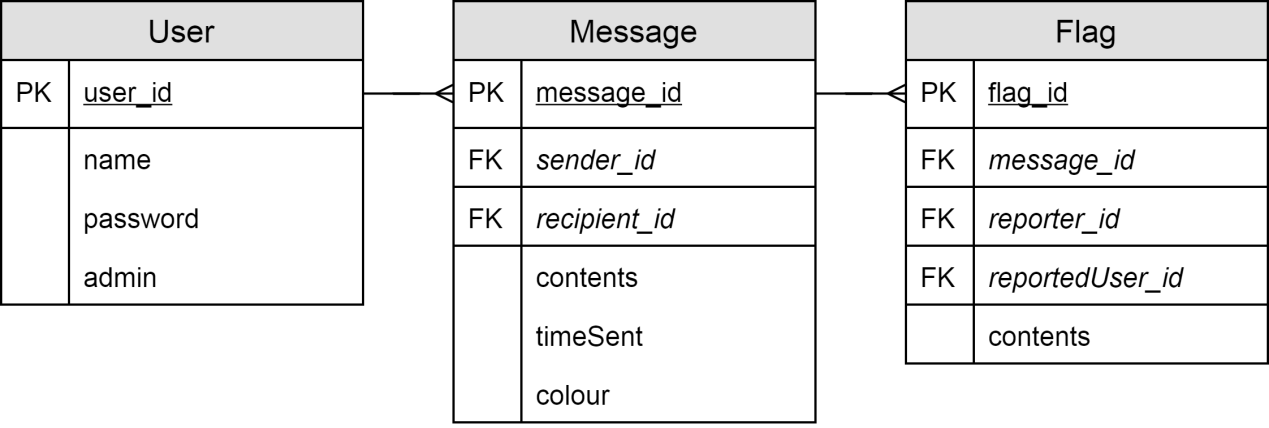
### Solution Decisions

I plan to write my program in Python, as it runs on multiple platforms easily and can be debugged without too much trouble. I will use IDLE to develop the project as it has all the features I need, and is nice and lightweight without too much bloat. I will use PyQt for my GUI as tKinter is not advanced or powerful enough to meet the client needs. I will use SQLite for my database as it will likely be quite simple; SQLite is a good compromise between features and functionality.

# Documented Design

## 2.1 Database Structure & SQL Statements

### 2.1.1 Entity Relationships and Attributes



### 2.1.2 Normalised Data Structures

User(user\_id, name, password, admin)

Message(message\_id, *sender\_id*, *recipient\_id*, contents, timeSent, colour)

Flag(flag\_id, *message\_id*, *reporter\_id*, *reportedUser\_id*, contents)

### 2.1.3 Data Dictionaries

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table Name | User | | | |
| Primary Key | user\_id | | | |
| Foreign Keys | N/A | | | |
| **Data Item** | **Data Type** | **Validation** | **Sample Data** | **Description** |
| name | String | Not null | “user” | The username. |
| password | String | Not null | “O.L&ZXJL3=!VK&\>@,!” | The (one way) hashed password. |
| admin | Boolean | Not null | False | Indicates whether this account has admin privileges. |

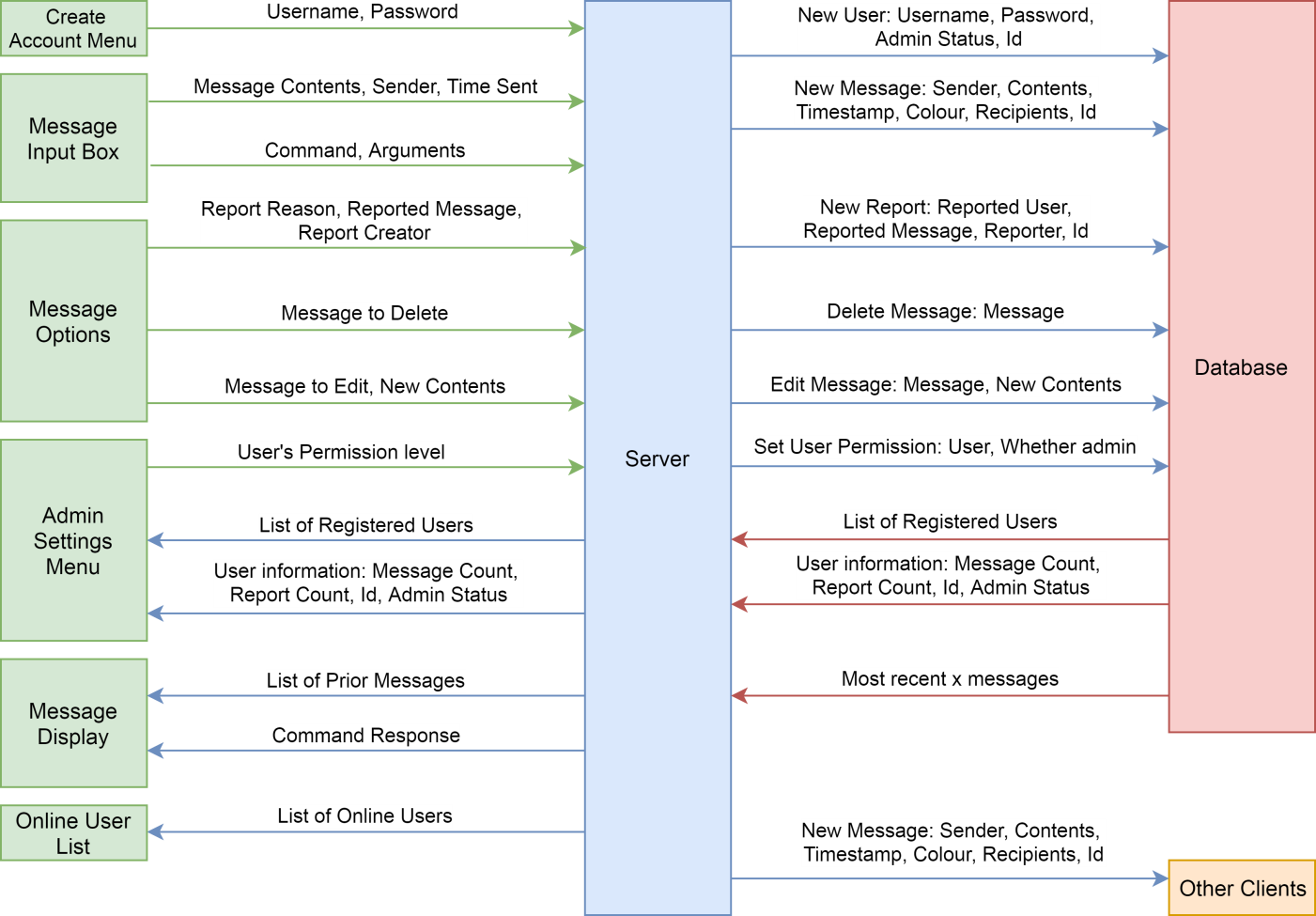
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table Name | Message | | | |
| Primary Key | message\_id | | | |
| Foreign Keys | sender\_id, recipient\_id | | | |
| **Data Item** | **Data Type** | **Validation** | **Sample Data** | **Description** |
| sender\_id | Integer | Not null | 2 | The id of the user who sent the message. |
| recipient\_id | Integer | Not null | 1 | The id of the recipient of the message (1 is the server which means everyone). |
| contents | String | Not null | “Hello World” | The content of the message. |
| timeSent | String | Not null | “26-10-00 11:10” | The date/time the message was sent. |
| colour | String | Not null | “#000000” | The hex colour value to display the message as. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table Name | Flag | | | |
| Primary Key | user\_id | | | |
| Foreign Keys | message\_id, reporter\_id, reportedUser\_id | | | |
| **Data Item** | **Data Type** | **Validation** | **Sample Data** | **Description** |
| message\_id | Integer | Not null | 1 | The id of the message that was reported. |
| reporter\_id | Integer | Not null | 2 | The id of the user who reported the message. |
| reportedUser\_id | Integer | Not null | 3 | The user who’s message was reported. |
| conents | String | Not null | “They insulted me” | The reason why the message was reported . |

### 2.1.4 SQL Statements

|  |  |
| --- | --- |
| Statement | Use |
| SELECT \* FROM User | Extracts all users in order to verify logins. |
| SELECT \* FROM Message limit ? offset (SELECT count(\*) FROM Message)-? | Selects x most recent message from the database where ? is the amount to select. |
| SELECT name FROM User WHERE user\_id == ? | Gets the username of a user from their id. |
| SELECT name FROM User WHERE name == ? | Used to check if a user with a certain name exists, returns null if they do not. |
| SELECT user\_id, name, admin FROM User WHERE name != 'SERVER' | Gets a list of all registered users apart from the ‘SERVER’ user as that is not a true user. |
| SELECT user\_id, admin FROM User WHERE name == ? | Gets a user’s user id and whether they’re an admin from their name. |
| SELECT count(\*) FROM Message WHERE sender\_id == ? | Returns the amount of messages a specific user has sent. |
| SELECT \* FROM Flag WHERE reportedUser\_id == ? | Selects all the reports corresponding to a specific user. |
| SELECT contents FROM Message WHERE message\_id == ? | Gets the contents of a message from its id. |
| INSERT into User(name, password) values (?, ?) | Creates a new non admin user, as ‘admin’ has a default value of False. |
| INSERT into Message(sender\_id, contents, timeSent, recipient\_id, colour) values (?,?,?,?,?) | Creates a new message. |
| SELECT last\_insert\_rowid() | Gets the Id (primary key) of the most recent insert. |
| SELECT sender\_id FROM Message where message\_id == ? | Gets the sender id of a message from the message Id. |
| INSERT into Flag(reportedUser\_id, message\_id, reporter\_id, reportReason) values (?,?,?,?) | Creates a new message report. |
| UPDATE Message SET contents=? WHERE message\_id=? | Changes a message’s contents by Id. |
| UPDATE Message SET contents=?,sender\_id=?,colour=? WHERE message\_id=? | Changes a message’s contents, who sent it and its colour. Used to ‘delete’ messages. |
| UPDATE User SET admin=? WHERE user\_id=? | Sets a users’ privelidges. |

### 2.1.5 Data Flow Diagram

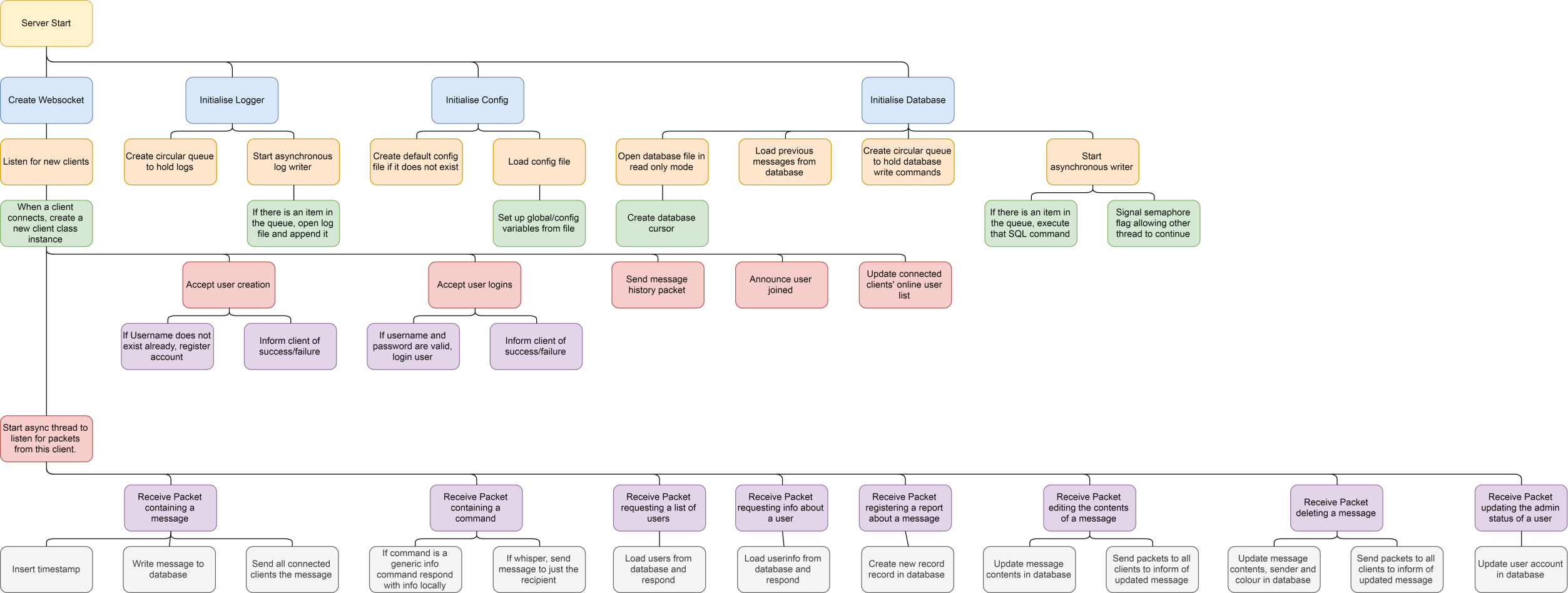


## 2.2 System Design

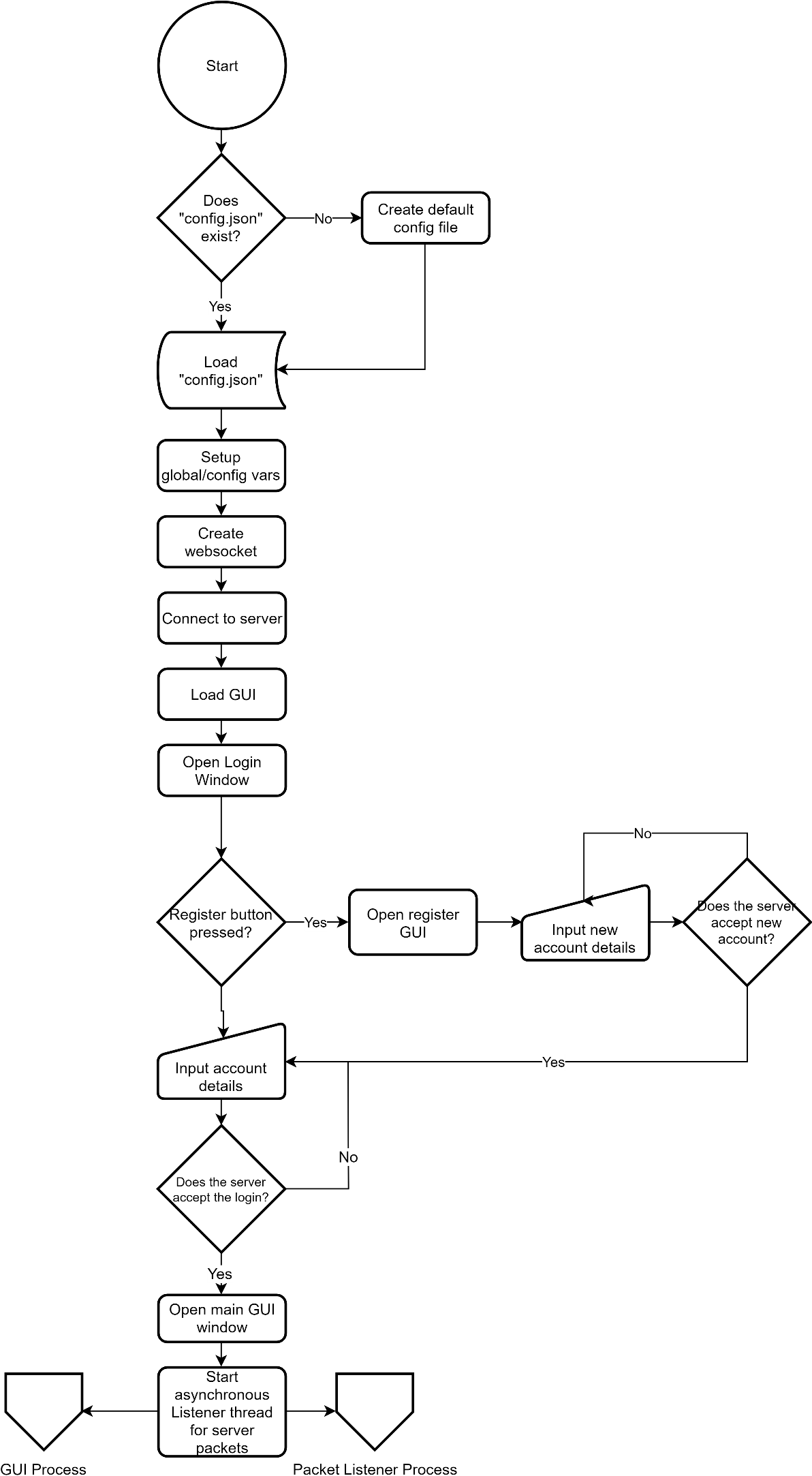
### 2.2.1 IPSO Chart

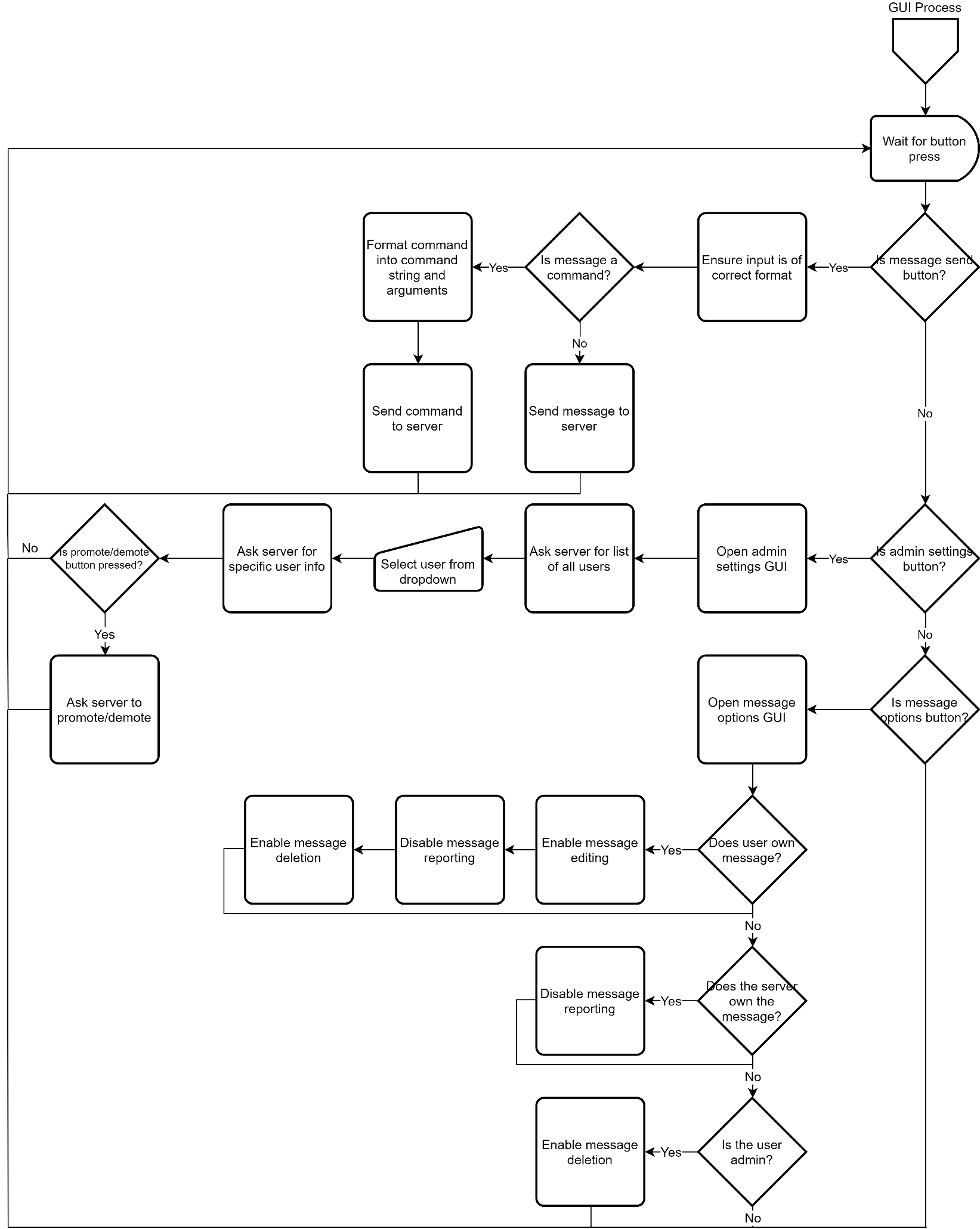
|  |  |  |
| --- | --- | --- |
| IPSO | Program Section | Item |
| Input | Registering User | Username  Password  Confirm Password |
| Logging in | Username  Password |
| Sending Message | Message Contents |
| Reporting User | Report Reason |
| Processing | Registering User | Check if username is available  Check passwords match |
| Logging in User | Check username and passwords match |
| Sending Message | Check if command  Insert username  Insert timestamp  Attach colour  Designate recipients  Send to other clients |
| Process command | Determine if valid command  Generate corresponding response  Return response to correct recipients |
| Reporting User | Attach message id  Attach sender id |
| Storage | Register User | Create new record in User table in database |
| Sending Message | Create new record In Message table in database |
| Reporting User | Create new record in Flag table in database |
| Promoting user to admin | Update user record in User database table |
| Editing message | Edit message record in Message database table |
| Deleting message | Edit message record in Message database table |
| Output | Display Message | Create new message widget  Update text elements |
| Display Online Users | Update online users widget |

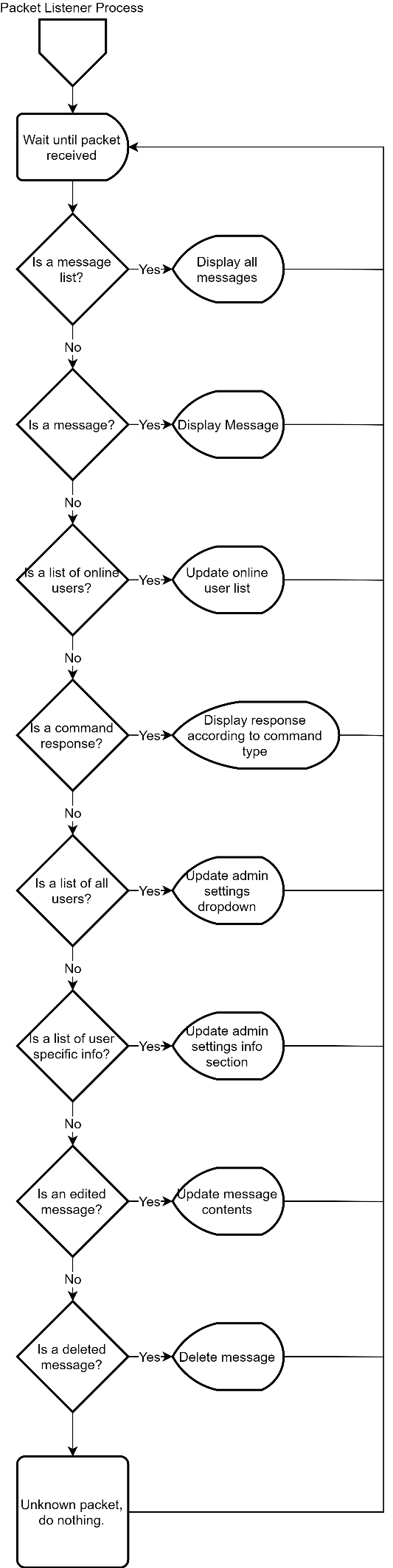
### N:\A2\NEA\Writeup\Diagrams\top-down-client.png2.2.2 Top-down diagrams



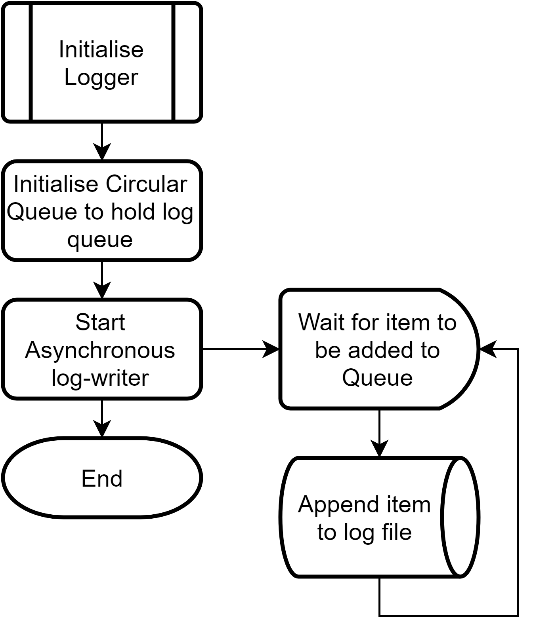
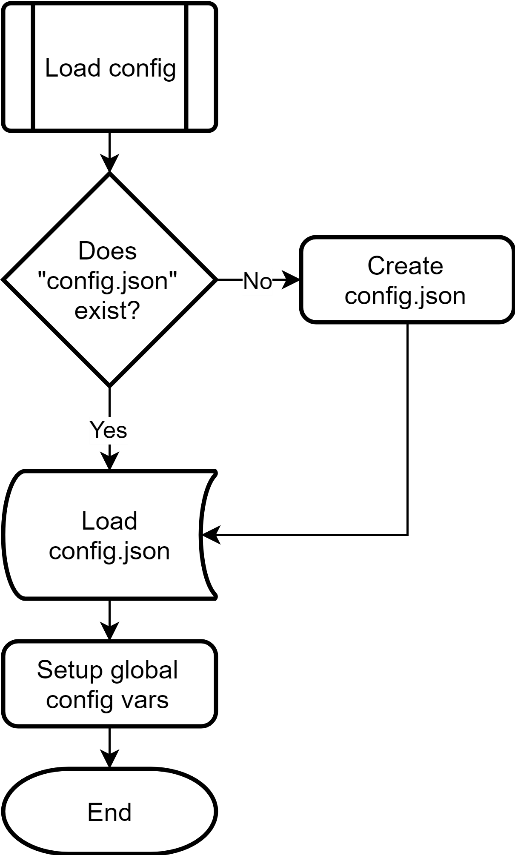
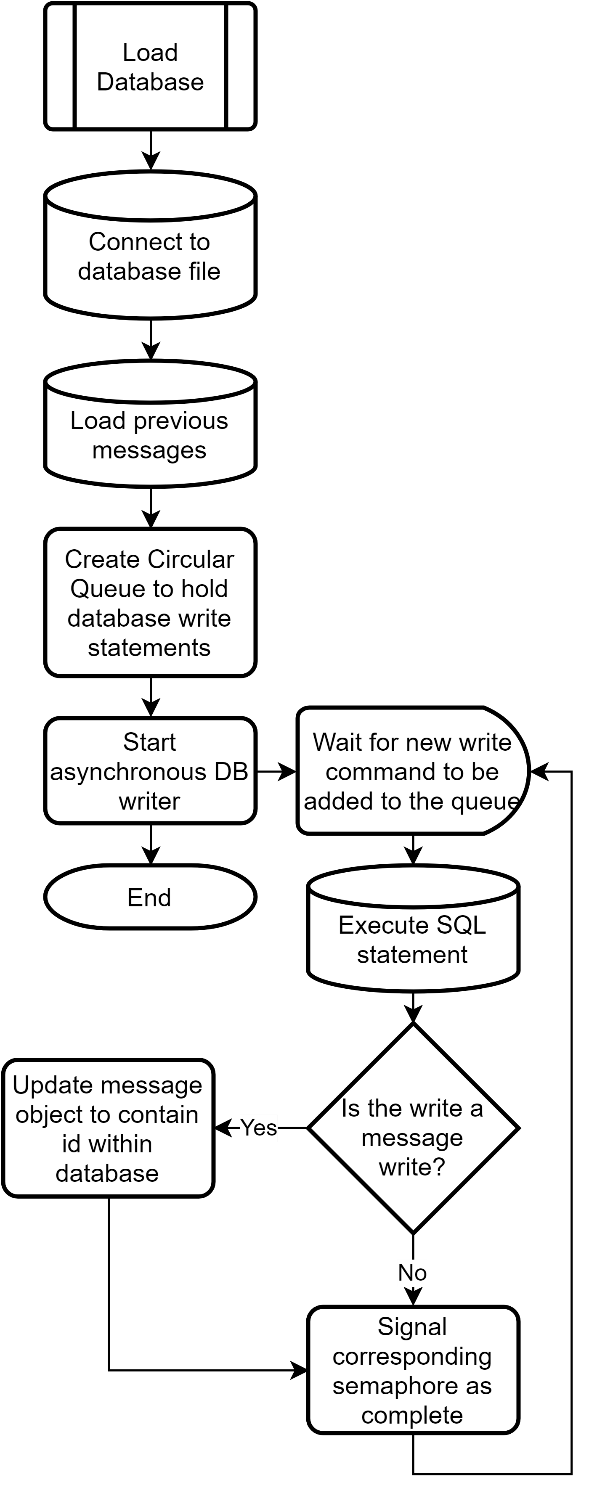
### 2.2.3 Client Flowchart

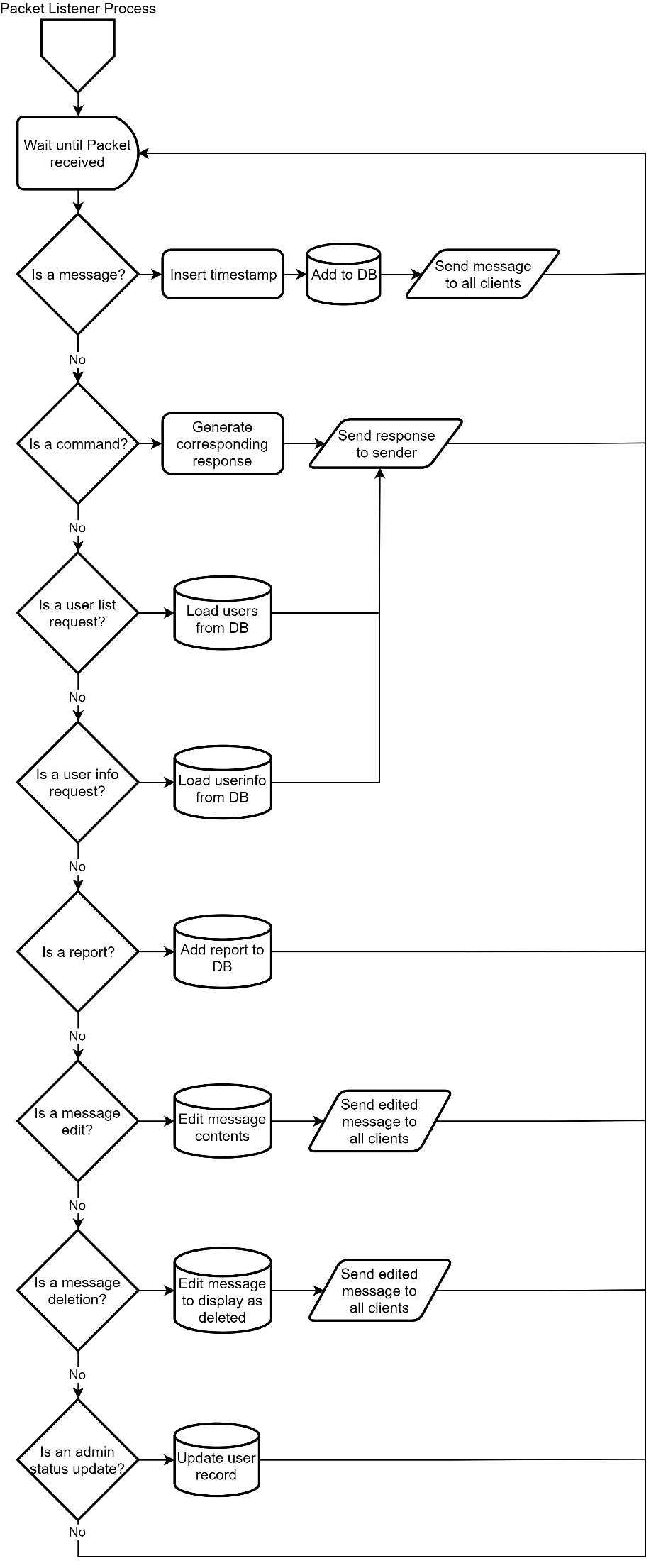






### 2.2.4 Server Flowchart





## 2.3 Data Structures & Dictionary

### 2.3.1 Circular Queue

Both the database writer and logger will operate using a circular queue to ensure there is never an error due to two threads attempting to write at once. The queue will hold each write to be made chronologically, with the oldest item at the front.

Variables will be needed for the FrontPointer, RearPointer and Length in order to keep track of the size. Custom methods to EnQueue and DeQueue will be written rather than accessing the data array directly.

|  |  |  |
| --- | --- | --- |
| Index | Data Example |  |
| 0 | **INSERT into Message ...** |  |
| 1 | **Update User ...** |  |
| 2 |  |  |
| 3 |  |  |
| ⋮ |  |  |
| n |  |  |

### 2.3.2 Client Data Dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| Data Item | Data Type | Validation | Sample Data/Info |
| \_app | QApplication |  | The instance of the PyQy app |
| \_mainGui | MainWindow |  | The main GUI window |
| \_serverSocket | socket |  | The websocket that all server connections will go through |
| \_username | String | Not Null | “admin” |
| \_userId | Integer | Not Null | 1 |
| \_admin | Boolean | Not Null | True |
| NONPRINTINGCHAR | Char (1 length string) | ==“\u200B” | “\u200B” |
| MAXTRANSMISSIONSIZE | Integer | Not Null | 40960 |
| COMMANDCHAR | Char (1 length string) | Not Null | “/” |
| DEBUG | Boolean | Not Null | False |
| New message | String | Not Null | “Hello World” |
| New Username | String | Not Null | “Sam” |
| New Password | String | Not Null | “secret” |
| Confirm Password | String | == New Password | “secret” |
| Username Login | String | Not Null | “User” |
| Password Login | String | Not Null | “shhhhh” |
| Report User | String | Not Null | “Said a rude word” |
| Edit Message | String | Not Null | “Hello World!” |

### 2.3.3 Server Data Dictionary

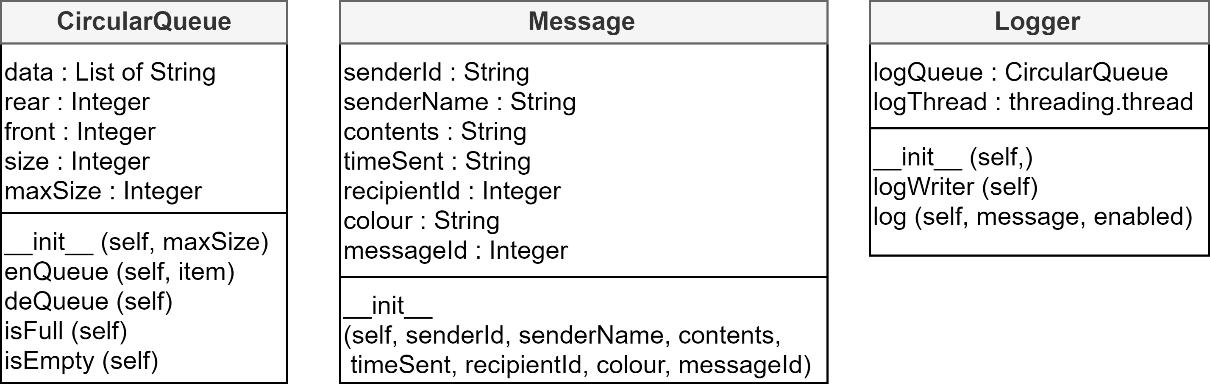
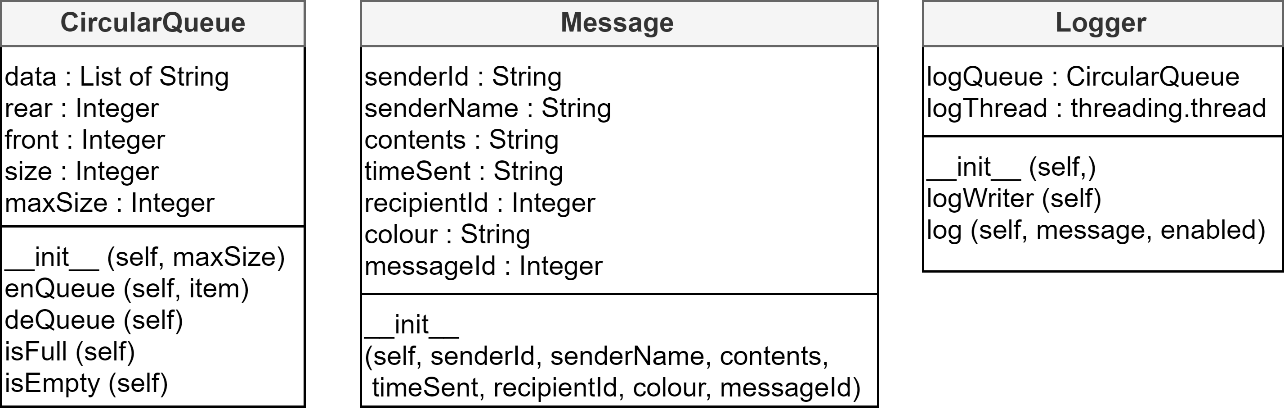
|  |  |  |  |
| --- | --- | --- | --- |
| Data Item | Data Type | Validation | Sample Data/Info |
| \_clients | List of Client | Not Null | A list containing all connected clients |
| \_database | Database | Not Null | Database management class |
| \_logger | Logger | Not Null | Logger Class |
| \_configManager | ConfigManager | Not Null | Config Manager Class |
| INFOLOGGINGENABLED | Boolean | Not Null | False |
| MAXTRANSMISSIONSIZE | Integer | Not Null | 40960 |

## 2.4 Class Design

### 2.4.1 Client Classes

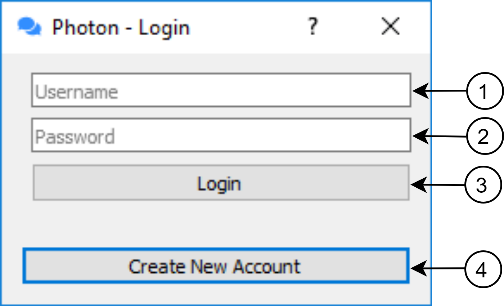
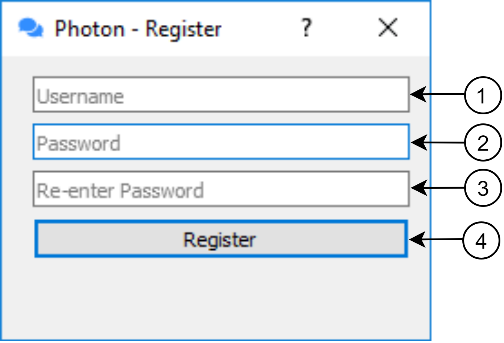
### 2.4.2 Server Classes

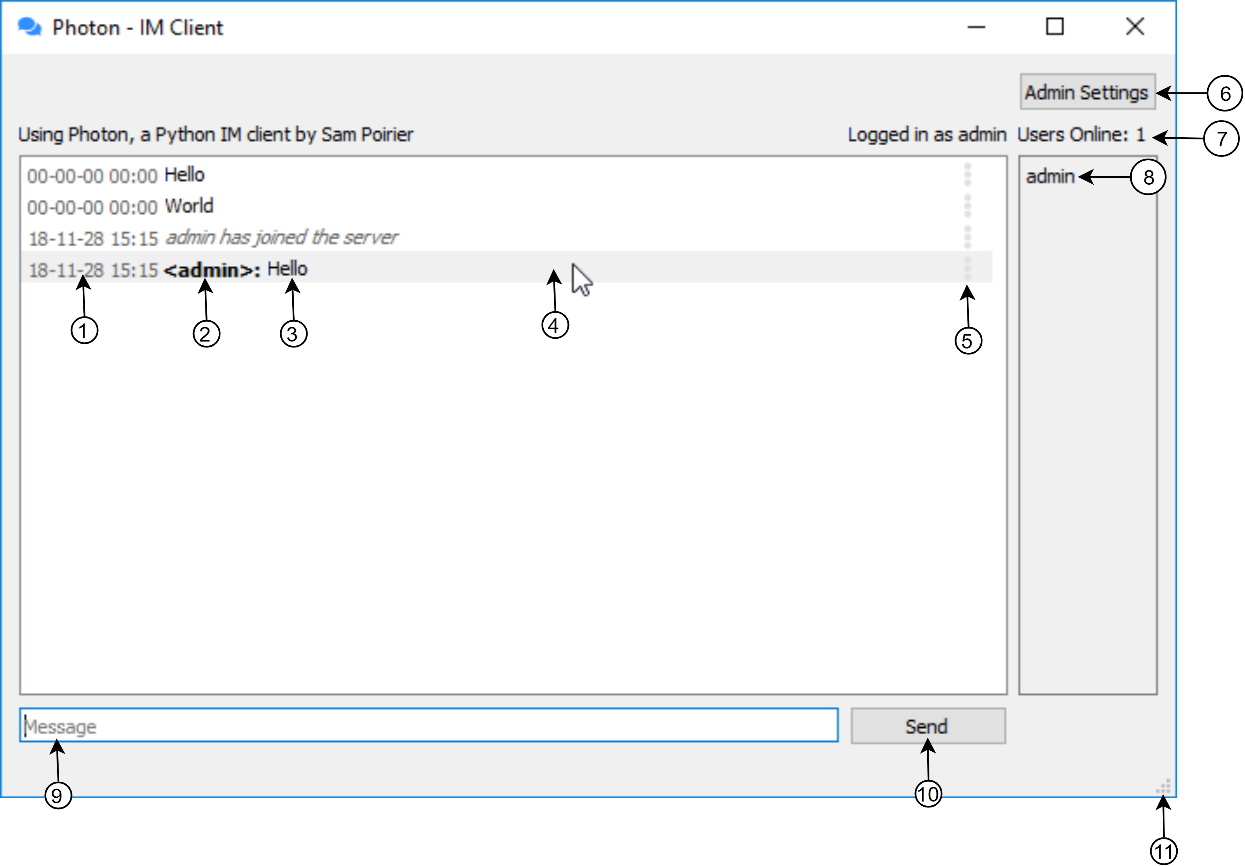
### 2.4.3 Generic Classes



## 2.5 User Interface Design

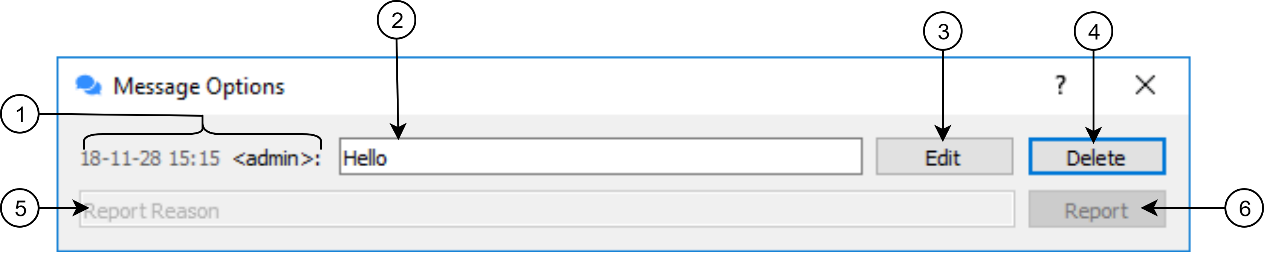
1. Input for the username of the account to login to.
2. Input for the password of the account. Displays text as ‘•’ characters.
3. Login button sends login request to server, taking the input from the text boxes.
4. Create new account button opens up the register window, allowing users to create a new account.

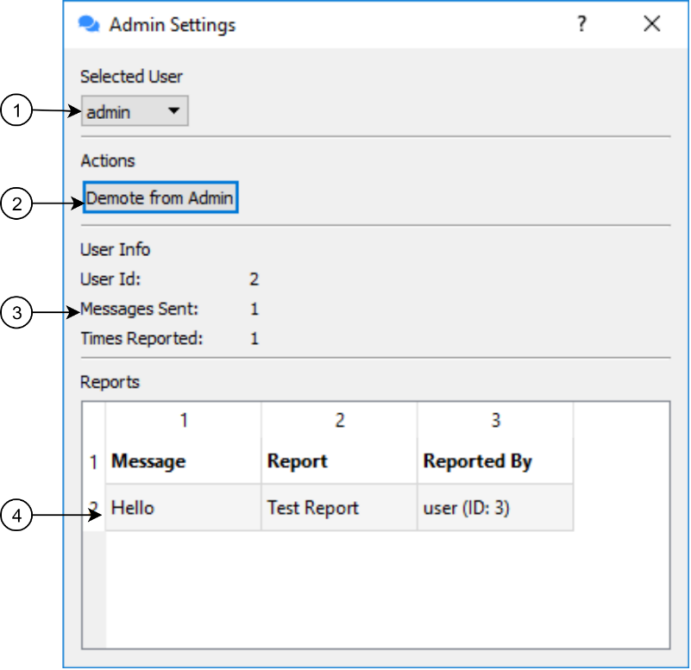


1. Input for the username of the new account.
2. Input for the password of the new account. Displays text as ‘•’ characters.
3. Input for the confirm password of the new account. Displays text as ‘•’ characters. Must equal the box above.
4. Submits the account details to the server, attempting to create a new account
5. Label to display message timestamp.
6. Label to display message sender username.

Part of the message widget

1. Label to display actual message contents.
2. When hovered, the message widget darkens for ease of reading.
3. Button to open message options widget.
4. Button to open admin settings window. Only visible if logged in as admin.
5. Label to display count of online users.
6. Multiline label to display list of online users.
7. Text input to write new messages in.
8. Button to send messages. Also empties Message text input.
9. Window is fully resizable, with all elements scaling properly.



1. Timestamp & Sender display labels.
2. Text input to edit messages. Only enabled if user sent the message.
3. Submits message edit. Only enabled if user sent the message.
4. Deletes message. Only enabled if admin or sender of message.
5. Text input to register a report. Disabled on own or server messages.
6. Button to submit report.
7. Dropdown contains all registered users. Used to select user to edit.
8. Promotes/Demotes a user to/from admin.
9. Labels to display statistics/information about the selected user.
10. If a user has been reported, reports are loaded into this table for admins to look over.

## Algorithms

### Circular Queue

The Circular Queue Class is used for the database write queue and the logger in the server file.

1. **class** CircularQueue():
2. """
3. A custom circular queue implementation.
4. """
5. **def** \_\_init\_\_(self, maxSize):
6. **if** maxSize < 1:
7. **raise** ValueError("Queue size must be at least 1")
8. self.data = [''] \* maxSize
9. self.rear = -1
10. self.front= 0
11. self.size = 0
12. self.maxSize = maxSize
14. **def** enQueue(self, item):
15. """
16. Add an item to the queue.
18. Args:
19. item (\*): Item to add to the queue.
20. """
21. **if** self.size == self.maxSize:
22. **raise** ValueError("Cannot enqueue when the queue is full")
23. **else**:
24. self.rear = (self.rear + 1) % self.maxSize
25. self.data[self.rear] = item
26. self.size = self.size + 1
28. **def** deQueue(self):
29. """
30. Removes an item from the queue.
32. Returns:
33. (\*): The item at the front of the queue.
34. """
35. self.front += 1
36. self.size -= 1
37. **return** self.data[self.front-1]
39. **def** isFull(self):
40. """ Determines if the queue is full. """
41. **return** self.size == self.rear
43. **def** isEmpty(self):
44. """ Determines if the queue is empty. """
45. **return** self.size == 0

### Hashing Algorithm

The hashing algorithm is one-directional and is used for encrypting passwords before storing them in the database.

1. **def** hashString(string):
2. """
3. Custom implementation of a simple one way hashing algorithm
5. Args:
6. string (string): The string to hash.
8. Returns:
9. (string): The hashed string.
10. """
11. bitValueChunk = ""
12. bitSum = 0
14. **for** char **in** string:
15. bitSum += ord(char)
16. bitValue = format(ord(char), 'b') # Convert char to binary
17. bitValueChunk += bitValue # Append to 'binary chunk'
19. n = 9
20. bitValues = [bitValueChunk[i:i+n] **for** i **in** range(0, len(bitValueChunk), n)] # Split 'binary chunk' into list of 9 bit binary numbers
22. moddedBitChunk = ""
23. **for** bitValue **in** bitValues:
24. bitValue = int(bitValue)
25. moddedBitValue = bitValue + (bitValue % 37) # Modulo is a one way function, so we modulo by a prime as the core of the hash. This is the step that ensures the hash is unidirectional
26. moddedBitValue = moddedBitValue \* bitSum # Multiply by sum of the ascii values of the chars to ensure similar input strings look different
27. moddedBitChunk += format(moddedBitValue, 'b') # Convert into binary again
29. n = 6 # 6 bit chunks to avoid strange characters
30. moddedBitValues = [moddedBitChunk[i:i+n] **for** i **in** range(0, len(moddedBitChunk), n)] # Split modded 'binary chunk' into list of 8 bit binary numbers
32. hashed = ""
33. **for** moddedBitValue **in** moddedBitValues:
34. hashed += chr(int(moddedBitValue, 2) + 33) # Convert binary value into decimal value then into the corresponding character, skipping the first 33 as they are non-printing/whitespace
36. **return** hashed

### Integer Merge Sort

The integer merge sort is not actually used anywhere in the program, but was written as a prototype in order to get the string merge sort working, as they are very similar algorithms.

1. **def** integerMergeSort(mergelist):
2. """
3. Custom implementation of a mergesort algorithm for integers.
5. Args:
6. mergeList (list of int): The list to sort.
8. Returns:
9. (list of int): The sorted list.
10. """
11. **if** len(mergelist) > 1:
12. mid = len(mergelist) // 2 # Perform integer division
13. lefthalf = mergelist[:mid] # Left half of merglist into lefthalf
14. righthalf = mergelist[mid:] # Right half of merglist into righthalf
15. lefthalf = MergeSort(lefthalf)
16. righthalf = MergeSort(righthalf)
18. i = 0
19. j = 0
20. k = 0
21. **while** i < len(lefthalf) **and** j < len(righthalf):
22. **if** lefthalf[i] < righthalf[j]:
23. mergelist[k] = lefthalf[i]
24. i += 1
25. **else**:
26. mergelist[k] = righthalf[j]
27. j += 1
28. k += 1
30. # Check if left half has elements not merged
31. **while** i < len(lefthalf):
32. mergelist[k] = lefthalf[i] # If so, add to mergelist
33. i += 1
34. k += 1
35. # Check if right half has elements not merged
36. **while** j < len(righthalf):
37. mergelist[k] = righthalf[j] # If so, add to mergelist
38. j += 1
39. k += 1
40. **return** mergelist

### String Merge Sort

The String Merge Sort is used to sort the list of online users alphabetically.

1. **def** stringListMergeSort(mergelist):
2. """
3. Custom implementation of a mergesort algorithm for strings.
5. Args:
6. mergeList (list of string): The list to sort.
8. Returns:
9. (list of string): The sorted list.
10. """
11. **if** len(mergelist) > 1:
12. mid = len(mergelist) // 2 # Perform integer division
13. lefthalf = mergelist[:mid] # Left half of merglist into lefthalf
14. righthalf = mergelist[mid:] # Right half of merglist into righthalf
15. lefthalf = stringListMergeSort(lefthalf)
16. righthalf = stringListMergeSort(righthalf)
18. i = 0
19. j = 0
20. k = 0
21. **while** i < len(lefthalf) **and** j < len(righthalf):
22. l = 0
23. **while** ord(lefthalf[i][l]) == ord(righthalf[j][l]) **and** l < len(lefthalf[i])-1 **and** l < len(righthalf[j])-1: # If the charachers are the same, we must look at the next one until the end
24. l += 1
25. **if** ord(lefthalf[i][l]) < ord(righthalf[j][l]):
26. mergelist[k] = lefthalf[i]
27. i += 1
28. **else**:
29. mergelist[k] = righthalf[j]
30. j += 1
31. k += 1
33. # Check if left half has elements not merged
34. **while** i < len(lefthalf):
35. mergelist[k] = lefthalf[i] # If so, add to mergelist
36. i += 1
37. k += 1
38. # Check if right half has elements not merged
39. **while** j < len(righthalf):
40. mergelist[k] = righthalf[j] # If so, add to mergelist
41. j += 1
42. k += 1
43. **return** mergelist

# Technical Solution

## 3.1 Server

### 3.1.1 main.py

1. # Main Server file
3. **import** socket
4. **from** threading **import** \*
5. **import** select
6. **import** re
8. # Load classes and functions from shared libs
9. **import** sys
10. sys.path.insert(0, '../Libs')
11. **from** packets **import** \*
12. **from** photonUtilities **import** \*
13. **from** database **import** \*
14. **from** logger **import** \*
15. **from** configManager **import** \*


19. # Global Variables
21. \_clients  = []
22. \_database = None
23. \_logger = None
24. \_configManager = None
26. INFOLOGGINGENABLED = None
27. MAXTRANSMISSIONSIZE = None


31. # Classes
33. **class** Client:
34. """
35. Represents one client that is connected to the server. A new instance is made for each connection, each client only interacts with their corresponding thread.
37. Properties:
38. socket (socket.socket): The websocket that this client is connected through.
39. address (string): The IP of the connected client.
40. id (int): the id of the connected client (not constant between sessions).
41. thread (threading.thread): The asyncronous listener thread for this client.
42. username (string): The username of the user.
43. userid (int): The id of the user account (constant between sessions).
44. admin (bool): Denotes whether the user account is admin.
45. """
46. **def** \_\_init\_\_(self, clientSocket, clientAddress):
47. """
48. Initialises the connection process, starts asynchronous listener thread and handles login handshake and account creation
50. Args:
51. clientSocket (socket.socket): the websocket that the connection is handled over.
52. clientAddress (string, int): The IP and id of the connected client.
54. ToDo:
55. Break this up!
56. """
57. **try**:
58. **global** \_database, \_clients
59. self.socket = clientSocket
60. self.address = clientAddress[0]
61. self.id = clientAddress[1]
62. self.thread = None
63. self.username = "UNKNOWN"
64. self.userid = ""
65. self.admin = False
66. \_clients.append(self)
68. \_logger.log(f"Received a connection from {self.address}, id {self.id}", INFOLOGGINGENABLED)
70. # Client Login
71. loginInvalid = True
72. **while** loginInvalid:
73. loginRequestPacket = decode(self.socket.recv(MAXTRANSMISSIONSIZE)) # Wait for client login packet TODO: time this out
75. **if** loginRequestPacket.type == "CREATEUSER":
77. usernameExists = \_database.userExists(loginRequestPacket.username)
79. **if** usernameExists:
80. userRegistered = RegisterResponsePacket(False, "A user with that name already exists")
82. **else**:
83. \_database.addUser(loginRequestPacket.username, loginRequestPacket.password)
84. userRegistered = RegisterResponsePacket(True)
85. \_logger.log(f"Attempted to register user: {loginRequestPacket.username}. Successful: {userRegistered.valid}", INFOLOGGINGENABLED)
87. self.socket.send(encode(userRegistered))
89. **elif** loginRequestPacket.type == "LOGINREQUEST":
90. err = "Incorrect username or password"
91. **for** client **in** \_clients:
92. **if** client.username == loginRequestPacket.username:
93. valid = False
94. err = "That user is already logged in"
95. **break**
97. **else**:
98. ret = \_database.queryLogin(loginRequestPacket.username, loginRequestPacket.password) # Query credentials against database
99. valid = ret[0]
101. **if** **not** valid:
102. \_logger.log(f"Invalid login from: {self.address}, id {self.id} - {err}", INFOLOGGINGENABLED)
103. loginResponse = LoginResponsePacket(False, err) # Tell the client the login was invalid
104. self.socket.send(encode(loginResponse))
106. **else**:
107. \_logger.log(f"Valid login from: {self.address}, id {self.id}", INFOLOGGINGENABLED)
108. loginResponse = LoginResponsePacket(True, userId=ret[1], admin=ret[2]) # Tell the client the login was valid
109. self.socket.send(encode(loginResponse))
110. self.userid = ret[1]
111. self.username = loginRequestPacket.username
112. self.admin = ret[2]
113. loginInvalid = False
115. readyToListenPacket = decode(self.socket.recv(MAXTRANSMISSIONSIZE)) # Wait until the client is ready to receive packets
117. # Get as many previous messages as possible that will fit into the max transmision size
118. messagesToSend = []
119. newMessagesToSend = []
120. **for** message **in** reversed(\_database.messages): # reversed as we want the latest messages
121. **if** message.recipientId == 1 **or** message.senderId == self.userid **or** message.recipientId == self.userid:
122. newMessagesToSend.append(message)
123. **if** len(encode(newMessagesToSend)) >= MAXTRANSMISSIONSIZE - 200: # We need a buffer of ~200 as when we construct the class the encoded size increases
124. newMessagesToSend = messagesToSend
125. **break**
126. **else**:
127. messagesToSend = newMessagesToSend
128. newMessageListPacket = MessageListPacket(reversed(messagesToSend)) # Send the client the previous messages
129. self.socket.send(encode(newMessageListPacket))

132. newMessage = generateJoinLeaveMessage("joined", self.username)
133. newMessage = \_database.addMessage(newMessage)
134. announceUserPacket = MessagePacket(newMessage) # Client has joined message
135. sendToClients(announceUserPacket)
137. self.listenerThread = Thread(target=self.ListenForPackets) # Start thread to listen for packets from client
138. self.listenerThread.start()
140. sendOnlineUsersPacket() # Tell clients a new user has joined
142. **except** ConnectionResetError: # Lost connection with client
143. \_logger.log(f"Lost connection with: {self.address}, id {self.id}; closing connection", INFOLOGGINGENABLED)
145. self.socket.close() # Close socket
146. **for** i **in** range(0, len(\_clients)):
147. **if** \_clients[i].id == self.id:
148. **del** \_clients[i] # Delete class instance
149. **break**
151. **if** self.username != "UNKNOWN":
152. newMessage = generateJoinLeaveMessage("left", self.username)
153. newMessage = \_database.addMessage(newMessage)
154. announceUserPacket = MessagePacket(newMessage)
155. sendToClients(announceUserPacket)
156. sendOnlineUsersPacket() # Tell clients a user has left
158. **return** # Return from thread
159. **except** Exception as err:
160. reportError(err, \_logger)

163. **def** ListenForPackets(self):
164. """
165. Listens for all communication from client. Should be called asyncronously in it's own thread.
167. ToDo:
168. Break this up?
169. """
170. **try**:
171. **global** \_clients, \_database, \_logger
172. **while** True:
174. packet = decode(self.socket.recv(MAXTRANSMISSIONSIZE)) # Wait for message from client
175. **if** packet.type == "MESSAGE":
176. packet.message.timeSent = getDateTime() # Update the message with the time it was received
177. packet.message = \_database.addMessage(packet.message)
178. sendToClients(packet)
179. \_logger.log(f"{packet.message.senderName}: {packet.message.contents}", INFOLOGGINGENABLED)
181. **elif** packet.type == "COMMAND":
182. command = packet.command
183. args = packet.args
184. success = False
185. err = ""
186. response = ""
187. targetClient = self
188. \_logger.log(f"User {self.username}, {self.id} executed command {command} with args {args}", INFOLOGGINGENABLED)
190. **if** command == "help":
191. success = True
192. response = ["!\*Available Commands\*!",
193. ("help","provides a list of available commands"),
194. ("ping","pings the server"),
195. ("whisper <user> <message>","sends a direct message to <user>"),
196. ("markup","displays balsamiq markup syntax")
197. ]
199. **elif** command == "markup":
200. success = True
201. response = ["!\*Markup Syntax\*!",
202. "Formats can be combined and symbols can be escaped with '\\'",
203. ("bold", "\*example\*", "\\*example\*"),
204. ("italic", "\_example\_", "\\_example\_"),
205. ("strikethrough", "~example~", "\~example~"),
206. ("underline", "!example!", "\!example!")
207. ]
209. **elif** command == "ping":
210. success = True
211. response = "Pong!"
213. **elif** command == "whisper":
214. targetName = args[0]
215. **del** args[0]
216. **for** client **in** \_clients:
217. **if** client.username == targetName:
218. targetClient = client
219. success = True
220. message = "\_ (Whisper) " + " ".join(args) + "\_"
221. response = formatUsername(self.username) + message
222. newMessage = Message(self.userid, self.username, message, getDateTime(), targetClient.userid, INFO)
223. newMessage = \_database.addMessage(newMessage)
224. **break**
225. **else**:
226. err = f"Could not find user with name {targetName}"
228. **else**:
229. err = "Unrecognised command"
231. response = CommandResponsePacket(command, success, err, response, getDateTime())
232. self.socket.send(encode(response))
233. **if** targetClient != self:
234. targetClient.socket.send(encode(response))
236. **elif** packet.type == "REQUESTUSERLIST":
237. userlist = \_database.listUsers()
238. self.socket.send(encode(UserListPacket(userlist)))
240. **elif** packet.type == "REQUESTUSERINFO":
241. userinfo = \_database.getUserDetails(packet.user)
242. userInfoPacket = UserInfoPacket(userinfo[0], userinfo[1], userinfo[2], userinfo[3])
243. self.socket.send(encode(userInfoPacket))
245. **elif** packet.type == "REPORTPACKET":
246. \_database.addReport(packet.messageId, packet.reporterId, packet.reportReason)
247. \_logger.log(f"{self.username} registered report: {packet.reportReason}", INFOLOGGINGENABLED)
249. **elif** packet.type == "EDITMESSAGE":
250. \_database.editMessage(packet.messageId, packet.newContents)
251. i = 0
252. oldMessage = None
253. **for** message **in** \_database.messages:
254. **if** message.messageId == packet.messageId:
255. oldMessage = \_database.messages[i].contents
256. \_database.messages[i].contents = packet.newContents
257. **break**
258. i+=1
259. sendToClients(packet) # Tell clients that the message has been edited.
260. \_logger.log(f"{self.username} edited message from '{oldMessage}' to '{packet.newContents}'", INFOLOGGINGENABLED)
262. **elif** packet.type == "DELETEMESSAGE":
263. \_database.deleteMessage(packet.messageId)
264. i = 0
265. oldMessage = None
266. **for** message **in** \_database.messages:
267. **if** message.messageId == packet.messageId:
268. oldMessage = \_database.messages[i].contents
269. \_database.messages[i].contents = "\_message deleted\_"
270. \_database.messages[i].senderId = 1
271. \_database.messages[i].senderName = "SERVER"
272. \_database.messages[i].colour = INFO
273. **break**
274. i+=1
275. sendToClients(packet) # Tell clients that the message has been deleted.
276. \_logger.log(f"{self.username} deleted message: {oldMessage}", INFOLOGGINGENABLED)
278. **elif** packet.type == "SETADMINSTATUS":
279. \_database.setAdmin(packet.admin, packet.userId)
280. \_logger.log(f"User id {packet.userId} set to admin: {packet.admin}", INFOLOGGINGENABLED)
282. **else**:
283. \_logger.log(f"Unknown packet received: {packet.type}", INFOLOGGINGENABLED)
285. **except** ConnectionResetError: # Lost connection with client
286. \_logger.log(f"Lost connection with: {self.address}, id {self.id}; closing connection", INFOLOGGINGENABLED)
288. self.socket.close() # Close socket
289. **for** i **in** range(0, len(\_clients)):
290. **if** \_clients[i].id == self.id:
291. **del** \_clients[i] # Delete class instance
292. **break**
294. newMessage = generateJoinLeaveMessage("left", self.username)
295. newMessage = \_database.addMessage(newMessage)
296. announceUserPacket = MessagePacket(newMessage)
297. sendToClients(announceUserPacket)
298. sendOnlineUsersPacket() # Tell clients a user has left
300. **except** Exception as err:
301. reportError(err, \_logger)


305. # Functions
307. **def** sendToClients(packet):
308. """
309. Sends a packet to every connected client.
311. Args:
312. packet (packets.packet): The packet instance to send to the clients.
313. """
314. **try**:
315. **global** \_clients
316. **for** client **in** \_clients:
317. client.socket.send(encode(packet))
318. **except** ConnectionResetError: # Client has lost connection
319. **pass**
320. **except** Exception as err:
321. reportError(err, \_logger)

324. **def** sendOnlineUsersPacket():
325. """
326. Constructs and sends a packet containing all users that are online.
327. """
328. **try**:
329. **global** \_clients
330. users = []
331. **for** client **in** \_clients:
332. users.append(client.username)
333. users = stringListMergeSort(users)
334. onlineUsersPacket = OnlineUsersPacket(users)
335. sendToClients(onlineUsersPacket)
336. **except** Exception as err:
337. reportError(err, \_logger)

340. **def** \_\_main\_\_():
341. """
342. The main body of the program; it all starts here.
343. Loads database, then listens for connections and assigns them new threads as they come in.
344. """
345. **try**:
346. **global** \_database, \_logger, INFOLOGGINGENABLED, MAXTRANSMISSIONSIZE
347. \_logger = Logger()
349. \_configManager = ServerConfig("config.json")
350. INFOLOGGINGENABLED = \_configManager.data["infoLoggingEnabled"]
351. MAXTRANSMISSIONSIZE = \_configManager.data["maxTransmissionSize"]
353. \_logger.log("Server started up", INFOLOGGINGENABLED)
355. \_logger.log("Loading Database...", INFOLOGGINGENABLED)
356. \_database = Database(\_configManager.data["dbFile"])
357. \_database.loadMessages()
358. # Create a socket object
359. serverSocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)
361. # Get local machine name and assign a port
362. host = socket.gethostname()
363. port = \_configManager.data["port"]
365. # Bind to the port
366. serverSocket.bind((host, port))
368. # Queue up to 5 requests
369. serverSocket.listen(5)
370. \_logger.log("Listening for connections...", INFOLOGGINGENABLED)
372. **while** True:
373. # Wait for connections
374. clientSocket, clientAddress = serverSocket.accept()
375. newClient = Client(clientSocket, clientAddress)
376. **except** Exception as err:
377. reportError(err, \_logger)


381. **if** \_\_name\_\_ == "\_\_main\_\_":
382. \_\_main\_\_()

### 3.1.2 logger.py

1. **from** threading **import** \*
3. # Load classes and functions from shared libs
4. **import** sys
5. sys.path.insert(0, '../Libs')
6. **from** photonUtilities **import** \*
8. **class** Logger:
9. """
10. A simple logger to log server actions.
12. Attributes:
13. logQueue (photonUtilities.CircularQueue): The queue used for log write commands in the logWriter() method.
14. logThread (threading.Thread): The separate thread started for the logger.
15. """
16. **def** \_\_init\_\_(self):
17. """" Initialises the logger. """
18. self.logQueue = CircularQueue(999)
19. self.logThread = Thread(target=self.logWriter)
20. self.logThread.start()
22. **def** logWriter(self):
23. """
24. Writes all strings in the queue sequentially.
25. Should be run asynchronously.
26. """
27. **while** True:
28. **if** **not** self.logQueue.isEmpty():
29. with open("log.txt", "a+") as logFile:
30. logFile.write(self.logQueue.deQueue())
32. **def** log(self, message, enabled=True):
33. """ Appends a string to the queue in the correct format. """
34. **if** enabled:
35. message = formatDateTime(getDateTime()) + message
36. self.logQueue.enQueue(message + "\n")
37. **print**(message)

### 3.1.3 database.py

1. **import** sqlite3
2. **from** threading **import** \*
4. # Load classes and functions from shared libs
5. **import** sys
6. sys.path.insert(0, '../Libs')
7. **from** photonUtilities **import** \*
9. **class** Database:
10. """
11. Contains all methods relating to reading and writing from the database.
13. Attributes:
14. roConnection (sqlite3.Connection): The main read only database connection. Used for most get functions.
15. roCursor  (sqlite3.Cursor): The cursor for the main read only connection.
16. writeQueue (photonUtilities.CircularQueue): The queue used for database write commands in the dbWriter() method.
17. writeThread (threading.Thread): The separate thread started for the database writer.
18. """
19. **def** \_\_init\_\_(self, file):
20. """ Initialises the database by creating a read only connection and starting the asyncronous writer function """
21. **try**:
22. self.roConnection = sqlite3.connect(f"file:{file}?mode=ro", uri=True) # Load database from file in read only mode
23. self.roCursor = self.roConnection.cursor()
24. self.messages = self.loadMessages()
25. self.writeQueue = CircularQueue(999)
26. self.writeThread = Thread(target=self.dbWriter)
27. self.writeThread.start()
28. **except** Exception:
29. reportError()

32. **def** dbWriter(self):
33. """
34. Writes all SQL statements in the queue sequentially as writes to the database must be done one at a time.
35. Should be run asynchronously.
36. """
37. **try**:
38. **while** True:
39. **if** **not** self.writeQueue.isEmpty():
40. connection = sqlite3.connect("photon.db")
41. cursor = connection.cursor()
42. command = self.writeQueue.deQueue()
43. cursor.execute(command[0], command[1]) # Execute SQL command
44. connection.commit() # Save changes to DB
45. cursor.execute("SELECT last\_insert\_rowid()")
46. **if** len(command) == 4:
47. self.messages[command[3]].messageId = cursor.fetchall()[0][0]
48. command[2].release()  # Release semaphore flag so the client thread can continue
49. connection.close()
50. **except** Exception:
51. reportError()

54. **def** queryLogin(self, username, password):
55. """
56. Checks whether supplied login credenitals are valid.
58. Args:
59. username (string): the username of the user account to check.
60. password (string): the password of the user account to check. Should be a hash of the password, not the plaintext.
62. Returns:
63. (bool): True if login successful, False if unsuccessful.
64. """
65. **try**:
66. self.roCursor.execute("SELECT \* FROM User")
67. users = self.roCursor.fetchall()
68. **for** user **in** users: # [0]: id [1]: name [2]: password [3]: admin
69. **if** user[1] == username **and** user[2] == password:
70. **if** user[3] == 1:
71. **return** (True, user[0], True)
72. **else**:
73. **return** (True, user[0], False)
74. **return** (False)
75. **except** Exception:
76. reportError()

79. **def** loadMessages(self, count=510): # Load last x messages from database
80. """
81. Loads the most recent messages from the database, and stores them globally.
83. Args:
84. count (int, optional): the amount of messages to load from the database.
85. """
86. **try**:
87. constructedMessages = []
88. self.roCursor.execute("SELECT \* FROM Message limit ? offset (SELECT count(\*) FROM Message)-?", (str(count), str(count)))
89. messages = self.roCursor.fetchall()
90. **for** message **in** messages:
91. self.roCursor.execute("SELECT name FROM User WHERE user\_id == ?", (str(message[1]),))
92. username = self.roCursor.fetchall()[0][0]
93. constructedMessage = Message(messageId=message[0], senderId=message[1], senderName=username, contents=message[2], timeSent=message[3], recipientId=message[4], colour=message[5])
94. constructedMessages.append(constructedMessage)
95. **return** constructedMessages
96. **except** Exception:
97. reportError()

100. **def** userExists(self, username):
101. """
102. Check to see if a user with specific username exists.
104. Args:
105. username (string): the username of the user to lookup.
107. Returns:
108. (bool): True if the user exists, False if the user does not.
109. """
110. self.roCursor.execute("SELECT name FROM User WHERE name == ?", (username,))
111. **if** len(self.roCursor.fetchall()) > 0:
112. **return** True
113. **else**:
114. **return** False

117. **def** listUsers(self):
118. """
119. Gets a list of all registered users
121. Returns:
122. (list of (int, string, bool)): Returns a list of tuples conaining the userId, username and whether they're admin.
124. ToDo:
125. Limit the amount of users that can be fetched at once.
126. """
127. connection = sqlite3.connect("file:photon.db?mode=ro", uri=True)
128. cursor = connection.cursor()
129. cursor.execute("SELECT user\_id, name, admin FROM User WHERE name != 'SERVER'")
130. users = cursor.fetchall()
131. **return** users

134. **def** getUserDetails(self, user):
135. """
136. Gets the userId, message count, admin and reports for a specific user.
138. Args:
139. user (string): the username of the user who's info should be fetched.
141. Returns:
142. (int, int, bool list of (string, string, string, int)): The details of the user, corresponding to (user id, message count, whether admin, list of (reported message, report reason, reporter name, reporter id).
143. """
144. **try**:
145. connection = sqlite3.connect("file:photon.db?mode=ro", uri=True)
146. cursor = connection.cursor()
147. cursor.execute("SELECT user\_id, admin FROM User WHERE name == ?", (user,))
148. ret = cursor.fetchall()[0]
149. userId = ret[0]
150. admin = bool(ret[1])
151. cursor.execute("SELECT count(\*) FROM Message WHERE sender\_id == ?", (userId,))
152. messageCount = cursor.fetchall()[0][0]
153. cursor.execute("SELECT \* FROM Flag WHERE reportedUser\_id == ?", (userId,))
154. flagged = cursor.fetchall()
155. flags = []
156. **for** flag **in** flagged:
157. cursor.execute("SELECT contents FROM Message WHERE message\_id == ?", (flag[2],))
158. message = cursor.fetchall()[0][0]
159. cursor.execute("SELECT name FROM User WHERE user\_id == ?", (flag[3],))
160. reporterName = cursor.fetchall()[0][0]
161. flags.append((message, flag[4], reporterName, flag[3]))
163. **return** (userId, messageCount, admin, flags)
164. **except** Exception:
165. reportError()
167. **def** addUser(self, username, password):
168. """
169. Creates a new user entry in the database.
171. Args:
172. username (string): the username of the account to create.
173. password (string): the password of the account to create - this should be hashed.
174. """
175. semaphore = Semaphore(value=0) # Create a semaphore to be used to signal once the database write has been completed
176. self.writeQueue.enQueue(("INSERT into User(name, password) values (?, ?)", (username, password), semaphore))
177. semaphore.acquire() # Wait until semaphore has been released IE has db write is complete

180. **def** addMessage(self, message):
181. """
182. Creates a new message entry in the database.
184. Args:
185. message (Message): The instance of message class containing the information that needs to be written.
187. Returns:
189. """
190. semaphore = Semaphore(value=0) # Create a semaphore to be used to tell once the database write has been completed
191. self.messages.append(message)
192. messageIndex = len(self.messages) - 1
193. self.writeQueue.enQueue(("INSERT into Message(sender\_id, contents, timeSent, recipient\_id, colour) values (?,?,?,?,?)", (str(message.senderId), message.contents, message.timeSent, message.recipientId, message.colour), semaphore, messageIndex))
194. semaphore.acquire() # Wait until semaphore has been released IE has db write is complete
195. **return** self.messages[messageIndex] # Message object will have been updated with it's ID by the DB writer
197. **def** addReport(self, messageId, reporterId, reportReason):
198. connection = sqlite3.connect("file:photon.db?mode=ro", uri=True)
199. cursor = connection.cursor()
200. cursor.execute("SELECT sender\_id FROM Message where message\_id == ?", (messageId,))
201. reportedUserId = cursor.fetchall()[0][0]
202. semaphore = Semaphore(value=0) # Create a semaphore to be used to tell once the database write has been completed
203. self.writeQueue.enQueue(("INSERT into Flag(reportedUser\_id, message\_id, reporter\_id, reportReason) values (?,?,?,?)", (reportedUserId, messageId, reporterId, reportReason), semaphore))
204. semaphore.acquire() # Wait until semaphore has been released IE has db write is complete
206. **def** editMessage(self, messageId, newContent):
207. semaphore = Semaphore(value=0) # Create a semaphore to be used to tell once the database write has been completed
208. self.writeQueue.enQueue(("UPDATE Message SET contents=? WHERE message\_id=?", (newContent, messageId), semaphore))
209. semaphore.acquire() # Wait until semaphore has been released IE has db write is complete
211. **def** deleteMessage(self, messageId):
212. semaphore = Semaphore(value=0) # Create a semaphore to be used to tell once the database write has been completed
213. self.writeQueue.enQueue(("UPDATE Message SET contents=?,sender\_id=?,colour=? WHERE message\_id=?", ("\_message deleted\_", 1, INFO, messageId), semaphore))
214. semaphore.acquire() # Wait until semaphore has been released IE has db write is complete
216. **def** setAdmin(self, admin, userId):
217. semaphore = Semaphore(value=0) # Create a semaphore to be used to tell once the database write has been completed
218. self.writeQueue.enQueue(("UPDATE User SET admin=? WHERE user\_id=?", (admin, userId), semaphore))
219. semaphore.acquire() # Wait until semaphore has been released IE has db write is complete

### 3.1.4 config.json

1. {
2. "dbFile": "photon.db",
3. "infoLoggingEnabled": **true**,
4. "maxTransmissionSize": 40960,
5. "port": 9998
6. }

## 3.2 Client

### 3.2.1 main.py

1. # Main Client File
3. **import** sys
4. **import** socket
5. **import** pickle
6. **from** threading **import** Thread
7. **import** atexit
8. **import** os
9. **import** cgi
10. **import** time
11. **import** datetime
12. **import** re
14. **from** PyQt5.QtCore **import** pyqtSlot, pyqtSignal
15. **from** PyQt5.QtWidgets **import** QApplication, QMainWindow, QDialog, QMessageBox, QWidget, QFormLayout, QScrollArea, QTableWidgetItem
16. **from** PyQt5.QtGui **import** QColor
17. **from** PyQt5.uic **import** loadUi
19. # Load classes and functions from shared libs
20. **import** sys
21. sys.path.insert(0, '../Libs')
22. **from** packets **import** \*
23. **from** photonUtilities **import** \*
24. **from** configManager **import** \*


28. # Global vars
29. \_app = None
30. \_mainGui = None
31. \_serverSocket = None
32. \_username = ""
33. \_userId = None
34. \_admin = False
36. NONPRINTINGCHAR = '\u200B' # Used to replace a character in a string whilst keeping indexes the same
37. MAXTRANSMISSIONSIZE = None
38. COMMANDCHAR = None
39. DEBUG = None

42. # Classes
44. **class** MainWindow(QMainWindow):
45. """
46. GUI Class for main window. Inherits QMainWindow.
48. Elements:
49. centralWidget (QWidget): Main area containg window elements.
50. adminSettinsButton (QPushButton): Button to open admin settings.
51. messageInput (QLineEdit): Input area for sending messages.
52. messageInputButton (QPushButton): Send button for messages.
53. messageScrollArea (QScrollArea): Window section containing message elements.
54. messageWidget (QWidget): Widget to contain message elements.
55. userCountLabel (QLabel): Displays count of online users.
56. userListBox (QTextEdit): Contains all online users.
57. usernameLabel (QLabel): Displays username.
59. Properties:
60. writeSignal (QSignal): Signal to trigger message creation.
61. usersChangedSignal (QSignal): Signal to update online users.
63. ToDo:
64. Seperate this class out!
65. """
67. # Signals for updating the GUI
68. writeSignal = pyqtSignal(Message)
69. usersChangedSignal = pyqtSignal(list)
70. updateMessageSignal = pyqtSignal(int, str)
71. deleteMessageSignal = pyqtSignal(int)
73. **def** \_\_init\_\_(self, \*args):
74. """ Initialises the UI and connects all signals and button clicks. """
75. **try**:
76. super().\_\_init\_\_(\*args)
77. loadUi("mainWindow.ui", self)
78. self.messageInputButton.clicked.connect(self.onSendClick)
79. self.messageInput.returnPressed.connect(self.onSendClick)
80. # Point the signals to the corresponding functions
81. self.writeSignal.connect(self.WriteLine)
82. self.usersChangedSignal.connect(self.UpdateConnectedUsers)
83. self.updateMessageSignal.connect(self.updateMessageContents)
84. self.deleteMessageSignal.connect(self.deleteMessage)
85. self.messageWidget.setLayout(self.messageLayout)
87. **except** Exception:
88. ReportError()

91. **def** setUsername(self, username):
92. """
93. Sets the username label text.
95. Args:
96. username (string): The username to set to.
97. """
98. **try**:
99. self.usernameLabel.setText(f"Logged in as {username}")
100. **except** Exception:
101. reportError()

104. **def** postLogin(self):
105. """ Setup for after login is completed. """
106. **if** **not** \_admin:
107. self.adminSettingsButton.hide()
108. **else**:
109. self.adminSettingsButton.clicked.connect(self.openAdminSettings)
111. **def** openAdminSettings(self):
112. """ Opens the admin settings UI. """
113. **try**:
114. self.adminSettings = AdminSettingsWindow(self)
115. self.adminSettings.show()
116. **except** Exception:
117. reportError()

120. **def** closeEvent(self, event):
121. """ When main window closed, tidy up loose ends and exit. """
122. **try**:
123. # Not using onProgramExit() as it caused the program to hang when the UI is created
124. **global** \_serverSocket
125. debugPrint("Window closed: Force closing all threads and server socket", DEBUG)
126. \_serverSocket.close()
127. os.\_exit(1)
128. **except** Exception:
129. reportError()

132. **def** WriteLine(self, message):
133. """
134. Formats message, creates a new element for it and displays it properly.
136. Args:
137. message (Message): The message to display
138. """
139. rawMessage = message.contents
140. **try**:
141. newWidget = MessageWidget(message=message) # Create a new message widget
142. newWidget.updateText()
143. rowCount = self.messageLayout.rowCount() # Get the amount of rows in the message container
144. self.messageLayout.setWidget(rowCount, QFormLayout.LabelRole, newWidget) # Append the new message widget to the end of the container
145. newWidget.setFixedWidth(self.messageScrollArea.width() - 10) # Set widget width to match the parent width
147. debugPrint(rawMessage, DEBUG)
148. \_app.alert(\_mainGui, 1000) # Flash the taskbar icon for 1 second
149. **except** Exception:
150. reportError()

153. **def** UpdateConnectedUsers(self, userList):
154. """
155. Update connected users GUI elements.
157. Args:
158. userList (list of string): List of online users.
159. """
160. userCount = len(userList)
161. self.userCountLabel.setText(f"Users Online: {userCount}")
163. self.userListBox.clear()
164. **for** user **in** userList:
165. self.userListBox.insertHtml(f"{user} <br>")

168. **def** onSendClick(self):
169. """ When the send button is pressed, take input, parse it and then send it. """
170. **try**:
171. text = self.messageInput.text()
172. **if** **not** text.isspace() **and** text != "":
173. **if** text[0] == COMMANDCHAR:
174. ParseCommand(text)
175. **else**:
176. SendMessage(text)
177. self.messageInput.setText("")
178. **except** Exception:
179. reportError()

182. **def** resizeEvent(self, event):
183. """ Overrides the window resize event. Updates widget sizes. """
184. width = self.messageScrollArea.width() - 10
185. messageWidgets = (self.messageLayout.itemAt(i) **for** i **in** range(self.messageLayout.count())) # Get a list of widgets in layout
187. **for** layoutItem **in** messageWidgets:
188. layoutItem.widget().setFixedWidth(width)

191. **def** updateMessageContents(self, messageId, contents):
192. messageWidgets = (self.messageLayout.itemAt(i) **for** i **in** range(self.messageLayout.count())) # Get a list of widgets in layout
194. **for** layoutItem **in** messageWidgets:
195. **if** layoutItem.widget().message.messageId == messageId:
196. layoutItem.widget().message.contents = contents
197. layoutItem.widget().updateText()
199. **def** deleteMessage(self, messageId):
200. messageWidgets = (self.messageLayout.itemAt(i) **for** i **in** range(self.messageLayout.count())) # Get a list of widgets in layout
202. **for** layoutItem **in** messageWidgets:
203. **if** layoutItem.widget().message.messageId == messageId:
204. layoutItem.widget().message.contents = "\_message deleted\_"
205. layoutItem.widget().message.senderId = 1
206. layoutItem.widget().message.senderName = "SERVER"
207. layoutItem.widget().message.colour = INFO
208. layoutItem.widget().updateText()


212. **class** MessageWidget(QWidget):
213. """ Gui Class for each message. """
214. **def** \_\_init\_\_(self, parent=None, message=""):
215. super().\_\_init\_\_(parent)
216. loadUi("message.ui", self)
217. self.messageOptionBtn.clicked.connect(**lambda**: self.openMessageOptions())
218. self.message = message
219. self.setMouseTracking(1)
220. hoverColour = QColor("#f0f0f0")
221. self.hoverPalette = self.palette()
222. self.hoverPalette.setColor(self.backgroundRole(), hoverColour)
223. baseColour = QColor("#ffffff")
224. self.basePalette = self.palette()
225. self.basePalette.setColor(self.backgroundRole(), baseColour)
227. **def** openMessageOptions(self):
228. """ Opens UI for managing messages. """
229. **try**:
230. self.messageOptions = MessageOptions(self, message=self.message)
231. self.messageOptions.show()
232. **except** Exception:
233. reportError()
235. **def** updateText(self):
236. self.timeLabel.setText(self.message.timeSent)
237. self.usernameLabel.setText(formatUsername(self.message.senderName))
238. self.messageLabel.setText(formatTextForDisplay(self.message.contents, self.message.colour))
240. **def** enterEvent(self, event):
241. self.setPalette(self.hoverPalette)
243. **def** leaveEvent(self, event):
244. self.setPalette(self.basePalette)
246. **class** MessageOptions(QDialog):
247. """
248. GUI Class for managing messages
250. Args:
251. timestamp (string): Time that the message was sent.
252. user (string): Username of user who sent the message.
253. message (string): The contents of the message being managed.
254. """
255. **def** \_\_init\_\_(self, \*args, message=""):
256. **try**:
257. super().\_\_init\_\_(\*args)
258. loadUi("messageOptions.ui", self)
259. self.reportButton.clicked.connect(**lambda**: self.sendReport())
260. self.editButton.clicked.connect(**lambda**: self.editMessageContents())
261. self.deleteButton.clicked.connect(**lambda**: self.deleteMessage())
262. self.message = message
263. self.timeLabel.setText(message.timeSent)
264. self.usernameLabel.setText(formatUsername(message.senderName))
265. self.editMessage.setText(message.contents)
267. # Users can only edit and delete their own messages. Admins can delete any message. You cannot report your own or a server message. Local messages cannot be modified.
268. **if** self.message.senderId != \_userId **or** self.message.senderId == "":
269. self.editMessage.setEnabled(False)
270. self.editButton.setEnabled(False)
271. **if** **not** \_admin **or** self.message.senderId == "":
272. self.deleteButton.setEnabled(False)
274. **if** self.message.senderId == 1 **or** self.message.senderId == \_userId **or** self.message.senderId == "":
275. self.reportButton.setEnabled(False)
276. self.reportReason.setEnabled(False)
278. **except** Exception:
279. reportError()
281. **def** sendReport(self):
282. reason = self.reportReason.text()
283. **if** len(reason) < 1:
284. errNotifier = QMessageBox()
285. errNotifier.setIcon(QMessageBox.Critical)
286. errNotifier.setText("You must supply a reason when reporting a message.")
287. errNotifier.setWindowTitle("Report Error")
288. errNotifier.exec\_()
289. **else**:
290. reportPacket = ReportPacket(self.message.messageId, \_userId, reason)
291. \_serverSocket.send(encode(reportPacket))
292. successNotifier = QMessageBox()
293. successNotifier.setIcon(QMessageBox.Information)
294. successNotifier.setText("Successfully reported message.")
295. successNotifier.setWindowTitle("Report Result")
296. successNotifier.exec\_()
298. **def** editMessageContents(self):
299. editMessagePacket = EditMessagePacket(self.message.messageId, self.editMessage.text())
300. \_serverSocket.send(encode(editMessagePacket))
301. successNotifier = QMessageBox()
302. successNotifier.setIcon(QMessageBox.Information)
303. successNotifier.setText("Successfully edited message.")
304. successNotifier.setWindowTitle("Edit Result")
305. successNotifier.exec\_()
307. **def** deleteMessage(self):
308. deleteMessagePacket = DeleteMessagePacket(self.message.messageId)
309. \_serverSocket.send(encode(deleteMessagePacket))
310. successNotifier = QMessageBox()
311. successNotifier.setIcon(QMessageBox.Information)
312. successNotifier.setText("Successfully deleted message.")
313. successNotifier.setWindowTitle("Delete Result")
314. successNotifier.exec\_()
315. self.close()

318. **class** AdminSettingsWindow(QDialog):
319. """ GUI Class for the admin window. """
320. **def** \_\_init\_\_(self, \*args):
321. super().\_\_init\_\_(\*args)
322. loadUi("adminSettings.ui", self)
323. self.userListComboBox.currentIndexChanged.connect(self.ComboBoxUpdated)
324. self.toggleAdminStatusBtn.clicked.connect(self.toggleAdminStatus)
325. requestUserListPacket = Packet("REQUESTUSERLIST")
326. \_serverSocket.send(encode(requestUserListPacket))
327. self.selectedUserAdmin = None
328. self.selectedUserId = None
330. **def** UserListReceived(self, userList):
331. """
332. Populates the userList dropdown.
334. Args:
335. userList (list of string): The list of users.
336. """
337. **for** user **in** userList:
338. self.userListComboBox.addItem(user[1])
340. **def** ComboBoxUpdated(self):
341. """ Event for when a new user is selected from the combobox. Fetches information about that user. """
342. requestUserInfoPacket = RequestUserInfoPacket(self.userListComboBox.currentText())
343. \_serverSocket.send(encode(requestUserInfoPacket))
345. **def** UpdateUserInfo(self, userId, messageCount, admin, flags):
346. """
347. Updates the user specific info sections.
349. Args:
350. userId (int): The id of the selected user.
351. messageCount (int): The amount of messages sent by that user.
352. admin (bool): Whether the user is admin.
353. flags  (list of (string, string, string, int)): The report details of that user. Corresponding to list of (reported message, report reason, reporter name, reporter id)
354. """
355. self.userIdLabel.setText(str(userId))
356. self.selectedUserId = str(userId)
357. self.messageCountLabel.setText(str(messageCount))
358. self.reportCountLabel.setText(str(len(flags)))
360. rowPosition = self.reportTable.rowCount()
361. **for** row **in** range(0, rowPosition):
362. self.reportTable.removeRow(1)
364. self.selectedUserAdmin = admin
365. **if** admin:
366. self.toggleAdminStatusBtn.setText("Demote from Admin")
367. **else**:
368. self.toggleAdminStatusBtn.setText("Promote to Admin")
369. self.toggleAdminStatusBtn.setEnabled(True)
371. i = 1
372. **for** flag **in** flags:
373. self.reportTable.insertRow(i)
374. self.reportTable.setItem(i, 0, QTableWidgetItem(str(flag[0])))
375. self.reportTable.setItem(i, 1, QTableWidgetItem(str(flag[1])))
376. self.reportTable.setItem(i, 2, QTableWidgetItem(f"{flag[2]} (ID: {flag[3]})"))
377. i += 1
379. **def** toggleAdminStatus(self):
380. adminStatusPacket = SetAdminStatusPacket(**not** self.selectedUserAdmin, self.selectedUserId)
381. \_serverSocket.send(encode(adminStatusPacket))
382. self.ComboBoxUpdated()
384. **class** LoginWindow(QDialog):
385. """ GUI Class for login window """
386. **def** \_\_init\_\_(self, \*args):
387. **try**:
388. super().\_\_init\_\_(\*args)
389. loadUi("login.ui", self)
390. self.loginButton.clicked.connect(self.onLoginClick)
391. self.newAccButton.clicked.connect(self.openRegisterWindow)
392. self.usernameInput.returnPressed.connect(self.onLoginClick)
393. **except** Exception:
394. reportError()

397. **def** openRegisterWindow(self):
398. """ Opens register window GUI """
399. **try**:
400. registerGui = RegisterWindow()
401. registerGui.exec\_()
402. **except** Exception:
403. reportError()

406. **def** onLoginClick(self):
407. """ Called on login button click. Basic input validation. """
408. **try**:
409. **global** \_username
410. username = self.usernameInput.text()
411. password = self.passwordInput.text()
413. **if** **not** username.isspace() **and** username != "":
414. **if** **not** password.isspace() **and** password != "":
415. Password = password
416. self.Login(username, password)
418. **else**:
419. self.errLabel.setText("Passwords must not consist of whitespace only")
421. **else**:
422. self.errLabel.setText("Usernames must not consist of whitespace only")
423. **except** Exception:
424. reportError()

427. **def** Login(self, username, password):
428. """
429. Attempts to login user to server.
431. Args:
432. username (string): Username to login with.
433. password (string): Password to login with, not yet hashed.
434. """
435. **try**:
436. **global** \_serverSocket, \_username, \_userId, \_admin
437. password = hashString(password)
438. loginRequest = LoginRequestPacket(username, password)
439. \_serverSocket.send(encode(loginRequest))
441. loginResponsePacket = decode(\_serverSocket.recv(MAXTRANSMISSIONSIZE))
443. **if** loginResponsePacket.type != "LOGINRESPONSE":
444. self.Login(username, password) # Occasionally a left-over packet can make it's way here - if so we'll just try again
445. **return**
447. **if** loginResponsePacket.valid:
448. self.close()
449. \_username = username
450. \_userId = loginResponsePacket.id
451. \_admin = loginResponsePacket.admin
453. **else**:
454. self.errLabel.setText(loginResponsePacket.err)
455. **except** Exception:
456. reportError()

459. **class** RegisterWindow(QDialog):
460. """ GUI Class for register window """
461. **def** \_\_init\_\_(self, \*args):
462. **try**:
463. super().\_\_init\_\_(\*args)
464. loadUi("register.ui", self)
465. self.registerButton.clicked.connect(self.validateInputs)
466. **except** Exception:
467. reportError()
469. **def** validateInputs(self):
470. """ Called on register button click. Basic input validation. """
471. **try**:
472. username = self.usernameInput.text()
473. password1 = self.passwordInput1.text()
474. password2 = self.passwordInput2.text()
476. **if** **not** username.isspace() **and** username != "" **and** len(username) < 33:
477. **if** password1 == password2:
478. **if** **not** password1.isspace() **and** password1 != "":
479. self.register(username, password1)
480. **else**:
481. self.errLabel.setText("Passwords must not consist of whitespace only")
482. **else**:
483. self.errLabel.setText("Passwords must match")
484. **else**:
485. self.errLabel.setText("Usernames must not consist of whitespace only, and be\nless than 33 chars long")
486. **except** Exception:
487. reportError()
489. **def** register(self, username, password):
490. """
491. Attempts to register user with server.
493. Args:
494. username (string): Username to register.
495. password (string): Password to register, not yet hashed.
496. """
497. **try**:
498. password = hashString(password)
499. registerPacket = RegisterPacket(username, password)
500. **global** \_serverSocket
501. \_serverSocket.send(encode(registerPacket))
502. registerResponse = decode(\_serverSocket.recv(MAXTRANSMISSIONSIZE)) # Wait for user creation packet response
503. **if** registerResponse.valid:
504. successNotifier = QMessageBox()
505. successNotifier.setIcon(QMessageBox.Information)
506. successNotifier.setText(f"Successfully registered user '{username}'")
507. successNotifier.setWindowTitle("Account creation successful")
508. successNotifier.exec\_()
509. self.close()
510. **else**:
511. self.errLabel.setText(registerResponse.err)
512. **except** Exception:
513. reportError()



518. # Functions
519. """
520. Constructs a message object and sends it to the server.
522. Args:
523. message (string): The message contents.
524. """
525. **def** SendMessage(message):
526. **try**:
527. **global** \_serverSocket, \_username
528. debugPrint(\_userId, DEBUG)
529. newMessage = Message(\_userId, \_username, message)
530. newMessagePacket = MessagePacket(newMessage)
531. \_serverSocket.send(encode(newMessagePacket))
533. **except** Exception:
534. reportError()
536. """
537. Formats a command string into arguments and sends it to the server.
539. Args:
540. command (string): The raw command text before formatting.
541. """
542. **def** ParseCommand(command):
543. **try**:
544. **global** \_serverSocket
545. command = command[1:] # Strip command char
546. args = command.split(" ")
547. command = args[0]
548. **del** args[0]
549. commandPacket = CommandPacket(command, args)
550. \_serverSocket.send(encode(commandPacket))
552. **except** Exception:
553. reportError()
555. """
556. Ensures a message is in the correct format and then passes it to the GUI to display.
558. Args:
559. message (string, optional): The string to convert to message object.
560. message (Message, optional): The message object to pass to the GUI.
561. """
562. **def** printMessage(message):
563. **if** type(message) **is** str:
564. message = Message(contents=message)
566. \_mainGui.writeSignal.emit(message)

569. **def** onProgramExit():
570. **try**:
571. **global** \_serverSocket
572. debugPrint("Window closed: Force closing all threads and server socket", DEBUG)
573. \_serverSocket.close()
574. os.\_exit(1)
575. **except** Exception:
576. reportError()

579. **def** ListenForPackets(server):
580. **try**:
581. **global** \_serverSocket, \_mainGui
583. readyToListen = Packet("READYTOLISTEN") # Tell the server we are ready to listen using generic packet
584. \_serverSocket.send(encode(readyToListen))
586. **while** True:
587. packet = decode(server.recv(MAXTRANSMISSIONSIZE))
589. **if** packet.type == "MESSAGELIST":
590. **for** message **in** packet.messageList:
591. printMessage(message)
593. **elif** packet.type == "MESSAGE":
594. formatMessage(packet)
596. **elif** packet.type == "ONLINEUSERS":
597. \_mainGui.usersChangedSignal.emit(packet.userList)
599. **elif** packet.type == "COMMANDRESPONSE":
600. **if** packet.success:
601. **if** packet.command == "help":
602. printMessage(packet.response[0])
603. **for** i **in** range(1, len(packet.response)):
604. printMessage(f" - \*{packet.response[i][0]}\* : {packet.response[i][1]}")
606. **if** packet.command == "markup":
607. printMessage(packet.response[0])
608. printMessage(packet.response[1])
609. **for** i **in** range(2, len(packet.response)):
610. printMessage(f" - {packet.response[i][0]}, {packet.response[i][1]} : {packet.response[i][2]}")
612. **elif** packet.command == "ping":
613. printMessage(Message(contents=packet.response, colour=INFO))
615. **elif** packet.command == "whisper":
616. printMessage(Message(contents=formatDateTime(packet.timeSent) + packet.response, colour=INFO))
618. **else**:
619. printMessage(Message(contents=f"Error executing command '{packet.command}' - {packet.err}", colour=COMMANDERROR))

622. **elif** packet.type == "USERLIST":
623. \_mainGui.adminSettings.UserListReceived(packet.userList)

626. **elif** packet.type == "USERINFO":
627. \_mainGui.adminSettings.UpdateUserInfo(packet.id, packet.messageCount, packet.admin, packet.flags)
629. **elif** packet.type == "EDITMESSAGE":
630. \_mainGui.updateMessageSignal.emit(packet.messageId, packet.newContents)
632. **elif** packet.type == "DELETEMESSAGE":
633. \_mainGui.deleteMessageSignal.emit(packet.messageId)
635. **else**:
636. **print**(f"Unknown packet received: {packet.type}")


640. **except** Exception:
641. reportError()

644. **def** formatMessage(packet):
645. **try**:
646. printMessage(packet.message)
647. **except** Exception:
648. reportError()

651. **def** formatTextForDisplay(message, colour):
652. **try**:
653. message = cgi.escape(message) # Escape html code; sanitise input
654. # Replace balsamiq chars in pairs with html
655. message = formatBalsmaiq(message, "\*", "b")
656. message = formatBalsmaiq(message, "\_", "i")
657. message = formatBalsmaiq(message, "~", "s")
658. message = formatBalsmaiq(message, "!", "u")
660. message = f"<font color='{colour}'> {message} </font>"
662. **return** message
663. **except** Exception:
664. reportError()
666. **def** inverseFormatTextForDisplay(message):
667. **try**:
668. colour = re.search("<font color='(.+?)'>", message).group(1) # Use regex to extract the hex colour code of the message
669. # Replace html pairs with balsamiq
670. message = re.sub("<b>|</b>", "\*", message)
671. message = re.sub("<i>|</i>", "\_", message)
672. message = re.sub("<s>|</s>", "~", message)
673. message = re.sub("<u>|</u>", "!", message)
674. message = re.sub('<[^<]+?>', '', message) # strip remaining html code
676. **return** (message, colour)
677. **except** Exception:
678. reportError()

681. **def** formatBalsmaiq(message, specialChar, tag):
682. **try**:
683. **global** NONPRINTINGCHAR
684. charInstances = []
685. i = 0
686. **while** True: # Locate all instances of special char within message
687. charInstance = message.find(specialChar, i)
688. **if** charInstance == -1:
689. **break**
690. **else**:
691. charInstances.append(charInstance)
692. i = charInstance + 1
693. message = list(message) # Convert to list so we can substitute chars by index
694. i = 0
695. **for** charInstance **in** charInstances: # Ensure character is not escaped by '\'
696. **if** message[charInstance - 1] == "\\":
697. message[charInstance - 1] = NONPRINTINGCHAR
698. **del** charInstances[i]
699. **else**:
700. i+= 1
702. **if** len(charInstances) % 2 != 0: # Ignore special chars without a pair
703. charInstances = charInstances[:-1]
705. **for** i **in** range(0, len(charInstances), 2): # Replace pairs of balsamiq with html code
706. message[charInstances[i]] = f"<{tag}>"
707. message[charInstances[i + 1]] = f"</{tag}>"
709. message = "".join(message) # Convert back to string
710. **return** message
711. **except** Exception:
712. reportError()

715. **def** \_\_main\_\_():
716. **try**:
717. **global** \_serverSocket, \_username, MAXTRANSMISSIONSIZE, COMMANDCHAR, DEBUG
719. \_configManager = ClientConfig("config.json")
720. MAXTRANSMISSIONSIZE = \_configManager.data["maxTransmissionSize"]
721. COMMANDCHAR = \_configManager.data["commandChar"]
722. DEBUG = \_configManager.data["debug"]
724. atexit.register(onProgramExit)
726. # Print PyQt 'silent' errors
727. sys.\_excepthook = sys.excepthook
728. **def** exception\_hook(exctype, value, traceback):
729. **print**(exctype, value, traceback)
730. sys.\_excepthook(exctype, value, traceback)
731. sys.exit(1)
732. sys.excepthook = exception\_hook
734. # Create a socket object
735. \_serverSocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)
737. # Get local machine name and assign a port
738. host = socket.gethostname()
739. port = \_configManager.data["port"]
741. # Connect to hostname on the port.
742. \_serverSocket.connect((host, port))
744. # Display UI
745. **global** GuiDone, \_app, \_mainGui
746. \_app = QApplication(sys.argv)
747. \_mainGui = MainWindow()
748. loginGui = LoginWindow()
749. loginGui.exec\_()
750. \_mainGui.postLogin()
751. **if** \_username == "": # Window was closed without an input
752. os.\_exit(1)
753. \_mainGui.setUsername(\_username)
754. \_mainGui.show()
756. # Start listener thread for server responses
757. listenerThread = Thread(target=ListenForPackets, args=(\_serverSocket,))
758. listenerThread.start()
760. sys.exit(\_app.exec\_())
762. **except** Exception:
763. reportError()

766. **if** \_\_name\_\_ == "\_\_main\_\_":
767. \_\_main\_\_()

### 3.2.2 adminSettings.ui

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<ui** version="4.0"**>**
3. **<class>**AdminSettings**</class>**
4. **<widget** class="QWidget" name="AdminSettings"**>**
5. **<property** name="geometry"**>**
6. **<rect>**
7. **<x>**0**</x>**
8. **<y>**0**</y>**
9. **<width>**400**</width>**
10. **<height>**397**</height>**
11. **</rect>**
12. **</property>**
13. **<property** name="windowTitle"**>**
14. **<string>**Admin Settings**</string>**
15. **</property>**
16. **<layout** class="QFormLayout" name="formLayout"**>**
17. **<item** row="0" column="0"**>**
18. **<widget** class="QLabel" name="label"**>**
19. **<property** name="text"**>**
20. **<string>**Selected User**</string>**
21. **</property>**
22. **</widget>**
23. **</item>**
24. **<item** row="1" column="0"**>**
25. **<widget** class="QComboBox" name="userListComboBox"**/>**
26. **</item>**
27. **<item** row="2" column="0" colspan="2"**>**
28. **<widget** class="Line" name="line"**>**
29. **<property** name="orientation"**>**
30. **<enum>**Qt::Horizontal**</enum>**
31. **</property>**
32. **</widget>**
33. **</item>**
34. **<item** row="3" column="0"**>**
35. **<widget** class="QLabel" name="label\_2"**>**
36. **<property** name="text"**>**
37. **<string>**Actions**</string>**
38. **</property>**
39. **</widget>**
40. **</item>**
41. **<item** row="4" column="0"**>**
42. **<widget** class="QPushButton" name="toggleAdminStatusBtn"**>**
43. **<property** name="enabled"**>**
44. **<bool>**false**</bool>**
45. **</property>**
46. **<property** name="text"**>**
47. **<string>**Promote to Admin**</string>**
48. **</property>**
49. **</widget>**
50. **</item>**
51. **<item** row="5" column="0" colspan="2"**>**
52. **<widget** class="Line" name="line\_2"**>**
53. **<property** name="orientation"**>**
54. **<enum>**Qt::Horizontal**</enum>**
55. **</property>**
56. **</widget>**
57. **</item>**
58. **<item** row="6" column="0"**>**
59. **<widget** class="QLabel" name="label\_3"**>**
60. **<property** name="text"**>**
61. **<string>**User Info**</string>**
62. **</property>**
63. **</widget>**
64. **</item>**
65. **<item** row="7" column="0"**>**
66. **<widget** class="QLabel" name="label\_5"**>**
67. **<property** name="text"**>**
68. **<string>**User Id:**</string>**
69. **</property>**
70. **</widget>**
71. **</item>**
72. **<item** row="7" column="1"**>**
73. **<widget** class="QLabel" name="userIdLabel"**>**
74. **<property** name="text"**>**
75. **<string>**0**</string>**
76. **</property>**
77. **</widget>**
78. **</item>**
79. **<item** row="8" column="0"**>**
80. **<widget** class="QLabel" name="label\_4"**>**
81. **<property** name="text"**>**
82. **<string>**Messages Sent:**</string>**
83. **</property>**
84. **</widget>**
85. **</item>**
86. **<item** row="8" column="1"**>**
87. **<widget** class="QLabel" name="messageCountLabel"**>**
88. **<property** name="text"**>**
89. **<string>**0**</string>**
90. **</property>**
91. **</widget>**
92. **</item>**
93. **<item** row="9" column="0"**>**
94. **<widget** class="QLabel" name="label\_6"**>**
95. **<property** name="text"**>**
96. **<string>**Times Reported:**</string>**
97. **</property>**
98. **</widget>**
99. **</item>**
100. **<item** row="9" column="1"**>**
101. **<widget** class="QLabel" name="reportCountLabel"**>**
102. **<property** name="text"**>**
103. **<string>**0**</string>**
104. **</property>**
105. **</widget>**
106. **</item>**
107. **<item** row="10" column="0" colspan="2"**>**
108. **<widget** class="Line" name="line\_3"**>**
109. **<property** name="orientation"**>**
110. **<enum>**Qt::Horizontal**</enum>**
111. **</property>**
112. **</widget>**
113. **</item>**
114. **<item** row="11" column="0"**>**
115. **<widget** class="QLabel" name="label\_7"**>**
116. **<property** name="text"**>**
117. **<string>**Reports**</string>**
118. **</property>**
119. **</widget>**
120. **</item>**
121. **<item** row="12" column="0" colspan="2"**>**
122. **<widget** class="QTableWidget" name="reportTable"**>**
123. **<property** name="editTriggers"**>**
124. **<set>**QAbstractItemView::NoEditTriggers**</set>**
125. **</property>**
126. **<property** name="dragEnabled"**>**
127. **<bool>**true**</bool>**
128. **</property>**
129. **<property** name="alternatingRowColors"**>**
130. **<bool>**true**</bool>**
131. **</property>**
132. **<property** name="rowCount"**>**
133. **<number>**1**</number>**
134. **</property>**
135. **<property** name="columnCount"**>**
136. **<number>**3**</number>**
137. **</property>**
138. **<attribute** name="horizontalHeaderVisible"**>**
139. **<bool>**true**</bool>**
140. **</attribute>**
141. **<row/>**
142. **<column/>**
143. **<column/>**
144. **<column/>**
145. **<item** row="0" column="0"**>**
146. **<property** name="text"**>**
147. **<string** notr="true"**>**Message**</string>**
148. **</property>**
149. **<property** name="font"**>**
150. **<font>**
151. **<weight>**75**</weight>**
152. **<bold>**true**</bold>**
153. **</font>**
154. **</property>**
155. **</item>**
156. **<item** row="0" column="1"**>**
157. **<property** name="text"**>**
158. **<string>**Report**</string>**
159. **</property>**
160. **<property** name="font"**>**
161. **<font>**
162. **<weight>**75**</weight>**
163. **<bold>**true**</bold>**
164. **</font>**
165. **</property>**
166. **</item>**
167. **<item** row="0" column="2"**>**
168. **<property** name="text"**>**
169. **<string>**Reported By**</string>**
170. **</property>**
171. **<property** name="font"**>**
172. **<font>**
173. **<weight>**75**</weight>**
174. **<bold>**true**</bold>**
175. **</font>**
176. **</property>**
177. **</item>**
178. **</widget>**
179. **</item>**
180. **</layout>**
181. **</widget>**
182. **<resources/>**
183. **<connections/>**
184. **</ui>**

### 3.2.3 login.ui

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<ui** version="4.0"**>**
3. **<class>**LoginDialog**</class>**
4. **<widget** class="QDialog" name="LoginDialog"**>**
5. **<property** name="windowModality"**>**
6. **<enum>**Qt::ApplicationModal**</enum>**
7. **</property>**
8. **<property** name="enabled"**>**
9. **<bool>**true**</bool>**
10. **</property>**
11. **<property** name="geometry"**>**
12. **<rect>**
13. **<x>**0**</x>**
14. **<y>**0**</y>**
15. **<width>**248**</width>**
16. **<height>**145**</height>**
17. **</rect>**
18. **</property>**
19. **<property** name="sizePolicy"**>**
20. **<sizepolicy** hsizetype="Fixed" vsizetype="Fixed"**>**
21. **<horstretch>**0**</horstretch>**
22. **<verstretch>**0**</verstretch>**
23. **</sizepolicy>**
24. **</property>**
25. **<property** name="minimumSize"**>**
26. **<size>**
27. **<width>**248**</width>**
28. **<height>**145**</height>**
29. **</size>**
30. **</property>**
31. **<property** name="maximumSize"**>**
32. **<size>**
33. **<width>**248**</width>**
34. **<height>**147**</height>**
35. **</size>**
36. **</property>**
37. **<property** name="windowTitle"**>**
38. **<string>**Photon - Login**</string>**
39. **</property>**
40. **<property** name="windowIcon"**>**
41. **<iconset>**
42. **<normaloff>**icon.ico**</normaloff>**icon.ico**</iconset>**
43. **</property>**
44. **<property** name="sizeGripEnabled"**>**
45. **<bool>**false**</bool>**
46. **</property>**
47. **<layout** class="QFormLayout" name="formLayout"**>**
48. **<item** row="0" column="1"**>**
49. **<widget** class="QLineEdit" name="usernameInput"**>**
50. **<property** name="placeholderText"**>**
51. **<string>**Username**</string>**
52. **</property>**
53. **</widget>**
54. **</item>**
55. **<item** row="2" column="1"**>**
56. **<widget** class="QLineEdit" name="passwordInput"**>**
57. **<property** name="enabled"**>**
58. **<bool>**true**</bool>**
59. **</property>**
60. **<property** name="echoMode"**>**
61. **<enum>**QLineEdit::Password**</enum>**
62. **</property>**
63. **<property** name="placeholderText"**>**
64. **<string>**Password**</string>**
65. **</property>**
66. **</widget>**
67. **</item>**
68. **<item** row="3" column="1"**>**
69. **<widget** class="QPushButton" name="loginButton"**>**
70. **<property** name="text"**>**
71. **<string>**Login**</string>**
72. **</property>**
73. **</widget>**
74. **</item>**
75. **<item** row="4" column="0" colspan="2"**>**
76. **<widget** class="QLabel" name="errLabel"**>**
77. **<property** name="palette"**>**
78. **<palette>**
79. **<active>**
80. **<colorrole** role="WindowText"**>**
81. **<brush** brushstyle="SolidPattern"**>**
82. **<color** alpha="255"**>**
83. **<red>**255**</red>**
84. **<green>**0**</green>**
85. **<blue>**0**</blue>**
86. **</color>**
87. **</brush>**
88. **</colorrole>**
89. **</active>**
90. **<inactive>**
91. **<colorrole** role="WindowText"**>**
92. **<brush** brushstyle="SolidPattern"**>**
93. **<color** alpha="255"**>**
94. **<red>**255**</red>**
95. **<green>**0**</green>**
96. **<blue>**0**</blue>**
97. **</color>**
98. **</brush>**
99. **</colorrole>**
100. **</inactive>**
101. **<disabled>**
102. **<colorrole** role="WindowText"**>**
103. **<brush** brushstyle="SolidPattern"**>**
104. **<color** alpha="255"**>**
105. **<red>**120**</red>**
106. **<green>**120**</green>**
107. **<blue>**120**</blue>**
108. **</color>**
109. **</brush>**
110. **</colorrole>**
111. **</disabled>**
112. **</palette>**
113. **</property>**
114. **<property** name="text"**>**
115. **<string/>**
116. **</property>**
117. **</widget>**
118. **</item>**
119. **<item** row="5" column="0" colspan="2"**>**
120. **<widget** class="QPushButton" name="newAccButton"**>**
121. **<property** name="text"**>**
122. **<string>**Create New Account**</string>**
123. **</property>**
124. **</widget>**
125. **</item>**
126. **</layout>**
127. **</widget>**
128. **<resources/>**
129. **<connections/>**
130. **</ui>**

### 3.2.4 mainWindow.ui

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<ui** version="4.0"**>**
3. **<class>**MainWindow**</class>**
4. **<widget** class="QMainWindow" name="MainWindow"**>**
5. **<property** name="geometry"**>**
6. **<rect>**
7. **<x>**0**</x>**
8. **<y>**0**</y>**
9. **<width>**673**</width>**
10. **<height>**426**</height>**
11. **</rect>**
12. **</property>**
13. **<property** name="palette"**>**
14. **<palette>**
15. **<active>**
16. **<colorrole** role="Shadow"**>**
17. **<brush** brushstyle="SolidPattern"**>**
18. **<color** alpha="255"**>**
19. **<red>**170**</red>**
20. **<green>**255**</green>**
21. **<blue>**127**</blue>**
22. **</color>**
23. **</brush>**
24. **</colorrole>**
25. **</active>**
26. **<inactive>**
27. **<colorrole** role="Shadow"**>**
28. **<brush** brushstyle="SolidPattern"**>**
29. **<color** alpha="255"**>**
30. **<red>**170**</red>**
31. **<green>**255**</green>**
32. **<blue>**127**</blue>**
33. **</color>**
34. **</brush>**
35. **</colorrole>**
36. **</inactive>**
37. **<disabled>**
38. **<colorrole** role="Shadow"**>**
39. **<brush** brushstyle="SolidPattern"**>**
40. **<color** alpha="255"**>**
41. **<red>**170**</red>**
42. **<green>**255**</green>**
43. **<blue>**127**</blue>**
44. **</color>**
45. **</brush>**
46. **</colorrole>**
47. **</disabled>**
48. **</palette>**
49. **</property>**
50. **<property** name="windowTitle"**>**
51. **<string>**Photon - IM Client**</string>**
52. **</property>**
53. **<property** name="windowIcon"**>**
54. **<iconset>**
55. **<normaloff>**icon.ico**</normaloff>**icon.ico**</iconset>**
56. **</property>**
57. **<property** name="layoutDirection"**>**
58. **<enum>**Qt::LeftToRight**</enum>**
59. **</property>**
60. **<property** name="iconSize"**>**
61. **<size>**
62. **<width>**32**</width>**
63. **<height>**32**</height>**
64. **</size>**
65. **</property>**
66. **<widget** class="QWidget" name="centralwidget"**>**
67. **<layout** class="QGridLayout" name="gridLayout"**>**
68. **<property** name="leftMargin"**>**
69. **<number>**10**</number>**
70. **</property>**
71. **<property** name="topMargin"**>**
72. **<number>**10**</number>**
73. **</property>**
74. **<property** name="rightMargin"**>**
75. **<number>**10**</number>**
76. **</property>**
77. **<property** name="bottomMargin"**>**
78. **<number>**9**</number>**
79. **</property>**
80. **<item** row="1" column="2"**>**
81. **<widget** class="QLabel" name="userCountLabel"**>**
82. **<property** name="sizePolicy"**>**
83. **<sizepolicy** hsizetype="Preferred" vsizetype="Preferred"**>**
84. **<horstretch>**0**</horstretch>**
85. **<verstretch>**0**</verstretch>**
86. **</sizepolicy>**
87. **</property>**
88. **<property** name="layoutDirection"**>**
89. **<enum>**Qt::LeftToRight**</enum>**
90. **</property>**
91. **<property** name="text"**>**
92. **<string>**Users Online: 0**</string>**
93. **</property>**
94. **<property** name="alignment"**>**
95. **<set>**Qt::AlignLeading|Qt::AlignLeft|Qt::AlignVCenter**</set>**
96. **</property>**
97. **</widget>**
98. **</item>**
99. **<item** row="1" column="0"**>**
100. **<widget** class="QLabel" name="label"**>**
101. **<property** name="text"**>**
102. **<string>**Using Photon, a Python IM client by Sam Poirier**</string>**
103. **</property>**
104. **</widget>**
105. **</item>**
106. **<item** row="5" column="1"**>**
107. **<widget** class="QPushButton" name="messageInputButton"**>**
108. **<property** name="cursor"**>**
109. **<cursorShape>**PointingHandCursor**</cursorShape>**
110. **</property>**
111. **<property** name="text"**>**
112. **<string>**Send**</string>**
113. **</property>**
114. **<property** name="iconSize"**>**
115. **<size>**
116. **<width>**20**</width>**
117. **<height>**20**</height>**
118. **</size>**
119. **</property>**
120. **</widget>**
121. **</item>**
122. **<item** row="5" column="0"**>**
123. **<widget** class="QLineEdit" name="messageInput"**>**
124. **<property** name="placeholderText"**>**
125. **<string>**Message**</string>**
126. **</property>**
127. **</widget>**
128. **</item>**
129. **<item** row="1" column="1"**>**
130. **<widget** class="QLabel" name="usernameLabel"**>**
131. **<property** name="sizePolicy"**>**
132. **<sizepolicy** hsizetype="Preferred" vsizetype="Preferred"**>**
133. **<horstretch>**0**</horstretch>**
134. **<verstretch>**0**</verstretch>**
135. **</sizepolicy>**
136. **</property>**
137. **<property** name="layoutDirection"**>**
138. **<enum>**Qt::LeftToRight**</enum>**
139. **</property>**
140. **<property** name="text"**>**
141. **<string>**Not Logged in**</string>**
142. **</property>**
143. **<property** name="alignment"**>**
144. **<set>**Qt::AlignRight|Qt::AlignTrailing|Qt::AlignVCenter**</set>**
145. **</property>**
146. **</widget>**
147. **</item>**
148. **<item** row="4" column="2"**>**
149. **<widget** class="QTextEdit" name="userListBox"**>**
150. **<property** name="enabled"**>**
151. **<bool>**true**</bool>**
152. **</property>**
153. **<property** name="maximumSize"**>**
154. **<size>**
155. **<width>**80**</width>**
156. **<height>**16777215**</height>**
157. **</size>**
158. **</property>**
159. **<property** name="palette"**>**
160. **<palette>**
161. **<active>**
162. **<colorrole** role="Base"**>**
163. **<brush** brushstyle="SolidPattern"**>**
164. **<color** alpha="255"**>**
165. **<red>**240**</red>**
166. **<green>**240**</green>**
167. **<blue>**240**</blue>**
168. **</color>**
169. **</brush>**
170. **</colorrole>**
171. **</active>**
172. **<inactive>**
173. **<colorrole** role="Base"**>**
174. **<brush** brushstyle="SolidPattern"**>**
175. **<color** alpha="255"**>**
176. **<red>**240**</red>**
177. **<green>**240**</green>**
178. **<blue>**240**</blue>**
179. **</color>**
180. **</brush>**
181. **</colorrole>**
182. **</inactive>**
183. **<disabled>**
184. **<colorrole** role="Base"**>**
185. **<brush** brushstyle="SolidPattern"**>**
186. **<color** alpha="255"**>**
187. **<red>**240**</red>**
188. **<green>**240**</green>**
189. **<blue>**240**</blue>**
190. **</color>**
191. **</brush>**
192. **</colorrole>**
193. **</disabled>**
194. **</palette>**
195. **</property>**
196. **<property** name="readOnly"**>**
197. **<bool>**true**</bool>**
198. **</property>**
199. **</widget>**
200. **</item>**
201. **<item** row="4" column="0" colspan="2"**>**
202. **<widget** class="QScrollArea" name="messageScrollArea"**>**
203. **<property** name="palette"**>**
204. **<palette>**
205. **<active>**
206. **<colorrole** role="Base"**>**
207. **<brush** brushstyle="SolidPattern"**>**
208. **<color** alpha="255"**>**
209. **<red>**255**</red>**
210. **<green>**255**</green>**
211. **<blue>**255**</blue>**
212. **</color>**
213. **</brush>**
214. **</colorrole>**
215. **<colorrole** role="Window"**>**
216. **<brush** brushstyle="SolidPattern"**>**
217. **<color** alpha="255"**>**
218. **<red>**255**</red>**
219. **<green>**255**</green>**
220. **<blue>**255**</blue>**
221. **</color>**
222. **</brush>**
223. **</colorrole>**
224. **</active>**
225. **<inactive>**
226. **<colorrole** role="Base"**>**
227. **<brush** brushstyle="SolidPattern"**>**
228. **<color** alpha="255"**>**
229. **<red>**255**</red>**
230. **<green>**255**</green>**
231. **<blue>**255**</blue>**
232. **</color>**
233. **</brush>**
234. **</colorrole>**
235. **<colorrole** role="Window"**>**
236. **<brush** brushstyle="SolidPattern"**>**
237. **<color** alpha="255"**>**
238. **<red>**255**</red>**
239. **<green>**255**</green>**
240. **<blue>**255**</blue>**
241. **</color>**
242. **</brush>**
243. **</colorrole>**
244. **</inactive>**
245. **<disabled>**
246. **<colorrole** role="Base"**>**
247. **<brush** brushstyle="SolidPattern"**>**
248. **<color** alpha="255"**>**
249. **<red>**255**</red>**
250. **<green>**255**</green>**
251. **<blue>**255**</blue>**
252. **</color>**
253. **</brush>**
254. **</colorrole>**
255. **<colorrole** role="Window"**>**
256. **<brush** brushstyle="SolidPattern"**>**
257. **<color** alpha="255"**>**
258. **<red>**255**</red>**
259. **<green>**255**</green>**
260. **<blue>**255**</blue>**
261. **</color>**
262. **</brush>**
263. **</colorrole>**
264. **</disabled>**
265. **</palette>**
266. **</property>**
267. **<property** name="autoFillBackground"**>**
268. **<bool>**true**</bool>**
269. **</property>**
270. **<property** name="horizontalScrollBarPolicy"**>**
271. **<enum>**Qt::ScrollBarAlwaysOff**</enum>**
272. **</property>**
273. **<property** name="widgetResizable"**>**
274. **<bool>**true**</bool>**
275. **</property>**
276. **<widget** class="QWidget" name="scrollAreaWidgetContents"**>**
277. **<property** name="geometry"**>**
278. **<rect>**
279. **<x>**0**</x>**
280. **<y>**0**</y>**
281. **<width>**565**</width>**
282. **<height>**308**</height>**
283. **</rect>**
284. **</property>**
285. **<layout** class="QGridLayout" name="gridLayout\_2"**>**
286. **<property** name="leftMargin"**>**
287. **<number>**0**</number>**
288. **</property>**
289. **<property** name="topMargin"**>**
290. **<number>**0**</number>**
291. **</property>**
292. **<property** name="rightMargin"**>**
293. **<number>**0**</number>**
294. **</property>**
295. **<property** name="bottomMargin"**>**
296. **<number>**0**</number>**
297. **</property>**
298. **<item** row="0" column="0"**>**
299. **<widget** class="QWidget" name="messageWidget" native="true"**>**
300. **<property** name="sizePolicy"**>**
301. **<sizepolicy** hsizetype="Preferred" vsizetype="Expanding"**>**
302. **<horstretch>**0**</horstretch>**
303. **<verstretch>**0**</verstretch>**
304. **</sizepolicy>**
305. **</property>**
306. **<property** name="autoFillBackground"**>**
307. **<bool>**false**</bool>**
308. **</property>**
309. **<widget** class="QWidget" name="formLayoutWidget"**>**
310. **<property** name="geometry"**>**
311. **<rect>**
312. **<x>**1**</x>**
313. **<y>**4**</y>**
314. **<width>**562**</width>**
315. **<height>**250**</height>**
316. **</rect>**
317. **</property>**
318. **<layout** class="QFormLayout" name="messageLayout"**>**
319. **<property** name="sizeConstraint"**>**
320. **<enum>**QLayout::SetNoConstraint**</enum>**
321. **</property>**
322. **<property** name="fieldGrowthPolicy"**>**
323. **<enum>**QFormLayout::FieldsStayAtSizeHint**</enum>**
324. **</property>**
325. **<property** name="rowWrapPolicy"**>**
326. **<enum>**QFormLayout::WrapLongRows**</enum>**
327. **</property>**
328. **<property** name="horizontalSpacing"**>**
329. **<number>**0**</number>**
330. **</property>**
331. **<property** name="verticalSpacing"**>**
332. **<number>**0**</number>**
333. **</property>**
334. **</layout>**
335. **</widget>**
336. **</widget>**
337. **</item>**
338. **</layout>**
339. **</widget>**
340. **</widget>**
341. **</item>**
342. **<item** row="0" column="2"**>**
343. **<widget** class="QPushButton" name="adminSettingsButton"**>**
344. **<property** name="text"**>**
345. **<string>**Admin Settings**</string>**
346. **</property>**
347. **</widget>**
348. **</item>**
349. **</layout>**
350. **</widget>**
351. **<widget** class="QStatusBar" name="statusbar"**/>**
352. **</widget>**
353. **<resources/>**
354. **<connections/>**
355. **</ui>**

### 3.2.5 message.ui

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<ui** version="4.0"**>**
3. **<class>**Form**</class>**
4. **<widget** class="QWidget" name="Form"**>**
5. **<property** name="geometry"**>**
6. **<rect>**
7. **<x>**0**</x>**
8. **<y>**0**</y>**
9. **<width>**420**</width>**
10. **<height>**23**</height>**
11. **</rect>**
12. **</property>**
13. **<property** name="sizePolicy"**>**
14. **<sizepolicy** hsizetype="Expanding" vsizetype="Expanding"**>**
15. **<horstretch>**0**</horstretch>**
16. **<verstretch>**0**</verstretch>**
17. **</sizepolicy>**
18. **</property>**
19. **<property** name="minimumSize"**>**
20. **<size>**
21. **<width>**0**</width>**
22. **<height>**0**</height>**
23. **</size>**
24. **</property>**
25. **<property** name="maximumSize"**>**
26. **<size>**
27. **<width>**16777215**</width>**
28. **<height>**16777215**</height>**
29. **</size>**
30. **</property>**
31. **<property** name="windowTitle"**>**
32. **<string>**Form**</string>**
33. **</property>**
34. **<property** name="autoFillBackground"**>**
35. **<bool>**true**</bool>**
36. **</property>**
37. **<layout** class="QGridLayout" name="gridLayout"**>**
38. **<property** name="leftMargin"**>**
39. **<number>**4**</number>**
40. **</property>**
41. **<property** name="topMargin"**>**
42. **<number>**2**</number>**
43. **</property>**
44. **<property** name="rightMargin"**>**
45. **<number>**4**</number>**
46. **</property>**
47. **<property** name="bottomMargin"**>**
48. **<number>**0**</number>**
49. **</property>**
50. **<property** name="horizontalSpacing"**>**
51. **<number>**2**</number>**
52. **</property>**
53. **<item** row="0" column="6"**>**
54. **<widget** class="QPushButton" name="messageOptionBtn"**>**
55. **<property** name="styleSheet"**>**
56. **<string** notr="true"**>**background-color:transparent;
57. border:none**</string>**
58. **</property>**
59. **<property** name="text"**>**
60. **<string/>**
61. **</property>**
62. **<property** name="icon"**>**
63. **<iconset>**
64. **<normaloff>**verticalEllipsis.png**</normaloff>**verticalEllipsis.png**</iconset>**
65. **</property>**
66. **</widget>**
67. **</item>**
68. **<item** row="0" column="1"**>**
69. **<widget** class="QLabel" name="timeLabel"**>**
70. **<property** name="palette"**>**
71. **<palette>**
72. **<active>**
73. **<colorrole** role="WindowText"**>**
74. **<brush** brushstyle="SolidPattern"**>**
75. **<color** alpha="255"**>**
76. **<red>**99**</red>**
77. **<green>**99**</green>**
78. **<blue>**99**</blue>**
79. **</color>**
80. **</brush>**
81. **</colorrole>**
82. **</active>**
83. **<inactive>**
84. **<colorrole** role="WindowText"**>**
85. **<brush** brushstyle="SolidPattern"**>**
86. **<color** alpha="255"**>**
87. **<red>**99**</red>**
88. **<green>**99**</green>**
89. **<blue>**99**</blue>**
90. **</color>**
91. **</brush>**
92. **</colorrole>**
93. **</inactive>**
94. **<disabled>**
95. **<colorrole** role="WindowText"**>**
96. **<brush** brushstyle="SolidPattern"**>**
97. **<color** alpha="255"**>**
98. **<red>**120**</red>**
99. **<green>**120**</green>**
100. **<blue>**120**</blue>**
101. **</color>**
102. **</brush>**
103. **</colorrole>**
104. **</disabled>**
105. **</palette>**
106. **</property>**
107. **<property** name="layoutDirection"**>**
108. **<enum>**Qt::LeftToRight**</enum>**
109. **</property>**
110. **<property** name="text"**>**
111. **<string>**timestamp**</string>**
112. **</property>**
113. **<property** name="alignment"**>**
114. **<set>**Qt::AlignLeading|Qt::AlignLeft|Qt::AlignVCenter**</set>**
115. **</property>**
116. **</widget>**
117. **</item>**
118. **<item** row="0" column="2"**>**
119. **<widget** class="QLabel" name="usernameLabel"**>**
120. **<property** name="palette"**>**
121. **<palette>**
122. **<active>**
123. **<colorrole** role="Shadow"**>**
124. **<brush** brushstyle="SolidPattern"**>**
125. **<color** alpha="255"**>**
126. **<red>**240**</red>**
127. **<green>**240**</green>**
128. **<blue>**240**</blue>**
129. **</color>**
130. **</brush>**
131. **</colorrole>**
132. **</active>**
133. **<inactive>**
134. **<colorrole** role="Shadow"**>**
135. **<brush** brushstyle="SolidPattern"**>**
136. **<color** alpha="255"**>**
137. **<red>**240**</red>**
138. **<green>**240**</green>**
139. **<blue>**240**</blue>**
140. **</color>**
141. **</brush>**
142. **</colorrole>**
143. **</inactive>**
144. **<disabled>**
145. **<colorrole** role="Shadow"**>**
146. **<brush** brushstyle="SolidPattern"**>**
147. **<color** alpha="255"**>**
148. **<red>**240**</red>**
149. **<green>**240**</green>**
150. **<blue>**240**</blue>**
151. **</color>**
152. **</brush>**
153. **</colorrole>**
154. **</disabled>**
155. **</palette>**
156. **</property>**
157. **<property** name="font"**>**
158. **<font>**
159. **<pointsize>**8**</pointsize>**
160. **<weight>**75**</weight>**
161. **<bold>**true**</bold>**
162. **</font>**
163. **</property>**
164. **<property** name="text"**>**
165. **<string>**username**</string>**
166. **</property>**
167. **</widget>**
168. **</item>**
169. **<item** row="0" column="3"**>**
170. **<widget** class="QLabel" name="messageLabel"**>**
171. **<property** name="sizePolicy"**>**
172. **<sizepolicy** hsizetype="Expanding" vsizetype="Fixed"**>**
173. **<horstretch>**0**</horstretch>**
174. **<verstretch>**0**</verstretch>**
175. **</sizepolicy>**
176. **</property>**
177. **<property** name="maximumSize"**>**
178. **<size>**
179. **<width>**16777215**</width>**
180. **<height>**14**</height>**
181. **</size>**
182. **</property>**
183. **<property** name="text"**>**
184. **<string>**message**</string>**
185. **</property>**
186. **<property** name="textInteractionFlags"**>**
187. **<set>**Qt::LinksAccessibleByMouse|Qt::TextSelectableByKeyboard|Qt::TextSelectableByMouse**</set>**
188. **</property>**
189. **</widget>**
190. **</item>**
191. **</layout>**
192. **</widget>**
193. **<resources/>**
194. **<connections/>**
195. **</ui>**

### 3.2.6 messageOptions.ui

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<ui** version="4.0"**>**
3. **<class>**Dialog**</class>**
4. **<widget** class="QDialog" name="Dialog"**>**
5. **<property** name="geometry"**>**
6. **<rect>**
7. **<x>**0**</x>**
8. **<y>**0**</y>**
9. **<width>**600**</width>**
10. **<height>**72**</height>**
11. **</rect>**
12. **</property>**
13. **<property** name="minimumSize"**>**
14. **<size>**
15. **<width>**329**</width>**
16. **<height>**72**</height>**
17. **</size>**
18. **</property>**
19. **<property** name="maximumSize"**>**
20. **<size>**
21. **<width>**16777215**</width>**
22. **<height>**72**</height>**
23. **</size>**
24. **</property>**
25. **<property** name="windowTitle"**>**
26. **<string>**Message Options**</string>**
27. **</property>**
28. **<property** name="modal"**>**
29. **<bool>**true**</bool>**
30. **</property>**
31. **<layout** class="QGridLayout" name="gridLayout\_2"**>**
32. **<item** row="0" column="0"**>**
33. **<layout** class="QGridLayout" name="gridLayout"**>**
34. **<item** row="1" column="0"**>**
35. **<widget** class="QLabel" name="timeLabel"**>**
36. **<property** name="palette"**>**
37. **<palette>**
38. **<active>**
39. **<colorrole** role="WindowText"**>**
40. **<brush** brushstyle="SolidPattern"**>**
41. **<color** alpha="255"**>**
42. **<red>**99**</red>**
43. **<green>**99**</green>**
44. **<blue>**99**</blue>**
45. **</color>**
46. **</brush>**
47. **</colorrole>**
48. **</active>**
49. **<inactive>**
50. **<colorrole** role="WindowText"**>**
51. **<brush** brushstyle="SolidPattern"**>**
52. **<color** alpha="255"**>**
53. **<red>**99**</red>**
54. **<green>**99**</green>**
55. **<blue>**99**</blue>**
56. **</color>**
57. **</brush>**
58. **</colorrole>**
59. **</inactive>**
60. **<disabled>**
61. **<colorrole** role="WindowText"**>**
62. **<brush** brushstyle="SolidPattern"**>**
63. **<color** alpha="255"**>**
64. **<red>**120**</red>**
65. **<green>**120**</green>**
66. **<blue>**120**</blue>**
67. **</color>**
68. **</brush>**
69. **</colorrole>**
70. **</disabled>**
71. **</palette>**
72. **</property>**
73. **<property** name="text"**>**
74. **<string>**0:00:00**</string>**
75. **</property>**
76. **</widget>**
77. **</item>**
78. **<item** row="4" column="0" colspan="4"**>**
79. **<widget** class="QLineEdit" name="reportReason"**>**
80. **<property** name="inputMask"**>**
81. **<string/>**
82. **</property>**
83. **<property** name="text"**>**
84. **<string/>**
85. **</property>**
86. **<property** name="placeholderText"**>**
87. **<string>**Report Reason**</string>**
88. **</property>**
89. **</widget>**
90. **</item>**
91. **<item** row="1" column="4"**>**
92. **<widget** class="QPushButton" name="deleteButton"**>**
93. **<property** name="text"**>**
94. **<string>**Delete**</string>**
95. **</property>**
96. **</widget>**
97. **</item>**
98. **<item** row="1" column="3"**>**
99. **<widget** class="QPushButton" name="editButton"**>**
100. **<property** name="text"**>**
101. **<string>**Edit**</string>**
102. **</property>**
103. **</widget>**
104. **</item>**
105. **<item** row="1" column="2"**>**
106. **<widget** class="QLineEdit" name="editMessage"**>**
107. **<property** name="placeholderText"**>**
108. **<string>**Edited Text**</string>**
109. **</property>**
110. **</widget>**
111. **</item>**
112. **<item** row="4" column="4"**>**
113. **<widget** class="QPushButton" name="reportButton"**>**
114. **<property** name="text"**>**
115. **<string>**Report**</string>**
116. **</property>**
117. **</widget>**
118. **</item>**
119. **<item** row="1" column="1"**>**
120. **<widget** class="QLabel" name="usernameLabel"**>**
121. **<property** name="text"**>**
122. **<string><html><head/><body><p><span** style=" font-weight:600;"**>**&lt;user&gt;**</span></p></body></html></string>**
123. **</property>**
124. **</widget>**
125. **</item>**
126. **</layout>**
127. **</item>**
128. **</layout>**
129. **</widget>**
130. **<resources/>**
131. **<connections/>**
132. **</ui>**

### 3.2.7 register.ui

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<ui** version="4.0"**>**
3. **<class>**RegisterDialog**</class>**
4. **<widget** class="QDialog" name="RegisterDialog"**>**
5. **<property** name="windowModality"**>**
6. **<enum>**Qt::ApplicationModal**</enum>**
7. **</property>**
8. **<property** name="enabled"**>**
9. **<bool>**true**</bool>**
10. **</property>**
11. **<property** name="geometry"**>**
12. **<rect>**
13. **<x>**0**</x>**
14. **<y>**0**</y>**
15. **<width>**236**</width>**
16. **<height>**156**</height>**
17. **</rect>**
18. **</property>**
19. **<property** name="sizePolicy"**>**
20. **<sizepolicy** hsizetype="Fixed" vsizetype="Fixed"**>**
21. **<horstretch>**0**</horstretch>**
22. **<verstretch>**0**</verstretch>**
23. **</sizepolicy>**
24. **</property>**
25. **<property** name="minimumSize"**>**
26. **<size>**
27. **<width>**236**</width>**
28. **<height>**156**</height>**
29. **</size>**
30. **</property>**
31. **<property** name="maximumSize"**>**
32. **<size>**
33. **<width>**236**</width>**
34. **<height>**156**</height>**
35. **</size>**
36. **</property>**
37. **<property** name="windowTitle"**>**
38. **<string>**Photon - Register**</string>**
39. **</property>**
40. **<property** name="windowIcon"**>**
41. **<iconset>**
42. **<normaloff>**icon.ico**</normaloff>**icon.ico**</iconset>**
43. **</property>**
44. **<property** name="sizeGripEnabled"**>**
45. **<bool>**false**</bool>**
46. **</property>**
47. **<layout** class="QFormLayout" name="formLayout"**>**
48. **<item** row="2" column="1"**>**
49. **<widget** class="QLineEdit" name="passwordInput1"**>**
50. **<property** name="enabled"**>**
51. **<bool>**true**</bool>**
52. **</property>**
53. **<property** name="echoMode"**>**
54. **<enum>**QLineEdit::Password**</enum>**
55. **</property>**
56. **<property** name="placeholderText"**>**
57. **<string>**Password**</string>**
58. **</property>**
59. **</widget>**
60. **</item>**
61. **<item** row="3" column="1"**>**
62. **<widget** class="QLineEdit" name="passwordInput2"**>**
63. **<property** name="echoMode"**>**
64. **<enum>**QLineEdit::Password**</enum>**
65. **</property>**
66. **<property** name="placeholderText"**>**
67. **<string>**Re-enter Password**</string>**
68. **</property>**
69. **</widget>**
70. **</item>**
71. **<item** row="6" column="1"**>**
72. **<widget** class="QPushButton" name="registerButton"**>**
73. **<property** name="text"**>**
74. **<string>**Register**</string>**
75. **</property>**
76. **</widget>**
77. **</item>**
78. **<item** row="7" column="0" colspan="2"**>**
79. **<widget** class="QLabel" name="errLabel"**>**
80. **<property** name="maximumSize"**>**
81. **<size>**
82. **<width>**285**</width>**
83. **<height>**26**</height>**
84. **</size>**
85. **</property>**
86. **<property** name="palette"**>**
87. **<palette>**
88. **<active>**
89. **<colorrole** role="WindowText"**>**
90. **<brush** brushstyle="SolidPattern"**>**
91. **<color** alpha="255"**>**
92. **<red>**255**</red>**
93. **<green>**0**</green>**
94. **<blue>**0**</blue>**
95. **</color>**
96. **</brush>**
97. **</colorrole>**
98. **</active>**
99. **<inactive>**
100. **<colorrole** role="WindowText"**>**
101. **<brush** brushstyle="SolidPattern"**>**
102. **<color** alpha="255"**>**
103. **<red>**255**</red>**
104. **<green>**0**</green>**
105. **<blue>**0**</blue>**
106. **</color>**
107. **</brush>**
108. **</colorrole>**
109. **</inactive>**
110. **<disabled>**
111. **<colorrole** role="WindowText"**>**
112. **<brush** brushstyle="SolidPattern"**>**
113. **<color** alpha="255"**>**
114. **<red>**120**</red>**
115. **<green>**120**</green>**
116. **<blue>**120**</blue>**
117. **</color>**
118. **</brush>**
119. **</colorrole>**
120. **</disabled>**
121. **</palette>**
122. **</property>**
123. **<property** name="text"**>**
124. **<string/>**
125. **</property>**
126. **</widget>**
127. **</item>**
128. **<item** row="0" column="1"**>**
129. **<widget** class="QLineEdit" name="usernameInput"**>**
130. **<property** name="placeholderText"**>**
131. **<string>**Username**</string>**
132. **</property>**
133. **</widget>**
134. **</item>**
135. **</layout>**
136. **</widget>**
137. **<resources/>**
138. **<connections/>**
139. **</ui>**

### 3.2.8 config.json

1. {
2. "maxTransmissionSize": 40960,
3. "debug": **true**,
4. "commandChar": "/",
5. "port": 9998
6. }

## 3.3 Libs

### 3.3.1 configManager.py

1. **import** json
2. **import** os.path
4. **class** ConfigManager():
5. **def** \_\_init\_\_(self, file, defaultData):
6. self.data = self.loadJson(file)
7. **if** self.data == None:
8. self.initJson(file, defaultData)
9. self.data = self.loadJson(file)
11. **def** loadJson(self, file):
12. **if** os.path.isfile(file):
13. with open(file, "r") as jsonFile:
14. **return** json.load(jsonFile)
15. **else**:
16. **return** None
18. **def** initJson(self, file, defaultData):
19. jsonString = json.dumps(defaultData)
20. with open(file, "w") as jsonFile:
21. jsonFile.write(jsonString)

24. **class** ServerConfig(ConfigManager):
25. **def** \_\_init\_\_(self, file):
26. self.defaultData = {
28. "dbFile": "photon.db",
29. "infoLoggingEnabled": True,
30. "maxTransmissionSize": 40960,
31. "port": 9998
33. }
35. super().\_\_init\_\_(file, self.defaultData)

38. **class** ClientConfig(ConfigManager):
39. **def** \_\_init\_\_(self, file):
40. self.defaultData = {
42. "maxTransmissionSize": 40960,
43. "debug": True,
44. "commandChar": "/",
45. "port": 9998
47. }
49. super().\_\_init\_\_(file, self.defaultData)

### 3.3.1 packets.py

1. **class** Packet:
2. """
3. Base packet class. Can be used to signal something without the need to transfer dataself.
5. Args:
6. packetType (string): An identifier to determine what the packet denotes. Should be all caps.
7. """
8. **def** \_\_init\_\_(self, packetType):
9. self.type = packetType
11. **class** LoginRequestPacket(Packet):
12. """
13. Sends login details to the server.
15. Args:
16. username (string): The username to login with.
17. password (string): The password to login with. Should be hashed.
18. """
19. **def** \_\_init\_\_(self, username, password):
20. Packet.\_\_init\_\_(self, "LOGINREQUEST")
21. self.username = username
22. self.password = password
24. **class** LoginResponsePacket(Packet):
25. """
26. Sends login response to client.
28. Args:
29. valid (bool): Whether the login was successful.
30. err (string, optional): Why the login was not successful.
31. id (int, optional): The user id of the account, if successfully logged in.
32. admin (bool, optional): Whether the user account is an admin.
33. """
34. **def** \_\_init\_\_(self, valid, userId="", err="", admin=False):
35. Packet.\_\_init\_\_(self, "LOGINRESPONSE")
36. self.valid = valid
37. self.err = err
38. self.id = userId
39. self.admin = admin
41. **class** RegisterPacket(Packet):
42. """
43. Asks the server to create a user account.
44. Args:
45. username (string): The username of the account to create.
46. password (string): The password of the account to create. Should be hashed.
47. """
48. **def** \_\_init\_\_(self, username, password):
49. Packet.\_\_init\_\_(self, "CREATEUSER")
50. self.username = username
51. self.password = password
53. **class** RegisterResponsePacket(Packet):
54. """
55. Informs the client if the account was created succesfully.
57. Args:
58. valid (bool): Whether the account creation was a success.
59. err (string, optional): The reason why the account creation was not a success.
60. """
61. **def** \_\_init\_\_(self, valid, err=""):
62. Packet.\_\_init\_\_(self, "REGISTERRESPONSE")
63. self.valid = valid
64. self.err = err
66. **class** MessagePacket(Packet):
67. """
68. Sends a message to the client or the server.
70. Args:
71. message (photonUtilities.Message): The message to send.
72. """
73. **def** \_\_init\_\_(self, message):
74. Packet.\_\_init\_\_(self, "MESSAGE")
75. self.message = message # Utilises Message class
77. **class** MessageListPacket(Packet):
78. """
79. Sends multiple messages to the client or server.
81. Args:
82. messageList (list of photonUtilities.Message): The list of messages to send.
83. """
84. **def** \_\_init\_\_(self, messageList):
85. Packet.\_\_init\_\_(self, "MESSAGELIST")
86. self.messageList = messageList
88. **class** OnlineUsersPacket(Packet):
89. """
90. Sends a list of the usernames of all clients which are online.
92. Args:
93. userList (list of string): List of usernames of online clients.
94. """
95. **def** \_\_init\_\_(self, userList):
96. Packet.\_\_init\_\_(self, "ONLINEUSERS")
97. self.userList = userList
99. **class** UserListPacket(Packet):
100. """
101. Sends a list of all users registered.
103. Args:
104. userList (list of (int, string, bool)): A list of tuples conaining the userId, username and whether they're admin.
105. """
106. **def** \_\_init\_\_(self, userList):
107. Packet.\_\_init\_\_(self, "USERLIST")
108. self.userList = userList
110. **class** RequestUserInfoPacket(Packet):
111. """
112. Requests details about a specific user.
114. Args:
115. user (string): The username of the user to get the info of.
116. """
117. **def** \_\_init\_\_(self, user):
118. Packet.\_\_init\_\_(self, "REQUESTUSERINFO")
119. self.user = user
121. **class** UserInfoPacket(Packet):
122. """
123. Sends information about a specific user.
125. Args:
126. id (int): The id of the user
127. messageCount (int): The number of messages sent by the user.
128. admin (bool): Whether the user is an admin.
129. flags (list of (string, string, string, int): Contains info about reports for that user, corresponding to list of (reported message, report reason, reporter name, reporter id).
130. """
131. **def** \_\_init\_\_(self, id, messageCount, admin, flags):
132. Packet.\_\_init\_\_(self, "USERINFO")
133. self.id = id
134. self.messageCount = messageCount
135. self.admin = admin
136. self.flags = flags
138. **class** CommandPacket(Packet):
139. """
140. Contains information about an executed command
142. Args:
143. command (string): The command that was executed.
144. args (list of string): A list of the arguments supplied to the command.
145. """
146. **def** \_\_init\_\_(self, command, args=[]):
147. Packet.\_\_init\_\_(self, "COMMAND")
148. self.command = command
149. self.args = args
151. **class** CommandResponsePacket(Packet):
152. """
153. Contains information about the execution of a command.
155. Args:
156. command (string): The command that was executed.
157. success (bool): Whether the command was executed successfully.
158. err (string, optional): The reason why the command could not execute.
159. response (string, optional): The response of the command.
160. timeSent (string, optional): The time that the command was executed.
161. """
162. **def** \_\_init\_\_(self, command, success, err="", response="", timeSent=""):
163. Packet.\_\_init\_\_(self, "COMMANDRESPONSE")
164. self.command = command
165. self.success = success
166. self.err = err
167. self.response = response
168. self.timeSent = timeSent
170. **class** ReportPacket(Packet):
171. """
172. Tells the server to flag a message.
174. Args:
175. messageId (int): The id of the reported message.
176. reporterId (int): The id of the user who reported the message.
177. reportReason (string): The reason that the message was flagged.
178. """
179. **def** \_\_init\_\_(self, messageId, reporterId, reportReason):
180. Packet.\_\_init\_\_(self, "REPORTPACKET")
181. self.messageId = messageId
182. self.reporterId = reporterId
183. self.reportReason = reportReason
185. **class** DeleteMessagePacket(Packet):
186. **def** \_\_init\_\_(self, messageId):
187. Packet.\_\_init\_\_(self, "DELETEMESSAGE")
188. self.messageId = messageId
190. **class** EditMessagePacket(Packet):
191. **def** \_\_init\_\_(self, messageId, newContents):
192. Packet.\_\_init\_\_(self, "EDITMESSAGE")
193. self.messageId = messageId
194. self.newContents = newContents
196. **class** SetAdminStatusPacket(Packet):
197. **def** \_\_init\_\_(self, admin, userId):
198. Packet.\_\_init\_\_(self, "SETADMINSTATUS")
199. self.userId = userId
200. self.admin = admin

### 3.3.1 photonUtilities.py

1. **import** pickle
2. **import** traceback
3. **import** datetime
5. # Global Colours
6. BLACK = "#000000"
7. COMMANDERROR = "#ff3030"
8. INFO = "#636363"

11. **class** CircularQueue():
12. """
13. A custom circular queue implementation.
14. """
15. **def** \_\_init\_\_(self, maxSize):
16. **if** maxSize < 1:
17. **raise** ValueError("Queue size must be at least 1")
18. self.data = [''] \* maxSize
19. self.rear = -1
20. self.front= 0
21. self.size = 0
22. self.maxSize = maxSize
24. **def** enQueue(self, item):
25. """
26. Add an item to the queue.
28. Args:
29. item (\*): Item to add to the queue.
30. """
31. **if** self.size == self.maxSize:
32. **raise** ValueError("Cannot enqueue when the queue is full")
33. **else**:
34. self.rear = (self.rear + 1) % self.maxSize
35. self.data[self.rear] = item
36. self.size = self.size + 1
38. **def** deQueue(self):
39. """
40. Removes an item from the queue.
42. Returns:
43. (\*): The item at the front of the queue.
44. """
45. self.front += 1
46. self.size -= 1
47. **return** self.data[self.front-1]
49. **def** isFull(self):
50. """ Determines if the queue is full. """
51. **return** self.size == self.rear
53. **def** isEmpty(self):
54. """ Determines if the queue is empty. """
55. **return** self.size == 0

58. **class** Message():
59. """
60. Represents a message.
62. Args:
63. senderId (int): The user id of the message sender.
64. senderName (string, optional): The username of the
65. contents (string, optional): The actual message content
66. timeSent (string, optional): The time that the message was sent.
67. recipientId (int, optional): The user id of the user that sent the message.
68. colour (string, optional): The colour to display the message as.
69. messageId (int, optional): The id of the message.
70. """
71. **def** \_\_init\_\_(self, senderId="", senderName="", contents="", timeSent="", recipientId=1, colour="#000000", messageId=""):
72. self.senderId = senderId
73. self.senderName = senderName
74. self.contents = contents
75. self.timeSent = timeSent
76. self.recipientId = recipientId
77. self.colour = colour
78. self.messageId = messageId

81. **def** hashString(string):
82. """
83. Custom implementation of a simple one way hashing algorithm
85. Args:
86. string (string): The string to hash.
88. Returns:
89. (string): The hashed string.
90. """
91. bitValueChunk = ""
92. bitSum = 0
94. **for** char **in** string:
95. bitSum += ord(char)
96. bitValue = format(ord(char), 'b') # Convert char to binary
97. bitValueChunk += bitValue # Apend to 'binary chunk'
99. n = 9
100. bitValues = [bitValueChunk[i:i+n] **for** i **in** range(0, len(bitValueChunk), n)] # Split 'binary chunk' into list of 9 bit binary numbers
102. moddedBitChunk = ""
103. **for** bitValue **in** bitValues:
104. bitValue = int(bitValue)
105. moddedBitValue = bitValue + (bitValue % 37) # Modulo is a one way function, so we modulo by a prime as the core of the hash. This is the step that ensures the hash is unidirectional
106. moddedBitValue = moddedBitValue \* bitSum # Multiply by sum of the ascii values of the chars to ensure similar input strings look different
107. moddedBitChunk += format(moddedBitValue, 'b') # Convert into binary again
109. n = 6 # 6 bit chunks to avoid strange characters
110. moddedBitValues = [moddedBitChunk[i:i+n] **for** i **in** range(0, len(moddedBitChunk), n)] # Split modded 'binary chunk' into list of 8 bit binary numbers
112. hashed = ""
113. **for** moddedBitValue **in** moddedBitValues:
114. hashed += chr(int(moddedBitValue, 2) + 33) # Convert binary value into decimal value then into the corresponding character, skipping the first 33 as they are non-printing/whitespace
116. **return** hashed

119. **def** integerMergeSort(mergelist):
120. """
121. Custom implementation of a mergesort algorithm for integers.
123. Args:
124. mergeList (list of int): The list to sort.
126. Returns:
127. (list of int): The sorted list.
128. """
129. **if** len(mergelist) > 1:
130. mid = len(mergelist) // 2 # Perform integer division
131. lefthalf = mergelist[:mid] # Left half of merglist into lefthalf
132. righthalf = mergelist[mid:] # Right half of merglist into righthalf
133. lefthalf = MergeSort(lefthalf)
134. righthalf = MergeSort(righthalf)
136. i = 0
137. j = 0
138. k = 0
139. **while** i < len(lefthalf) **and** j < len(righthalf):
140. **if** lefthalf[i] < righthalf[j]:
141. mergelist[k] = lefthalf[i]
142. i += 1
143. **else**:
144. mergelist[k] = righthalf[j]
145. j += 1
146. k += 1
148. # Check if left half has elements not merged
149. **while** i < len(lefthalf):
150. mergelist[k] = lefthalf[i] # If so, add to mergelist
151. i += 1
152. k += 1
153. # Check if right half has elements not merged
154. **while** j < len(righthalf):
155. mergelist[k] = righthalf[j] # If so, add to mergelist
156. j += 1
157. k += 1
158. **return** mergelist

161. **def** stringListMergeSort(mergelist):
162. """
163. Custom implementation of a mergesort algorithm for strings.
165. Args:
166. mergeList (list of string): The list to sort.
168. Returns:
169. (list of string): The sorted list.
170. """
171. **if** len(mergelist) > 1:
172. mid = len(mergelist) // 2 # Perform integer division
173. lefthalf = mergelist[:mid] # Left half of merglist into lefthalf
174. righthalf = mergelist[mid:] # Right half of merglist into righthalf
175. lefthalf = stringListMergeSort(lefthalf)
176. righthalf = stringListMergeSort(righthalf)
178. i = 0
179. j = 0
180. k = 0
181. **while** i < len(lefthalf) **and** j < len(righthalf):
182. l = 0
183. **while** ord(lefthalf[i][l]) == ord(righthalf[j][l]) **and** l < len(lefthalf[i])-1 **and** l < len(righthalf[j])-1: # If the charachers are the same, we must look at the next one until the end
184. l += 1
185. **if** ord(lefthalf[i][l]) < ord(righthalf[j][l]):
186. mergelist[k] = lefthalf[i]
187. i += 1
188. **else**:
189. mergelist[k] = righthalf[j]
190. j += 1
191. k += 1
193. # Check if left half has elements not merged
194. **while** i < len(lefthalf):
195. mergelist[k] = lefthalf[i] # If so, add to mergelist
196. i += 1
197. k += 1
198. # Check if right half has elements not merged
199. **while** j < len(righthalf):
200. mergelist[k] = righthalf[j] # If so, add to mergelist
201. j += 1
202. k += 1
203. **return** mergelist

206. **def** reportError(exception="", logger=None):
207. """ Easier name for traceback.print\_exc(). """
208. traceback.print\_exc()
209. **if** logger != None:
210. logger.log(repr(exception))

213. **def** getDateTime():
214. """ Gets the current time in yy/mm/dd HH:MM format. """
215. **return** datetime.datetime.now().strftime("%y-%m-%d %H:%M")

218. **def** formatUsername(name):
219. """
220. Formats a string to confirm with username display style.
222. Args:
223. name (string): The name to format.
225. Returns:
226. (string): The formatted string.
227. """
228. **if** name == "" or name == "SERVER":
229. **return** ""
230. **else**:
231. **return** "<" + name + ">: "

234. **def** formatDateTime(time):
235. """
236. Formats a string to confirm with time display style.
238. Args:
239. time (string): The time to format.
241. Returns:
242. (string): The formatted string.
243. """
244. **if** time == "":
245. **return** ""
246. **else**:
247. **return** "" + time + " | "

250. **def** generateJoinLeaveMessage(direction, username):
251. """
252. Generates a standard message notifying of when users join/leave the server.
254. Args:
255. direction (string): Contains whether the user 'joined' or 'left' the server.
256. username (string): The user who joined or left.
258. Returns:
259. (Message): The generated message.
260. """
261. **return** Message(1, "SERVER", "\_" + username + " has " + direction + " the server\_", getDateTime(), colour=INFO)

264. **def** debugPrint(message, debug):
265. """
266. Conditional Print depending on whether launched in debug mode or not.
268. Args:
269. message (string): The message to print.
270. debug (bool): Whether the message should be printed.
271. """
272. **if** debug:
273. **print**(message)

276. **def** encode(packet):
277. """ Better name for pickle.dumps() """
278. **return** pickle.dumps(packet)
279. **def** decode(packet):
280. """ Better name for pickle.loads() """
281. **return** pickle.loads(packet)