

Messages Exchange Platform using C++

Group No.30 Members:

Hannah Ashna Jacob (N0865554)

Jarad Johnson Bailey (N0853071)

Hassaan Naveed (N0898071)

Nicholas McCaig (N0787115)

Lab Tutor:

Pedro Baptista Machado



SOFT 20091 Software Design and Implementation

NOTTINGHAM TRENT UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

United Kingdom

April 2021

Project Abstract:

The Covid-19 pandemic has significantly changed the way we communicate. Our reliance on digital applications for communication has increased, and thus, the demand for such applications has also grown. Our team was tasked with designing and implement a messages exchange platform using C++ for our Software Design and Implementation (SDI) module. This project uses MQTT to facilitate communication using a Client-Server model. We have based our application on two existing messaging services, Discord and Slack. Discord is a gaming-oriented application with a range of features that allows users to communicate with others while in-game (Discord, 2021). On the other hand, slack is a messages exchange platform service that is geared towards workplace communication (Slack, 2021).

The overall aim of our project is to allow users to log in to our platform using their credentials, connect to the service and begin exchanging messages amongst each other. Users can create and delete chat rooms. Within each room, a user can have multiple discussion channels wherein multiple conversations can run concurrently. The user can switch between these channels depending on which conversation they wish to participate in. Moreover, the application also has admin and moderator roles so that users within a room can control who is added or removed from a room. In addition to this, our team also implemented password encryption capabilities to ensure that user accounts were secure since they had to be stored locally.

Through our team's evaluation of the project upon completion, we found that we were able to effectively design and implement the application within the given time frame due to our detailed research process and diligent adherence to our project plan. We did conclude that for future iterations of this project it would be more effective to use a database instead of text files as this would improve the overall efficiency of the project and prevent data from being stored in numerous locations.

Plagiarism Declaration:

This report and the software it documents is the result of my own work. Any contributions to the work by third parties, other than tutors, are stated clearly below this declaration. Should this statement prove to be untrue I recognise the right and duty of the Board of Examiners to take appropriate action in line with the university's regulations on assessment.

Name: Hannah Ashna Jacob (ID – N0865554)

Name: Jarad Johnson-Bailey (ID - N0853071)

Name: Hassaan Naveed (ID – N0898071)

Name: Nicholas McCaig (ID – N0787115)

Revision History:

Version	Issue Date	Stage	Changes	Author
0.1	05/04/21	Draft V1.1	Draft Created	Hannah
0.2	05/04/21	Draft V1.1	Structured report template per guidelines	Hannah
0.3	05/04/21	Draft V1.1	Added Risk Analysis, Requirements List, Contribution guide and Coding Standards guide	Hannah
0.4	06/04/21	Draft V1.1	Added Abstract and Introduction write-up	Hannah
0.5	07/04/21	Draft V1.1	Added Gantt Chart	Hannah
1.1	08/04/21	Draft V1.2	Added Diagrams	Hannah
1.2	08/04/21	Draft V1.2	Added GUI write-up	Hassaan
1.3	10/04/21	Draft V1.2	Modified Deployment Diagram	Nick
1.4	10/04/21	Draft V1.2	Added Design Pattern write-up	Nick
2.1	11/04/21	Draft V1.3	Added Background Research	Hassaan
2.2	11/04/21	Draft V1.3	Added Monitoring Tools write-up	Hannah
2.3	12/04/21	Draft V1.3	Added User Manual	Jarad
2.4	13/04/21	Draft V1.3	Added Code Review	Hassaan
2.5	15/04/21	DraftV1.3	Added Planned Architecture write-up	Nick
3.1	17/04/21	DraftV1.4	Added Conclusion	Hannah
3.2	18/04/21	DraftV1.4	Added Individual Contributions	Hannah
4.1	20/04/21	DraftV1.5	Added Table of Contents, List of Tables and List of Images	Hannah
5.1	25/04/21	DraftV1.6	Added Overall Reflection	Hannah
5.2	27/04/21	DraftV1.6	Added Test Table	Jarad
5.3	29/04/21	DraftV1.6	Added Introduction	Hannah
5.4	29/04/21	DraftV1.6	Reviewed and Updates Test Table	Jarad
6.1	30/04/21	1 st Release	Reviewed entire document	Everyone
7.1	01/05/21	Final Release	Final Review	Everyone

Table of Contents

Project Abstract:	2
Plagiarism Declaration:	3
Revision History:	4
List of Tables:	7
List of Images:	8
Introduction:	9
Background Research.....	10
List of Requirements & Tasks:.....	11
Risk Analysis:	14
Gantt Chart:	16
Monitoring Tools:.....	20
Diagrams:	21
Use Case Diagrams:.....	21
Introduction:	21
Activity Diagrams:	24
Introduction:	24
Class Diagram:	28
Introduction:	28
Sequence Diagrams:.....	30
Introduction:	30
Component Diagram:.....	34
FSM Diagrams:	35
Introduction:	35
Communication Diagram:	39
Deployment Diagram:.....	40
GUI Mock-Up:.....	41
Introduction:	41
Final GUI Design:	42
Design Pattern:	43
Planned Architecture:	44
Code Review.....	45
Test Plans:	46

Conclusions & Future Work:	48
References	49
Individual Contributions:	50
Overall Reflection:.....	51
Appendix:	52
Risk Analysis Scale:.....	52
User Manual:.....	53
Start-up:	53
Setting up your Connection:	54
Navigation:	55
Create Room	56
Create Channel.....	56
Send Message	56

List of Tables:

Table 1 REQUIREMENTS TABLE.....	13
Table 2 RISK ANALYSIS TABLE	15
Table 3 GANTT CHART.....	19
Table 4 DESIGN PATTERN ANALYSIS	43
Table 5 TESTING OVERVIEW TABLE	47
Table 6 PROBABILITY AND IMPACT WEIGHTING	52

List of Images:

Figure 1 LOGIN USE CASE.....	22
Figure 2 SEND MESSAGE USE CASE.....	22
Figure 3 MAKE ROOM USE CASE.....	23
Figure 4 LOGIN ACTIVITY DIAGRAM.....	25
Figure 5 SEND MESSAGE ACTIVITY DIAGRAM.....	26
Figure 6 MAKE ROOM ACTIVITY DIAGRAM.....	27
Figure 7 CLASS DIAGRAM.....	28
Figure 8 LOGIN SEQUENCE DIAGRAM.....	31
Figure 9 SEND MESSAGE SEQUENCE DIAGRAM	32
Figure 10 MAKE ROOM SEQUENCE DIAGRAM	33
Figure 11 COMPONENT DIAGRAM.....	34
Figure 12 LOGIN FSM DIAGRAM	36
Figure 13 PUBLISHER FSM DIAGRAM.....	37
Figure 14 SUBSCRIBER FSM DIAGRAM.....	38
Figure 15 SEND MESSAGE COMMUNICATION DIAGRAM.....	39
Figure 16 DEPLOYMENT DIAGRAM.....	40
Figure 17 GUI MOCK-UP	41
Figure 18 LABELLED MAIN PAGE.....	42
Figure 19 LABELLED ADMIN VIEW	42
Figure 20 LOGIN PAGE	53
Figure 21 EXAMPLE OF USER INPUTTING CREDENTIALS TO LOGIN.....	53
Figure 22 EXAMPLE OF USER CREATING AN ACCOUNT.....	54
Figure 23 EXAMPLE OF USER ESTABLISHING THEIR CONNECTION.....	54
Figure 24 LABELLED GUI.....	55
Figure 25 ROOM CREATION POP-UP.....	56
Figure 26 GROUPS DROPDOWN MENU	56
Figure 27 CHANNELS DROPDOWN MENU	56
Figure 28 CHANNEL CREATION POP-UP	56
Figure 29 MESSAGE BAR	56
Figure 30 NOTIFICATION POP-UP	56
Figure 31 MESSAGE BEING DISPLAYED	56

Introduction:

Our world's communication methods have significantly evolved over time, from the use of letters to a single handheld device with thousands of communication channels available at the touch of a button. The ongoing Covid-19 pandemic has further highlighted the importance of high-quality message exchange platforms and the importance they play in upholding our social connections to one another.

Our team's aim for this project is to conduct our own research into existing message exchange solutions and to use this information to implement our own take on a messages exchange application. This is so that as software designers and developers, we can hone the skills we have acquired over this academic year through the creation and development of this project and its report.

Upon our initial stages of market research, we found that there are two main platforms that team intends to use as reference points for our application. These platforms being Discord (Discord, 2021) and Slack (Slack, 2021). Discord is a platform designed with the gaming community in-mind while Slack is designed with a corporate user-base in-mind instead.

Background Research

Throughout the implementation of the project, we made use of external libraries, to extend the functionality of the C++ language and to accomplish the goals of the project in a more efficient, modular way. The main libraries we used were: **Qt** (The Qt Company, 2020), **QtMqtt** (The Qt Company, 2020), **Boost** (Boost.org, 2021), and **Crypto++** (Dai, 2021).

Qt is an extensive library that provides a variety of different tools and features to the C++ language, such as GUI's, new data types, networking tools, and more. It also offers programs such as Qt Creator and Designer, which work in conjunction with each other to make UI design and implementation easier. We used a variety of classes offered in Qt libraries, most notably for creating the GUI of the application. We used Qt Designer to develop the layout of the UI, making use of the Widgets it offered to handle input and output. We integrated this with our program through Qt Creator and the QMainWindow class. In addition to this we used various data types offered by Qt such as QStrings, which enable new string handling methods, and QDateTime, for storing the users' local date and time. In addition to this, we have also used Qt to implement automated unit testing, creating test cases for each component of the application, allowing us to easily test and debug the program.

In addition to this, we also used an MQTT extension for the Qt libraries called **QtMqtt**. This library, offered as an add-on to the 'Qt for Automation' package, enables MQTT functionality for C++. This library was essential for the application as the base functionality of sending and receiving messages is powered through MQTT. We decided to use QtMqtt over other MQTT brokers, such as Eclipse Mosquitto, as it would be easier to implement and integrate with our use of Qt Creator and the Qt libraries, and its more thorough documentation.

Boost is a large collection of open-source, peer-reviewed C++ libraries that complement the C++ standard library, while vastly expanding on the features that C++ offers. It provides features such as new algorithms, iterators and functions. We decided to use this library as it would save a large amount of time, both in programming and fixing bugs, as it is a high-quality library written to be as efficient as possible. Our main usage of boost was for handling data types more efficiently, for example splitting strings by a delimiter into a vector, through the String Algorithms that Boost offers, or casting certain data types to others through Lexical Cast.

Finally, **Crypto++** is an open-source library that provides a large range of cryptographic algorithms and schemes. For example, it offers various encryption algorithms, cryptographically secure pseudo-random number generators, various arithmetic operations, and some non-cryptographic functions such as hexadecimal encoding. We decided to use this library as we wanted to implement a safer way of storing passwords, rather than in plaintext. Our main uses for Crypto++ were for hashing a given password with a SHA256 algorithm and encoding it to hexadecimal for storing in an external file.

List of Requirements & Tasks:

No	Requirement	Priority	Implications	Tasks
1	Users must be able to send, receive and view messages through the application	MUST	Without this feature, the application's primary function would not be possible as a message exchange platform relies on users being able to send and receive messages	T2.2.1: Setup and Configure Client/Server using MQTT T4.1.1: Implement User Interface
2	Users must be able to create chat rooms (rooms with more than two contacts)	MUST	A key feature in a messaging platform is the ability to chat with more than one person simultaneously, hence, the need for chat rooms	T2.2.2: Setup and Configure Client/Server using MQTT T3.3.1: Create separate user classes
3	The user that creates the chat room must be classified as Admin	MUST	This is to prevent other users from making modifications to a chat room when they did not create it	T3.1.1: Create separate user classes
4	The moderator must be able to invite and remove users from a chatroom	MUST	This allows for a chat room to expand its user base numbers or remove certain members if necessary	T3.2.2: Create separate user classes
5	Moderators must inherit all the admin permissions; however, Moderators cannot demote the Admin	MUST	The functionality of the Moderator and Admin is essentially the same, however, the Admin is the owner of the chat room, hence, moderators should not be allowed to demote or remove them as owner	T3.2.1: Create separate user classes
6	The application must provide a friendly User Interface (UI)	MUST	A welcoming and friendly UI ensures user retention and ease of use. The UI should not drive users away due to its complexity	T1.1.1: Design User Interface
7	Users must be able to see the active users in the chat room	MUST	Allowing a user to view other active users allows them to know who is available to chat	T4.1.3: Implement User Interface
8	Users must be notified when a new notification is received	MUST	A notification system ensures that users are always up to date with new conversations happening in their chats and chat rooms	T5.1.1: Implement event listeners
9	Clients must not connect directly to other clients without a server or a broker	MUST NOT	This is necessary as it would otherwise present itself as a security risk	T2.1.1: Setup and Configure Client/Server using MQTT
10	A server or a broker must allow multiple authorised clients to connect to it	MUST	A message exchange platform is going to have several users on it; hence, the broker must be capable of connecting with multiple clients	T2.1.2: Setup and Configure Client/Server using MQTT
11	Users must only access their space after the login	MUST	Having users access another user's space would breach security and privacy protocols	T6.1.2: Implement security features and protocols

12	Passwords must be saved securely locally	MUST	This is necessary as it would otherwise present itself as a security risk	T6.1.1: Implement security features and protocols T9.1.1: Setup a local text-file
13	The application must adhere to all local (and international) privacy laws	MUST	With laws like the General Data Protection Regulation (GDPR), modern-day applications must respect and handle user data appropriately	T6.1.3: Implement security features and protocols
14	The application must list all the personal contacts in the contacts pane	MUST	Users want to be able to view their contacts list and start a chat easily, this method provides them with the necessary tools in an intuitive manner	T4.1.2: Implement User Interface
15	The admin should be able to promote and demote users to moderators in chat rooms	SHOULD	This allows the Admin to provide chosen users with privileges to make modifications to the specific chat room; especially useful in cases where it is a large chat room, and the Admin cannot manage this on their own	T3.1.2: Create separate user classes
16	The moderator should be able to create and delete channels in the chatroom	SHOULD	Creating discussion threads within a chat room prevents the primary discussion thread from being flooded with several concurrent conversations	T3.2.3: Create separate user classes
17	Users should be able to change their status	SHOULD	Users may be preoccupied with other tasks or do not wish to be disturbed, hence, would like the ability to modify their availability status within the application	T7.1.1: Develop user profile and control panel
18	Messages should be sent and received within 5-10 seconds	SHOULD	A performance requirement so that users do not spend too long waiting on a response from a contact	T2.3.1: Setup and Configure Client/Server using MQTT
19	Users should be logged off automatically after a specific amount of time	SHOULD	Prevents the system from being burdened by a user that is not active. This also doubles as a security feature to prevent an unsupervised account from being compromised	T6.2.1: Implement security features and protocols
20	The moderator could be able to delete a user's messages in the chatroom	COULD	In the case a user sends an inappropriate message to a chat room, the moderator can then remove it immediately to avoid the message distressing other chat room members	T3.2.4: Create separate user classes T10.5.3: Consider the implementation of additional features
21	The user could be able to change their details including their picture	COULD	A customization feature that allows a user to share their details with their contacts (i.e., profile photo, name, email)	T10.1.1: Consider the implementation of additional features
22	User pictures could be displayed in the channels	COULD	An alternative identification method to just using a user's name	T4.2.1: Implement User Interface T10.1.2: Consider the implementation of additional features

23	The application could allow the exchange of files with contacts	COULD	Users may wish to share images or other files with another user, hence the need for file transfer capabilities between contacts	T8.0.1: Implement file transfer feature T10.6.1: Consider the implementation of additional features
24	Users could be able to send emoji's	COULD	Implements an interactivity feature to the application, allowing users to express themselves better	T10.3.4: Consider the implementation of additional features
25	Messages could come with sent and read receipts	COULD	Allows a user to know who has read their message and who has not. This is especially useful for urgent or messages of high importance	T10.2.1: Consider the implementation of additional features
26	The application could display the full history of the conversation when a specific contact is selected	COULD	Conversation history allows a user to rely on the system for information they may have forgotten about, hence, allowing them to refer to their older conversations	T2.4.1: Setup and Configure Client/Server using MQTT T10.5.2: Consider the implementation of additional features
27	Offline messages could be stored on the client-side and transmitted to the target user(s) once they are online	COULD	A user may not have internet access but would still like to send a message, this method ensures that as soon as they are connected to the internet, they can send those pending messages	T9.2.1: Setup a local text-file T10.5.1: Consider the implementation of additional features
28	The application could run on both macOS and Windows-based desktops and laptops	COULD	Not all users use windows devices, hence, the need for an application that runs on other operating systems	T10.4.1: Consider the implementation of additional features
29	The application could have a light and dark mode	COULD	Users with visual impairments may require alternative application colour schemes to make the application usable	T4.3.1: Implement User Interface T10.3.1: Consider the implementation of additional features
30	The application could have text-size customisation	COULD	Users with visual impairments may require alternative text sizes to be able to read their chat messages	T4.3.2: Implement User Interface T10.3.2: Consider the implementation of additional features
31	The application could have several language options	COULD	International users may require an application with features written in a language they are familiar with to use it	T4.4.1: Implement User Interface T10.3.3: Consider the implementation of additional features

Table 1 REQUIREMENTS TABLE

Risk Analysis:

The risk analysis aspect of the project design stage involves reviewing and planning out solutions for potentials issues that could put the project's success at risk. Highlighted below at several risks ranging from low to high probability and impact; they are each accompanied by a mitigation plan.

Risk Number	Description of Risk	Probability	Impact	Mitigation Plan
1	Unclear or unrealistic requirements and scope	2	5	Ensure that all requirements are reviewed by all members of the team and are thoroughly discussed before being confirmed. In addition to this, actively seek out support from experts (i.e., lecturers and industry professionals) to ensure that the project scope is attainable within the allocated development window
2	Insufficient knowledge and background research of messaging applications	3	4	Carry out an intensive research process before beginning the development process to ensure that all team members are well informed
3	Security breach due to passwords being compromised	2	5	Add password encryption and (if feasible) two-factor authentication.
4	Team member falls ill due to ongoing pandemic or is otherwise unable to support the team due to extenuating circumstances	3	4	Promote the Software Tester to the role available.
5	Lost data due to technical failure	2	4	Make regular backups to GitHub and other cloud storage options used by the team.
6	Tasks go over the allotted time	2	3	Give buffer for extra time at end of the project - work to a week before the actual deadline
7	Team member overwrites an existing file's contents on accident	3	1	Regularly use version control software (i.e., Git - GitHub) so that the file contents can be easily reverted to an older version
8	Users struggle to use the application due to the unintuitive user interface (UI)	2	2	Ensure that during the testing stages user feedback is gathered with regards to the usability of the UI
9	Major bug found in the testing stage	2	2	Agile development allows for regular testing to prevent large scale bugs at the end of the project.

10	Team member struggles to engage with the group or is not actively communicating with the rest of the team	3	4	The team must have frequent check-ins to ensure how all team members are handling their workload and if anyone requires assistance with managing their tasks, they are encouraged to seek help from the rest of the team.
11	Member conflict occurs due to differing opinions	3	5	The team makes use of the quality vote to make the final decision to resolve the conflict
12	Team experiences issues with computational resources	1	3	Contact the team's assigned tutor for support in gaining access to university resources.
13	Member experiences issues with handling the workload	3	4	The team reviews the assigned task to break it down amongst other members to help support the struggling member

Table 2 RISK ANALYSIS TABLE

Gantt Chart:

Tasks	December		January				February				March				April			
	W1	W2	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4
Formulate Requirements List Table																		
Formulate Risk Analysis Table																		
Formulate Gantt Chart Table																		
Compile and Review Documentation for Project Plan																		
Milestone: Creation and completion of Requirements List, Risk Analysis and Gantt Chart tables																		
Deliverable 1: Project Plan																		
Design Use Case, Activity and Class Diagrams																		
Design Sequence, Component and FSM Diagrams																		
Design Communication and Deployment Diagrams																		
Milestone: Completion project's design phase (includes several types of diagrams)																		
Deliverable 2: Project Design Document																		
T1.1.1: Design User Interface - Finalise User-Friendly UI for Application																		
T2.1.1: Setup and Configure Client/Server using MQTT - Must include a server or a broker																		

[illegible]

Monitoring Tools:

Our group aims to employ the use of several key monitoring tools to ensure that our team continues to develop and complete our project according to our outlined Gantt Chart and goals.

Our Tools:

- **Trello:** The team decided to use Trello due to its incredibly comprehensive design and capabilities for teamwork and collaboration. We intend to use the in-built task tracking system to help hold all team members accountable throughout the entire process.
- **Discord:** For our communications platform, the team decided to go with Discord as we are extremely familiar with it and it is an application that we use quite often. Hence, ensuring that all updates sent to the discord server's channels will be reviewed within a 24-hour window. Thus, ensuring that all team members are up to date with any ongoing changes.
- **GitHub:** To ensure that all the work is frequently backed up and that we can review the contribution of each member we have decided to use GitHub as our version control solution. The platform will allow for us to collaborate while being unable to meet in person due to the ongoing pandemic while also acting as a safeguard in case one of us experiences a corruption of files or accidentally overwrites one of our files.

Diagrams:

Use Case Diagrams:

Introduction:

Figure 1 (Login)

This diagram shows the login process. It assumes that the user is valid and will use their credentials to log in. The credentials introduced by the user are then validated by the text file.

Figure 2 (Send Message)

A user attempts to send a message, the system will first need to validate their connection to ensure that the target user is online and connected to the broker.

If the user is online:

- The source message is sent, and the source chat history is updated
- The target user is notified, and the target user's chat history is also updated
- The chat histories on both ends are updated within the text file

If the user is offline:

- The source message is stored within the text file temporarily
- The source user's connection is then repeatedly checked until they are confirmed to be online
- Once the source user is online, their message is then sent to the target user, and the following stages mentioned above occur

Figure 3 (Make Room)

The diagram shows the process that occurs when a user makes a new room. The user is assumed to be Admin as by making a room the user becomes that Room's Admin by default. The new room is made, and the text file is updated. The Admin then has the option to add users to the room and make these users moderators. The Admin can also make channels in the room. Whenever any changes are made the text file is updated.

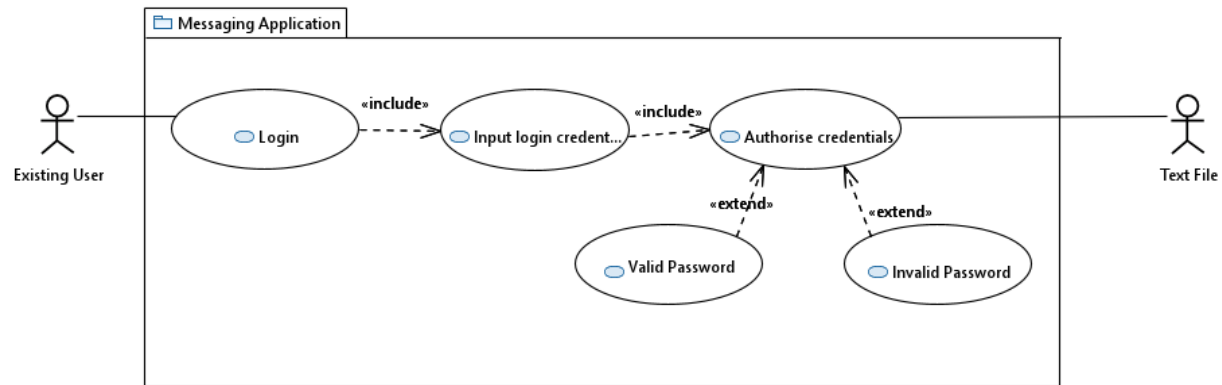


Figure 1 LOGIN USE CASE

The login use case includes the input login credentials. Input login credentials includes authorise credentials. Authorise credentials extends to valid password. Authorise credentials also extends to invalid password.

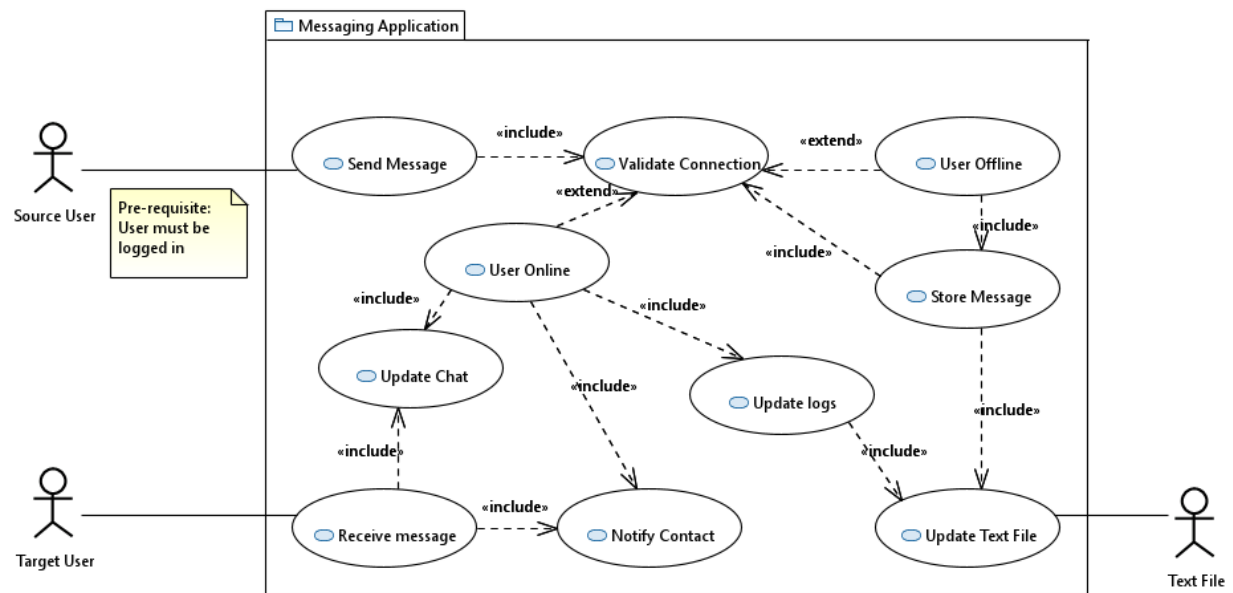


Figure 2 SEND MESSAGE USE CASE

Source user actor is associated with the send message use case. Send Message includes validate connection. Validate connection extends to user offline. User offline includes store message. Store message includes update text file. It also includes validate connection. Validate connection also extends to user online. User online includes update logs. Update logs includes update text file. Update text file is associated with the Text File actor. User online includes notify contact. User online also includes update chat. Receive message includes update chat. Receive message also includes notify contact. Target User actor is associated with the receive message use case. Admin actor is associated with make room use case. Make room use case includes update text file. Update text file is associated with the text file actor. Make room extends to create channel. Create channel includes update text file.

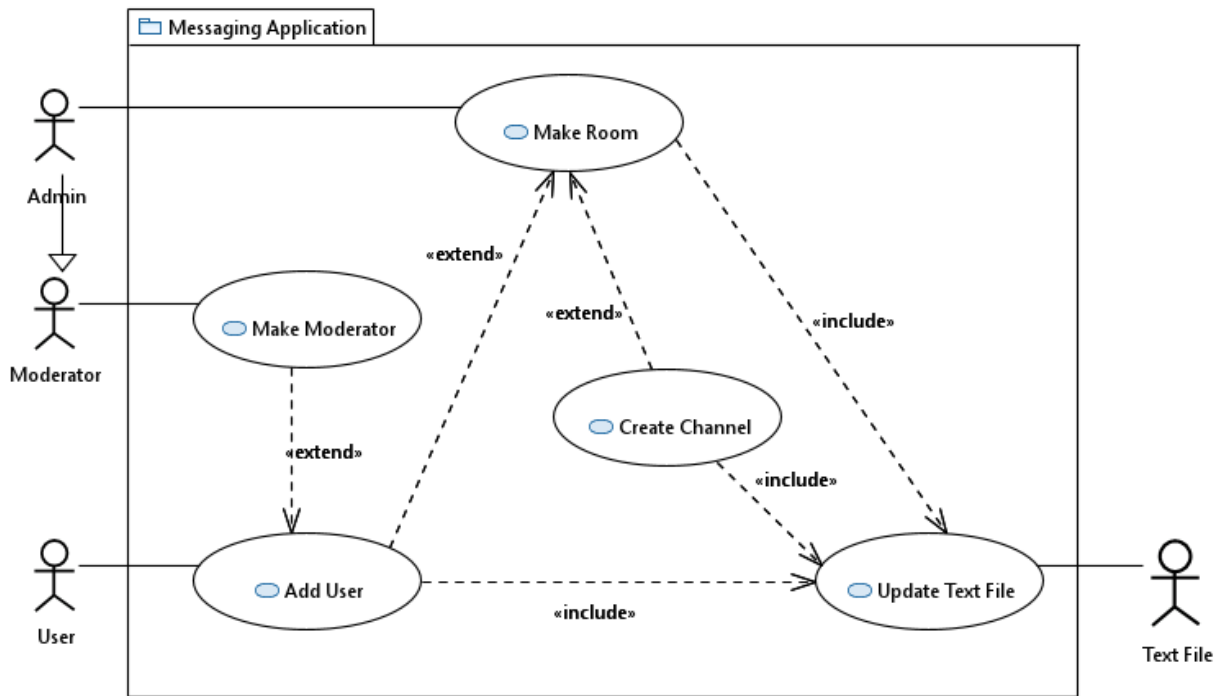


Figure 3 MAKE ROOM USE CASE

Make room also extends to add user. Add user is associated with the user actor. Add user also extends to make moderator. Make moderator is associated with the moderator actor. The admin is a child of the moderator.

Activity Diagrams:

Introduction:

Figure 4 (Login)

In this scenario, the User attempts to login into the application by inputting the User credentials (i.e., Username and Password). The credentials are then authenticated against the existing credentials within the text file.

- If the password is valid, the user gets access to their user space
- If the password is invalid, the user is notified of their credentials being incorrect and are asked to try again.

Figure 5 (Send Message)

The activity diagram covers the flow of events required to send a message. The user triggers an event when sending a new message using the application. The source user's status is validated to ensure they are connected to the broker. If the source user is offline, the message is temporarily stored in the text file. The application keeps checking the status of the users every 10 seconds. Once the source user's connection is established as being online, the system retrieves any messages that are stored in the text file. A queue of messages to be sent is created and messages are ordered accordingly in a first in, first out fashion. A fork then occurs for several activities to run concurrently, such as notifying the target user that a new message has been received, updating chat logs, and updating the chat history itself. Once these are all completed, they merge back to end the 'send message' activity.

Figure 6 (Make Room)

The activity diagram shows the process that occurs when a user makes a new room. First, the user is promoted to room Admin. The Admin is then given the option to add a user to the room. If they do, they are then given the option to make the user a moderator. Regardless of if they make the new user moderator or not, the Admin is given the option of adding a user again, looping back. Once the Admin has finished adding users, they are then given the option to add a channel to the room. If they choose yes, they can add the channel and then the option is given again, the same as when adding users. The text file is then updated, and process concludes.

While the add user and add channel options should not be unique to this process, they are still part of the process for making a new room and should be given immediately.

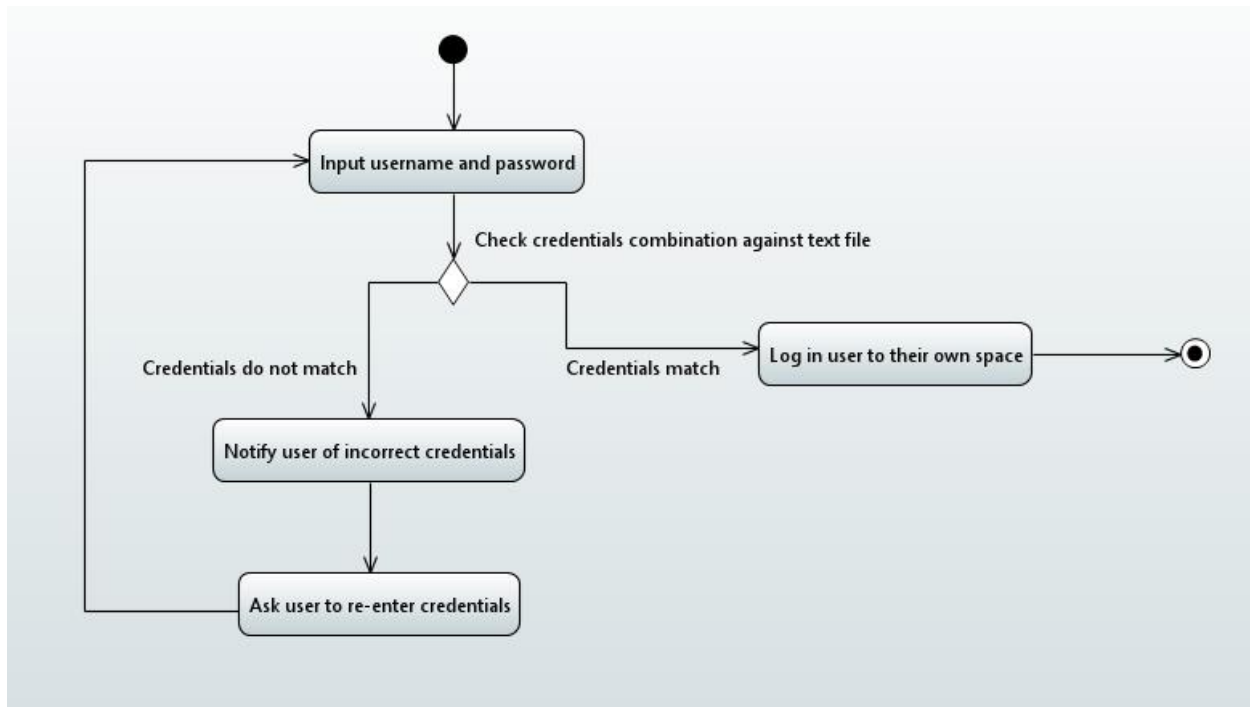


Figure 4 LOGIN ACTIVITY DIAGRAM

The login activity is initiated. User inputs their username and password. Control flow points to a decision node to check credentials combination against text file. If credentials match the control flow logs the user into their own space. Control flow points to activity final node and stops all control flows. If credentials do not match control flow notifies user of incorrect credentials. Control flow then asks the user to re-enter their credentials. The control flow then loops back to the input username and password activity.

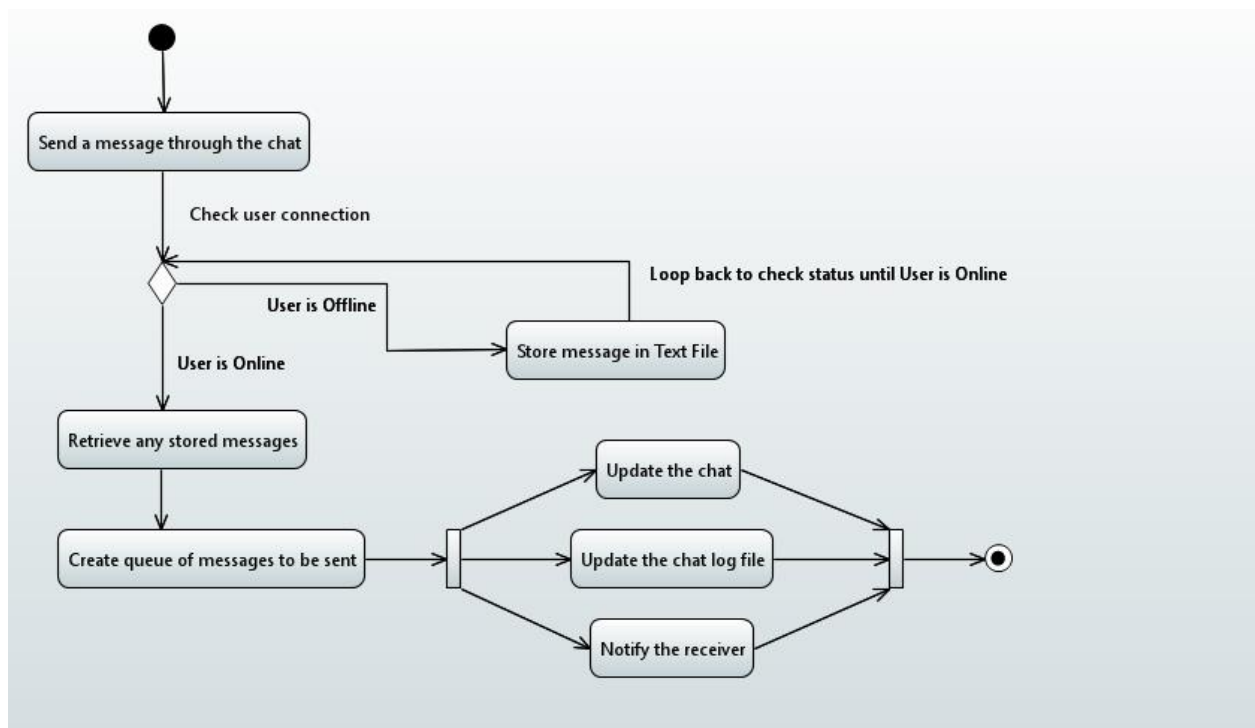


Figure 5 SEND MESSAGE ACTIVITY DIAGRAM

The send message activity is initiated. The control flow points to the send a message through the chat activity. Control flow moves then moves onto a decision node to check user's connection. If the user is offline, the control flow moves to the store message in text file activity. Control flow then loops back to check user connection. If the user is online, the control flow moves to the retrieve any stored messages activity. It then flows to create a queue of messages to be sent. Control flow forks at this point to run three activities concurrently, update the chat, update the chat log file, and notify the receiver. The control flow merges back and points to the activity final node and stops all control flows.

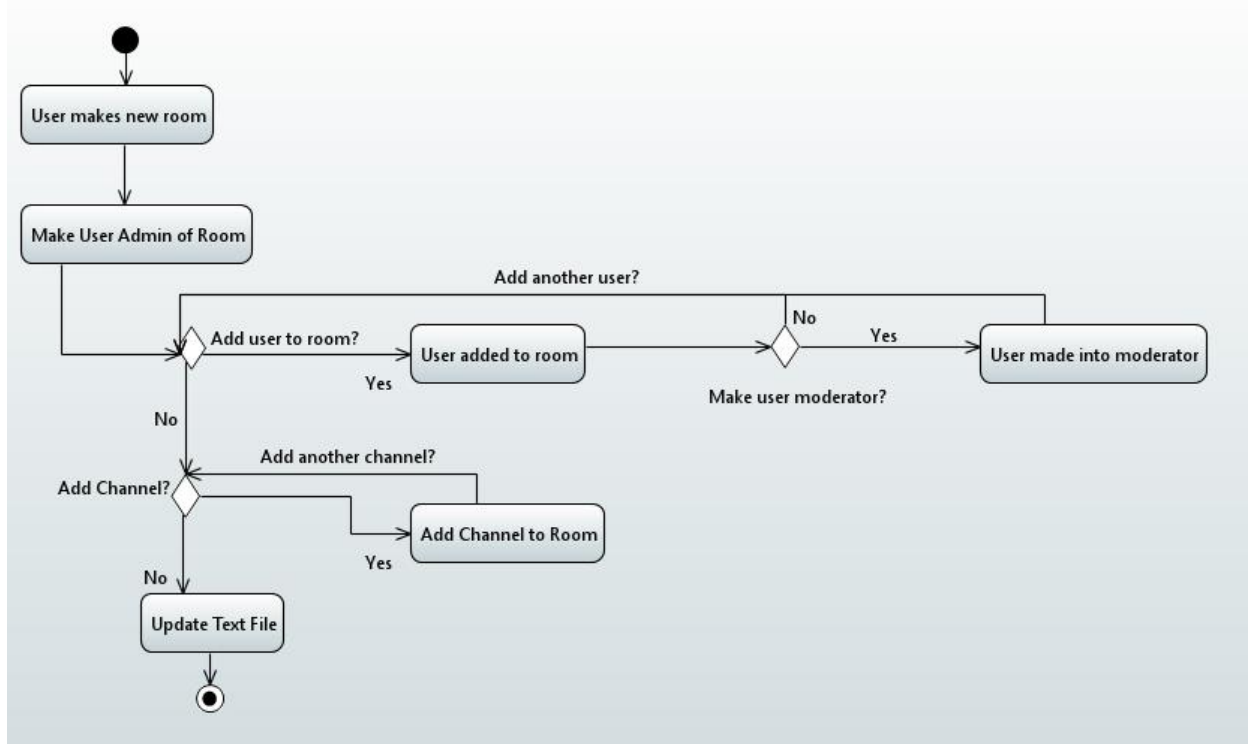


Figure 6 MAKE ROOM ACTIVITY DIAGRAM

The make room activity is initiated by the User. User is made Admin of room. Control flow moves to a decision node to ask whether to add a user to the room. If yes, the user is added to the room. Then, another decision node asks whether to make that user a moderator. If yes, a user is made a moderator. Control flow loops back to add user decision node. If no to moderator option, user control flow loops back to add a user to the room decision node. If no more users to add, the control flow moves to another decision node asking to add a channel. If yes, the channel is added. Control flow then loops back to the add channel decision node. If no, the text file is updated. Control flow moves to the activity final node.

Class Diagram:

Introduction:

The class diagram will demonstrate how data is stored and related to each other, this includes the functions and variables and how they will work together to create a unified OOP design.

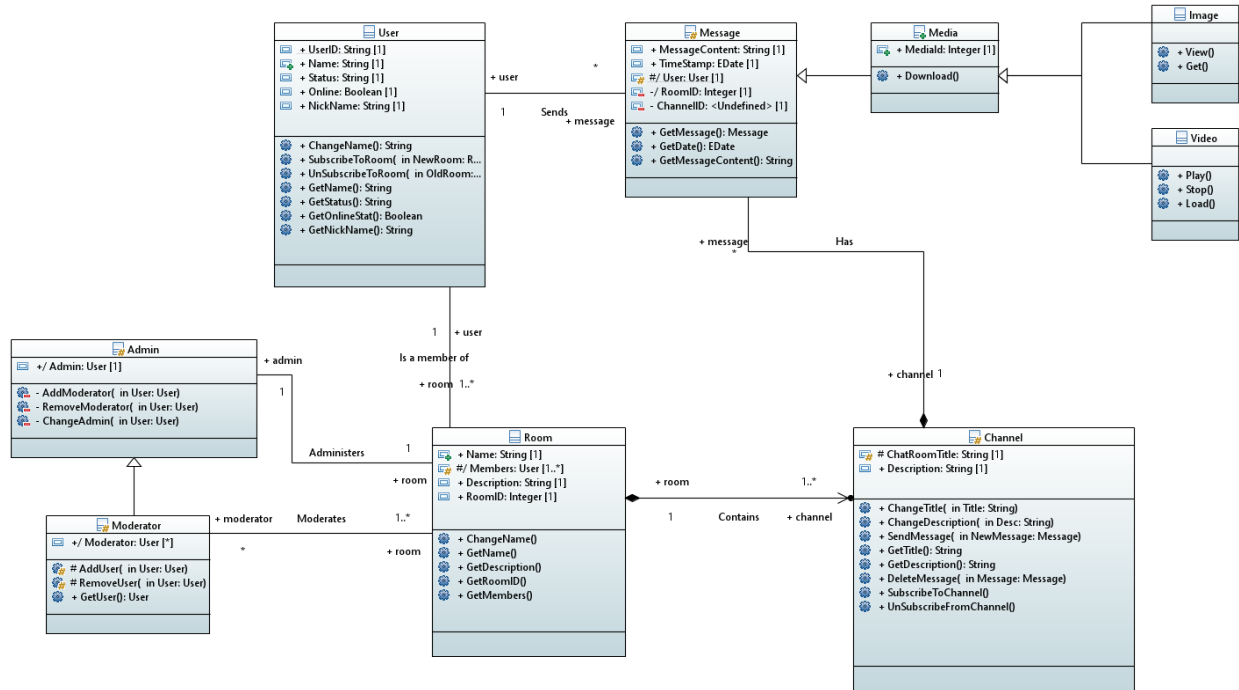


Figure 7 CLASS DIAGRAM

User

The User class contains each unique person's information, they have key functions to obtain and change the information. Note that the UserID can never be changed however the other information can.

Admin/Moderator

The Moderator class contains the functions needed to add and remove users as well as obtain the details of the moderator. The admin class is the same as the moderator, but they can add and remove Moderators as well as change who the Admin is.

Room

The room class contains channels and information about the purpose of the room and its current members.

Channel

The Channel class is the main functionality of the program it contains the functions to send messages and subscribe and unsubscribe to the channels as well as metadata about the channel such as its name.

Message

The Message class contains the text and files sent by users into channels, this class contains the time it was sent where it was sent to and the contents. They will inherit the Media class if a user chooses to send a file

Relationships:

Admin (1) Administers Room (1)

The admin class contains the functions used to administer the Room, there can be only one admin for each room.

Moderator (*) Moderates Room (1..*)

The Moderator class has all the functions to add/remove users and moderate said users there can be no moderators or many and a moderator may have several rooms under their control. The power to be a moderator is controlled by the room administrator.

USER (1) IS A MEMBER OF ROOM(1...*)

A user can be a member of none or many rooms however a room must have one member who must also be the administrator the administrator can remove others and add others but not themselves unless they decide to delete the room.

Room(1) Contains Channel(1..*)

One room must have at least one channel (the default channel is named 'General') The first channel created can change its name but it cannot be deleted. More rooms can be created but must have different names.

Message(*) Has Channel(1)

There can be an infinite number of Messages however these messages can only be assigned to one channel. E.g you can send the same message content to many channels but these must be unique classes.

User(1) Sends Message(*)

One user can create/send an infinite number of messages, however, the message sender can only be one unique use

Sequence Diagrams:

Introduction:

Figure 8 (Log in)

This diagram shows the login process. It highlights the interactions between the user, the application, and the text file.

Figure 9 (Send Message)

A user attempts to send a message, the system will first need to validate their connection to ensure that the target user is online and connected to the broker.

If the user is online:

- The message is sent by the sender
- The recipient requests new messages stored in the text file
- The text file returns a queue of all the messages
- The chat log is updated and the recipient is notified

If the user is offline:

- The message is stored within the text file temporarily.
- The user's connection is then repeatedly checked until they are confirmed to be online.
- Once the user is online, their message is then sent to the recipient, and the following stages mentioned above occur.

Figure 10 (Make Room)

The diagram shows the process that occurs when a user makes a new room. The user is assumed to be Admin as by making a room the user becomes that Room's Admin by default. Users are added to the new room, and the admin decides whether they are moderators or not. Then channels are created in the room, and finally, the text file is updated.

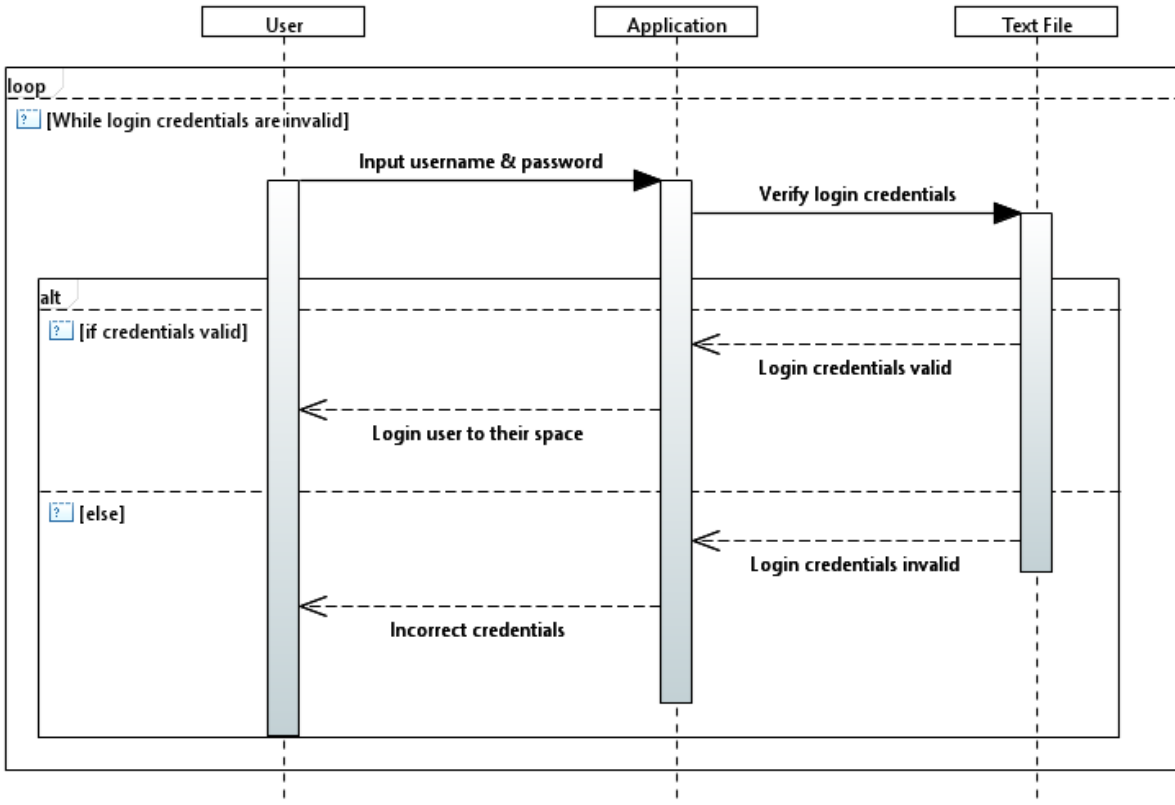


Figure 8 LOGIN SEQUENCE DIAGRAM

While the login credentials are invalid, the User lifeline sends a call message to input their username and password, which is sent to the application lifeline with a call message. The Application sends these to the text file lifeline for verification. Within an alt operator, the text file lifeline returns whether they are valid or not. The application lifeline sends a return message to the user lifeline to allow the user to log in if the credentials are valid. In the guard of the alt operator, the text file lifeline sends a return message with login credentials invalid to the application, and a return message is sent to the user lifeline to inform them of incorrect credentials. This all occurs within a loop operator, that exits when the login credentials are valid.

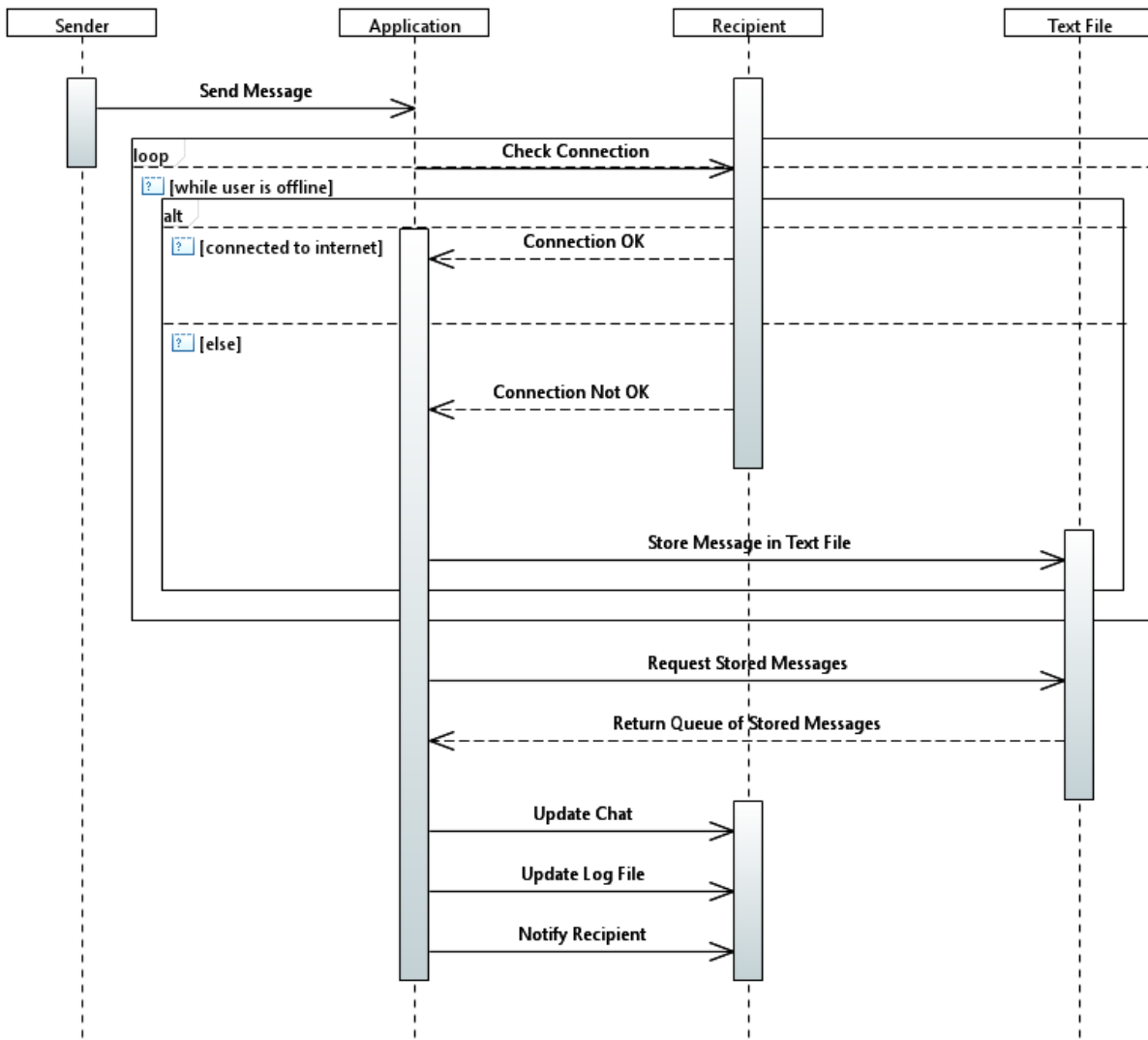


Figure 9 SEND MESSAGE SEQUENCE DIAGRAM

The sender lifeline sends a call message to send a message to the application. In a loop operator, while the user is offline, the application lifeline sends a call message to the recipient lifeline to check if they are connected.

Within an alt operator, if the user is connected a return message is sent to confirm. In the guard, a return message is sent to inform that there is not a connection, and a call message is sent from the application to the text file lifelines to store the message in the text file. After a connection has been established, the application lifeline sends a call message to the text file lifeline to request any stored messages, and a return message is sent back to return a queue of stored messages. Multiple call messages are then sent to the recipient lifeline to update the char, update the log file, and notify the recipient.

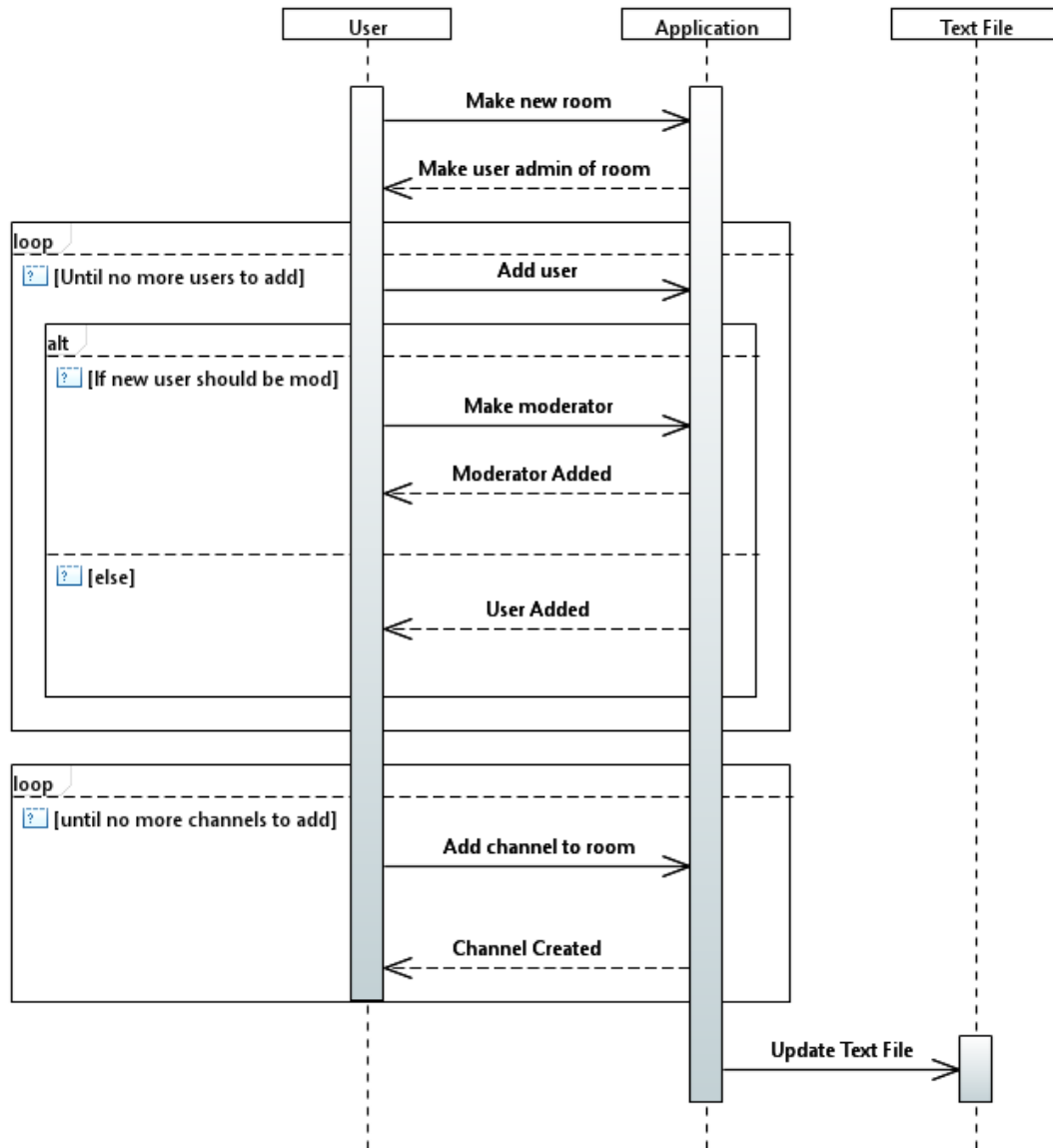


Figure 10 MAKE ROOM SEQUENCE DIAGRAM

The user lifeline sends a call message to the application lifeline to make a new room. The application sends a return message back to make the user the admin of the new room. Within a loop operator, until there are no more users to add, a call operator is sent from the user to the application lifelines to add a new user. Within an alt operator, if the new user should be a mod, a call message is sent to make the user a moderator, and a return message is sent back from the application lifeline to confirm that a moderator has been added. In the guard, a return message is sent back to confirm a user has been added. Then, in another loop operator, until there are no more channels to be added, a call message is sent from the user to the application lifeline to add a new channel to the room. A return message is sent back to confirm a channel has been created. Finally, the application lifeline sends a call message to the text file lifeline to update the text file.

Component Diagram:

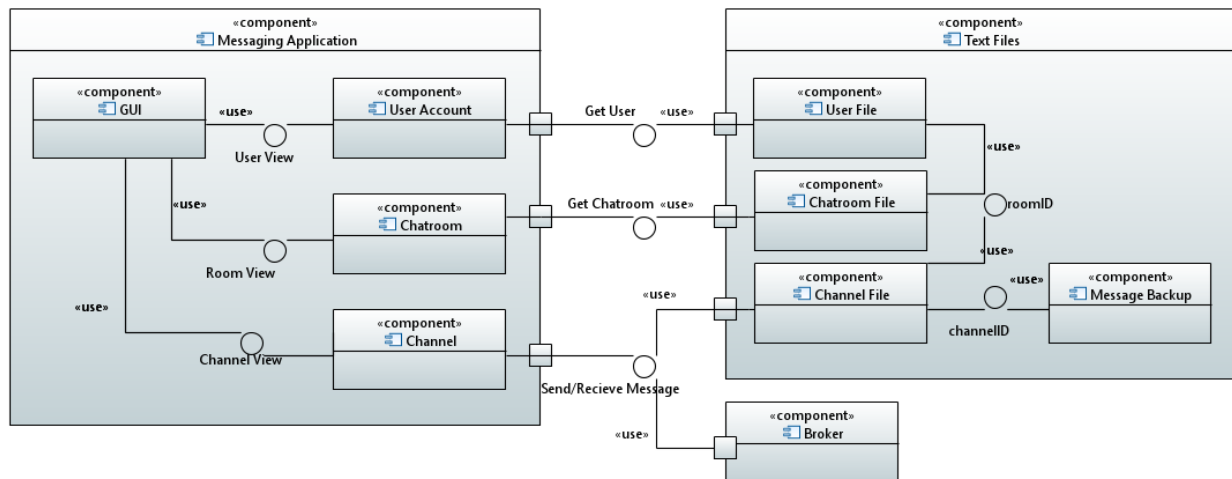


Figure 11 COMPONENT DIAGRAM

The Component Diagram shows the structure of the system's components. There are 3 main components; Messaging Application, text file and Broker. The Messaging Application contains the GUI, User Account, Chatroom and Channel components. The GUI component is linked to the user account component using the User View interface, which is provided by the User Account component. The GUI Component is also linked to the Chatroom component using the Room View interface, which is provided by User Account. Finally, the GUI also links to the Channel component using the Channel View interface.

The text file component contains the User text file, Chatroom text file, Channel text file and Message Backup components. The Chatroom text file component provides the roomID interface, which is used by the User and Channel text file components. The Channel text file also provides the channelID interface which is used by the Message Backup component. The User Account component, which is contained within Messaging Application, provides the Get User interface, which is outside Messaging Application, via a port. User text file uses Get User from within text file via a port. The Chatroom component, which is contained within Messaging Application, provides the Get Chatroom interface, which is outside Messaging Application, via a port. Chatroom text file uses Get Chatroom from within text file via a port. The Channel component provides, via a port from Messaging Application, the Send/Receive Message interface, which is used by the Broker component through a port. The interface is also used by the Channel text file component via a port into the text file.

FSM Diagrams:

Introduction:

Figure 12 (Log in):

This FSM Diagram considers the shift in states from when a user attempts to log in to when they are successful with their login process. The diagram also considers a potential issue of the text file being unavailable. Our team's solution is to immediately notify the text file admin and inform the user of the temporary unavailability. The application then ensures that the user is unavailable to log in instead of allowing them to take advantage of a temporary system vulnerability.

Figure 13 and Figure 14 (Publisher & Subscriber):

Our team decided to split this FSM diagram into two parts that run concurrently as we believe it is important to consider both the perspectives of a publisher and subscriber when it comes to message exchanges. From the publisher's perspective, once a message is created it begins to process of transmitting this data to the necessary topics. The subscriber then picks up from this point by having the broker distribute the message and its data to the relevant topic subscribers. The application then handles the process of displaying the message to the target recipient by modifying the user interface to reflect the updates made.

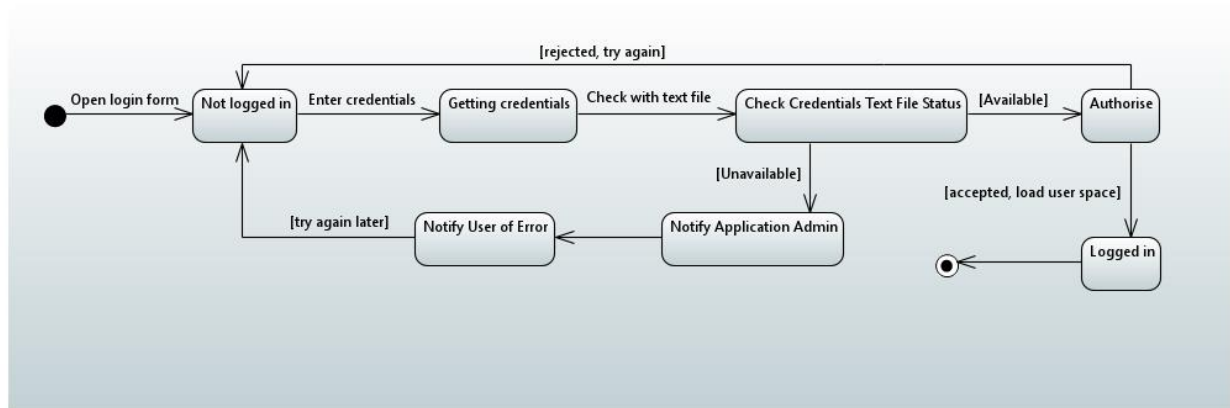


Figure 12 LOGIN FSM DIAGRAM

The login state begins with the transition to the login form on the application. The current state is that the user is not logged in. This transitions to getting credentials state where the user enters their credentials. The next transition is to the check credentials text file status. If the text file is unavailable, it transitions to notifying the admin and the user of the error and unavailability. It then loops back to the not logged in state. If the text file is available, it transitions to the authorise state where the credentials are checked against the text file. If rejected, it transitions back to the not logged in state and the user must try again. If accepted, it transitions to a logged-in state and loads the user's space before ending with the final state.

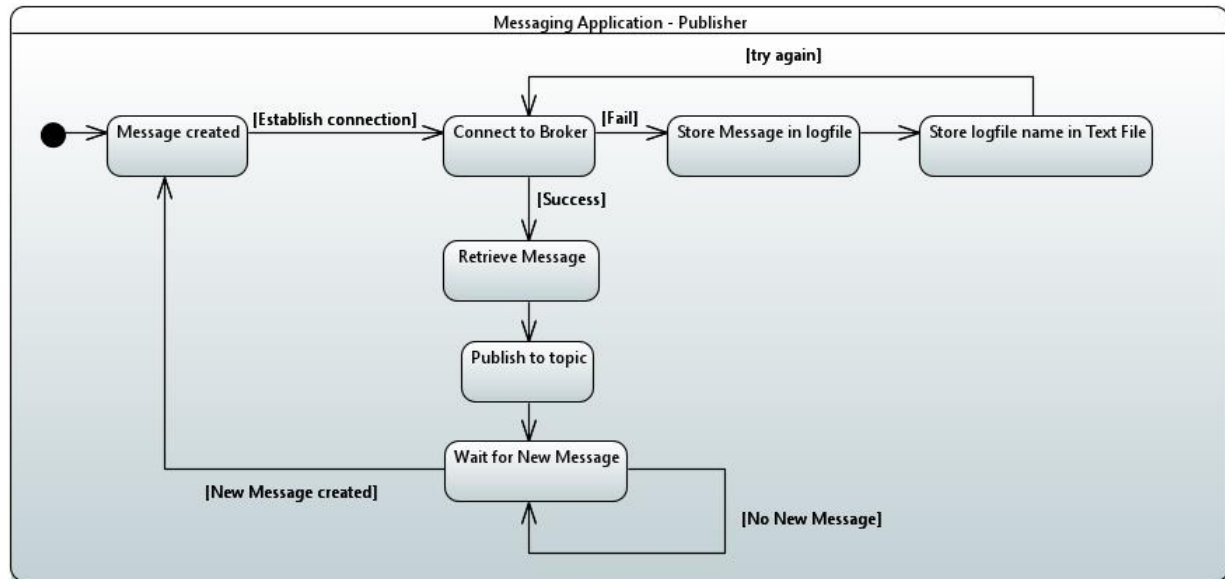


Figure 13 PUBLISHER FSM DIAGRAM

The publisher FSM diagram begins with the message being created. This then transitions to the connect to broker state to establish a connection between the publisher and broker. If it fails, it transitions to the store message in logfile state and then to the store logfile name in text file state to ensure the message contents are not lost while waiting on a successful connection. This then loops back to the connect to broker state to try again. If it is a success, it transitions to the retrieve message state where all stored message contents are retrieved before moving to the publish to topic state. Message contents are published to the relevant topic using the broker before transitioning to the wait for new message state. It then waits till a message is received to begin the process of looping back through the states again.

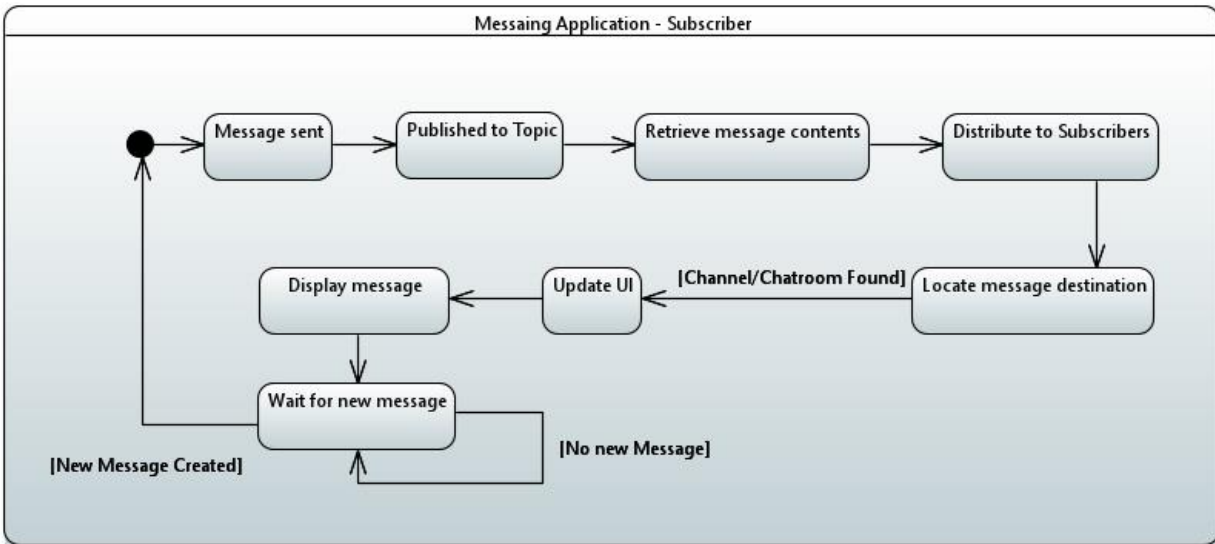


Figure 14 SUBSCRIBER FSM DIAGRAM

The diagram begins with the message being sent (assuming the concurrent states on the Publisher's side have already occurred). This state then transitions to the state where the message and its data are published to the specific topic. This then transitions to the retrieve message contents state. Once the necessary data is acquired, it transitions to the distribute to topic subscriber's state. The locate message destination state is then triggered to identify the target recipient of the message. Once identified, the transition is made to the update user interface state and then to the display message state so that the target recipient can view the message they have been sent. Once the message is displayed on the recipient's side, the transition moves to the wait for new message state. It then waits till a message is sent via the publisher and received to begin the process of looping back through the states again.

Communication Diagram:

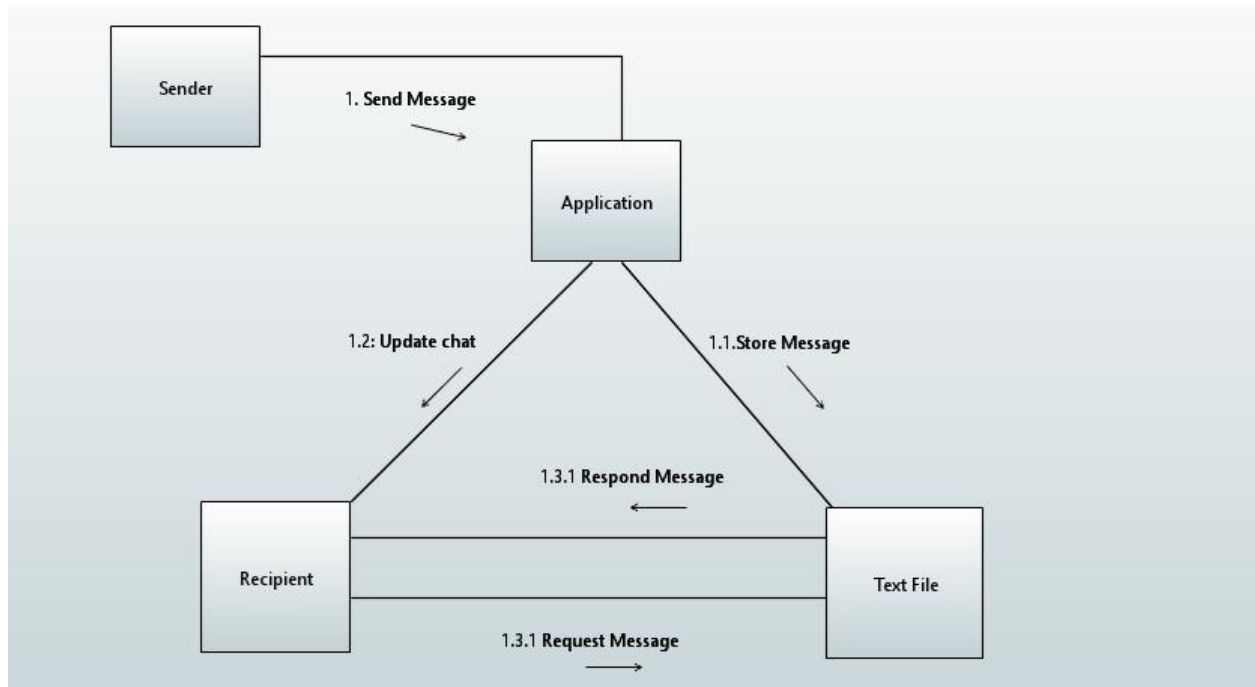


Figure 15 SEND MESSAGE COMMUNICATION DIAGRAM

The communication diagram demonstrates how the system will send and receive messages. First, the sender will send a message to the application server which will then store that information in the text file. Then the message will be sent to update the chat on the recipient device. If the recipient is not online, then they will request the information from the text file which will respond with the same message which was sent by the sender.

Deployment Diagram:

The Deployment diagram demonstrates how the system will be implemented in the real world. The Client PC will contain the application which will only directly communicate with the server via the MQTT client. The application will depend on the MQTT client to function and send messages to the server and receive messages.

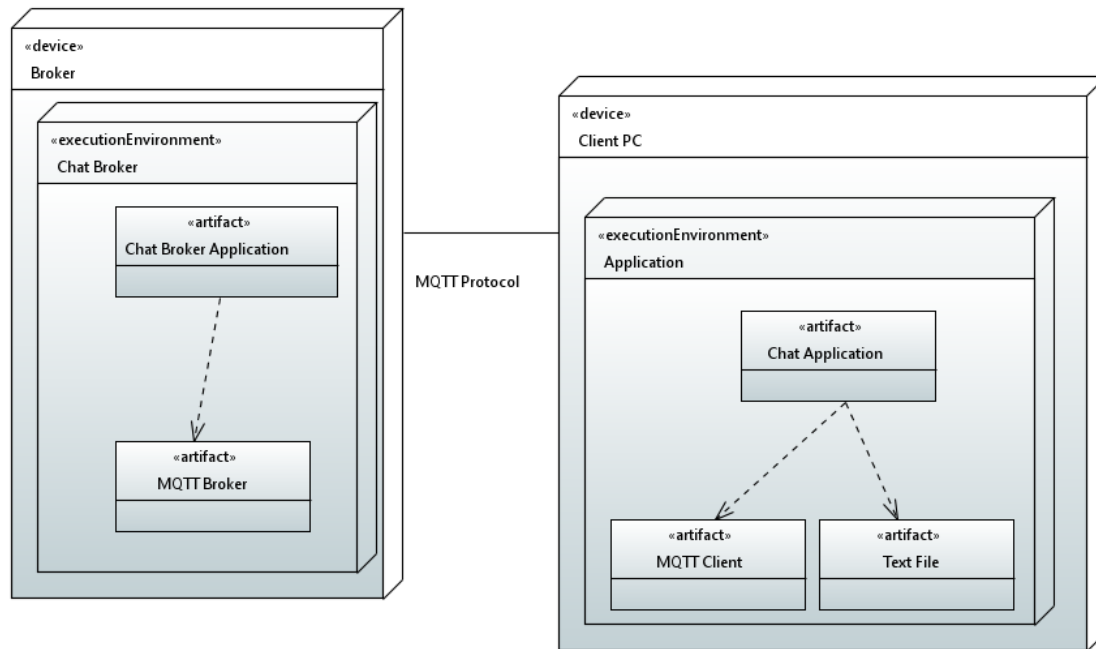


Figure 16 DEPLOYMENT DIAGRAM

The Deployment diagram demonstrates how the system will be implemented in the real world. The Client PC will contain the application which will only directly communicate with the server via the MQTT client. The application will depend on the MQTT client to function and send messages to the server and receive messages.

The Server will have 2 separate executables the Chat server will depend on the MQTT broker to send and receive messages while communicating with the client devices. The Chat server will use a text file to store information about users and their messages on the client PC.

GUI Mock-Up:

Introduction:

The GUI mock-up outlines the essentials of the application's basic functionality and user interface. It displays how various elements of the GUI would be laid out in the final version of the app and provides an idea of how the application may function prior to development.



Figure 17 GUI MOCK-UP

This GUI Mock-up shows how the basic user interface of the application should look. Inspiration was taken from research into other popular messaging applications, such as Slack and Discord.

- (1) All the users' available chatrooms are shown on the leftmost column on the screen.
- (2) All the available text channels for the currently selected server are displayed next to that, the channels can be named and grouped.
- (3) At the bottom of that column, the user can access their profile and application settings, as well as change their status.
- (4) The main, middle section of the screen displays the contents of the currently selected text channel. The user can send via the input bar at the bottom, and messages that are sent and received are displayed in the main area. The user can scroll through messages using the scroll bar and can search through message history using the search function at the top.
- (5) On the rightmost column, all the users in the currently selected chatroom are displayed, grouped by their status.

Final GUI Design:

Labelled below is the final design for the GUI of our application post development.

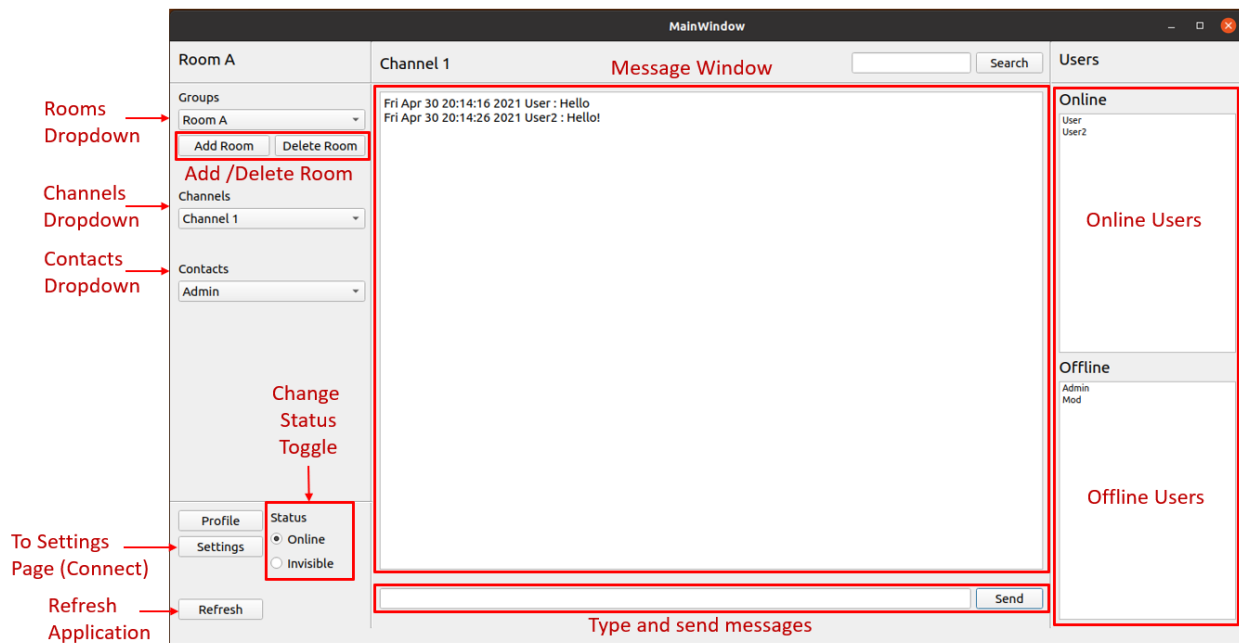


Figure 18 LABELLED MAIN PAGE

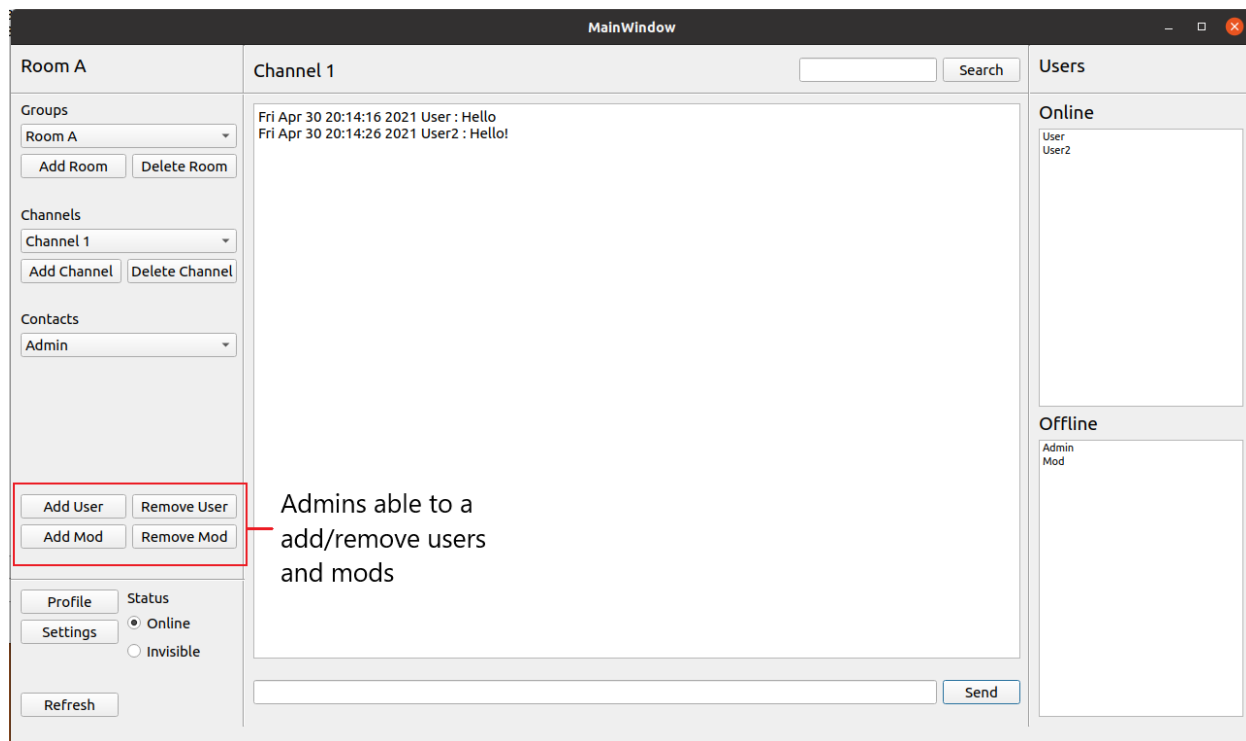


Figure 19 LABELLED ADMIN VIEW

Design Pattern:

We have looked at 3 main options for the design patterns Builder, Factory Method and Abstract Factory.

Pros	Cons
Factory Method	
<ul style="list-style-type: none">- Implementation and creation are separate- Object creation can be kept in one place- New types can be added without breaking existing code	<ul style="list-style-type: none">- Subclasses may be needed to implement the pattern- The design pattern works well in particular circumstances but can be very complex if not appropriate.
Abstract Factory Method	
<ul style="list-style-type: none">- New variants of objects can be added without modifying existing code- Object creation can be kept in one place	<ul style="list-style-type: none">- Makes code more complex as lots of interfaces and classes are introduced when using the pattern.- We are unlikely to add new functionality to existing code.
Builder Method	
<ul style="list-style-type: none">- Objects can be made step-by-step depending on the required parts- The components can be reused.	<ul style="list-style-type: none">- Code can become much more complex as each new pattern will require multiple classes.- The builder is not necessary for the messages as they commonly inherit one type like a message that contains a file or a user who is also an administrator.

Table 4 DESIGN PATTERN ANALYSIS

As a result of the analysis above of our potential options, our team decided on the **Factory Method** as our design pattern (Refactoring guru, 2021). We intend to use creator classes to declare the new objects such as the User or a Message wherein each object is unique but has the same functions. Moreover, different users can inherit different functions such as a user who has the admin role has more functionality such as adding or removing other users.

Planned Architecture:

When it came to finalising decisions on the architecture and implementation of our program we made use of our extensive design process to determine the most appropriate choice as well as considering each individual's skills to determine if any option we consider is practical. Moreover, our risk assessment also helped guide our decision-making process.

Using a text file for storing our data was a decision made after considering the need for transactional consistency and data validation, we can have reasonable confidence in the data provided to each client from the broker thanks to the MQTT protocol. We needed to consider the practicality of implementing both options, as only two group members had extensive experience with MYSQL we felt it would be too complex to implement while allowing all group members to contribute equally and be able to understand and implement high-quality code. We also considered that the application would be difficult to implement in the real world when using a database as we would have to include a database on each client. Given the nature of the application, we felt it appropriate to keep it simple as to keep the application small and accessible to potential users.

We felt that the structure of chatrooms and channels over a simple list of chatrooms with the users makes the program much more practical in the real world. As we can have Rooms for groups of users such as a single department and then channel for related groups of people such as a channel for a particular project within the business. This approach meant that we could have administrators and moderators for individual rooms rather than for every channel which made it easy for users to be added as needed to rooms rather than each channel individually.

Code Review

Throughout the program, we made use of a variety of data structures for storing variables in an organized manner. Most prominently, vectors were integral to the program as we used them to store all data relating to the Chatrooms and Channels, and the Users, Moderators, and Admins. We decided to use vectors over arrays as vectors are dynamically allocated, whereas arrays would need a predetermined size. Vectors were also integral to the idea of storing all the applications' data in text files, so the state of the application could be saved. In addition to this, we made use of Structs to organize our constant variables. This greatly helped with the programming process and the readability of the code and ensured data has not been repeated unnecessarily.

We made some use of searching algorithms in our program. Most notably, we encountered many instances where we would need to split a string by a delimiter, such as when reading data stored in a text file. To do this, we employed an algorithm included in the boost library to perform a linear search of the string and remove any instances of a given delimiter, storing the split strings in a vector. Additionally, we made use of "for loops" regularly to perform linear searches through vectors to find a specific piece of data, for example, to remove a specific user from a chatroom.

For the entire duration of the project, we made use of Github as a version control system to easily work collaboratively. The use of git has allowed us to work on different parts of the project simultaneously and keep an extensive and thorough history of changes to the project. This has meant we have easily been able to check if we are aligned with the timescale of the project or revert to earlier versions in the event of something going wrong. It has also meant that an up-to-date version of the project has been stored locally on all our machines, as well as a master version stored on the cloud, meaning it would be easy to recover from data loss.

Finally, we have made extensive use of the concepts of Pair Programming throughout the implementation of the project. This has involved one person at a machine, implementing a feature, while one or more of the other members think about the logic, guide the programming and review the code being written, with the person doing the programming alternating every so often. This has greatly assisted us in achieving high quality, efficient code with minimal bugs or errors, and has been a useful tool in implementing the larger, more intensive features.

Test Plans:

The following table is an overview of the results of the acceptance testing done. The complete tests can be found in the Test Report and a video of the tests being performed can also be seen [here](#). The complete plan for the tests covering all requirements can be found in the Test Plan document. The tests done is acceptance testing as it is based on the requirements and the type of testing is grey box as while there is some understanding of the inner workings of the code for white box testing, the understanding is not ubiquitous, and a major proportion of the tests are black box. As seen in the table, all the tests passed, however, a few bugs were uncovered in the testing process. Specifically, when a user closes the application while online, other users will still see them as online despite being offline. When a user goes offline in an active chatroom or is removed, they can still see the messages until they refresh the application and a problem with the contacts drop-down related to incorrect logins.

Test ID	Description	Passed?	Comments
01	Users must be able to send, receive and view messages through the application.	Passed	Both users able to send, receive and view messages without issue.
02	Users must be able to create chat rooms (rooms with more than two contacts).	Passed	All users added successfully and able to send and receive messages within the channel.
03	The user that creates the chat room must be classified as Admin.	Passed	Able to check if admin by viewing permissions.
04	The moderator must be able to invite and remove users from a chatroom.	Passed	Passed but still has a bug where user can still see old room until refresh.
05	Moderators must inherit all the admin permissions; however, Moderators cannot demote the Admin.	Passed	Moderators can Add and Remove users and channels, they are not able to promote or demote users however.
06	Application must provide a friendly User Interface (UI)	Passed	Positive user feedback overall; average of 73.5% positive. Lowest feedback score on whether users would recommend the app.
07	Users must be able to see the active users in the chat room.	Passed	Passed except that when the user goes offline by closing the application they are viewed as still being online.
08	Users must be notified when a new notification is received.	Passed	Notifications received for messages and most error events.
09	Clients must not connect directly to other clients without a server or a broker.	Passed	Passed in that you cannot send messages while offline.
10	A server or a broker must allow multiple authorized clients to connect to it.	Passed	Multiple clients successfully connected.
11	Users must only access their space after the login.	Passed	Only able to sign in when both username and password are correct.

12	Passwords must be saved securely locally.	Passed	User credentials text file has encrypted passwords, failed to decrypt using rainbow table.
14	Application must list all the personal contacts in the contacts pane.	Passed	Test passed but bug found in Test 11 where users duplicate in table.
16	The moderator should be able to create and delete channels in the chatroom.	Passed	Admin / Moderator able to add and delete rooms.
18	Users should be able to change their status.	Passed	User status when connected can be changed and is visible to other users.
19	Messages should be sent and received within 5-10 seconds.	Passed	Almost immediate delivery of messages, hard to test the exact time.
20	Users should be logged off automatically after a specific amount of time.	Passed	Test passed both when timer ran out without interaction and whether it reset on interaction with the reset button. Timer usually set to 15 minutes but set to 10 seconds for test.

Table 5 TESTING OVERVIEW TABLE

Conclusions & Future Work:

In conclusion, the team believes that we were able to successfully implement all the primary requirements outlined in the requirements list as our 'must' and 'should' have for the application. The entire process of moving from our initial planning stages to a final product has been a vital learning experience of the key skills required for one to be successful in software design and implementation.

The application the team has developed can carry out the basic functionality of a messages exchange platform in addition to providing additional features such as password encryption, account creation and displayable users list. The application is designed to be easily scalable and maintainable as the code was designed per the rules outlined in our coding style and contribution guides. The application is extremely reliable and computationally efficient. However, given more time our team would be like to refactor other aspects of the code using external libraries to improve the efficiency of the entire application. Within its current capacity, the application is capable of being used for communications between multiple instances on one single device. Ideally, within a real-world setting for portability, the application is modified to reflect more intricate use of networking. This will allow the application to support communication across multiple platforms/devices.

Moreover, the team would like to further develop the application in the future by integrating all our 'could' have requirements outlined in the requirements list above. These consist of both functional and non-functional requirements that would positively impact users of the application. One area of focus for these features is accessibility. As software designers and developers, it is our duty to ensure that our applications are accessible to all to the best of our ability. Hence, a user being able to modify text sizes or change the colour scheme of the application would be an accessibility feature for those with visual impairments for example.

Alternatively, another area for development would be the user interface so that it may contain features that are popular amongst other message exchange platforms. Some of these features include the ability to use emojis, display and change profile pictures and the ability to know whether a message has been read or not. Though they are not fundamentally necessary to carry out basic message exchange functionality, they are however application features that can improve user experience.

Finally, another area the team would like to modify in the future regarding the application would be the use of cloud-based databases to store user data instead of offline text files. Despite our use of encryption to protect user credential data, users are still able to access another user's data by simply navigating to the specific text files and decrypting them. A cloud-hosted database would increase the level of security of the application and thus make the application more favourable to users looking to maintain their data privacy.

References

Boost.org, 2021. *Boost*. [Online] Available at: boost.org
[Accessed 11 April 2021].

Dai, W., 2021. *Crypto++*. [Online] Available at: <https://www.cryptopp.com/>
[Accessed 11 April 2021].

Discord, 2021. *Discord*. [Online] Available at: <https://discord.com/>
[Accessed 29 April 2021].

Refactoring guru, 2021. *Factory method*. [Online] Available at: <https://refactoring.guru/design-patterns/factory-method>
[Accessed 8 April 2021].

Slack, 2021. *Slack*. [Online] Available at: <https://slack.com/intl/en-gb/>
[Accessed 29 April 2021].

The Qt Company, 2020. *Qt*. [Online] Available at: <https://doc.qt.io/>
[Accessed 11 April 2021].

The Qt Company, 2020. *QtMQTT*. [Online] Available at: <https://doc.qt.io/QtMQTT/index.html>
[Accessed 11 April 2021].

Individual Contributions:

When it came to the division of tasks within the team, we elected to primarily focus on the tasks assigned to each of our roles. During the team's preparation for the first deliverable, our Project Manager, Hannah oversaw the entire process of preparing the requirements list, Gantt chart and risks and mitigation plan. The rest of the team supported her during this process through the equal division of work amongst each other. Hassaan and Jarad assisted with the formulation of the Risks and Mitigation plan while Nick helped design the initial template of the Gantt Chart.

Subsequently, when the second deliverable was due and the team had to design diagrams to reflect our application's design, our Software Architect, Nick took on the role of reviewing each piece of work to ensure that it was up to the team's standards. Hannah assigned all team members with a set of diagrams to each to help even the workload for the deliverable.

During preparation for the third deliverable, our Software Developer, Hassaan oversaw the integration of some of our 'must' have requirements into the application in addition to preparing the Doxygen reference manual. Hannah took on the task of formulating the contribution guide while Nick and Jarad worked together to create the coding standards and style guide.

Moving on to our fourth deliverable, Jarad handled the design and creation of the project's test plans before moving onto filling out the test report. Hassaan and Hannah both worked together towards the completion of all the project's 'must' have requirements and a few of the 'should' have requirements as well. Nick assisted the team by refactoring parts of the code to reflect our standards outlined in the style guide.

Finally, before the submission of the final deliverable, the team worked together to complete all requirements of the project submission. Hannah assigned each member with role-specific tasks. Hassaan continued the final stages of the application's development. Jarad developed unit tests and then reviewed and finalised the project's test report. Nick supported Hannah with the formulation of the report. In addition to these role-specific tasks, team members would have frequent meeting calls with each other to assist in the completion of each other's tasks.

Overall Reflection:

A key feature of this module was the experience of communicating and working within a team. As friends, we decided that this would be a great opportunity to further build upon our ability to work on projects together. Completing a project of this scale during a pandemic has been tough, however, due to us already having an established communications channel through Discord (Discord, 2021), we were able to stay up to date with the progress each of us was making on this project.

The delegation of group work allowed for us to ensure that all group members received an equal workload as to not burden a single member with several tasks. The team also ensured that we communicated any difficulties we were having with a task early on so that any other available team member could set up a group call and we could resolve the issue together.

The team faced a few challenges in the initial process of getting papyrus to work alongside Eclipse for the creation of our diagrams. However, through working on-call together and by researching solutions in our own time, we were able to surpass the hurdle and generate high-quality diagrams for our documentation portion of this project.

Moreover, pair-programming played a significant role in how the team approached our development process. At times when our software developer would get stuck on a problem, they would pass the challenge over to another member so that we could make use of alternative perspectives and experiences as a means of solving a challenge. This method worked beyond our expectations and is the primary reason for our team successfully implementing all our 'must' and 'should' have requirements on time.

In summary, this module and its project have helped the team strengthen our capabilities of working together on a project of this scale in addition to equipping us with the skill set necessary for success.

Appendix:

Risk Analysis Scale:

Probability	Impact	Description
5	5	A risk event that if it were to occur, will have a serious impact on the project achieving its desired result. To the extent that one or more stated outcome objectives will not be achieved.
4	4	A risk event that if it were to occur, will have a significant impact on the project achieving its desired result. To the extent that one or more stated outcome objectives will fall below acceptable levels.
3	3	A risk event that if it were to occur, will have a moderate impact on the project achieving its desired result. To the extent that one or more stated outcome objectives will fall below goals but above minimum acceptable levels.
2	2	A risk event that if it were to occur, will have a minor impact on the project achieving its desired result. To the extent that one or more stated outcome objectives will fall below goals but well above minimum acceptable levels.
1	1	A risk event that if it were to occur, will have a minimal impact or no impact on achieving outcome objectives.

Table 6 PROBABILITY AND IMPACT WEIGHTING

User Manual:

Start-up:

When first starting the application, you will see the login page.

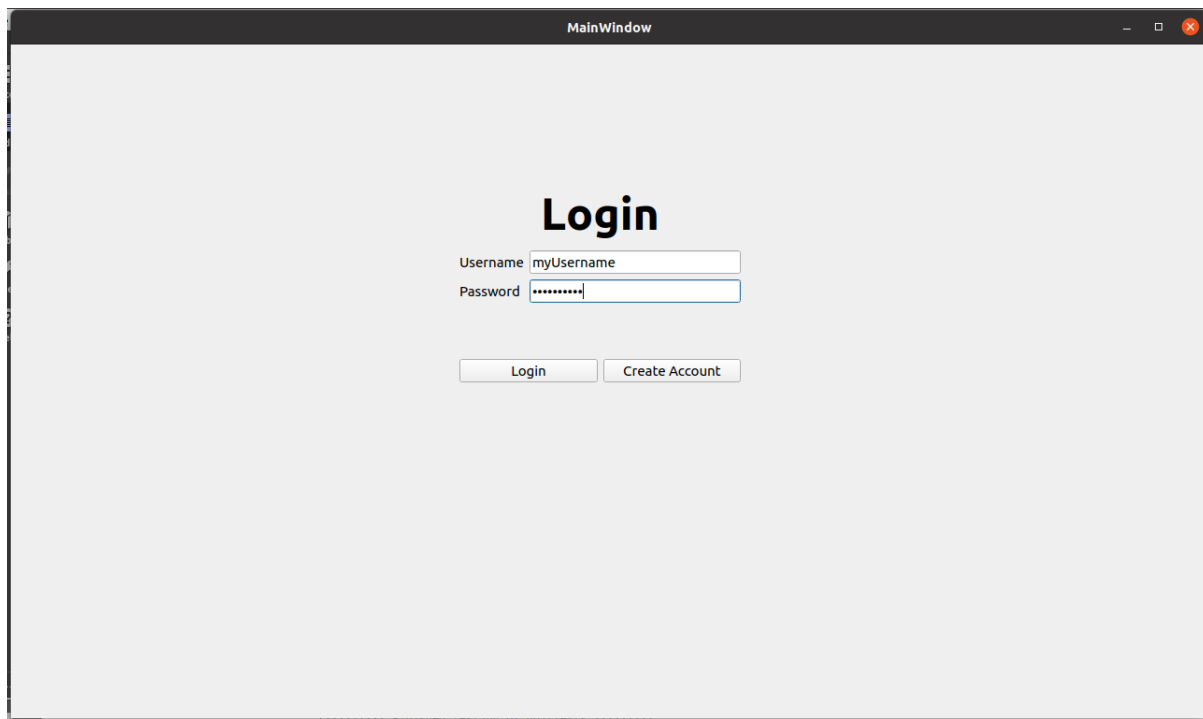


Figure 20 LOGIN PAGE

Login:

The login page is first shown when logging in. At the centre of the screen is two textboxes; one to enter the username and one to enter the password. Below that is two buttons. The leftmost button labelled "Login" will allow you to log in if you have entered the correct username and password combination, if not, an error message should appear. The rightmost button labelled "Create Account" takes you to the Account Creation page.

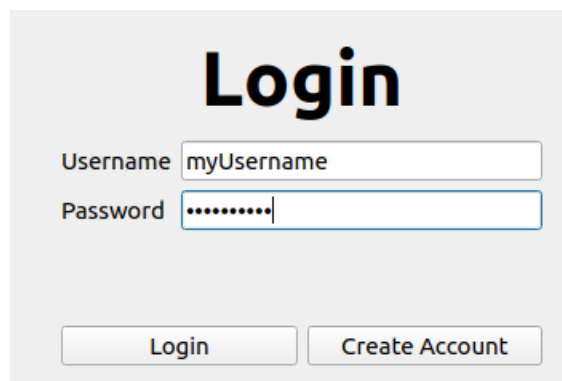


Figure 21 EXAMPLE OF USER INPUTTING CREDENTIALS TO LOGIN

Account Creation:

On the account creation window, enter the username and password for the account you want to create, and click the button labelled “Sign Up” to create the count. After clicking Sign Up, you will automatically be returned to the login page where you can log into your new account. If you change your mind about creating a new account, you can click the button labelled “Back” to return to the login page.

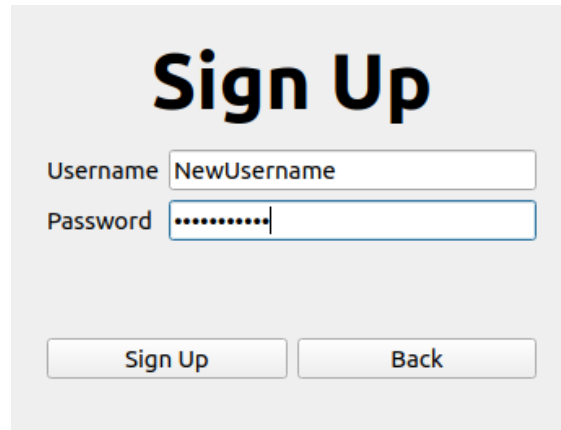
A screenshot of a 'Sign Up' form. The title 'Sign Up' is at the top in a large, bold, black font. Below the title are two input fields: 'Username' with the text 'NewUsername' and 'Password' with a masked password '.....'. At the bottom are two buttons: 'Sign Up' and 'Back'.

Figure 22 EXAMPLE OF USER CREATING AN ACCOUNT

Setting up your Connection:

Once logged in, you will need to connect to the Server to send and receive messages. To do so, select the “Settings” button at the bottom left of the application main page. This will take you to the Application Settings window. The Host will automatically set to “127.0.0.1” and the Port will be set “1883”. This can be set to a custom server or the default can be used. To connect, click the “Connect” button. To return to the main page, click the “Back” button at the bottom left of the window.

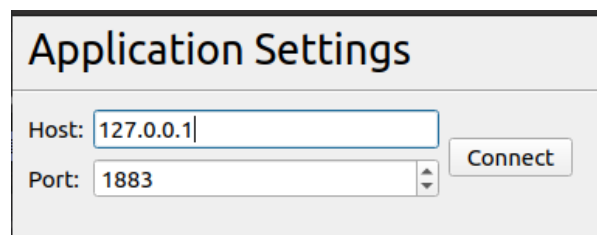
A screenshot of an 'Application Settings' window. The title 'Application Settings' is at the top in a large, bold, black font. Below the title are two input fields: 'Host' with the text '127.0.0.1' and 'Port' with the text '1883'. To the right of the 'Port' field is a 'Connect' button.

Figure 23 EXAMPLE OF USER ESTABLISHING THEIR CONNECTION

Navigation:

1. Rooms – You can select the room you want from the dropdown, create a new room, or delete the selected room.
2. Channels – You can select a channel in the currently selected room. If you are an admin or moderator of the room, you can also add a channel to the room selected or delete the currently selected channel.
3. Contacts – The contacts dropdown shows all your contacts.
4. Room Settings – The room settings are only visible as Admin or Moderator of the room; they allow you to add/remove users from the room and add/remove moderators from the room.
5. Settings – The settings button takes you to the Application Settings window, which lets you connect/disconnect from the server.
6. Refresh button – The refresh button allows you to refresh the application, this may be helpful if you have been added to a room and it is not yet appearing on your window.
7. Status – You can toggle your online/offline visibility, it also refreshes your online/offline users (9).
8. Message Window – the message window is where all messages sent to the selected channel appears. Type your message in the text window at the bottom and click the Send button to send them.
9. Users – Shows which of the users are online in the box labelled “Online”, shows all the offline users in the box labelled “Offline”.

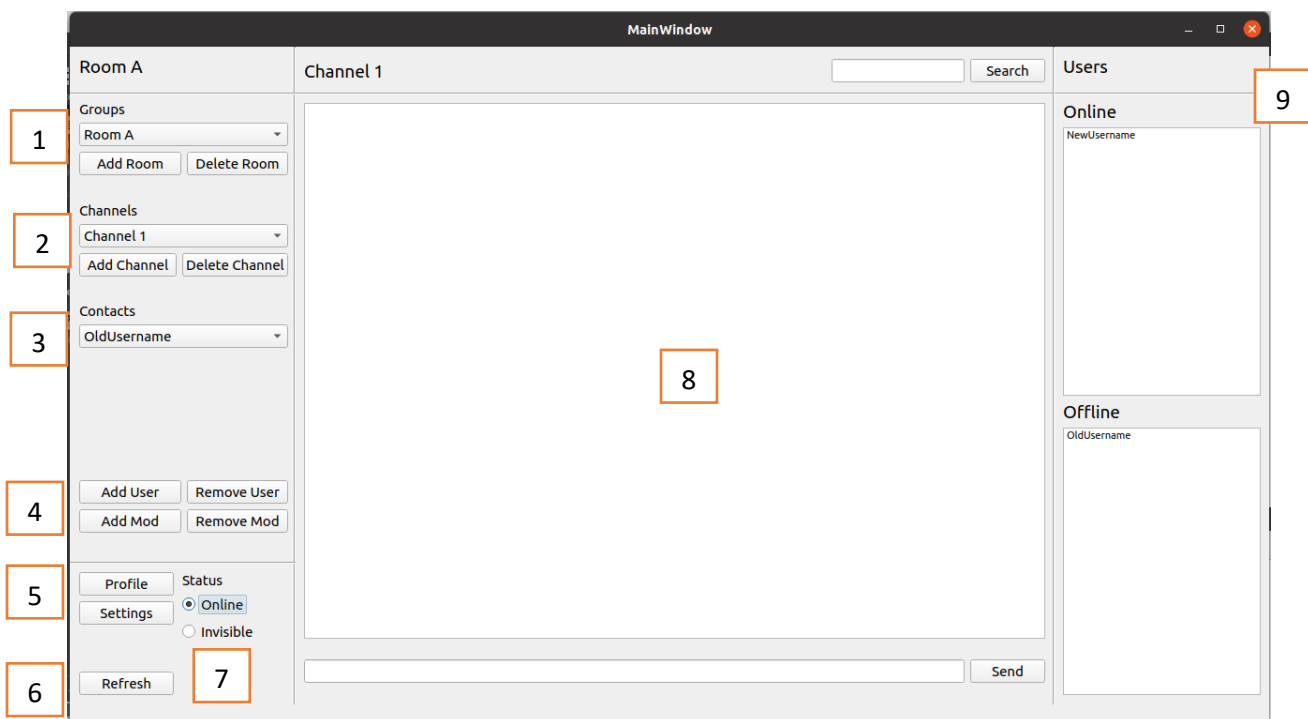


Figure 24 LABELLED GUI

Create Room

To create a room, under the Groups at the top left of the main window and select the “Add Room” button on the left. A pop-up dialogue will come up where you can enter the name of the room you want to create. Click OK to create the room or Cancel to cancel the operation. Once the room is created, you will automatically be Admin of the room and have all Admin privileges.

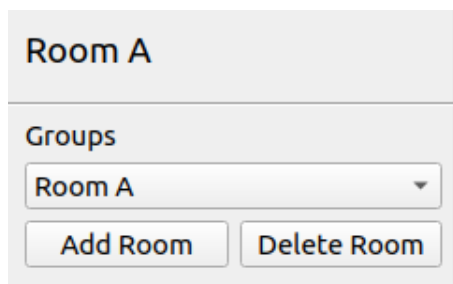


Figure 26 GROUPS DROPDOWN MENU

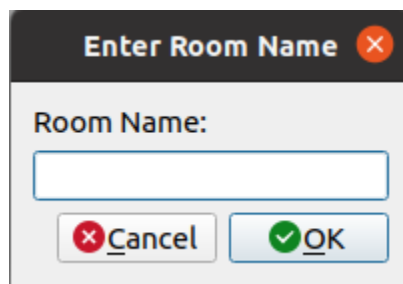


Figure 25 ROOM CREATION POP-UP

Create Channel

To create a channel, first, make sure you have a room selected that you are Admin/Moderator of. Then you can click the Add Channel button under the channels section on the left of the window. A pop-up dialogue will come up where you can enter the name of the channel you want to create. Click OK to create the channel or Cancel to cancel the operation.

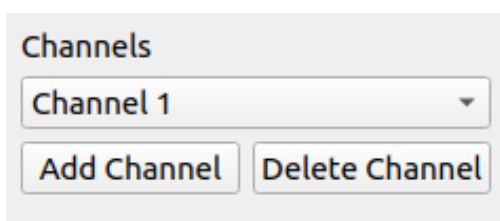


Figure 27 CHANNELS DROPDOWN MENU

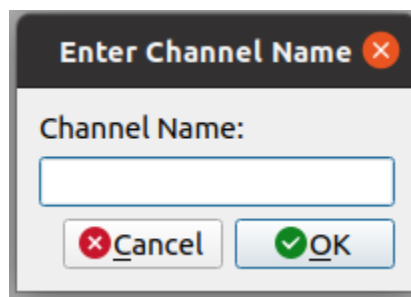


Figure 28 CHANNEL CREATION POP-UP

Send Message

To send a message, first make sure you are connected to the server and have a room and channel selected. Then type the message in the dialogue box at the bottom-centre of the main window and click the send button. A notification will come up for all the users in the channel when sent, and the message will appear in the main window.



Figure 29 MESSAGE BAR

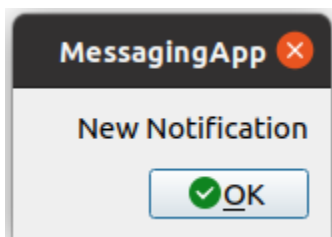


Figure 30 NOTIFICATION POP-UP

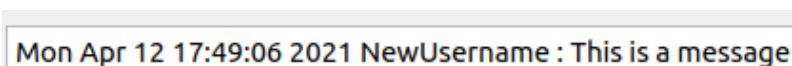


Figure 31 MESSAGE BEING DISPLAYED