|  |
| --- |
| Software Design & Implementation |
| Project Design |
| Group 30 |

|  |
| --- |
| Hannah Ashna Jacob (N0865554)  Hassaan Naveed (N0898071)  Jarad Johnson-Bailey (N0853071)  Nicholas McCaig (N0787115) |

## Contents

[Contents 1](#_Toc63517498)

[Figures 2](#_Toc63517499)

[Introduction 2](#_Toc63517500)

[1. Use Case Diagrams: 3](#_Toc63517501)

[Introduction: 3](#_Toc63517502)

[Figure 1.1 (Login) 3](#_Toc63517503)

[Figure 1.2 (Send Message) 3](#_Toc63517504)

[Figure 1.3 (Make Room) 3](#_Toc63517505)

[2. Activity Diagrams: 7](#_Toc63517506)

[Introduction: 7](#_Toc63517507)

[Figure 2.1 (Login) 7](#_Toc63517508)

[Figure 2.2 (Send Message) 7](#_Toc63517509)

[Figure 2.3 (Make Room) 7](#_Toc63517510)

[3. Class Diagram 11](#_Toc63517511)

[Figure 3.1: Class Diagram 12](#_Toc63517512)

[4. Sequence Diagrams 13](#_Toc63517513)

[Introduction: 13](#_Toc63517514)

[Figure 4.1 (Login) 13](#_Toc63517515)

[Figure 4.2 (Send Message) 13](#_Toc63517516)

[Figure 4.3 (Make Room) 13](#_Toc63517517)

[5. Component Diagram 17](#_Toc63517518)

[6. FSM Diagrams 18](#_Toc63517519)

[Introduction: 18](#_Toc63517520)

[Figure 6.1 (Login): 18](#_Toc63517521)

[Figure 6.2 and Figure 6.3 (Publisher & Subscriber): 18](#_Toc63517522)

[7. Communication Diagram 22](#_Toc63517523)

[8. Deployment Diagrams 23](#_Toc63517524)

[9. GUI Mock-up 24](#_Toc63517525)

[Figure 9.1 (GUI Mock-up) 24](#_Toc63517526)

[Libraries 26](#_Toc63517527)

[References 27](#_Toc63517528)

## Figures

[Figure 1.1 Login Use Case 4](file:///C:\Users\Nicholas%20McCaig\Documents\workspace\SDI\SDI-Project\Documentation\Software%20Design%20-%20Project%20Design.docx#_Toc63517529)

[Figure 1.2 Send Message Use Case 5](file:///C:\Users\Nicholas%20McCaig\Documents\workspace\SDI\SDI-Project\Documentation\Software%20Design%20-%20Project%20Design.docx#_Toc63517530)

[Figure 1.3 Make Room Use Case 6](file:///C:\Users\Nicholas%20McCaig\Documents\workspace\SDI\SDI-Project\Documentation\Software%20Design%20-%20Project%20Design.docx#_Toc63517531)

[Figure 2.1 Login Activity Diagram 8](file:///C:\Users\Nicholas%20McCaig\Documents\workspace\SDI\SDI-Project\Documentation\Software%20Design%20-%20Project%20Design.docx#_Toc63517532)

[Figure 2.2 Send Message Activity Diagram 9](file:///C:\Users\Nicholas%20McCaig\Documents\workspace\SDI\SDI-Project\Documentation\Software%20Design%20-%20Project%20Design.docx#_Toc63517533)

[Figure 2.3 Make Room Activity Diagram 10](file:///C:\Users\Nicholas%20McCaig\Documents\workspace\SDI\SDI-Project\Documentation\Software%20Design%20-%20Project%20Design.docx#_Toc63517534)

[Figure 3.1 Class Diagram 11](#_Toc63517535)

[Figure 5.1 Component Diagram 17](#_Toc63517536)

[Figure 6.2 Login FSM 19](#_Toc63517537)

[Figure 6.3 Publisher FSM 20](#_Toc63517538)

[Figure 6.4 subscriber FSM 21](#_Toc63517539)

[Figure 7.1 Send Message Communication Diagram 22](#_Toc63517540)

[Figure 8.8.1 Deployment Diagram 23](#_Toc63517541)

[Figure 9.1 GUI Mock-up 24](file:///C:\Users\Nicholas%20McCaig\Documents\workspace\SDI\SDI-Project\Documentation\Software%20Design%20-%20Project%20Design.docx#_Toc63517542)

## Introduction

Our application will be a messaging/chat service using MQTT to facilitate communication using a Server/Client model. We have based our application on two existing messaging services. Discord a gamer-oriented application with a range of features for allowing users to communicate with others (Discord, 2021). We have also based our application on Slack, a messaging service which is geared towards a workplace (Slack, 2021).

We will use the principles of UML design through Diagrams to create an effective solution to the problem using information gathered from our research to make sure the design is fit for purpose and effective. Working as a group with regular meetings helps us maintain a high-quality design which can then be transferred into an effective codebase for deployment.

# Use Case Diagrams:

## Introduction:

### Figure 1.1 (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 database.

### Figure 1.2 (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 database
* If the user is offline:
  + The source message is stored within the database 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 1.3 (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 database is updated. The Admin then has the option to add users to room and make these users moderators. The Admin can also make channels in the room. Whenever any changes are made the database is updated.

****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 .1: Login Use Case

Figure . Login 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 database. It also includes validate connection. Validate connection also extends to user online. User online includes update logs. Update logs includes update database. Update database is associated with the Database 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.

Figure 1.2: Send Message Use Case

Figure . Send Message Use Case

Admin actor is associated with make room use case. Make room use case includes update database. Update database is associated with the database actor. Make room extends to create channel. Create channel includes update database. 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.

Figure .3: Make Room Use Case

Figure . Make Room Use Case

# Activity Diagrams:

## Introduction:

### Figure 2.1 (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 database.

* 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 2.2 (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 database. 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 database. 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 2.3 (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 database 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 . Login Activity Diagram

#### Figure . 1: 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 database. 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 2.2: Send Message Activity Diagram

Figure . 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 database 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.

The make room activity is initiated by 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, 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 decision node. If no more users to add, 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, database is updated. Control flow moves to activity final node.

Figure 2.3 Make Room Activity Diagram

Figure . Make Room Activity Diagram

# Class Diagram

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 . Class Diagram

### Figure 3.1: 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 has 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 the 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 meta data 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 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 is able to be a member of none or many rooms however a room must have one member who must also be the administrator the administrator is able to remove other 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 4.1 (Login)

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

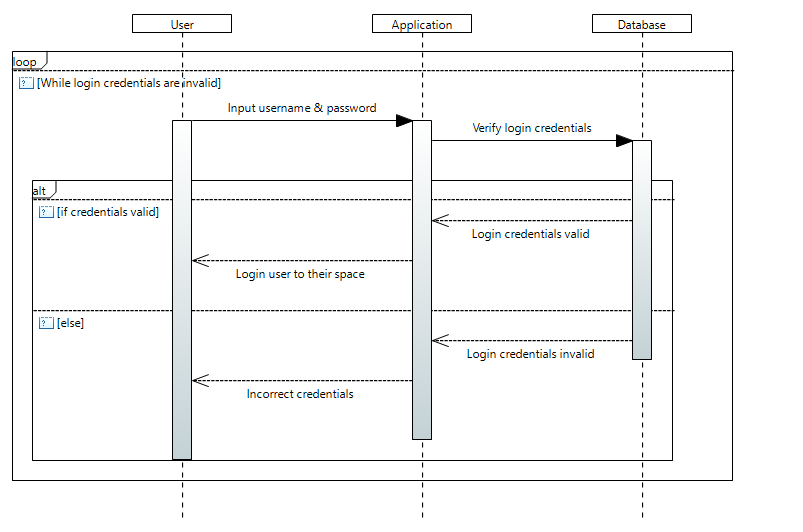
### Figure 4.2 (Send Message)

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.

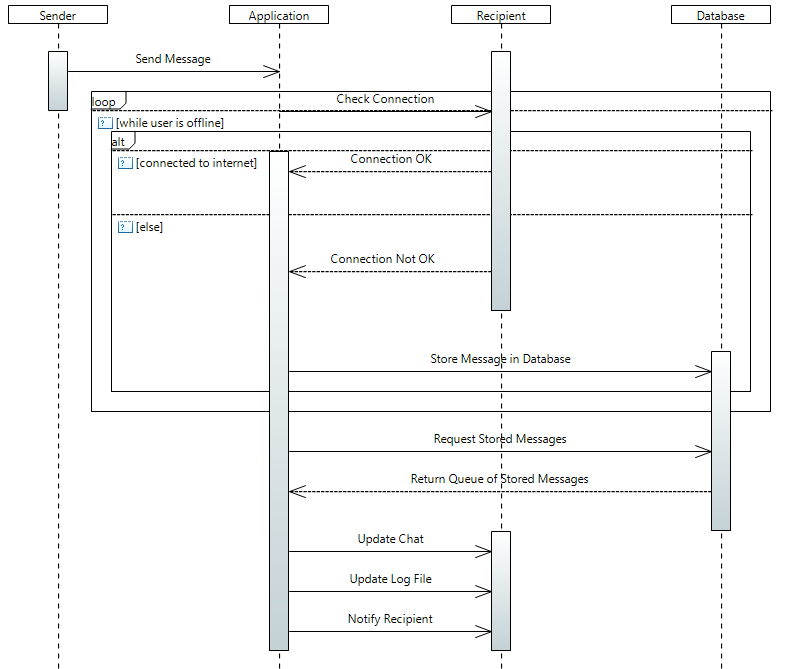
* If the user is online:
  + The message is sent by the sender
  + The recipient requests new messages stored in the database
  + The database returns a queue of all the messages
  + The chat and log and updated and the recipient is notified
* If the user is offline:
  + The message is stored within the database 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 4.3 (Make Room)

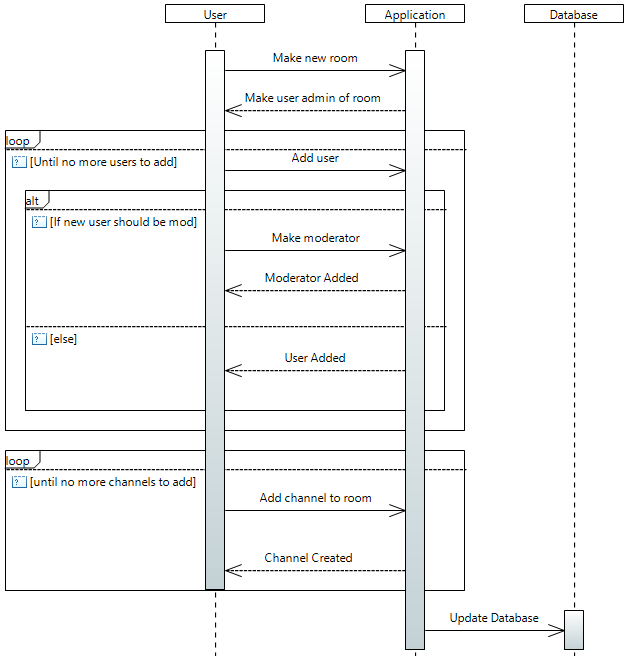
The diagram shows the process that occurs when a user makes a new room. The user is assumed to be Admin as by making a room the user becomes that Room’s Admin by default. Users are added to the new room, and the admin decides whether they are moderators or not. Then channels are created in the room,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

Figure 4.1 (Login)

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 Database lifeline for verification. Within an alt operator, the database lifeline returns whether they are valid or not. The application lifeline sends a return message to the user lifeline to allow the user to login if the credentials are valid. In the guard of the alt operator, the database 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 4.2 (Send Message)

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 database lifelines to store the message in the database. After a connection has been established, the application lifeline sends a call message to the database 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 to notify the recipient.

Figure 4.3 (Make Room)

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 calloperator 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 database lifeline to update the database.

# Component Diagram

A picture containing text, screenshot, computer, indoor

Description automatically generated

Figure . Component Diagram

The Component Diagram shows the structure of the systems components. There are 3 main components; Messaging Application, Database, 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 componenet using the Channel View interface. The Database component contains the User Table, Chatroom Table, Channel Table and Message Backup components. The Chatroom Table component provides the roomID interface, which is used by the User and Channel Table components. The Channel Table 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 Table uses Get User from within Database 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 Table uses Get Chatroom from within Database 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 Table component via a port into Database.

# FSM Diagrams

## Introduction:

### Figure 6.1 (Login):

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

### Figure 6.2 and Figure 6.3 (Publisher & Subscriber):

Our team decided to split this FSM diagram into two parts that runs 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.

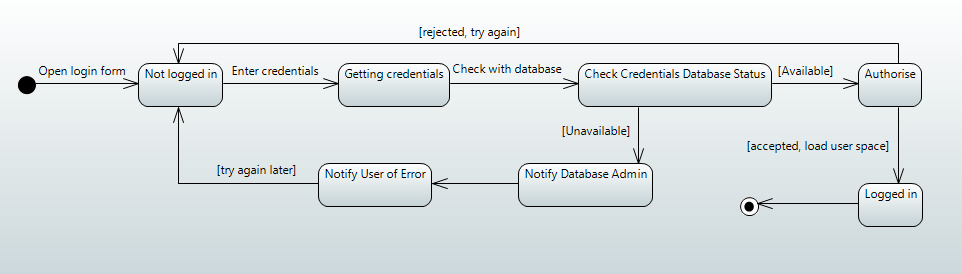


Figure . Login FSM

The login state begins with the transition to the login form on the application. The current state being 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 database status. If the database is unavailable, it transitions to notifying the database admin and the user of the error and unavailability. It then loops back to the not logged in state. If the database is available, it transitions to the authorise state where the credentials are checked against the database. 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.

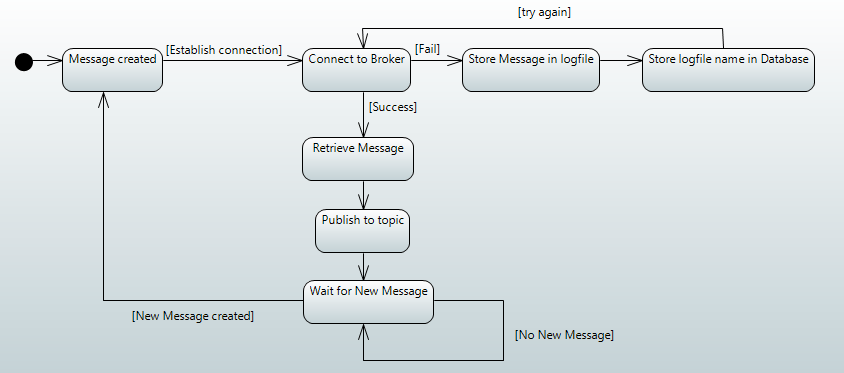


Figure . Publisher FSM

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 database 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.

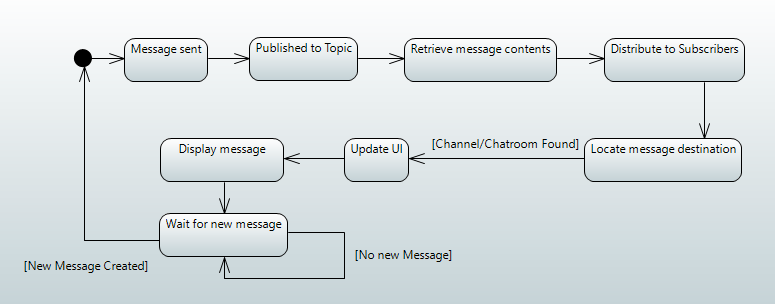


Figure . subscriber FSM

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

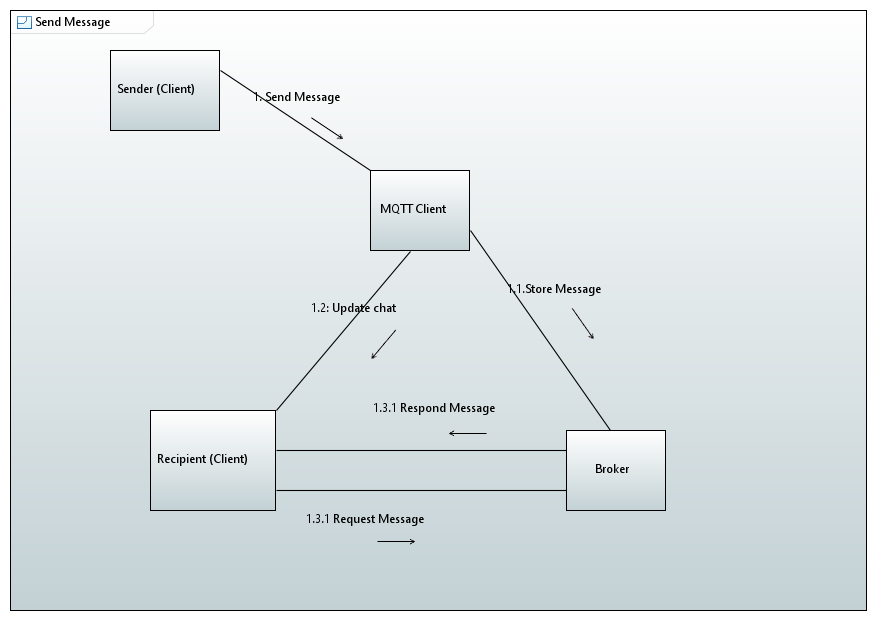


Figure . Send Message Communication Diagram

The Send message 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 database. 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 database which will respond with the same message which was sent by the sender.

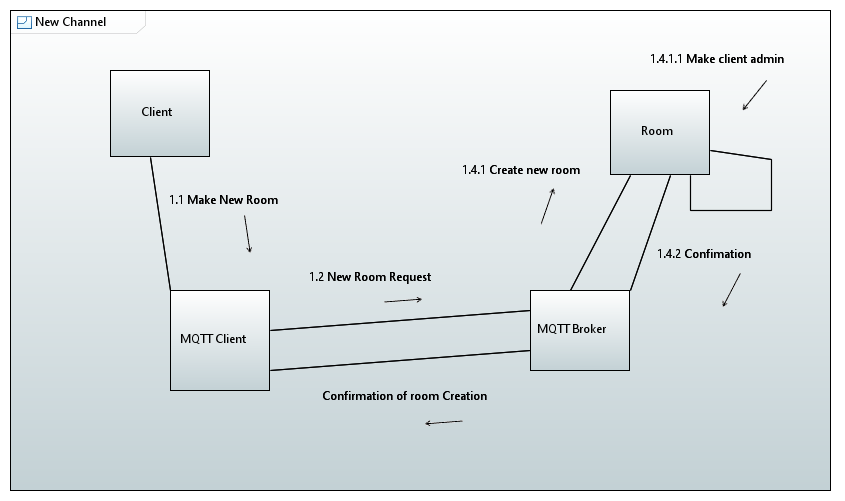


Figure . New Channel Communication Diagram

The New channel Communication Diagram demonstrates how a new Channel is created. First the client Sends the request for a new room to be created which happens at the broker side then it updates the newly created room with the clients ID to set them as the admin. Finally the confirmation and room is sent back to the client.

# Deployment Diagrams



Figure 8.. 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 depend on a RDBMS to store information about users and their messages. The RDBMS will use Boost as a file management system to store the user’s uploaded files. We will use MySQL as the RDBMS.

# GUI Mock-up

### Figure 9.1 (GUI Mock-up)

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 the application may function.



Figure . 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.

# Design Pattern

### 

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

|  |  |
| --- | --- |
| Pros | Cons |
| Factory Method | |
| * We are able to keep implementation and the creation separate. * We keep object creation in one place. * You are able to add new types without breaking existing code | * You may need many subclasses to implement pattern * The design pattern works well in particular circumstances but can be very complex if not appropriate. |
| Abstract Factory Method | |
| * We are able add new variants of objects without modifying existing code * You can keep the object creation in one place | * 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 | |
| * You can make objects step-by-step depending on the required parts. * The components can be reused. | * Makes code 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. |

We have selected **Factory Method** as our design pattern (Refactoring guru, 2021), we use creator classes to declare the new objects such as the User or a Message each object is unique but has the same functions but different kinds of messages or users can inherit different functions such as a user which as an admin has more functionality such as adding or removing a user or and message which contains a file which can be downloaded.

We chose this method as we do not know exactly what types and dependencies the objects may rely on as they depend on the kind of message the user may send which allows for inheritance depending on what message the user sends or what level of permissions they may have in a selected channel. We also know that most will use the same foundation and we want to use existing objects rather then rebuild them each time as most functionality remains constant for each object, this helps avoid duplicated code within the codebase.

# Libraries

The following libraries will be used to complete this project:

##### QT DESIGNER 5.15

Qt Designer will be used for **GUI** design.

##### Paho MQTT

Paho MQTT will be used for **Message Passing** between the server and the client devices.

##### Github

Github will be used for **GIT** project management.

##### STL library

The STL library will be used for **data structures**.

##### Doxygen

Will be used to automatically create **documentation** for the program from the **source files**.

##### Eclipse Papyrus

Papyrus will be used for the **UML design** of the program.

##### Boost

Boost will be used for **file Management.**

##### Trello

Trello will be used for **workflow management**.

##### Code Together

Code together will be used for group collaboration.

#### MYSQL

MySql will be used as the **database**.

# References

Discord, 2021. *Discord.* [Online]   
Available at: https://discord.com/  
[Accessed 4 Feburary 2021].

Slack, 2021. *Slack.* [Online]   
Available at: https://slack.com/intl/en-gb/  
[Accessed 4 Febuary 2021].