Client-Server Chat Project
Software Workshop

Team Chicago	**Student ID**	**Email**
Minhua Guan	1554057	mxg558@cs.bham.ac.uk
Robert McNally	1452543	rxm448@cs.bham.ac.uk
Jiayu Liu	1796736	jxl1106@cs.bham.ac.uk
William Nunn	1559536	wxn537@cs.bham.ac.uk
Vasileios Manolis	1772373	vxm773@cs.bham.ac.uk

Tutor: Cory Knapp

Contents

Introduction

The project report entails the documentary process in construction of a Client-Server Chat system. Its intention is to document the thought process, planning, construction and eventually the active running of the system. The Client-Server Chat created by Team Chicago embarked on challenges in designing and implementing a wholesome chat program that allows their users to login, converse to other users and have the capability to access previous chat histories.

Furthermore, where possible, a number of additional features (such as translation or changing GUI colours) will be implemented.

1. Scope

The scope of the project is to design and implement a simple chat program that allows users to login, chat to other users and access previous chat histories offering real-time text transmission based on a Client-Server architecture. Apart from the basic functionalities, the system will also provide a translation tool.

The core running of the system involves the communication of the database to the server via JDBC and the University PostgreSQL, allowing chat histories to be stored. Sockets are also explored for communication integration between the server and clients. The multi-threaded server implemented are to handle multiple clients simultaneously. To aid client interaction, the client user interfaces are constructed based on Swing which displays the necessary components to guide the user throughout the chat and to allow the ease of use.

The basic features are as follows:

User Registration:
User can sign-up selecting their username and password.

User Sign-In:
After a successful registration, users should sign-in by entering their username and password.

Common Chat Room:
A common Chat Room with all the online users is available for all users to chat with each other. When a user sends a message, the message is echoed to all the clients.

Private Chats:
Users can select users who they would like to create a private chat room and select the “New Group” button to create a new Chat with the selected users only.

Translation:
User can translate short messages in 91 languages by calling an external API. The team has acknowledged the security and performance risks of integrating an external API but decided to proceed adopting an experimental approach towards this functionality. In the future, only a trusted API should be used in the system. The translation service aims to build bridges between users that do not speak the same language.

Future Additional Functionalities:
In the future, the system will be able to support additional functionalities such as spell-checking and message encryption.

2. Requirement Analysis

The requirements for the system, consist of two main categories: functional and non-functional requirements.

Overview of Functional Requirements
The system must allow the user to:

1. Create a new account. *- High*
2. Login to the server using a username and password. *- High*
3. View the list of other users who are currently online. *- High*
4. Initiate chat with any user who is online. *- High*
5. Engage in multiple chats with different users *- High*
6. Allow group chats and inviting of other users to join a chat. *- Medium*
7. Allow a user to view previous chats of which they have been a part of. *- Medium*

2.1 Functional Requirements

REQ1. **Sign-up:** The system must allow the user to create a new account.

1. The system must allow the user to select a **username** between 5-20 characters and a **password** between 8-32 characters. *- High*
2. The system must inform the user if the username or password does not fulfil the length criteria. *- High*
3. The system must inform the user if the username has been taken by a registered user. *- High*
4. The system must inform the user if the server cannot be found. *- High*
5. The system must show a confirmation message when their registration is successful. *- High*
6. The system must allow the user to select the IP and the port they wish to connect. - Medium

REQ2. **Log-in:** The system must let the user log-in using their username and password

1. The system must allow the user to enter in their username. *- High*
2. The system must allow the user to enter in their password. *- High*
3. The system must inform the user if the username does not exist. *- High*
4. The system must inform the user if the password does not match the username. *- High*
5. The system must allow the user enter the chat after correctly entering their username and password tapping on the "Login" button. *- High*

REQ3. **View a List of Online Users:** The system must allow the user to view a list of other users who are currently online.

1. The system must allow the user to view a list of all the users who are currently online. *- High*

2. The system must allow the user to view a list of other the users participating in each Chat room. *- Medium*
3. The system should allow the user to view a list of other users who are currently offline *- Low*
4. The system should allow the user to view a list of other users who are busy. *- Low*
5. The system should inform the user that they are online and active when they've logged *- Low*

REQ4. **Chats:** The system must allow the user to initiate Chat with multiple chats with different users.

1. The system must have a Chat Room with all the current online users. *- High*
2. The system must allow the user to view the names of other online users. *- High*
3. The system must allow to initiate a private Group chat. *- High*
4. The system must be able to process and accept keyboard inputs from the user. *- High*
5. The system must allow user to send message via means of clicking “Send”. *- High*
6. The system must indicate the sender of the message. *- Medium*
7. The system must indicate the time the message was sent. *- Medium*
8. The system must inform the user about the activity (online, offline, busy) status of anyone they are trying to initiate chat. *- Low*
9. The system must allow the user to receive messages from the interlocutor. *- High*
10. The system must allow the user to edit their message anytime they wish to before sending. *- High*
11. The system must allow user to send message via means of clicking the “Enter” key *- Low*
12. The system must let the user to delete anyone they no longer wish to be in the new group. *- Low*

REQ5. **Invite More Users:** The system must allow the user to invite others join an existing chat .

1. The system must allow the user to view all the participants in the existing group chat. *- Medium*
2. The system must allow the user to invite/add new participants from their contact list. *- Low*
3. The system must all allow the user to remove existing participants from their contact list. *- Low*

REQ6. **Leave Group:** The system must allow the user to exit any chat room they wish to do so.

1. The system must allow the user to view all the group chats they are involved in. *- High*
2. The system must allow the user to leave any group they wish. *- Medium*
3. The system must inform the user they have left the group. *- Low*
4. The system must inform the user they cannot converse in group(s) they have left. *- Low*

REQ7. **Chat Logs:** The system must allow a user to view previous chats of which they've been part of.

1. The system must allow the user to view their chat history. - *Medium*
2. The system must allow the user to click on anyone in their contact list and see their chat history. - Medium

REQ8. **Advanced Features (Translation, Spell Checker etc)**
1. The system should underline misspelled English words while a user is typing. *- Low*
2. The system should allow the users to translate input. - *Medium*
3. The system should allow the user to change the colour of the Chat windows according user preference. - *Medium*
4. The system should provide encryption tools. *- Low*
5. The system should allow users to enter the chat anonymously. - *Medium*

2.2. Non-functional Requirements

REQ1. **Performance:** The system's response must be fast.
1. The system must send messages to the recipients within 2 seconds of sending. - *High*
2. Actions such as logging on, registering, opening a chat window should be accomplished in under 5 seconds - *Medium*

REQ2. **Survivability:** The system should recover from bugs.
1. The system should be easily restartable in the case of a crash. - *Medium*
2. The system should not completely crash in the case of small errors (e.g. users failing to connect/open a chat windows with each other/dropping out mid conversation). - *High*

REQ3. **Maintainability:** The system should be easily maintainable
1. Any error should be easily locatable and fixable due to simple architecture and defensive programs. - *High*

REQ4. **Security:** The system must be secure.
1. The system must securely store the data collected, preventing third parties to access it - *Medium*
2. No user should be able to access the private chat logs and the password of any other user - *Medium*
3. No user should be able to edit the database other than to register an account. - *Medium*

REQ5. **Usability:** The system must be simple and attractive.
1. It should be clear on any given interface what the users' options are for interacting with the software. - *High*

2. The system must take less than 5 minutes for an average tech-savvy user to understand it. - *Medium*
3. The system must have a minimal uncluttered user interface and a pleasant design. - *Medium*

REQ6. **Data Integrity & Retention:** The system should reduce data integrity risk.

1. The database should reliably store chat histories and user profiles (e.g. correctly rejecting register attempts with a taken username - *High*
2. The system should maintain all chat histories even between instances of the system going down and back online. - *Medium*

REQ8. **Testability:** The system should support JUnit and user testing.

1. The system should be delivered with a test plan. - *High*
2. The system should support testing of each component separate from any other component. - *Medium*

REQ9. **Scalability:** The system should be able to meet larger operational demands in the future.

1. The system must be able to support the addition of message encryption. - *Low*
2. The system must be able to support the addition of translation via connecting to an external API. - *Low*
3. The system should support the addition of further functionality (e.g. games in the chat windows) - *Low*
4. The system should be able to be adapted to an increased load of simultaneously active clients. - *Low*

REQ10. **Availability:**

1. The system should be open to any connection whilst it is operational. - *Medium*

3. Use Case Diagram

A Use Case diagram for the system indicating the actors involved and making use of stereotypes such as <<extend>> and <<include>> can be found below. This is intended to give an impression of how the system works from the user's perspective.

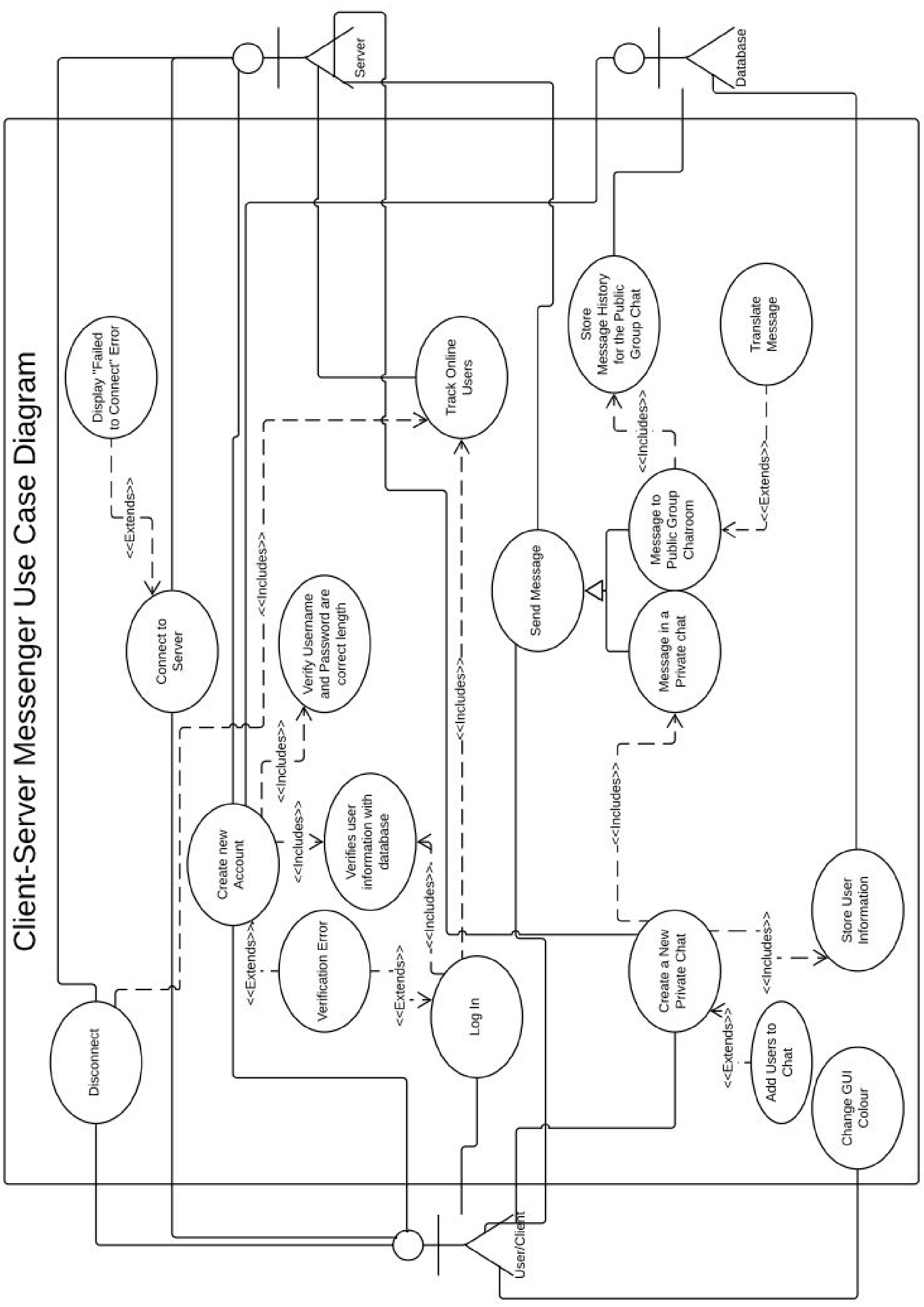


Figure 1: Use Case Diagram

Use Case I: Connect to Server

Actors: User and Server

Preconditions

The server is open.

Flow of events

The user enters the port number and IP address and presses the connect button. The server waits in an infinite loop to listen for connections. When a client connects it creates a new client socket and creates a thread for that user.

Post-conditions

If successful the client will form a connection with the server.

Scenarios

1. The user enters the correct information and connects successfully.
2. The user enters incorrect information and fails to connect. An error box explaining this appears.

Use Case II: Create New Account

Actors: User, Server and Database

Preconditions

The client has connected to the server.

Flow of events

The user enters their desired username and password and clicks "OK". The server checks the username and password contain an acceptable number of characters (within a certain range) and checks whether this username is already in the database. If not this new username and password are inputted into the database.

Post-conditions

If successful the client is informed and they can now go on to sign in with this information. Otherwise they are presented with an error depending on the problem they have encountered.

Scenarios

1. The username and password meet the correct conditions and a new user is added to the database.
2. Either the username or password are not of appropriate length so an error message is sent to inform the user.
3. The username is already in use (previously stored in the databases) so an error message is sent to the user.

Use Case III: Log In

Actors: User, Server and Database

Preconditions

The client is connected to the server and the user has signed up to create an account.

Flow of events

The user submits their username and password. The server then checks this information is stored in the database (i.e. this is a valid user) and then checks that this user is not already logged in. The client is informed of a successful login and the username is added by a thread to the list of users that is online. The thread then enters a while true loop to infinitely listen for requests from the client.

Post-conditions

The user has successfully logged in and is presented with the public group chatroom.

Scenarios

1. The user logs in successfully
2. The username is not contained in the database so log-in is not possible
3. The username is valid but the submitted password does not match that stored in the database so log-in fails.
4. The username submitted is already online, the server prevents this log-in so that the user does not sign in twice.

Use Case IV: Send Message

Actors: User and Server

Preconditions

The user has logged in successfully.

Flow of events

The user types and submits a message. If they choose this message can be translated from an inputted language to another inputted language. The server adds the message to a list of objects of type “message” that have been sent. All logged-in client loop infinitely to receive all messages from their offsets (i.e everything they haven’t received so far). if the message is in the public group chat the translate feature is available, and the database will store a log of the messages.

Post-conditions

The message is received by all relevant recipients (all in the group chat, specific users in private chats)

Scenarios

1. The user sends the message to the group chat. It is received by all online users and recorded by the database.

2. The user translates the message to a different language and sends it to group chat as in scenario 1.
3. The user sends a message to a private chat and it is received by all online users in that chat.

Use Case V: Create a New Private Chat

Actors: User and Server

Preconditions
The user has successfully logged in.

Flow of events
The user clicks the "group" button and a new window opens. They select all the other users they wish to add to the chat from a list of all the users that are currently online, name the chat and create a new private chat. The chat is stored as one of each user's valid chats in the server. They can later add new users to pre-existing chats.

Post-conditions
A new chat opens containing all the selected users. Than can send and receive messages.

Use Case VI: Change GUI Colours

Actors: User

Preconditions
The user has successfully logged in

Flow of events
The user selects a new colour from a colour palette menu and the GUI colours are changed.

Post-conditions
The GUI's primary colour is now that which was selected by the user.

Use Case VII: Disconnect

Actors: User and Server

Preconditions
The user is logged in and online.

Flow of events
The user clicks the disconnect button. The server receives their name and then executes a loop to remove them from their private chats. A response is sent to each member of each chat to inform them the user is gone. The disconnecting user is removed from the list of users that are currently online, via another loop that messages all online users to inform

them of the disconnect, the users client will be sent their own username and the client will recognise this and exit.

Post-conditions
The client is removed from all their chats and is no longer connected to the server.

4. Architecture

4.1 Rationale

The system's architecture consists of three main components:
1. Server
2. Client
3. Database

The architecture of the chat server should allow interchangeable communication between the database and the server and efficient client-server communication for the chat to work according to user discretion.

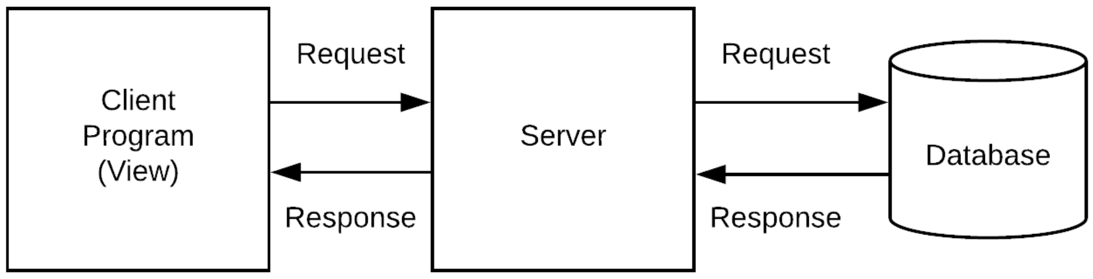


Figure 2: Architecture Diagram

4.2 Server

The Server consists of six total classes; Message, User and Chat are for the creation of those respective objects, used to store the messages the users have sent, the information of users currently online, and the currently active private chats respectively. The Server_GUI class is used to start the server with a chosen port. The Server class itself listens for connections on that port and accepts the connections, starting a ServerThread for each one which it then acts as a model for, containing the methods used for interacting with the data stored on the server such as all the public messages which have been sent, the currently online users and active chats.

The serverThread itself is what receives communication from the client, reading the head of the message and taking the appropriate action, which usually involves calling a method from the server, responses are either then sent directly back from the ServerThread or from the server in the case where several other clients need to be notified.

the connection to the database is handled via the server, if needed it queries the database for information such as the details of existing users, and also requests the public chat history each time a new user signs in.

4.3 Client

The Client side of the architecture consists of 8 classes, The Client class acts as a model, sending and receiving all communication from the server, as directed by the views. Most of the decision making is done through the ChatGUI class which is the main lobby view for all users, it contains a thread which reads the messages sent to the client and takes the appropriate action, mostly either opening new views or updating the display of ChatGui itself.

While actually using the client the user is first presented with the login page, in order to perform any actions they must input a port and ip address and connect to the server, after being notified if the connection has been successfully formed they are free to take other actions, such as logging in with an existing account, creating a new account, or changing their password. each of which opens a new view, with login opening the ChatGui which also starts the thread and starts checking for messages from the server.

After logging in the user is then free to send and receive messages from other online users, change the appearance of the main lobby and start private group chats with as many other users as they desire. Within which they can communicate, add other users or leave, the user may close everything by closing the main window via the red x or clicking the disconnect button.

4.4 Protocol

A simple protocol was being used to enhance the server and client communications. Various protocol implementations were discussed before deciding using a simple protocol that was used in the Operating Systems module because all team members had previous experience using it. The used protocol simply packs all the fields into a single string which is then split back into an array of strings when it is received, so that each part of the message can be read.

the following table details the types of messages sent to the server from the client, some result in a direct response from the server, and some cause the server to contact several clients of which the original sender may be one of. For example if a client sends a private message then the server messages all the clients in that private chat, treating the sender no differently from the others, if a user disconnects, all clients are notified with the same message, and by receiving their own name the disconnecting client knows they have done so

Shown in table one below is a breakdown of the messages sent between client and server, they consist of a “type” which is the first string in the array, shown in the action column. This is followed sometimes by a distinction as in the case of update-list, where the request may be to add or remove a user from the list. The rest of the information then follows in the tail of the array.

Action	Request from Client	Response from Server	description
“sign-up”	(type, username, password)	(type, boolean_string:result, message)	a request from the client to create a new user profile
“sign-in”	(type, username, password, boolean_string:annon)	(type, boolean_string:result, message/anon username, server history...)	a request from the client to sign in to the server and begin using it.
“get-message”	(type, offset)	(type, message1, message2, ...)	a request from the client to receive the most recent list of public messages.
“send-message”	(type, content)	(type, result, message)	a request to send a message to the public chat.
“update-list”	none	(type, distinction, username)*	a message received whenever a user signs in or disconnects, telling the client to update its list of online users.
"create-group"	(type, Chat name, username1, username2, ...)	(type, Chat name, username1, username2, ...)	a request from the client to create a private chat group with the specified other users.
"send-private-message”	(type, chat id, content)	(type, chat id, content)	a request to send a message to a private group chat
"add-to-chat"	(type, chat id, username1, username2, ...)	(type, chat id, username1, username2, ...)	a request to add one or more members to an existing private group chat
"disconnect"	(type, username)	(type, distinction, username)*	a request to disconnect this user completely, response is actually an “update-list” message
"leave-chat"	(type, chat id, username)	(type, chat_id, username)*	a request to disconnect from a private group chat

Table 1: Protocol Description, response from server elements with a *, indicate the action messages multiple users, and the user who initiated the action treats the message differently, as they receive their own name.

4.5 Database

As a group we've decided to create 3 tables dedicated to storing account details, public chats and private (group) messages sent between users.

To elaborate, the ‘accountprofile’ table holds the username, password and user_id of the respected user. The user_id acts a primary key for each new registered user. The key features of the user_id is that it automatically increments with each new registered user thus allow us to observe the amount of registered users.

The ’groupchat’ table was dedicated to storing chats between private messages or group chats shared by more than two users. The ‘groupname’ column consists of the communicating users, each user is distinct by a colon. The first username is the sender and the remaining username’s after the colon are the receivers. The second column ‘message’ is to store the string of message that was sent from the first username. Finally, we included a ‘message_id’ column, which allows the controller to keep track of how many messages being stored. This may be beneficial for future prospect if the database has limited space.

The third table ‘logstore’ also known as the public chat log storage which stores the chats of any registered users who has access to the server. Any messages that were posted to the server will be displayed into the GUI. The table includes a username, message, time and msg_id column. The username column encapsulates the username, the msg column contains the string of message and the ‘time’ column holds the time of when the message was sent. The fourth column ‘msg_id’ assign a primary key to each string message sent. This is an important characteristic which allows SQL syntax to be used, whereby the messages are ordered according to the msg_id thus messages that are sent back to the GUI are of ascending order (chat-log history display).

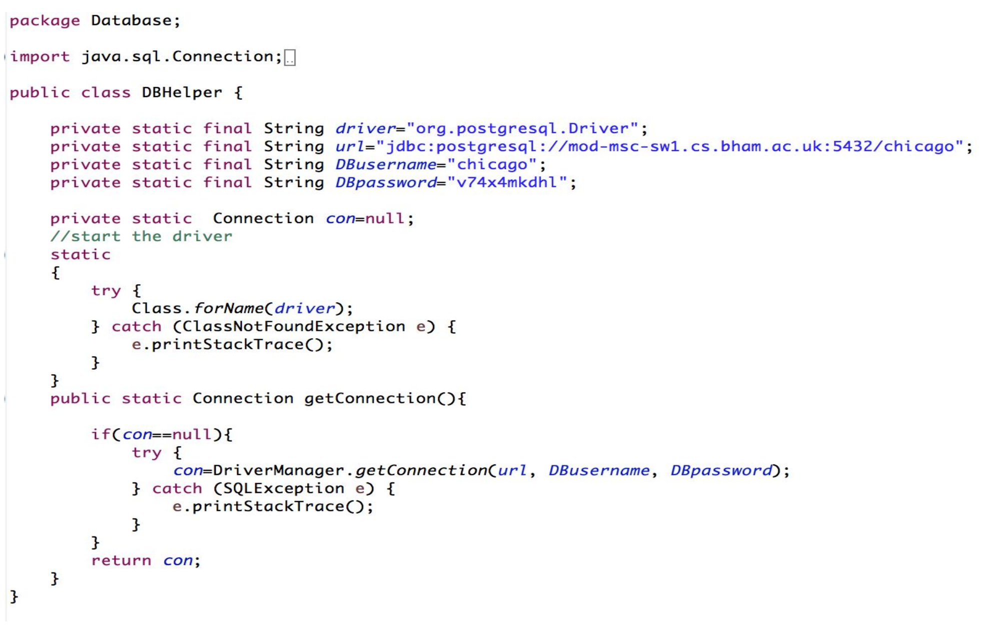
How the database was populated?

The database was populated using the following methods under ‘Method Name’ column in respect to their sql syntax which performed the insertion.

Table Name	Method Name	SQL syntax
Account profile	createUser	String sql = "insert into accountprofile values(?,?,(select Max(user_id)+1 from accountprofile))";
	isValidUser	String sql = "select username, password " + "from accountprofile " + "where username = ? and password = ?";
	isExistUser	String sql = "select username " + "from accountprofile " + "where username = ?";
groupchat	groupLog	String sql = "insert into groupchat values(?,?,(select Max(id_message)+1 from groupchat))";
logstore	getHistory	String sql = "select msg,time, username from logstore order by msg_id";
	saveLog	String sql = "insert into logstore values(?,?,?,(select Max(msg_id)+1 from logstore))";

Table 2: SQL Statements

How was it connected?



```
package Database;

import java.sql.Connection;

public class DBHelper {

    private static final String driver="org.postgresql.Driver";
    private static final String url="jdbc:postgresql://mod-msc-sw1.cs.bham.ac.uk:5432/chicago";
    private static final String DBusername="chicago";
    private static final String DBpassword="v74x4mkdhl";

    private static  Connection con=null;
    //start the driver
    static
    {
        try {
            Class.forName(driver);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
    public static Connection getConnection(){

        if(con==null){
            try {
                con=DriverManager.getConnection(url, DBusername, DBpassword);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        return con;
    }
}
```

Figure 3: Java Source Code for Connecting to the Database

The above DBHelper class connects to the database Chicago. The elements to connect includes the driver, url, database name and the password for access.

Steps to connection:

1. Download the JDBC driver and implements into the database connection class via the build path route.
2. Insert the driver name, url, database name and password.
3. Class.forName initiates the driver.
4. Then attempts to connect to the database using the url, database name and password.

How does the JDBC work?

The JDBC which stands for the 'Java Database Connectivity' sends out connection to the desired database and merge with the protocol that we've constructed. This allows us to perform sql queries on our database tables. *As a summary, JDBC purpose is to establish a connection to our database.*

The registration steps from the sign up request to database storage

The diagram below displays the steps from the stage where the client enters their username and password to the stage of storing it into the database.
The sign-in method is similarly the same apart from it is not stored into the database but compared with the information that's already stored in the database from previous registration.

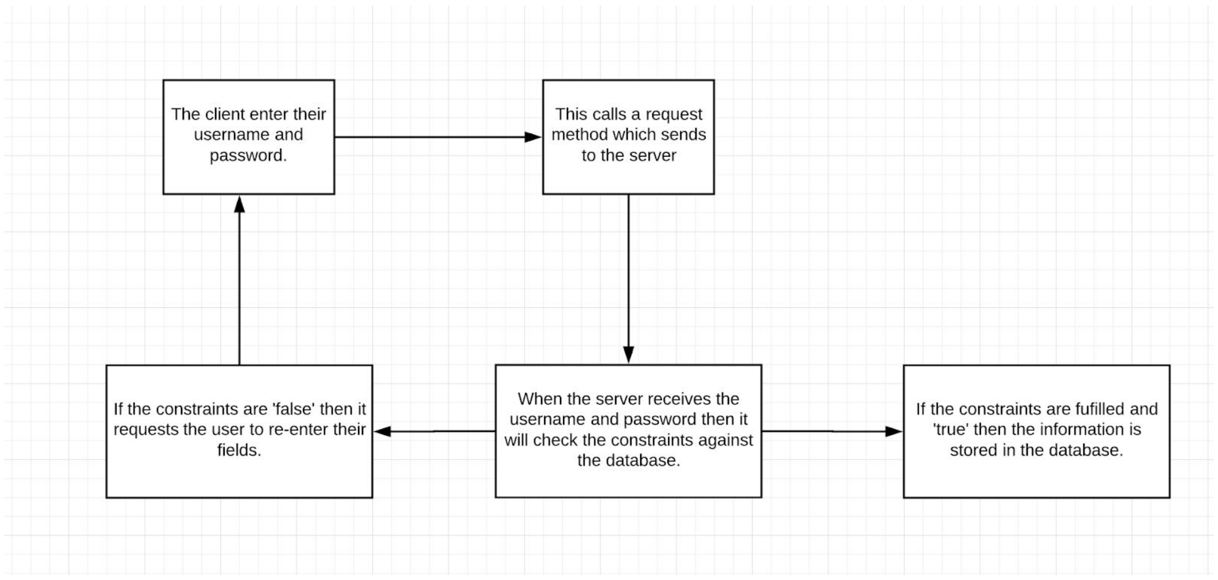

The client enter their username and password.
This calls a request method which sends to the server
When the server receives the username and password then it will check the constraints against the database.
If the constraints are 'false' then it requests the user to re-enter their fields.
If the constraints are fufilled and 'true' then the information is stored in the database.

Figure 4: Database Diagram

Preference of using prepared statements and not statements?

Using prepared statements allows the same instruction to be reused in multiple contexts. If we were to use regular statements we would have to have individual statements for each new incoming user. Therefore, using prepared statements reduce the amounts of sql syntax we have to construct significantly for each new user.

4.6 Graphical User Interface (GUI)

Prototype Design: Before implementing the graphical user interface the team designed the prototype using Balsamiq.

Login Screen: The initial screen users see it the login screen. Users should enter the port and the IP address they wish to connect and press the Connect button. After connecting to the server, they can create a new account or log-in if they are already registered users.

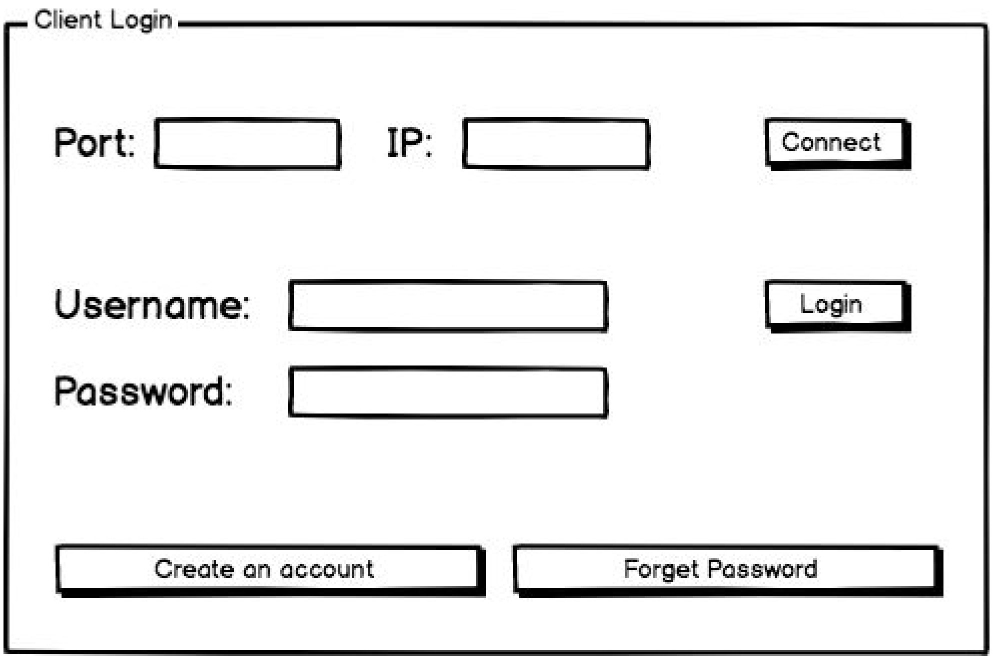


Figure 5: GUI Login - Prototype

Chat Screen: After a successful login, users see the Chat screen which let them chat with each other, see a list of currently online users or customise their user interface by picking the color of their choice. Furthermore, they can create a new private Chat Room with the users of their choice who are currently online, by picking the usernames and then pressing the “Group” button.

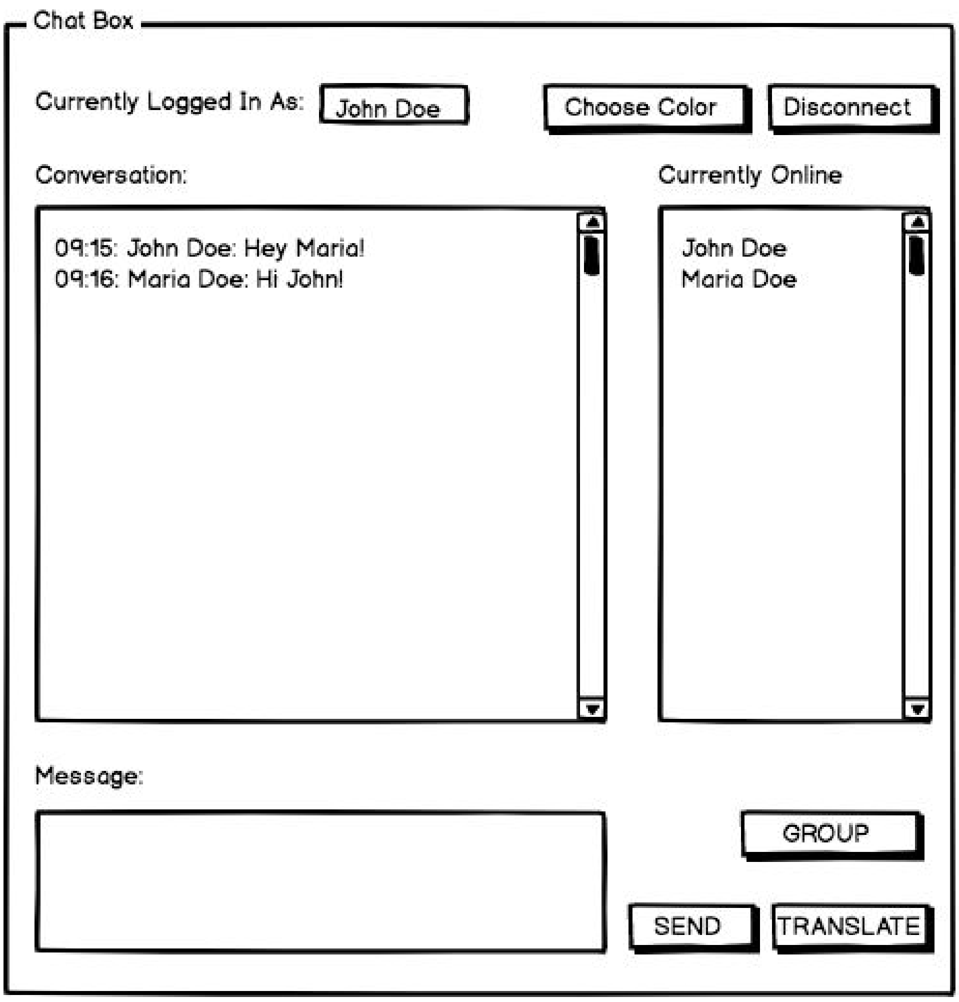


Figure 6: GUI Chatroom - Prototype

Final: The final GUI of the system can be found below.

Login Screen: The initial screen allows the users to connect to a specific Port and IP address. After they are connected to the server, they can login, create a new account or join the chat room Anonymously.

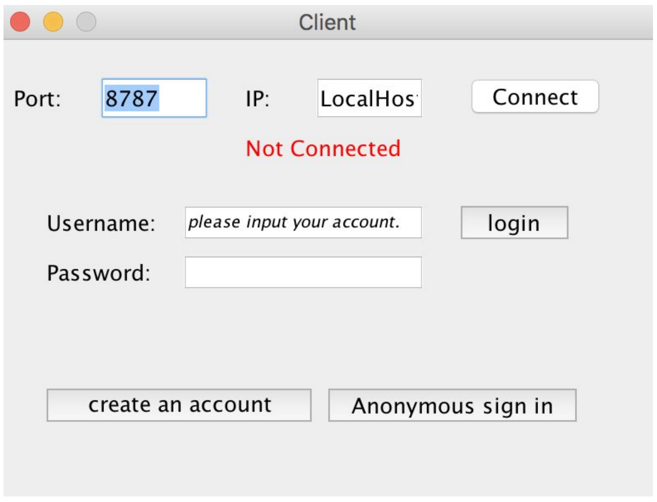


Figure 7: GUI - User Login

Main Chat: After a successful log-in users see the common Chat room. At this window, users can Chat with each other in a common Chat room, see the currently online users and their username. Furthermore, this screen allows them translate small messages as well as customize the program according to their favorite colour. They can also create new private Chat Rooms with select users by clicking the “Group” button.

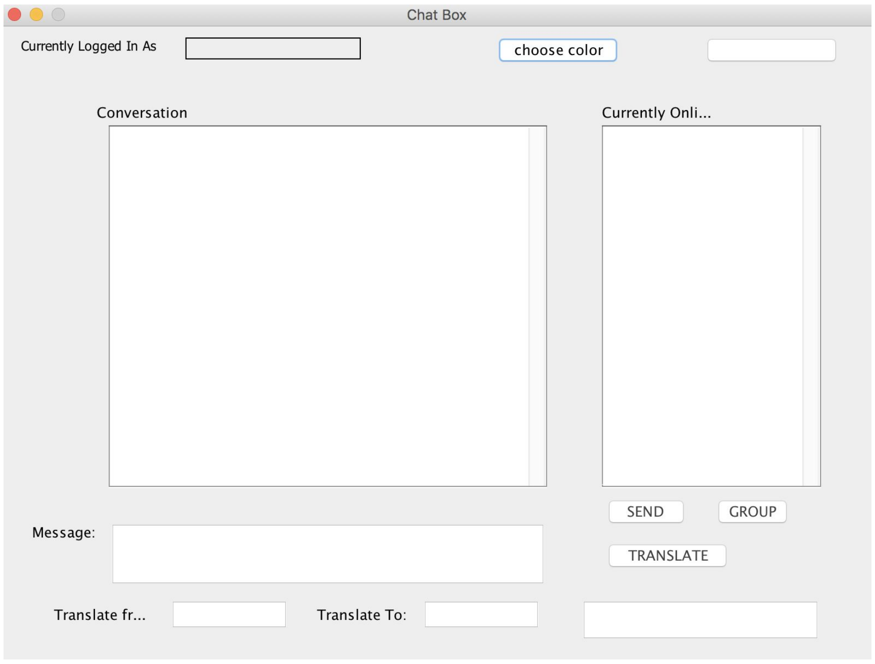
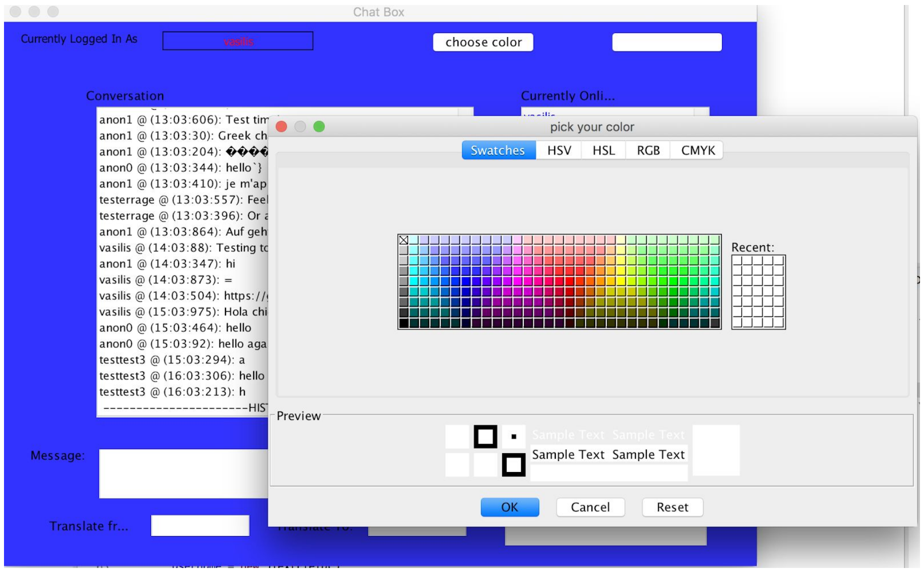


Figure 8: GUI - Chatroom



```
anon1 @ (13:03:606): Test tim
anon1 @ (13:03:30): Greek ch
anon1 @ (13:03:204): ����
anon0 @ (13:03:344): hello`}
anon1 @ (13:03:410): je m'ap
testerrage @ (13:03:557): Feel
testerrage @ (13:03:396): Or a
anon1 @ (13:03:864): Auf geh
vasilis @ (14:03:88): Testing to
anon1 @ (14:03:347): hi
vasilis @ (14:03:873): =
vasilis @ (14:03:504): https://
vasilis @ (15:03:975): Hola chi
anon0 @ (15:03:464): hello
anon0 @ (15:03:92): hello aga
testtest3 @ (15:03:294): a
testtest3 @ (16:03:306): hello
testtest3 @ (16:03:213): h
---------------------HIS
```

Figure 9: GUI - Colour Customization

Create a Private Chat Room: The following screen allows users to create private Chat Rooms by selecting the users they would like to invite to a private conversation.

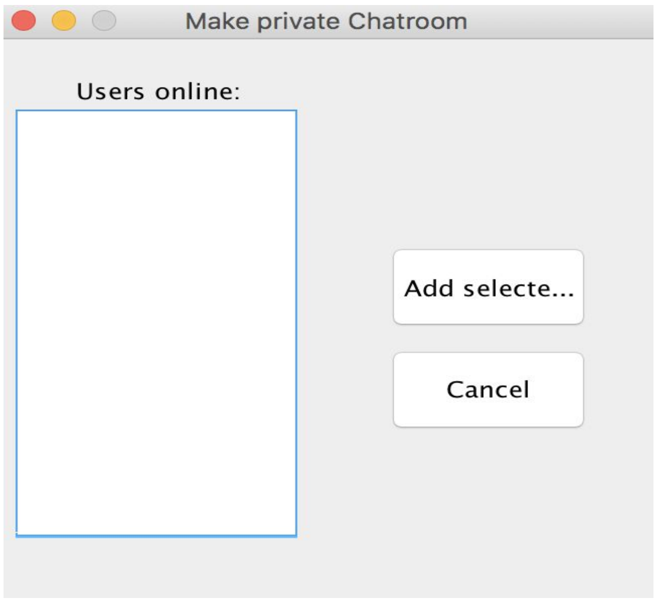


Figure 10: GUI - Select Users for Private Chat Room

Private Chat Room: This is the GUI of a private conversation.

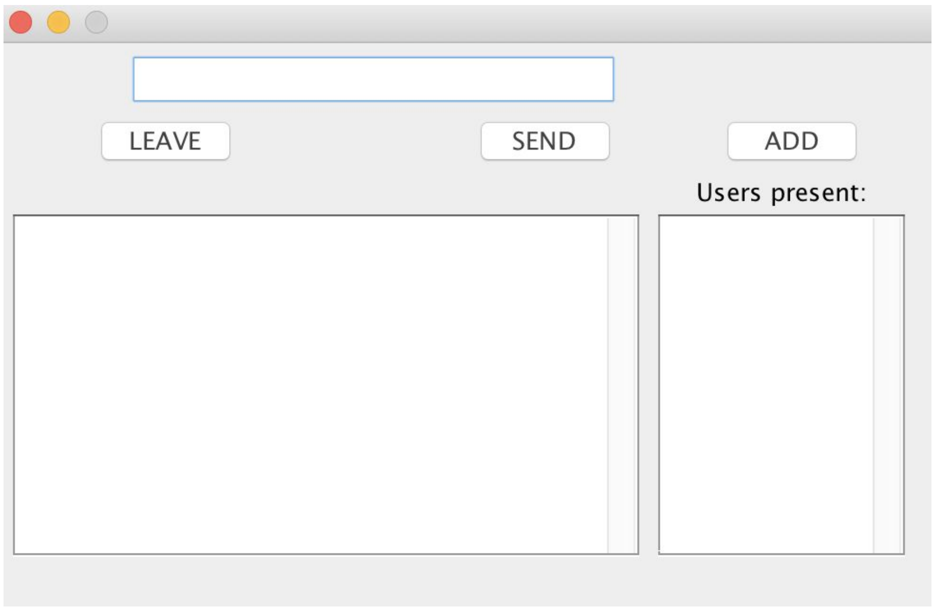


Figure 10: GUI - Private Chat Room

4.7 Testing

The tested product is the Client-Server Chat system created by the Chicago team for the Software Workshop module.

Test Objective: On the server side, the team wanted to ensure that the server is able to handle multiple requests correctly. On the client side, the team wanted to ensure that multiple clients show the right information to the users.

Testing Strategy: Three kinds of test were used to test the system: unit testing, manual testing and user testing.

a. Unit Testing

Definition: JUnit is a unit testing framework for the Java programming language which has been important part of the development of test-driven development.
Methodology: The team wrote JUnit tests for all the non-void methods. All the JUnit tests successfully passed.

b. Manual Testing

Definition: Manual testing was another main testing method that was used.
Methodology: The team manually wrote test cases to test all system's components including the server-client communication, database, GUI, and external API. A detailed table with the Test Cases descriptions as well the Server-Client messages can be found in the Appendix.

c. User Testing

Definition: User Testing was another main testing method used by team Chicago.
Methodology: Due to time constraints, only an external tester was involved in the user testing process that provided feedback about the system. After evaluating the system, the overall feedback from the tester was easy to use in comparison to the already established messengers. However, the tester mentioned that he would prefer the translation functionality to be integrated better with the rest of the system. The team agreed with the tester's opinion but decided to keep the translation and the messaging features separate in order to prevent bugs related to the external translation API.

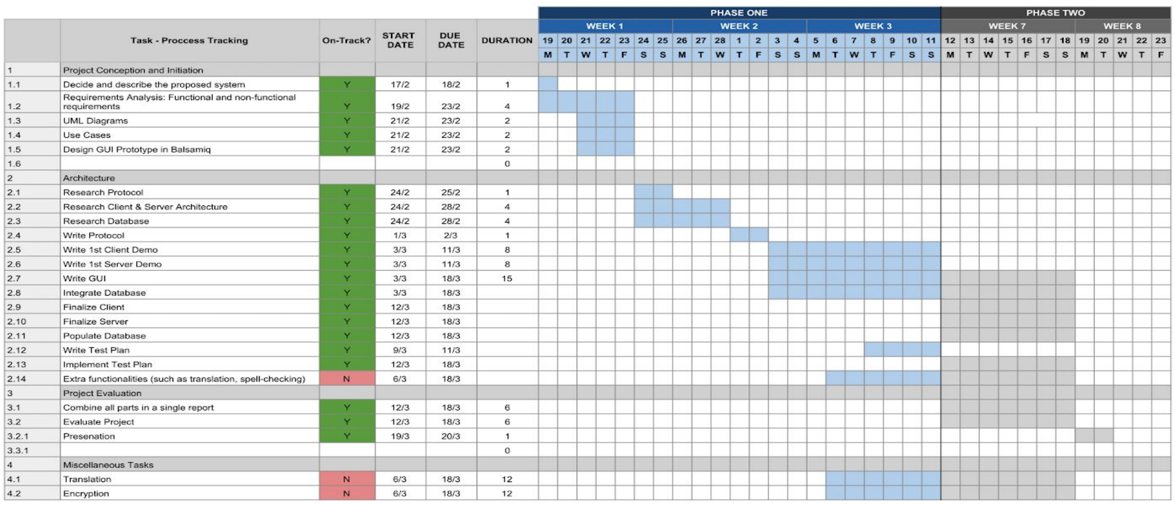
Team Planning & Process

Several ideas were discussed during the first meeting and after carefully considering the project requirements the team decided to implement a Client-Server Chat system. In terms of communication and daily planning, the team selected email and a messaging app as the preferred way of communication in order to progress as fast as possible.

During the first meeting the team discussed each team member's interests, strengths and weaknesses. According to these, the tasks were initially allocated and in pairs of two we started to research about the client-server communication, protocol, database and the GUI. The initial idea was to work in teams of at least two whenever is possible in order to always have a backup person and successfully manage any potential risks or unpredicted situations. Initially, Robert and Jiayu researched about the GUI, William and Vasilis about the client-server communication and Minhua about the database. After the initial research, each team member shared insights and written pieces of code with the rest of the team. Based on that, we continued having programming sessions together as a team as well as in groups of two. The majority of the challenges in each part were presented daily to the group and all the team members provided valuable input. At a later stage of the project, Robert developed a specialization towards the client-server communication, Minhua and Jiayu managed the database while Vasilis and WIlliam started researching for the translation API solutions and started conducting the report. All team members contributed to the testing of the service and to the writing of the report. At the end, the contribution of each team members was equal with multiple rotations according to each team members' interests and needs at each point of the project. The daily meetings and cooperative coding sessions played an important role to be flexible and have rotational roles several times during the project.

Project Planning

In order to monitor the projects processes a Gantt Chart was created and updated regularly. A full size version of the Gantt Chart can be found in the Appendix.

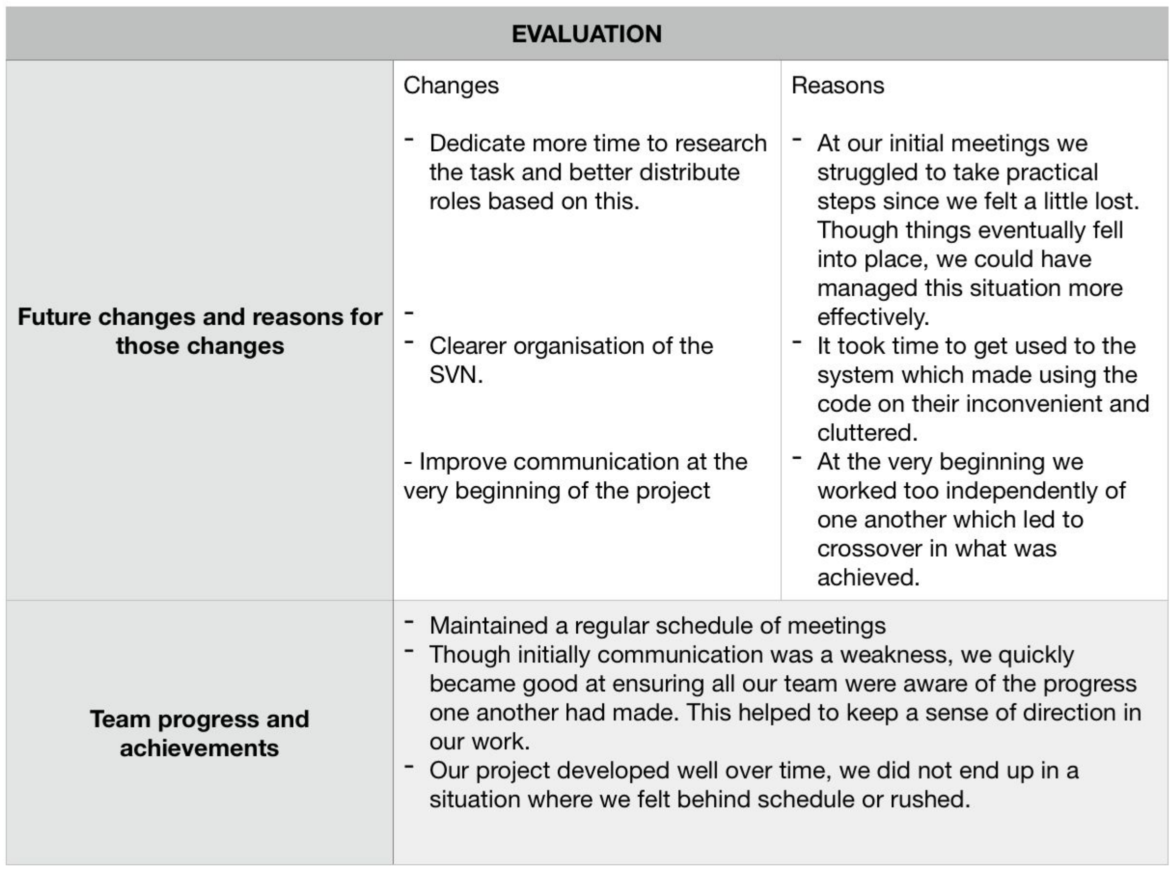


	Task - Proccess Tracking	On-Track?	START DATE	DUE DATE	DURATION
1	Project Conception and Initiation				
1.1	Decide and describe the proposed system	Y	17/2	18/2	1
1.2	Requirements Analysis: Functional and non-functional requirements	Y	19/2	23/2	4
1.3	UML Diagrams	Y	21/2	23/2	2
1.4	Use Cases	Y	21/2	23/2	2
1.5	Design GUI Prototype in Balsamiq	Y	21/2	23/2	2
1.6					0
2	Architecture				
2.1	Research Protocol	Y	24/2	25/2	1
2.2	Research Client & Server Architecture	Y	24/2	28/2	4
2.3	Research Database	Y	24/2	28/2	4
2.4	Write Protocol	Y	1/3	2/3	1
2.5	Write 1st Client Demo	Y	3/3	11/3	8
2.6	Write 1st Server Demo	Y	3/3	11/3	8
2.7	Write GUI	Y	3/3	18/3	15
2.8	Integrate Database	Y	3/3	18/3	
2.9	Finalize Client	Y	12/3	18/3	
2.10	Finalize Server	Y	12/3	18/3	
2.11	Populate Database	Y	12/3	18/3	
2.12	Write Test Plan	Y	9/3	11/3	
2.13	Implement Test Plan	Y	12/3	18/3	
2.14	Extra functionalities (such as translation, spell-checking)	N	6/3	18/3	
3	Project Evaluation				
3.1	Combine all parts in a single report	Y	12/3	18/3	6
3.2	Evaluate Project	Y	12/3	18/3	6
3.2.1	Presenation	Y	19/3	20/3	1
3.3.1					0
4	Miscellaneous Tasks				
4.1	Translation	N	6/3	18/3	12
4.2	Encryption	N	6/3	18/3	12

Figure 11: Gantt Chart

Evaluation

EVALUATION	
Achievements	- We have successfully constructed a messenger program that allows client to communicate with other online clients via the server and clients can also access the server with either information from the database or as an anonymous user. - Overall we have established a connection between multiple clients to the server server and the server with the database. Therefore, the fundamental basis of a messenger program (databse-server-client) has been successfully implemented. - In terms of substantial of achievements, we have included users to engage in private chats, group chats, previous chat history viewing, anonymous user login, message translation and user interface personalisation.
Improvements	- Improve the usability of the translation - Improve the security of the system (e.g. API security, add encryption) - Add a change password functionality - Allocate more time for testing - Add additional features such as games in the chat window - Ask for feedback from more users - Allow saving for private chats, not only the public group chat.
Satisfaction with The Program	- We are satisfied we have completed the assignment to achieve the core goals along with a number of accessory features. - Given additional time we feel we could produce a more streamlined product, but since this is the first time any of us have made anything like this we are pleased with how far things have come.
Proportion of the Brief Met	- We have met all the criteria from the specification, and included features which go beyond what was asked.



EVALUATION		
Future changes and reasons for those changes	Changes - Dedicate more time to research the task and better distribute roles based on this. - - Clearer organisation of the SVN. - Improve communication at the very beginning of the project	Reasons - At our initial meetings we struggled to take practical steps since we felt a little lost. Though things eventually fell into place, we could have managed this situation more effectively. - It took time to get used to the system which made using the code on their inconvenient and cluttered. - At the very beginning we worked too independently of one another which led to crossover in what was achieved.
Team progress and achievements	- Maintained a regular schedule of meetings - Though initially communication was a weakness, we quickly became good at ensuring all our team were aware of the progress one another had made. This helped to keep a sense of direction in our work. - Our project developed well over time, we did not end up in a situation where we felt behind schedule or rushed.	

Conclusion

The project provided a different context for coding to what we were used to. Learning to collaborate on the big picture while working individually proved a challenge. On an individual basis, cooperation helped to highlight various strengths and weakness we were not previously aware of and to build on them.

The content of the assignment was outside of our prior areas of knowledge. We were pushed to research on our own terms to improve our understanding beyond the course content, and effectively share this knowledge to keep our teammates up to date. The nature of the task was more demanding than we were used to and was dependent on combining our understanding from multiple modules (where we usually find modules fairly self-contained).

We are pleased with the end result. Our program functions to meet the required specification and goes beyond with a number of features. The work we have done here feels more relevant to real world situations, as our application is much closer to useful software than the more abstract tasks we have previously undertaken.

Appendix

The Appendix includes meeting minutes, test cases, a detailed requirements evaluation, and the organizational Gantt Chart.

Agenda Minutes of Meetings

Chicago Meeting Agendas

Title:	Deciding the idea
Location:	Tutorial LC room
Date:	20.02.18

Participants:	Robert McNally;
	William Nunn;
	Vasileios Manolis;
	Jiayu Liu;
	Minhua Guan;

Start	End	Discussion	Outcomes	Progress
14:00	14:10	· Decide on an idea to take forward and discuss it with the tutor.	Everyone agreed to take forward the Client-Server Messenger Project.	N/A

14:10	14:30	· The team expressed their skills and roles they would like to take up within the project. · Assign someone to be the secretary and one person to be point of contact. · Vasilis, Rob, Jiayu expressed they were more comfortable with coding so will be playing a bigger part in the coding section but everyone in the end will partake in coding, merge, and exchange parts along the way.	Role Assignment: *Vasilis – Database, Point of contact. *Minhua – Database, Secretary. *Rob - GUI *Jiayu - GUI *Will - Threads	N/A
14:30	14:40	· The next step of action	Everyone should look over their parts over the weekend and have a better understanding of what to do next. RESEARCH – homework.	N/A
14:40	14:50	· Next meeting date. · Secretary to write up a plan/timetable of what's been discussed in the meeting and implement everyone's roles and to do list.	02.03.18 after last lecture in Sloman Lounge. (subject to change to a closer date depending how much reading and knowledge gathered over the weekend. Secretary will inform any	N/A

			changes to plans.	

Title: Update on what research or work has been carried out over the weekend

Location: Computer Science meeting room

Date: 27.02.18

Participants: Robert McNally;
William Nunn;
Vasileios Manolis;
Jiayu Liu;
Minhua Guan;

Start	End	Discussion	Outcomes	Progress
10:00	10:20	· Presentation of the work that’s been done over the weekend. · Jiayu and Rob managed to do the GUI for the login stage but both needed a clearer plan of which part of the GUI they should do instead of doing the same part. · Minhua managed to connect to code to the database and written up the functional requirements and introduction.	Jiayu and Rob plan which parts of the GUI they are doing.	Started a bit on the GUI and database of the project.

10:20	10:30	· Will still unsure how threads or sockets is going to implement into the project. · Group still bit unsure about the connection of the different elements from database to server.	*Group suggested Will to focus on Operating System assignment which will be beneficial when he comes to do the thread part. Assigned Vasilis to help him along the way. *Group thought of some questions to ask tutor for better understand of the connection between the different elements.	N/A
10:30	10:40	· Discuss how messages will be stored and sent between users. · Will, Rob and Vasileios discuss about using String buffers and arrays to store and present the messages that will be sent over users. · Group suggested creating separate tables in database for storing messages (chat histories).	*Ask personal tutor for more advice on this topic.	N/A
10:40	10:50	· Next meeting date.	*28th of February to discuss how to take the project further.	N/A

Title:	Meeting with the Tutor
Location:	Tutorial LC room
Date:	27.02.18

Participants:	Robert McNally;
	William Nunn;
	Vasileios Manolis;
	Jiayu Liu;
	Minhua Guan;

Start	End	Discussion	Outcomes	Progress
14:15	14:30	· Explain to the tutor what everyone’s done so far · Asked questions about the project. · Tutor explained what stage we should be at.	· List of questions asked are displayed below the table.	

Tutorial questions

1. How the program will work?

- The user will access the server using their own desktop – this will also be the testing method to see if the program work. TA's will access the server and send messages over 3 computers.

2. Do we have to java build the database?

- Create own table - need to populate (account details, account details, chat histories).
- Want to store chat histories separately, in a different database.
- Password don't need to be secured.

3. Vpn problems?

- Best use the lab computer.

4. Svn access problem?

- Use own CS username and password.

5. What stage should we be at?

- By the meeting of next week – should have a protocol (partial) and have a skeleton of the implementation of sending and receiving the messages.
- Should have a start of the connection of the protocol from client to server and database in place and business logic to implement the database.
- Have a dummy GUI but don't necessarily have to implement everything yet.
- Once have connection – server (server-client -> message exchanged) —> protocol —> then know what to do next.

The protocol needs to state what messages are and what messages are being sent (a description of the responses received).

The login – need a username and password and some sort of response and if login is successful e.g. client ID get sent back.

Messages – need a send page and received response page.

Have the client -server documents in place —> for report.

Have in place before next week's meeting.

Title:	Review what tutor said needs to be done.
Location:	Tutorial LC room
Date:	28.02.18

Participants:	Robert McNally;
	William Nunn;
	Vasileios Manolis;
	Jiayu Liu;
	Minhua Guan;

Start	End	Discussion	Outcomes	Progress
13:30	13:40	· Who is doing the protocol and what the protocol consists of. · Everyone looked over the Operating System(OS) assignment protocol to get a better understanding what a protocol is and how to write it up.	*Went through the OS protocol as a group. *Vasileios and Will try to complete the OS assignment to get a better understanding in writing up the protocol.	

13:40	13:50	· What to include in the database. · Friend list, online/offline activities, storage of chat histories.	*Have separate database tables for storing chat histories and friends. Needed different tables set up for different storage section of the GUI. *Online active – storing in an array list. When someone is active then it is shifted into an array list.	
13:50	14:00	· Storage of the chat histories and revisit of old chats. · Group discussions on whether the user can view previous chat histories via clicking on the client name to initiate a chat (whether on/offline) to view previous chats or a separate tab to view histories (a look into the chat history database).	*Likely to take on the separate tab database histories view depending on timing left to research upon other ways.	
14:10	14:20	· Discuss what work need to be done by next week before meeting tutor. · Recognized that the priority should be client-server connection.	*Set to have the protocol completed, connection to the server from client to server.	
14:20	14:30	· Client-Server connection and how it connects to the database.	* Rob theorized that it needs to send string to server along with some instructions -	

			protocol decode and display the string.	
14:30	14:40	· Profile picture implementation –allow user to add profile picture to their account. · Next meeting.	*Next meeting arranged for the 2nd of March to discuss the connection of the server to clients and weekend targets.	

Title:	Elaboration of the GUI, connecting the GUI with client inputs.
Location:	CS ROOM 245
Date:	02.03.18

Participants:	Robert McNally;
	William Nunn;
	Vasileios Manolis;
	Jiayu Liu:
	Minhua Guan;

Start	End	Discussion	Outcomes	Progress
14:20	14:40	· Go through the layout of the database. · Rob demonstrated on the board what he wants/ what the database should look like.	*4 Tables to be constructed: -Friends -Users -Link (links users and chat logs) -Users -Chat Logs.	
14:40	15:00	· Minhua and Jiayu demonstrated what they've uploaded on svn. · A client-server was uploaded on svn – needed group contribution to make it multithreaded for multiple clients to use. · Need to link the register GUI to the database to store (new) users.	-Assigned as a weekend task to make the client-server multithreaded.	

15:00	15:20	· Discuss the idea of storing the chat histories as each individual text file. · Discuss the idea of storing the histories as an arrayList.	- Neglected the text file method as it could take too much memory and inexperience of coding to make it work.	
15:20	15:40	· Discuss what java classes need to be doing to connect the program. · Will and Vasileios decides to take up the Client-Server connection over the weekend and aim to complete the connection for next week's tutorial.	- Vasileios wrote a list of all the classes needs to be done.	
15:40	16:00	· Will and Vasileios decides to work on the protocol together. · Jiayu and Minhua decides to work on the database connection to the GUI.		
16:00	16:20	· Rob suggested adding a user ID which allow users to be identified and might help with future queries and ease of filtering through the many clients.	-Minhua and Jiayu work together to help code an id column to be continually incremented which each new user.	
16:00	16:20	· Vasileios continuously working/researching on Protocol. · Minhua and Jiayu continue with working/researching		

		on the connection code.		
16:20	17:00		-Jiayu and Minhua managed to connect the database to the GUI and added incremental to the user_id column with each new entry.	

Title:	Attempt to connect server with database.
Location:	Sloman Lounge
Date:	3.03.18

Participants:	Vasileios Manolis;
	Jiayu Liu;
	Minhua Guan;

Start	End	Discussion	Outcomes	Progress
14:00	14:20	· 20 minutes' discussion of how to proceed with the project.	· Vasileios focus on the protocol. · Jiayu and Minhua aims to	· Minhua and Jiayu connected to the database which allows new users to register and then inserted into the database.
14:20	14:40	· Jiayu and Minhua tries to connect the server to the database.	· Jiayu realized that the client input must be transferred into the server then to the database.	· Database must be disconnected from the GUI, and instead connect to the server class.
15:00	16:00	· Jiayu and Minhua decides to research more about the database-server connection and establish a better understanding.	· Research online and watch online tutorials.	
16:30	17:00	· Next meeting arrangement.	· 4.03.18	· Negative progress as database had to be disconnected from the GUI.

Title:	Group coding session to maintain group progress.
Location:	CS ROOM 222
Date:	5.03.18

Participants:	Robert McNally;
	Vasileios Manolis;
	Jiayu Liu;
	Minhua Guan;

Start	End	Discussion	Outcomes	Progress
15:20	15:40	· Minhua demonstrated the GUI she constructed which implements server and client connection. The class also allows many users to talk to each other. · Rob demonstrated the GUI he created where private message is going to be held. Also, demonstrated how the group functionality works.	· Group decides to work around the GUI and decides to work with the java classes and build up around it. · Rob going to improve some parts of the GUI.	· Multithreading implementation where there is multiple user interaction.

15:40	17:30	· Group decides it was better to dedicate an hour of each day where the group comes together and code. This is to prevent duplicate work being done. · Jiayu attempts to connect the message sent from the server to the database. · 16:00 – 17:30: Everyone do their own research and work on their codes.	· Future planning of meeting where everyone is available to code together and discuss any problems that arises.	
17:30	17:50	· Vasileios grouped some questions from the team to ask Cory tomorrow. · Arranged meeting after tutorial meeting.	· Group meeting tomorrow after tutorial.	· Multithreading communication in place. · Dummy GUI to show Cory tomorrow.

Title:	Meeting with Tutor
Location:	LC tutorial room
Date:	6.03.18

Participants:	Robert McNally;
	William Nunn;
	Vasileios Manolis;
	Jiayu Liu;
	Minhua Guan;

Start	End	Discussion	Outcomes	Progress

14:20	15:00	· Meeting finish with Cory.	· Follow his advice for future project planning and decides on the next stage. · List of questions asked are displayed below the table.	
15:00	16:00	· Rob demonstrates that he implemented protocol within the server class. · Group decides to meet again at 17:00 to have a short meeting on how to proceeds with the project.		· The protocol is implemented into the program.
17:00	17:50	· Vasileios suggested to import a translation function into the GUI. · Minhua and Jiayu decides to work on connecting the server with the database (store registered user into the database). · Will was arranged to do the UML diagrams for the report. · Rob and Minhua sent their functional and non-functional requirements to Vasileios to compile.	· Managed to connect registered user into the database via client à server à database.	· The database is connected to server and store user registration information.

<u>Tutorial questions</u>

1. **Are we allowed to use the OS protocol?**

    · Yes as long as relates to our project and that the protocol changes according to the communication between the necessary classes.

2. **How to send chat messages to the database?**

    · There should be a new thread for each new client when client connects.
    · Then when messages are being sent, it is stored as an object in the arrayList.
    · Then send the object to the database where it is split into parts.

3. **What sorts of content needs to be added into the report?**

    · Include information about the protocol and the database design. In the report, have to document the user flow. In addition, includes testing arrangement to clarify the program is working as it was coded to.

4. **What is a test plan?**

    · Events that will occur when something is inflicted.

5. **What does the substantial of achievement section of the project imply?**

    · Having the baseline of database-server-client and implemented extra features: encryption, sending files, sending emoji, typing notification.

Title:	Storing chat log histories methodology.
Location:	CS ROOM 245.
Date:	9.03.18

Participants:	Robert McNally;
	William Nunn;
	Vasileios Manolis;
	Jiayu Liu;
	Minhua Guan;

Start	End	Discussion	Outcomes	Progress
15:20	15:40	· Jiayu demonstrated her server GUIs with connection port entry and IP address. · Rob demonstrated the newly improved private chat GUI.		
15:40	16:00	· Minhua and Jiayu tries to connect the private chat GUIs with the public chat GUIs. · Vasileios and Will work on the report.		
16:00	16:30	· Group discussed how chats will be stored in the database.	· Further research into how to store chat log histories in the database.	

16:30	17:50	· Meeting arrangement for Sunday.	· Next meeting 11.03.18.	· Managed to merge private chat with the public chat. · There will still some bugs/errors with the server thread class after merging private and public chats.

Title:	Preparation for tomorrow tutorial meeting (question list construction).
Location:	Sloman Lounge
Date:	11.03.18

Participants:	Robert McNally;
	Vasileios Manolis;
	Jiayu Liu;
	Minhua Guan;

Start	End	Discussion	Outcomes	Progress
14:00	15:00	· Group realized client list does not update when a user leaves the chat. · Rob decides to figure out the bug during this meeting. · Vasileios, Minhua, Jiayu discuss ways to store group message.	· Attempted many methods to store the messages exchanged between different users but failed. · Jiayu found a way to store public chat messages.	
15:30	16:00	· Constructed a list of questions to ask Cory in the tutorial. · Arrange meeting after Tuesday tutorial.		· So far have established the fundamental connection of database-server-client communication.

Title:	Meeting with Tutor
Location:	CS ROOM 245.
Date:	13.03.18

Participants:	Robert McNally;
	William Nunn;
	Vasileios Manolis;
	Jiayu Liu;
	Minhua Guan;

Start	End	Discussion	Outcomes	Progress
14:20	15:00	· Meeting with tutor. · Demonstrated what the client-server messenger looks like so far. · Ask what is the next stage of the messenger program.	· Tutor wanted the group to add in additional features e.g. translation.	· Increase extra features in the messenger, e.g. translation, encryption.
15:00	15:30	· Vasileios decides it was better to start focusing more on the report during this meeting. · Written up a plan of what needs to include in the report.	· Arranged who's writing up which parts in the report.	
15:30	16:00	· Jiayu and Vasileios discuss the test plans and thought about what to include. · Group decided it was better for Rob to focus on the coding		

		aspect to fix some bugs in the private chat and the update list.		
16:00	16:30	· Vasileios discussed adding the translation feature into the messenger. · Arranged next meeting.	15.03.18	

Title:	Report distribution.
Location:	LC tutorial room.
Date:	15.03.18

Participants:	Robert McNally;
	Vasileios Manolis;
	Jiayu Liu;
	Minhua Guan;

Start	End	Discussion	Outcomes	Progress
13:20	13:40	· Rob wants to add some changes to some GUI features. Jiayu and Minhua decided it was better to compile all their parts then send it to Rob to make the final changes.	· Added in the translation feature.	
13:40	14:00	· Vasileios decides he was going to focus on the report in this meeting. · Rob will await the most updated compilation before implementing his changes.		
14:00	14:30	· Vasileios distributed more parts of the report for the group to start writing up and build up the report.	· Group decides the focus for the remaining days before deadline is to focus more on the report.	

14:30	14:50	· Arranged next meeting.	18.03.18	

Title:	Finalization of the report
Location:	UG LAB
Date:	18.03.18

Participants:	Robert McNally;
	William Nunn;
	Vasileios Manolis;
	Jiayu Liu;
	Minhua Guan;

Start	End	Discussion	Outcomes	Progress
14:00	14:40	· The group decided to test the program on the lab machine. 2 client logged in and one server opened (as instructed from the worksheet) · The group then detected some bugs and analyzed which one would be easy to fix within the time frame of the deadline.	· Some bugs were fixed but others are to be reported for future improvements.	· Established that the program works on the lab machines.
14:40	15:20	· Jiayu and Vasileios decides to do the test plan and the rest of the team are to focus on their dedicated sections of the report.	· Everyone agreed that the finalization of the report is to be today.	
15:20	N/A	· Team decides to work until the project report is compiled and ready for submission.		Report submission.

TEST PLAN - TEST CASES

ID	Asset	Feature	Env.	Type	Test Case Description	User Testing Passed [Y/N]	Comments	Server Message	Client Message	Server-Client Message Approved
1	Server	Booting	Linux	Functionality	Verify that the moderator is able to open the server	Passed		Waiting for clients...		Passed
2	Server-Client	Booting	Linux	Functionality	Verify that user is able to connect to the server using the requested Port and IP Address	Passed	If server is not found, a warning message appears to the user.	Welcome to the server!	receive: send-message:__:Welcome to the server! Welcome to the server!	Passed
3	Server-Client-Database	User sign-up	Linux	Functionality	Verify that user is able to create an account.	Passed	If username and password has the correct lenght it creates a new account, if not it shows a warning message to the user. If username is already exists, send a "false" response to client	send: sign-up:__username:__;username receive: sign-up:__;true:__; sign up successfull receive: sign-up:__;false:__; Username already exists	send: sign-up:__; username:__; password receive: sign-up:__; true:__;sign up succesfull receive: sign-up:__; false:__;Username already exists	Passed
4	Database	User sign-up	Linux	Functionality	Verify that the created account populates the Database successfully.	Passed	The database was checked in real time to ensure that is updated properly.			
5	Server-Client	User log-in	Linux	Functionality	Verify that user is able to login anonymously.	Passed	User is allowed to sign in directly, without entering username and password.	receive: sign-in:__;anon:__; anon:__;true send: sign-in:__;true:__; anon0:__;chat history	send: sign-in:__;anon __;anon:__;true receive: sign-in:__; true:__;anon1:__;chat History Sign in successful.	Passed
6	Client-Server-Database	User log-in	Linux	Functionality	Verify that a user is not able to login with the same username from two different machines.	Passed	User is not able to connect from two machines with the same password. A warning message tat the username/password combo is incorrect appears.	receive: sign-in:__;natalie:__; 15601560:__;false trying to sign in normally send: sign-in:__;false:__;user already signed in!	send: sign-in:__;natalie __;15601560:__;false receive: sign-in:__; false:__;user already signed in! User already signed in	Passed
7	Client	User log-in	Linux	Functionality	Verify that more than two users can log-in and Chat.	Passed				
8	Server	User log-in	Linux	Functionality	Verify that the Currently Online Lists shows all the users currently online and is updated when a new user log-in	Passed		send: update-list:__;add-user:__;Lucky7	receive: update-list:__; add-user:__;Lucky7	Passed
9	Client	Translation	Linux	Functionality	Verify how many number of words or characters can be translated in a time.	Passed	The translation API can take up to 2500 characters at a time and translate text up to 100times/hour.			
10	Client	Translation	Linux	Functionality	Verify that the translation API translates text input.	Pending	Needs improvement: Currently, the language input and output should be written exactly in a specific format (e.g. "English" not "english"). Apart from that, when there is an error translation a shorter warning message should be added instead of the long string that currently appears in the translation box.	No message, because it uses API.	2222222222English 3333333333Spanish 4444444hey friends! what's up? Status Code: 200 ¡Hola amigos! ¿que pasa?	Passed
11	Client	Chats	Linux	Functionality	Verify how many number characters can be send at a time.	Pending	Needs improvement: There was no limit in the number of characters that are sent, however, the client will show only the first 54 characters of the concatenated string of Username+TimeStamp+Message. The GUI needs an additional horizontal scroll pane.			
12	Client	Chats	Linux	Functionality	Verify that user is able to see the time of comment in Chat.	Pending	Needs improvement: The timestamp does not show the correct value of minutes.			
13	Client	Chats	Linux	Functionality	Verify that client is able to recognize special characters and characters in other languages (e.g. Chinese, Greek) in Chat.	Pending	Only latin characters are recognized at the moment.			
14	Client	Chats	Linux	Functionality	Verify that user is able to choose the color of their preference.	Passed	User may click choose color button and chose what color they want for the background.			
15	Client - Server	Chats	Linux	Functionality	Verify that user is able to create a new private chat.	Passed	User is able to create a private chat and also have a name for their private chat room.	receive: create-group:__; chicago!!!:__;Ken:__;anon0 makeChat got called! send: create-group:__;1:__; chicago!!!:__;Ken:__;anon0	send: create-group:__; chicago!!!:__;Ken:__; anon0 receive: create-group __;1:__;chicago!!!:__; Ken:__;anon0	Passed
16	Client - Server	Chats	Linux	Functionality	Verify that the private chat works for all the clients and the messages appear in real time for all the users.	Passed				
17	Client - Server	Chats	Linux	Functionality	Verify that users are able to create a Chat group or not.	Passed	Users are able to create a chat group and also add new group member after created. And users can create a group name for their group chat room	receive: create-group:__; testtest3's chat Room:__; testtest3:__;anon0:__;Ken makeChat got called! send: create-group:__;3:__; testtest3's chat Room:__; testtest3:__;anon0:__;Ken	send: create-group:__; testtest3's chat Room __;testtest3:__;anon0 __;Ken receive: create-group: __;3:__;testtest3's chat Room:__;testtest3:__; anon0:__;Ken	Passed
18	Client - Server	Chats	Linux	Functionality	Verify that users are able to Chat with their group or not.	Passed	Users can chat with their group and group members will receive the message as well	receive: send-private-message:__;3:__;hello world send: send-private-message: __;3:__;testtest3:__;16:03: 00:__;hello world	send: send-private-message:__;3:__;hello world receive: send-private-message:__;3:__; testtest3:__;16:03:00: __;hello world	Passed

TEST PLAN - TEST CASES

ID	Asset	Feature	Env.	Type	Test Case Description	User Testing Passed (Y/N)	Comments	Server Message	Client Message	Server-Client Message Approval
19	Client - Server	Chats Logs	Linux	Functionality	Verify that users are able to see the previous messages in private Group Chats.	Passed	User can see the current chat in the message box. And the current chat will seperate with chat history by "-------HISTORY-------"			
20	Client - Server - Database	Chats Logs	Linux	Functionality	Verify that users are able to see the full Chat history of the public chat room.	Passed	After sign in, user is allowed to see all chat history in the public chat room.			
21	Client - Server	Chats	Linux	Functionality	Verify that multiple users can chat in private groups simultaneously and every Chat should be visible to every member of the group.	Passed				
22	Client - Server	Chats	Linux	Functionality	Verify that name of the User should be displaying to others in Chat application while chatting with other Users or group.	Passed				
23	Client - Server	Chats	Linux	Functionality	Verify that whenever any member join or leave the Chat its occasion then it should be notified in the group and is visible to every member of the group.	Pending	The user list will update while group member join or leave. In rare cases, the list wasn't updated immediately.			
24		Chats	Linux	Functionality	Verify that user is able to send emojis.	Pending	Users are not able to send emojis.			
25	Client - Server	Chats	Linux	Functionality	Verify that user is able to use copy/paste in Chat.	Passed	Users can copy/paste their messages and the messages of other users.			
26	Client - Server	Chats	Linux	Functionality	Verify how much time does it take to send a message in the common Chat room.	Passed	Less than 1 second.			
27	Client - Server	Chats	Linux	Functionality	Verify how much time is it taking to send a message in a group.	Passed	Less than 1 second.			
28	Client - Server		Linux	Functionality	Verify what happens if the connection between the client and the server breaks.	Pending	Needs improvement: If the server closes abruptly, there is no warning message to the Client.		java.net. SocketException: Broken pipe (Write failed)	Pending
29	Client - Server	User log-out	Linux	Functionality	Verify that users can close the Chat and get logged out.	Passed	User is allowed to log out through disconnect to the server.	receive: disconnect:__; testtest3 send: update-list:__remove-user:__testtest3	send: disconnect:__; testtest3 "you have disconnected!"	Passed
30	Client - Server	Chats	Linux	Functionality	Verify that users can continue talking in a Group Chat even if a users disconnects from the Chat.	Pending	After user leave the group, if someone invite this user again, this user can also receive messages that sent after he left.			
31	Client - Server	Chats	Linux	Functionality	Verify the number of users that can participate in a chat.	Passed	No limit has been noticed.			
32	Client - Server - Database	Chats Logs	Linux	Functionality	Verify that messages are ordered chronologically.	Passed	The messages are sort in chronological order.			
33	Database	Sign-up/log-in		JUnit	isExistUser, isValidUser	Passed				
34	Class Message	Class Message		JUnit	Methods of Class Message	Passed				
35	Class Chat	Class Chat		JUnit	Methods of Class Chat	Passed				
36	Class User	Class User		JUnit	Methods of Class User	Passed				
37	Class Translator	Class Translator		JUnit	Methods of Class Translator (translate, findLanguageCode)	Passed				

Requirements Evaluation

REQUIREMENTS EVALUATION			
Category	**Requirement**	**Accomplished**	**Comment**
Sign-up	The system must allow the user to select a username between 5-20 characters and a password between 8-32 characters. - High	Y	
Sign-up	The system must inform the user if the username or password does not fulfil the length criteria. - High	Y	
Sign-up	The system must inform the user if the username has been taken by a registered user. - High	Y	
Sign-up	The system must inform the user if the server cannot be found. - High	Y	
Sign-up	The system must show a confirmation message when their registration is successful. - High	Y	
Sign-up	The system must allow the user to select the IP and the port they wish to connect. - Medium	Y	
Log-in	The system must allow the user to enter in their username. - High	Y	
Log-in	The system must allow the user to enter in their password. - High	Y	
Log-in	The system must inform the user if the username does not exist. - High	Y	
Log-in	The system must inform the user if the password does not match the username. - High	Y	
Log-in	The system must allow the user enter the chat after correctly entering their username and password tapping on the “Login” button. - High	Y	
Log-in	The system should allow users to enter the chat anonymously. - Low	Y	
View a List of Online Users	The system must allow the user to view a list of all the users who are currently online. - High	Y	
View a List of Online Users	The system must allow the user to view a list of other the users participating in each Chat room. - Medium	Y	
View a List of Online Users	The system should allow the user to view a list of other users who are currently offline - Low	Y	
View a List of Online Users	The system should allow the user to view a list of other users who are busy. - Low	N	Due to time constraints, decided not to implement this feature.
View a List of Online Users	The system should inform the user that they are online and active when they've logged - Low	Y	
Chats	The system must have a Chat Room with all the current online users. - High	Y	

Chats	The system must allow the user to view the names of other online users. - High	Y	
Chats	The system must allow to initiate a private Group chat. - High	Y	
Chats	The system must be able to process and accept keyboard inputs from the user. - High	Y	
Chats	The system must allow user to send message via means of clicking “Send”. - High	Y	
Chats	The system must indicate the sender of the message. - Medium	Y	
Chats	The system must indicate the time the message was sent. - Medium	Y / To Be Improved	There is a minor bug with the timestamp.
Chats	The system must allow the user to receive messages from the interlocutor. - High	Y	
Chats	The system must allow the user to edit their message anytime they wish to before sending. - High	Y	
Chats	The system must allow user to send message via means of clicking the “Enter” key - Low	N	Due to time constraints, decided not to implement this feature.
Chats	The system must let the user to delete anyone they no longer wish to be in the new group. - Low	N	Due to time constraints, decided not to implement this feature.
Invite More Users	The system must allow the user to view all the participants in the existing group chat. - Medium	Y / To Be Improved	Minor sporadic bugs needs to be improved.
Invite More Users	The system must allow the user to invite/add new participants from their contact list. - Low	Y	
Invite More Users	The system must all allow the user to remove existing participants from their contact list. - Low	N	Due to time constraints, decided not to implement this feature.
Leave Group	The system must allow the user to view all the group chats they are involved in. - High	Y	
Leave Group	The system must allow the user to leave any group they wish. - Low	Y	
Leave Group	The system must inform the user they have left the group. - Low	N	Due to time constraints, decided not to implement this feature. However, the list is updated.
Leave Group	The system must inform the user they cannot converse in group(s) they have left. - Low	N	Due to time constraints, decided not to implement this feature. However, the list is updated.
Chat Logs	The system must allow the user to view their chat history. - Medium	Y / To Be Improved	History in private chat too should be added.
Chat Logs	The system must allow the user to click on anyone in their contact list and see their chat history. - Medium	N	Due to time constraints, decided not to implement this feature.
Advanced Features	The system should underline misspelled English words while a user is typing. - Low	N	Due to time constraints, decided not to implement this feature.
Advanced Features	The system should allow the users to translate input. - Low	Y	

Advanced Features	The system should allow the user to change the colour of the Chat windows according user preference. - Low	Y	
Advanced Features	The system should provide encryption tools. - Low	N	Due to time constraints, decided not to implement this feature.
Performance	The system must send messages to the recipients within 2 seconds of sending. - High	Y	
Performance	Actions such as logging on, registering, opening a chat window should be accomplished in under 5 seconds - Medium	Y	
Survivability	The system should be easily restartable in the case of a crash. - Medium	Y	
Survivability	The system should not completely crash in the case of small errors (e.g. users failing to connect/open a chat windows with each other/dropping out mid conversation). - High	Y / To Be Improved	
Maintainability	Any error should be easily locatable and fixable due to simple architecture and defensive programs. - High	Y / To Be Improved	
Security	The system must securely store the data collected, preventing third parties to access it - Medium	Y / To Be Improved	
Security	No user should be able to access the private chat logs and the password of any other user - Medium	Y / To Be Improved	
Security	No user should be able to edit the database other than to register an account. - Medium	Y	
Usability	It should be clear on any given interface what the users' options are for interacting with the software. - High	Y	
Usability	The system must take less than 5 minutes for an average tech-savvy user to understand it. - Medium	Y	
Usability	The system must have a minimal uncluttered user interface and a pleasant design. - Medium	Y	
Data Integrity & Retention	The database should reliably store chat histories and user profiles (e.g. correctly rejecting register attempts with a taken username - High	Y	
Data Integrity & Retention	The system should maintain all chat histories even between instances of the system going down and back online. - Medium	Y	
Testability	The system should be delivered with a test plan. - High	Y	
Testability	The system should support testing of each component separate from any other component. - Medium	Y	
Scalability	The system must be able to support the addition	Y	

	of message encryption. - Low		
Scalability	The system must be able to support the addition of translation via connecting to an external API. - Low	Y	
Scalability	The system should support the addition of further functionality (e.g. games in the chat windows) - Low	Y / To Be Improved	
Scalability	The system should be able to be adapted to an increased load of simultaneously active clients. - Low	Y	
Availability	The system should be open to any connection whilst it is operational. - Medium	Y	

Gantt Chart

						PHASE ONE																					PHASE TWO											
						WEEK 1							WEEK 2							WEEK 3							WEEK 7							WEEK 8				
	Task - Proccess Tracking	On-Track?	START DATE	DUE DATE	DURATION	19	20	21	22	23	24	25	26	27	28	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
						M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F
1	Project Conception and Initiation																																					
1.1	Decide and describe the proposed system	Y	17/2	18/2	1	🟦																																
1.2	Requirements Analysis: Functional and non-functional requirements	Y	19/2	23/2	4	🟦	🟦	🟦	🟦	🟦																												
1.3	UML Diagrams	Y	21/2	23/2	2			🟦	🟦	🟦																												
1.4	Use Cases	Y	21/2	23/2	2			🟦	🟦	🟦																												
1.5	Design GUI Prototype in Balsamiq	Y	21/2	23/2	2			🟦	🟦	🟦																												
1.6					0																																	
2	Architecture																																					
2.1	Research Protocol	Y	24/2	25/2	1						🟦	🟦																										
2.2	Research Client & Server Architecture	Y	24/2	28/2	4						🟦	🟦	🟦	🟦	🟦																							
2.3	Research Database	Y	24/2	28/2	4						🟦	🟦	🟦	🟦	🟦																							
2.4	Write Protocol	Y	1/3	2/3	1											🟦	🟦																					
2.5	Write 1st Client Demo	Y	3/3	11/3	8													🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦												
2.6	Write 1st Server Demo	Y	3/3	11/3	8													🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦												
2.7	Write GUI	Y	3/3	18/3	15													🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	▨	▨	▨	▨	▨	▨	▨					
2.8	Integrate Database	Y	3/3	18/3														🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	▨	▨	▨	▨	▨	▨	▨					
2.9	Finalize Client	Y	12/3	18/3																							▨	▨	▨	▨	▨	▨	▨					
2.10	Finalize Server	Y	12/3	18/3																							▨	▨	▨	▨	▨	▨	▨					
2.11	Populate Database	Y	12/3	18/3																							▨	▨	▨	▨	▨	▨	▨					
2.12	Write Test Plan	Y	9/3	11/3																			🟦	🟦	🟦	🟦												
2.13	Implement Test Plan	Y	12/3	18/3																							▨	▨	▨	▨	▨	▨	▨					
2.14	Extra functionalities (such as translation, spell-checking)	N	6/3	18/3																	🟦	🟦	🟦	🟦	🟦	🟦	▨	▨	▨	▨	▨	▨	▨					
3	Project Evaluation																																					
3.1	Combine all parts in a single report	Y	12/3	18/3	6																						▨	▨	▨	▨	▨	▨	▨					
3.2	Evaluate Project	Y	12/3	18/3	6																						▨	▨	▨	▨	▨	▨	▨					
3.2.1	Presenation	Y	19/3	20/3	1																													▨	▨			
3.3.1					0																																	
4	Miscellaneous Tasks																																					
4.1	Translation	N	6/3	18/3	12																🟦	🟦	🟦	🟦	🟦	🟦	▨	▨	▨	▨	▨	▨	▨					
4.2	Encryption	N	6/3	18/3	12																🟦	🟦	🟦	🟦	🟦	🟦	▨	▨	▨	▨	▨	▨	▨					

References

1. Bahsoon, R (2010) Software Engineering Part II Lecture Notes, University of Birmingham
2. Booch, G. et al. (2000) Using UML: Software Engineering with Objects and Components, Addison Wesley
3. Pilone, D. & Pitman, N. (2005) UML 2.0 In a Nutshell, O`Reilly Media
4. Sommerville, I. (2011) Software Engineering ed. 9, Pearson

Statement of Contribution

All team members have contributed according to their strengths, weaknesses and interests. The team communicated daily, held meetings regularly for discussion, planning and actual coding, and had informal team working sessions when possible. Initially, Robert and Jiayu researched about the GUI, Minhu the Database, and William with Vasilis researched the Client-Server communication. As the project evolved there were several rotational roles and all team members had coding sessions in all aspects of the projects: Server-Client communication, Database and GUI. As the project progressed, Robert developed a specialization in the Server-Client communication, Minhua in the Database, Jiayu in the GUI and Vasilis paired with William to research about the additional features such as the translation and encryption, and also started preparing the report. The product was tested by the entire team and the final version of the report was co-edited by all team members allowing for input from each member on each section.