

---

# Small Team Project

## Client-Server Messenger Project

Team Frankfurt - 21 March 2016

---



Krisztina Ildikó Nógrádi: 25%

Dominic Parfitt: 25%

Natalie Reynolds: 25%

Bill Tian: 25%

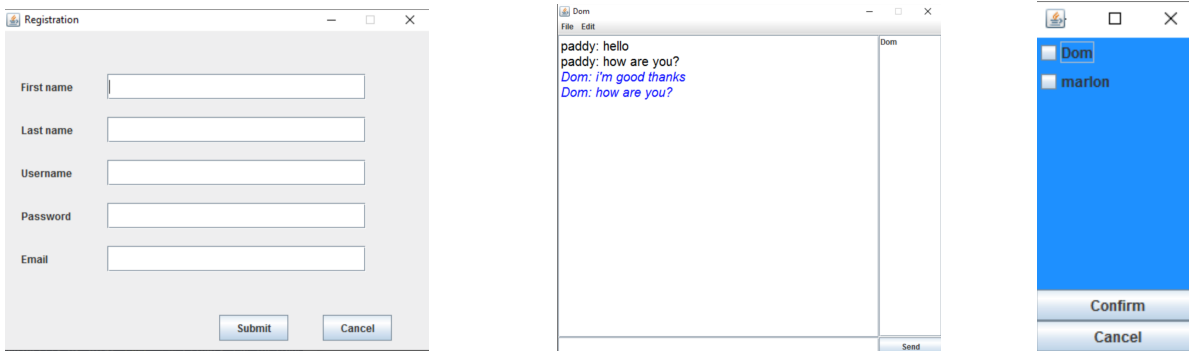
---

Small Team Project	1
1. Brief description of the system	3
2. System Design	5
3. Test Plan	10
4. Team organisation	13
5. Project Diary	13
6. Evaluation	14
7. Bibliographic References	15
8. Appendices	16

---

# 1. Brief description of the system

The messenger system has three main features: registering a new user, send/receive chat messages and creating group chats. These form separate windows in the navigation.



From an architectural point of view, the application can be separated into four major elements:

1. The **database** to store user information and chat history
2. The **server** to handle communication with the clients and the database
3. The **client** to communicate with the server
4. The **GUI** to present a user-friendly form of navigation

There is also a protocol established to create an explicit form of communication between the client and the server. These are explained in detail in the following sections of the report. Firstly, the specification is reviewed to introduce expectations towards the system.

## Specification

The user should be able to:

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

---

## Project Requirements

The following requirements were derived from the specification

### Core Requirements

- R.1. The system should allow the user to register
- R.2. The system should allow the user to log in once they have registered
- R.3. The system should validate the provided user credentials with those that were provided upon registration
- R.4. The system should store encrypted user passwords
- R.5. The system should encrypt user passwords before transmitting them along a network
- R.6. The system should provide a mechanism to allow users to log in when they have forgotten their password
- R.7. The system should allow users to send messages to other individual users

### Additional Requirements

- R. 8. The system should provide functionality for group chats consisting of three or more users
- R.9. The system should only allow users to send messages to other users who are listed as friends / contacts
- R.10. The system should display whether a user is online or not to other users
- R.11. The system should allow the user to access their chat history
- R.12. The system could allow two users to play a simple game using the chat system

---

## 2. System Design

The different architectural elements within the system are introduced in the following sections. Firstly, the protocol is explained in detail in order to elaborate on the communication elements.

### The protocol

The protocol established between the client and the server is determined in the **Message** class within the `chatComponents` package. In its constructors, there are different contracts created for different instances of the **Message** class. Firstly, each message has a type that informs the server of the request type and prompts appropriate response. These are the fields of the **Message** class and are listed as `final` integers. There are eight types of messages that can be sent between the client and the server, as follows: register, login, forgotten password, chat, logout, error, add new contact and online contacts.

Depending on the type of request being sent, **Message** objects are wrapped up with additional information. These are created by the client.

### The client

The client is responsible for communicating user requests to the server and informing the user of the response sent by the server. This suggests a two-way communication: the client wraps up messages that are sent to the server and unwraps the messages sent by the server. Each message created by the client receives a type as explained in the protocol section; moreover, each will have a time stamp and a textfield with information depending on the type of message that is being sent. To interpret messages received from the server, the client first checks the type of the message and takes appropriate action in response.

### Sending

Depending on the user request, the client may send seven types of **Message** objects to the server.

1. **A registration request**, including first name, last name, username, password and email address of the user. The password is encrypted using the `passwordEncryption` method, which uses `MessageDigest` and the SHA-256 algorithm to encrypt passwords. Hence, passwords are never passed along or stored as plain text. The String elements are concatenated with commas, which are used to indicate the end of

---

an element so that the String can be separated again later to meaningful substrings. The wrapped-up message is then sent to the server.

2. **A login request**, which simply communicates the username and the password encrypted similarly to the registration request. The String elements are concatenated with commas again. The message is wrapped up with its type, the current time and the String elements, and is sent to the server.

3. **A forgotten password request**, which concatenates the username and email address entered by the user, and the wrapped-up message is sent to the server.

4. **A chat request**. In this case, the **Message** object is wrapped with the type, time stamp, the intended recipient of the message, the sender of the message and the text message itself. The **Message** object is then sent to the server.

5. **A logout request**, which simply wraps up the type, the time stamp and a non-meaningful String to inform the server that the user wishes to be logged out.

6. **An add contact request**, which includes the type, the time stamp and the username of the user to be added to the contact list. The object is sent to the server.

## Receiving

Depending on the response from the server, the client may receive seven types of messages from the server. The type of the different message is determined as set out in the protocol.

1. **A registration response**, which updates the model to reflect successful registration.

2. **A login response**, which updates the model to reflect successful login.

3. **A forgotten password response**, which updates the model to log in.

4. **A chat response**, where the client goes to display the incoming message.

5. **An error message**, which is displayed to the user.

6. **An online user message**, which is displayed in the contact list.

## **The server**

The server is responsible for receiving requests from the client and passing it on to another client or to the database. Similarly to the client, it unwraps the **Message** object received from the client and checks its type. It then goes on to fulfil the request and wrap up a **Message** object as a response to the client.

## Sending - Receiving

---

Depending on the user request delivered by the client, the server may receive several types of Message objects from the client or may receive information from the database.

1. **Registration request**, which directs user information towards the database.
2. **A log in request**, which prompts checking credentials with the database.
3. **A forgotten password request**, which logs the user in.
4. **A chat message request**, which is delivered to the recipient.
5. **A logout request**, which is logs the user on the given client off the system.
6. **An add contact request**, which is directed at the database.
7. A chat history request, which is not fully implemented yet.

## The database

The database is responsible for receiving information from the server and storing it appropriately. The database includes the following tables:

1. **Users**: contains the information related to users, including first name, last name, username, encrypted password and email address. Each user is referenced via their unique id.
2. **Groups**: includes the group id of each chat that has been initialised and the username of participating users; a chat between two people is implemented as a group chat of two. Group member is a foreign key, which references user id in the Users table.
3. **Group message**: stores the messages sent between users along with the timestamp, sender and recipient(s)
4. **User contacts**: consists of each registered user and the users they are connected, i.e. their friends/contacts.

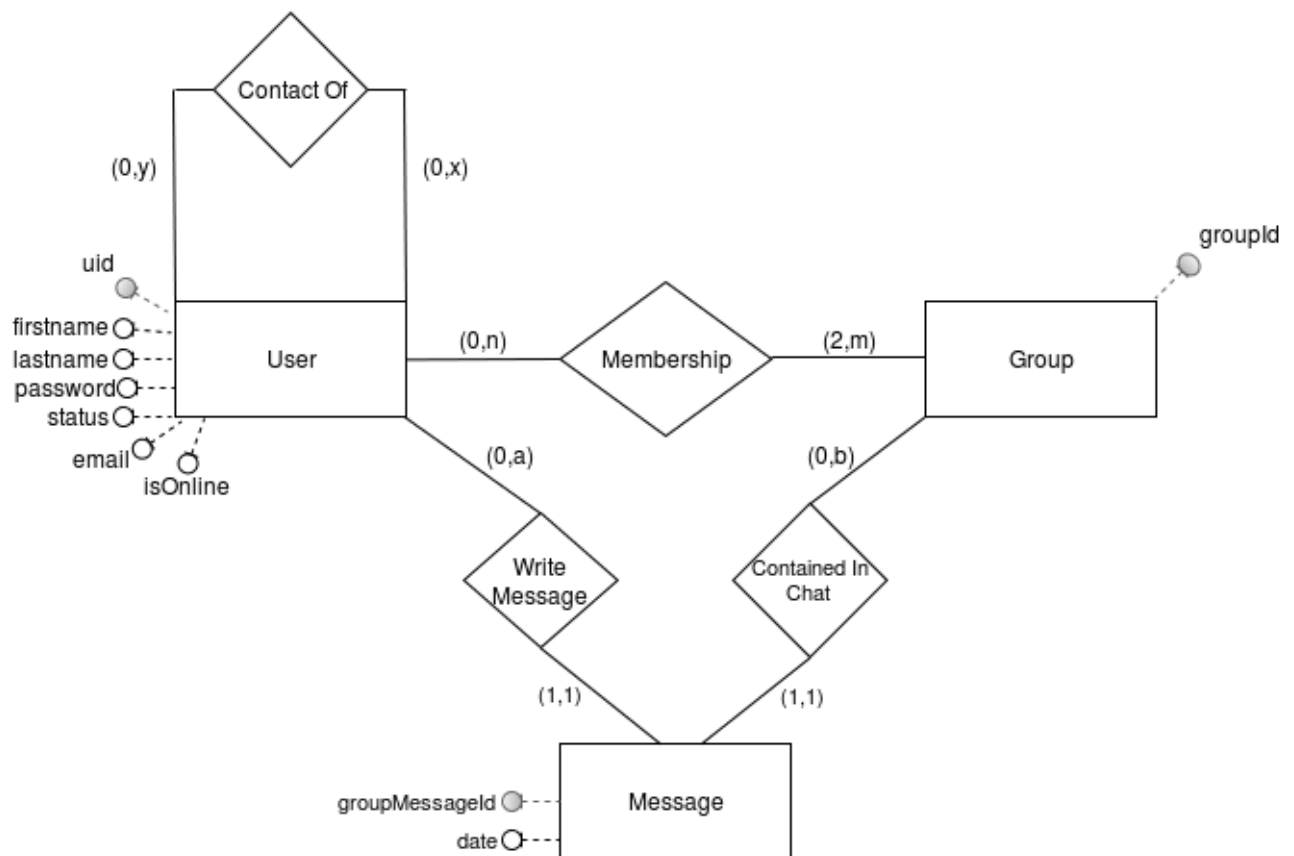


Figure 1: E-R Model

## The GUI

The GUI ensures user-friendly communication with the rest of the components. It uses a number of models and views to inform the user of the client/server/database activities:

1. **Login view:** includes textfields, username and password, clicking the login button takes the user to the list of contacts, a button for registration and a button for forgotten password (Appendix 1).
2. **Registration view:** includes textfields for first name, last name, username, password and email address. The submit button sends the registration to the server, while the cancel button redirects the user to the login page without registration (Appendix 2).



---

3. **Forgotten password view:** includes textfields for the username and email address. The submit button sends the login request to the server, while the cancel button redirects the user to the main page.

4. **Contacts view:** includes list of contacts, a button for adding a new contact, logging out and group chats. Clicking a contact name initiates new chat (Appendix 3).

5. **Add friends view:** includes a textfield for the username, the add button sends the request to the server

6. **Chat view:** includes the chat window that is updated by the client model and chat model (Appendix 4).

7. **Start group chat view:** includes a list of contacts, the confirm button starts the group chat, while the cancel button takes the user back to the list of contacts (Appendix 5).

8. **Chat model:** updates the chat view with new messages

9. **Client model:** updates the chat view with new messages

---

## 3. Test Plan

### Objective

The objective of this test plan is to lay out the testing strategies that will be used to test the different components of the Chat Server system. Where applicable, it will also include the expected results from each test.

### Scope/Coverage

The Database, Server, Client and any additional components (e.g. Messages, Users) will be tested in isolation to ensure there are no bugs. This will consist of unit testing the individual methods of each class within the component using a Black-Box and/or a Glass-Box where applicable.

Once each component has been thoroughly tested in isolation then integration testing will be performed to ensure that the components interact with each other as expected. This testing will cover the three links between the components: Database-Server, Server-Client, Client-GUI. This will conclude with a full system test as some processes, such as logging in, will require communications to be passed along multiple links and back. Finally, the system will be performance tested to ensure that it meets certain performance requirements. Most important of these will be CPU usage and time lags, as it is important for usability that both of these are kept as low as possible.

### Unit Testing

In order to ensure that the system works as expected, each method in every class has to be tested using JUnit.

### Integration Testing

#### Database-Server

Testing will need to be performed to ensure that the Server is able to correctly access entries in the database as well as inserting new entries.

#### Server-Client

---

Integration between the Server and the Client components is of high importance. Testing will assert that communication between the two components is ongoing and not interrupted.

### Client-GUI

Testing will need to be performed in order to ensure that the client correctly updates the GUI of the changes made.

## **Performance Testing**

**Resource Usage:** Testing will need to be performed to ensure that the multiple threads used by the server to connect each client are not too resource hungry. To do this multiple tests will be run with an increasing number of clients connected and the CPU usage of the server will be measured whilst clients are both active (sending messages to the server) and idle.

**Time Lags:** Testing will need to be performed to ensure that there is not a significant delay between the client sending a message and the appropriate action being taken by the server. This is most important in the case of two or more clients chatting to one another, where time delays should be at an absolute minimum. This will be the main focus of testing, with the time taken to send and receive messages monitored with an increasing number of clients connected to the system and also with different numbers of clients connected to a single chat (group chat). The secondary focus will be ensuring minimal time lags for messages sent by the client that require the server to access the database (for example, logging in, registering, etc.). The time delays here should also be at a minimum but there is more leeway for higher delays given the nature of the process. This will be tested with a single client attempting to log in with a varying number of other clients already connected. If possible it may be necessary to test the timings of multiple clients attempting to log in simultaneously.

**Network Connections:** Testing will need to be performed to ensure that if the connection is lost then it is handled in an appropriate manner. This could be from either the Client or the Server unexpectedly terminating or from an external loss of connection.

**GUI Testing:** As the GUI cannot be easily unit tested in the normal manner it is important that all the functions of each view are tested by hand to ensure that the correct action is performed.

---

**General Usage:** General usage testing should be performed to ensure that the system runs as would be expected by an average user. This should include ensuring that there are no large delays performing actions, such as opening a chat window, and also that the Client GUI is laid out in such a way to make it easy for the user to navigate.

---

## 4. Team organisation

It was agreed that each person would be responsible for coding one main component of the project, while team members would aid each other when necessary.

Members	Responsibilities	
Dom Parfitt	Server, team secretary	
Natalie Reynolds	Database, point of contact	
Krisztina Nógrádi	Client, project report, presentation	
Bill Tian	GUI	

*Table 1: Team organisation*

## 5. Project Diary

For a detailed description, please see Appendix 6.

Date	Key points
23/02/2016	Discussed team organisation, responsibilities and goals
29/02/2016	Implemented Knock-Knock example
01/03/2016	Discussed database tables
04/03/2016	Established protocol
08/03/2016	Started implementing methods for protocol, database set up
17/03/2016	Fully implemented, working system

*Table 2: Project diary*

---

## 6. Evaluation

The system evaluation can be executed by comparing the specification to the actual output of the project.

The user should be able to:

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

With the exception of chat history, the team has managed to deliver on every other part of the specification. Areas of possible improvement are the chat history and a more advanced password encryption.

---

## 7. Bibliographic References

Oracle (2015) *"Java Tutorials: Writing the Server side of the socket"* Available from: <<https://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html>> [29/02/2016]

Deitel P. , Deitel H. (2015) *"Java: How to program"*, Tenth Edition, London: Pearson

---

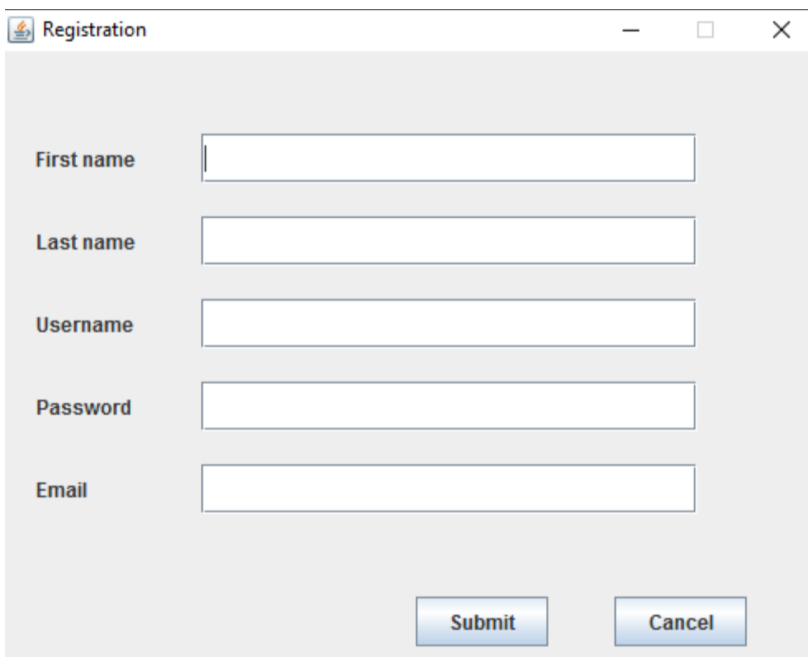
## 8. Appendices

### Appendix 1: Login



A login form with a blue background. At the top, there is a large, stylized graphic that says "(hi)" in a dark blue font. Below this, there are two input fields: "Username" and "Password". Under the "Password" field, there are three buttons: "Forgot password", "Sign Up", and "Log In".

### Appendix 2: Registration



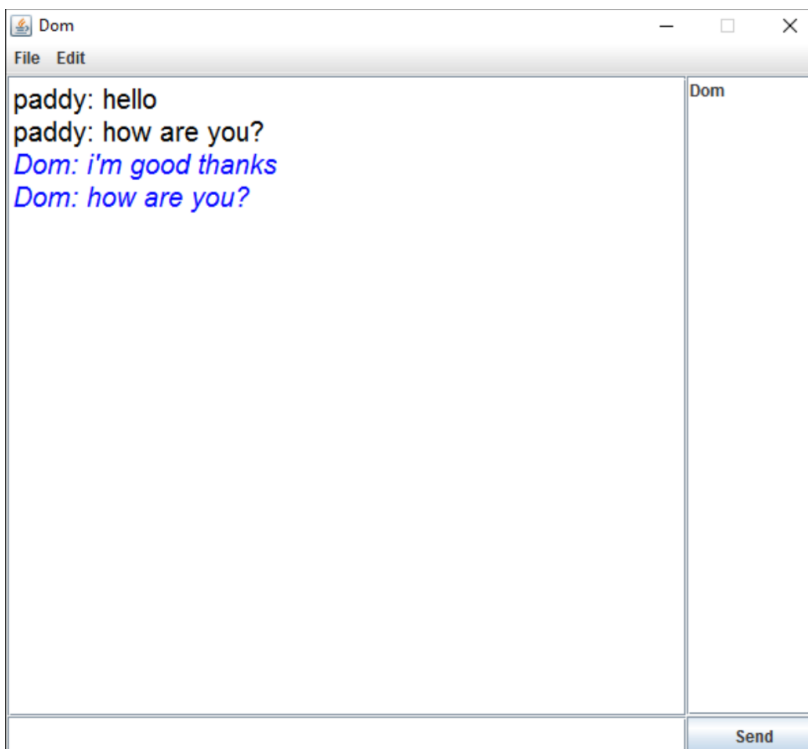
A registration form window titled "Registration". It contains five input fields: "First name", "Last name", "Username", "Password", and "Email". At the bottom right, there are two buttons: "Submit" and "Cancel".



## Appendix 3: Contacts



## Appendix 4: Chat



---

## Appendix 5: Group



---

## Appendix 6: Meeting minutes

23/02/2016

Present: Dom Parfitt, Natalie Reynolds, Krisztina Nógrádi, Bill Tian

Meeting points:

- Meetings every week: Tuesday 1pm - 3pm and Friday 11am-1pm
- Additional meetings will be scheduled as necessary
- Dom to be secretary
- Natalie to be point of contact
- Project will be to create a Client-Server Messenger software
- Krisztina doing Client Side
- Dom doing Server side
- Bill doing Database
- Natalie doing GUI
- Aiming for high standard of work (grade of 80%+ is target)

Action points:

- Questions for James
- Where to start?
- Timeframes and what to aim to achieve each week?
- Meeting James on Thursday 2:30pm
- Everyone to research individual sections and get an idea for what is required

29/02/2016

Present: Dom Parfitt, Natalie Reynolds, Krisztina Nógrádi, Bill Tian

Meeting points:

- Having discussed further following the first meeting it was decided that Nat will do the databases element of the project and Bill will do the GUI. Bill will be able to help out with databases as necessary.

- Started discussing what we felt would be needed from the system. Nat took notes to form the basis of a requirements document.

- Agreed that we want to get a basic skeleton of the system working and then build upon that with additional features that we feel would enhance it.

- We got the Java Knock Knock Server running between two laptops. We then managed to get the same server to allow connections from multiple clients.

- Put together a rough plan for the tables that the database will require.

- Set up the SVN repository and got everyone successfully connected to it.

Action points:

- 
- Natalie to write up requirements document
  - Dom to look over the code from the Java Knock Knock Server to get an understanding of how it works
  - Kriszti to look over the code from the Java Knock Knock Client to get an understanding of how it works
  - Everyone to think about how each element will link together so we can start deriving classes
  - At the next meeting tomorrow we will start planning out the interaction of classes and start drawing some UML diagrams, aim to have interfaces written by the end of next meeting

### 01/03/2016

Present: Dom Parfitt, Natalie Reynolds, Krisztina Nógrádi, Bill Tian

Meeting points:

- Discussed the database requirements and continued with planning the tables that would be required for the database.
- Discussed implementing an MVC framework for the GUI
- Discussed the different views that would be required for the GUI
- Discussed the beginnings of the interfaces that will be required for each section by working through different situations (logging in, sending a message, etc.)
- Started writing out various classes that will be required
- Got a very simple implementation of a chat running with two clients communicating via the server

Action points:

- Nat to continue with requirements document
- Nat to write up database schema based on planned tables
- Dom to work on better implementation of server

### 04/03/2016

Present: Dom Parfitt, Natalie Reynolds, Bill Tian

Meeting points:

- Krisztina is unwell, so could not attend in person but is attending remotely via Facebook
- Nat currently writing up requirements document
- Discussed the required protocols for the client-server connection
- Protocols to be used will be:
  - Registration

- 
- Logging in
  - Forgotten password
  - Chatting
  - Logging out
- Nat has set up a dummy database along with a database java class to act as the connection.
- The Database class is able to connect to the DB using JDBC and can run some simple queries.
  - The server will hold an instance of the Database class as a way to interact with it.
  - Bill has created views for the login screen, the contacts screen and the chat screen.
  - Discussed implementing a Message class that will be used as a type of protocol. The class would hold a type variable indicating what kind of message it is (and therefore what the server/client needs to do with that message) along with other elements such as an actual message and a timestamp.
  - The types of message will correspond to the different protocols we agreed above.
  - Discussed adding a Contacts table to the DB so that users don't necessarily have access to chat to all other users

Action points:

- Dom to create a Message class
- Dom to look at threads and thread pools for the server
- Nat to finish Database interface and then start implementing methods
- Kriszti to look at protocols
- Bill to continue working on the different views

08/03/2016

Present: Dom Parfitt, Natalie Reynolds, Krisztina Nógrádi, Bill Tian

Meeting points:

- Discussed the use of the Message class and its implementations of protocols.
- Discussed the methods that will be required by the Client component to create and receive the different types of Message
- Client needs to be able to create messages for:
  - Registration: user is registering on system
  - Logging in: user is logging to system
  - Forgotten password: user has forgotten password
  - Chatting: user is sending a chat message to another user
  - Logging out: user is logging out
- Client needs to be able receive and interpret messages for:

- 
- Registration: user has been successfully registered
  - Logging in: user has been successfully logged in
  - Forgotten password
  - Chatting: message from another user
  - Logging out: user has been successfully logged out
  - Error: registration/logging in/etc. has been unsuccessful
  - Server needs to be able to create messages for:
    - Registration: registration has been successful
    - Logging in: log in successful
    - Forgotten password: password changed?
    - Logging out: user successfully logged out
  - Server needs to be able receive and interpret messages for:
    - Registration: user requesting registration on the system
    - Logging in: user requesting to log in to the system
    - Forgotten password: user needs password resetting
    - Chatting: user is sending chat message to another user
    - Logging out: user is requesting to log out of the system
  - Nat has created methods in the DB class to get different fields, registering an new user and inserting sent messages into the DB
    - Using comma as a delimiter in the message text string to break up different elements (e.g. username and password)
- Action points:
- Kriszti to implement required methods for the Client to send and receive Messages
  - Nat to keep working on DB methods
  - Dom to finish implementing methods to send and receive Messages
  - James about setting IP and port number at meeting on Thursday

## Appendix 7: Protocol

Protocol		Server Expects				Client Expects			
No.	Name	From	Destination	User	Text	From	Destination	User	Text
0	Registration	n/a	n/a	n/a	Firstname Lastname Username Password Email	n/a	n/a	n/a	n/a
1	Login	n/a	n/a	n/a	Username Password	n/a	n/a	User object created with provided log in details	Username s of other online users
2	Forgotten Password	n/a	n/a	n/a	Username Email	n/a	n/a	User object created with provided log in details	Username s of other online users
3	Chat	n/a	Group of usernames to pass message on to	n/a	n/a	Usernam e of user who sent the message	Group containing usernames of users who the message	n/a	Some chat text

							was sent to		
4	Logout	n/a	n/a	n/a	n/a	n/a	n/a	n/a	Confirmation message
5	Error					n/a	n/a	n/a	Reason for error
6	Add Contact	n/a	n/a	n/a	Username Contact's username	n/a	n/a	User object with updated contacts	Username s of other online users
7	<i>Chat History</i>	<i>User who sent the request</i>	<i>Other users involved in the chat</i>	<i>n/a</i>	<i>Number of messages to retrieve</i>	<i>n/a</i>	<i>Group containing everyone involved in the chat</i>	<i>n/a</i>	<i>List of messages (timestamp, from, text)</i>
8	Online Contacts					n/a	n/a	User who has come online or gone offline	n/a