# Distributed Computing Project

## Eoin Galvin

## T00213033

## Summary

## Table of Contents

Contents

[Distributed Computing Project 1](#_Toc130926415)

[Eoin Galvin 1](#_Toc130926416)

[T00213033 1](#_Toc130926417)

[1.1 Summary 2](#_Toc130926418)

[1.2 Table of Contents 2](#_Toc130926419)

[1.3 Introduction 2](#_Toc130926420)

[1.4 Design and Implementation 2](#_Toc130926421)

[1.4.1 Client Application 2](#_Toc130926422)

## Introduction

## Design and Implementation

The objective of the design is to create a secure, concurrent, client-server short message protocol using TCP. The system should support the ability for users to login securely, upload messages and download all messages stored on the server. The user should also be able to log-out of the server and disconnect.

### 1.4.1 Client Application

The Client application follows the Model-view-controller architectural pattern. This pattern allows the user to interact with the view, while the controller controls all interaction between the view and the model. The model, or in this case the Client interacts with the service layer, ClientSteamSocket to ensure smooth communication with the server application.

A screenshot of a video game

Description automatically generated

The presentation logic of the client application is created using Java JFrame and JPanels. The frame is instantiated by the controller at runtime and the frame then instantiates a JPanel which contains the user interface components such as text fields and buttons.

Graphical user interface

Description automatically generated

The application has six different functionalities. It allows a user to connect to the server, disconnect from the server, login and logout of the server and send messages and download all messages stored on the server.

#### Connect

When the connect button is clicked, a connection to the server is attempted to be established. If the server is not running or inaccessible, an error message “Connection refused” is displayed. If the connection is successful, a “Connection accepted” message is displayed.

Graphical user interface, application, Word

Description automatically generatedGraphical user interface, application, Word

Description automatically generated

A connection is made by instantiating a ClientStreamSocket with the given server host and server port. A screenshot of a computer

Description automatically generated

In the ClientStreamSocket constructor, an SSL socket is created using an SSLSocketFactory. This socket will be used for secure communication between the client and the server. The client and server should use the same protocol so the protocol TLSv1.2 is defined. The socket must also know the location of the client trust store and its password so it can access its keys and certificates that will allow it to create a secure, encrypted connection with the server. It must also support the same cipher suites as the server does so they are also defined. When the ClientStreamSocket is instantiated, the handshake process between the client and the server begins. The connection will fail if the server is unavailable, so a Connect Exception will be caught, and the user will be notified. Connecting to the server does not mean you are logged in; it just means a secure connection has been created and the user can now communicate its login details with the server safely with the SSL encryption.

Text

Description automatically generated

Text

Description automatically generated

#### Disconnect

The disconnect function will close the connection between the client and the server. If the user has not logged out, it will be automatically logged out when the current connection is disconnected.

Text

Description automatically generated

The client sends a final message to the server to indicate it is disconnecting before closing the socket.



The user is notified that the disconnection was a success on the UI.



#### Login

Logging in to the server is required to have access to send and download messages from the server. A user will enter a username and password into the text fields and then click the login button to send the request to the server. The client initially sends a protocol message to the server, notifying it that the next two messages received are the username and password.

Text

Description automatically generated

The client will then expect a protocol message to be returned from the server. This message will indicate if the user has been successful at logging in. The user is notified of the outcome of the request on the user interface.



#### Logout

To log out of the server, the client will send the current logged in username back to the server with a protocol message.

Text

Description automatically generated

The server will return a protocol message to the client to indicate if the logout has been successful or not. The client will update the user interface to inform the user.



The client also has checks to ensure a connection is made before a login/logout request is made and if no connection has been created the user is notified.



#### Send Message

A message can be sent to the server once the client has connected, and a user has logged in successfully. The client sends the server a message that it wishes to be saved.

Text

Description automatically generated with low confidence

Graphical user interface, application, Word

Description automatically generated  
If a user is not logged in to the server an error will be displayed on the UI.



#### Download Messages

A user can allow all messages that have been stored on the server. The client must be connected using a secure SSL socket to the server and the user must be logged in. If the user is not logged in an error will be displayed on the UI.



To download all messages, the client first sends a protocol message to the server to request all messages. The client then expects to receive all the messages on the server in String format. Once the messages are received, they are returned to the client to be displayed on the UI.

Text

Description automatically generated



### 1.4.2 Server Application

The server is designed to run in a multi-threaded environment, allowing multiple users to connect at once by allocating a thread when connected. The server communicates with the client using an SSL socket using the TLSv1.2 protocol. It uses the SunX509 key algorithm for security.

Text

Description automatically generated

The old unsecure socket is commented out in the code below as it was replaced with the secure SSL socket. Text

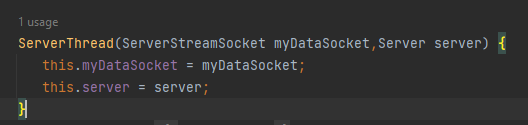
Description automatically generated

Once the server receives a connection request from a client, a data socket is created using the SSL socket and a new Thread is also created. This allows the client to have its own secure connection to the server and it will not interfere with any other clients that are also communicating with the server.

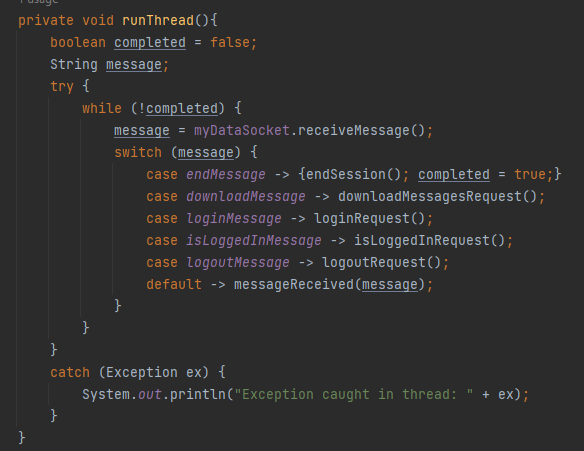
Text

Description automatically generated

The server sends a reference to itself when creating the thread, this allows a lock to be put in place to allow all data stored on the server to be synchronized across all server threads. All get and set methods in the server have the synchronized keyword to ensure that an intrinsic lock is applied to all the data on the server. This lock allows only a single thread to execute at a single time when modifying the data stored on the server. The Thread is instantiated using the SSLSocket and a reference to the Server which created the thread is stored in the Thread object.



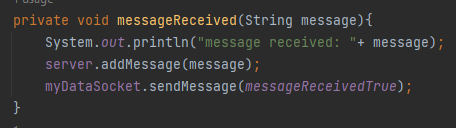
Once the Thread is created, it is ready to receive messages from the client. The thread runs until an end protocol message is received. The different cases are set up to deal with different protocol messages being received.



#### Receive Message

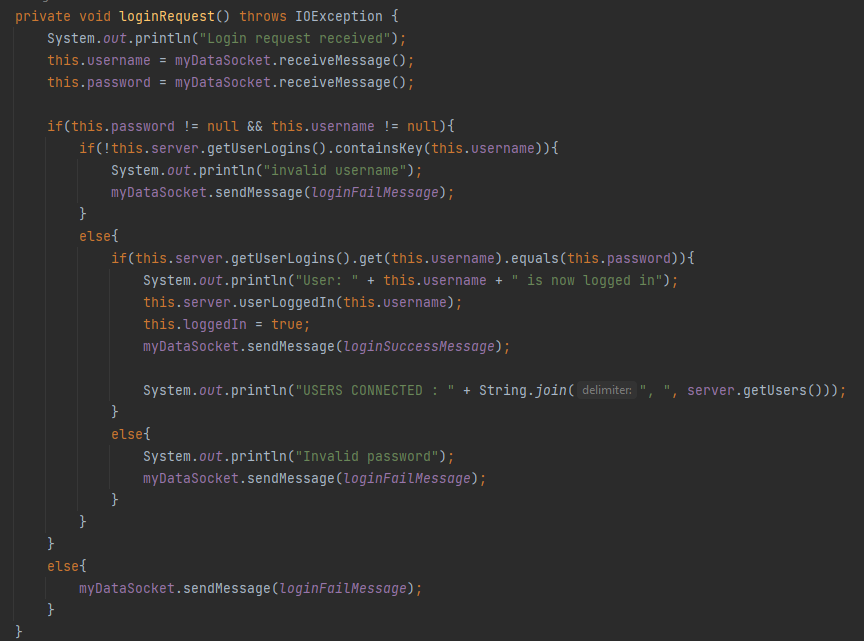
When the Thread receives a message, it checks if the message received is a protocol message. If the message is not a protocol message, it is treated as a regular message and is stored on the server.

The server sends back a protocol message to notify the Client that the message was received.



#### Login Request

A thread may receive a login message. If a login message is detected, the server then acknowledges that the next two messages received securely will be the username and password of the user. Once the username and password are received, the login credentials are validated. If the credentials are validated, the user is added to the currently logged in users on the server and a login successful message is sent back to the client. If the credentials are invalid, a login fail protocol message is sent back to the client.



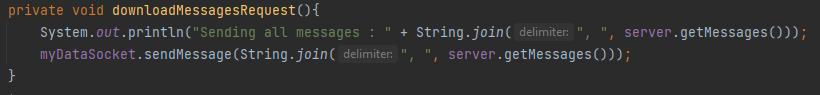
#### Logout Request

When a thread receives a logout request, the server then expects to receive the username wishing to logout through the SSL socket. Once the username is received the server ensures that the user is currently logged. If the user is logged in then the user is removed from the current user list and a logout success message is sent back to the client.



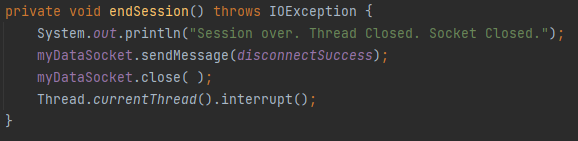
#### Download Messages Request

When the server receives a download all messages protocol messages, all the currently stored messages are concatenated to a String and sent back to the client. The server.getMessages method is synchronized to ensure that the requested messages are up to date.



#### Disconnect

When the server receives an end session protocol message the server will close the connection between the client. It will then end the current thread so that server resources are available for the next user that connects to the server.



#### Login Check

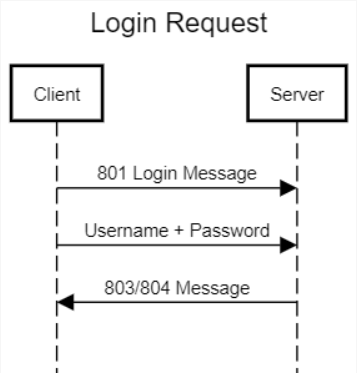
The server also supports a request to check if a user is currently logged in. This check is used in the client interface to display messages on the UI if the user is logged in. When the request is received through a protocol message, it sends back a message to let the Client know if the current user is logged in or not.



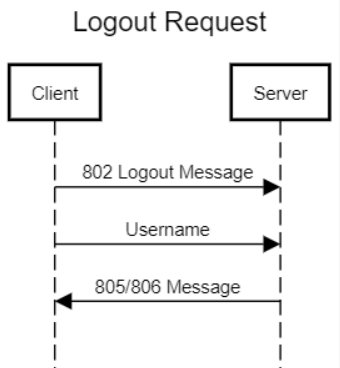
## Protocol Documentation

|  |  |  |
| --- | --- | --- |
| **Protocol Code** | **Protocol Description** | **Parameters** |
| **801** | The user can login to the server with a username and password. | Username, Password |
| **802** | The user can log out of the server. | Username |
| **803** | Login Success message. |  |
| **804** | Login Fail message. |  |
| **805** | Logout Success Message. |  |
| **806** | Logout Fail Message. |  |
| **807** | This message notifies the server to disconnect the current session. |  |
| **808** | The user can download all messages stored on the server. |  |
| **809** | A request to check if a user is logged in to the server. | Username |
| **810** | The user is currently logged in to the server. |  |
| **811** | The user is not currently logged in to the server. |  |
| **812** | Message has been received. |  |
| **813** | Disconnection Successful |  |
| **N/A (Default)** | A message that will be stored on the server. | Message |

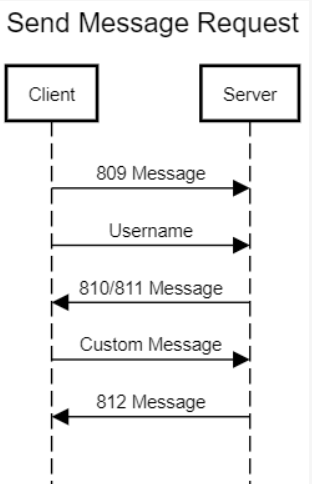
#### Login Request



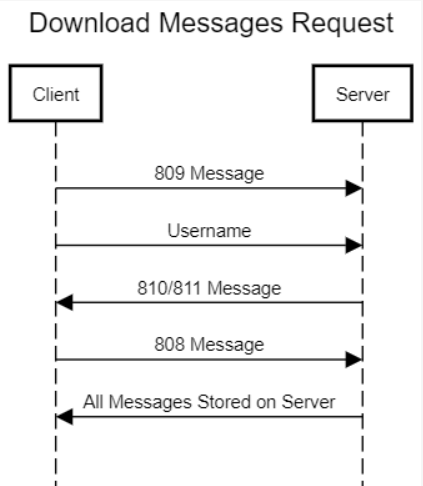
#### Logout Request



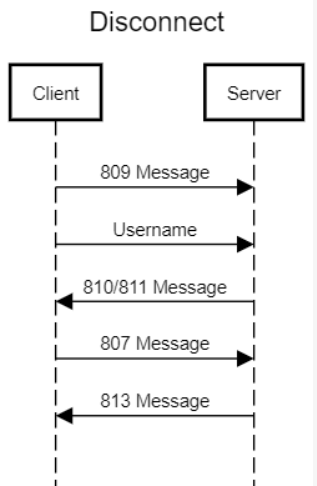
#### Send Message Request



#### Download Message Request



#### Disconnect Request



#### Connect Request

