

# Creating a Twitter Message Protocol.

Emma Fitzgerald

BSc(Hons) Computing with Games Development

# Summary:

This report will show how the TMP (Twitter Message Protocol) is built and show how the client and server interact with each other through messages. It will demonstrate how the system is built and how the system uses the three-layer approach for the client-side and the server-side. This report will demonstrate each phase of the system from the protocol design to the implementation phase.

# Contents

[Summary: 2](#_Toc69146501)

[Contents 3](#_Toc69146502)

[Introduction: 4](#_Toc69146503)

[Section 1: Twitter Message Protocol 5](#_Toc69146504)

[1.1 Introduction 5](#_Toc69146505)

[1.2 Objectives 5](#_Toc69146506)

[1.3 Log On To Server 5](#_Toc69146507)

[1.4 Upload Message 7](#_Toc69146508)

[1.5 Download Messages 9](#_Toc69146509)

[1.6 Log Off Of Server 11](#_Toc69146510)

[Section 2: Design Phase 14](#_Toc69146511)

[2.1 Objectives and Overview 14](#_Toc69146512)

[2.2 Design Philosophy 14](#_Toc69146513)

[2.3 Client-side Design UML 14](#_Toc69146514)

[2.4 Server-side Design UML 15](#_Toc69146515)

[Section 3: Implementation Phase 16](#_Toc69146516)

[3.1 Objectives and Overview 16](#_Toc69146517)

[3.2 Application Layer 16](#_Toc69146518)

[3.3 Presentation Layer 19](#_Toc69146519)

[3.4 Service Layer 21](#_Toc69146520)

[3.5 User Manual 22](#_Toc69146521)

[Conclusion 26](#_Toc69146522)

[Bibliography 27](#_Toc69146523)

# Introduction:

The objective of this report is to demonstrate how to create a TMP (Twitter Message Protocol) using sockets and threads in a secure manner. The TMP protocol works by sending messages between the client and the server through a secure SSL connection using sockets and threads. These messages will consist of a code and messages that can be parsed by the system to determine what operation needs to be executed at that time. Every message that is passed between the server and client is encrypted for more security. The main operations that will be executed by the protocol are allowing the user to log in, allowing the user to upload messages to the server, allowing the user to download and display all their messages on the server, and allowing the user to log off. The system consists of a three-layer approach for both the client-side and server-side. The three layers are the application layer, the presentation layer, and the service layer. The result of this report will be to fully understand how the TMP works and to show how the system works.

# Section 1: Twitter Message Protocol

## 1.1 Introduction

The TMP (Twitter Message Protocol) will provide the user with a message-saving service. The user will have to log onto the server to gain access to their messages and to be able to upload new messages onto the server. These messages are saved within a file that only the server can access. The protocol uses SSL for its connection and encrypts the messages between the client and the server.

## 1.2 Objectives

The user should be able to:

* Log on with a username and password to access the messages on the server
* Upload new messages to the server
* Download and display all the user’s messages on the server
* Log off the server when finished

The protocol will allow the client to communicate with the server to:

* Check if the user is registered on the server by checking a file with all available users
* To upload messages to the server for the logged-in user
* To download messages from the server for the logged-in user
* To log off the user from accessing the server once they are finished

## 1.3 Log On To Server

**Message Format:**

* Message: Login
* Description: Once the user enters their username and password the user will be able to log onto the server.
* Code/Number: 100
* Message Parameters: username, password (both string)
* Response Message
  + Code: 110
  + Text: Login successful! Welcome
* Response Message
  + Code: 120
  + Text: Username or Password is incorrect please try again!

**Message Functionality:**

Implementing the connection message on the client as a function called LogOn()

* Send the message to the server Login;100;username;password

Implementing the connection message on the server as a function called LogOn()

* The method will read in a list of users from the users.csv file
* Compare the username and password entered by the user to the users registered
* If a match is found then return 110;Login successful! Welcome username
* If no match found then return 120;Username or Password is incorrect please try again!

**Pseudocode**:

Client method:

Function LogOn(username, password)

try

Declare returnMessage

Declare message -> Login;100;encrypted username;encrypted password

Send message to server

Set returnMessage -> message received from server

Call method SetUsername(username)

return returnMessage

End try

catch

When exception type IOException

Print message “Error occurred when logging in”

End catch

return “”

End function

Server method:

Function LogOn(msg)

Read in users from users.csv into a list called users

Declare returnMessage

for each user in users do

if the msg username and password match a username and password from users do

Set username -> decrypted username from msg

Set returnMessage -> 110;Login successful! Welcome username

else

Set returnMessage -> 120;Username or Password is incorrect please try again!

End if

End for

return returnMessage

End function

**Sequence diagram:**

Server

Client

Login;100;username;password

110;Login successful! Welcome username

120;Username or Password is incorrect please try again!

## 1.4 Upload Message

**Message Format:**

* Message: Upload
* Description: The user can upload a string message to their file on the server
* Code/Number: 200
* Message Parameters: message (string)
* Response Message
  + Code: 210
  + Text: Upload Successfully
* Response Message
  + Code: 220
  + Text: Message was empty. Upload Failed!

**Message Functionality:**

Implementing the connection message on the client as a function called Upload()

* Send the message to the server Upload;200;clientMessage

Implementing the connection message on the server as a function called Upload()

* The method checks to make sure the user entered the message
* If there is a message the message will be entered into the users file and it returns 210; Upload Successfully
* If there is no message return 220;Message was empty. Upload Failed!

**Pseudocode**:

Client method:

Function Upload(clientMessage)

try

Declare returnMessage

Declare message -> Upload;200;clientMessage

Send message to server

Set returnMessage -> message received from server

Call method SetUsername(username)

return returnMessage

End try

catch

When exception type IOException

Print message “Error occurred when uploading message”

End catch

return “”

End function

Server method:

Function Upload(msg)

if the message isn’t empty do

write to the users file with the message

returnMessage -> 210; Upload Successfully

else do

returnMessage -> 220;Message was empty. Upload Failed!

End if

End function

**Sequence diagram**

Server

Client

Upload;200;clientMessage

210; Upload Successfully

220;Message was empty. Upload Failed!

## 1.5 Download Messages

**Message Format:**

* Message: Download
* Description: The user can download and display all their messages on the server
* Code/Number: 300
* Message Parameters: message (string)
* Response Message
  + Code: 310
  + Text: Download Successful
* Response Message
  + Code: 320
  + Text: There are no messages available for download!

**Message Functionality:**

Implementing the connection message on the client as a function called Download()

* Send the message to the server Download;300

Implementing the connection message on the server as a function called Download()

* The method will read in all the messages from the users file
* If there are messages in the file it returns 310; Download Successfully
* If there are no messages return 320;There are no messages available for download!

**Pseudocode**:

Client method:

Function Download()

try

Declare returnMessage

Declare message -> Download;300

Send message to server

Set returnMessage -> message received from server

return returnMessage

End try

catch

When exception type IOException

Print message “Error occurred when downloading messages”

End catch

return “”

End function

Server method:

Function LogOn(msg)

Read in messages from users file into a list called clientMessages

if clientMessage is not empty do

Declare messages -> “All messages:”

for each message in clientMessage

Add the message to messages

End for

return 310;messages

else do

return 320;There are no messages available for download!

End if

End function

**Sequence diagram**

Server

Client

Download;300

310;messages

320;There are no messages available for download!

## 1.6 Log Off Of Server

**Message Format:**

* Message: Logoff
* Description: The user can log off the server
* Code/Number: 400
* Message Parameters: none
* Response Message
  + Code: 410
  + Text: Log Off Successful

**Message Functionality:**

Implementing the connection message on the client as a function called LogOff()

* Send the message to the server Logoff;400

Implementing the connection message on the server as a function called LogOff()

* return 410;Log Off Successful

**Pseudocode**:

Client method:

Function Download()

try

Declare returnMessage

Declare message -> Logoff;400

Send message to server

Set returnMessage -> message received from server

return returnMessage

End try

catch

When exception type IOException

Print message “Error occurred when logging off”

End catch

return “”

End function

Server method:

Function LogOn(msg)

return “410;Log Off Successful”

End function

**Sequence diagram**

Server

Client

Logoff;400

410;Log Off Successful

# Section 2: Design Phase

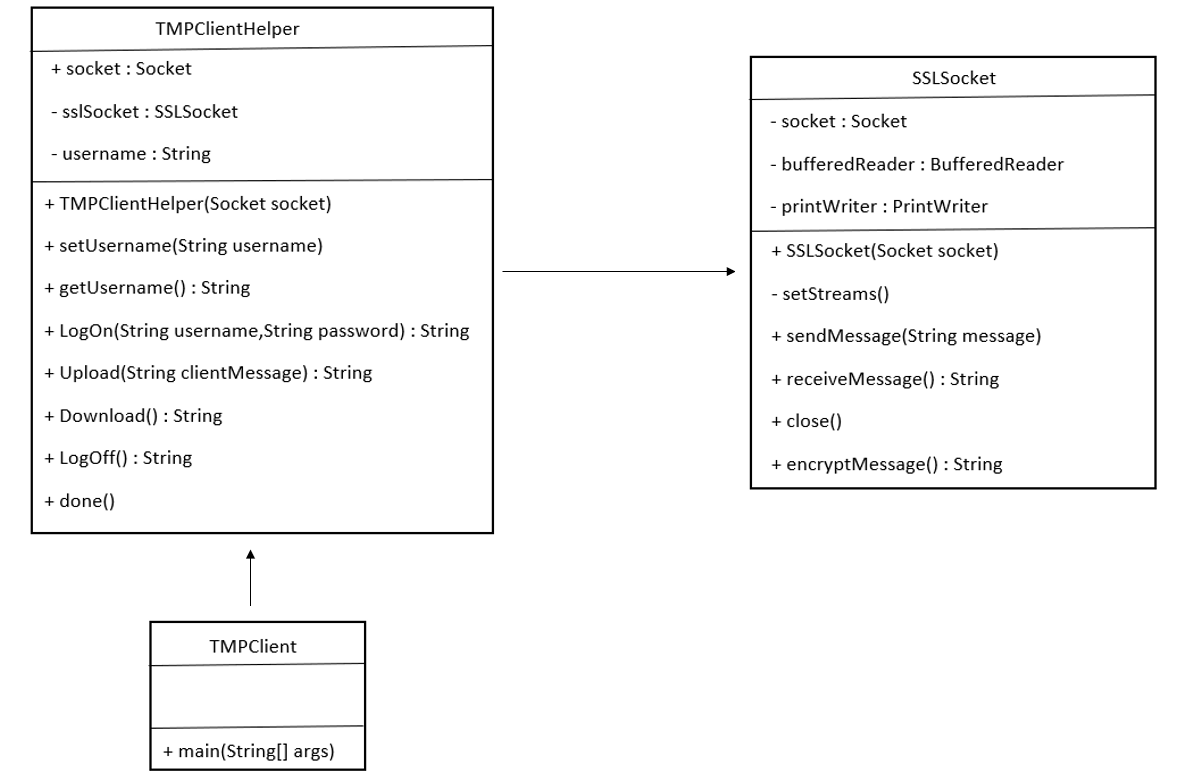
## 2.1 Objectives and Overview

The object of the design phase was to create a plan for the classes that will create the TMP system. The result of the design phase was that there would be a clear UML of the client-side and server-side classes to properly clarify how each class would interact with each other.

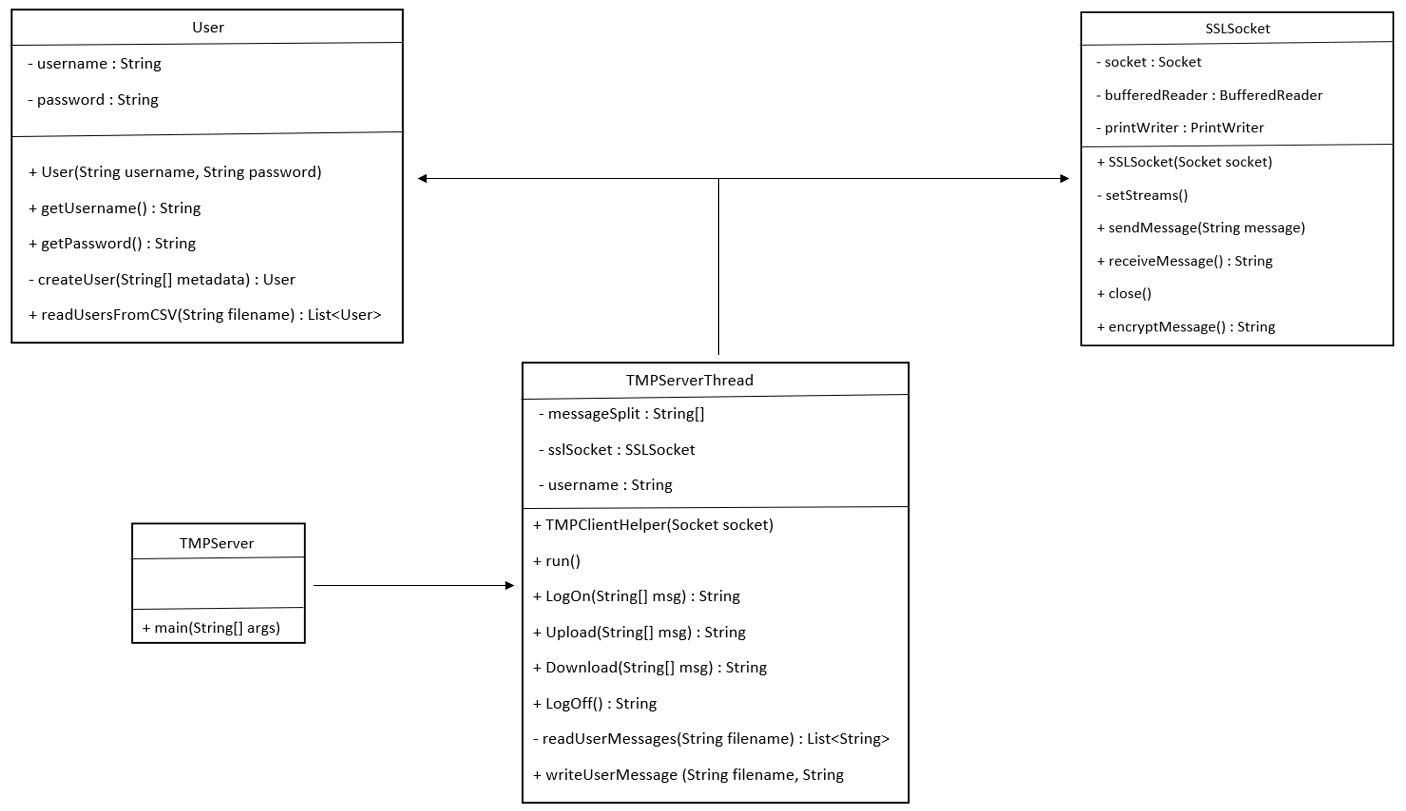
## 2.2 Design Philosophy

The design philosophy was to create a system that would have three layers: the application layer, the presentation layer, and the service layer. The application layer for the client-side is the TMPClientHelper class and the server-side is the TMPServerThread class. The code inside these classes will handle the sending and receiving messages between the client and the server. The presentation layer for the client is the TMPClient class and for the server-side, it’s the TMPServer class. The TMPServer class will just handle opening the server connection once it is called. The TMPClient class will handle the user interface and the user input. Each time the user enters data into the system a method within the TMPClientHelper class will interact with the server. The service layer for both the client and the server is the SSLSocket class. This class handles the sending and receiving of messages between the client and the server in a secure manner. The User class was incorporated with the server-side so that once the users on the server are read in from a file, it would be easier to access the username and password for each user.

## 2.3 Client-side Design UML



## 2.4 Server-side Design UML



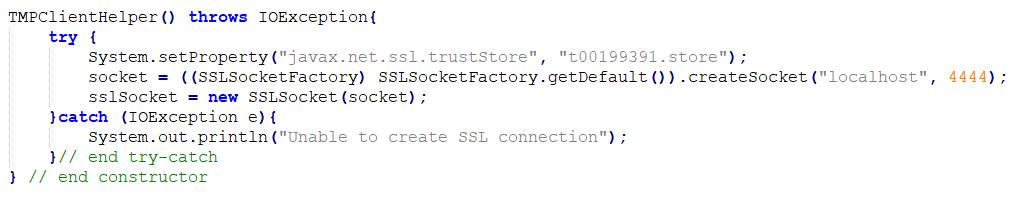
# Section 3: Implementation Phase

## 3.1 Objectives and Overview

The objective of the implementation phase was to create the code for the protocol from the design phase. The objective of the code was to create a system that would send messages between the client and the server to complete the operations log on, upload message, download messages, and log off. The system would have an SSL connection and message encryption for the secure transfer of data over the server. The system would be split into three layers: application layer, presentation layer, and service layer.

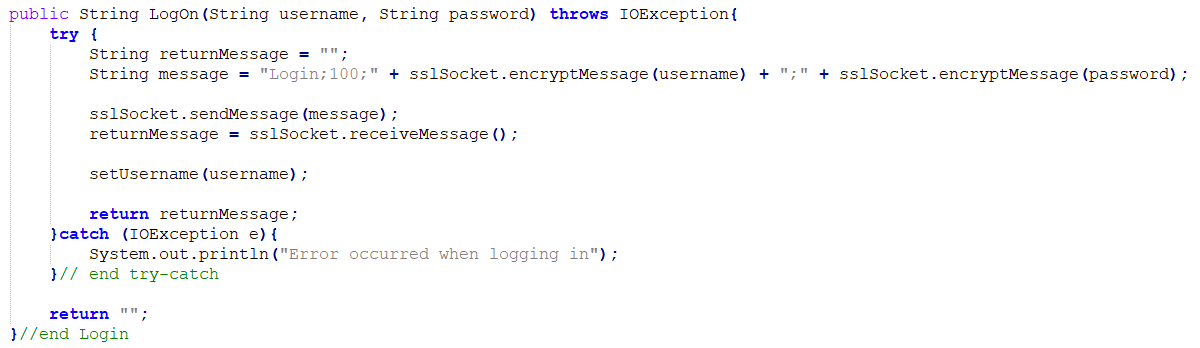
## 3.2 Application Layer

For the client-side of the protocol, the application layer is represented by the TMPClientHelper class. This class handles the SSL socket creation and handles and information being passed from the client to the server.



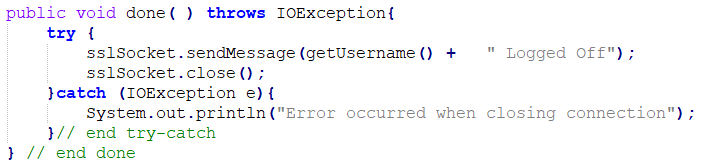
***Figure 3.2.1***

Figure 3.2.1 shows the constructor of the TMPClientHelper class. The constructor will create the SSL socket connection for the client-side by using the t00199391.store trust store. This will allow for a secure connection to be established for the client-side.



***Figure 3.2.2***

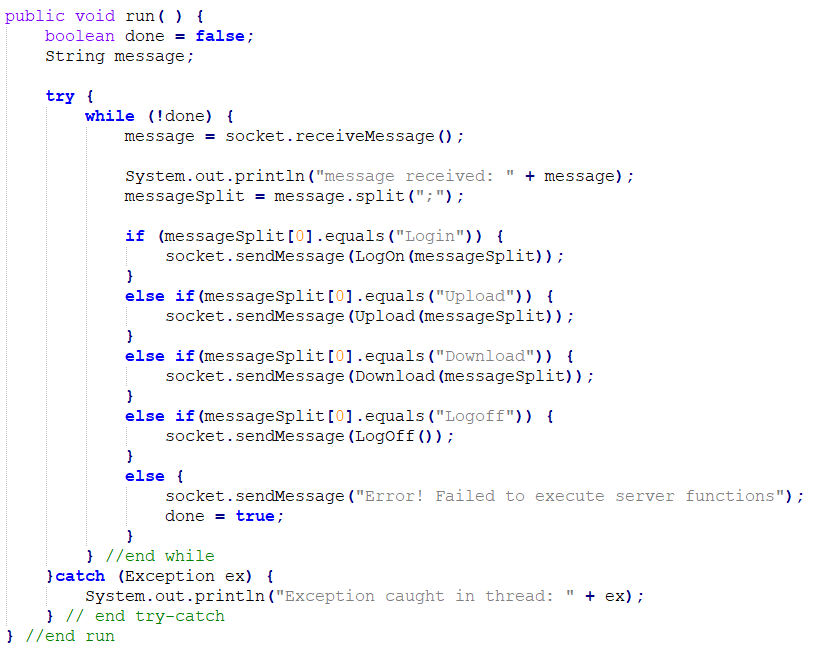
The TMPClientHelper class has methods for each operation the user can do with the system. There is a method to handle the user log in, the user uploading new messages to the server, the user downloading and viewing all messages on the server, and logging off. Figure 3.2.2 displays the LogOn() method. The parameters for this method are the username and password entered by the user. A message will be created that will be passed to the server so the user can log in. The message will consist of the operation name, the operation code, the username, and the password of the user. This message will then be passed into the SSLSocket method sendMessage(). The server will then pass back a message that will be used in the TMPClient to determine if the user can log in or not. This method of creating a message, sending the message to the server, and waiting for a response message is used within the other methods of this class.



***Figure 3.2.3***

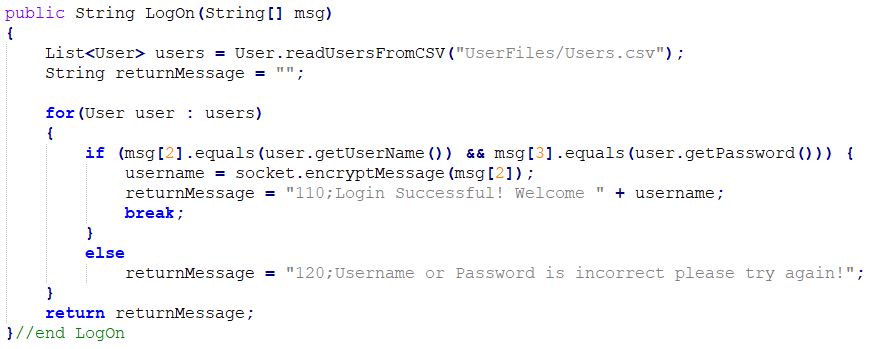
Figure 3.2.3 shows the method that is called once the user decides to log off. The method will close the socket connection and send a message to the server to tell it that that user has now logged off.

For the server-side of the protocol, the application layer is represented by the TMPServerThread class. This class will handle the messages received from the client-side and determine the response message.



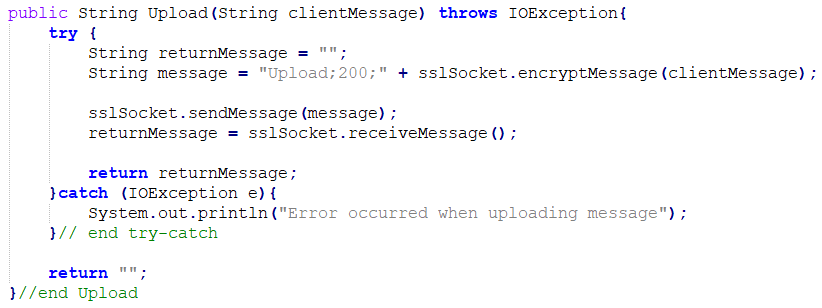
***Figure 3.2.4***

The server will receive the message and determine which method within the class to call. Figure 3.2.4 shows how this operation takes place. Once the server is running this while loop will continue to run checking each message that is received from the client. The message will be parsed into an array. The arrays first element is then checked to see that operation the client-side is trying to perform.



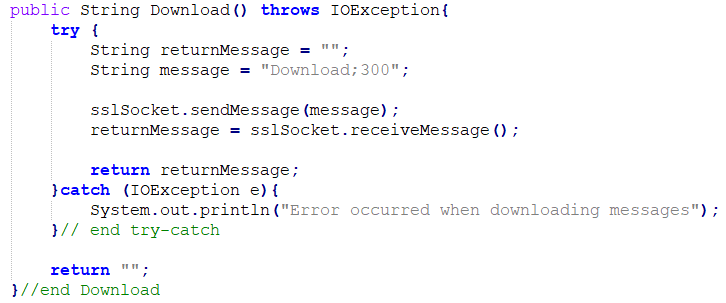
***Figure 3.2.5***

Figure 3.2.5 shows the LogOn() method that is called once while loop from Figure 3.2.4 determines this is the correct method to use. This method will read in from a file all the registered users on the server and place them in an array of type User. These users’ usernames and passwords will be compared to the username and password passed in through the message received from the client. Once a match has been made a return message will be created with a code of 110 and a message for the user. If the user is not found within this file, then a return message with a code of 120 and a failed to log in message will be returned to the client-side.



***Figure 3.2.6***

Figure 3.2.6 shows the Upload() method. This method will check first to see if a message was passed in. If there is a message, then this message will be appended to the file that belongs to the user currently logged in. It will then return a message with the code 210 and a message stating that the message was upload correctly to the client. If no message was passed in by the user a return message with the code 220 and an upload unsuccessful message is passed back to the client.

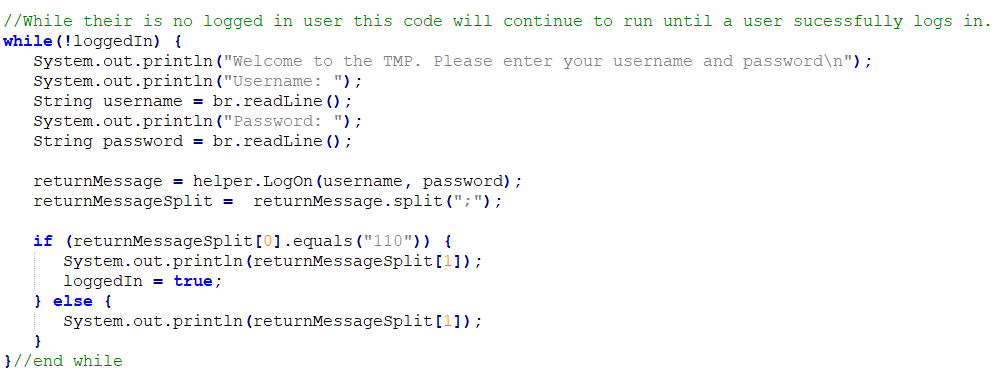


***Figure 3.2.7***

Similarly to the LogOn() method, the Download() method in Figure 3.2.7 will read in the messages from the user’s file and display them to the user. The method will first check to see if there are messages in the users’ file. If there are messages then it will read in each line (message) from the file and append it to a string value. This string value will be passed back to the user through the return message with the code 310. If there are no messages within the file, a return message with the code 320 and a no messages available message will be returned to the client.

## 3.3 Presentation Layer

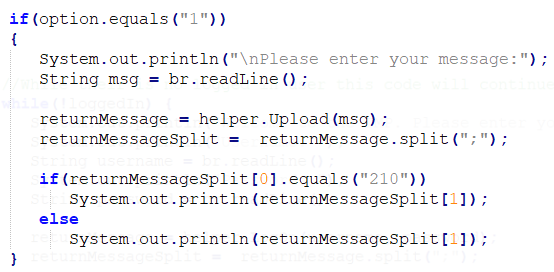
For the client-side of the protocol, the presentation layer is represented by the TMPClient class. The TMPClient class handles what the user sees on the interface, handles passing in data that the user entered to the TMPClientHelper class and handles the return messages from the server.



***Figure 3.3.1***

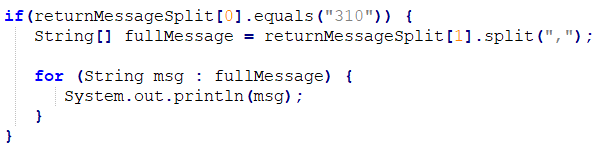
Once the user isn’t logged in they will be presented with a user interface that will allow them to enter their username and password which is shown in Figure 3.3.1. This while loop will continue to run until the user correctly enters valid data. Once the user enters their username and password, these values will be passed into the LogOn() method in the helper class and will store the return message from this method. It will then parse this message into an array to gain access to the code that will determine if the login was successful or unsuccessful. If the code return is 110 the then the method will print the message section of the return message to the user and will then set the logged in state to true. If the return code is 120 then the message section will be displayed to the user and the user will be able to enter their username and password again.

Once the user is logged in the interface will display the three options the user can complete: upload message, download all messages and log off. There is a check to make sure the user only selects one of these options i.e. 1-3.



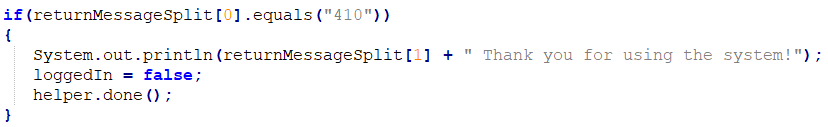
***Figure 3.3.2***

Similarly to the logon functionality, once the user enters their option it will send a message to the server through the helper class and this will return a message to the client. Figure 3.3.2 shows the upload function. Once the user chooses the upload message option and enters their message, the Upload() method from the helper class will be called with the users’ message as the parameter. It will then return a message to the client that will be parsed into an array. The code section of the message will be checked to see if the upload was successful or unsuccessful. The message section of the return message will then be shown to the user. Every option has a similar function where it passes in data to the helper methods and checks the return message to see if the operation was successful or unsuccessful.



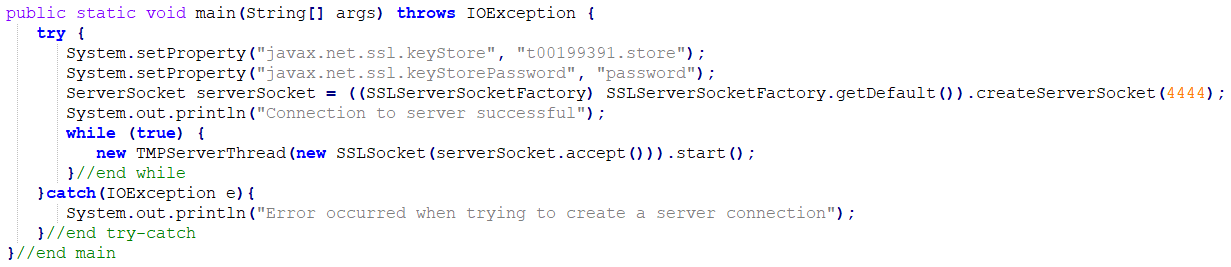
***Figure 3.3.3***

The download option has an added function that will display each message returned from the user’s file in a separate line shown in Figure 3.3.3.



***Figure 3.3.4***

The log off option has an added function that will call the done() method form the helper class to close the connection to the server for the users shown in Figure 3.3.4.

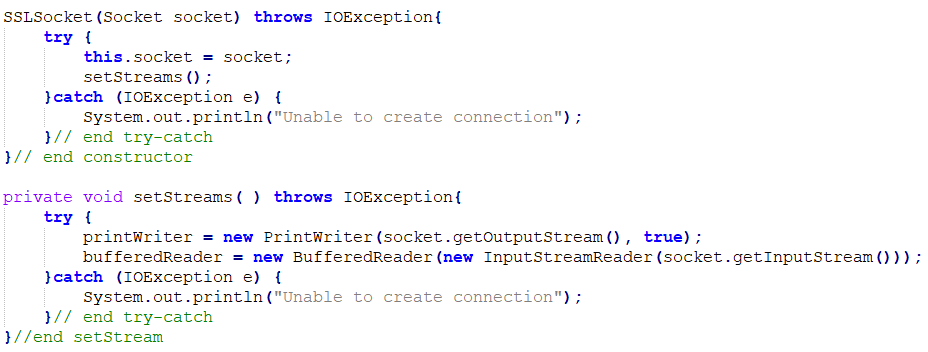


***Figure 3.3.5***

For the server-side of the protocol, the presentation layer is represented by the TMPServer class. The only function that this class has is to create an SSL socket connection. The class will check the key store t00199391.store exists and enters the password for the key store to gain access shown in Figure 3.3.5.

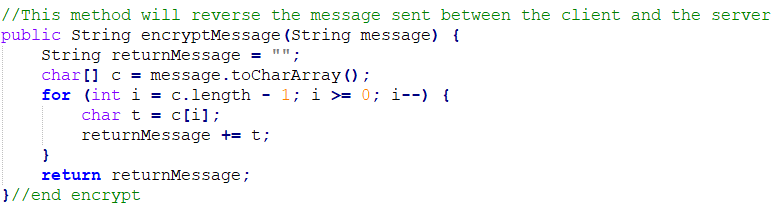
## 3.4 Service Layer

For both the client-side and the server-side, the class SSLSocket is used for the service layer of the application. The SSLSocket class handles setting up the socket streams, the receiving and sending of messages between the client and server, closing of connections and encrypting messages passed in.



***Figure 3.4.1***

Figure 3.4.1 shows the constructor and the stream setting method for the class. The constructor is called within the TMPClientHelper and TMPServer classes to set up the socket streams for both the client and server. This will allow the messages to be securely passed between the client-side and the server-side.



***Figure 3.4.2***

Figure 3.4.2 shows the encrypt method that will reverse the message that is passed through. This method will reverse the messages sent between the client and the server. Messages saved on the server are reversed so if they need to be displayed to the user this method will be called to reverse the messages to make them readable. Once the client enters any data that will be passed into the server this method will be called to encrypt the messages before they enter the server.

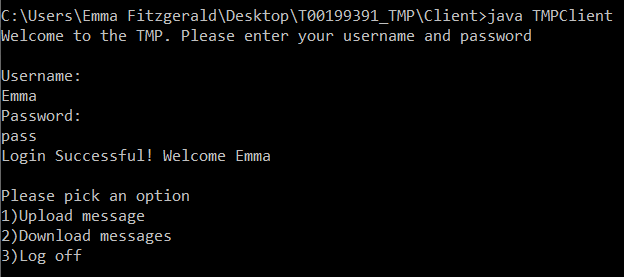
The refences for the code written for the classes above were taken from EchoServer3, EchoServerThread, EchoClientHelper2 and EchoClient2. The code for the SSLSocket was referenced from Prototype Project(2015).

## 3.5 User Manual



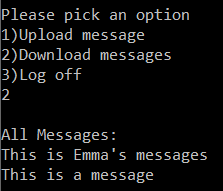
***Figure 3.5.1***

The first step to running the TMP system is to first start the server. To start the server the class TMPServer is ran through the command line. Once the server connection is established, the message in Figure 3.5.1 will be displayed.



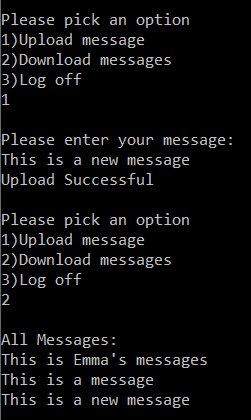
***Figure 3.5.2***

To start the client-side of the system, the class TMPClient is executed. Figure 3.5.2 shows the interface that will be displayed to the user. They will be asked to enter their username and password. A list of valid users is in the file Users.csv in the Server\UserFiles folder. Once the user successfully logs in they will be able to choose an operation they wish to run.



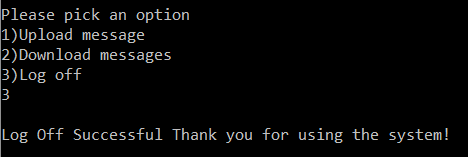
***Figure 3.5.3***

If the user picks option number 2, this will execute the download methods within the system. Once the messages are found they are displayed to the user. This is shown in Figure 3.5.3.



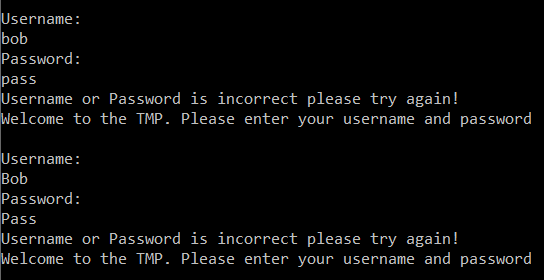
***Figure 3.5.4***

If the user picks option number 1, this will execute the upload methods within the system. The user will be asked to enter a message they want to save to the server. Figure 3.5.4 shows the users’ message and then displaying all their messages to show the message was uploaded onto the server.



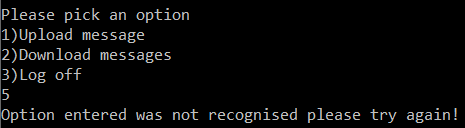
***Figure 3.5.5***

If the user picks option number 3, this will execute the log off methods within the system. Figure 3.5.5 shows that when the users selects option 3 they will be greeted with a log off message.



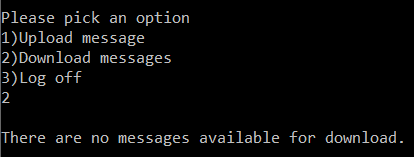
***Figure 3.5.6***

Figure 3.5.6 shows what happens if the user was to incorrectly enter their username or password. If the username or password are wrong a message will be displayed to the user to tell them they haven’t incorrectly entered their credentials.



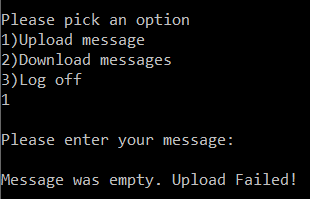
***Figure 3.5.7***

Figure 3.5.7 shows what happens if the user tries to choose an option number that doesn’t exist. They will be prompted to try again with a valid option number.



***Figure 3.5.8***

Figure 3.5.8 shows what happens if the user has no saved messages on the server. If no messages are found on the users’ file that is saved on the server, the system will display the message stating they have no messages available for download.



***Figure 3.5.9***

Figure 3.5.9 shows what happens when the user tries to upload an empty message. If the user clicks the enter button without entering any data, a message will be displayed to the user stating that the message was empty and that the upload failed.

# Conclusion

In conclusion, using sockets and threads to create a TMP was easily archivable. The result of this report was a working system that allowed messages to be sent across a secure connection to allow for different operations to occur. The end system successfully allowed users to log on, upload and download messages and log off while handling user or system errors. It would be very easy to add on more operations if needed in the future to make the system more robust. These could include registering users to the system.

# Bibliography

*Java67, 2021. How to load data from CSV file in Java - Example. [Online]   
Available at: https://www.java67.com/2015/08/how-to-load-data-from-csv-file-in-java.html*

*Stack Overflow, 2021. How to append text to an existing file in Java?. [Online]   
Available at: https://stackoverflow.com/questions/1625234/how-to-append-text-to-an-existing-file-in-java*

*Prototype Project(2015) Secure Sockets – Java Sockets Tutorial 06. [online video] Available at:  
https://www.youtube.com/watch?v=l4\_JIIrMhIQ  [Accessed 01/04/2021]*