Michael Edgar T00194492

Michael.edgar@students.ittralee.ie

Distributed Computing TMP Project

Twitter Messaging Protocol

# Summary

This report will outline a twitter messaging protocol that allows the user to securely upload and download short messages using TCP and SSL/TLS. The report will detail the protocol’s design and implementation, highlighting the format of the messages, the communication between processes and a pseudo code overview of the implementation of the requirements. The report will also include a number of diagrams to illustrate the design and implementation. The report will give instructions on use of the protocol, alongside sample screenshots illustrating the protocol in use.

# Contents

[1. Summary 1](#_Toc38299138)

[2. Contents 1](#_Toc38299139)

[3. Introduction 2](#_Toc38299140)

[4. Section 1: Protocol Design 3](#_Toc38299141)

[4.1 Message Description 3](#_Toc38299142)

[4.1.1 Message 100 Login 3](#_Toc38299143)

[4.1.2 Message 101 Upload 3](#_Toc38299144)

[4.1.3 Message 102 Download 3](#_Toc38299145)

[4.1.4 Message 103 Log Off 3](#_Toc38299146)

[4.2 Sequence Diagrams 4](#_Toc38299147)

[4.2.1 Login Sequence 4](#_Toc38299148)

[4.2.2 Upload Tweet Sequence 5](#_Toc38299149)

[4.2.3 Download Tweets Sequence 5](#_Toc38299150)

[4.2.4 Log Off Sequence 6](#_Toc38299151)

[4.3 Protocol Functionality 6](#_Toc38299152)

[4.3.1 Login 6](#_Toc38299153)

[4.3.2 Upload 6](#_Toc38299154)

[4.3.3 Download 6](#_Toc38299155)

[4.3.4 Log off 6](#_Toc38299156)

[4.4 Inter Process Communication 7](#_Toc38299157)

[5. Software Architecture and Implementation 7](#_Toc38299158)

[5.1 Source Files and Architecture Tiers 7](#_Toc38299159)

[5.2 UML Class Diagram 7](#_Toc38299160)

[5.3 Code Snippets 8](#_Toc38299161)

[5.4 Sample Client/Server Output 11](#_Toc38299162)

[5.5 SSL/TLS 12](#_Toc38299163)

[5.6 Instructions on Running Code 12](#_Toc38299164)

[6. Conclusion 12](#_Toc38299165)

[7. References 12](#_Toc38299166)

[8. Table of figures 12](#_Toc38299167)

# Introduction

This report aims to outline the design and implementation of a protocol using TCP stream sockets, along with SSL/TLS to securely transport short messages, called “tweets”, between a client and server. The protocol will allow contain four message types: login, upload, download and logoff. The first half of the report will outline the design of the protocol, outlining the format of each message, the functionality using sequence diagrams and pseudocode. The latter half of the report will show the implementation of the protocol, explaining the breakdown of the three software architecture tiers (presentation, application, and service) and showing code snippets and sample outputs.

# Section 1: Protocol Design

## Message Description

### Message 100 Login

Message: Login

Description: Message Code 100 sends a login request to the server. It requires a username and password as parameters. If the request is successful, the server will respond with a response code of 200.

Code: 100

Parameters: username, password (String text)

Response Message

Code: 200

Text: User Logged In

Response Message

Code: 404

Text: Missing Username or Password

Message Format: 100 [username] [password]

### Message 101 Upload

Message: Upload

Description: Message Code 101 sends an upload request to the server. It requires the tweet as a parameter. If the upload is successful, the server will respond with a response code of 200.

Code: 101

Parameters: tweet (String text)

Response Message

Code: 200

Text: Successful Upload

Response Message

Code: 404

Text: Missing Tweet Text

Response Message

Code: 400

Text: Error Uploading Tweet

Message Format: 101 [tweet]

### Message 102 Download

Message: Download

Description: Message Code 1002 sends a download request to the server. If the request is successful, the server will respond with all tweets on the server.

Code: 102

Parameters: NA

Response Message

Code: 200

Text: Tweet(s) (String text)

Response Message

Code: 404

Text: No Tweets Found

Message Format: 102

### Message 103 Log Off

Message: Log Off

Description: Message Code 103 sends a log off request to the server. It requires a username as a parameter. If the request is successful, the server will respond with a response code of 200.

Code: 103

Parameters: username (String text)

Response Message

Code: 200

Text: Successfully Logged User Off

Response Message

Code: 404

Text: User Not Found

Message Format: 103 [username]

## Sequence Diagrams

### A picture containing drawing Description automatically generatedLogin Sequence

Figure . SSL/TLS Handshake Explained (Cloudflare, 2019)

Figure . Login Message Sequence Diagram

A screenshot of a cell phone

Description automatically generated

### A picture containing drawing Description automatically generatedUpload Tweet Sequence

Figure . Upload Message Sequence Diagram

### A picture containing drawing Description automatically generatedDownload Tweets Sequence

Figure . Download Message Sequence Diagram

### Log Off Sequence

A picture containing drawing

Description automatically generated

Figure . Log Off Message Sequence Diagram

## Protocol Functionality

### Login

//Receive message 100 from the client

//Split the username and password within the message

//Save the username in the usernames ArrayList

//Return response code 200

### Upload

//Receive message 101 from the client

//Save the tweet in the tweets ArrayList

//Return response code 200

### Download

//Receive message 102 from the client

//Create StringBulder listOfTweets

//Start for loop from 0 to < tweets.size

//Add tweet at position i to listOfTweets with a comma to separate tweets

//If last tweet (i = tweets.size – 1), don’t add a comma

//Return response listOfTweets as a String

### Log off

//Receive message 103 from the client

//Search for username in usernames ArrayList

//if username is found, remove username from usernames and return response code 200

//if username not found, return response code 404 with message USERNOTFOUND

## Inter Process Communication

Inter process communication is achieved using connection-oriented stream sockets. Stream sockets transfer data between the client and server using a continuous data stream. Data is written by the process that controls the source and is read by the process controlling the destination. Each process creates a socket which is used to form the connection. The server creates a socket which then listens for connection requests from client processes. Once a connection request has been accepted, a data socket is created to read and write data to the data stream. This data socket is closed once the session is closed. Similarly, the client creates a socket, which sends a connection request to the server (Given, 2020).

# Software Architecture and Implementation

## Source Files and Architecture Tiers

The client and server consist of five source files: TwitterClient, TwitterClientHelper, MyStreamSocket, TwitterServer and TwitterServerThread. These files each handle separate parts of the software architecture. The presentation layer is handled within the TwitterClient and TwitterServer files, handling the client and server presentation, respectively. TwitterClient has the UI for the client application in the form of a command line interface and TwitterServer has the UI for the server application, similarly to the UI for the client. TwitterClientHelper and TwitterServerThread are responsible for the application layer of the software architecture. TwitterClientHelper sends requests from the user to the server and displays the response from the server to the user. TwitterServerThread performs the functionality for the various requests from the client, e.g. compiles all existing tweets and returns them to the client. Finally, MyStreamSocket is responsible for the service layer, using stream sockets and SSL for both the client and server.

## A close up of a logo Description automatically generatedUML Class Diagram

Figure . UML Class Diagram

## A screenshot of a computer Description automatically generatedCode Snippets

Figure . MyStreamSocket Class

A screenshot of a social media post

Description automatically generated

Figure . Twitter Client Login Authentication

A screenshot of a social media post

Description automatically generated

Figure . TwitterClient Main User Interface

A screenshot of a cell phone

Description automatically generated

Figure . TwitterClientHelper Login Method

A screenshot of a cell phone

Description automatically generated

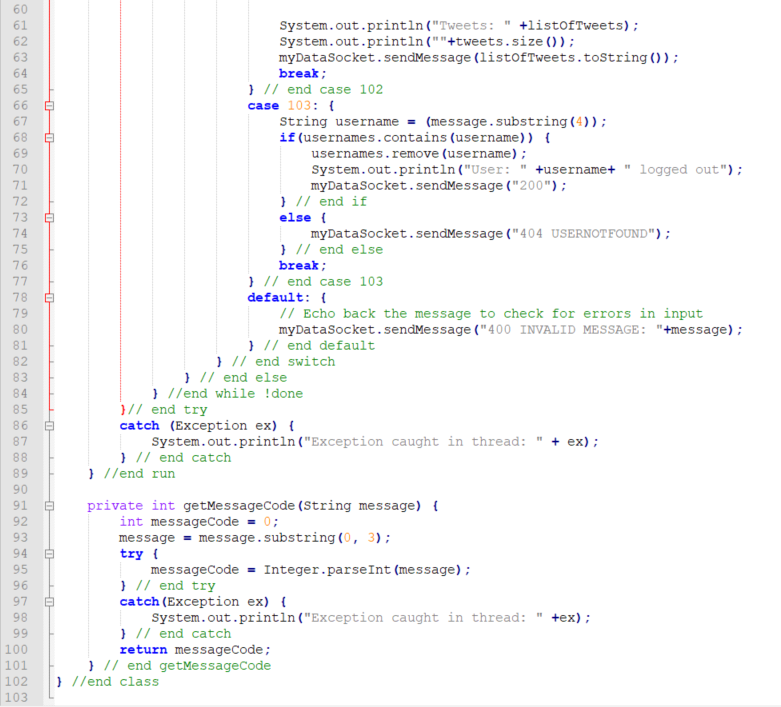
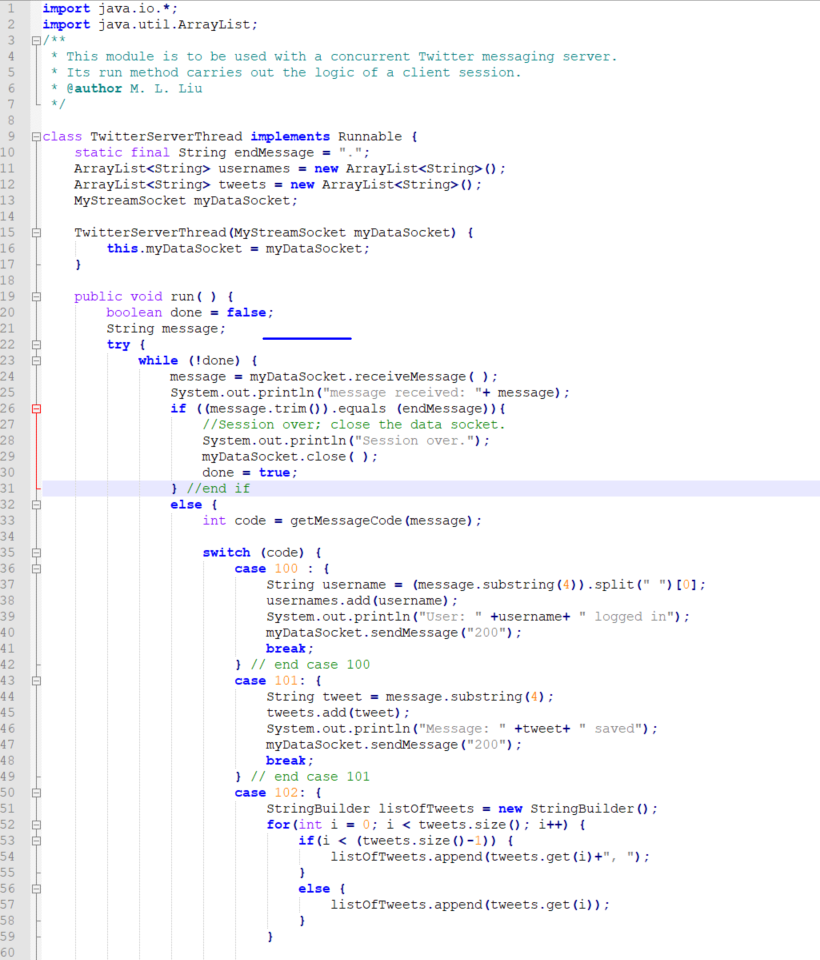
Figure . TwitterClient Helper Done Method

A screenshot of a social media post

Description automatically generated

Figure . Twitter Server SSLSocket Initialisation

Figure . TwitterServerThread Message Handling



A screenshot of a social media post

Description automatically generated

Figure . TwitterServerThread Message Code Parsing

## A screenshot of a cell phone Description automatically generatedSample Client/Server Output

Figure . Sample Client Output

A screenshot of a cell phone

Description automatically generated

Figure . Server Sample Output

## SSL/TLS

SSL or Secure Sockets Layer is a transport protocol for creating authenticated and encrypted links between networked computers. It was succeeded by the TLS (Transport Layer Security) protocol in 1999, but many still refer to them as “SSL” or “SSL/TLS”. The latest version of SSL/TLS is TLS 1.3, which was defined in August 2018 in RFC 8446. SSL/TLS is often used to identify websites or companies by pairing them with cryptographic keys using a digital document called a certificate. (Russell, 2019). This key/pair value can be used in SSL/TLS to authenticate connections between a client server. In the case of this protocol, this authentication allows a secure connection to be made between the client and server. The server registers a key and accesses this key when initializing the SSL socket. This allows the client to authenticate the server instance to the key, preventing an exception from occurring during the client server handshake process.

## Instructions on Running Code

There are two ways in which the code can be run. The first method is to open the Code folder in the file explorer. Run the serverCommand.bat file and the clientCommand.bat file and follow the onscreen instructions. The clientCommand.bat file can be run multiple times to set up multiple clients. The other method requires opening two or more (depending on the number of clients) command prompt windows. Use the cd command to go to the code folder and run the ‘serverCommand’ and ‘clientCommand’ commands on separate windows to set up the server and client, once again following the onscreen instructions.

# Conclusion

This report aimed to design and implement a short-message protocol using TCP stream sockets and SSL/TLS to handle communication between the client and server. The protocol design was used to implement the messaging protocol using Java, that could be run in a command line interface. Possible future improvements could include adding a graphical user interface to the client, moving the login authentication to the server or adding more response messages based upon certain edge cases.

# References

Cloudflare, 2019. *What Happens in a TLS Handshake? | SSL Handshake.* [Online]   
Available at: https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/  
[Accessed 17 April 2020].

Given, P., 2020. *Sockets NIO SSL 2020.* Tralee: Institute of Technology Tralee.

Russell, A., 2019. *What is SSL?.* [Online]   
Available at: https://www.ssl.com/faqs/faq-what-is-ssl/  
[Accessed 20 April 2020].

# Table of figures

[Figure 4.1 SSL/TLS Handshake Explained (Cloudflare, 2019) 4](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/DistributedComputing/CA1/T00194492MichaelEdgarTMP.docx#_Toc38299118)

[Figure 4.2 Login Message Sequence Diagram 4](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/DistributedComputing/CA1/T00194492MichaelEdgarTMP.docx#_Toc38299119)

[Figure 4.3 Upload Message Sequence Diagram 5](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/DistributedComputing/CA1/T00194492MichaelEdgarTMP.docx#_Toc38299120)

[Figure 4.4 Download Message Sequence Diagram 5](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/DistributedComputing/CA1/T00194492MichaelEdgarTMP.docx#_Toc38299121)

[Figure 4.5 Log Off Message Sequence Diagram 6](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/DistributedComputing/CA1/T00194492MichaelEdgarTMP.docx#_Toc38299122)

[Figure 5.1 UML Class Diagram 7](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/DistributedComputing/CA1/T00194492MichaelEdgarTMP.docx#_Toc38299123)

[Figure 5.2 MyStreamSocket Class 8](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/DistributedComputing/CA1/T00194492MichaelEdgarTMP.docx#_Toc38299124)

[Figure 5.3 Twitter Client Login Authentication 8](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/DistributedComputing/CA1/T00194492MichaelEdgarTMP.docx#_Toc38299125)

[Figure 5.4 TwitterClient Main User Interface 9](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/DistributedComputing/CA1/T00194492MichaelEdgarTMP.docx#_Toc38299126)

[Figure 5.5 TwitterClientHelper Login Method 9](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/DistributedComputing/CA1/T00194492MichaelEdgarTMP.docx#_Toc38299127)

[Figure 5.6 TwitterClient Helper Done Method 9](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/DistributedComputing/CA1/T00194492MichaelEdgarTMP.docx#_Toc38299128)

[Figure 5.7 Twitter Server SSLSocket Initialisation 9](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/DistributedComputing/CA1/T00194492MichaelEdgarTMP.docx#_Toc38299129)

[Figure 5.8 TwitterServerThread Message Handling 10](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/DistributedComputing/CA1/T00194492MichaelEdgarTMP.docx#_Toc38299130)

[Figure 5.9 TwitterServerThread Message Code Parsing 10](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/DistributedComputing/CA1/T00194492MichaelEdgarTMP.docx#_Toc38299131)

[Figure 5.10 Sample Client Output 11](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/DistributedComputing/CA1/T00194492MichaelEdgarTMP.docx#_Toc38299132)

[Figure 5.11 Server Sample Output 11](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/DistributedComputing/CA1/T00194492MichaelEdgarTMP.docx#_Toc38299133)