|  |
| --- |
| Implementation Generic Blue icon |
| Application Design & Implementation Document  Protocol Design & Proposed Implementation Software Architecture & Implementation |
| |  |  |  | | --- | --- | --- | | Deirdre Lee | 3/12/24 | Distributed Computing | |

Contents

[Part 1: Protocol Design 1](#_Toc162346166)

[1. Objectives and overview 1](#_Toc162346167)

[2. Design Philosophy 2](#_Toc162346168)

[3. Protocol for Client-Server Communication 2](#_Toc162346169)

[4. Pseudo-code for Functional Requirements 2](#_Toc162346170)

[4.1. Client Side 2](#_Toc162346175)

[4.2. Server Side 3](#_Toc162346176)

[5. Inter-process Communication 3](#_Toc162346181)

[5.1. Sequence of Inter-process Communication 3](#_Toc162346183)

[6. Conclusion 3](#_Toc162346184)

[Part 2: Application Implementation 3](#_Toc162346185)

[1. Objectives and Overview 4](#_Toc162346186)

[2. Source Files 4](#_Toc162346187)

[3. Tiers 4](#_Toc162346188)

[3.1. Application Tier 4](#_Toc162346189)

[3.2. Presentation Tier 4](#_Toc162346190)

[3.3. Session Tier 4](#_Toc162346191)

[4. UML Diagrams 4](#_Toc162346192)

[5. Sample Input & Output Data 4](#_Toc162346193)

[6. Running Instructions 4](#_Toc162346194)

[6.1. IDE Used 4](#_Toc162346195)

[6.2. Username/Password 5](#_Toc162346196)

[6.3. Dependencies 5](#_Toc162346197)

[6.4. Steps to Run: 5](#_Toc162346198)

[7. Conclusion 5](#_Toc162346199)

[8. Link to Project Repository 5](#_Toc162346200)

[References 5](#_Toc162346201)

# Part 1: Protocol Design

# Objectives and overview

The protocol aims to ensure secure, concurrent, reliable and error-free communication. It provides guidelines for implementation, ensuring the application meets the criteria for simplicity, security, and concurrency.

# Design Philosophy

This protocol facilitates communication between clients and a server prioritising simplicity, security and concurrency. It offers the following functionality: allow users to log in, upload messages to the server, download single or all messages from the server, log out, and quit the program. Security measures are implemented to ensure secure communication and authentication, Concurrency is achieved through multithreading, allowing the server to handle multiple client connections simultaneously.

# Protocol for Client-Server Communication

A secure connection will be established using Secure Sockets Layer/Transport Layer Security (SSL/TLS) protocol.

The client will communicate with the server using TCP/IP as outlined in the requirements.

The server will be equipped to manage concurrent communication through the use of multithreading.

Messages will be exchanged between the client and server in plaintext format. Messages will follow a predefined structure, as outlined both below and in the protocol document, with appropriate error messages provided for incorrect input.

SSL/TLS will be used to encrypt and decrypt communication between the client and the server for the duration of the session.

# 

# 

# Pseudo-code for Functional Requirements



## Client Side

* Connect to the server
* Establish an initial handshake to establish a secure connection
* Send requests to the server
* Receive responses from the server
* Close the connection ( LOGOUT ) when done

## Server Side

* Listen for incoming client connections
* Accept client connections and establish communication
* Receive requests from the client(s)
* Process requests and generate responses
* Send responses to the client(s)
* Close the connection with the client(s) when done



# Inter-process Communication

In the system, communication between the client and the server processes is facilitated through a secure connection established using SSL/TLS.



## Sequence of Inter-process Communication

* The client initiates a connection request to the server using SSL/TLS.
* The server listens for incoming client connections and accepts them.
* Upon successful connection establishment, the client and server perform an SSL/TLS handshake to establish a secure communication channel.
* The client acknowledges the server's acknowledgement over the secure connection.
* The client sends requests to the server.
* The server processes requests.
* The server sends back responses to the client.
* Communication continues until the session is terminated by either the client or the server.

# Conclusion

In conclusion, the protocol specification provides essential guidelines for secure and efficient communication between clients and server. It ensures procedures for authentication, error handling, concurrency, and termination.

# **Part 2: Application Implementation**

# Objectives and Overview

The application serves as a platform for secure and efficient client-server communication. Key functionalities include establishing secure connections, transmitting messages, managing user authentication, error handling, and supporting concurrent communication.

# Source Files

### Client.java

This file contains the main class for the client application. It handles user input, establishes connection with the server using SSL/TLS, sends requests, and receives responses. The class also implements error handling and logging functionalities to catch and report any exceptions or issues during communication process.

### ClientHelper.java

This class helps the client in handling operations such as login, sending and receiving messages, logging out, and quitting the application. It also manages the clients socket connection to the server, ensuring proper initialisation and error handling is implemented throughout the communication process.

### Server.java

This file contains the main class for the server application. It listens for incoming client connections, establishes SSL/TLS connections, and creates threads to handle multiple client communication.

### ServerThread.java

This class represents a thread on the server side responsible for handling communication with a single client. It implements the Runnable interface and overrides the run() method to execute the threads tasks. The class contains methods for processing client requests and generating responses.

### MyStreamSocket.java

This class provides a wrapper for Java socket functionality, enabling the handling of input and output streams in the communication process. It provides methods such as **sendMessage()** and **receiveMessage()** for sending and receiving messages, and **close()** for closing the socket.

# Tiers

## Application Tier

The application tier is responsible for executing the business logic and essential application functions. (TechTarget Contributor, 2021)

In this application, the classes that make up the application tier are the Server, MyStreamSocket and ClientHelper classes.

The Server class manages the creation of SSL/TLS sockets, handles incoming client connections and manages communication with clients via threads.

The MyStreamSocket class also facilitates communication between the client and server by managing input and output streams over the socket.

The ClientHelper class helps the client, providing methods for necessary operations, and contains the logic for establishing connections with the server.

## Presentation Tier

The presentation tier provides the interface for the application, allowing users to communicate with the system. (TechTarget Contributor, 2021) It establishes communication with the other tiers through user input and forwards requests for processing.

In this application, the class that fulfils this role is the Client class, which manages user interaction and initiates communication with the server. It provides a user interface through console input and output.

## Session Tier

The session tier, or data tier, typically manages reading and writing to a database or storage system. In this application, although there is no database involved, the ServerThread class fulfils a similar role. While its primary responsibility is managing communication with clients, it also handles individual client sessions, including the storage of the message and all related read and write operations.

# UML Diagrams

Include UML class diagrams representing the structure of the code…client and server

# Sample Input & Output Data

Include sample input/output data in the form of screenshots or textual representation.

# Running Instructions

## IDE Used

The IDE used when building, testing and running the application is VSCode. I have also run it using IntelliJ with no apparent issues.

## Username/Password

The login credentials for the application is lowercase ‘admin’ for the username and for the password. Authentication entails checking there are two words provided, as shown in Fig 1, and that they match the hardcoded values, as shown in Fig 2.

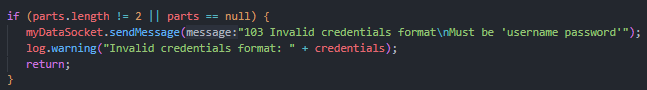


Fig 1



Fig 2

## Dependencies

No additional dependencies are needed for the application.

## Steps to Run:

Step 1: Description of the first step…

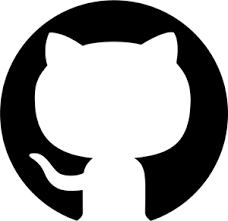
Step 2: Description of the second step…

Step 3: Description of the third step…

# Conclusion

Summarise the key points covered in the documentation…

# Link to Project Repository

 <https://github.com/EmoSense/distributed_computing>

C:\Users\Deirdre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\D63FA85E.tmp <https://deetralee.atlassian.net/jira/software/projects/DC/boards/3>

# References

**There are no sources in the current document.**