Software Requirements Specification

for

Multi-Client Chat Application

Version: 1.1

Prepared by Adventure Team
Margaret Reagan
Evie Kyritsis

16.35 Real-Time Systems and Software

Revision History

Name	Date	Reason For Changes	Version
Document Creation	5.2.2015	Creation of Document	1.0
Document Revisions	5.14.2015	Expanded/ modified requirements*: 3.2.1, 3.2.1.1.2, 3.2.1.1.3, 3.2.1.1.4, 3.2.2, 3.2.2.1.1, 3.2.2.1.2, 3.2.2.1.3, 3.2.2.2.2.2, 3.2.2.2.3, 3.2.3.1 (all)**, 3.3.1.1.1 (all), 3.3.1.1.2 (all), 3.3.1.2.1.3, 3.4 (all) Added: 3.2.1.2 (all), 3.2.2.1.4, 3.2.2.2.1, 3.2.2.2.3 (all), 3.2.3.2.5.2 (all), 3.3.1.2.1.4	1.1

* If the only change in the requirement was its numbering or an added/updated reference, it was not added to this list.
**(all) indicates all sub-sections within this section have been changed or added.

1. Introduction

1.1 Identification

This system is unique and is not a continuation of any specific past software system.

1.2 System overview

The purpose of the proposed system is to be a simple, text-based application between many clients that has the capability of being hosted on multiple computers, using a single server that is hosted on one computer.

The general system consists of a simple GUI interface for each client which communicates with the server hosted on either a client's own computer or a remote computer. The clients send

messages that are received by the server which then dispatches a single client's message to all of the clients that are connected to the server.

A chat log on the client GUI will allow a user of the system to view past messages sent in a given chat session. The message input of the client GUI will receive the keyboard input of the user. A general overview of the system structure can be seen in Figure 1 below. The rest of the components labeled in Figure 1 will be explained in Section 3.2 of this document.

This system was originally conceived as a final project for the MIT undergrad course: 16.35 - Real-Time Systems and Software and may be used by anyone who wishes to communicate via a chat client with other users of the system.

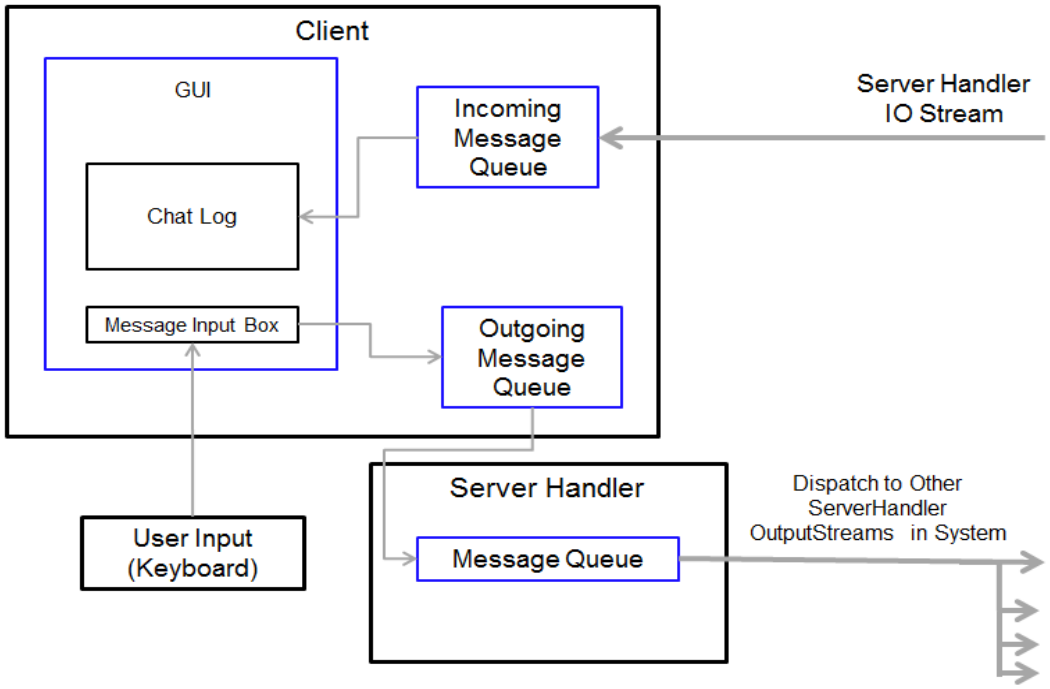

Figure 1: Messaging Flow Diagram - Details a broad overview of the path that messages take between a client and the server handlers.

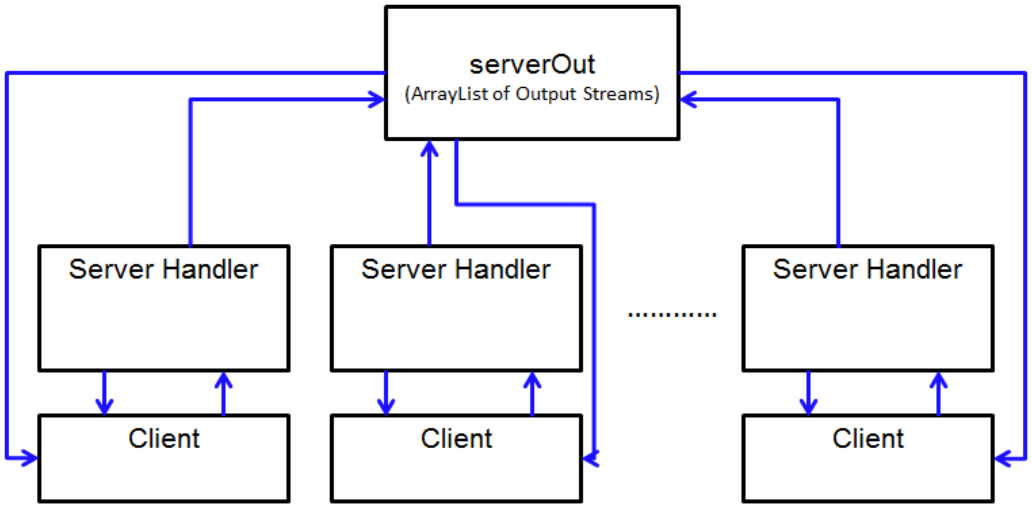

Figure 2: General System Diagram - Details a broad overview of the relationship between a client/server handler pair to the others. A server handler dispatches a message to the IO Output streams of the clients in the system.

1.3 Document overview

The intent of this document is to be a contract with the current 16.35 MIT professor, promising the capability and features of the system we introduced in Section 1.2. This document is thus intended to define the capability and features of a Single Server, Multiple Client instant messaging system that will be developed and executed using the Java programming language. This document enumerates the requirements, listed below under Section 3, used to develop the system, and is available for anyone with knowledge of object-oriented programming and development in Java who wishes to develop this same system or one similar.

1.4 Document Conventions

Error handling and qualification text for the methods and variables

2. References

1. ANSI/IEEE Std 830, IEEE standard for Software Requirements Specification documents.
2. Oracle Documentation - Package java.net
3. Oracle Documentation - Package java.io

3. Requirements

3.1 Required states and modes

There are no distinct states or modes that shall be required for this software.

3.2 Capability requirements

3.2.1 ChatServer

3.2.1.1 Variables

3.2.1.1.1 LISTENER_PORT - The internal computer address on which the server is connected and listening for services on shall be an integer between the values of [1, 65535].

3.2.1.1.2 serverOut - This list shall store each of the individual ServerHandler PrintWriters that send messages to the ChatClient (Section 3.2.3) along the socket IO output stream.

3.2.1.1.3 usernames- The ChatServer shall maintain a list of all the usernames of the users in the chatroom.

3.2.1.1.4 serverSocket- This shall be the internal representation of the ServerSocket a ChatClient connects to in the main method.

3.2.1.2 Methods

3.2.1.2.1 stopServer()

3.2.1.2.1.1 This method shall close *serverSocket* (Section 3.2.1.1.4).

3.2.1.2.1.2 Qualification - This method shall be qualified with test(s) that confirm the method's intended functionality.

3.2.1.2.2 run()

3.2.1.2.2.1 This method shall establish a *serverSocket* on the *LISTENER_PORT* (Section 3.2.1.1.1) and set a *serverSocket* timeout of 60 seconds;

3.2.1.2.2.2 This method shall listen for clients trying to connect to the ChatServer, and create a new ServerHandler (Section 3.2.2) for each ChatClient (Section 3.2.3) that tries to connect.

3.2.1.2.2.3 This method shall start each ServerHandler thread created and then set the *serverSocket* timeout to never timeout.

3.2.1.2.2.4 Qualification - This method shall be qualified with test(s) that confirm the method's intended functionality.

3.2.1.2.3 main(String [] args)

3.2.1.2.3.1 The main method shall construct and start a ChatServer

thread.

3.2.1.2.3.1 Qualification - This method shall be qualified through inspection that the ChatServer thread was created and started.

3.2.1.2.4 Error Handling - Any attempt to modify or set up a socket IO stream shall be surrounded by a try/catch block in order to catch any IOExceptions.

3.2.2 ServerHandler

3.2.2.1 Variables

3.2.2.1.1 messageQueue - This list that shall temporarily hold processed string messages received from individual clients until the time they are dispatched to all of the clients in the system.

3.2.2.1.2 serverOut - This list that shall store each of the individual ServerHandler PrintWriters that send messages to the ChatClient (Section 3.2.3).

3.2.2.1.3 usernames - Each ServerHandler shall maintain a list of all the usernames of the users in the chatroom.

3.2.2.1.4 chatServer - This variable shall represent the ChatServer (Section 3.2.1) which spawned the ServerHandler.

3.2.2.2 Methods

3.2.2.2.1 Constructor(Socket socket, ArrayList<PrintWriter> writers, ArrayList<String> names, ChatServer chatServer**)**

3.2.2.2.1.1 This method shall initialize the ServerHandler with the accepted Socket connection to the *serverSocket* (Section 3.2.1.1.4), the list of ServerHandler PrintWriters *serverOut* (Section 3.2.1.1.2), the list of usernames *usernames* (Section 3.2.1.1.3), and the ChatServer that spawned it *chatServer* (Section 3.2.2.1.4).

3.2.2.2.1.2 Qualification- This method shall be qualified with test(s) that confirm the method's intended functionality.

3.2.2.2.1 addToMsgQueue(String new_message**)**

3.2.2.2.1.1 This method shall be synchronized.

3.2.2.2.1.2 This method shall take incoming messages from the server socket IO input stream and add them to the end of the *messageQueue* (Section 3.2.2.1.1).

3.2.2.2.1.3 This method shall notify waiting processes once it has completed 3.2.2.2.1.2.

3.2.2.2.1.4 Qualification- This method shall be qualified with test(s) that confirm the method's intended functionality.

3.2.2.2.2 sendMessage() -

3.2.2.2.2.1 This method shall be synchronized.

3.2.2.2.2.2 This method shall pop the first message off queue and send it to all of the ServerHandler output streams in *serverOut* (Section 3.2.2.1.2).

3.2.2.2.2.2 If the *messageQueue* is larger than one message, it shall send all of the messages before relinquishing the hold on the message queue.

3.2.2.2.2.3 If no messages are in the queue, the server shall wait for new messages.

3.2.2.2.2.4 Error Handling - Shall throw an InterruptedException

3.2.2.2.2.5 Qualification- This method shall be qualified with test(s) that confirm the method's intended functionality.

3.2.2.2.3 setUsername()-

3.2.2.2.3.1 This method shall check whether or not the client has a username.

3.2.2.2.3.2 If user does not yet have a username, this method shall prompt the ChatClient (Section 3.2.3) to ask the user for their username by sending the command "USERNAME" along the ServerHandler's output stream.

3.2.2.2.3.3 The method shall receive the user's username from the socket input stream and check that username against *usernames* (Section 3.2.2.1.3), the list of usernames in the chat room.

3.2.2.2.3.4 If the username is valid (i.e. not taken by anyone else), then under synchronization the username shall be added to the list of usernames.

3.2.2.2.3.5 Qualification- This method shall be qualified with test(s) that confirm the method's intended functionality.

3.2.2.2.3 run()-

3.2.2.2.3.1 This method shall establish the IO input and output streams from the socket.

3.2.2.2.3.2 This method shall call the method *setUsername* (Section 3.2.2.2.3) in order to begin a "handshake" with the client.

3.2.2.2.3.3 Once the username has been set, this method shall print out a short string message to the client to affirm the handshake.

3.2.2.2.3.4 This method shall add the output stream of the ServerHandler to the list of output streams of all serverHandlers in the system, *serverOut* (Section 3.2.2.1.2).

3.2.2.2.3.5 The run method shall implement *addToMsgQueue()* (Section 3.2.2.2.2) and *sendMessage*() (Section 3.2.2.2.1) in order to alert other clients of the new client's entrance into the current chat session.

3.2.2.2.3.6 This method shall be waiting for incoming messages from other IO output streams in *serverOut*, process them, and add them to the message queue.

3.2.2.2.3.7 When the client is closed, this method shall remove the username of the client from *usernames* (Section 3.2.2.1.3).

3.2.2.2.3.7.1 If there are no more usernames left, then the ServerHandler shall close the ChatServer *serverSocket* (Section 3.2.1.1.4).

3.2.2.2.3.8 When the client is closed, this method shall remove the output stream of the client from *serverOut*.

3.2.2.2.3.9 When the client is closed, this method shall notify the other users that the client has left the chat session

3.2.2.2.3.10 When the client is closed, this method shall close the socket.

3.2.1.2.3.11 Error Handling - Shall catch IOExceptions and InterruptedExceptions

3.2.1.2.3.12 Qualification This functionality shall be qualified via demonstration of capability.

3.2.3 ChatClient

3.2.3.1 Variables

3.2.3.1.1.1 inputMsgQ - This list shall temporarily hold processed string messages received from the server until the time they are to be displayed on the GUI.

3.2.3.1.1.2 outputMsgQ - This list shall temporarily hold processed string messages received from the GUI until the time they are to be sent to the server.

3.2.3.1.1.3 clientOut - This variable shall be the ChatClient's internal representation of the output IO stream that is data that the ChatClient sends out.

3.2.3.1.1.4 clientIn - This variable shall be the ChatClient's internal representation of the input IO stream that is receiving data into the ChatClient.

3.2.3.1.1.5 SERVER_PORT - Shall represent the port that the socket is to be established on and on which the server is awaiting client connections.

3.2.3.2 Methods

3.2.3.2.1 Constructor() - There shall be no requirement on the constructor arguments of the ChatClient.

3.2.3.2.2 addToOutputMsgQ(String message**)**

3.2.3.2.2.1 This method shall be synchronized.

3.2.3.2.2.2 This method shall take an input string as a message and add it to the *outputMsgQ* (Section 3.2.3.1.1.2).

3.2.3.2.2.3 This message shall notify any waiting processes

once it has completed 3.2.3.2.2.2.

3.2.3.2.2.4 Qualification This method shall be qualified with test(s) that confirm the method's intended functionality.

3.2.3.2.3 printFromOutputMsgQ()

3.2.3.2.3.1 This method shall be synchronized.

3.2.3.2.3.2 This method shall pop the first message off the *outputMsgQ* and print the message to the socket IO output stream of the client *clientOut* (Section 3.2.3.1.1.3).

3.2.3.2.3.3 If the *outputMsgQ* is larger than one message, this method shall send all of the messages before relinquishing the hold on the message queue.

3.2.3.2.3.4 If no messages are in *outputMsgQ,* the client shall wait for new messages.

3.2.3.2.3.5 Error Handling - This method shall catch InterruptedExceptions.

3.2.3.2.3.6 Qualification This method shall be qualified with test(s) that confirm the method's intended functionality.

3.2.3.2.4 addToInputMsgQ(String message**)**

3.2.3.2.4.1 This method shall be synchronized.

3.2.3.2.4.2 This method shall take the incoming socket IO input stream *clientIn* (Section 3.2.3.1.1.4) and add any messages to the *inputMsgQ* (Section 3.2.3.1.1.1).

3.2.3.2.4.3 This method shall notify any waiting threads once it has completed 3.2.3.2.4.2.

3.2.3.2.4.4 Qualification This method shall be qualified with test(s) that confirm the method's intended functionality.

3.2.3.2.5 displayFromInputMsgQ()-

3.2.3.2.5.1 This method shall be synchronized.

3.2.3.2.5.2 This method shall pop the first message off *inputMsgQ* and display the message on the client GUI (Section 3.2.2.2) depending on the starting word in the input message.

3.2.3.2.5.2.1 If the incoming message begins with "USERNAME", the client shall ask the user for their desired username and print the desired username along *clientOut.*

3.2.3.2.5.2.2 If the incoming message begins with "SOUNDSGOOD", the client shall allow the new user to type messages in the GUI.

3.2.3.2.5.2.3 If the incoming message begins with "MESSAGE",

then the GUI shall print the subsequent message along *clientOut*.

3.2.3.2.5.2.3 If the incoming message begins with "NEWUSER," then the GUI shall announce to the chat room that a new user has arrived.

3.2.3.2.5.3 If the *inputMsgQ* is larger than one message, it shall display all of the messages before relinquishing the hold on the message queue.

3.2.3.2.5.4 If no messages are in the queue, the ChatClient shall wait for new messages.

3.2.3.2.5.5 Qualification This functionality shall be qualified via demonstration of capability.

3.2.3.2.6 run() -

3.2.3.2.6.1 The run method of the ChatClient shall initially establish a socket connection with the server.

3.2.3.2.6.2 The run method shall continuously read the IO input and output streams of the socket, *clientIn* and *clientOut*.

3.2.3.2.6.3 It shall take any incoming messages on the IO input stream *clientIn*, run *addToInputMsgQ* (Section 3.2.3.2.4) on any incoming messages, and run *displayFromInputMsgQ* (Section 3.2.3.2.5).

3.2.3.2.6.4 Error Handling - This method shall catch IOExceptions.

3.2.3.2.6.5 Qualification This functionality shall be qualified via demonstration of capability.

3.2.4 Client GUI

3.2.4.1 Chat Log

3.2.4.1.1 The chat log shall display at minimum the last 10 lines of text from messages that were most recently received.

3.2.4.1.2 The GUI shall show which client sent which message by displaying the client's username before the message.

3.2.4.1.3 Next to each message, the time in which it was sent shall be displayed.

3.2.4.1.4 Qualification This functionality shall be qualified via demonstration of capability.

3.2.4.2 Message Input

3.2.4.2.1 The GUI shall display the keyboard input as the user is typing in a text field.

3.2.4.2.2 The text field shall have an actionListener attached to it.

3.2.4.2.2.1 When the user hits enter, an ActionHandler shall send the message to the client's *outputMsgQ* (Section 3.2.3.1.1.2) with the method *addToOutputMsgQ* (Section 3.2.3.2.2).

3.2.4.2.2.2 After completing section 3.2.4.2.2.1, the ActionHandler shall call *printFromMsgQ* (Section 3.2.3.2.2).

3.2.4.2.2.3 Qualification This method shall be qualified with test(s) that confirm the method's intended functionality.

3.2.4.2.3 When the user hits enter, the previous text shall be cleared from the text field.

3.2.4.2.4 Qualification This functionality shall be qualified via demonstration of capability.

3.3 Internal interface requirements

3.3.1 Synchronization

3.3.1.1. Shared resources

This system has multiple shared resources that shall only be modified within a synchronized block.

3.3.1.1.1 ChatServer

3.3.1.1.1.1 usernames (Section 3.2.1.1.3)

3.3.1.1.1.2 serverOut (Section 3.2.1.1.4)

3.3.1.1.2 ServerHandler

3.3.1.1.2.1 messageQueue (Section 3.2.2.1.1)

3.3.1.1.2.2 usernames (Section 3.2.2.1.3)

3.3.1.1.2.3 serverOut (Section 3.2.2.2.1)

3.3.1.1.3 ChatClient

3.3.1.1.3.1 inputMsgQ (Section 3.2.3.1.1.1)

3.3.1.1.3.2 outputMsgQ (Section 3.2.3.1.1.2)

3.3.1.2. Synchronized methods

3.3.1.2.1 ServerHandler

3.3.1.2.1.1 addToMsgQueue() (Section 3.2.2.2.1) shall modify *messageQueue* (Section 3.3.1.1.2.1) and shall notify *sendMessage()* (see 3.3.1.2.1.2).

3.3.1.2.1.2 sendMessage() (Section 3.2.2.2.2) shall modify m*essageQueue* and shall wait for *addToMsgQ()* (see 3.3.1.2.1.1).

3.3.1.2.1.3 setUsername() (Section 3.2.2.2.3) shall modify *usernames* (Section 3.3.1.1.2.2) under a synchronized block.

3.3.1.2.1.4 serverOut - The modification of *serverOut* (Section 3.2.2.2.1) shall be implemented in the run method. Any new method or functionality that modifies *serverOut* shall be synchronized.

3.3.1.2.2 ChatClient

3.3.1.2.2.1 addToInputMsgQ() (Section 3.2.3.2.4) shall modify *inputMsgQ* (Section 3.3.1.1.3.1) and shall notify *displayFromInputMsgQ()* (see 3.3.1.2.2.2)

3.3.1.2.2.2 displayFromInputMsgQ()(Section 3.2.3.2.5) shall Modify *inputMsgQ* and shall wait for *addToInputMsgQ()* (see 3.3.1.2.2.1)

3.3.1.2.2.3 addToOutputMsgQ()(Section 3.2.3.2.2) shall modify *outputMsgQ* (Section 3.2.3.1.1.2) and shall notify *printFromOutputMsgQ()* (see 3.3.1.2.2.4)

3.3.1.2.2.4 printFromOutputMsgQ()(Section 3.2.3.2.2) shall modify *outputMsgQ* and shall wait for *addToOutputMsgQ()* (see 3.3.1.2.2.2)

3.4 External interface requirements

3.4.1 Socket protocol

This system makes use of the Java Socket object in order to establish a low-level TCP communication between the Server and multiple server handlers and clients.

3.4.1.1 Upon startup, the server shall create a ServerSocket *serverSocket* (Section 3.2.1.1.4) on the LISTENER_PORT (Section 3.2.1.1.1)

3.4.1.2 For each ChatClient (Section 3.2.3) thread that attempts to connect to the *serverSocket*, a ServerHandler (Section 3.2.2) thread shall be created and started to accept the socket connection.

3.4.1.3 The first message the ServerHandler sends to the ChatClient shall be the message “USERNAME,” prompting the ChatClient to ask the user for their username.

3.4.1.4 The ChatClient shall send a username back along the socket IO stream for the the ServerHandler to check that a valid username has been chosen.

3.4.1.5 If a valid (not taken) username has been chosen, the ServerHandler shall send back a “SOUNDSGOOD” message along the socket IO stream. If an invalid username is chosen, the ChatClient shall continue to prompt the user for a valid username.

3.4.1.6 Once a ChatClient is validated, they shall be able to send messages along the socket IO stream with the header “MESSAGE” which is placed in front of any message input to the GUI.

3.4.1.7 The ServerHandler shall read messages from its ChatClient and append the “MESSAGE” header to the front of the message before sending it to the ChatClient socket IO input stream.

3.4.2.8 Once these actions have been completed, the ChatClient shall only print messages from the ServerHandler that start with the "MESSAGE" or "NEWUSER" header.

3.4.2 Closing the socket

This system shall follow a strict procedure in closing any sockets or ServerSockets.

3.4.2.1 Upon exit of the ChatClient GUI, the corresponding ServerHandler shall remove the user from the list of usernames and shall remove the ServerHandler output stream from the list *serverOut* (Section 3.2.2.1.2).

3.4.2.2 If the user exiting the ChatClient is the last user in the chatroom, the *serverSocket* shall close after they leave.

3.4.2.3 If a ChatClient does not connect to the ServerSocket within 60 seconds of the ServerSocket startup, the s*erverSocket* shall close.

3.5 Internal data requirements

The messages sent between client/server and vice versa shall be java string data. A string in java is an internal array, and arrays (and thus strings) shall have a maximum size of 2^31 -1.

3.6 Environment requirements

The device running the application shall have at least a Java SE 1.6 runtime environment or later version.

3.7 Computer software requirements

The computer running this program shall have java installed, shall be able to execute and compile java, and shall be able to connect to the internet.