Member 1 : _____ Section: _____

Member 2 : _____

1.0 Machine Project Specifications – Message Board System

Hello Class, the final output of this course would be a **Message Board System**, allowing clients to be able to communicate with other clients through the server using the UDP protocol. The Message Board System would need to be comprised to a server application and a client application. The project must be done either individually or by pair. You may also choose to use either C, Java, or Python programming languages for this project.

Client Application Specifications:

1. The client application would function as the User Interface of a user when using the Message Board System.
2. The client application should contain an input field to allow the following input commands:

Description	Input Syntax	Sample Input Script
Connect to the server application	`/join <server_ip_add> <port>`	`/join 192.168.1.1 12345`
Disconnect to the server application	`/leave`	`/leave`
Register a unique handle or alias	`/register <handle>`	`/register Student1`
Send message to all	`/all <message>`	`/all Hello World!`
Send direct message to a single handle	`/msg <handle> <message>`	`/msg Student1 Hello!`
Request command help to output all Input Syntax commands for references	`/?`	`/?`

3. The client application should also contain an output area to display messages from other users as well as system messages from the interaction of the client and the server application.

Description	Sample Output Script
Message upon successful connection to the server	`Connection to the Message Board Server is successful!`
Message upon successful disconnection to the server	`Connection closed. Thank you!`
Message upon successful registration of a handle or alias	`Welcome Student1!`
Message upon successful sending of a message to all (i.e. Student1 is sending a message "Hello World!" to all clients)	`Student1: Hello World!`
Message upon successful sending of a direct message (i.e. Student1 is sending a message "Hello" to Student2)	`[To Student2]: Hello`
Message upon successful receipt of a direct message (i.e. Student2 receives a message "Hello" from Student1)	`[From Student1]: Hello`

4. The client application should be also be able to display error messages

Description	Sample Output Script
Message upon unsuccessful connection to the server due to the server not running or incorrect IP and Port combination	`Error: Connection to the Message Board Server has failed! Please check IP Address and Port Number.`
Message upon unsuccessful disconnection to the server due to not currently being connected	`Error: Disconnection failed. Please connect to the server first.`
Message upon unsuccessful registration of a handle or alias due to registered "handle" or alias already exists	`Error: Registration failed. Handle or alias already exists.`
Message upon unsuccessful sending of a direct message (i.e. Student1 is sending a message to a non-existing handle)	`Error: Handle or alias not found.`
Message due to command syntax	`Error: Command not found.`
Message due to incorrect or invalid parameters	`Error: Command parameters do not match or is not allowed.`

5. The client application should be able to receive commands from the user following the identified input commands and parameters, but should be able to convert the user commands into JSON format when communicating with the server application.

Server Application Specifications:

1. The server application would function as the service or program where client applications would connect to, in order to interact with other clients in the Message Board Application

2. The server application should only communicate with the client application through the use of JSON format containing commands and parameters

Description	Input Syntax
Connect to the server application	`{"command":"join"}`
Disconnect to the server application	`{"command":"leave"}`
Register a unique handle or alias	`{"command":"register", "handle":"<handle>"}`
Send message to all	`{"command":"all", "message":"<message>"}`
Send direct message to a single handle	`{"command":"msg", "handle":"<handle>", "message":"<message>"}`
Return error message	`{"command":"error", "message":"<error_message>"}`

3. Since the server application would be using JSON format for sending and receiving commands and parameters with the client application, this would mean that your work should be interoperable with the works of your other classmates, despite using different programming languages

Note that additional features and functionalities may be considered as bonus points only when all of the requirements have been accomplished correctly. Additional features include a Graphical User Interface, support for groups or channels as well as multi-cast communication, support for emojis, among others. Additional bonus may not exceed 20% of the total score for the machine project, and excess marks may be reallocated to other assessments in the course.

The rubrics for grading as well as the demo kit would be provided to you on a latter date. The rubrics would include the breakdown of points for each identified functionality, as well as would include the scope for possible bonus points. Your instructor would deploy, test, and evaluate your projects using the submitted server and client application, however a project demo may also be requested by your instructor.

The due date of the machine project would be on December 12, 11:59PM, with the last day of accepting late submissions by December 15 11:59PM, resulting a deduction of 10% per day from the due date. You may also opt to submit your machine project early, on or before December 7, 11:59PM, gaining at most an additional 10% bonus points for early submission (to be subject to the quality of output and decision of your instructor) assuming that all required functionalities are working as intended.

No submission or incomplete submission would result in a grade of 0.0 for this assessment.

To aid you in understanding how to implement a basic client-server chat application, you may use the following tutorials and examples for reference:

https://pythontic.com/modules/socket/udp-client-server-example
https://www.studytonight.com/network-programming-in-python/working-with-udp-sockets
https://tutorialedge.net/python/udp-client-server-python/
https://github.com/ratanak1010/Java-UDP-Chat
https://www.geeksforgeeks.org/working-udp-datagramsockets-java/

After completing the machine problem, don't forget to save and submit both the client and the server application in AnimoSpace. If you are using any special libraries for your applications, include a list of those libraries and their respective versions for reference during the testing.

Have fun!