Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-003-F2024/it114-module-5-project-milestone-1/grade/mcp62

Course: IT114-003-F2024
Assigment: [IT114] Module 5 Project Milestone 1
Student: Michael P. (mcp62)

Submissions:

Submission Selection

1 Submission [submitted] 10/21/2024 11:09:11 PM

Instructions

COLLAPSE

Overview Video: https://youtu.be/A2yDMS9TS1o

1. Create a new branch called Milestone1
2. At the root of your repository create a folder called Project if one doesn't exist yet
    1. You will be updating this folder with new code as you do milestones
    2. You won't be creating separate folders for milestones; milestones are just branches
3. Copy in the code from Sockets Part 5 into the Project folder (just the files)
    2. https://github.com/MattToegel/IT114/tree/M24-Sockets-Part5
4. Fix the package references at the top of each file (these are the only edits you should do at this point)
5. Git add/commit the baseline and push it to github
6. Create a pull request from Milestone1 to main (don't complete/merge it yet, just have it in open status)
7. Ensure the sample is working and fill in the below deliverables 1. Note: Don't forget the client commands are /name and /connect
8. Generate the output file once done and add it to your local repository
9. Git add/commit/push all changes
10. Complete the pull request merge from the step in the beginning
11. Locally checkout main
12. git pull origin main

Branch name: Milestone1

Group

100%

Group: Start Up
Tasks: 2
Points: 3

COLLAPSE

Task

100%

Group: Start Up
Task #1: Start Up
Weight: ~50%
Points: ~1.50

COLLAPSE

Details:

Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.

Columns: 1

Sub-Task

100%

Group: Start Up
Task #1: Start Up
Sub Task #1: Show the Server starting via command line and listening for connections

Task Screenshots

Gallery Style: 2 Columns

4 2 1

server connecting and waiting for client

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: Start Up
Task #1: Start Up
Sub Task #2: Show the Server Code that listens for connections

Task Screenshots

Gallery Style: 2 Columns

4 2 1

code that starts port

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code related to starting up and waiting for connections

Response:

The start(int port) method sets up a server that listens for client connections on a specified port. It creates a ServerSocket and initializes a lobby room for clients. The method enters a loop to accept incoming connections, logging each one, and spawns a new ServerThread for each client to handle interactions concurrently. It includes error handling for connection issues and ensures proper resource cleanup upon shutdown.

Sub-Task

100%

Group: Start Up

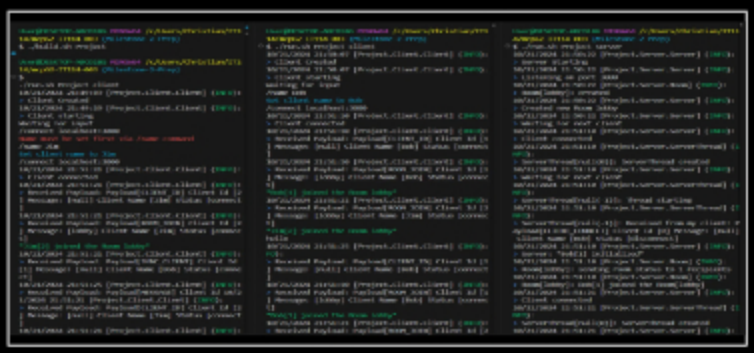
Task #1: Start Up

Sub Task #3: Show the Client starting via command line

Task Screenshots

Gallery Style: 2 Columns

4 2 1



shows two terminals on the left are the client

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: Start Up

Task #1: Start Up

Sub Task #4: Show the Client Code that prepares the client and waits for user input

Task Screenshots

Gallery Style: 2 Columns

4 2 1

code that checks connection

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow leading up to and including waiting for user input

Response:

The connect(String address, int port) method establishes a connection to a server at the specified address and port. It creates input and output streams for communication and logs a success message upon connection. The method runs listenToServer() in a separate thread to handle incoming messages concurrently. It includes error handling for connection issues and returns the connection status.

End of Task 1

Task

100%

Group: Start Up
Task #2: Connecting
Weight: ~50%
Points: ~1.50

COLLAPSE

Details:

Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.

Columns: 1

Sub-Task

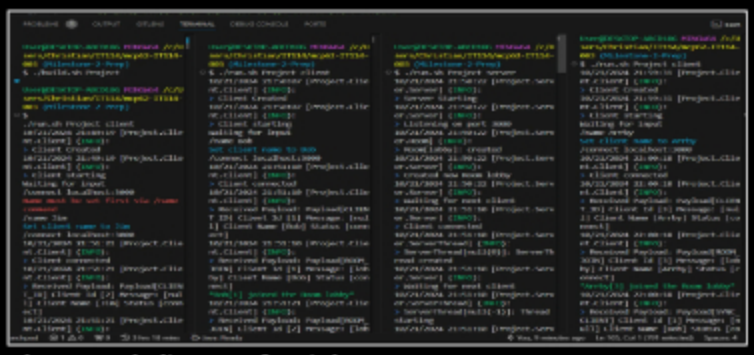
100%

Group: Start Up
Task #2: Connecting
Sub Task #1: Show 3 Clients connecting to the Server

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Shows 3rd client on far right

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: Start Up

Task #2: Connecting

Sub Task #2: Show the code related to Clients connecting to the Server (including the two needed commands)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

shows process commands for name

shows regex pattern for local host connectivity

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow

Response:

initializes a logging utility for the client, configuring it to limit log file size to 2MB, keep one file, and store logs in "client.log." It declares several member variables: a Socket for the server connection, output and input streams for communication, and patterns to match connection commands for both IP addresses and localhost. Additonally creates a regex patter to verify localhost connect command. Proccess commands goes over /name functionality

End of Task 2

End of Group: Start Up
Task Status: 2/2

Group

100%

Group: Communication
Tasks: 2
Points: 3

COLLAPSE

Task

100%

Group: Communication
Task #1: Communication
Weight: ~50%
Points: ~1.50

COLLAPSE

Details:
Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.

Columns: 1

Sub-Task

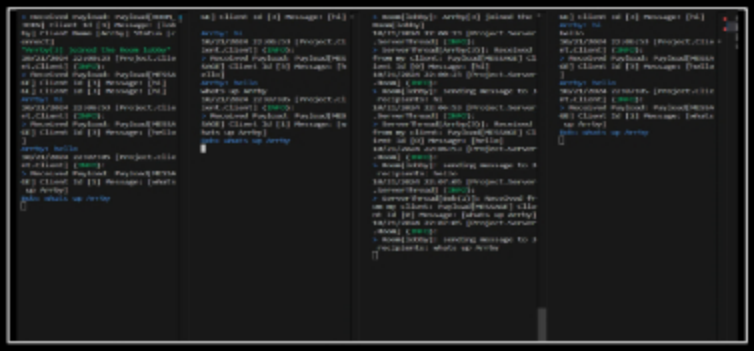
100%

Group: Communication
Task #1: Communication
Sub Task #1: Show each Client sending and receiving messages

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Shows clients sending and receiving

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

Group: Communication

100%

Task #1: Communication

Sub Task #2: Show the code related to the Client-side of getting a user message and sending it over the socket

Task Screenshots

Gallery Style: 2 Columns

4 2 1

send over socket w payload

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

send(Payload payload) method attempts to send a Payload object to the client using the output stream (out). If the client is not running, it returns true immediately. Upon successful writing and flushing of the object, it returns true. If an IOException occurs (indicating a potential disconnection), it logs an error message, calls a cleanup method to handle the disconnection, and returns false.

Sub-Task

100%

Group: Communication

Task #1: Communication

Sub Task #3: Show the code related to the Server-side receiving the message and relaying it to each connected Client

Task Screenshots

Gallery Style: 2 Columns

4 2 1

server side relay

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

The sendMessage(long senderId, String message) method constructs a Payload object to represent a message being sent by a client. It sets the client's ID, the message content, and specifies the payload type as a message. Finally, it calls the send(p) method to transmit the payload, facilitating the communication from the client to the server or other clients.

Sub-Task

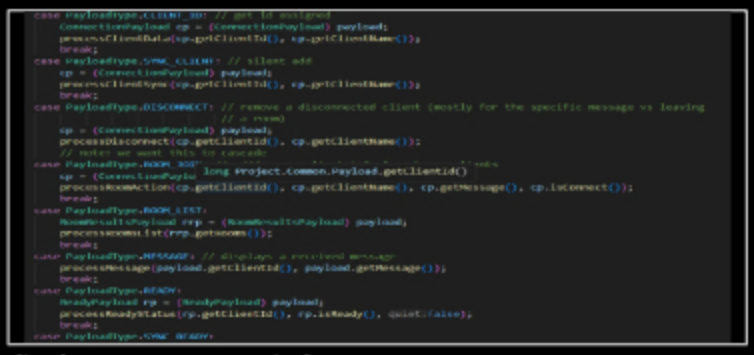
100%

Group: Communication
Task #1: Communication
Sub Task #4: Show the code related to the Client receiving messages from the Server-side and presenting them

Task Screenshots

Gallery Style: 2 Columns

4 2 1



displays message switch case

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

The processPayload(Payload payload) method handles different types of incoming payloads by logging the received data and processing it based on its type using a switch statement. Each case triggers specific actions, such as updating client information, managing room actions, displaying messages, or changing readiness statuses, with dedicated methods handling the actual logic for each type.

End of Task 1

Task

100%

Group: Communication
Task #2: Rooms
Weight: ~50%
Points: ~1.50

COLLAPSE

Details:

Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.

Columns: 1

Sub-Task

100%

Group: Communication

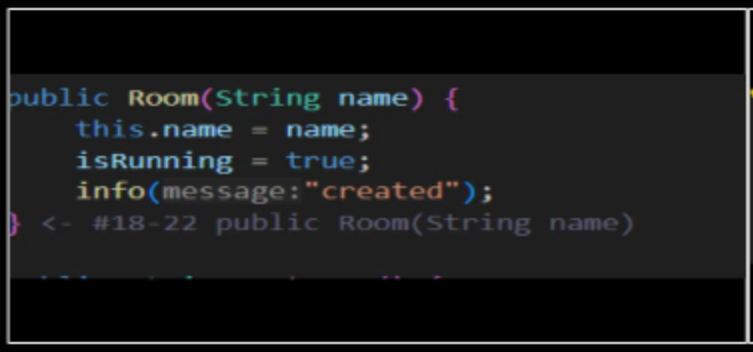
Task #2: Rooms

Sub Task #1: Show Clients can Create Rooms

Task Screenshots

Gallery Style: 2 Columns

4 2 1



room constructor

cases for join and create room

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: Communication

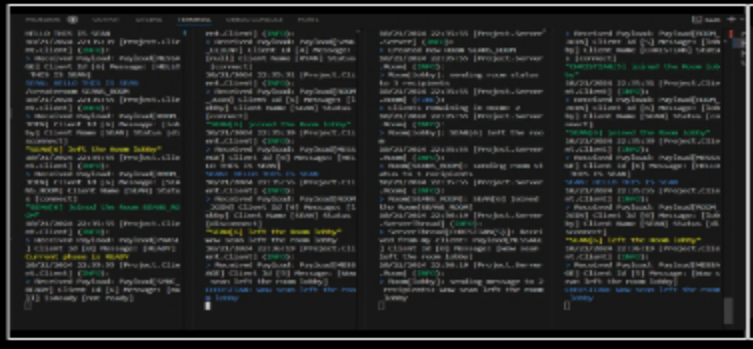
Task #2: Rooms

Sub Task #2: Show Clients can Join Rooms (leave/join messages should be visible)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Shows SEAN leaving room and not getting message in lobby

Shows christian joining seans room and ryan not getting

lobby

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

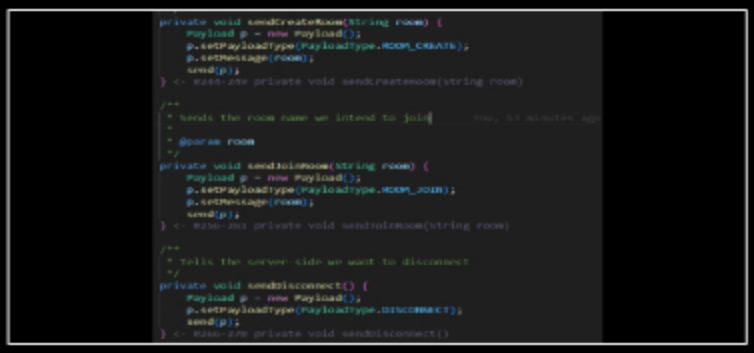
Sub-Task 100%

Group: Communication
Task #2: Rooms
Sub Task #3: Show the Client code related to the create/join room commands

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
private void sendCreateRoom(String room) {
    Payload p = new Payload();
    p.setPayloadType(PayloadType.ROOM_CREATE);
    p.setMessage(room);
    send(p);
}

/**
 * Sends the room name we intend to join
 *
 * @param room
 */
private void sendJoinRoom(String room) {
    Payload p = new Payload();
    p.setPayloadType(PayloadType.ROOM_JOIN);
    p.setMessage(room);
    send(p);
}

/**
 * Tells the server-side we want to disconnect
 */
private void sendDisconnect() {
    Payload p = new Payload();
    p.setPayloadType(PayloadType.DISCONNECT);
    send(p);
}
```

Client side create, join and leave methods

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

sendCreateRoom(String room): Constructs a Payload indicating a request to create a new room with the specified name and sends it.

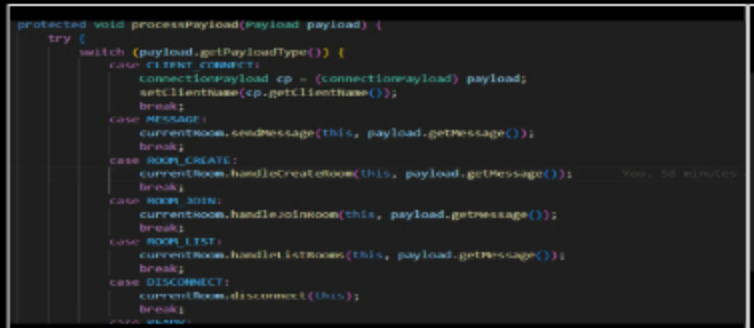
Sub-Task 100%

Group: Communication
Task #2: Rooms
Sub Task #4: Show the ServerThread/Room code handling the create/join process

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
protected void processPayload(Payload payload) {
    try {
        switch (payload.getPayloadType()) {
            case CLIENT_CONNECT:
                ConnectionPayload cp = (ConnectionPayload) payload;
                setClientName(cp.getClientName());
                break;
            case MESSAGE:
                currentRoom.sendMessage(this, payload.getMessage());
                break;
            case ROOM_CREATE:
                currentRoom.handleCreateRoom(this, payload.getMessage());
                break;
            case ROOM_JOIN:
                currentRoom.handleJoinRoom(this, payload.getMessage());
                break;
            case ROOM_LIST:
                currentRoom.handleListRooms(this, payload.getMessage());
                break;
            case DISCONNECT:
                currentRoom.disconnect(this);
                break;
```

handling commands

adding client method

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

addClient(ServerThread client) method adds a new client to a room, ensuring thread safety with the synchronized keyword. It first checks if the room is running; if not, it exits without action. If the client is already in the room, it logs a message and returns. If the client is new, it adds them to the clientsInRoom map

Sub-Task

100%

Group: Communication

Task #2: Rooms

Sub Task #5: Show the Server code for handling the create/join process

Task Screenshots

Gallery Style: 2 Columns

4 2 1

server code for join

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

allows a client to join a specified room. It first converts the room name to lowercase for consistency and checks if the room exists in the rooms map. If the room does not exist, it returns false. If the client is already in another room, it removes them from that room by calling removedClient(client). It then retrieves the specified room and adds the client to it by calling addClient(client), returning true to indicate a successful join.

Sub-Task

100%

Group: Communication

Task #2: Rooms

Sub Task #6: Show that Client messages are constrained to the Room (clients in different Rooms can't talk to each other)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

shows (far right) RYAN user is isolated from far left 2 clients

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain why/how it works this way

Response:

This is done via server threads, rooms manage said threads and once a new room is made it sync only those on the same thread.

End of Task 2

End of Group: Communication
Task Status: 2/2

Group

100%

Group: Disconnecting/Termination
Tasks: 1
Points: 3

COLLAPSE

Task

100%

Group: Disconnecting/Termination
Task #1: Disconnecting
Weight: ~100%
Points: ~3.00

COLLAPSE

Details:

Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.

Columns: 1

Sub-Task

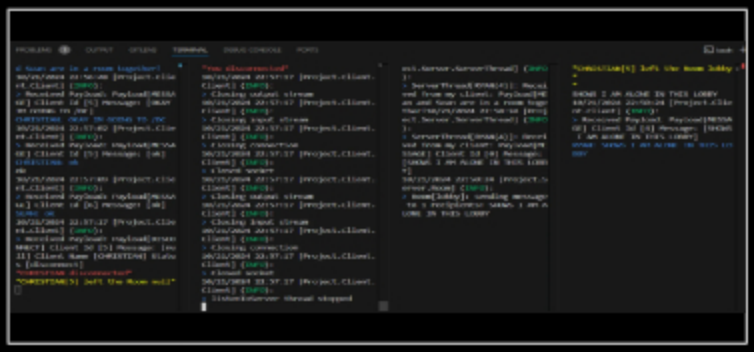
100%

Group: Disconnecting/Termination
Task #1: Disconnecting
Sub Task #1: Show Clients gracefully disconnecting (should not crash Server or other Clients)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



shows gracefull dc

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: Disconnecting/Termination
Task #1: Disconnecting
Sub Task #2: Show the code related to Clients disconnecting

Task Screenshots

Gallery Style: 2 Columns

4 2 1

graceful dc

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

closes a chat room by notifying clients of the shutdown, migrating them to a lobby, and then removing the room from the server while clearing its client list.

Sub-Task

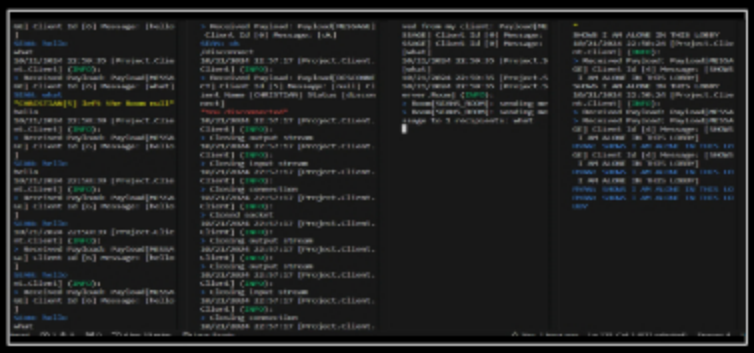
100%

Group: Disconnecting/Termination
Task #1: Disconnecting
Sub Task #3: Show the Server terminating (Clients should be disconnected but still running)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



shows other clients sending a message and server detects it after D/c

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: Disconnecting/Termination
Task #1: Disconnecting
Sub Task #4: Show the Server code related to handling termination

Task Screenshots

Gallery Style: 2 Columns

4 2 1

room removing

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

removes a specified room from the server's list of rooms, logging the action for informational purposes. The room name is converted to lowercase to ensure case-insensitive removal.

End of Task 1

End of Group: Disconnecting/Termination
Task Status: 1/1

Group

100%

Group: Misc
Tasks: 3
Points: 1

COLLAPSE

Task

100%

Group: Misc
Task #1: Add the pull request link for this branch
Weight: ~33%
Points: ~0.33

COLLAPSE

Task URLs

URL #1
https://github.com/Onervv/mcp62-IT114-003/pull/12

URL
https://github.com/Onervv/mcp62-IT114-003/pul

End of Task 1

Task

100%

Group: Misc
Task #2: Talk about any issues or learnings during this assignment
Weight: ~33%
Points: ~0.33

COLLAPSE

Details:
Few related sentences about the Project/sockets topics

Task Response Prompt

Response:

Since I did a previous pull to milestone 1 most of the files were interchanged between milestone1 and 2 greatly increasing the difficulty of finding the related code however i tried my best to find the declerations where it was mostly handling said component.

End of Task 2

Task

100%

Group: Misc
Task #3: WakaTime Screenshot
Weight: ~33%
Points: ~0.33

COLLAPSE

Details:
Grab a snippet showing the approximate time involved that clearly shows your repository.

The duration isn't considered for grading, but there should be some time involved.

Task Screenshots

Gallery Style: 2 Columns

4 2 1

waka

End of Task 3

End of Group: Misc
Task Status: 3/3

End of Assignment