Submission Worksheet

Submission Data

Course: IT114-003-F2025
Assignment: IT114 Milestone 1
Student: Yusaf W. (yaw4)
Status: Submitted | **Worksheet Progress:** 100%
Potential Grade: 10.00/10.00 (100.00%)
Received Grade: 0.00/10.00 (0.00%)
Started: 11/4/2025 12:17:21 PM
Updated: 11/4/2025 12:17:21 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-003-F2025/it114-milestone-1/grading/yaw4
View Link: https://learn.ethereallab.app/assignment/v3/IT114-003-F2025/it114-milestone-1/view/yaw4

Instructions

- Overview Link: https://youtu.be/9dZPFwi76ak

1. Refer to Milestone1 of any of these docs:
    2. Rock Paper Scissors
    3. Basic Battleship
    4. Hangman / Word guess
    5. Trivia
    6. Go Fish
    7. Pictionary / Drawing
2. Ensure you read all instructions and objectives before starting.
3. Ensure you've gone through each lesson related to this Milestone
4. Switch to the Milestone1 branch
    1. `git checkout Milestone1` (ensure proper starting branch)
    2. `git pull origin Milestone1` (ensure history is up to date)
5. Copy `Part5` and rename the copy as `Project` (this new folder should be in the root of your repo)
6. Organize the files into their respective packages `Client`, `Common`, `Server`, `Exceptions`
    1. Hint: If it's open, you can refer to the Milestone 2 Prep lesson
7. Fill out the below worksheet
    1. Ensure there's a comment with your UCID, date, and brief summary of the snippet in each screenshot
    2. Since this Milestone was majorly done via lessons, the required comments should be placed in areas of analysis of the requirements in this worksheet. There shouldn't need to be any actual code changes beyond the restructure.
8. Once finished, click "Submit and Export"
9. Locally add the generated PDF to a folder of your choosing inside your repository folder and move it to Github
    1. `git add .`
    2. `git commit -m "adding PDF"`
    3. `git push origin Milestone1`
    4. On Github merge the pull request from `Milestone1` to `main`
10. Upload the same PDF to Canvas
11. Sync Local

1. git checkout main
2. git pull origin main

Section #1: (1 pt.) Feature: Server Can Be Started Via Command Line And Listen To Connections

Progress: 100%

Task #1 (1 pt.) - Evidence

Progress: 100%

Part 1:

Progress: 100%

Details:

- Show the terminal output of the server started and listening
- Show the relevant snippet of the code that waits for incoming connections

```
$ java Project/Server/Server.java
Server Starting
Server: Listening on port 3000
Room[lobby]: Created
Server: Created new Room lobby
Server: Waiting for next client
```

Server output

```
private void start(int port) {
    this.port = port;
    // server listening
    info("Listening on port " + this.port);
    // Simplified client connection loop
    try (ServerSocket serverSocket = new ServerSocket(port)) {
        createRoom(Room.LOBBY);// create the first room (lobby)
        while (isRunning) {
            info(message: "Waiting for next client");
            Socket incomingClient = serverSocket.accept(); // Blocking action, wait for a client connection
            info(message: "Client connected");
            // wrap socket in a ServerThread, pass a callback to notify the server when
            // they're initialized
            ServerThread serverThread = new ServerThread(incomingClient, this::onServerThreadInitialized);
            // aware the thread (typically an external entity manages the lifecycle and we
            // don't have the thread start itself)
            serverThread.start();
            // Note: We don't yet add the ServerThread reference to our connectedClients map
        }
    } catch (DuplicateRoomException e) {
        System.err.println(Text.COLORIZE(text: "Lobby already exists (this shouldn't happen)", Color.RED));
    } catch (IOException e) {
        System.err.println(Text.COLORIZE(text: "Error accepting connection", Color.RED));
        e.printStackTrace(); // yeah it's!// Server listens on port for incoming connections and then creates a new ServerThread for that port when connection is created
    } finally {
        info(message: "Closing server socket");
    }
```

Server code

Saved: 11/4/2025 11:54:10 AM

Part 2:

Progress: 100%

Details:

- Briefly explain how the server-side waits for and accepts/handles connections

Your Response:

Server listens on port for incoming connections and then creates a new ServerThread for that port when connection is created

Saved: 11/4/2025 11:54:10 AM

Section #2: (1 pt.) Feature: Server Should Be Able To Allow More Than One Client To Be Connected At Once

Progress: 100%

Task #1 (1 pt.) - Evidence

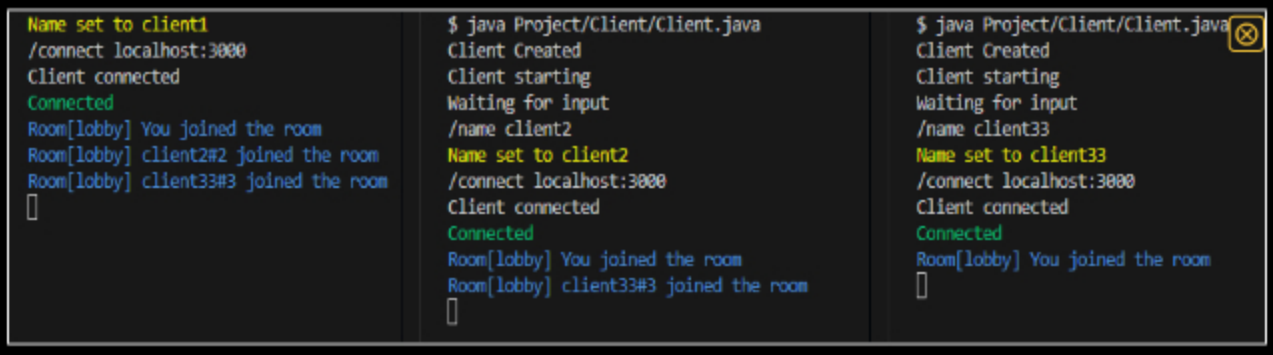
Progress: 100%

Part 1:

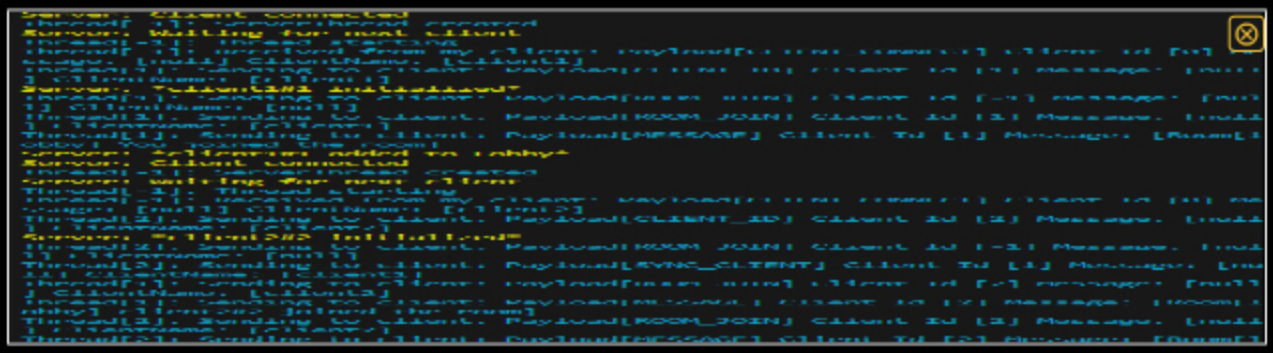
Progress: 100%

Details:

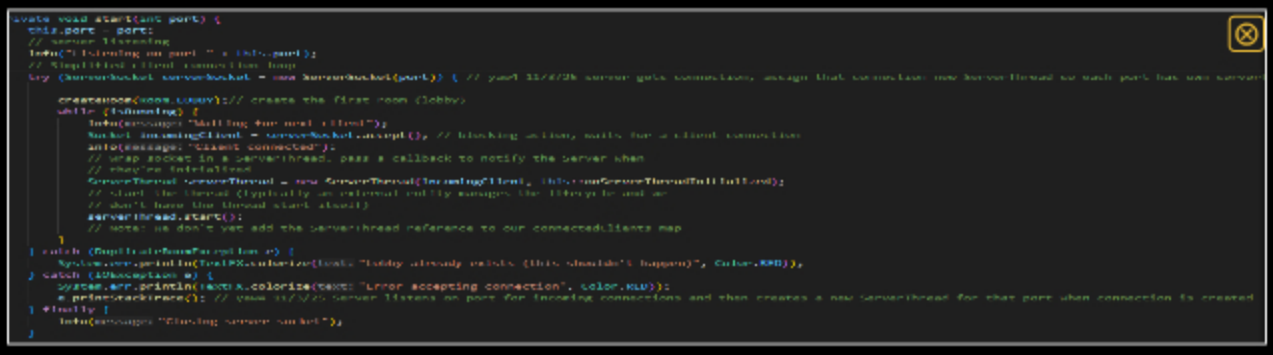
- Show the terminal output of the server receiving multiple connections
- Show at least 3 Clients connected (best to use the split terminal feature)
- Show the relevant snippets of code that handle logic for multiple connections



Client Connecting



Server Running



Serverside Code

Saved: 11/4/2025 11:57:36 AM

Part 2:

Progress: 100%

Details:

- Briefly explain how the server-side handles multiple connected clients

Your Response:

When the server gets a connection, it will assign that connection to a new ServerThread so that each port has its own serverthread and then each Client has its own thread.

Saved: 11/4/2025 11:57:36 AM

Section #3: (2 pts.) Feature: Server Will Implement The Concept Of Rooms (With The Default Being "l obby")

Progress: 100%

Task #1 (2 pts.) - Evidence

Progress: 100%

Part 1:

Progress: 100%

Details:

- Show the terminal output of rooms being created, joined, and removed (server-side)
- Show the relevant snippets of code that handle room management (create, join, leave, remove) (server-side)

```
Thread[1]: Received from my client: Payload[ROOM_CREATE] Client Id [0] Message: [test]
Room[test]: Created
Server: Created new Room test
Server: Removing client from previous Room lobby
Thread[1]: Sending to client: Payload[ROOM_LEAVE] Client Id [1] Message: [null] ClientName: [client1]
Thread[1]: Sending to client: Payload[MESSAGE] Client Id [1] Message: [Room[lobby] You left the room]
ient Id [2] Message: [Room[lobby] client2#2 left the room]
Thread[2]: Sending to client: Payload[ROOM_JOIN] Client Id [-1] Message: [null] ClientName: [null]Thread[2]: Ser
ding to client: Payload[SYNC_CLIENT] Client Id [1] Message: [null] ClientName: [client1]
Thread[1]: Sending to client: Payload[ROOM_JOIN] Client Id [2] Message: [null] ClientName: [client2]
Thread[1]: Sending to client: Payload[MESSAGE] Client Id [2] Message: [Room[test] client2#2 joined the room]
Thread[2]: Sending to client: Payload[ROOM_JOIN] Client Id [2] Message: [null] ClientName: [client2]
Thread[2]: Sending to client: Payload[MESSAGE] Client Id [2] Message: [Room[test] You joined the room]
```

Serverside room created

Serverside Room Code

Saved: 11/4/2025 12:01:40 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the server-side handles room creation, joining/leaving, and removal

Your Response:

The server checks to see if that room was already created, if not, it creates a room and then puts someone into the room. If someone requests to join a room, it removes them from the previous room first and then sends them to the other room. If a client wants to leave the room, it simply removes the client from the room via its Id.

Saved: 11/4/2025 12:01:40 PM

Section #4: (1 pt.) Feature: Client Can Be Started Via The Command Line

Progress: 100%

Task #1 (1 pt.) - Evidence

Progress: 100%

Part 1:

Part 1:

Progress: 100%

Details:

- Show the terminal output of the /name and /connect commands for each of 3 clients (best to use the split terminal feature)
- Output should show evidence of a successful connection
- Show the relevant snippets of code that handle the processes for /name, /connect, and the confirmation of being fully setup/connected

```
Client Created
Client starting
Waiting for input
/name client1
Name set to client1
/connect localhost:3000
Client connected
Connected
Room[lobby] You joined the room
Room[lobby] client2#2 joined the room
Room[lobby] client3#3 joined the room
```

```
$ java Project/Client/Client.java
Client Created
Client starting
Waiting for input
/name client2
Name set to client2
/connect localhost:3000
Client connected
Connected
Room[lobby] You joined the room
Room[lobby] client3#3 joined the room
```

```
$ java Project/Client/Client.java
Client Created
Client starting
Waiting for input
/name client3
Name set to client3
/connect localhost:3000
Client connected
Connected
Room[lobby] You joined the room
```

Client output

Clientside Code

```
private void joinStatusRelay(ServerThread client, boolean didJoin) {
    clientsInRoom.values().removeIf(serverThread -> {
        String formattedMessage = String.format(format: "Room[%s] %s %s the room",
                getName(),
                client.getClientId() == serverThread.getClientId() ? "You"
                        : client.getDisplayName(),
                didJoin ? "joined" : "left"); // yeet 11/4/25 used to relay which users are in what rooms and make sure they are connected
        final long senderId = client == null ? Constants.DEFAULT_CLIENT_ID : client.getClientId();
        // Share info of the client joining or leaving the room
        boolean failedToSync = !serverThread.sendClientInfo(client.getClientId(),
                client.getClientName(), didJoin ? RoomAction.JOIN : RoomAction.LEAVE);
        // Send the server generated message to the current client
        boolean failedToSend = !serverThread.sendMessage(senderId, formattedMessage);
        if (failedToSend || failedToSync) {
            System.out.println(
                    String.format(format: "Removing disconnected %s from list", serverThread.getDisplayName()));
            disconnect(serverThread);
        }
        return failedToSend;
    });
}
```

Serverside Code

Saved: 11/4/2025 12:05:20 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the /name and /connect commands work and the code flow that leads to a successful connection for the client

Your Response:

The client sets its name and then when connecting, it sends a payload to request a connection on a port and then the server gets that clientID and then puts it into a room.

Saved: 11/4/2025 12:05:20 PM

Section #5: (2 pts.) Feature: Client Can Create/j oin Rooms

Progress: 100%

Task #1 (2 pts.) - Evidence

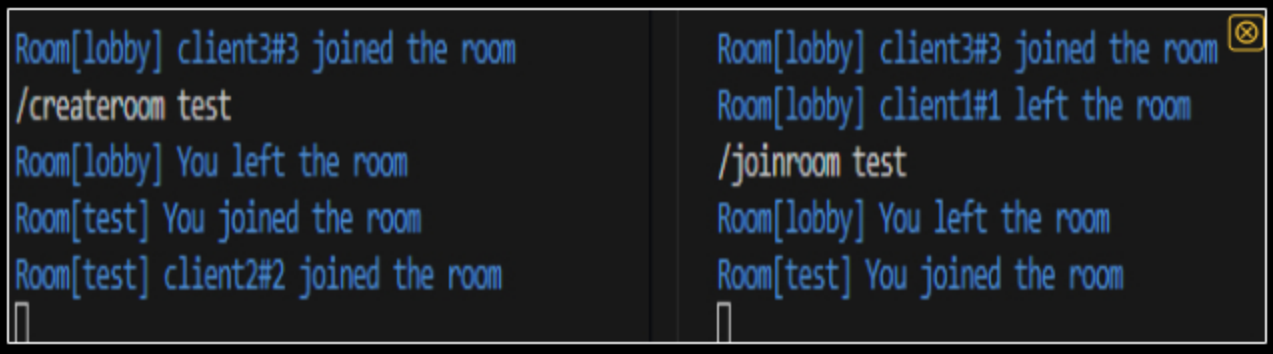
Progress: 100%

Part 1:

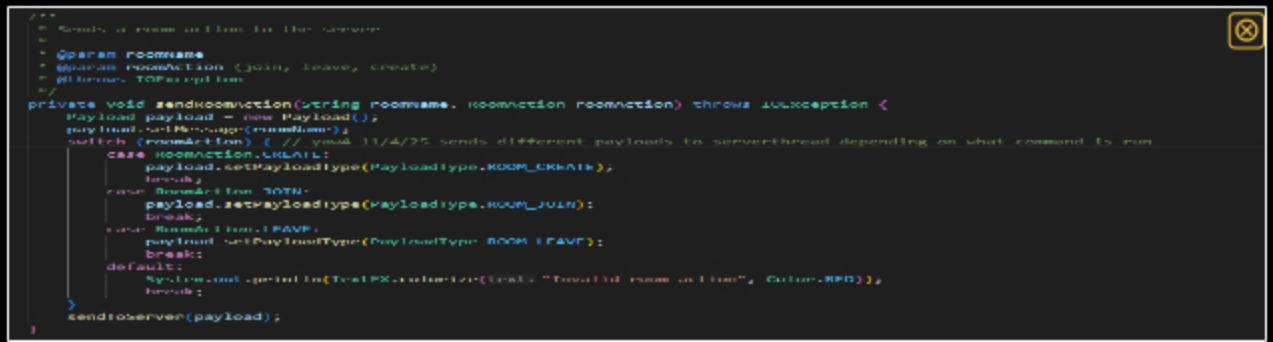
Progress: 100%

Details:

- Show the terminal output of the /createroom and /joinroom
- Output should show evidence of a successful creation/join in both scenarios
- Show the relevant snippets of code that handle the client-side processes for room creation and joining



Clients joining rooms



Clientside Code

Saved: 11/4/2025 12:07:30 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the /createroom and /join room commands work and the related code flow for each

Your Response:

The client finds out what command is meant to be sent and sends a payload to the server that the room should be created or joined and the server then creates or joins the room requested and adds the client that created it automatically.

Saved: 11/4/2025 12:07:30 PM

Section #6: (1 pt.) Feature: Client Can Send Messages

Progress: 100%

Task #1 (1 pt.) - Evidence

Progress: 100%

Part 1:

Progress: 100%

Details:

- Show the terminal output of a few messages from each of 3 clients
- Include examples of clients grouped into other rooms
- Show the relevant snippets of code that handle the message process from client to server-side and back

```
Room[test] client3#3 left the room
client4 cant hear this
client1#1: client4 cant hear this
```

```
client3#3: that was a message.
Room[test] client3#3 left the room
client1#1: client4 cant hear this
```

```
/createroom test2
Room[lobby] You left the room
Room[test2] You joined the room
test
client4#5: test
test
client4#5: test
```

message cant hear

```
Room[test] client3#3 joined the room
this is a message
client1#1: this is a message
client2#2: is that a message?
client3#3: that was a message.
```

```
client1#1: this is a message
is that a message?
client2#2: is that a message?
client3#3: that was a message.
```

```
client1#1: this is a message
client2#2: is that a message?
that was a message.
client3#3: that was a message.
```

Message output

```java
/**
 * Sends a message to the server
 *
 * @param message
 * @throws IOException
 */
private void sendMessage(String message) throws IOException {
    Payload payload = new Payload();
    payload.setMessage(message); // yaw4 11/4/25 used to send message to serverThread in payload
    payload.setPayloadType(PayloadType.MESSAGE);
    sendToServer(payload);
}
```

Clientside Code

```java
/**
 * Sends a message to the client
 *
 * @param clientId who it's from
 * @param message
 * @return true for successful send
 */
protected boolean sendMessage(long clientId, String message) {
    Payload payload = new Payload();
    payload.setPayloadType(PayloadType.MESSAGE);
    payload.setMessage(message);
    payload.setClientId(clientId); // yaw4 11/4/25 used to relay message to server
    return sendToClient(payload);
}
```

ServerThread Code

```java
/**
 * Used to send a message to all Rooms.
 * This is just an example and we likely won't be using this
 *
 * @param sender
 * @param message
 */
public synchronized void broadcastMessageToAllRooms(ServerThread sender, String message) {
    relayToAllRooms(sender, message); // yaw4 11/4/25 used to broadcast message to all rooms
}
```

Serverside Code

Saved: 11/4/2025 12:13:13 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the message code flow works

Your Response:

The client sends a payload to the serverthread which finds what client is came from and then sends that payload to the server. When the server gets the payload, it is relayed to all the clients that are connected.

Saved: 11/4/2025 12:13:13 PM

Section #7: (1 pt.) Feature: Disconnection

Progress: 100%

Task #1 (1 pt.) - Evidence

Progress: 100%

Part 1:

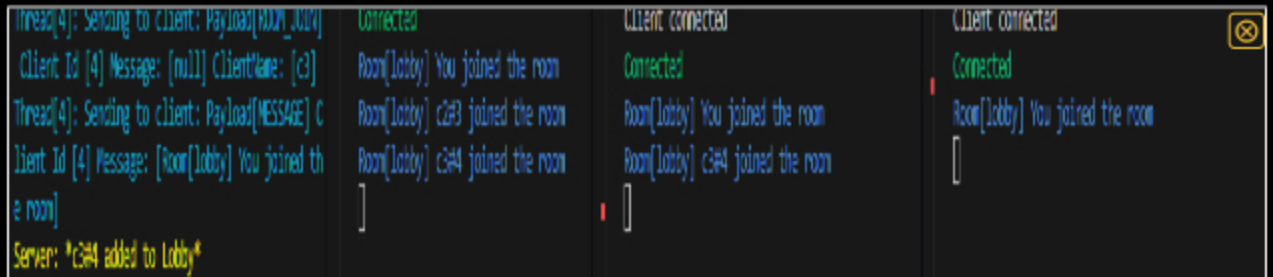
Progress: 100%

Details:

- Show examples of clients disconnecting (server should still be active)
- Show examples of server disconnecting (clients should be active but disconnected)
- Show examples of clients reconnecting when a server is brought back online
- Examples should include relevant messages of the actions occuring
- Show the relevant snippets of code that handle the client-side disconnection process
- Show the relevant snippets of code that handle the server-side termination process

```
Room[test]: sending message to 1 recipients: Room[test]: client1#1 disconnected
Thread[2]: Sending to client: Payload[MESSAGE] Client Id [-1] Message: [Room[test]:
client1#1 disconnected]
Thread[1]: Thread being disconnected by server
Thread[1]: ServerThread cleanup() start
Thread[-1]: Closed Server-side Socket
Thread[-1]: ServerThread cleanup() end
Thread[-1]: Exited thread loop. Cleaning up connection
Thread[-1]: ServerThread cleanup() start
Thread[-1]: Closed Server-side Socket
Thread[-1]: ServerThread cleanup() end
```

Server Client disconnect



Client Reconnection

```
Thread[6]: Thread being disconnected by server
Thread[6]: ServerThread cleanup() start
Thread[-1]: Closed Server-side Socket
Thread[-1]: ServerThread cleanup() end
Thread[-1]: Thread interrupted during read (likely from the disconnect() method)
Thread[-1]: Exited thread loop. Cleaning up connection
Thread[-1]: ServerThread cleanup() start
Room[test2]: Disconnect All finished
Room[lobby]: Disconnect All triggered
Room[lobby]: Disconnect All finished
Room[test]: Disconnect All triggered
Room[test]: sending message to 0 recipients: Room[test]: client2#2 disconnected
Thread[2]: Thread being disconnected by server
Thread[2]: ServerThread cleanup() start
Thread[-1]: Closed Server-side Socket
Thread[-1]: ServerThread cleanup() end
Server: Removed room test
Thread[-1]: Thread interrupted during read (likely from the disconnect() method)
Thread[-1]: Exited thread loop. Cleaning up connectionThread[-1]: ServerThread cleanup() start
Thread[-1]: Closed Server-side Socket

Thread[-1]: ServerThread cleanup() end
Room[test]: closed
```

Server shutting down

```
/**
 * Gracefully disconnect clients
 */
private void shutdown() {
    try {
        // chose removeIf over forEach to avoid potential
        // ConcurrentModificationException
        // since empty rooms tell the server to remove themselves
        rooms.values().removeIf(room -> {
            room.disconnectAll();
            return true;
        });  //yaw4 11/4/25 shuts down server and removes all from room when shutdown
    } catch (Exception e) {
        e.printStackTrace();
    }        You, 14 hours ago • adding baseline files ...
}
```

Serverside disconnect

```
private void processDisconnect(Payload payload) {
    if (payload.getClientId() == myUser.getClientId()) {
        knownClients.clear();
        myUser.reset(); //yaw4 11/4/25 deals with disconnect in client      You, 1 second ago • Uncommitted changes
        System.out.println(TextFX.colorize(text: "You disconnected", Color.RED));
    } else if (knownClients.containsKey(payload.getClientId())) {
        User disconnectedUser = knownClients.remove(payload.getClientId());
        if (disconnectedUser != null) {
            System.out.println(TextFX.colorize(String.format(format: "%s disconnected", disconnectedUser.getDisplayName()),
                    Color.RED));
        }
    }

}
```

Clientside Code

Saved: 11/4/2025 12:17:21 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how both client and server gracefully handle their disconnect/termination logic

Your Response:

The client sends information to the server that the client is requesting disconnection and then

disconnects the client and removing the id. The server closes all the rooms and makes sure that clients are send information that server is disconnecting then shuts down the server and the clients disconnect.

Saved: 11/4/2025 12:17:21 PM

Section #8: (1 pt.) Misc

Progress: 100%

--- Section Collapsed ---