Submission Worksheet

Submission Data

Course: IT114-450-M2025
Assignment: IT114 Milestone 1
Student: Anthony L. (agl8)
Status: Submitted | **Worksheet Progress:** 100%
Potential Grade: 10.00/10.00 (100.00%)
Received Grade: 0.00/10.00 (0.00%)
Started: 9/25/2025 11:12:47 PM
Updated: 9/26/2025 10:53:51 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-1/grading/agl8
View Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-1/view/agl8

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

```
anthonyleone@MacBookAir ~/Library/CloudStorage/OneDrive-Personal/NJIT Coursework/Summer 2025/agl8-IT114-450 $ javac Project/Server/Server.java
anthonyleone@MacBookAir ~/Library/CloudStorage/OneDrive-Personal/NJIT Coursework/Summer 2025/agl8-IT114-450 $ java Project.Server.Server
Server Starting
Server: Listening on port 3000
Room[lobby]: Created
Server: Created new Room lobby
Server: Waiting for next client
```

terminal output - server started and listening

```
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, waits 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);
            // start 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
        } // #59-70 while (isRunning)
    } catch (DuplicateRoomException e) {
        System.err.println(TextFX.colorize(text:"Lobby already exists (this shouldn't happen)", Color.RED));
    } catch (IOException e) {
```

code for incoming connections

Saved: 9/26/2025 9:47:03 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

The server uses ServerSocket to open port 3000 and listen for connections. Then it uses the while loop to endlessly listen for a port.

To accept and handle connections it uses the accept method and then it creates a ServerThread to handle the new connection.

Saved: 9/26/2025 9:47:03 PM

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

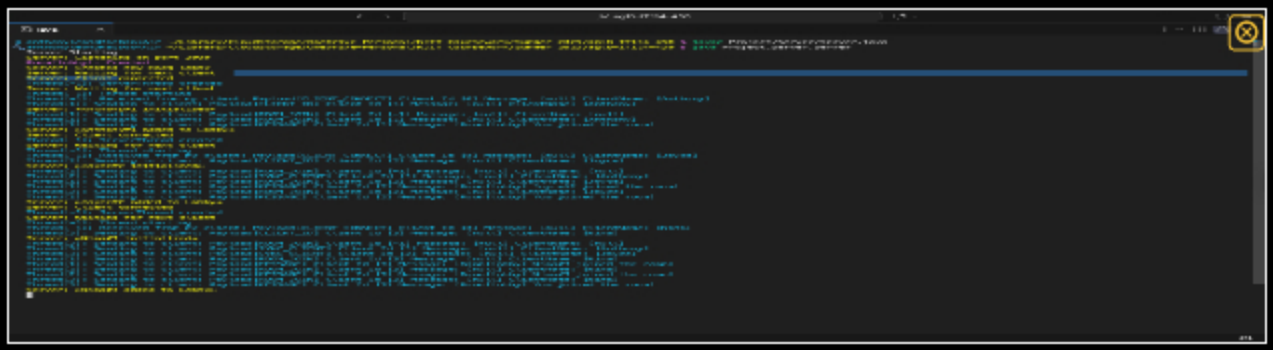
Progress: 100%

Part 1:

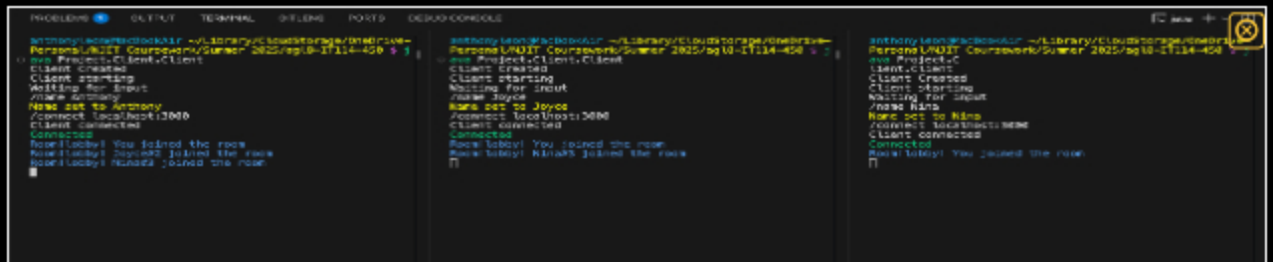
Progress: 100%

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



server receiving multiple connections



3 clients connected

code for server listening

code for client connection

Saved: 9/26/2025 10:11:46 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

When a new client connects to the Server creates a new ServerThread. That way the Server is free to listen for other connections instead of not doing so or only handling the arrived client. The Server being free after creating the new ServerThread lets it handle multiple connections.

Saved: 9/26/2025 10:11:46 PM

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

Progress: 100%

Task #1 (2 pts.) - Evidence

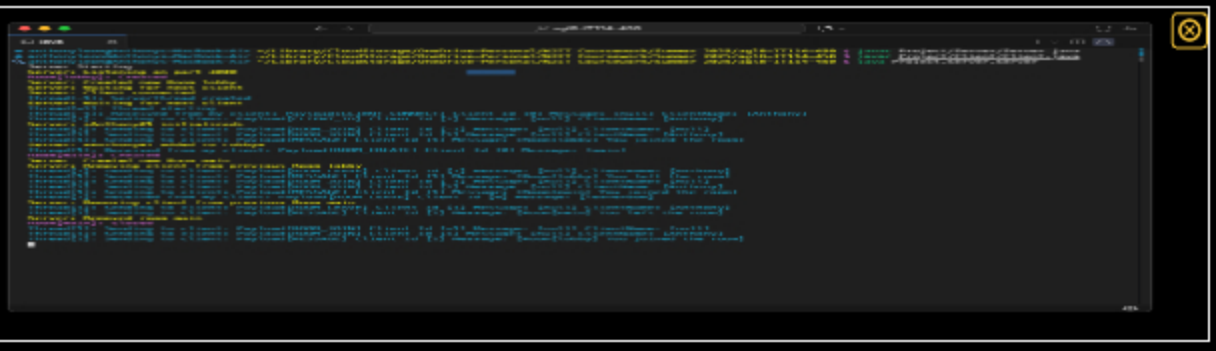
Progress: 100%

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)



server activity

code create and join

code add and remove

Saved: 9/26/2025 10:13:04 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

ServerThread sends the request for creating a room and sends it to the Server for processing. The Server creates the room and then places the creator in the room. The Server also manages moving the client from room to room. Once the client is placed in the room, the room is in charge of managing its clients via addClient and removeClient.

Saved: 9/26/2025 10:13:04 PM

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

Progress: 100%

Task #1 (1 pt.) - Evidence

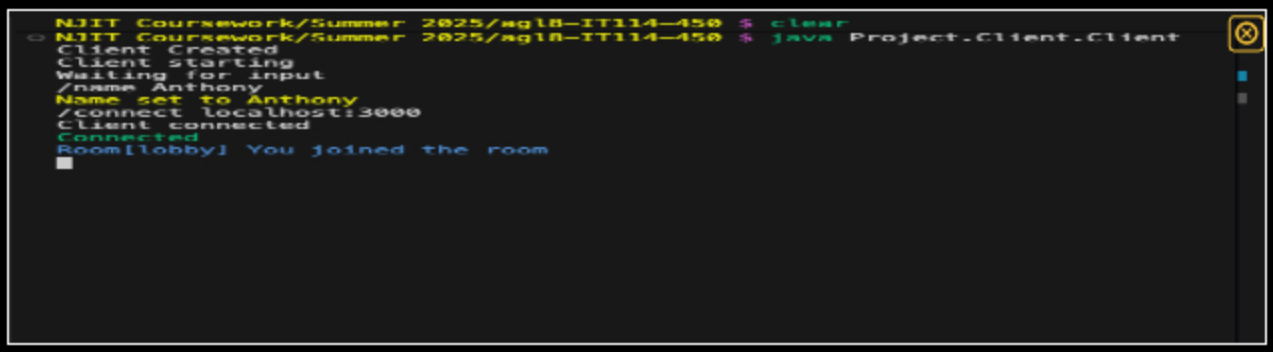
Progress: 100%

Part 1:

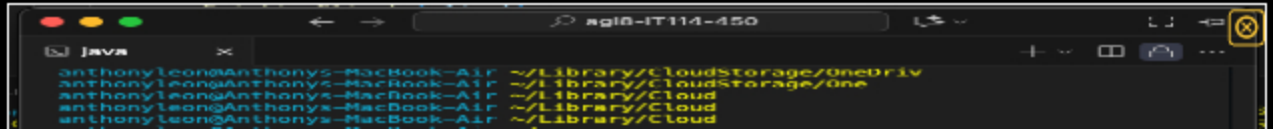
Progress: 100%

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 1



```
anthonyleon@Anthonys-MacBook-Air ~/
anthonyleon@Anthonys-MacBook-Air ~/
anthonyleon@Anthonys-MacBook-Air ~/
anthonyleon@Anthonys-MacBook-Air ~/Library/CloudStorage/OneDrive-Personal/NJIT Course
work/Summer 2025/agl8-IT114-450 $ java Project.Client.Client
Client Created
Client Starting
Waiting for input
/name Joyce
Name set to Joyce
/connect localhost:3000
Client connected
Connected
Room[lobby] You joined the room
```

client 2

client 3

```
    text = text.substring(beginIndex:1); // remove the /
    // System.out.println("Checking command: " + text);
    if (isConnection("/" + text)) {
        if (myUser.getClientName() == null || myUser.getClientName().isEmpty()) {
            System.out.println(
                    TextFX.colorize(text:"Please set your name via /name <name> before connecting", Color.RED));
            return true;
        } <-- #324-329 if (myUser.getClientName() == null || myUser.getClientName()....
        // replaces multiple spaces with a single space
        // splits on the space after connect (gives us host and port)
        // splits on : to get host as index 0 and port as index 1
        String[] parts = text.trim().replaceAll(regex:" +", replacement:" ").split(regex:" ")[1].split(regex:":");
        connect(parts[0].trim(), Integer.parseInt(parts[1].trim()));
        sendClientName(myUser.getClientName());// sync follow-up data (handshake)
        wasCommand = true;
    } else if (text.startsWith(Command.NAME.command)) {
        text = text.replace(Command.NAME.command, replacement:"").trim();
        if (text == null || text.length() == 0) {
            System.out.println(TextFX.colorize(text:"this command requires a name as an argument", Color.RED));
            return true;
        }
        myUser.setClientName(text);// temporary until we get a response from the server
        System.out.println(TextFX.colorize(String.format(format:"Name set to %s", myUser.getClientName()),
                Color.YELLOW));
        wasCommand = true;
```

code for commands

```
// Start process*() methods
private void processClientData(Payload payload) {
    if (myUser.getClientId() != Constants.DEFAULT_CLIENT_ID) {
        System.out.println(TextFX.colorize(text:"Client ID already set, this shouldn't happen", Color.YELLOW));

    }
    myUser.setClientId(payload.getClientId());
    myUser.setClientName(((ConnectionPayload) payload).getClientName());// confirmation from Server
    knownClients.put(myUser.getClientId(), myUser);
    System.out.println(TextFX.colorize(text:"Connected", Color.GREEN));
} <- #354-363 private void processClientData(Payload payload)
```

code for connect/confirmation

Saved: 9/26/2025 10:53:51 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:

/name: saves the clients name (not in Server).

/connect: connects to Server and sends name to Server in a ConnectionPayload.

Server returns client ID as confirmation.

Saved: 9/26/2025 10:53:51 PM

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

Progress: 100%

Task #1 (2 pts.) - Evidence

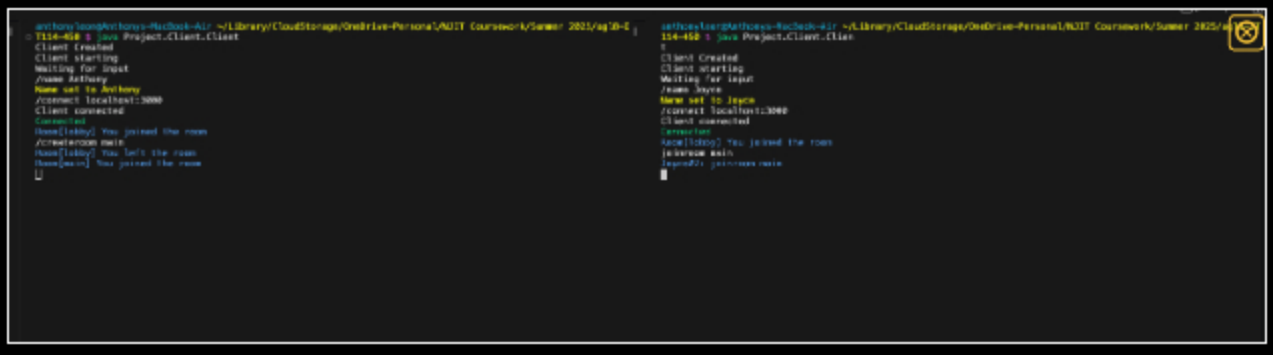
Progress: 100%

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



createroom and joinroom

```
} else if (text.startsWith(Command.CREATE_ROOM.command)) {
    text = text.replace(Command.CREATE_ROOM.command, replacement:"").trim();
    if (text == null || text.length() == 0) {
        System.out.println(TextFX.colorize(text:"This command requires a room name as an argument", Color.RED));
        return true;
    }
    sendRoomAction(text, RoomAction.CREATE);
    wasCommand = true;
} else if (text.startsWith(Command.JOIN_ROOM.command)) {
    text = text.replace(Command.JOIN_ROOM.command, replacement:"").trim();
    if (text == null || text.length() == 0) {
        System.out.println(TextFX.colorize(text:"This command requires a room name as an argument", Color.RED));
        return true;
    }
    sendRoomAction(text, RoomAction.JOIN);
    wasCommand = true;
```

code create and join

code create and join

Saved: 9/26/2025 10:36:07 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 processClientCommand method does alot of the handling with /createroom and /join. Depending on what is used, processClientCommand gets the name of the room and hands it to sendRoomAction. sendRoomAction creates an object that further handles the room name and then it sends it back to the Server.

Saved: 9/26/2025 10:36:07 PM

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

Progress: 100%

Task #1 (1 pt.) - Evidence

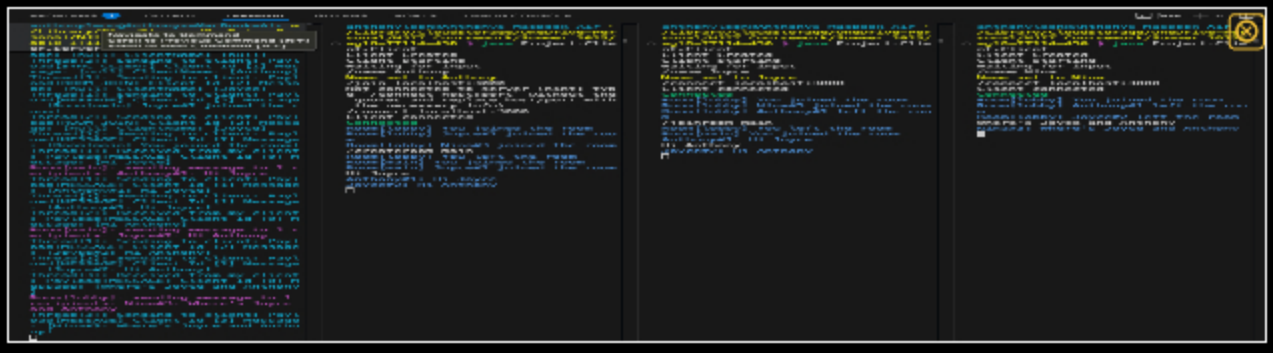
Progress: 100%

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



output messages

```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);
    payload.setPayloadType(PayloadType.MESSAGE);
    sendToServer(payload);
} <- #249-254 private void sendMessage(String message) throws IOException
```

message process 1

```java
switch (incoming.getPayloadType()) {
    case CLIENT_CONNECT:
        setClientName(((ConnectionPayload) incoming).getClientName().trim());

        break;
    case DISCONNECT:
        currentRoom.handleDisconnect(this);
        break;
    case MESSAGE:
        currentRoom.handleMessage(this, incoming.getMessage());
        break;
    case REVERSE:
        currentRoom.handleReverseText(this, incoming.getMessage());
        break;
    case ROOM_CREATE:
        currentRoom.handleCreateRoom(this, incoming.getMessage());
        break;
    case ROOM_JOIN:
        currentRoom.handleJoinRoom(this, incoming.getMessage());
        break;
    case ROOM_LEAVE:
        currentRoom.handleJoinRoom(this, Room.LOBBY);
        break;
    default:
        System.out.println(TextFX.colorize(text:"Unknown payload type received", Color.RED));
        break;
```

message process 2

message process 3

Saved: 9/26/2025 10:37:03 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the message code flow works

Your Response:

When a message is sent the client creates a Payload with the MESSAGE type. This gets sent to the ServerThread. The ServerThread then gives the message to the room you're in. The room then sends that message to everyone else in that same room. Other clients wont see the message if you're not in the same room.

Saved: 9/26/2025 10:37:03 PM

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

Progress: 100%

Task #1 (1 pt.) - Evidence

Progress: 100%

Part 1:

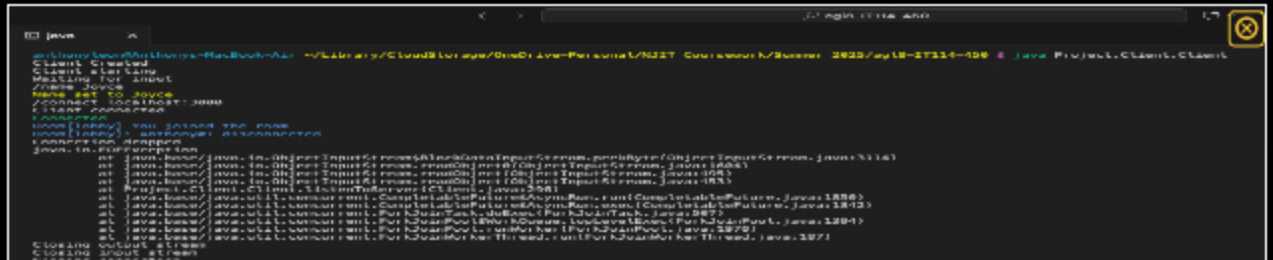
Progress: 100%

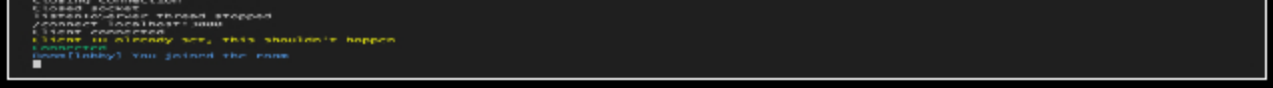
Details:

- 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

server showing client disconnect

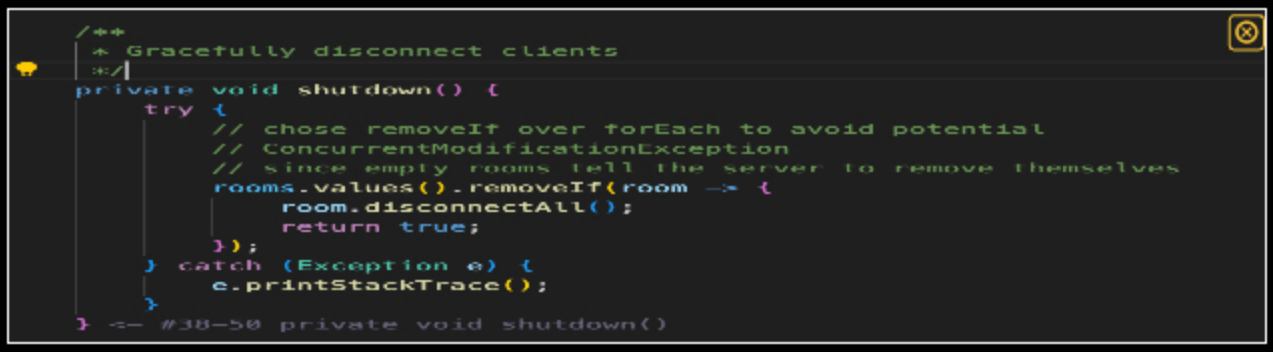
client showing server disconnect





client reconnect after restarting server

code client disconnection



code server disconnection

Saved: 9/26/2025 10:40:32 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

closeServerConnection method shuts down the connection on the clients end. The Server has a hook that runs when closed which tells every room to disconnect all of its clients.

Saved: 9/26/2025 10:40:32 PM

Section #8: (1 pt.) Misc

Progress: 100%

Task #1 (0.25 pts.) - Show the proper workspace structure

with the new Client, Common, Server, and Exceptions packages

Progress: 100%

file structure part 1

file structure part 2

Saved: 9/26/2025 9:10:12 PM

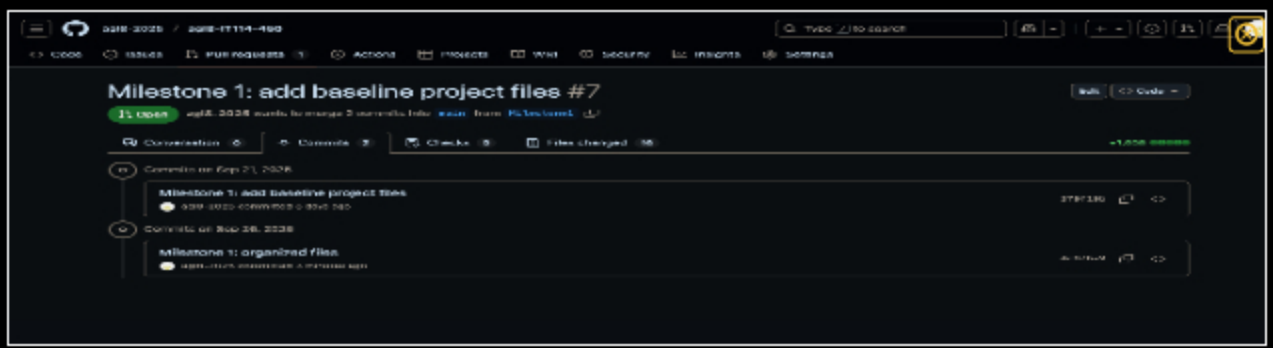
Task #2 (0.25 pts.) - Github Details

Progress: 100%

Part 1:

Progress: 100%

Details:
From the Commits tab of the Pull Request screenshot the commit history



commits screenshot

Saved: 9/26/2025 9:27:37 PM

Part 2:

Progress: 100%

Details:
Include the link to the Pull Request (should end in `/pull/#`)

URL #1
https://github.com/agl8-2025/agl8-IT114-450/

URL
https://github.com/agl8-2025/agl

Saved: 9/26/2025 9:27:37 PM

Task #3 (0.25 pts.) - WakaTime - Activity

Progress: 100%

Details:

- Visit the WakaTime.com Dashboard
- Click `Projects` and find your repository
- Capture the overall time at the top that includes the repository name
- Capture the individual time at the bottom that includes the file time
- Note: The duration isn't relevant for the grade and the visual graphs aren't necessary

bottom

top

Saved: 9/26/2025 9:29:24 PM

Task #4 (0.25 pts.) - Reflection

Progress: 100%

Task #1 (0.33 pts.) - What did you learn?

Progress: 100%

Details:
Briefly answer the question (at least a few decent sentences)

Your Response:

I learned the foundations of sockets that are used in communication between a server and client. I also reinforced my learning of how a server can handle multiple incoming connections.

Saved: 9/26/2025 10:48:47 PM

Task #2 (0.33 pts.) - What was the easiest part of the assignment?

Progress: 100%

Details:
Briefly answer the question (at least a few decent sentences)

Your Response:

Easiest part of the assignment was setting up the file structure.

Saved: 9/26/2025 10:48:54 PM

Task #3 (0.33 pts.) - What was the hardest part of the assignment?

Progress: 100%

Details:
Briefly answer the question (at least a few decent sentences)

Your Response:

Hardest part of the assignment was finding all the blocks that communicated with each other throughout different files but after multiple reviews you start seeing the whole picture and it clicks.

Saved: 9/26/2025 10:49:01 PM