Submission Worksheet

Submission Data

Course: IT114-003-F2025
Assignment: IT114 Milestone 1
Student: Rayyan K. (rk975)
Status: Submitted | **Worksheet Progress:** 100%
Potential Grade: 10.00/10.00 (100.00%)
Received Grade: 0.00/10.00 (0.00%)
Started: 11/3/2025 5:12:30 PM
Updated: 11/3/2025 6:46:11 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-003-F2025/it114-milestone-1/grading/rk975
View Link: https://learn.ethereallab.app/assignment/v3/IT114-003-F2025/it114-milestone-1/view/rk975

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

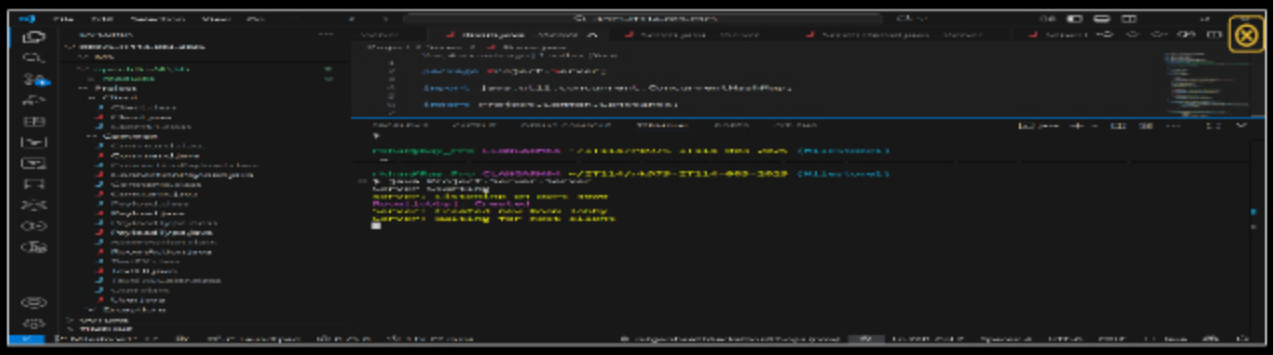
Progress: 100%

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



server starting

code snippet

Saved: 11/3/2025 5:18:07 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

A ServerSocket is created by the server to listen for incoming client requests on a designated port. In order to guarantee that the server is always prepared to accept new clients, the call to serverSocket.accept() stops the program until a client tries to connect. A Socket object, which represents the client's connection, is returned once a connection has been made. In order to manage communication with that client, the server then builds and launches a new ServerThread while keeping an eye out for new connections.

Saved: 11/3/2025 5:18:07 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

3 client connected

multiple connections

Saved: 11/3/2025 5:26:38 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

For every incoming connection, the server creates a new ServerThread to manage numerous clients. As soon as a client connects, the main server loop receives the socket and launches a new thread specifically for that client's communication. Because each ServerThread operates separately, clients can transmit and receive data at the same time without interfering with one another. The server can effectively handle numerous active connections at once thanks to this multithreaded strategy.

Saved: 11/3/2025 5:26:38 PM

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

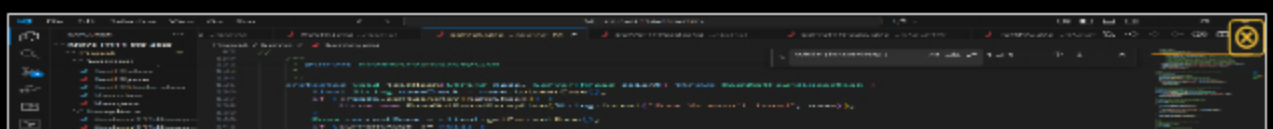
Progress: 100%

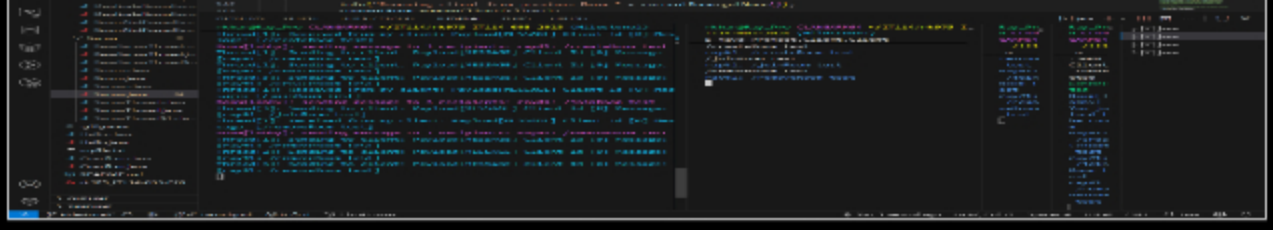
Part 1:

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)





rooms created etc

code snippet

Saved: 11/3/2025 5:56:35 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

The server creates, joins, removes, and leaves rooms using particular techniques. The createRoom() function adds newly formed rooms to a ConcurrentHashMap of active rooms. While the removeRoom() method removes empty rooms from the map, the joinRoom() method places a client in the specified room and removes them from any previous one. For several connected users, these coordinated techniques provide dynamic, well-organized, and thread-safe room administration.

Saved: 11/3/2025 5:56:35 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

terminal

snippet name

snippet connect

Saved: 11/3/2025 6:04:28 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 code eliminates the command keyword and saves the name in the myUser object using myUser when the user types the /name command followed by a name.setClientName(). This creates a temporary username that the server will subsequently get. The client then parses the host and port, creates a socket connection using the connect() method, and calls sendClientName() to send the username to the server as a ConnectionPayload when the user inputs the /connect command (for instance, /connect localhost:3000). In response, the server gives the client a unique ID and verifies the connection, which the client processes in processClientData() before outputting "Connected" to indicate that the setup was successful.

Saved: 11/3/2025 6:04:28 PM

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

Progress: 100%

Task #1 (2 pts.) - Evidence

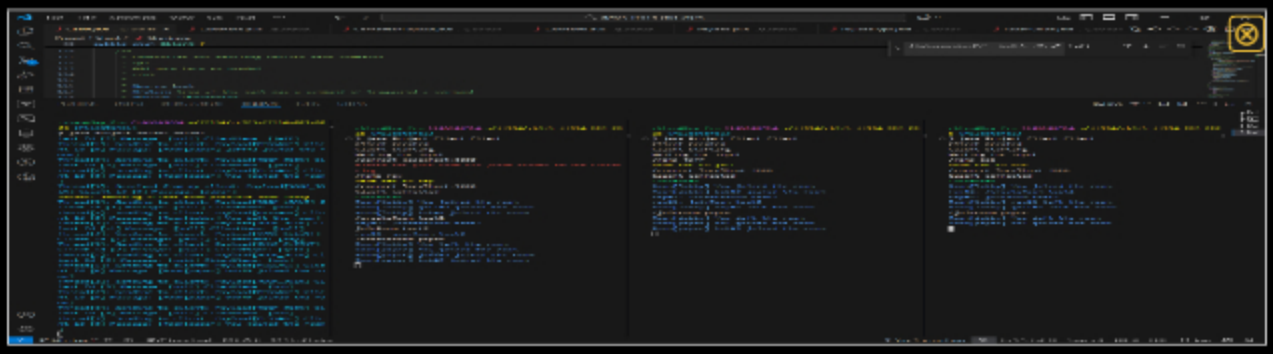
Progress: 100%

Part 1:

Progress: 100%

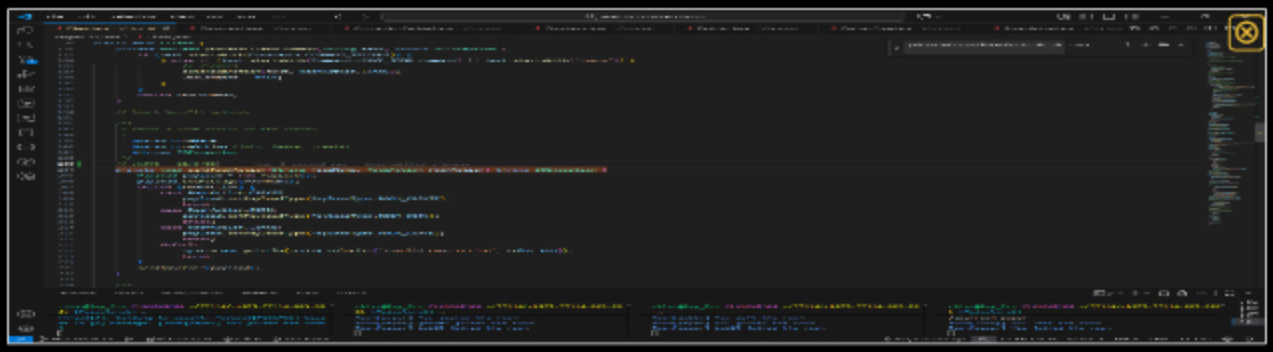
Details:

- 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



terminal

code snippet 1



code snipper 2

Saved: 11/3/2025 6:15:56 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:

When the user types /createroom or /joinroom , the client detects these commands inside the processClientCommand() method. The code removes the command keyword, extracts the room name, and then calls sendRoomAction(roomName, RoomAction.CREATE) or sendRoomAction(roomName, RoomAction.JOIN) depending on the command. The sendRoomAction() method creates a Payload object, sets its PayloadType (either ROOM_CREATE or ROOM_JOIN), attaches the room name as a message, and sends it to the server using sendToServer(). Once received, the server handles the request—either creating a new room or adding the client to the specified room—completing the process.

Saved: 11/3/2025 6:15:56 PM

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

Progress: 100%

Task #1 (1 pt.) - Evidence

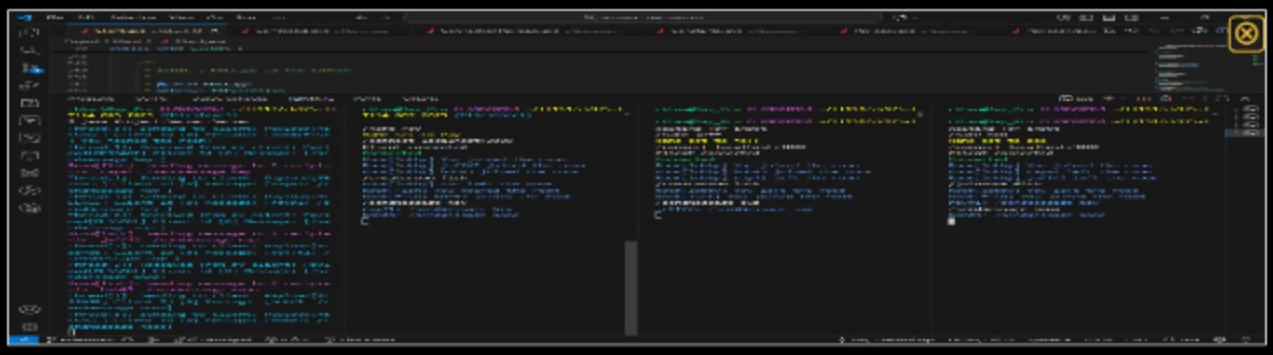
Progress: 100%

Part 1:

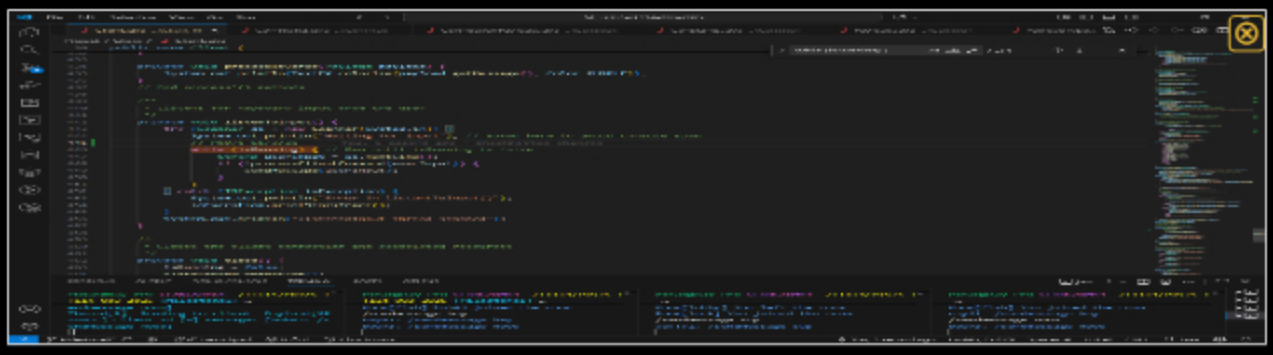
Progress: 100%

Details:

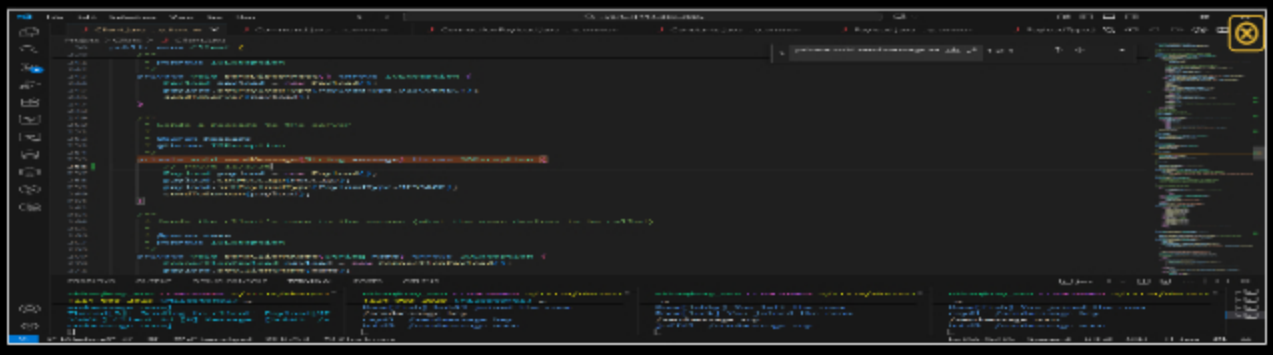
- Show the terminal output of a few messages from each of 3 clients
- Include examples of clients grouped into other rooms
- Show the relevant snippets of code that handle the message process from client to server-side and back



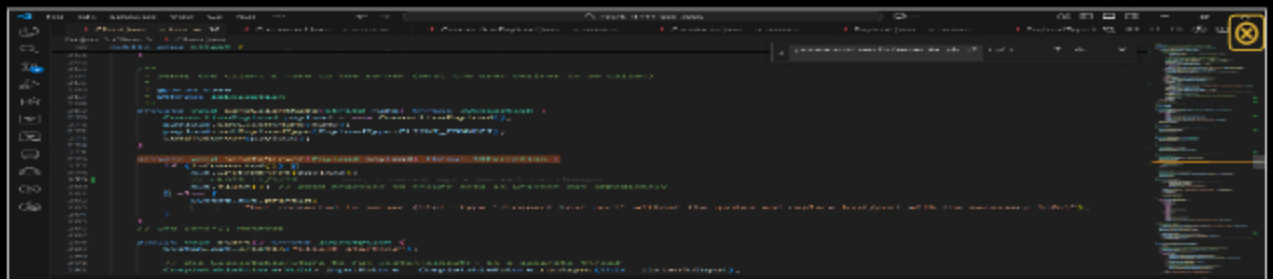
terminal



code snippet 1



code snippet 2



code snippet 3

Saved: 11/3/2025 6:29:58 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the message code flow works

Your Response:

The first function to handle a message typed by a client is listenToInput(), which then invokes sendMessage(). This method uses sendToServer() to send the payload to the server over the socket after creating a Payload object containing the message text and a PayloadType.MESSAGE. Each connected client operates on a ServerThread on the server side, which executes processPayload() after receiving the payload. After that, the message is sent to the relevant room, where it is distributed to every client who is connected. The message flow from one client to every other client is then completed when each client's listenToServer() method receives the broadcasted message and shows it to the user.

Saved: 11/3/2025 6:29:58 PM

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

Progress: 100%

Task #1 (1 pt.) - Evidence

Progress: 100%

Part 1:

Progress: 100%

Details:

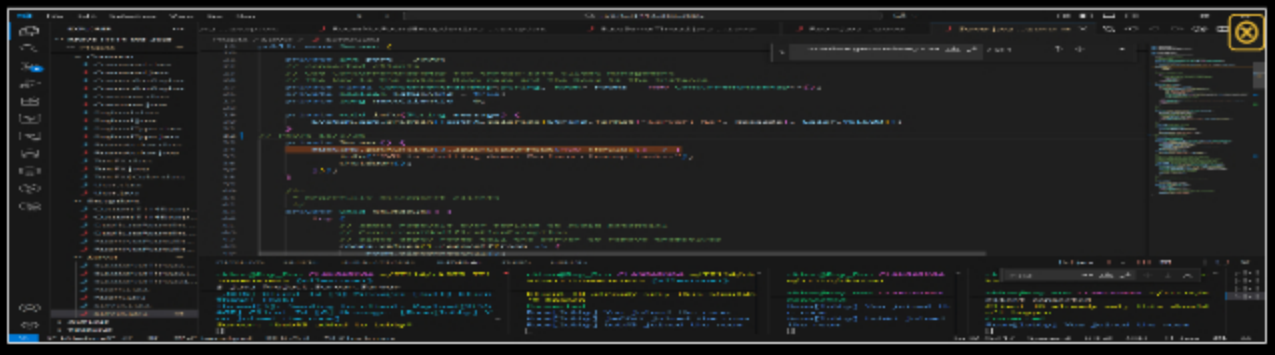
- Show examples of clients disconnecting (server should still be active)
- Show examples of server disconnecting (clients should be active but disconnected)
- Show examples of clients reconnecting when a server is brought back online
- Examples should include relevant messages of the actions occuring
- 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

disconnect clients

server disconnect

back online terminal

client termination



server termination

Saved: 11/3/2025 6:46:11 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

Typing /disconnect on the client side initiates the sendDisconnect() method, which transmits a DISCONNECT payload to the server. After that, the client closes its socket connection, ObjectInputStream, and ObjectOutputStream inside closeServerConnection(), clears its local state (knownClients), and resets the user data. This guarantees that the client leaves the network without leaving any open connections.

A Runtime shutdown hook in the constructor on the server side recognizes when the process quits or the JVM is ending. After that, the shutdown() method is invoked, which iterates through every room that is currently in use and disconnects any clients that are connected before deleting them from memory. This guarantees that all sockets are correctly closed and resources are safely released when the server quits or a crash).

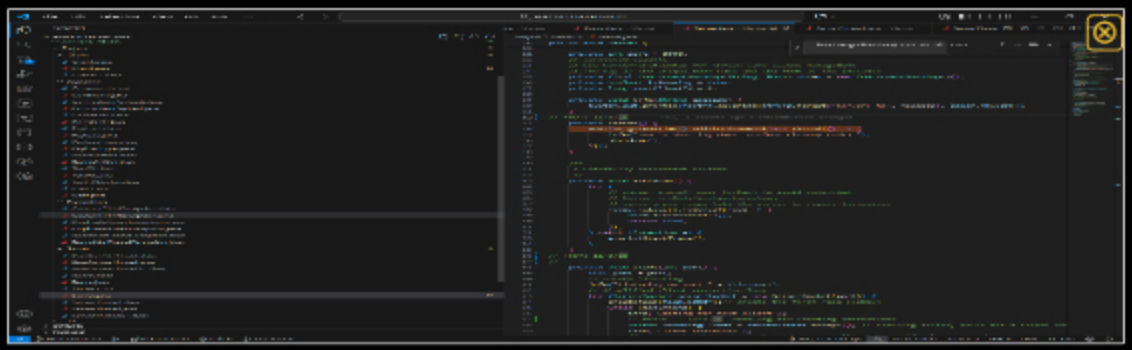
Saved: 11/3/2025 6:46:11 PM

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%



structure

Saved: 11/3/2025 6:40:16 PM

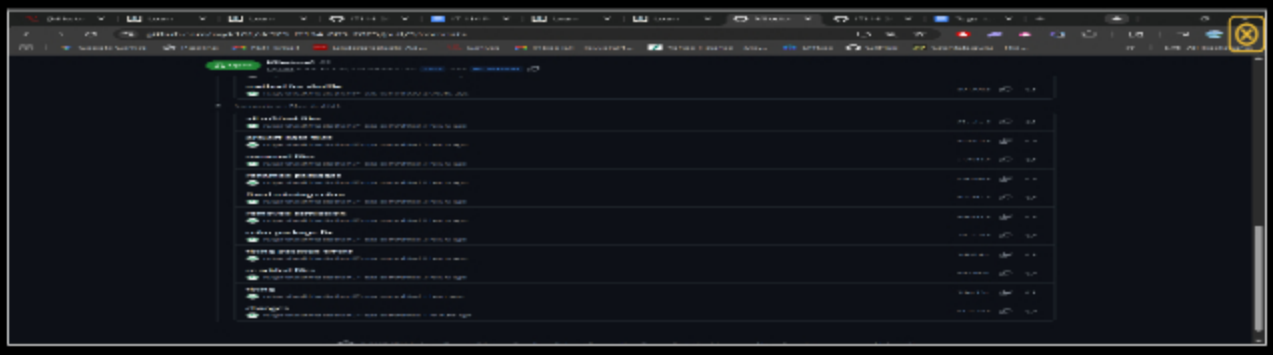
Task #2 (0.25 pts.) - Github Details

Progress: 100%

Part 1:

Progress: 100%

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



commit

Saved: 11/3/2025 6:42:43 PM

Part 2:

Progress: 100%

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

URL #1
https://github.com/rayk101/rk975-IT114-003-2025/

URL
https://github.com/rayk101/rk975

Saved: 11/3/2025 6:42:43 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

wakatime

wakatime2

Saved: 11/3/2025 6:44:05 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:

In order to maintain stability and stop resource leaks, I discovered how the client and server in a networked application handle graceful disconnections. To properly terminate its socket connection, close streams, and send a signal to the server, the client employs commands like /disconnect. In the meantime, the server performs cleanup procedures to disconnect clients and release resources after using a shutdown hook to recognize when it is closing. I learned from this procedure how crucial it is to handle termination properly in order to preserve dependable communication and avoid crashes in multi-client systems.

Saved: 11/3/2025 6:45:07 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:

Getting the client and server to connect and exchange simple messages after the packages and compilation problems were resolved was the simplest part of the assignment. It was easy to run the server and connect clients using /connect after the proper folder structure and package names were set up. The connection logs and message exchanges were easily visible in the terminal, confirming that the networking configuration was operating as intended.

Saved: 11/3/2025 6:45:29 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:

Resolving compilation and package structure problems to ensure proper communication across all client, server, and common files was the most challenging aspect of the task. Debugging required time to ensure that each file had the proper package declaration and that dependencies like LoggerUtil, Payload, and RoomAction were correctly identified. It was also difficult to handle imports over several folders and comprehend how each component interacted, particularly when handling connections and room management. The rationale became lot clearer after everything was successfully compiled and executed.

Saved: 11/3/2025 6:45:57 PM