

Course: IT114-006-S2025

Assignment: IT114 Milestone 1

Student: Muhammad K. (muk)

Status: Submitted | Worksheet Progress: 100%

Potential Grade: 10.00/10.00 (100.00%)

Received Grade: 0.00/10.00 (0.00%)

Grading Link: <https://learn.ethereallab.app/assignment/v3/IT114-006-S2025/it114-milestone-1/grading/muk>

Instructions

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
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
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

100%

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

Started via Command Line And Listen To Connections

100%

Task #1 (1 pt.) - Evidence

Combo Task:

Weight: 100%

Objective: Evidence

⇒ Image Prompt

Weight: 50%

Details:

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

I did not know server was started and ended up started it again.

卷之三十一

Snippet of the code



Saved: 4/10/2025 12:33:42 AM

⇒ Text Prompt

Weight: 50%

Details:

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

Your Response:

The default port 3000 is what the server listens on and uses `ServerSocket`. It uses `accept()` method which waits for a client to connect. `ServerThread` is used to read messages from the client where each thread is used.



Saved: 4/10/2025 12:33:42 AM

100%

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

100%

Task #1 (1 pt.) - Evidence

Combo Task:

Weight: 100%

Objective: *Evidence*

⇒ Image Prompt

Weight: 50%

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

```
terminal1: ~ % telnet localhost 2000
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
terminal2: ~ % telnet localhost 2000
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
terminal3: ~ % telnet localhost 2000
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
```

The terminal showing 3 clients connected

```
public class Server {
    INSTANCE; // Singleton instance
    {
        // STATICALLY INITIALIZED THE SERVER-SIDE LOGIC
        Listener server = new Listener(Server.class);
        Client client = new Client(server);
        ConcurrentHashMap<String, Client> clients = new ConcurrentHashMap<String, Client>();
        // Set the logger configuration
        LogConfigurator.setConfig();
    }
    private int port = 2000;
    // Connected clients
    // Map maintained for thread-safe client management
    // and the room of the client
    private final ConcurrentHashMap<String, Room> rooms = new ConcurrentHashMap<String, Room>();
    private Room awaiting = null;
    private long nextClientId = 0;
    private void incomingMessage() {
        logger.info("Received message from client " + client.getClientId() + " with message: " + client.getMessage());
    }
    private void() {
        Runtime.getRuntime().addShutdownHook(new Thread(() -> {
            if (client != null) {
                client.close();
            }
        }));
    }
}
```

Snippet of code shows where ConcurrentHashMap keeps track of connected clients

```
terminal1: ~ % telnet localhost 2000
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
terminal2: ~ % telnet localhost 2000
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
terminal3: ~ % telnet localhost 2000
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
```

Terminal output of the server receiving multiple connections



Saved: 4/10/2025 12:38:06 AM

Text Prompt

Weight: 50%

Details:

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

Your Response:

The server creates ServerThreads for each client's connection. ConcurrentHashMap keeps track of all clients' messages, helps send messages between clients, and removes clients if needed.

clients' messages, helps send messages between clients, and removes clients if needed.



Saved: 4/10/2025 12:38:06 AM

100%

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

100%

Task #1 (2 pts.) - Evidence

Combo Task:

Weight: 100%

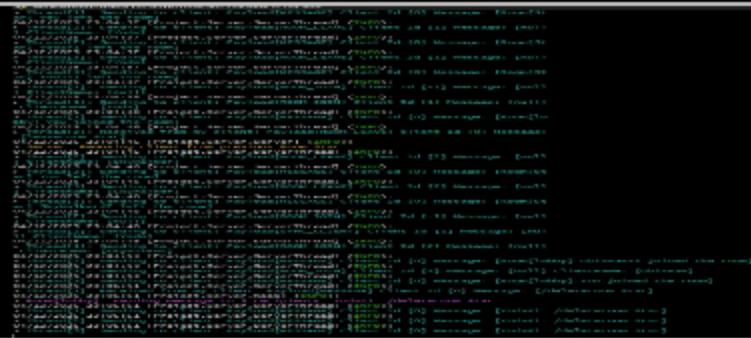
Objective: Evidence

→ Image Prompt

Weight: 50%

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)



A screenshot of a terminal window displaying a log of room management events. The log shows several entries of rooms being created, joined, and removed. The text is too small to read individually but follows a consistent pattern of room names and status changes.

Terminal output of rooms = created,joined and removed



Snippet of code



Saved: 4/13/2025 12:03:45 AM

Text Prompt

Weight: 50%

Details:

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

Your Response:

The server creates new room with the method /creatroom, joined rooms with method /joinroom and leave room with the method /removeroom . For create room the server checks if the room exists if not then it is created. For join room, the server removes the client from their orginal room and adds them to the new room. When a room is no longer needed the server removes the empty room.



Saved: 4/13/2025 12:03:45 AM

00%

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

10

Task #1 (1 pt.) - Evidence

Combo Task:

Weight: 100%

Objective: Evidence

☞ Image Prompt

Weight: 50%

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

The screenshot shows three separate terminal windows. Each window contains several lines of text, likely logs or command-line output, related to the execution of network commands like /name and /connect. The windows are arranged horizontally, representing the split terminal feature mentioned in the details.

Terminal output of /name and /connect

This screenshot displays a single terminal window with a large amount of detailed log output. The log entries are timestamped and show various messages related to network connections, user interactions, and system operations, providing evidence of the application's behavior when executing the specified commands.

Output showing evidence

This screenshot shows a code editor displaying a Java class. The code includes annotations for configuration, such as @Configuration and @Component. It features several private fields including a logger, a socket server, and various connection-related variables. Methods shown include error handling logic using loggerutil and a constructor for Client objects. The code is annotated with Javadoc-style comments explaining its purpose.

```

// Set the logger configuration
LoggerUtil.INSTANCE.setConfig(ConfigUtil.get());
private Socket server = null;
private ObjectInputStream in = null;
private ObjectOutputStream out = null;
final Pattern ipAddressPattern = Pattern.compile("\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}");
final Pattern localHostPattern = Pattern.compile("localhost|127.0.0.1");
private volatile boolean isRunning = true; // volatile for thread-safe visibility
private final ConcurrentHashMap<Long, User> knownClients = new ConcurrentHashMap<Long, User>();
private User myUser = new User();
private void error(String message) {
    loggerUtil.instance.severe(extra, colorize(String.format("%s", message), Color.RED));
}
// needs to be private now that the main logic is handling this
private Client() {
    loggerUtil.INSTANCE.info("Client created!");
}
public boolean isDisconnected() {
    if (server != null) {
        return false;
    }
}

```

Snippet of code for /connect and /name



⇒ Text Prompt

Weight: 50%

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:

For /connect the client does /connect localhost:3000 which is matched against the regex inside the code. If it matches, then the method connectToServer is used to connect. As for /name the client name would be passed into the sendName(String name) method. This would be used to send the name over to the server.



Saved: 4/13/2025 10:17:04 PM

100%

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

100%

Task #1 (2 pts.) - Evidence

Combo Task:

Weight: 100%

Objective: Evidence

⇒ Image Prompt

Weight: 50%

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

joining

Terminal output

Evidence of successful creation/join in both scenario

Snippet of code



Saved: 4/13/2025 12:33:10 AM

≡, Text Prompt

Weight: 50%

Details:

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

Your Response:

The / createroom command would be detected and if it is a room is created. Same thing for /joinroom, if it detected the client joins the room.



Saved: 4/13/2025 12:33:10 AM

100%

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

100%

Task #1 (1 pt.) - Evidence

Combo Task:

Weight: 100%

Objective: Evidence

→ Image Prompt

Weight: 50%

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

A screenshot of a terminal window showing three separate sessions. Each session has a different color background (light blue, light green, and light yellow) and displays several lines of text, likely representing messages sent by clients to a room. The text is too small to read individually but shows a pattern of messages being exchanged.

Terminal output of messages



Examples of clients grouped

```
    * 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);
```

snippet of code Client side

server-side snippet code



Saved: 4/13/2025 3:07:19 PM

Text Prompt

Weight: 50%

Details:

- Briefly explain how the message code flow works.

Your Response:

100%

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

100%

Task #1 (1 pt.) - Evidence

Combo Task:

Weight: 100%

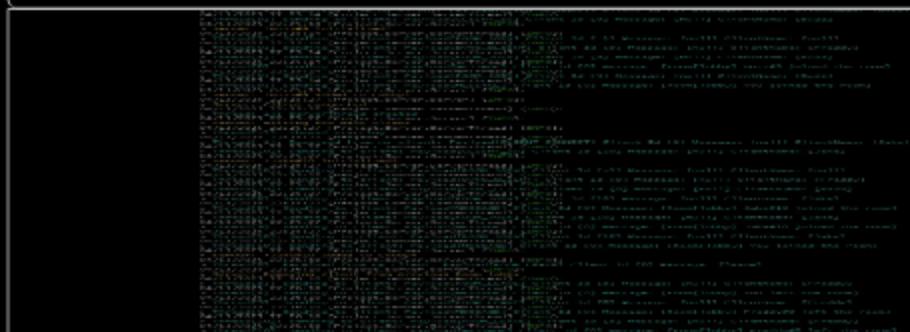
Objective: Evidence

→ Image Prompt

Weight: 50%

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 occurring
- 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



example of clients disconnecting



example of servers disconnecting after a client has disconnected the connection will remain



examples of server disconnecting with clients leaving server and using the command /kill server

clients reconnecting when a server is brought back

```
/**  
 * Sends a disconnect action to the server  
 *  
 * @throws IOException  
 */  
private void sendDisconnected() throws IOException {  
    Payload payload = new Payload();  
    payload.setPayloadType(PayloadType.DISCONNECT);  
    sendToServer(payload);  
}
```

Snippets of code that handle the client side

snippet of code that handle the server side



Saved: 4/13/2025 11:33:54 PM

Section #8: (1 pt.) Misc

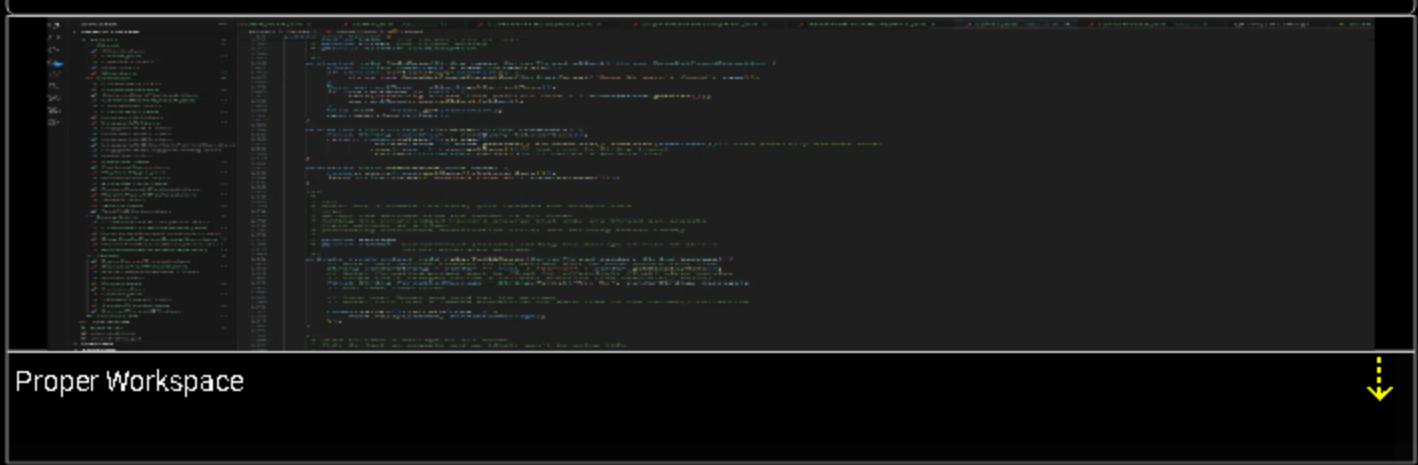
Task #1 (0.25 pts.) - Show the proper workspace structure with the

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

Image Prompt

Weight: 25%

Objective: Show the proper workspace structure with the new Client, Common, Server, and Exceptions packages



Proper Workspace



Saved: 4/14/2025 6:54:32 PM

100%

Task #2 (0.25 pts.) - Github Details

Combo Task:

Weight: 25%

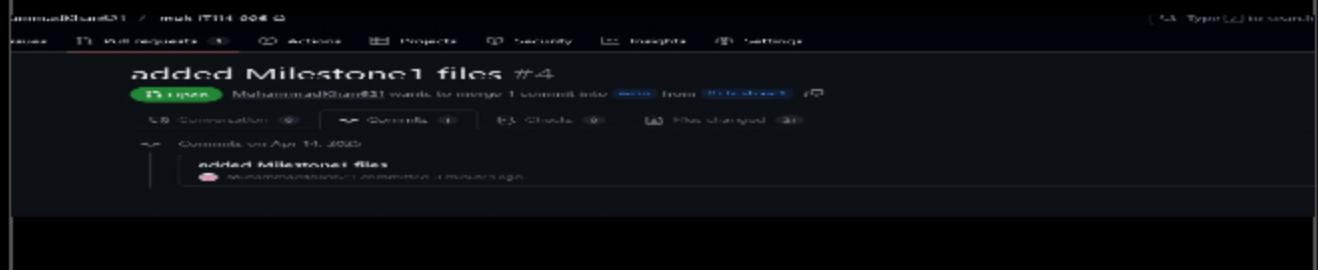
Objective: Github Details

Image Prompt

Weight: 60%

Details:

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



Commit history



Saved: 4/14/2025 8:08:56 PM

⇒ Url Prompt

Weight: 40%

Details:

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

URL #1

<https://github.com/MuhammadKhan621/muk-IT114-006/pull/4/commits>



URL

<https://github.com/MuhammadKhan621/muk-IT114-006/pull/4/commits>



Saved: 4/14/2025 8:08:56 PM

100%

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

📷 Image Prompt

Weight: 25%

Objective: WakaTime - Activity

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

File	Time
40 minutes	Project1/Client/Client.java
24 minutes	Project1/Server/Server.java
19 minutes	Project1/Client/User.java
18 minutes	Project1/Server/User.java
12 minutes	Project1/Client/ClientThread.java
10 minutes	Project1/Client/UserThread.java
9 minutes	Project1/Client/UserThread.java
8 minutes	Project1/Client/UserThread.java
7 minutes	Project1/Client/UserThread.java
6 minutes	Project1/Client/UserThread.java
5 minutes	Project1/Client/UserThread.java
4 minutes	Project1/Client/UserThread.java
3 minutes	Project1/Client/UserThread.java
2 minutes	Project1/Client/UserThread.java
1 minute	Project1/Client/UserThread.java
33 seconds	Project1/Client/UserThread.java
23 seconds	Project1/Client/UserThread.java
22 seconds	Project1/Client/UserThread.java

2 hrs 20 mins 1000ms

individual time



100%

Task #4 (0.25 pts.) - Reflection

Weight: 25%

Objective: *Reflection*

Sub-Tasks:

100%

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

→ Text Prompt

Weight: 33.33%

Objective: *What did you learn?*

Details:

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

Your Response:

I learned how to start up the server, create clients, give names to those clients, create rooms, leave rooms, also learned how /listrooms would show the lists of rooms, quitting would terminate the clients and also doing /kill would shutdown server. I also learned how some commands happen between client and server classes.



Saved: 4/13/2025 11:50:47 PM

100%

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

→ Text Prompt

⇒ Text Prompt

Weight: 33.33%

Objective: *What was the easiest part of the assignment?*

Details:

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

Your Response:

The easiest part of the assigment learning how to use each commands when performing the Milestone1 assigment. Another easy part was understanding how the server-side handles multiple connected clients.



Saved: 4/14/2025 12:01:20 AM



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

⇒ Text Prompt

Weight: 33.33%

Objective: *What was the hardest part of the assignment?*

Details:

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

Your Response:

The hardest part was understanding some parts of the server class. (not all) Another hard part was understanding some part of the client class(not all).



Saved: 4/14/2025 12:13:34 AM