Submission Worksheet

Submission Data

Course: IT114-007-F2025
Assignment: IT114 Milestone 1
Student: Lukas P. (lap5)
Status: Submitted | **Worksheet Progress:** 100%
Potential Grade: 10.00/10.00 (100.00%)
Received Grade: 0.00/10.00 (0.00%)
Started: 11/24/2025 2:16:12 AM
Updated: 11/24/2025 1:03:47 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-007-F2025/it114-milestone-1/grading/lap5
View Link: https://learn.ethereallab.app/assignment/v3/IT114-007-F2025/it114-milestone-1/view/lap5

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

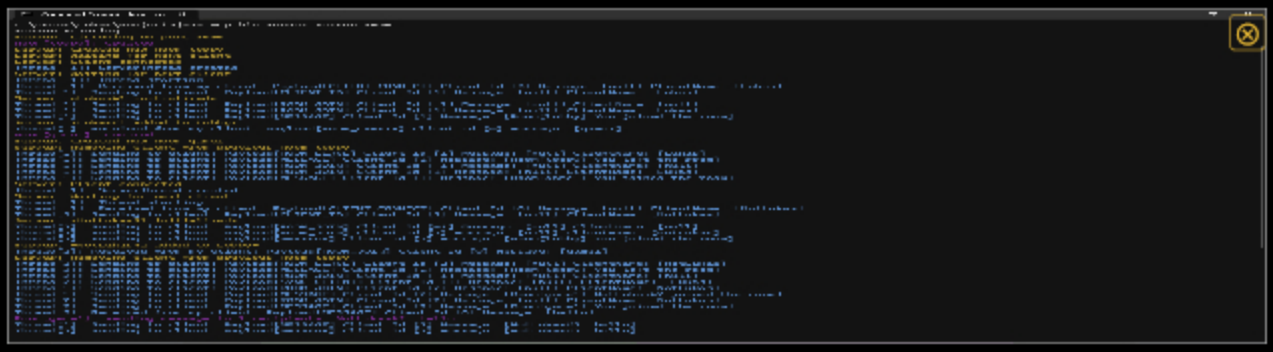
Progress: 100%

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



cmd line

```
try (ServerSocket serverSocket = new ServerSocket(port)) {
    createRoom(Room.LOBBY);// create the first room (lobby)
    while (isRunning) {
        info("Waiting for next client");
        Socket incomingClient = serverSocket.accept(); // blocking action, waits for a client connection
        info("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
    }
```

incoming

Saved: 11/24/2025 2:44:12 AM

Part 2:

Progress: 100%

Details:

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

Your Response:

ServerSocket listens on a port. A loop calls accept() to receive a Socket, wraps it in ServerThread, and starts it.

Saved: 11/24/2025 2:44:12 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

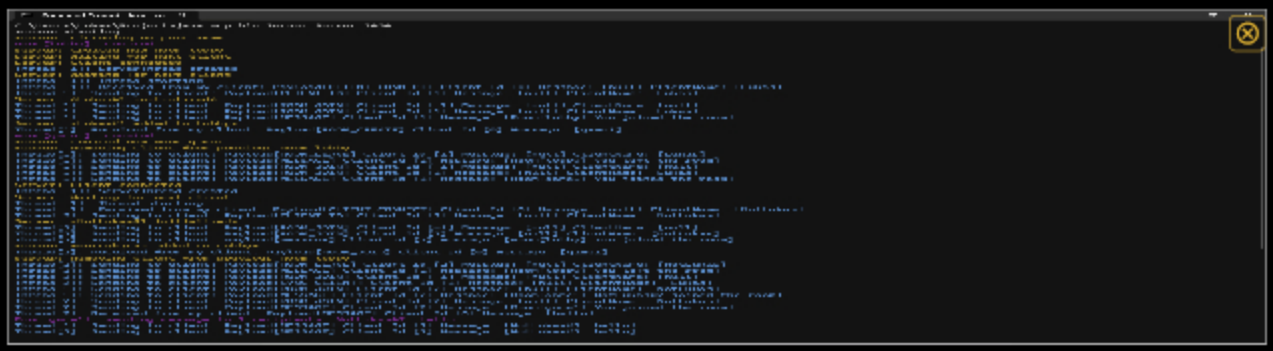
Progress: 100%

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



connections

```
private void sendMessage(String message) throws IOException {
    Payload payload = new Payload();
    payload.setMessage(message);
    payload.setPayloadType(PayloadType.MESSAGE);
    sendToServer(payload);
}
```

code

Saved: 11/24/2025 12:35:42 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

Each client runs in a separate ServerThread. The Server tracks them using ConcurrentHashMap and manages them within Room instances.

Saved: 11/24/2025 12:35:42 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

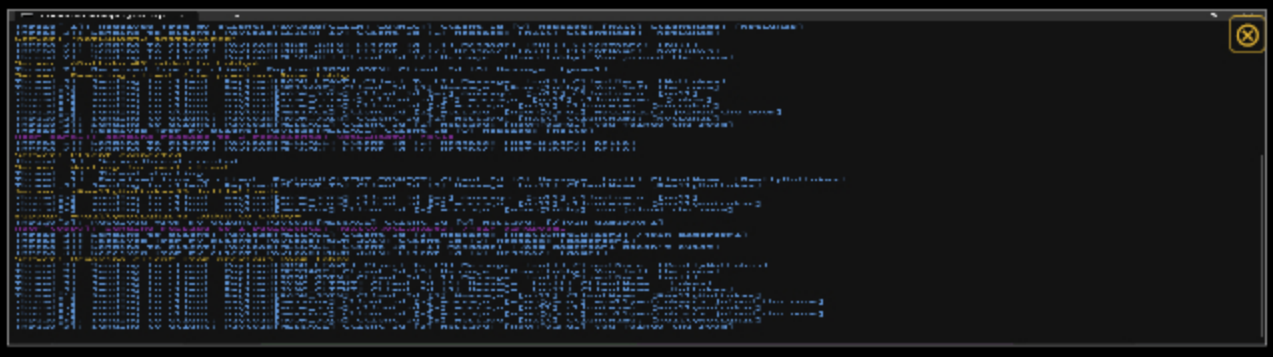
Progress: 100%

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)



terminal

code

Saved: 11/24/2025 12:31:11 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

Server maps names to Room objects. Methods create or find rooms, move ServerThreads between them, and remove empty rooms.

Saved: 11/24/2025 12:31:11 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

```
        } else if (text.startsWith(Command.NAME.command)) {
            text = text.replace(Command.NAME.command, "").trim();
            if (text == null || text.length() == 0) {
                System.out.println(TextFX.colorize("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("Name set to %s", myUser.getClientName()),
                    Color.YELLOW));
            wasCommand = true;
```

code

Saved: 11/24/2025 12:38:07 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 sets local name. /connect opens socket, sends name payload. Server assigns ID, adds to Lobby, and sends confirmation.

Saved: 11/24/2025 12:38:07 PM

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

Progress: 100%

Task #1 (2 pts.) - Evidence

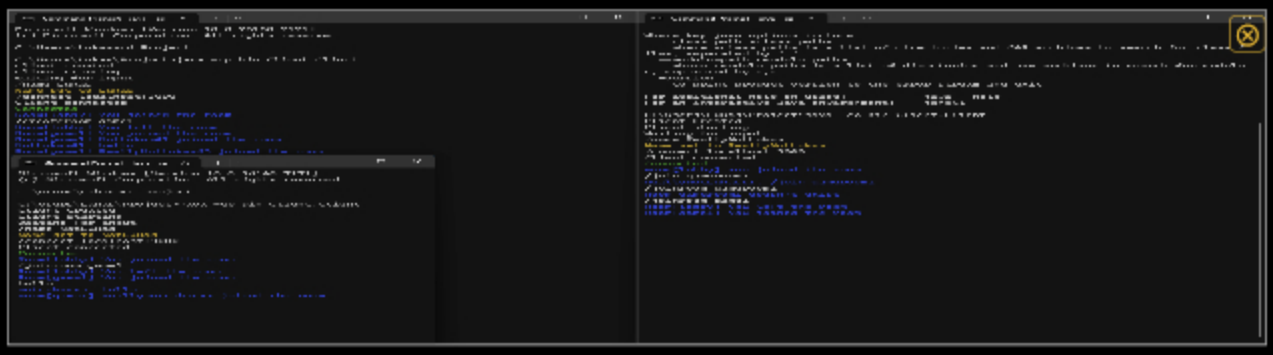
Progress: 100%

Part 1:

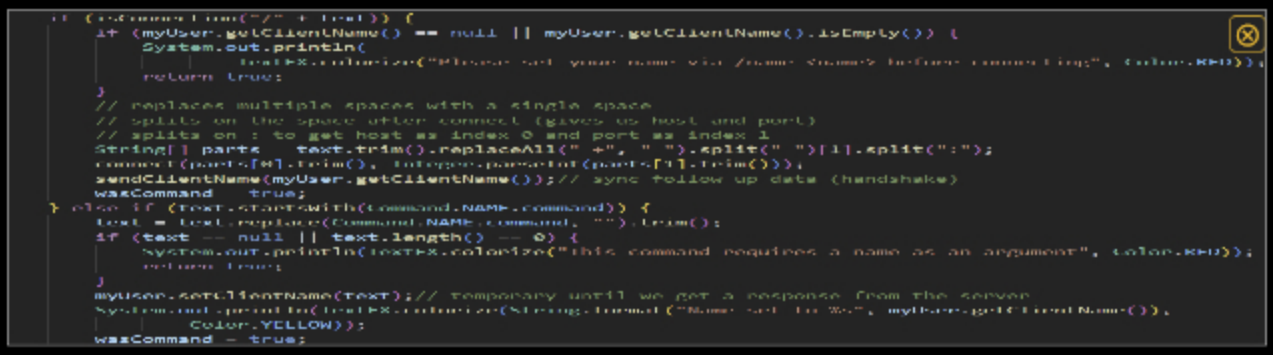
Progress: 100%

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



```
if (text.startsWith("/" + text)) {
    if (myUser.getClientName() == null || myUser.getClientName().isEmpty()) {
        System.out.println(
                textFX.colorize("Please set your name via /name <name> before connecting", Color.RED));
        return true;
    }
    // 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(" +", " ").split(" ")[1].split(":");
    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, "").trim();
    if (text == null || text.length() == 0) {
        System.out.println(textFX.colorize("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("Name set to %s", myUser.getClientName()),
            Color.YELLOW));
    wasCommand = true;
```

code

Saved: 11/24/2025 12:38:55 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:

Client sends payload. ServerThread calls Room handler. Server creates/finds room, moves client, and syncs new room state.

Saved: 11/24/2025 12:38:55 PM

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

Messages

Progress: 100%

Task #1 (1 pt.) - Evidence

Progress: 100%

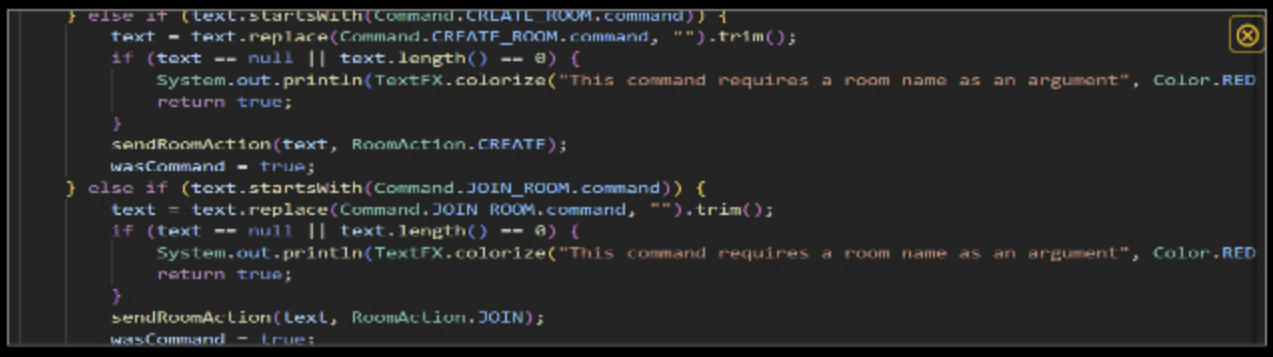
Part 1:

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



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

code

Saved: 11/24/2025 12:42:13 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the message code flow works

Your Response:

Client sends payload. ServerThread calls Room handler. Room iterates clients, sending formatted

message payload to each connected client.

Saved: 11/24/2025 12:42:13 PM

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

Progress: 100%

Task #1 (1 pt.) - Evidence

Progress: 100%

Part 1:

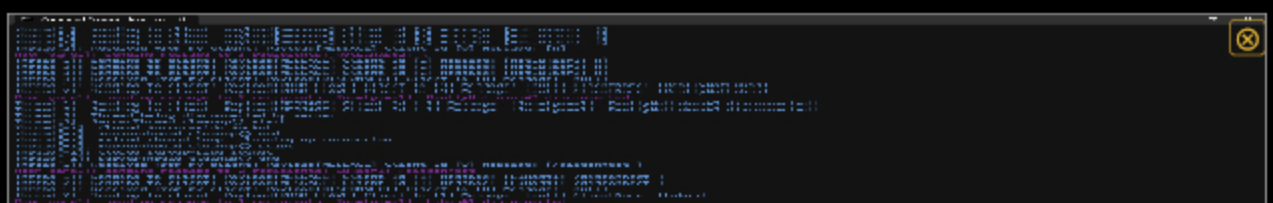
Progress: 100%

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

terminal

terminal



terminal

```java
private void sendMessage(String message) throws IOException {
    Payload payload = new Payload();
    payload.setMessage(message);
    payload.setPayloadType(PayloadType.MESSAGE);
    sendToServer(payload);
}
```

code

```
            } else if (Command.QUIT.command.equalsIgnoreCase(text)) {
                close();
                wasCommand = true;
            } else if (Command.DISCONNECT.command.equalsIgnoreCase(text)) {
                sendDisconnect();
                wasCommand = true;
            } else if (text.startsWith(Command.REVERSE.command)) {
                text = text.replace(Command.REVERSE.command, "").trim();
                sendReverse(text);
                wasCommand = true;
```

code

Saved: 11/24/2025 12:48:44 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

Client sends disconnect payload. Server removes client, notifies room, closes socket. Client loop ends, closing resources.

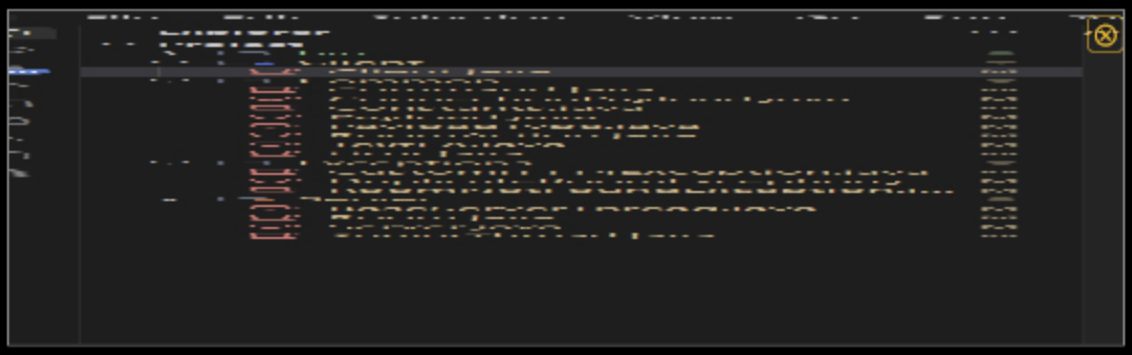
Saved: 11/24/2025 12:48:44 PM

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: 11/24/2025 12:49:20 PM

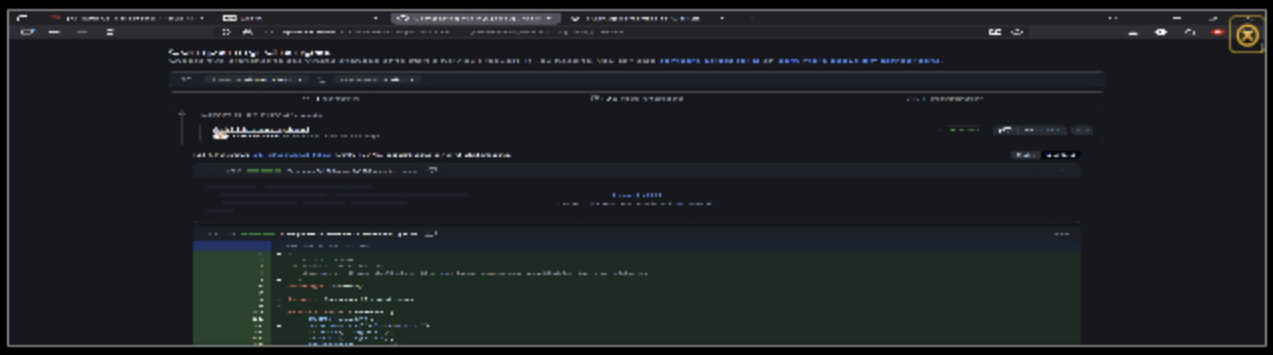
Task #2 (0.25 pts.) - Github Details

Progress: 100%

Part 1:

Progress: 100%

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



commits (i think)

Saved: 11/24/2025 1:01:41 PM

Part 2:

Progress: 100%

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

URL #1

URL

https://github.com/LukasPresti/lap5-it1114-007/compare/main%40%7B1day%7D...main

https://github.com/LukasPresti/la

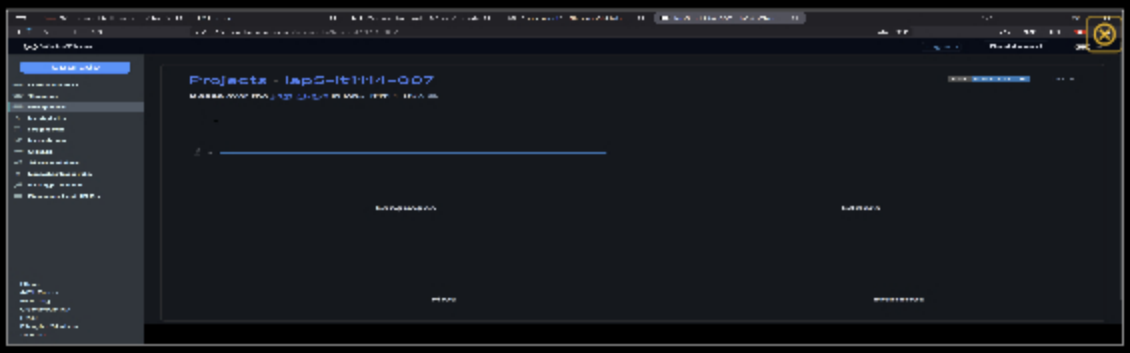
Saved: 11/24/2025 1:01:41 PM

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 for some reason didnt track anything I did for this milestone. No idea what happened

Saved: 11/24/2025 1:03:47 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 really dislike coding

Saved: 11/24/2025 12:49:58 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:

taking the pictures

Saved: 11/24/2025 12:50:05 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:

the rest of it

Saved: 11/24/2025 12:50:11 PM