Submission Worksheet

Submission Data

Course: IT114-003-F2025
Assignment: IT114 Milestone 1
Student: Bryan S. (bs768)
Status: Submitted | **Worksheet Progress:** 92%
Potential Grade: 9.00/10.00 (90.00%)
Received Grade: 0.00/10.00 (0.00%)
Started: 11/4/2025 1:23:51 PM
Updated: 11/4/2025 2:47:37 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-003-F2025/it114-milestone-1/grading/bs768
View Link: https://learn.ethereallab.app/assignment/v3/IT114-003-F2025/it114-milestone-1/view/bs768

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

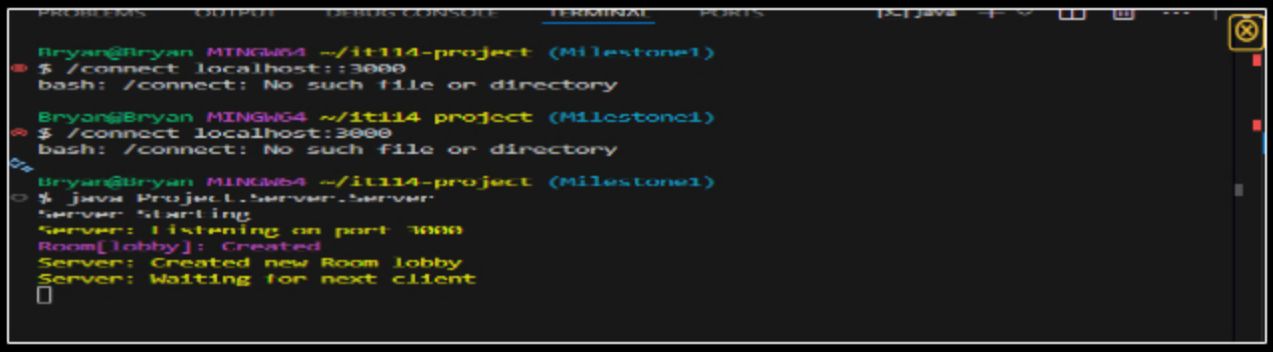
Progress: 100%

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



Terminal of the server.

```
private void start(int port) {
    this.port = port;
    // server listening
    // bs/08,11/4,code that waits for connections
    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("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
```

Code that waits for connection.

Saved: 11/4/2025 1:15:57 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

Once the server is running it will create a lobby where clients join. It wil run in a loop always checking to see if there are new connections waiting to join.

Saved: 11/4/2025 1:15:57 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

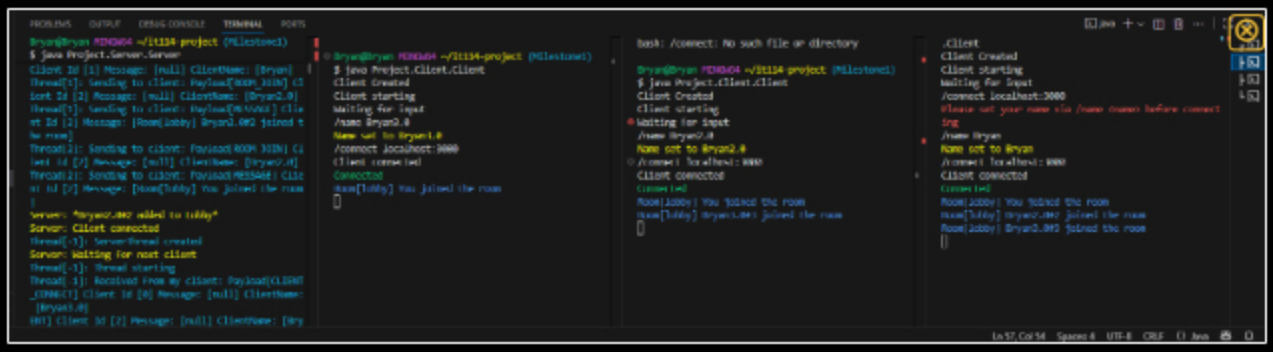
Progress: 100%

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



Terminal Output & 3 Clients Connected.

```
protected void joinRoom(String name, ServerThread client) throws RoomNotFoundException {
    //bs768,11/4,code that shows how many clients are in a room
    final String nameCheck = name.toLowerCase();
    if (!rooms.containsKey(nameCheck)) {
        throw new RoomNotFoundException(String.format("Room %s wasn't found", name));
    }
    Room currentRoom = client.getCurrentRoom();
    if (currentRoom != null) {
        info("Removing client from previous Room " + currentRoom.getName());
        currentRoom.removeClient(client);
    }
    Room next = rooms.get(nameCheck);
    next.addClient(client);
}
```

Shows code that handles logic for multiple connections.

Saved: 11/4/2025 1:34:25 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

The server waits for clients to connect, and when one does, creates a thread to handle that client. This way, each client runs independently, so multiple people can stay connected and talk at the same time without slowing each other down.

Saved: 11/4/2025 1:34:25 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)

```
Thread[3]: Received from my client: Payload[ROOM_CR
EATE] Client Id [0] Message: [Room1]
Room[Room1]: Created
Server: Created new Room Room1
Server: Removing client from previous Room lobby
: [Room[lobby] You left the room]
```

Creating Room

```
Server: Removing client from previous Room lobby
Thread[1]: Sending to client: Payload[ROOM_LEAVE] C
lient Id [2] Message: [null] ClientName: [Bryan2.0]

Thread[1]: Sending to client: Payload[MESSAGE] Clie
nt Id [2] Message: [Room[lobby] Bryan2.0#2 left the
 room]
Thread[2]: Sending to client: Payload[ROOM_LEAVE] C
lient Id [2] Message: [null] ClientName: [Bryan2.0]

Thread[2]: Sending to client: Payload[MESSAGE] Clie
nt Id [2] Message: [Room[lobby] You left the room]
```

joining room.

```
Server: Removing client from previous Room Room1
Thread[1]: Sending to client: Payload[ROOM_LEAVE] C
lient Id [3] Message: [null] ClientName: [Bryan3.0]
Thread[1]: Sending to client: Payload[MESSAGE] Clie
nt Id [3] Message: [Room[Room1] Bryan3.0#3 left the
 room]
Thread[2]: Sending to client: Payload[ROOM_LEAVE] C
lient Id [3] Message: [null] ClientName: [Bryan3.0]
Thread[2]: Sending to client: Payload[MESSAGE] Clie
nt Id [3] Message: [Room[Room1] Bryan3.0#3 left the
 room]
```

Removing client from room.

```
protected synchronized void removeClient(ServerThread client) {
    if (!isRunning) { // block action if Room isn't running
        return;
    }
    if (!clientsInRoom.containsKey(client.getClientId())) {
        info("Attempting to remove a client that doesn't exist in the room");
        return;
    }
    ServerThread removedClient = clientsInRoom.get(client.getClientId());
    if (removedClient != null) {
        // notify clients of someone joining
        joinStatusRelay(removedClient, false);
        clientsInRoom.remove(client.getClientId());
        autoCleanup();
    }
}
```

Leave/Remove

```
//bs768,11/4,code that creates a room
protected void createRoom(String name) throws DuplicateRoomException {
    final String nameCheck = name.toLowerCase();
    if (rooms.containsKey(nameCheck)) {
        throw new DuplicateRoomException(String.format("Room %s already exists", name));
    }
    Room room = new Room(name);
    rooms.put(nameCheck, room);
    info(String.format("Created new Room %s", name));
}
```

Create Room

```
//bs768,11/4,code that joins a room
protected void joinRoom(String name, ServerThread client) throws RoomNotFoundException {
    //bs768,11/4,code that shows how many clients are in a room
    final String nameCheck = name.toLowerCase();
    if (!rooms.containsKey(nameCheck)) {
        throw new RoomNotFoundException(String.format("Room %s wasn't found", name));
    }
    Room currentRoom = client.getCurrentRoom();
    if (currentRoom != null) {
        info("Removing client from previous Room " + currentRoom.getName());
        currentRoom.removeClient(client);
    }
```

```
    Room next = rooms.get(nameCheck);
    next.addClient(client);
}
```

Join Room

Saved: 11/4/2025 1:54:53 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

The server lets users make new rooms and join them when they want to switch. When someone leaves a room, the server removes them from that room's user list.

Saved: 11/4/2025 1:54:53 PM

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

Progress: 100%

Task #1 (1 pt.) - Evidence

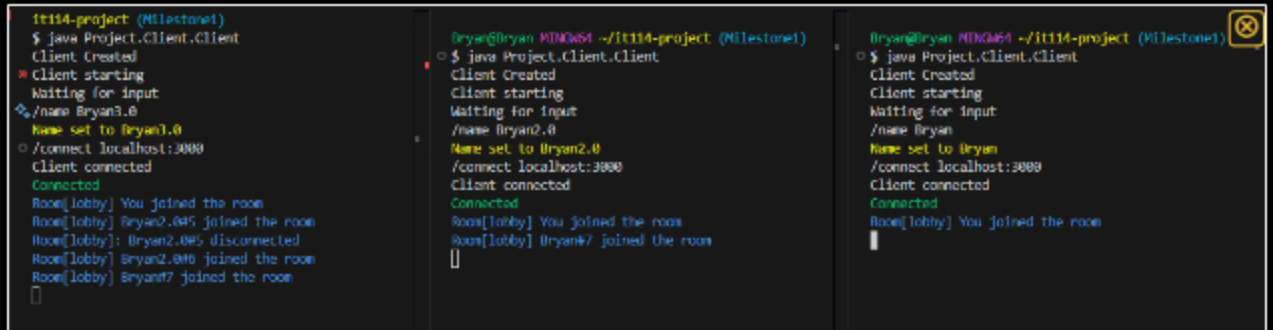
Progress: 100%

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 output

```
private boolean processClientCommand(String text) throws IOException {
    boolean wasCommand = false;
    //bs768,11/4,code that processes client commands
    if (text.startsWith(Constants.COMMAND_TRIGGER)) {
        text = text.substring(1); // remove the /
        // System.out.println("Checking command: " + text);
        if (isConnection("/" + 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;
            }
```

/name

```
public enum Client {
    INSTANCE;
    //bs768,11/4
    private Socket server = null;
    private ObjectOutputStream out = null;
    private ObjectInputStream in = null;
    final Pattern ipAddressPattern = Pattern
            .compile("/connect\\s+(\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}:\\d{3,5})");
    final Pattern localhostPattern = Pattern.compile("/connect\\s+(localhost:\\d{3,5})");
    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();
```

/connect

```
private boolean connect(String address, int port) {
    try {
        server = new Socket(address, port);
        // channel to send to server
        out = new ObjectOutputStream(server.getOutputStream());
        // channel to listen to server
        in = new ObjectInputStream(server.getInputStream());
        System.out.println("Client connected");
        // Use CompletableFuture to run listenToServer() in a separate thread
        CompletableFuture.runAsync(this::listenToServer);
    } catch (UnknownHostException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return isConnected();
}
```

confirmation

Saved: 11/4/2025 2:08:05 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:

When you use the /name command, the client saves your chosen name. Then, with the /connect command, it connects to the server using the port you provide. After connecting, the client sends

your name to the server, which responds by assigning you an ID this finishes setting up the connection.

Saved: 11/4/2025 2:08:05 PM

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

Progress: 100%

Task #1 (2 pts.) - Evidence

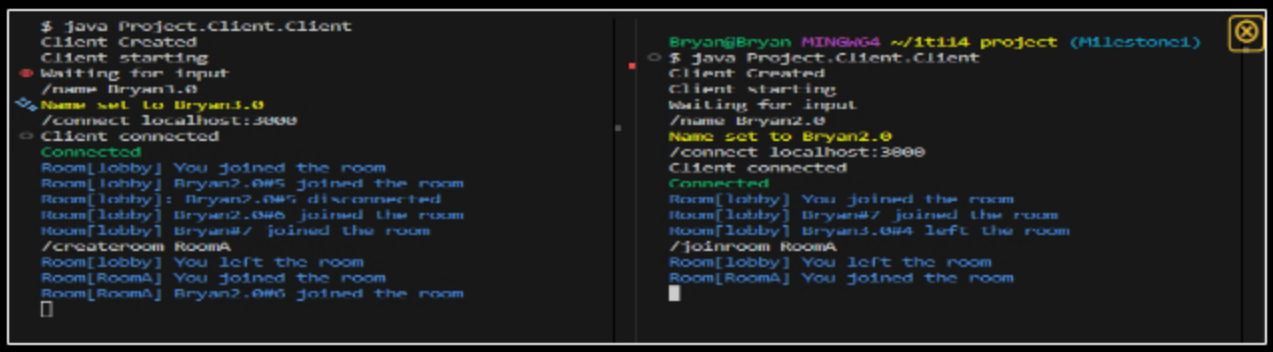
Progress: 100%

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 & /joinroom

Saved: 11/4/2025 2:12:06 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 /createroom command creates a new room and adds the client to it, or tells them if it already exists. The /joinroom command adds the client to an existing room or notifies them if the room

doesn't exist.

Saved: 11/4/2025 2:12:06 PM

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

Progress: 0%

Task #1 (1 pt.) - Evidence

Progress: 0%

Part 1:

Progress: 0%

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

Missing Caption

Saved: 11/3/2025 10:13:01 AM

Part 2:

Progress: 0%

Details:

- Briefly explain how the message code flow works

Your Response:

Missing Response

Saved: 11/3/2025 10:13:01 AM

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

Progress: 100%

Task #1 (1 pt.) - Evidence

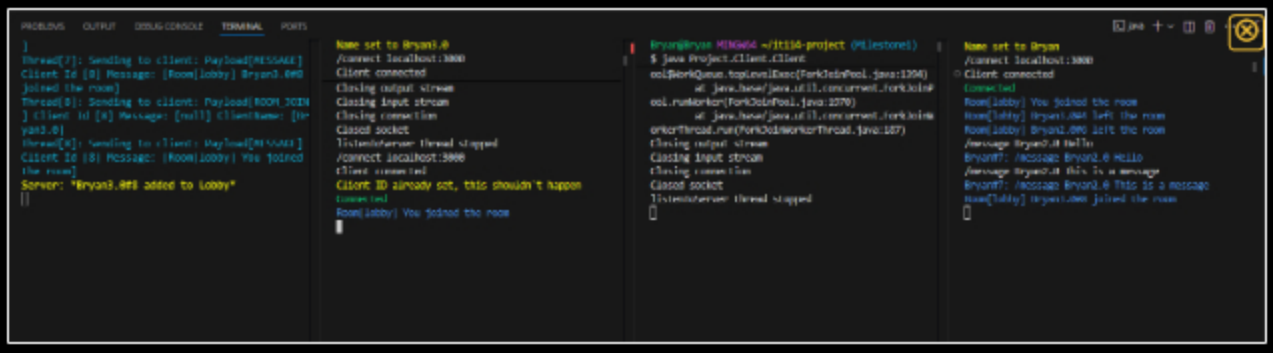
Progress: 100%

Part 1:

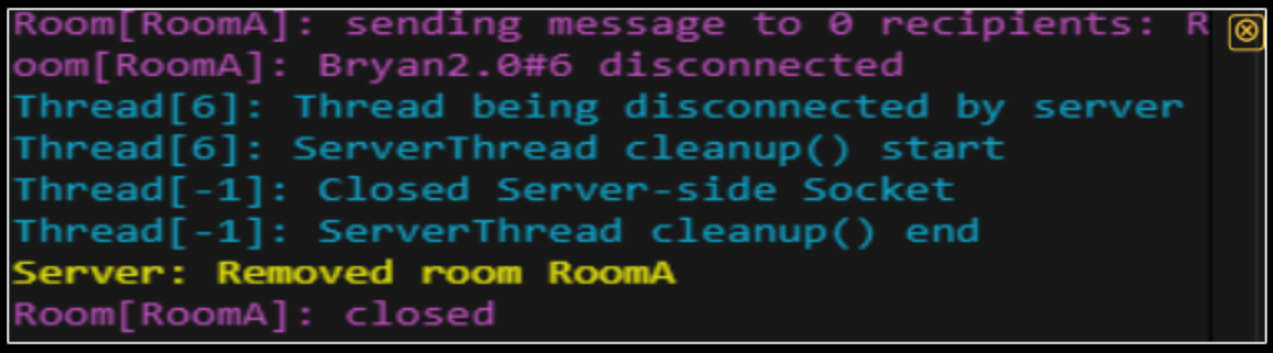
Progress: 100%

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



Disconnecting client side/ Reconnecting Server



Disconnect server side

```java
//bs768,11/4,code that disconnects a client
protected void disconnect() {
    if (!isRunning) {
        // prevent multiple triggers if this gets called consecutively
        return;
    }
    info("Thread being disconnected by server");
    isRunning = false;
    this.interrupt(); // breaks out of blocking read in the run() method
    cleanup(); // good practice to ensure data is written out immediately
}
```

Server Side

```java
//bs768,11/4,code that disconnects a client
private void processDisconnect(Payload payload) {
    if (payload.getClientId() == myUser.getClientId()) {
        knownClients.clear();
        myUser.reset();
        System.out.println(TextFX.colorize("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("%s disconnected", disconnectedUser.getDisplayName()),
                    Color.RED));
        }
    }
```

Client Side

Saved: 11/4/2025 2:30:45 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

The client handles disconnecting by telling the server it's leaving and then closing its connection and stopping its tasks. The server stops the client's thread, cleans up its connection, and lets the chat room know the client left.

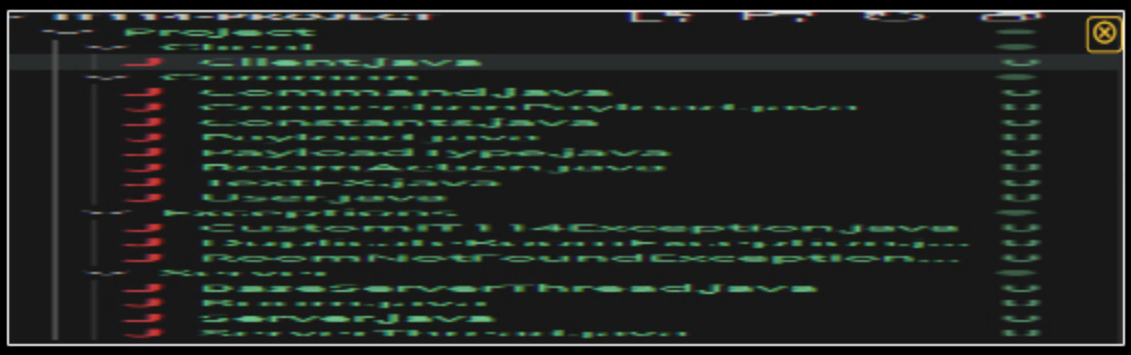
Saved: 11/4/2025 2:30:45 PM

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%



Workspace

Saved: 11/4/2025 2:32:09 PM

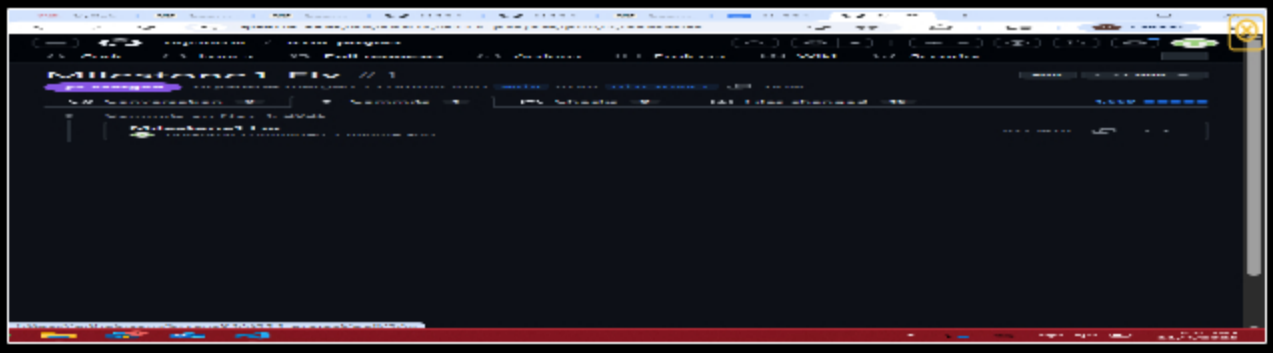
Task #2 (0.25 pts.) - Github Details

Progress: 100%

Part 1:

Progress: 100%

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



Commit history

Saved: 11/4/2025 2:36:09 PM

Part 2:

Progress: 100%

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

URL #1
https://github.com/bryans04/it114-project/pull/1/commits

URL
https://github.com/bryans04/it11

Saved: 11/4/2025 2:36:09 PM

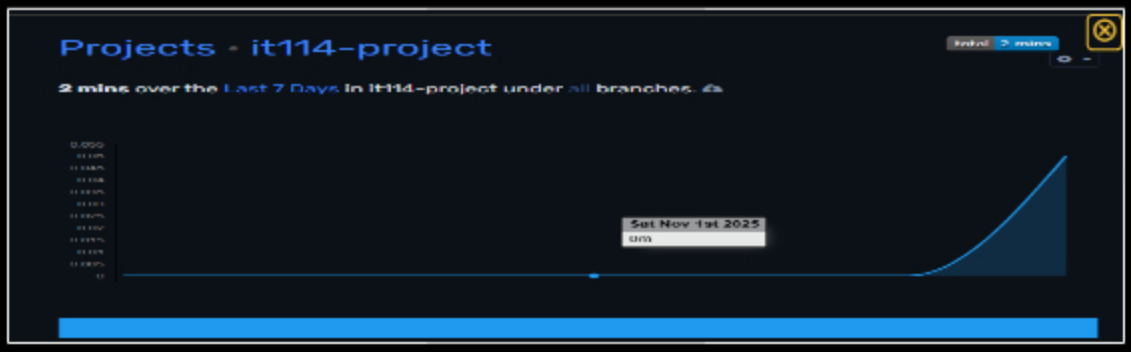
Task #3 (0.25 pts.) - WakaTime - Activity

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



I had to reinstall WakaTime and enter the API key as my new repo wasnt showing up.

Saved: 11/4/2025 2:44:57 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 did learn a lot from this specific project, It was like a puzzle because I had to keep solving each error one at a time until it finally compiled.

Saved: 11/4/2025 2:46:04 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:

The easiest part was probably just the structuring the folders and putting each file

into the correct folder.

Saved: 11/4/2025 2:46:41 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:

It wasn't necessarily hard just a bit annoying but, having to find what import statement each file was missing was the only "hard" part of this assignment.

Saved: 11/4/2025 2:47:37 PM