

# Submission Worksheet

CLICK TO GRADE

<https://learn.ethereallab.app/assignment/IT114-451-M2024/it114-module-5-project-milestone-1/grade/st278>

IT114-451-M2024 - [IT114] Module 5 Project Milestone 1

## Submissions:

Submission Selection

1 Submission [active] 6/23/2024 6:18:44 AM

## Instructions

^ COLLAPSE ^

Overview Video: <https://youtu.be/A2yDMS9TS1o>

1. Create a new branch called Milestone1
2. At the root of your repository create a folder called Project if one doesn't exist yet
  1. You will be updating this folder with new code as you do milestones
  2. You won't be creating separate folders for milestones; milestones are just branches
3. Copy in the code from Sockets Part 5 into the Project folder (just the files)
  2. <https://github.com/MattToegel/IT114/tree/M24-Sockets-Part5>
4. Fix the package references at the top of each file (these are the only edits you should do at this point)
5. Git add/commit the baseline and push it to github
6. Create a pull request from Milestone1 to main (don't complete/merge it yet, just have it in open status)
7. Ensure the sample is working and fill in the below deliverables 1. Note: Don't forget the client commands are /name and /connect
8. Generate the output file once done and add it to your local repository
9. Git add/commit/push all changes
10. Complete the pull request merge from the step in the beginning
11. Locally checkout main
12. git pull origin main

Branch name: Milestone1

Tasks: 8 Points: 10.00

^COLLAPSE ^

^COLLAPSE ^

**Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)**

**Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade. The goal is to show you understand what segments are related to the prompts.**



**Caption (required)**  
Describe/highlight  
what's being shown  
(ucid/date must be  
present)

**Explanation (required)**

✓

*Briefly explain the code/logic/flow leading up to and including waiting for user input*

The client application sets up a connection to a server, using a singleton Client class. It initializes socket communication, regular expression patterns for commands, and client

it waits for client connections using a `ServerSocket`. When a client connects, it accepts the connection, creates a `ServerThread` to handle the client, and starts the thread. This process continues while the server is running. If an error occurs during connection acceptance, it prints an error message, and the finally block ensures the server performs necessary cleanup, such as closing the server socket and disconnecting all clients.

data. When started, the client listens for user input in a separate thread, processing commands like connecting to the server, setting the client name, and managing rooms. These commands are handled by the `processClientCommand()` method, which performs the necessary actions or sends messages to the server. The client also listens for messages from the server in a separate thread, updating the client's state or displaying messages as needed. Helper methods handle sending various types of data to the server, such as creating or joining rooms. Cleanup is ensured by closing socket connections and I/O streams when the client shuts down. The application is initiated from the `main()` method, starting the client and managing its lifecycle.

## Task #2 - Points: 1

Text: Connecting

### Details:

Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

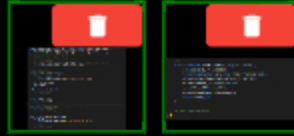
Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade. The goal is to show you understand what segments are related to the prompts.

#1) Show 3  
Clients  
connecting



**Caption (required)** ✓  
*Describe/highlight  
what's being shown*  
3 connections

#2) Show the  
code related  
to Clients



**Caption (required)** ✓  
*Describe/highlight  
what's being shown  
(ucid/date must be  
present)*  
Showing the code  
related to Clients  
connecting to the Server

**Explanation (required)** ✓  
*Briefly explain the  
code/logic/flow*

PREVIEW RESPONSE

The `ServerThread` class encapsulates the server-side logic for individual client connections. When a client connects to the server, a `ServerSocket` listens for incoming connections on a specified port. Upon connection, a new `ServerThread` instance is instantiated, which initializes with the client's socket and a callback mechanism (`onInitializationComplete`). This class manages crucial operations such as setting the client's identity (`clientId` and `clientName`), processing various message payloads (`PayloadType`), and facilitating actions like joining rooms (`ROOM_JOIN`) or disconnecting (`DISCONNECT`). Each

(DISCONNECT). Each ServerThread operates within its own thread context, ensuring concurrent handling of multiple clients and maintaining robust communication channels between the server and connected clients.

Communication (3 pts.)

^COLLAPSE ^

Task #1 - Points: 1

Text: Communication

#### Details:

Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade. The goal is to show you understand what segments are related to the prompts.

#1) Show each Client sending and



**Caption (required) ✓**  
*Describe/highlight what's being shown*  
Showing each Client sending and receiving messages

#2) Show the code related to the Client-



**Caption (required) ✓**  
*Describe/highlight what's being shown (ucid/date must be present)*  
Showing the code related to the Client-side of getting a user message and sending it over the socket

#3) Show the code related to the Server-



**Caption (required) ✓**  
*Describe/highlight what's being shown (ucid/date must be present)*  
Showing the code related to the Server-side receiving the message and relaying it to each connected Client

#4) Show the code related to the Client-



**Caption (required) ✓**  
*Describe/highlight what's being shown (ucid/date must be present)*  
Showing the code related to the Client receiving messages from the Server-side and presenting them

#### Explanation (required)



*Briefly explain the code/logic/flow involved*

 PREVIEW RESPONSE

The `listenToInput()` method reads user input from the console, checks for client commands, and sends messages to the server if connected. It prompts the user to connect if not already connected and handles errors gracefully. This method manages user interaction with the client application, ensuring seamless communication with the server.

#### Explanation (required)



*Briefly explain the code/logic/flow involved*

 PREVIEW RESPONSE

The `Payload` class in the provided Java code encapsulates data transmitted between server and clients in a networked application. It defines fields for `payloadType` (indicating the type of data being sent), `clientId` (identifying the sender or recipient), and `message` (content being transmitted). This class is designed to be serialized, allowing its instances to be efficiently transmitted over networks. Its `toString()` method provides a formatted string representation for debugging and logging purposes, summarizing the payload's type, client ID, and message content. Overall, `Payload` simplifies the handling and exchange of structured data within the application's communication framework.

#### Explanation (required)



*Briefly explain the code/logic/flow involved*

 PREVIEW RESPONSE

The `start` method in the `Server` class initializes the server to listen on a specified port, creates an initial "LOBBY" room, and enters a loop where it waits for client connections using a `ServerSocket`. When a client connects, it accepts the connection, creates a `ServerThread` to handle the client, and starts the thread. This process continues while the server is running. If an error occurs during connection acceptance, it prints an error message, and the `finally` block ensures the server performs necessary cleanup, such as closing the server socket and disconnecting all clients.

 ^COLLAPSE ^

Task #2 - Points: 1

Text: Rooms

#### Details:

Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)



**Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade. The goal is to show you understand what segments are related to the prompts.**


## #1) Show Clients can Create



**Caption (required)** ✓  
*Describe/highlight what's being shown*  
Showing Clients can Create Rooms

## #2) Show Clients can Join Rooms



**Caption (required)**   
*Describe/highlight what's being shown*  
 Showing Clients can Join Rooms

### #3) Show the Client code related to the



**Caption (required)** ✓  
*Describe/highlight what's being shown (ucid/date must be present)*  
Showing the Client code related to the create/join room commands

**Explanation (required)**



**Briefly explain the code/logic/flow involved**

 PREVIEW RESPONSE

In the Client program, commands entered by the user start with / and are processed to either create or join rooms. The `processClientCommand(String text)` method checks if the command is for creating (`createroom`) or joining (`joinroom`) a room, extracts the room name from the command, and sends a corresponding message (`Payload`) to the server. The `sendCreateRoom(String room)` method prepares a message indicating the intention to create a room with the specified

#### #4) Show the ServerThread/Router code



**Caption (required)** ✓  
*Describe/highlight what's being shown (ucid/date must be present)*  
Showing the ServerThread/Room code handling the create/join process

**Explanation (required)**



**Briefly explain the code/logic/flow involved**

 PREVIEW RESPONSE

In the Room class, messages from the server to clients are managed through synchronized methods like `sendMessage`, `sendRoomStatus`, and `sendDisconnect`. These methods ensure that messages are sent safely to all clients in the room without conflicts.

`sendMessage` broadcasts messages from either a client or the server to all clients in `clientsInRoom`, removing any client that fails to receive the message due to connectivity issues.

`sendRoomStatus`

room with the specified name, while `sendJoinRoom(String room)` signals the server that the client wishes to join a specific room. These methods facilitate communication between the client and server, enabling room management functionality in a multi-client environment.

informs clients about join or leave events, removing clients who cannot be notified. `sendDisconnect` specifically notifies clients when another client disconnects, handling failures similarly by removing disconnected clients. These methods safeguard against concurrency issues, ensuring reliable communication between the server and clients in the room.

#### #5) Show the Server code for handling



```
CREATE_ROOM_REQUEST {
    // Handle the request to create a new room
    // Check if the room name is unique
    // If not, return an error
    // If yes, create the room
    // Return the room ID
}

// Handle the request to join a room
// Check if the room exists
// If not, return an error
// If yes, add the client to the room
// Return the room ID
```

**Caption (required)** ✓  
*Describe/highlight what's being shown (ucid/date must be present)*

Showing the Server code for handling the create/join process

**Explanation (required)** ✓  
*Briefly explain the code/logic/flow involved*

**PREVIEW RESPONSE**

This `createRoom` method first checks if a room with the given name already exists in the server. If it does, it does nothing and

#### #6) Show that Client messages



```
// Handle the request to join a room
// Check if the room exists
// If not, return an error
// If yes, add the client to the room
// Return the room ID

// Handle the request to send a message
// Check if the client is in the room
// If not, return an error
// If yes, send the message to the room
// Return the message ID
```

**Caption (required)** ✓  
*Describe/highlight what's being shown*  
Showing that Client messages are constrained to the Room

**Explanation (required)** ✓  
*Briefly explain why/how it works this way*

**PREVIEW RESPONSE**

In this server setup, each Room maintains a list of clients (`clientsInRoom`) who are currently connected to that specific room. When a client sends a message through the



does nothing and returns false. If the room doesn't exist, it creates a new Room object with that name, adds it to the server's list of rooms, prints a message saying the room was created, and returns true to confirm the room was successfully created. This helps keep track of unique rooms on the server.

through the sendMessage method of the Room class, it iterates over the list of clients in that room. This ensures that the message is only delivered to clients who belong to the same room as the sender. Therefore, clients in different rooms cannot communicate directly with each other through this system, maintaining the integrity of room-based communication as intended by the server design.

## Disconnecting/Termination (3 pts.)

^COLLAPSE ^

### Task #1 - Points: 1

Text: Disconnecting

#### Details:

Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade. The goal is to show you understand what segments are related to the prompts.

#1) Show Clients gracefully



**Caption (required) ✓**  
Describe/highlight what's being shown

#2) Show the code related to Clients



**Caption (required) ✓**  
Describe/highlight what's being shown

#3) Show the Server terminating



**Caption (required) ✓**  
Describe/highlight what's being shown

#4) Show the Server code related to



**Caption (required) ✓**  
Describe/highlight what's being shown

Showing Clients  
disconnecting

(ucid/date must be  
present)

Showing the code  
related to Clients  
disconnecting

**Explanation (required)**



*Briefly explain the  
code/logic/flow involved*

 **PREVIEW RESPONSE**

The processDisconnect function manages when a client disconnects from the server. It prints a message saying "You disconnected" if it's the current client (clientId == myData.getClientId()), or "[clientId] disconnected" for others, highlighting the message in red. If the current client disconnects, it also closes the server connection. This ensures all disconnections are handled gracefully with appropriate feedback.

Showing the Server  
terminating

(ucid/date must be  
present)

Showing the Server code  
related to handling  
termination

**Explanation (required)**



*Briefly explain the  
code/logic/flow involved*

 **PREVIEW RESPONSE**

The shutdown method manages the server shutdown process smoothly. It iterates through the rooms collection using remove to safely remove rooms if they are empty. For each room removed, it ensures all clients are disconnected by calling room.disconnectAll(). This approach avoids issues with simultaneous modifications to the room collection during shutdown. Any errors that occur during this cleanup process are caught and printed to the console. Overall, shutdown ensures the server shuts down gracefully, handling all necessary cleanup tasks effectively.



Misc (1 pt.)

 **COLLAPSE** 



### Task #1 - Points: 1

Text: Add the pull request link for this branch

#### URL #1

<https://github.com/st278/st278-IT114-M2024/pull/8>



### Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

#### Details:

Few related sentences about the Project/sockets topics

#### Response:

The only issue I really had was manually fining the codes but it helped me be familiar with it.



### Task #3 - Points: 1

Text: WakaTime Screenshot

#### Details:

Grab a snippet showing the approximate time involved that clearly shows your repository.

The duration isn't considered for grading, but there should be some time involved.

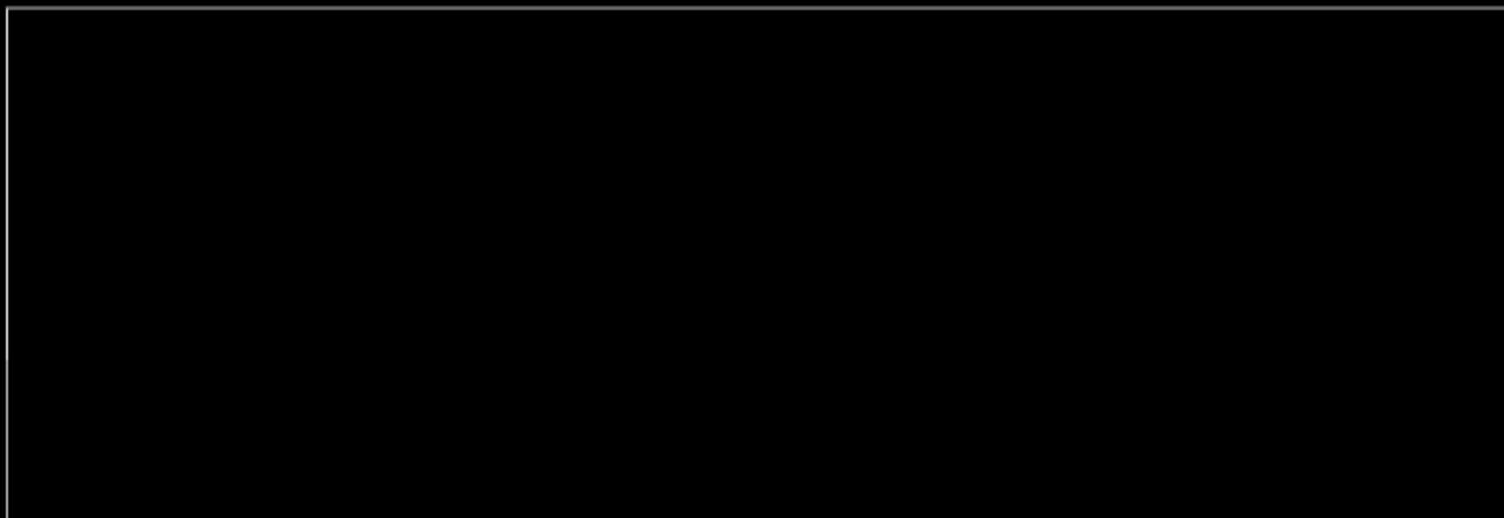
#### Task Screenshots:

Gallery Style: Large View

Small

Medium

Large



Overall timer of the last 7 days.

Individual timers on each files that were worked on.

End of Assignment