

# Submission Worksheet

CLICK TO GRADE

<https://learn.ethereallab.app/assignment/IT114-450-M2024/it114-milestone-3-chatroom-2024-m24/grade/yh68>

IT114-450-M2024 - [IT114] Milestone 3 Chatroom 2024 M24

## Submissions:

Submission Selection

1 Submission [active] 7/22/2024 5:52:34 AM

## Instructions

^ COLLAPSE ^

Implement the Milestone 3 features from the project's proposal document:

<https://docs.google.com/document/d/1ONmvEveI97GTFPGfVwwQC96xSsobbSbk56145XizQG4/view>

Make sure you add your ucid/date as code comments where code changes are done All code changes should reach the Milestone3 branch Create a pull request from Milestone3 to main and keep it open until you get the output PDF from this assignment. Gather the evidence of feature completion based on the below tasks. Once finished, get the output PDF and copy/move it to your repository folder on your local machine. Run the necessary git add, commit, and push steps to move it to GitHub Complete the pull request that was opened earlier Upload the same output PDF to Canvas

Branch name: Milestone3

Tasks: 8 Points: 10.00



Basic UI (2 pts.)

^ COLLAPSE ^



Task #1 - Points: 1

Text: UI Panels

^ COLLAPSE ^

## Details:

All code screenshots must include ucid/date.

#1) Show the  
ConnectionPane  
by running



**Caption (required)** ✓  
*Describe/highlight  
what's being shown*  
Show the  
ConnectionPanel by  
running the app

#2) Show the  
code related  
to the



**Caption (required)** ✓  
*Describe/highlight  
what's being shown*  
Showing the code  
related to the CP

**Explanation (required)**



*Briefly explain how it  
works and how it's used*

**PREVIEW RESPONSE**

The ConnectionPanel class is a JPanel that provides UI with a host address and port number to establish a connection. The ConnectionPanel uses BorderLayout and BoxLayout for its layout. It has input fields for host ("127.0.0.1") and port ("3000"), with error labels and a "Next" button. Clicking "Next" checks if the port is a valid number, then stores the host and port and moves to the next card using ICardControls and also provides methods to get the entered host and port.

#3) show the  
UserDetailsPane  
by running



**Caption (required)** ✓  
*Describe/highlight  
what's being shown*  
Showing the UDP by  
running the app

#4) Show the  
code related  
to the



**Caption (required)** ✓  
*Describe/highlight  
what's being shown*  
Showing the code  
related to UDP

**Explanation (required)**



*Briefly explain how it  
works and how it's used*

**PREVIEW RESPONSE**

The UserDetailsPanel is a UI for entering a username. It uses BorderLayout and BoxLayout for layout. It has a username input field with an error label, a "Previous" button to go back, and a "Connect" button to proceed. If the username is empty it shows an error, otherwise it logs the username and calls controls.connect(). The panel is named and added to ICardControls for navigation and provides a method to get the entered username.

#5) Show the  
ChatPanel



#6) Show the  
code related



(there should

to the



**Caption (required)** ✓

*Describe/highlight*

*what's being shown*

Showing the ChatPanel

with 3 users present



**Caption (required)** ✓

*Describe/highlight*

*what's being shown*

Showing the code

related to the ChatPanel

**Explanation (required)**



*Briefly explain how it works and how it's used (note the important parts of the ChatPanel)*

 PREVIEW RESPONSE

The ChatPanel is a chat UI where users send and receive messages. It uses BorderLayout and GridBagLayout to organize components. The left side has a scrollable chat area for messages, and the right side has a user list. There's a text input field and a "Send" button at the bottom. When a message is sent, the text input is checked. If it starts with @, it's treated as a private message and sent to the specified user; otherwise, it's sent as a normal message. Messages appear



Build-up (3 pts.)

^COLLAPSE ^



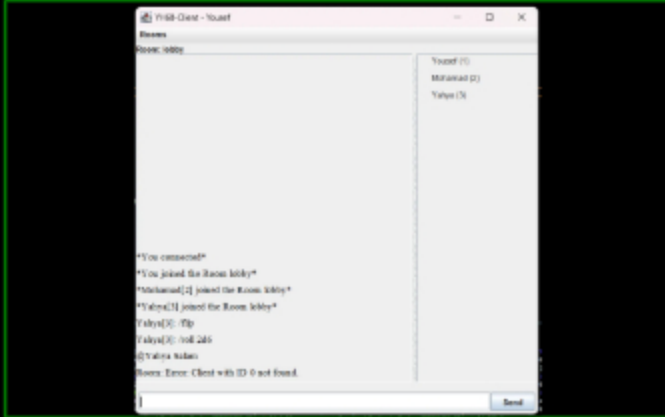
Task #1 - Points: 1

## Details:

All code screenshots must include ucid/date.

App screenshots must have the UCID in the title bar like the lesson gave.

### #1) Show examples of it printing on screen

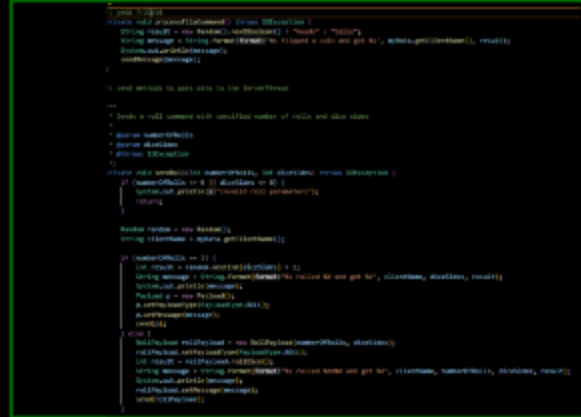


#### Caption (required) ✓

Describe/highlight what's being shown

They were working but I must have tweaked something because they dont even process how they did last week

### #2) Show the code on the Room side that changes this format



#### Caption (required) ✓

Describe/highlight what's being shown

Showing the code

#### Explanation (required) ✓

Explain what you did and how it works

PREVIEW RESPONSE

I have to go back and fix this. Didn't realize how long it took me trying to incorporate all my methods from Milestone2 into the UI.

## Task #2 - Points: 1

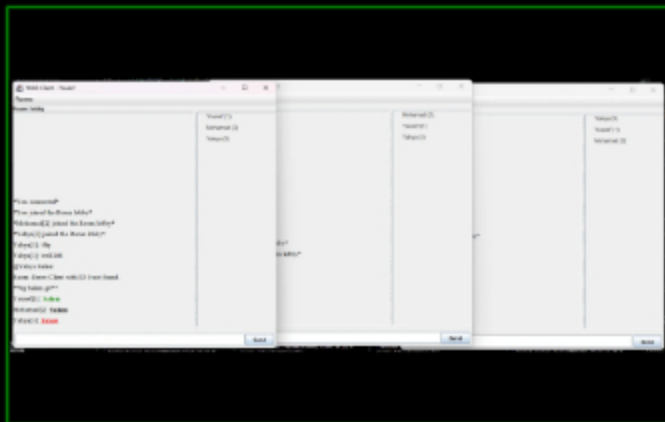
Text: Text Formatting appears correctly on the UI

## Details:

All code screenshots must include ucid/date.

App screenshots must have the UCID in the title bar like the lesson gave.

#1) Show examples of bold, italic, underline, each color implemented and a combination of bold, italic, underline, and



### Caption (required) ✓

Describe/highlight what's being shown

Showing examples of bold italic, underline, each color implemented and a combination of bold, italic, underline, and one color

#2) Show the code changes necessary to get this to work



```
// ucid/11/11/24
public void addText(String text) {
    SwingUtilities.invokeLater(() -> {
        JTextPane textContainer = new JTextPane(JEditorPane.getDefaultEditor());
        textContainer.setText("");
        textContainer.setCaretPosition(0);
        textContainer.setBackground(new Color(255, 255, 255));

        // Formatting constraints settings for each message
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.gridx = 0; // Column index 0
        gbc.gridy = GridBagConstraints.RELATIVE; // Automatically move to the next row
        gbc.weightx = 1; // Let the component grow horizontally to fill the space
        gbc.fill = GridBagConstraints.HORIZONTAL; // Fill horizontally
        gbc.insets = new Insets(10, 0, 0, 0); // Add spacing between messages

        chatArea.add(textContainer, gbc);
        chatArea.repaint();
        chatArea.scrollRectToVisible(chatArea.getBounds());

        // Scroll down on new message
        JScrollPane parentScrollPane = (JScrollPane) SwingUtilities.getWindowAncestor(chatArea);
        if (parentScrollPane != null) {
            SwingUtilities.invokeLater(() -> {
                JScrollPane vertical = parentScrollPane.getVerticalScrollbar();
                vertical.setValue(vertical.getMaximum());
            });
        }
    });
}
```

### Caption (required) ✓

Describe/highlight what's being shown

Showing the changes needed to implement this

### Explanation (required) ✓

Briefly explain what was necessary and how it works

#### PREVIEW RESPONSE

The addText method adds an HTML-formatted message to a chat area, updating the UI on the Event Dispatch Thread. It creates a non-editable JEditorPane for the message, positions it with layout constraints, and scrolls the view to the bottom to show the latest message.

## New Features (4 pts.)

^COLLAPSE ^



### Task #1 - Points: 1

Text: Private messages via @username

^COLLAPSE ^

### Details:

All code screenshots must include ucid/date.

App screenshots must have the UCID in the title bar like the lesson gave.

- **Note:** This will not be a slash command
- **Note:** The writer and the receiver are the only two that will receive the message from the server-side
- It's not valid to just hide it on the client-side (i.e., data must not be sent from the server-side)
- The Client-side will capture the message/target, find the appropriate client id, and send that along with the original message to the server-side
  - If a client id isn't found for the target, a message will be shown to the Client stating so and will not cause a payload to be sent to the server-side
- The ServerThread will receive this payload and pass the id and message to the Room
- The Room will match the id to the respective target and send the message to the sender and target (receiver)

### #1) Show a few examples



**Caption (required) ✓**  
*Describe/highlight what's being shown*  
Showing the result of @username

### #2) Show the client-side code that



**Caption (required) ✓**  
*Describe/highlight what's being shown*  
Showing client side code

#### Explanation (required)

✓  
*Explain in concise steps how this logically works*

[PREVIEW RESPONSE](#)

The code handles private messages that start with "@" by splitting the text into a username and message content. If the format is incorrect, it shows an error dialog. If valid, it attempts to send the private message using `sendPrivateMessage`, logging an error if the operation fails.

### #3) Show the ServerThread code



**Caption (required) ✓**  
*Describe/highlight what's being shown*  
Showing serverThread receiving it and passing it to Room

#### Explanation (required)

✓  
*Explain in concise steps how this logically works*

[PREVIEW RESPONSE](#)

The `handlePrivateMessage` method processes a private message payload by retrieving the target client from the current room using their ID. If the target client exists, it sends the message to them and confirms to the sender. If the target client is not found or if the sender is not in a room, it sends

### #4) Show the Room code that verifies



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

#### Explanation (required)

✓  
*Explain in concise steps how this logically works*

[PREVIEW RESPONSE](#)

The `handlePrivateMessage` method processes a private message payload by retrieving the target client from the current room using their ID. If the target client exists, it sends the message to them and confirms to the sender. If the target client is not found or if the sender is not in a room, it sends an error message to the sender.



an error message to the sender.

## Task #2 - Points: 1

Text: Mute and Unmute

### Details:

All code screenshots must include ucid/date.

App screenshots must have the UCID in the title bar like the lesson gave.

- Client-side will implement a /mute and /unmute command (i.e., /mute Bob or /unmute Bob)
  - Client side grabs the target and finds the client id related to the target
    - If no target found, an appropriate message will be displayed and no message will be sent
    - If a target is found, the id will be sent in a payload to the server-side with the appropriate action
- ServerThread will receive the payload and extract the data, then pass it to a Room method
- The Room will confirm the id against the list of clients
  - If found, it'll record the client's name on a list of the sender's ServerThread for a mute, otherwise it'll remove the name from the list
    - **Note:** This list must be unique and must not be directly exposed, the ServerThread must provide method accessors like add()/remove()
  - Upon success mute/unmute, the sender should receive a confirmation of the action clearly stating what happened
- Any time a message would be received (i.e., normal messages or private messages) the sender's name will be compared against the receiver's mute list
  - **Note:** The mute list won't be exposed directly, there should be a method on the ServerThread that takes the name and returns a boolean about whether or not the person is muted
  - If the user is muted, the receive must not be sent the message (i.e., they get skipped)
    - You must log in the terminal that the message was skipped due to being muted, but no message should be sent in this regard

### #1) Show a few examples



**Caption (required) ✓**  
Describe/highlight what's being shown  
Showing examples across different clients

### #2) Show the client-side code that



**Caption (required) ✓**  
Describe/highlight what's being shown  
Showing the client side code

### #3) Show the ServerThread code



**Caption (required) ✓**  
Describe/highlight what's being shown  
Showing the Serverthread code

### #4) Show the Room code that verifies



**Caption (required) ✓**  
Describe/highlight what's being shown  
Showing the room code

**Explanation (required)**

**Explanation (required)**



***Explain in concise steps  
how this logically works***

 [PREVIEW RESPONSE](#)

The `sendMute` and `sendUnmute` methods create a `Payload` object for muting or unmuting a user by setting the payload type and username. They then send this payload using the `send` method, which handles the communication process.

**Explanation (required)**



***Explain in concise steps  
how this logically works***

 [PREVIEW RESPONSE](#)

The `handleMute` and `handleUnmute` methods manage client muting and unmuting within the current room. They retrieve the target client using their ID and either mute or unmute them by calling `addMutedClient` or `removeMutedClient`. The sender receives a confirmation message if the operation is successful, or an error message if the target client is not found or if no room is available.

✓

*Explain in concise steps  
how this logically works*

 PREVIEW RESPONSE

The `sendMessage` method sends a formatted message to all clients in the room, except for those muted by the sender. If the room is not active, the method does nothing. It first checks if the room is running, then formats the message, and iterates through clients, skipping muted ones and removing any clients that fail to receive the message.

#5) Show the Room code that checks



**Caption (required)** ✓

*Describe/highlight  
what's being shown*  
Showing room code

**Explanation (required)**

✓

***Explain in concise steps  
how this logically works***

#6) Show terminal supplement



**Caption (required)** ✓

*Describe/highlight  
what's being shown*  
Showing the log




Had too many errors and not enough time to correct them but the `sendMessage` method sends a formatted message to all clients in the room who have not muted the sender, and handles disconnects if message delivery fails. The `sendPrivateMessage` method sends a formatted message only to the specified recipient and the sender, including a note indicating that the message was sent to the recipient. Both methods use a `markdownConverter` to format the message before sending.

Misc (1 pt.)

^COLLAPSE ^

Task #1 - Points: 1

Text: Add the pull request link for the branch

 Details:Note: the link should end with `/pull/#`

URL #1

<https://github.com/FreePalestine7/yh68-it114-45076>

URL

<https://github.com/FreePalestine7/yh68-it114-45076>

+ ADD ANOTHER URL

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

I had so many issues especially when trying to implement the features. Some stuff would just not work and I was looking through the lessons to see what I could find to help but there wasn't any lessons specific for my issues. I did learn a lot though, UI might just be one of my favorite things to play with, from panels to integration between server and client it was cool and fun.

COLLAPSE

### 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:

