Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-451-M2024/it114-milestone-3-chatroom-2024-m24/grade/st278

IT114-451-M2024 - [IT114] Milestone 3 Chatroom 2024 M24

Submissions:

Submission Selection

1 Submission [active] 7/24/2024 3:28:32 AM

Instructions

COLLAPSE

Implement the Milestone 3 features from the project's proposal document: https://docs.google.com/document/d/1ONmvEvel97GTFPGfVwwQC96xSsobbSbk56145XizQG4/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

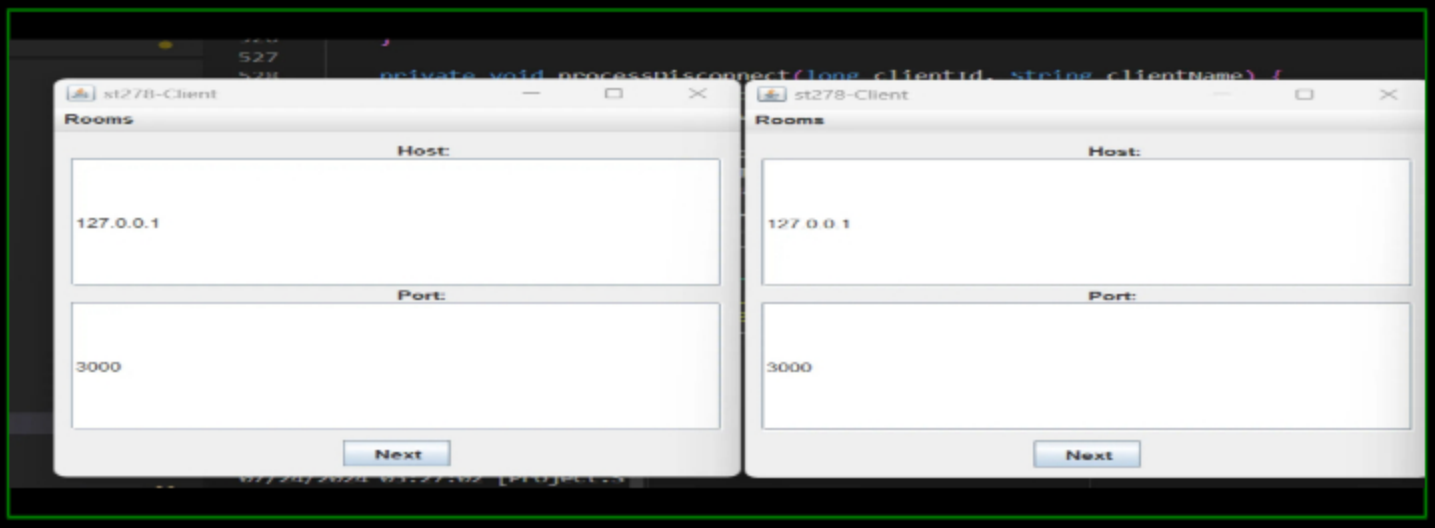
COLLAPSE

Details:

All code screenshots must include ucid/date.

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

#1) Show the ConnectionPanel by running the app (should have host/port)

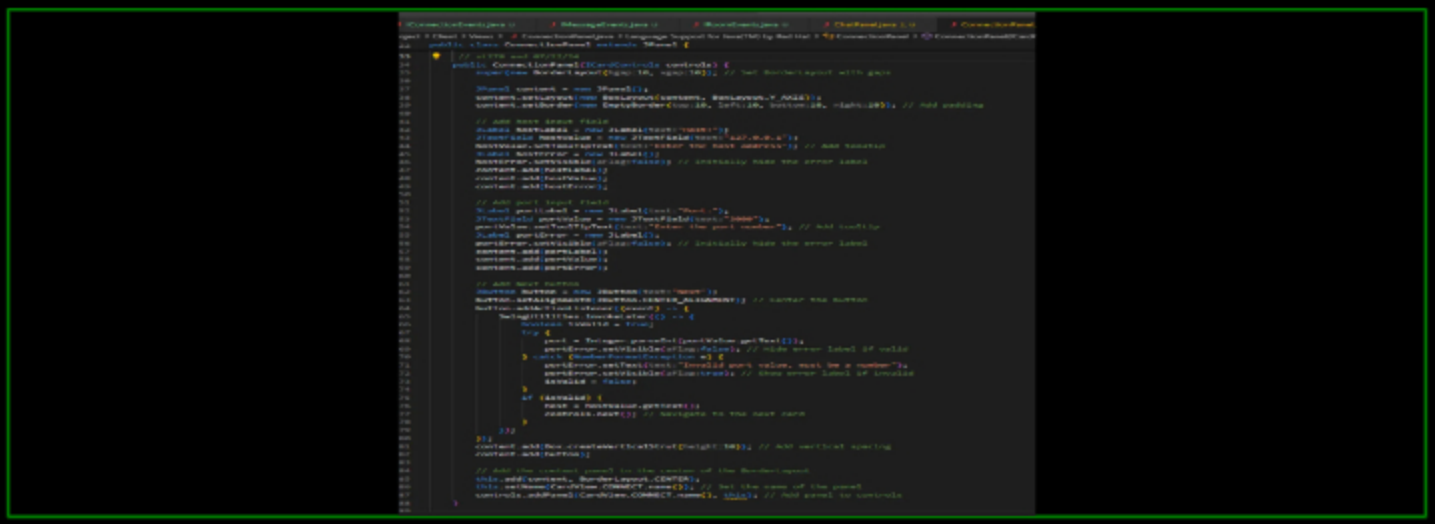


Caption (required) ✓

Describe/highlight what's being shown

Showing the ConnectionPanel by running the app

#2) Show the code related to the ConnectionPanel



Caption (required) ✓

Describe/highlight what's being shown

Showing the code related to the ConnectionPanel

Explanation (required) ✓

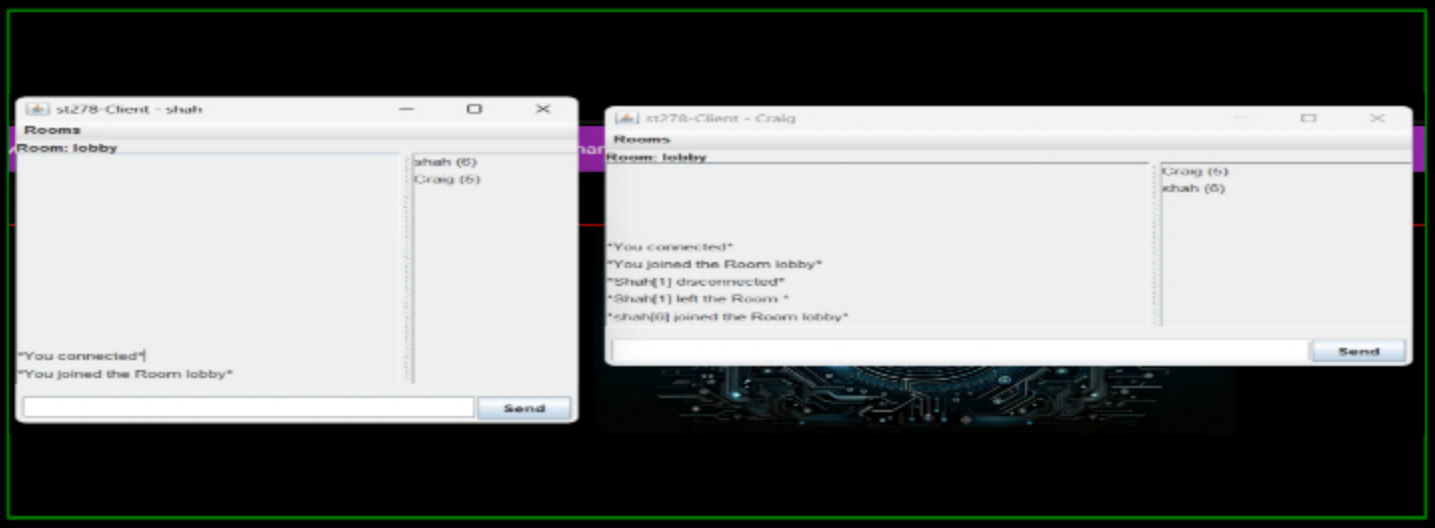
Briefly explain how it works and how it's used

PREVIEW RESPONSE

The ConnectionPanel constructor sets up a user interface with a BorderLayout and a central JPanel using

The ConnectionPanel constructor sets up a user interface with a BorderLayout and a central JPanel using BoxLayout to arrange components vertically. It includes input fields for host and port values, each with labels, text fields, and hidden error labels for validation messages. A "Next" button is added with an ActionListener that validates the port number when clicked. If the port is valid, the panel hides any error messages and transitions to the next panel; otherwise, it displays an error message. The panel is named and registered with the controls object for navigation.

#3) show the UserDetailsPanel by running the app (should have username)



Caption (required) ✓

Describe/highlight what's being shown

showing the UserDetailsPanel by running the app

#4) Show the code related to the UserDetailsPanel

Caption (required) ✓

Describe/highlight what's being shown

Showing the code related to the UserDetailsPanel

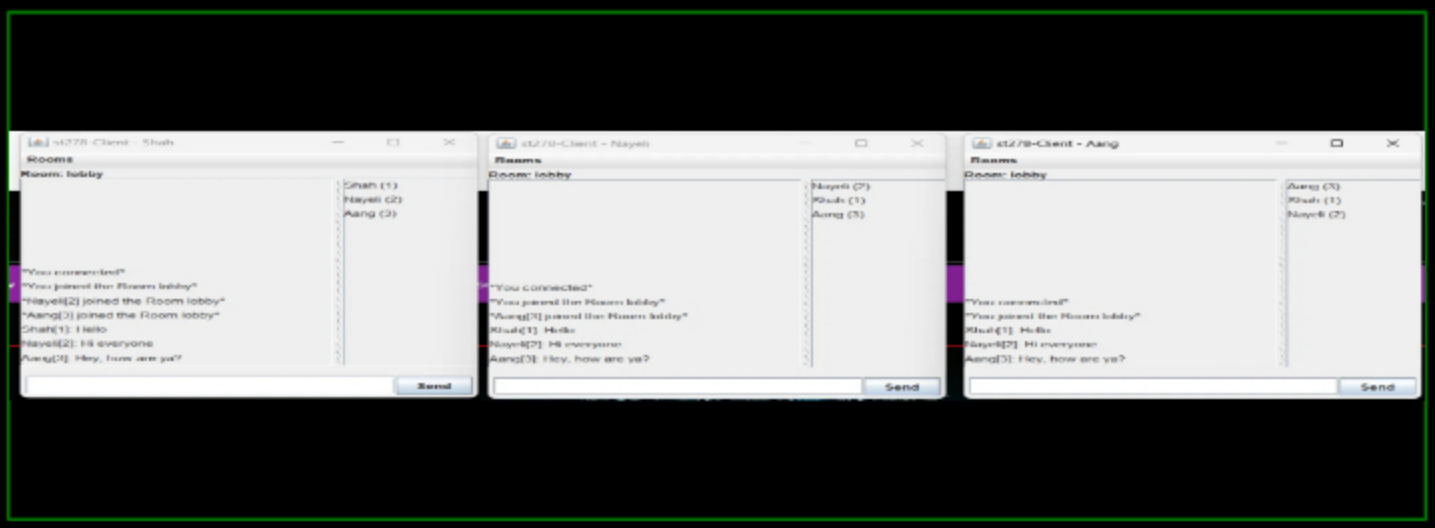
Explanation (required) ✓

Briefly explain how it works and how it's used

PREVIEW RESPONSE

The UserDetailsPanel constructor sets up a UI with a vertical layout that includes a username input field, labels for the username and error messages, and "Previous" and "Connect" buttons. The "Previous" button navigates back, while the "Connect" button checks if the username is provided; if it is, the username is logged and the connection process starts, otherwise an error message is shown. The buttons are positioned at the bottom, and the panel is padded and named for navigation.

#5) Show the ChatPanel (there should be at least 3 users present and some example messages)

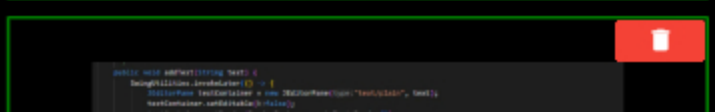


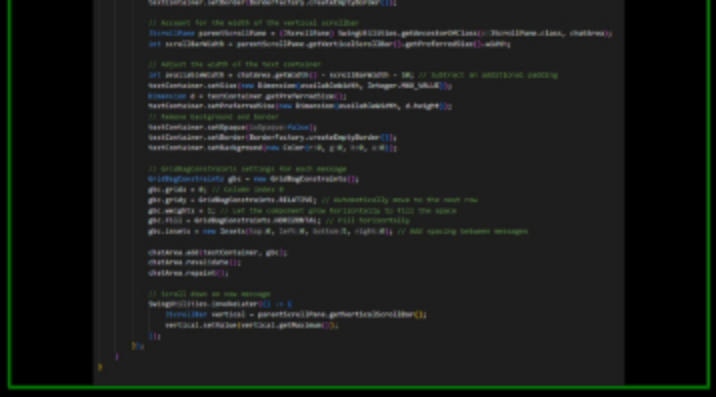
Caption (required) ✓

Describe/highlight what's being shown

Showing the ChatPanel

#6) Show the code related to the ChatPanel





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

This class defines the ChatPanel which handles the chat interface, including the chat area for messages, user list, and input field for sending messages. It uses JSplitPane to separate the chat and user list panels, handles input through a text field and a send button, and manages updates to the chat and user list dynamically.

Build-up (3 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

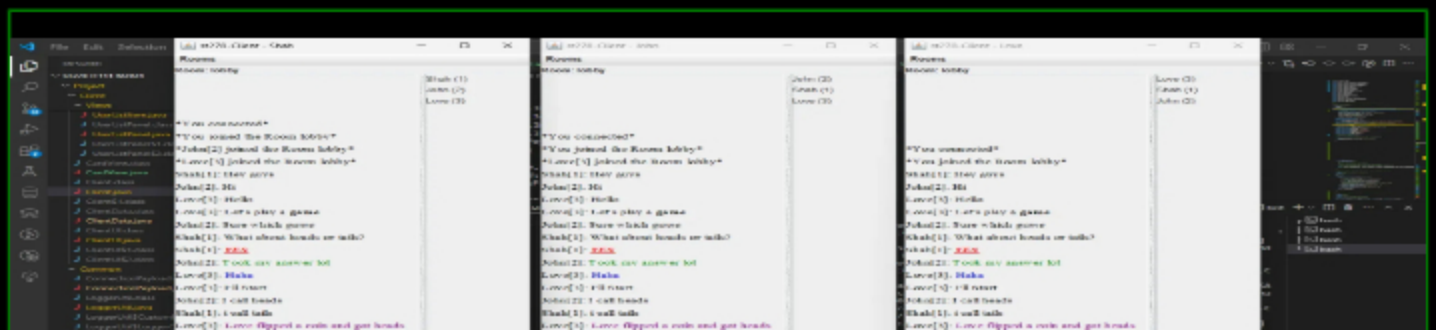
Text: Results of /flip and /roll appear in a different format than regular chat text

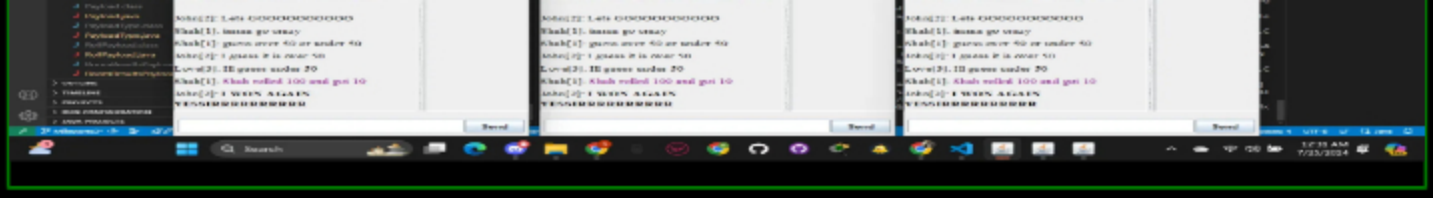
Details:

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

Showing examples of /flip and /roll appear in a different format than regular chat text

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

```
protected void handleRoll(ServerThread sender, RollPayload payload) {
    int result;
    String formattedMessage;

    if (payload.isSimpleRoll()) {
        result = (int) (Math.random() * payload.getSides()) + 1;
        formattedMessage = String.format(format:"%s rolled %d and got %d", sender.getClientName(), payload.getSides(), result);
    } else {
        int total = 0;
        for (int i = 0; i < payload.getQuantity(); i++) {
            total += (int) (Math.random() * payload.getSides()) + 1;
        }
        formattedMessage = String.format(format:"%s rolled %dd%d and got %d", sender.getClientName(), payload.getQuantity(), payload.getSides(), total);
    }
    // st270 and 07/24/24
    String message = String.format(format:"ROLL: %s", formattedMessage);
    sendMessage(sender, message);
}

protected void handleFlip(ServerThread sender) {
    boolean isHeads = Math.random() < 0.5;
    String result = isHeads ? "heads" : "tails";
    String message = String.format(format:"FLIP: %s flipped a coin and got %s", sender.getClientName(), result);
    sendMessage(sender, message);
}
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the code on the Room side that changes this format

Explanation (required) ✓

Explain what you did and how it works

PREVIEW RESPONSE

The program processes the roll command by determining whether it is a simple roll or not. It then formats the result message to include the roller's name, dice details, and the result. The message is colored purple using the TextFX.colorize() method before being sent through the sendMessage() function. Also, The program randomly determines the coin flip result and formats a message that includes the flipper's name and the result. The message is then colored purple using the TextFX.colorize() method before being sent through the sendMessage() function.

COLLAPSE

Task #2 - Points: 1

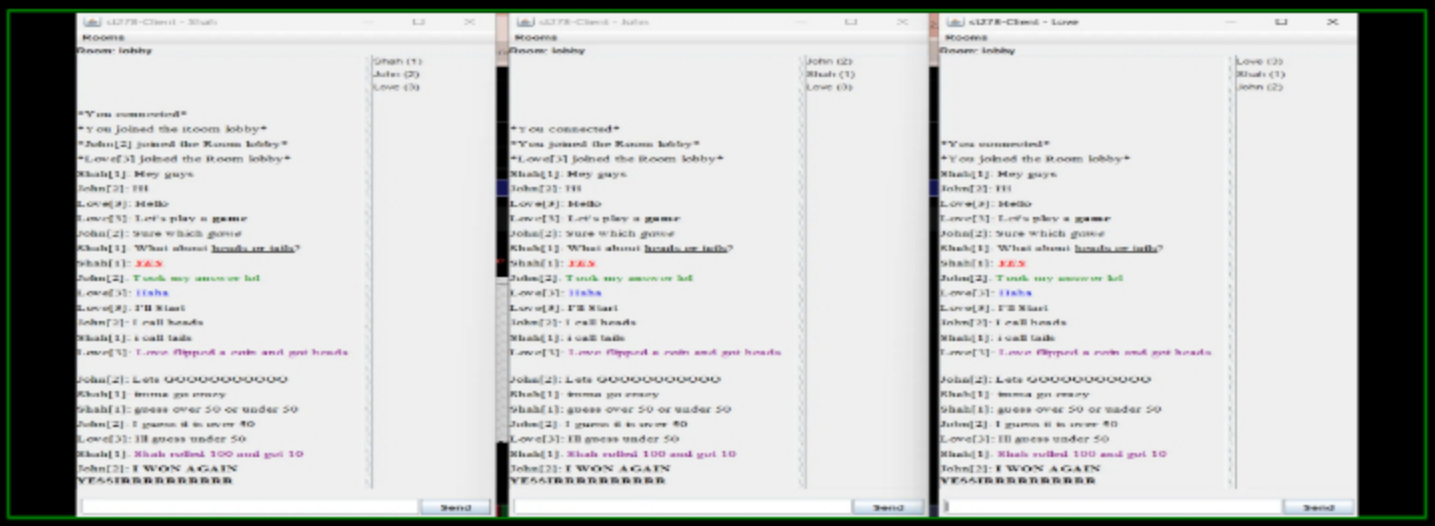
Text: Text Formatting appears correctly on the UI

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 one color in the same message

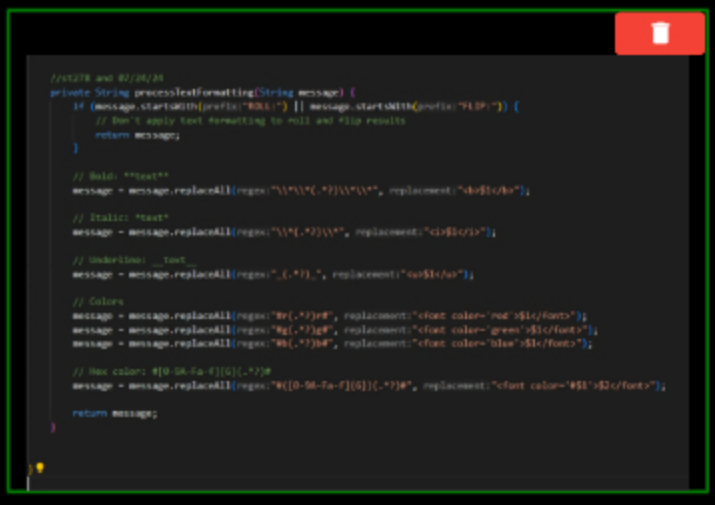


Caption (required) ✓

Describe/highlight what's being shown

Showing examples of all the html related things

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



```
//st278 and 07/24/24
private String processTextFormatting(String message) {
    if (message.startsWith(prefix:"ROLL:") || message.startsWith(prefix:"FLIP:")) {
        // Don't apply text formatting to roll and flip results
        return message;
    }

    // Bold: **text**
    message = message.replaceAll(regex:"\\*\\*(.*?)\\*\\*", replacement:"<b>$1</b>");

    // Italic: *text*
    message = message.replaceAll(regex:"\\*(.*?)\\*", replacement:"<i>$1</i>");

    // Underline: __text__
    message = message.replaceAll(regex:"__(.*?)__", replacement:"<u>$1</u>");

    // Colors
    message = message.replaceAll(regex:"@r(.*?)@", replacement:"<font color='red'>$1</font>");
    message = message.replaceAll(regex:"@g(.*?)@", replacement:"<font color='green'>$1</font>");
    message = message.replaceAll(regex:"@b(.*?)@", replacement:"<font color='blue'>$1</font>");

    // Hex color: @[0-9A-Fa-f]{6}(.*?)@
    message = message.replaceAll(regex:"@([0-9A-Fa-f]{6})(.*?)@", replacement:"<font color='#$1'>$2</font>");

    return message;
}
```

```
//st278 and 07/24/24
public void addText(String text) {
    SwingUtilities.invokeLater(() -> {
        JEditorPane textContainer = new JEditorPane(type:"text/html", "<html>" + text + "</html>");
        textContainer.setEditable(b:false);
        textContainer.setBorder(BorderFactory.createEmptyBorder());
        textContainer.setOpaque(isOpaque:false);
        textContainer.setBackground(new Color(r:0, g:0, b:0, a:0));
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the code changes necessary to get this to work

Explanation (required) ✓

Briefly explain what was necessary and how it works

PREVIEW RESPONSE

I updated the processTextFormatting method in the Room class to replace custom markup (like **bold**, *italic*, *underline*, #rred#r, etc.) with corresponding HTML tags. Also, I modified the addText method in the ChatPanel class to use HTML for rendering messages by setting the JEditorPane to "text/html". This allows formatted text to be displayed properly in the chat UI, enabling combinations of different text styles and colors.

New Features (4 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Private messages via @username

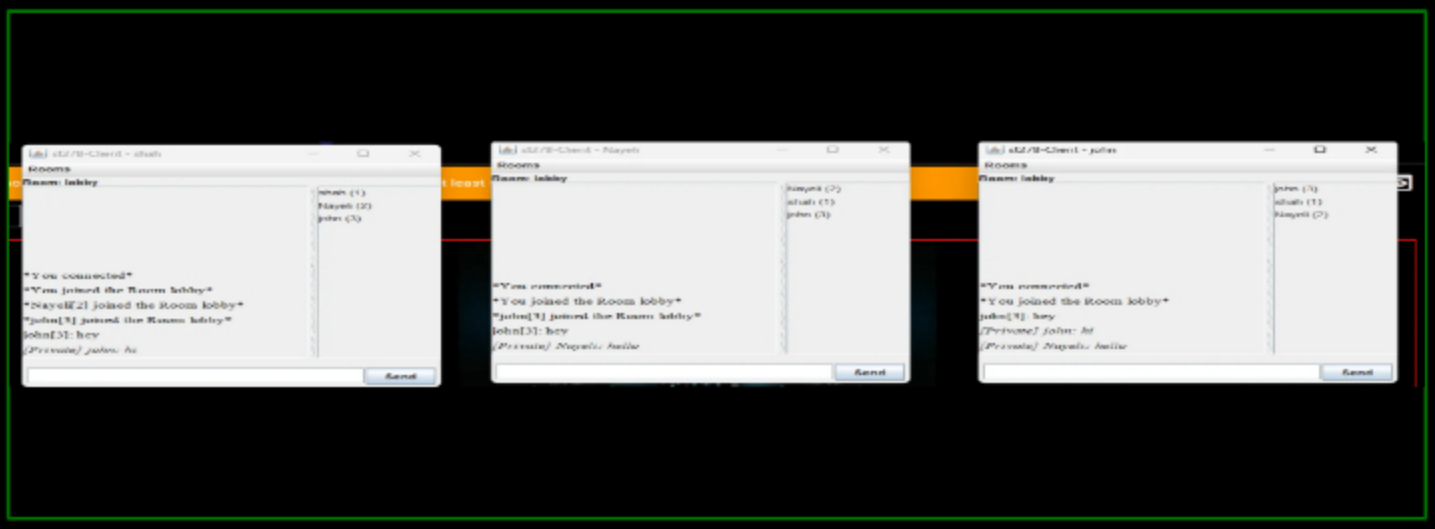
Details:

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 across different clients (there should be at least 3 clients in the Room)



Caption (required) ✓

Describe/highlight what's being shown

Showing a few examples across different clients

#2) Show the client-side code that processes the text per the requirement

```
//st278 and 07/24/24        You, 1 second ago • Uncommitted changes
public void sendMessage(String message) throws IOException {
    if (processClientCommand(message)) {
        return;
    }
    if (message.startsWith(prefix:"@")) {
        String[] parts = message.split(regex:" ", limit:2);
        if (parts.length == 2) {
            String targetUsername = parts[0].substring(beginIndex:1);
            long targetId = findClientIdByUsername(targetUsername);
            if (targetId != -1) {
                sendPrivateMessage(targetId, parts[1]);
            } else {
                System.out.println("User " + targetUsername + " not found.");
            }
        }
    } else {
        Payload p = new Payload();
        p.setPayloadType(PayloadType.MESSAGE);
        p.setMessage(message);
        send(p);
    }
}

private long findClientIdByUsername(String username) {
    for (ClientData client : knownClients.values()) {
        if (client.getClientName().equalsIgnoreCase(username)) {
            return client.getClientId();
        }
    }
    return -1;
}

private void sendPrivateMessage(long targetId, String message) throws IOException {
    Payload p = new Payload();
    p.setPayloadType(PayloadType.PRIVATE_MESSAGE);
    p.setClientId(targetId);
    p.setMessage(message);
    send(p);
}
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the client-side code that processes the text per the requirement

Explanation (required) ✓

Explain in concise steps how this logically works

PREVIEW RESPONSE

The sendMessage method first checks if the input is a client command with processClientCommand(message) and returns if it is. If the message starts with "@", it splits the message to get the target username and actual message. It then finds the target user's client ID using findClientIdByUsername. If the user is found, it sends a private message by creating a Payload with type PRIVATE_MESSAGE, setting the client ID and message, and sending it with send. If the message doesn't start with "@", it creates a public message Payload with type MESSAGE, sets the message, and sends it. The findClientIdByUsername method finds and returns the client ID of the matching username or -1 if not found. The sendPrivateMessage method sends a private message to the specified client ID.

#3) Show the ServerThread code receiving the payload and passing it to Room

```
//st278 and 07/24/24        You, 1 second ago • Uncommitted changes
case PRIVATE_MESSAGE:
    currentRoom.sendPrivateMessage(this, payload.getClientId(), payload.getMessage());
    break;
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the ServerThread code receiving the payload and passing it to Room

Explanation (required) ✓

Explain in concise steps how this logically works

PREVIEW RESPONSE

The processPayload method, when encountering a PRIVATE_MESSAGE type, calls the sendPrivateMessage method on the currentRoom object, passing the current object (this), the target client ID (payload.getClientId()), and the message content (payload.getMessage()), ensuring private messages are correctly routed to the specified client.

#4) Show the Room code that verifies the id and sends the message to both the sender and receiver

```
// st278 and 07/24/24
    public void sendPrivateMessage(ServerThread sender, long targetId, String message) {
        ServerThread target = clientsInRoom.get(targetId);
        if (target != null) {
            String formattedMessage = String.format(format:"[Private] %s: %s", sender.getClientName(), message);
            sender.sendMessage(formattedMessage);
            target.sendMessage(formattedMessage);
        } else {
            sender.sendMessage(message:"User not found in this room.");
        }
    }
}
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the Room code that verifies the ID and sends the message to both the sender and receiver

Explanation (required) ✓

Explain in concise steps how this logically works

PREVIEW RESPONSE

The sendPrivateMessage method first finds the target client using their ID. If the client is present, it formats the message with "[Private]", the sender's name, and the message content, and sends it to both the sender and the target client. If the target client is not found, it sends an error message to the sender.

Task #2 - Points: 1

^COLLAPSE ^

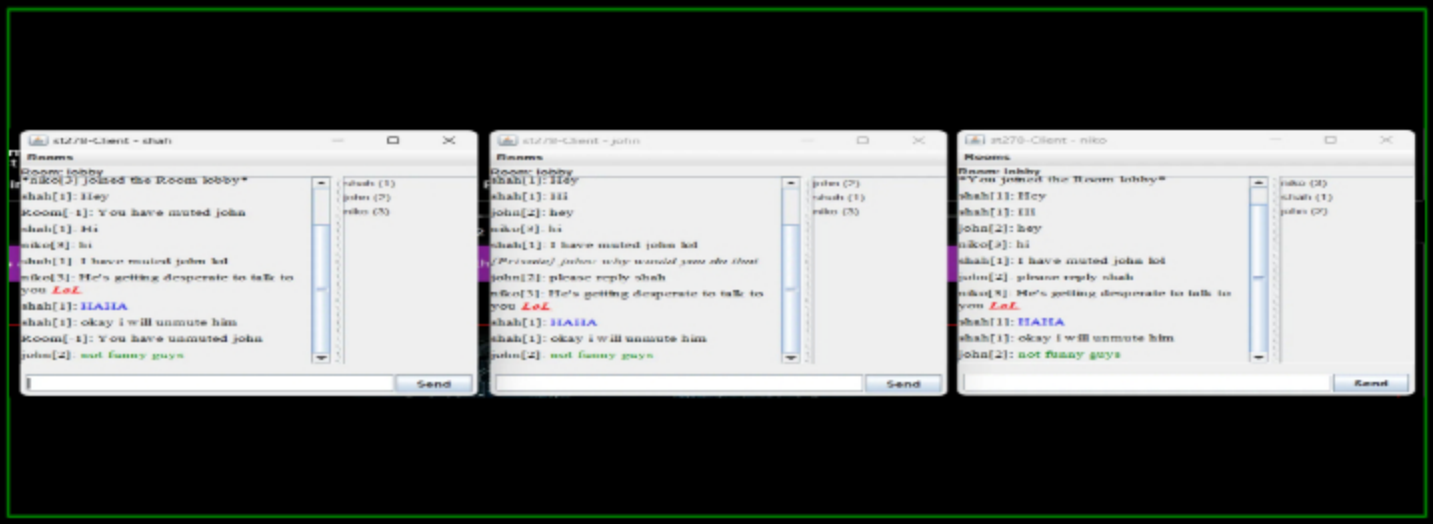
Text: Mute and Unmute

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 across different clients (there should be at least 3 clients in the Room)



Caption (required) ✓

Describe/highlight what's being shown

Showing a few examples across different clients

#2) Show the client-side code that processes the text per the requirement

```
//st278 and 07/24/24
if (text.startsWith(prefix:"/mute") || text.startsWith(prefix:"/unmute")) {
    String[] parts = text.split(regex:" ", limit:2);
    if (parts.length == 2) {
        String target = parts[1];
        long targetId = findClientIdByUsername(target);
        if (targetId != -1) {
            Payload p = new Payload();
            p.setPayloadType(text.startsWith(prefix:"/mute") ? PayloadType.MUTE : PayloadType.UNMUTE);
            p.setClientId(targetId);
            send(p);
        } else {
            System.out.println("User " + target + " not found.");
        }
    } else {
        System.out.println(x:"Usage: /mute <username> or /unmute <username>");
    }
    return true;
}
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the client-side code that processes the text per the requirement

Explanation (required) ✓

Explain in concise steps how this logically works

PREVIEW RESPONSE

The snippet of the code checks if the text starts with "/mute" or "/unmute" and splits it to get the target username. If there are two parts, it finds the client's ID using the username. If the client ID is valid, it creates a Payload for either muting or unmuting the user, sets the target ID, and sends the payload. If the user is not found or the command is incorrectly formatted, it prints an error message.

#3) Show the ServerThread code receiving the payload and passing it to Room

```
//st278 and 07/24/24
case MUTE:
    currentRoom.handleMute(this, payload.getClientId());
    break;
case UNMUTE:
    currentRoom.handleUnmute(this, payload.getClientId());
    break;
default:
```

```
//st278 and 07/24/24
public boolean addMutedUser(String username) {
    return mutedUsers.add(username);
}

public boolean removeMutedUser(String username) {
    return mutedUsers.remove(username);
}

public boolean isUserMuted(String username) {
    return mutedUsers.contains(username);
}
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the ServerThread code receiving the payload and passing it to Room

Explanation (required) ✓

Explain in concise steps how this logically works

PREVIEW RESPONSE

The code updates the processPayload method to handle MUTE and UNMUTE commands. When such a payload is received, the ServerThread calls the appropriate method on the currentRoom, passing itself and the target client ID. Each ServerThread has a private Set String to track muted usernames, with public methods to add, remove, and check muted users. This setup makes sure that mute/unmute commands are processed correctly, managed privately per client, and logged using LoggerUtil.

#4) Show the Room code that verifies the id and add/removes the muted name to/from the ServerThread's list

```
//st278 and 07/24/24
protected void handleMute(ServerThread sender, long targetId) {
    ServerThread target = clientsInRoom.get(targetId);
    if (target != null) {
        if (sender.addMutedUser(target.getClientName())) {
            sender.sendMessage("You have muted " + target.getClientName());
        } else {
            sender.sendMessage(target.getClientName() + " is already muted");
        }
    } else {
        sender.sendMessage(message:"User not found in this room.");
    }
}

protected void handleUnmute(ServerThread sender, long targetId) {
    ServerThread target = clientsInRoom.get(targetId);
    if (target != null) {
        if (sender.removeMutedUser(target.getClientName())) {
            sender.sendMessage("You have unmuted " + target.getClientName());
        } else {
            sender.sendMessage(target.getClientName() + " was not muted");
        }
    } else {
        sender.sendMessage(message:"User not found in this room.");
    }
}
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the Room code that verifies the ID and adds/removes the muted name to/from the ServerThread's list

Explanation (required) ✓

Explain in concise steps how this logically works

PREVIEW RESPONSE

The handleMute method checks if the target user exists in the room using the provided ID. If the user is found, it tries to add their name to the sender's muted list and confirms success or notifies the sender if the user was already muted. If the user isn't found, it sends an error message. The handleUnmute method works the same way in removing the user from the muted list if they are found and notifying the sender if the unmute was successful or if the user wasn't muted. Both methods send system messages using ServerThread.DEFAULT_CLIENT_ID and log actions with LoggerUtil, keeping the muted users list private within the ServerThread class.

#5) Show the Room code that checks the mute list during send message. private message, and any other relevant location

Caption (required) ✓

Describe/highlight what's being shown

Showing the Room code that checks the mute list while sending a message

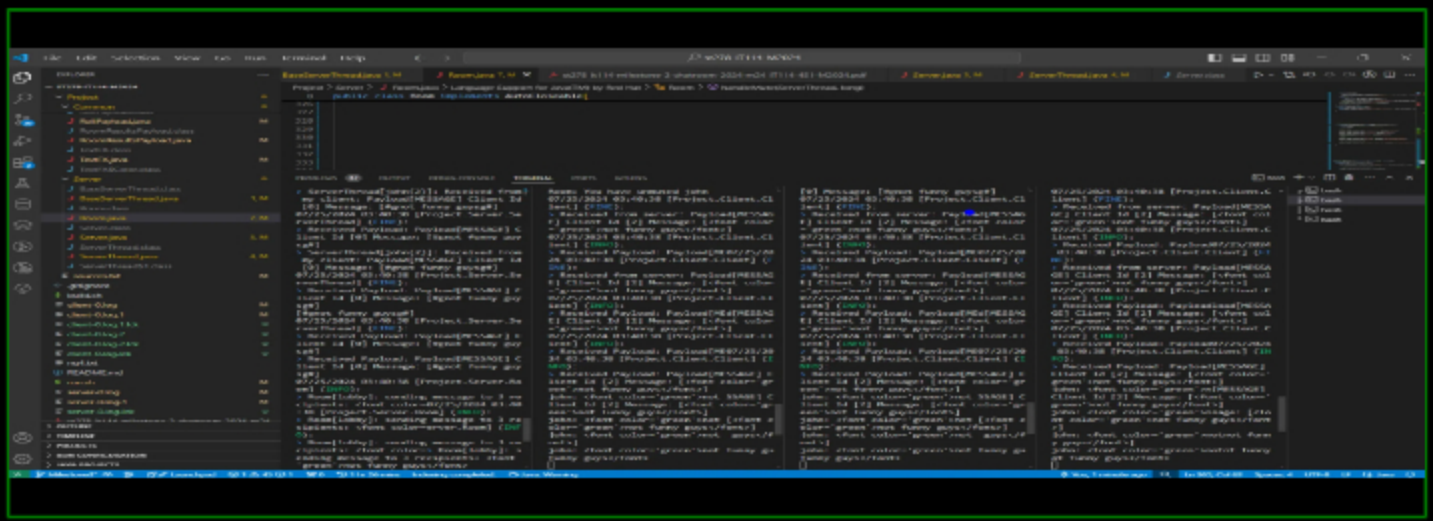
Explanation (required) ✓

Explain in concise steps how this logically works

PREVIEW RESPONSE

In sendMessage, the system checks if each recipient has muted the sender before delivering the message and logs any messages that are skipped due to muting. In sendPrivateMessage, the message is always sent to the sender, but it checks if the receiver has muted the sender before delivering the message and logs any private messages that are skipped for this reason.

#6) Show terminal supplemental evidence per the requirements (refer to the details of this task)



Caption (required) ✓

Describe/highlight what's being shown

Showing terminal supplemental evidence per the requirements

Misc (1 pt.)

COLLAPSE

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/st278/st278-IT114-M2024/pull/10

URL
https://github.com/st278/st278-IT114-M2024/pl

+ ADD ANOTHER URL

COLLAPSE

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

The main issue I'm always having is the code I wrte based on what needs to be implemented never works until I switch up everything I write as well as get help from peers and professor. I learned that greed is very bad and I should set aside time for classwork rather than working my job all week.

COLLAPSE

Task #3 - Points: 1

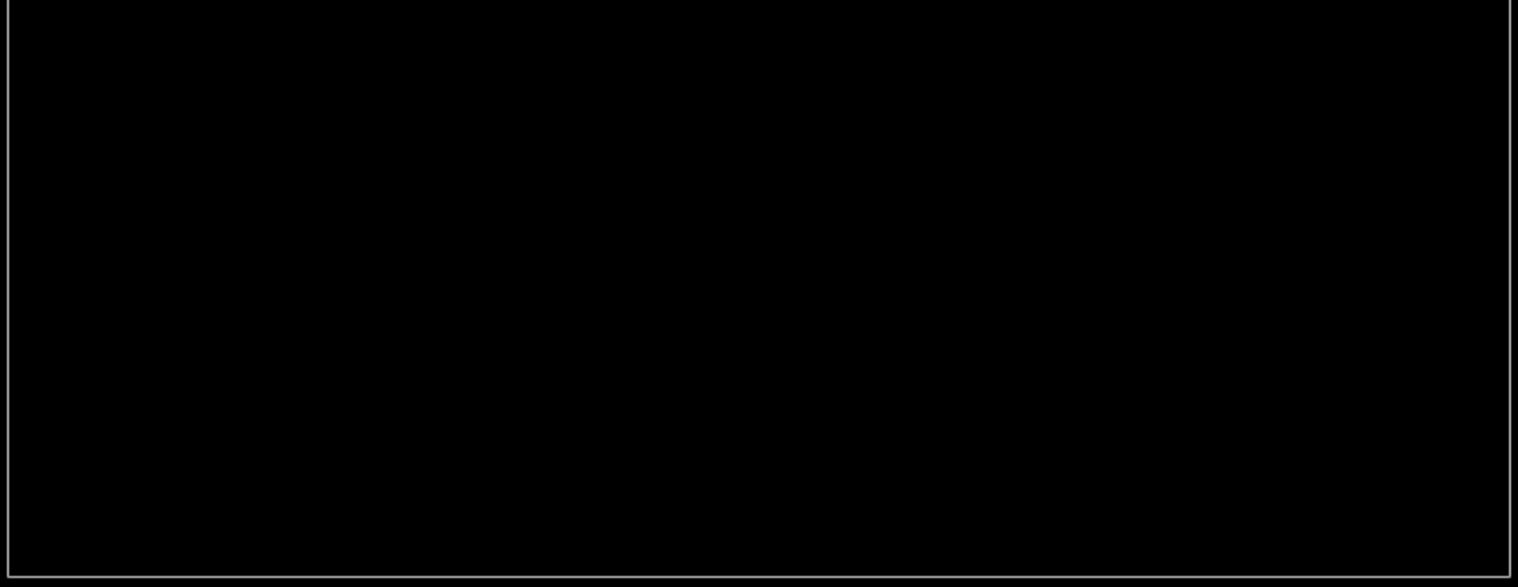
Text: WakaTime Screenshot

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:

Gallery Style: Large View

Small Medium Large



The overall time

Files

2 hrs 39 mins	Project/client/Client.java
1 hr 50 mins	…ct/server/ServerThread.java
1 hr 46 mins	…/views/ConnectionPanel.java
1 hr 41 mins	Project/server/Room.java
1 hr 6 mins	…client/views/ChatPanel.java
1 hr	Project/client/ClientUI.java
22 mins	…ect/common/RollPayload.java
15 mins	…views/UserDetailsPanel.java
15 mins	Project/client/ClientData.java
14 mins	run.sh
12 mins	Project/common/TextFX.java
10 mins	…erver/BaseServerThread.java
10 mins	…lient/views/RoomsPanel.java
7 mins	Project/client/CardView.java
5 mins	…terfaces/ICardControls.java
5 mins	…nt/views/UserListPanel.java
3 mins	…ect/common/PayloadType.java
2 mins	…mmon/ConnectionPayload.java
2 mins	Project/common/LoggerUtil.java
2 mins	Project/server/Server.java
2 mins	README.md
2 mins	Project/common/Payload.java
1 min	Project/client/views/Menu.java
1 min	…mon/RoomResultsPayload.java

Branches

11 hrs 12 mins	Milestone3
1 hr 13 mins	Milestone2

Specific times for each files.

End of Assignment