Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-450-M2024/it114-milestone-3-chatroom-2024-m24/grade/jah89

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

Submissions:

Submission Selection

1 Submission [active] 7/20/2024 2:03:42 PM

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

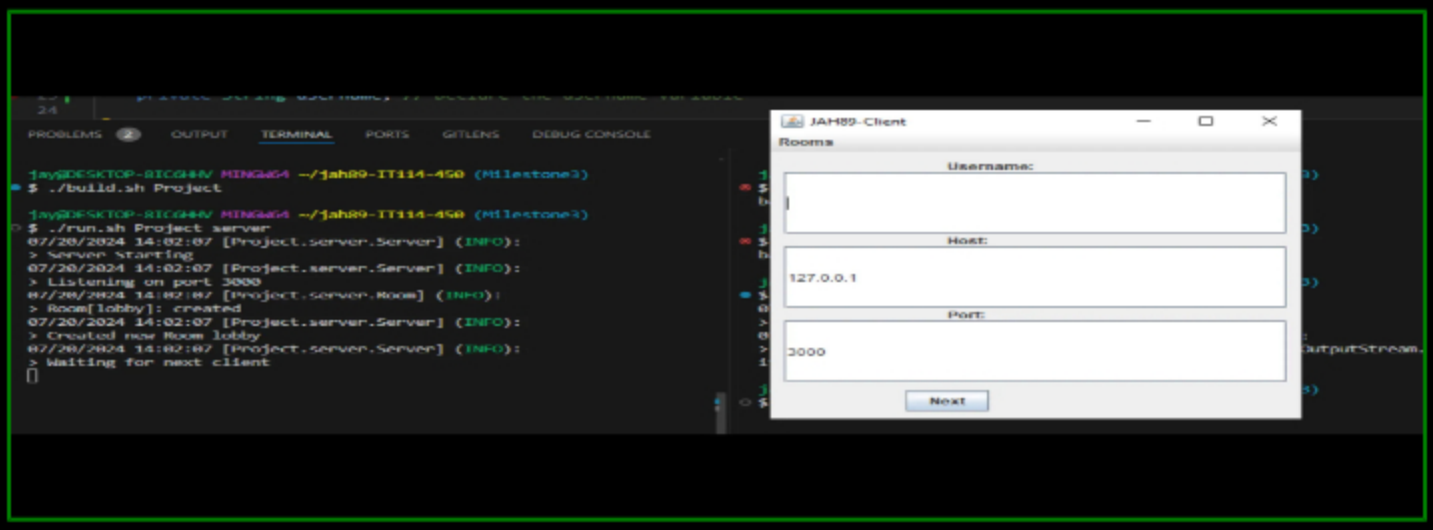
COLLAPSE

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

#2) Show the code related to the ConnectionPanel

```
// Validate username - jah89 07/19/2024
username = userValue.getText().trim();
if (username.isEmpty() || username.contains(s:" ")) {
    userError.setText(text:"Invalid username, must not be empty or contain spaces");
    userError.setVisible(aFlag:true);
    isValid = false;
} else {
    userError.setVisible(aFlag:false);
}

if (isValid) {
    host = hostValue.getText();
    controls.next();
}
});
```

```
// Add username - jah89 07/19/2024
JLabel userLabel = new JLabel(text:"Username:");
JTextField userValue = new JTextField();
userValue.setToolTipText(text:"Enter your username (no spaces)");
JLabel userError = new JLabel();
userError.setVisible(aFlag:false);
content.add(userLabel);
content.add(userValue);
content.add(userError);
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the code related to the ConnectionPanel

Explanation (required) ✓

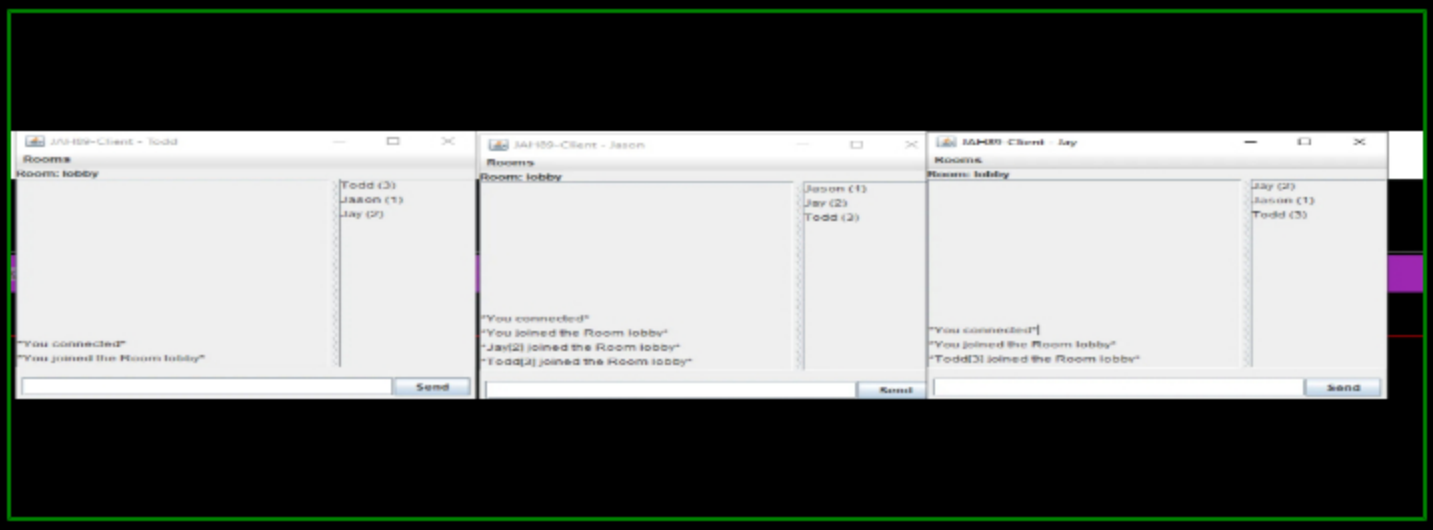
Briefly explain how it works and how it's used

PREVIEW RESPONSE

In the first code screenshot I added code for the UI components for the username input field. I used userLabel to

prompt the user to enter their username. I then used userValue as a text field where they will input their username. And finally I used userError to display a error message if the username is invalid. In the next screenshot I have the validation for the username input. I retrieve the username using userValue and trim the whitespace out of it. I then check the username to make sure it isn't empty and doesn't contain any spaces. If the username ends up to be invalid it will display the userError and make the isValid false.

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



Caption (required) ✓

Describe/highlight what's being shown

Showing User Detail Panel

#4) Show the code related to the UserDetailsPanel

```
public UserDetailsPanel(ICardControls controls) {  //jah80 07 19 2024      You, 1 second ago • Uncommitted changes
    super(new BorderLayout(hgap:10, vgap:10));

    JPanel content = new JPanel();
    content.setLayout(new BoxLayout(content, BoxLayout.Y_AXIS));
    content.setBorder(new EmptyBorder(top:10, left:10, bottom:10, right:10));

    // Add username input
    JLabel userLabel = new JLabel(text:"Username: ");
    JTextField userValue = new JTextField();
    JLabel userError = new JLabel();
    userError.setVisible(aFlag:false);

    content.add(userLabel);
    content.add(userValue);
    content.add(userError);
    content.add(Box.createRigidArea(new Dimension(width:0, height:300))); // Add vertical space

    //Connect buttons
    JButton previousButton = new JButton(text:"Previous");
    previousButton.addActionListener(event -> controls.previous());
    JButton connectButton = new JButton(text:"Connect");
    connectButton.addActionListener(event -> {
        String incomingUsername = userValue.getText().trim();
        if (incomingUsername.isEmpty()) {
            userError.setText(text:"Username must be provided");
            userError.setVisible(aFlag:true);
        } else {
            username = incomingUsername;
            LoggerUtil.INSTANCE.info("Chosen username: " + username);
            userError.setVisible(aFlag:false);
            controls.connect();
        }
    });
```

Caption (required) ✓

Describe/highlight what's being shown

Showing Userdetailspanel

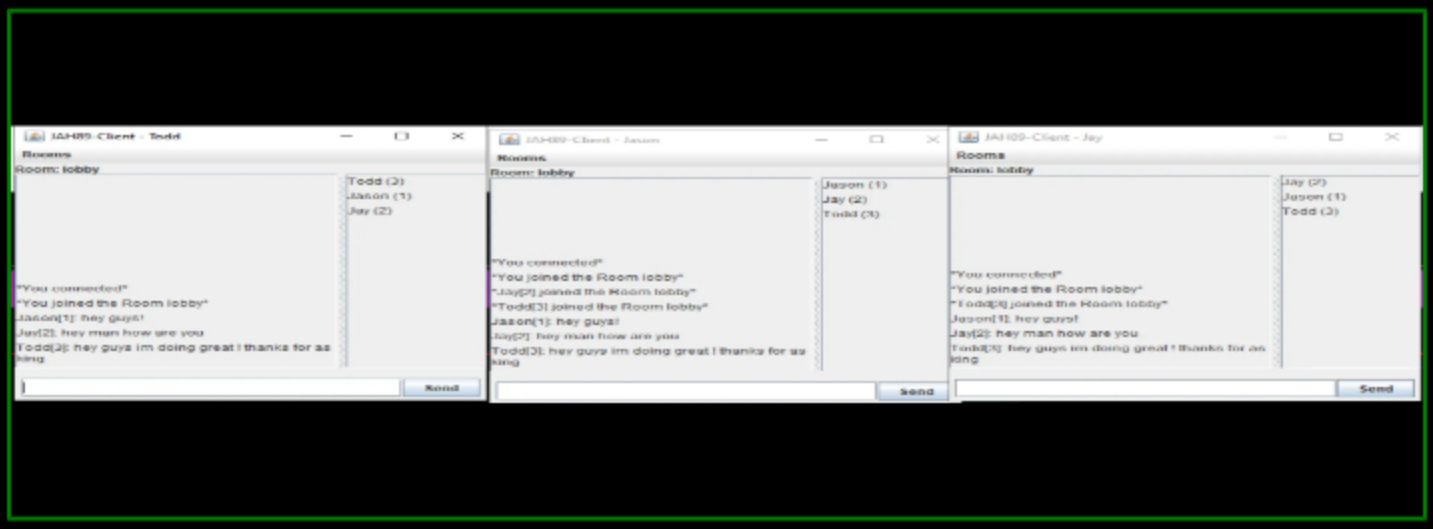
Explanation (required) ✓

Briefly explain how it works and how it's used

PREVIEW RESPONSE

In this code the first thing that happens is it initializes the panel and adds a jpanel that is named "content.". A label "userLabel" is created to have the text "Username" to show the user this is where they will input the username. Then there is a text field so the user can enter their username into the app. Then there is a userError that creates and displays an error message that is related to username problems, such as spaces or empty. Then all these are added to the content panel. Then there is a previousButton and a connectButton to connect to the room with the username entered. When previousButton is triggered it calls the controls.previous() method to navigate to the last page. The connectButton makes sure the username input is correct and will connect them to the lobby. If the username is invalid it will display an error message.

#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

```
chatHistory = new JTextArea(); // jah09 07-20-2024
chatHistory.setEditable(b:false);
JScrollPane chatScrollPane = new JScrollPane(chatHistory);

JPanel input = new JPanel();
input.setLayout(new BoxLayout(input, BoxLayout.X_AXIS));
input.setBorder(new EmptyBorder(top:5, left:5, bottom:5, right:5));

messageInput = new JTextField(); // jah09 07-20-2024
input.add(messageInput);

sendButton = new JButton(text:"Send");

messageInput.addKeyListener(new KeyListener() {
    @Override
    public void keyTyped(KeyEvent e) {
    }

    @Override
    public void keyPressed(KeyEvent e) {
        if (e.getKeyCode() == KeyEvent.VK_ENTER) {
            sendButton.doClick();
        }
    }
    @Override
    public void keyReleased(KeyEvent e) {
    }
});

sendButton.addActionListener((event) -> {
    SwingUtilities.invokeLater(() -> {
        try {
```

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

In the code there is a JPanel that is named input with a boxlayout to have the components horizontally. There is a message input field with a JTextField that is named "messageInput that allows the user to type their messages. The messageInput was added to the input panel. Then there is a JButton which is named sendButton with the tag "Send". Then the sendButton is added to the input panel. A keylistener is added to the messageInput field to detect when the enter key is pressed. When the enter key is pressed it triggers the "doCLick" method in the sendButton which simulates a button click.

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 example of coinflip and roll

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

```
// jah89 07-04-2024
protected synchronized void processRollCommand(ServerThread sender, RollPayload rollPayload) { // jah89 07-04-2024
    String message = String.format(format:"<b>%s (roll result): %s</b>", sender.getClientName(), rollPayload.getMessage());
    sendMessage(sender, message);
}

protected synchronized void processFlipCommand(ServerThread sender, Payload flipPayload) { // jah89 07-04-2024
    String message = String.format(format:"<b>%s (flip result): %s</b>", sender.getClientName(), flipPayload.getMessage());
    sendMessage(sender, message);
}
// jah89 07-07-2024
private String processMessageFormatting(String message) {
    // Bold **
    message = message.replaceAll(regex:"\\*\\*(.*?)\\*\\*", replacement:"<b>$1</b>");

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

Caption (required) ✓

Describe/highlight what's being shown

Showing room code to change format

Explanation (required) ✓

Explain what you did and how it works

PREVIEW RESPONSE

I wasn't able to get this code to work, but i'm not sure how else to go about it so I will explain what my thought process was in trying to get it to work. I used processMessageFormatting to apply formatting such as bold or italics. When the user sends the command it's suppose to show the format with the senders name and the result wrapped in bold format . Then the formatted message would be sent to all the clients in the room using the sendMessage method. I'm not too sure how to get the html tags to work for this code but left this part for last because I assumed it was the easiest but I was wrong.

COLLAPSE

Task #2 - Points: 1

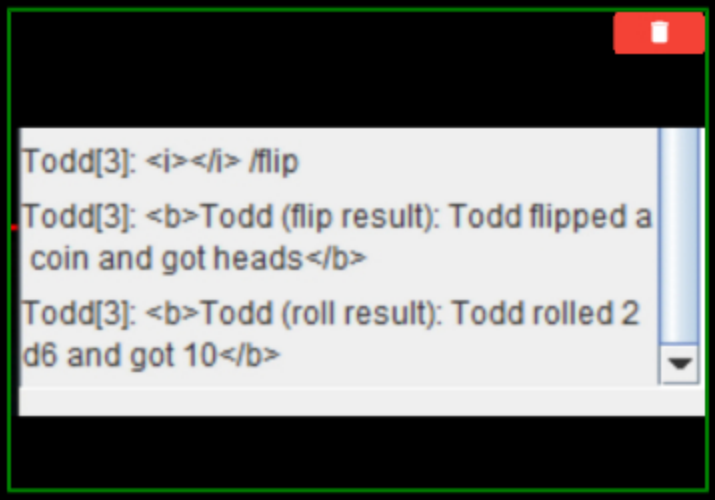
Text: Text Formatting appears correctly on the UI

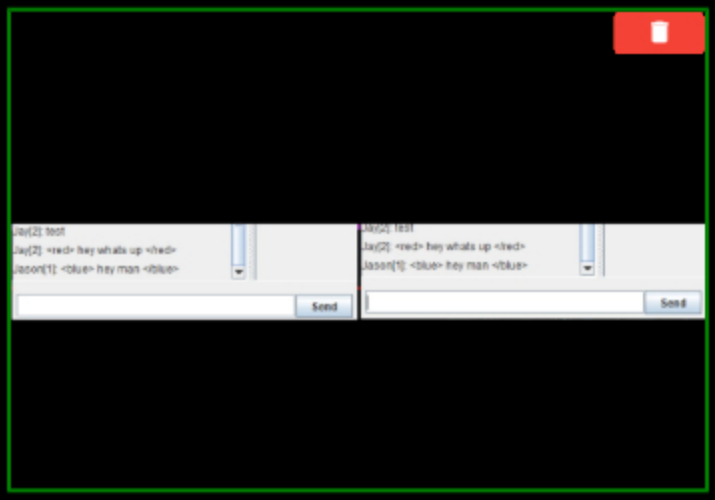
Details:

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 what I got for formatting text styles

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

```
}
// jah89 07-07-2024
private String processMessageFormatting(String message) {
    // Bold **
    message = message.replaceAll(regex:"\\*\\*(.*?)\\*\\*", replacement:"<b>$1</b>");

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

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

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

    return message;
}
```

Caption (required) ✓

Describe/highlight what's being shown

Showing code to change color and style

Explanation (required) ✓

Briefly explain what was necessary and how it works

PREVIEW RESPONSE

This code looks for texts surrounded by the formats, such as ** would represent bold text and one * to represent italics. For color formatting it would look for a hashatg then the color and a closing hastag as well. For example for text to be red I would do "#r Hey guys hows it going r#". This in theory should make the text red for all clients in the chat room.

New Features (4 pts.)

^COLLAPSE^

^COLLAPSE^

Task #1 - Points: 1

Text: Private messages via @username

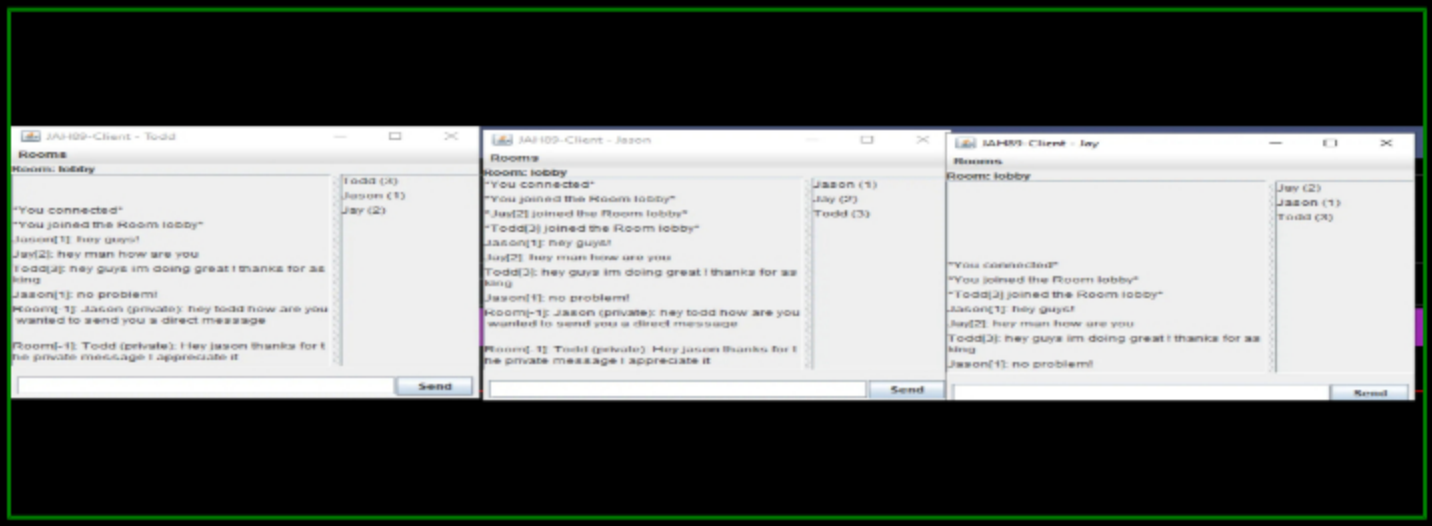
Details:

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 examplef private messages

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

```
sendButton.addActionListener((event) -> {
    SwingUtilities.invokeLater(() -> {
        try {
            String text = messageInput.getText().trim(); //JAH09 07-20-2024
            if (!text.isEmpty()) {
                if (text.startsWith(prefix:"@")) {

                    int spaceIndex = text.indexOf(str:" ");
                    if (spaceIndex != -1) {
                        String targetName = text.substring(beginIndex:1, spaceIndex);
                        String privateMessage = text.substring(spaceIndex + 1);
                        long targetId = Client.INSTANCE.getClientIdByName(targetName);
                        if (targetId == -1) {
                            chatHistory.append("User " + targetName + " not found.\n");
                        } else {
                            Client.INSTANCE.sendPrivateMessage(targetId, privateMessage);
                            chatHistory.append("To " + targetName + ": " + privateMessage + "\n");
                        }
                    }
                } else {
                    Client.INSTANCE.sendMessage(text);
                    chatHistory.append("Me: " + text + "\n");
                }
                messageInput.setText(t:""); // Clear the original text
            }
        } catch (NullPointerException | IOException e) {
            LoggerUtil.INSTANCE.severe(message:"Error sending message", e);
        }
    });
});
```

Caption (required) ✓

Describe/highlight what's being shown

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

Explanation (required) ✓

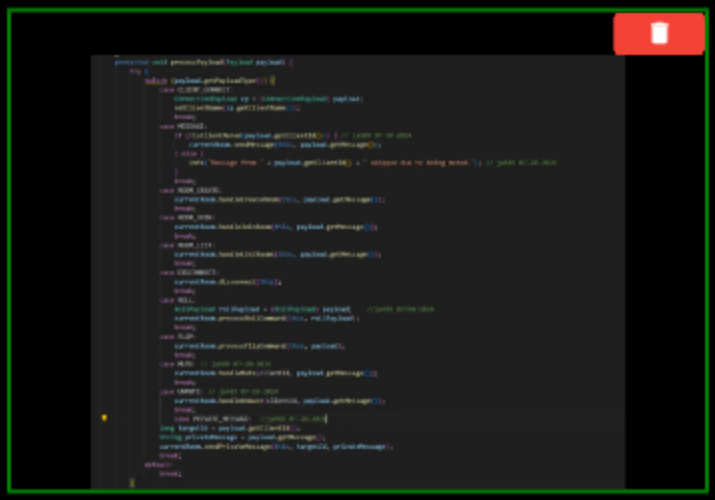
Explain in concise steps how this logically works

PREVIEW RESPONSE

In the code the sendbutton has an action listener that listens for button clicks and also processes the message input by the user. For private message handling it checks for "@" symbol to determine if its a private message or not. It then extracts the username and the message by finding the first space in the message to seperate the target username and the private message itself. In the code to be able to get the target ID for the message it uses "Client.INSTANCE.getClientIdByName(targetName)". If the target user is found it will send the message to the user by using Client.INSTANCE.sendPrivateMessage(targetId, privateMessage). After the message is sent it is also appended to the chat history. If the target user is not found an error will be sent in the chat message. If the message doesn't start with @, it's sent as a normal chat message using "Client.INSTANCE.sendMessage(text)"

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

```
                break;
                case PRIVATE_MESSAGE:  //jah09 07-20-2024
                long targetId = payload.getClientId();
                String privateMessage = payload.getMessage();
                currentRoom.sendPrivateMessage(this, targetId, privateMessage);
                break;
            default:
                break;
        }
    } catch (Exception e) {
        LoggerUtil.INSTANCE.severe("Could not process Payload: " + payload, e);
    }
```



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

I added the private message case to the processpayload method which switches the payload type to determine how to handle the incoming payload. When the payload type is "PRIVATE_MESSAGE" it will extract the targetId and the private message from the payload. It then calls "currentRoom.sendPrivateMessage(this, targetId, privateMessage)" to pass on the sender, target Id and the message to Room to process further.

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

```
    }
    public void sendPrivateMessage(ServerThread sender, long targetId, String message) { //jah89 07-28-2024
        ServerThread target = getClient(targetId);
        if (target == null) {
            sender.sendMessage(message:"User not found.");
            return;
        }
        String formattedMessage = sender.getClientName() + " (private): " + message;
        sender.sendMessage(formattedMessage);
        target.sendMessage(formattedMessage);
    }
```

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

In the room code the first thing I did was target verification. In the method sendPrivateMessage it takes the parameters sender, targetid and message. It verifies that both the sender and target are valid but using "sender != null && target != null". Then the target is retrieved by using getClient(targetId). If the sender and target are both valid the method sends the private message to both the sender and the target. The code "sender.sendMessage sends the message to the sender, which includes the senders ID and targets ID inside the payload. And then target.sendMessage will send the message to the target, which includes the senders ID and the target ID's in the payload.

COLLAPSE

Task #2 - Points: 1

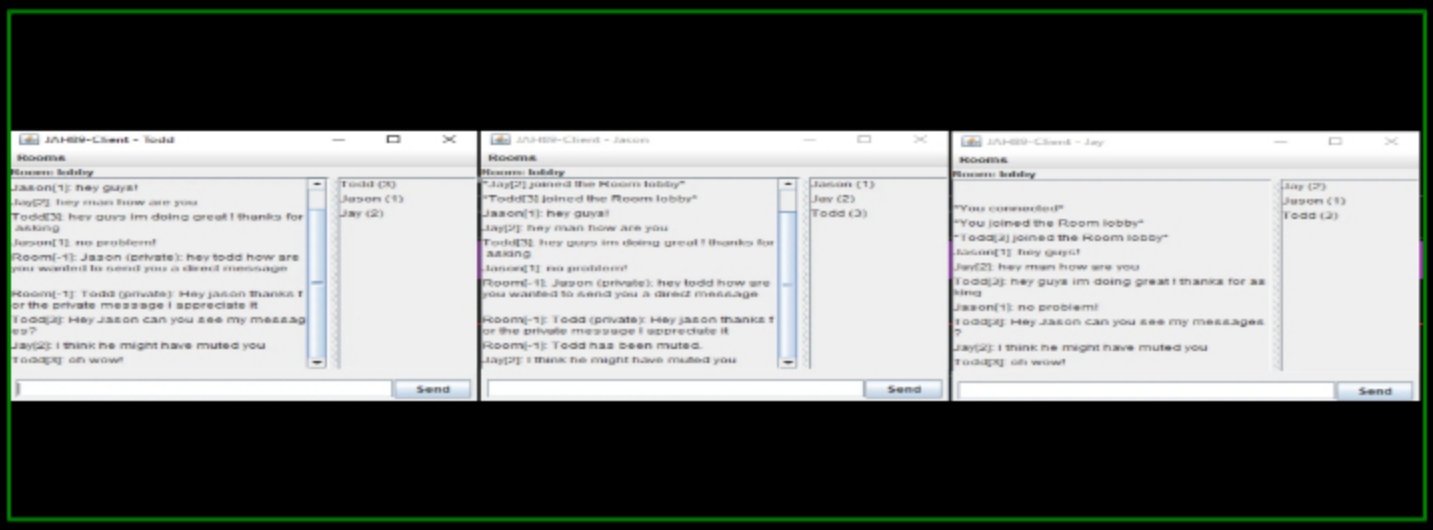
Text: Mute and Unmute

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



Caption (required) ✓

Describe/highlight what's being shown

Showing example of someone getting muted

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

```
}
public void sendMute(String targetName) throws IOException { // jah89 07-20-2024
    Payload p = new Payload();
    p.setPayloadType(PayloadType.MUTE);
    p.setMessage(targetName);
    send(p);
}
public void sendUnmute(String targetName) throws IOException { // jah89 07-20-2024
    Payload p = new Payload();
    p.setPayloadType(PayloadType.UNMUTE);
    p.setMessage(targetName);
    send(p);
}
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the client-side code that processes unmute and mute

Explanation (required) ✓

Explain in concise steps how this logically works

PREVIEW RESPONSE

In this code the first thing the sendMute method does is create a new payload object. It then sets the payload type to "MUTE" using p.setPayloadType(PayloadType.MUTE). It then sets the message of the payload to the target name of the user to mute by using p.setMessage(targetName). It will then pass the payload to the server using send(p) method. This is the same for the UNMUTE payload except the payload type is set to "UNMUTE". How this works is when a user enters /mute (username) the client code will detect the /mute or /unmute command and it will call the method corresponding to the command either sendMute or sendUnmute with the username. Then the new payload will be created and will get sent to the server.

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

```
            break;
        case MUTE: // jah89 07-20-2024
            currentRoom.handleMute(clientId, payload.getMessage());
            break;
        case UNMUTE: // jah89 07-20-2024
            currentRoom.handleUnmute(clientId, payload.getMessage());
            break;
            case PRIVATE_MESSAGE:   //jah89 07-20-2024
        long targetId = payload.getClientId();
        String privateMessage = payload.getMessage();
        currentRoom.sendPrivateMessage(this, targetId, privateMessage);
        break;
    default:
        break;
}
} catch (Exception e) {
    LoggerUtil.INSTANCE.severe("Could not process Payload: " + payload, e);
}
```

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

This code is located inside the processPayload method where the method will check the payload type by using a switch statement. For each payload type there is an action to handle it which in this case is handleUnmute and handleMute. For these payload types the handleMute method calls the MUTE payloads and passes the clientID and the target username. The handlueUnmute method is called for unmute and passes the client and target username as well. So when a user uses /mute or /unmute from the client side the payload will be created and sent to the server where it will invoke the proper method (handleMute or handleUnmute).

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

```
// jah89 07-20-2024
public void handleMute(long senderId, String targetName) {
    ServerThread sender = getClient(senderId);
    if (sender == null) return;

    ServerThread target = getClientByName(targetName);
    if (target == null) {
        sender.sendMessage(message:"User not found.");
        return;
    }

    sender.addMutedClient(target.getClientId());
    sender.sendMessage(targetName + " has been muted.");
}

// jah89 07-20-2024
public void handleUnmute(long senderId, String targetName) {
    ServerThread sender = getClient(senderId);
    if (sender == null) return;

    ServerThread target = getClientByName(targetName);
    if (target == null) {
        sender.sendMessage(message:"User not found.");
        return;
    }

    sender.removeMutedClient(target.getClientId());
    sender.sendMessage(targetName + " has been unmuted.");
}
```

Caption (required) ✓

Describe/highlight what's being shown

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

Explanation (required) ✓

Explain in concise steps how this logically works

PREVIEW RESPONSE

In the handleMute method it gets the serverthread instance for the sender using getClient method. If the sender ends up being null it will return. It then retrives the serverthread for the target user by using getClientByname method. If the target is null it will send a "User not found" message and then returns. If the target is found then it will add the targets client id to the senders muted clients list using the "addMutedClient" method. It then sends a confirmation message to the sender that the target has been muted. This process is basically identical for the handleUnmute method.

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

```
info(String.format(format:"sending message to %s recipients: %s", getName(), clientsInRoom.size(), message));
clientsInRoom.values().removeIf(client -> {
    if (client.isClientMuted(senderId)) { //jah89 07-22-2024
        info(String.format(format:"Message from %s to %s avoided due to mute", senderId, client.getClientId()));
        return false;
    }
    boolean failedToSend = !client.sendMessage(senderId, finalMessage);
    if (failedToSend) {
        info(String.format(format:"Removing disconnected client[%s] from list", client.getClientId()));
        disconnect(client);
    }
    return failedToSend;
});
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the Room code that checks the mute list during send message. private message

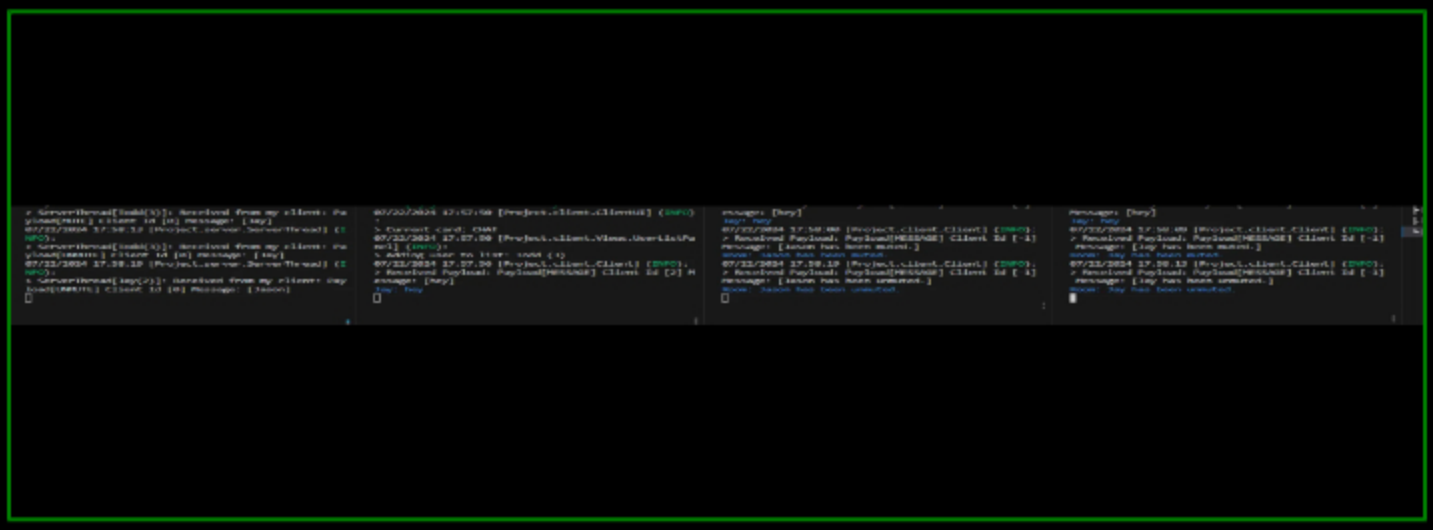
Explanation (required) ✓

Explain in concise steps how this logically works

PREVIEW RESPONSE

In the code for each client it will check if the senders ID is in the client's mute list using the "isClientMuted" method. If the sender is muted by the client a log is created to indicate that the message from the sender was skipped due to them being muted. This iteration will proceed to the next client without sending the message to the muted client. If the sender is not muted by the client the message is sent to the client using the sendMessage method.

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



Caption (required) ✓

Describe/highlight what's being shown

Describe/highlight what's being shown

Showing terminal events when users are muted and unmuted

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/jah89/jah89-IT114-450/pull/16

URL

https://github.com/jah89/jah89-IT114-450/pull/

+ ADD ANOTHER URL

COLLAPSE

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

I had issues with getting the formatting to work correctly. Unfortunately I saw this part of the assignment and thought it was a simple task and left it for the last day and ended up coming into trouble. I tried researching online but when I tried to implment it gave me errors and I couldn't get it to work so I had to leave it the way it is. Hopefully when I have time later after the submission I can go in and try to get it to work correctly for the assignment.

COLLAPSE

Task #3 - Points: 1

Text: WakaTime Screenshot

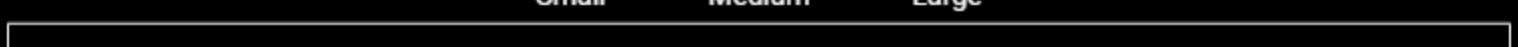
Details:

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



Wakatime

wakatime

End of Assignment