Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-002-S2024/it114-chatroom-milestone-4-2024/grade/mth39

IT114-002-S2024 - [IT114] Chatroom Milestone 4 2024

Submissions:

Submission Selection

1 Submission [active] 4/29/2024 10:08:45 AM

Instructions

COLLAPSE

Implement the Milestone 4 features from the project's proposal document: https://docs.google.com/document/d/1ONmvEvel97GTFPGfVwwQC96xSsobbSbk56145Xf
Make sure you add your ucid/date as code comments where code changes are done
All code changes should reach the Milestone4 branch
Create a pull request from Milestone4 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: Milestone4

Tasks: 15 Points: 10.00

Demonstrate Chat History Export (2.25 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots of code

Checklist

*The checkboxes are for your own tracking

#	Points	Details

☐ #1	1	Show the code that gets the messages and writes it to a file (recommended to use a StringBuilder)
☐ #2	1	File name should be unique to avoid overwriting (i.e., incorporate timestamp)
☐ #3	1	Screenshots should include ucid and date comment
☐ #4	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
private void exportChatHistory() throws IOException {   // UCID: mth39, Date: 04/29/24, Milestone 4
    StringBuilder chatHistory = new StringBuilder();

    // Iterate through chat messages and append them to StringBuilder
    for (Component component : chatArea.getComponents()) {       You, 1 second ago • Uncommitted changes
        if (component instanceof JEditorPane) {
            JEditorPane textContainer = (JEditorPane) component;
            chatHistory.append(textContainer.getText()).append("\n");
        }
    } <- #178-183 for (Component component : chatArea.getComponents())

    // Use JFileChooser to prompt the user for file location and name
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setDialogTitle("Save Chat History");
    fileChooser.setFileFilter(new FileNameExtensionFilter("Text files", "txt"));

    int userSelection = fileChooser.showSaveDialog(this);
    if (userSelection == JFileChooser.APPROVE_OPTION) {
        File fileToSave = fileChooser.getSelectedFile();

        // Add timestamp to filename
        String fileName = fileToSave.getAbsolutePath();
        if (!fileName.toLowerCase().endsWith(".txt")) {
            fileName += ".txt";
        }

        // Write chat history to the selected file
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileName))) {
            writer.write(chatHistory.toString());
        }

        System.out.println("Chat history exported to: " + fileName);
    } <- #191-206 if (userSelection == JFileChooser.APPROVE_OPTION)
} <- #174-207 private void exportChatHistory() throws IOException
```

Screenshot of exportChatHistory method

Checklist Items (0)

COLLAPSE

Task #2 - Points: 1

Text: Screenshot of the file

Checklist *The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Show content with variation of messages (i.e., flip, roll, formatting, etc)
☐ #2	1	It should be clear who sent each message
☐ #3	1	Each screenshot should be clearly captioned

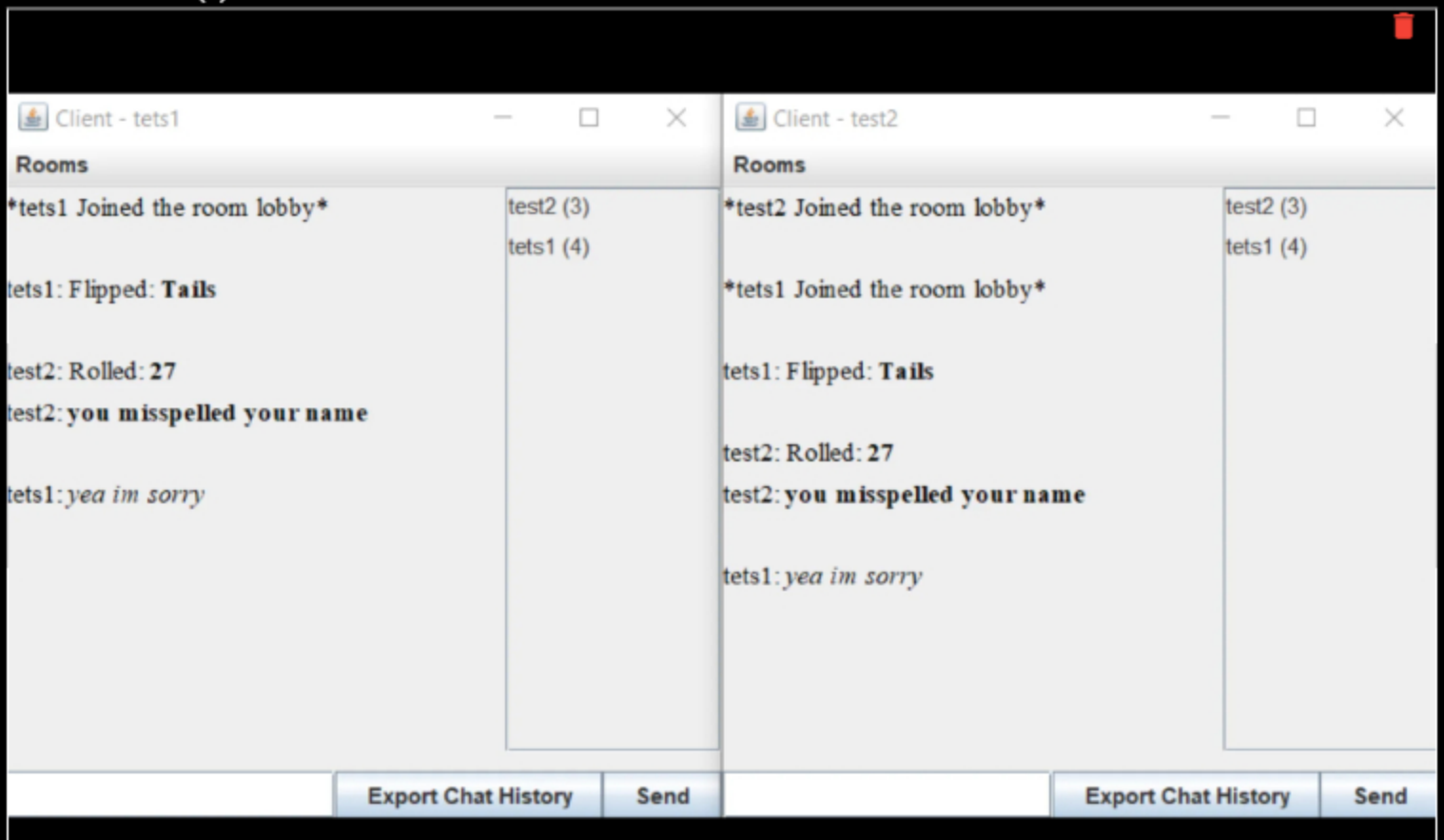
Task Screenshots:

Gallery Style: Large View

Small Medium Large

Screenshot of the contents of the chat history export

Checklist Items (0)



Screenshot of the chat that was exported

Checklist Items (0)

COLLAPSE

Task #3 - Points: 1

Text: Explain solution

Checklist *The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Mention where the messages are stored and how you fetched them
☐ #2	1	Mention how the file is generated and populated

Response:

The messages are stored in the chatHistory StringBuilder. They are fetched by iterating through the chatArea JPane. Each part is checked if it is an instance of JEditorPane. If it is, the text content of that component is appended to the chatHistory StringBuilder.

A JFileChooser dialog is used to prompt the user to select a location and filename for saving the chat history. The user's selection is checked. If they approve the selection, the selected file is obtained from the JFileChooser. A timestamp is added to the filename and the chat history stored in the chatHistory StringBuilder is written to the selected file using a BufferedWriter. A message is then printed to the console indicating the successful export of the chat history along with the file path.

COLLAPSE

Demonstrate Mute List Persistence (2.25 pts.)

COLLAPSE

Task #1 - Points: 1

Text: Screenshots of the code

Checklist *The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Show the code that saves the mute list to a file with the name of the user it belongs to
☐ #2	1	Show the code that loads the mute list when a ServerThread is connected
☐ #3	1	Screenshots should include ucid and date comment
☐ #4	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large

Small Medium Large

```java
    private void loadMuteListFromFile() { // UCID: mth39, Date: 04/29/24, Milestone 4
    try {
        String fileName = clientName + MUTE_LIST_FILE_SUFFIX;
        File file = new File(fileName);
        if (file.exists()) {
            try (BufferedReader br = new BufferedReader(new FileReader(file))) {
                String line;
                while ((line = br.readLine()) != null) {
                    muteList.add(line);
                }
            } <- #187-192 try (BufferedReader br = new BufferedReader(new FileReader(fi...
        } <- #186-193 if (file.exists())
    } catch (IOException e) {
        logger.severe("Error loading mute list from file: " + e.getMessage());
    }
} // UCID: mth39, Date: 04/24/24, Milestone 4 <- #182-197 private void loadMuteListFromFile()

private void saveMuteListToFile() { // UCID: mth39, Date: 04/29/24, Milestone 4
    try {
        String fileName = clientName + MUTE_LIST_FILE_SUFFIX;
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileName))) {
            for (String username : muteList) {
                writer.write(username);
                writer.newLine();
            }
        } <- #202-207 try (BufferedWriter writer = new BufferedWriter(new FileWrite...
    } catch (IOException e) {
        logger.severe("Error saving mute list to file: " + e.getMessage());
    }
} // UCID: mth39, Date: 04/24/24, Milestone 4 <- #199-211 private void saveMuteListToFile()
```

This screenshot shows the saveMuteListToFile & loadMuteListFromFile methods.

Checklist Items (0)

COLLAPSE

Task #2 - Points: 1

Text: Screenshots of the demo

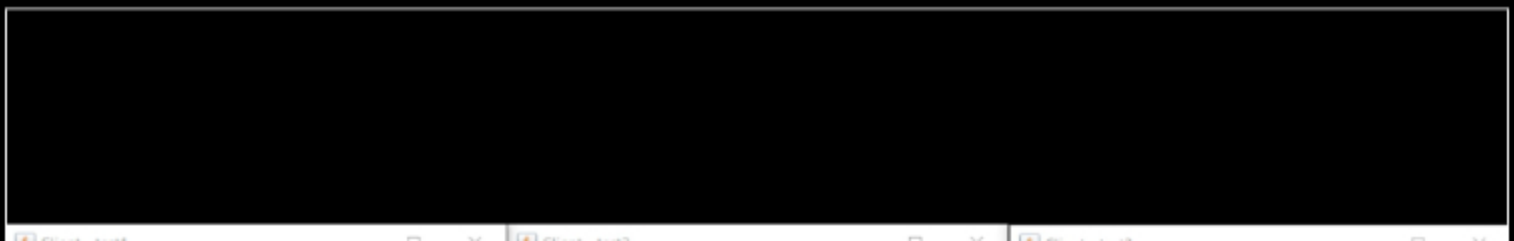
Checklist *The checkbox are for your own tracking

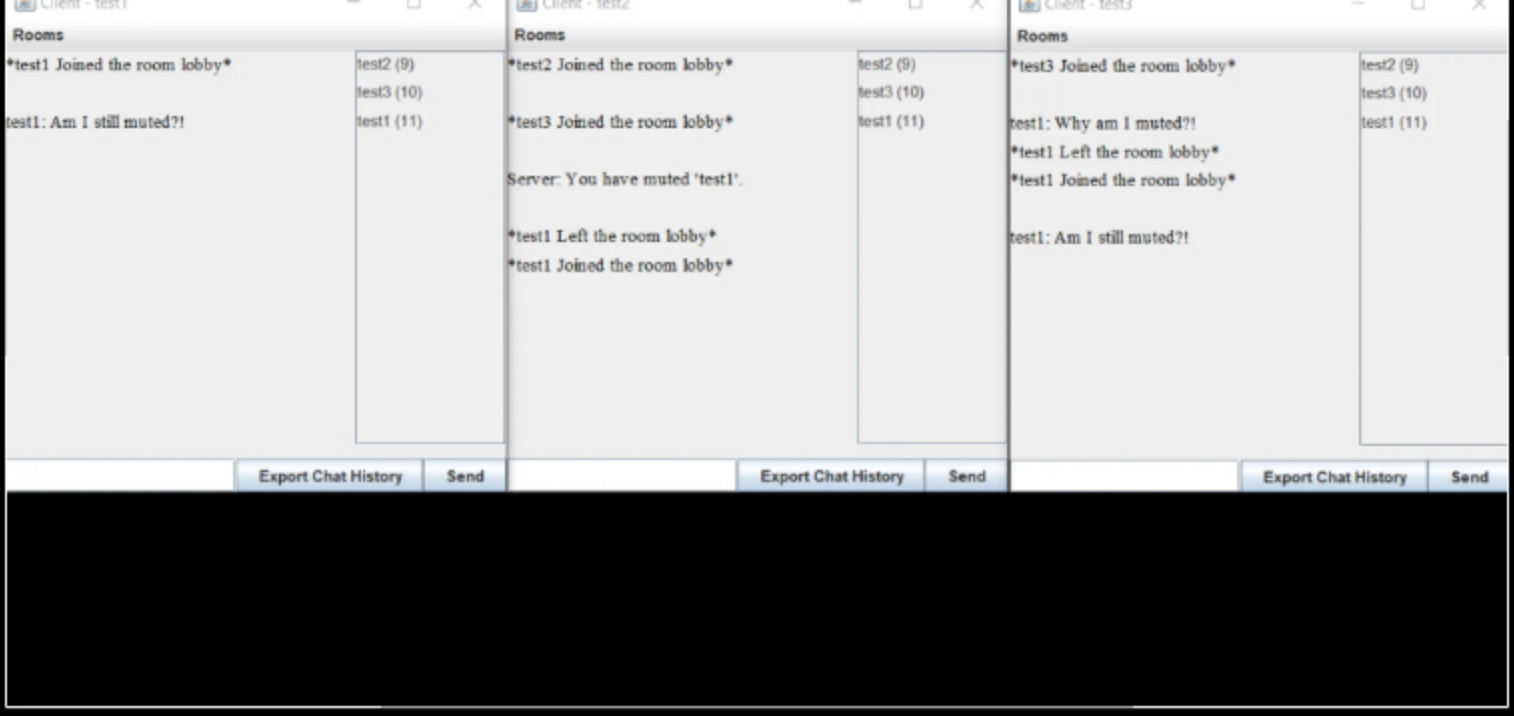
#	Points	Details
☐ #1	1	Show a user muting another user, disconnecting, reconnecting, and still having that user muted (same should be possible if the server restarts)
☐ #2	1	This should also be reflected in the UI per related feature in this milestone
☐ #3	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large





Screenshot shows test2 muting test1. Test1 then disconnects and reconnects but is still muted.

Checklist Items (0)

COLLAPSE

Task #3 - Points: 1

Text: Explain solution

Checklist

*The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Mention how you got the mute list to save and load
☐ #2	1	Discuss the steps to sync the data to the client/ui

Response:

I implemented the saving logic by creating the saveMuteListToFile method, which uses a BufferedWriter to write each muted username from the muteList to a file named after the client's name with the "_mute_list.txt" suffix. For loading, the loadMuteListFromFile method utilizes a BufferedReader to read the usernames from the file associated with the client's name and populates the muteList. Both methods are called during the initialization of a ServerThread instance to ensure persistence and retrieval of the mute list across server sessions.

Demonstrate Mute/Unmute notification (2.25 pts.)

COLLAPSE

Task #1 - Points: 1

Text: Screenshots of the code

Checklist

*The checkboxes are for your own tracking

#	Points	Details

☐ #1	1	Show how the message is sent to the target user only if their mute/unmute state changes (i.e., doing mute twice for the same user shouldn't send two mute messages)
☐ #2	1	Screenshots should include ucid and date comment
☐ #3	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
private void processMuteCommand(ServerThread sender, String targetUsername) {  // UCID: mth39, Date: 04/29/24, Milestone 4
    // Check if the sender and targetUsername are valid
    if (sender != null && targetUsername != null && !targetUsername.isEmpty()) {
        // Find the ServerThread representing the target user
        ServerThread targetClient = findClientByUsername(targetUsername);

        // If the target user is found      You, 10 seconds ago • Uncommitted changes
        if (targetClient != null) {
            // Check if the target user is currently muted
            boolean wasMuted = targetClient.isMuted(sender.getClientName());

            // Mute the target user by adding the sender's username to their mute list
            targetClient.mute(sender.getClientName());

            // Check if the mute status changed
            if (!wasMuted) {
                // Inform the sender that they have successfully muted the target user
                sender.sendMessage(Constants.DEFAULT_CLIENT_ID, "You have muted '" + targetUsername + "'.");
                // Inform the target user that they have been muted by the sender
                targetClient.sendMessage(Constants.DEFAULT_CLIENT_ID, "You have been muted by " + sender.getClientName());
            } <- #257-262 if (!wasMuted)
        } else {
            // If the target user is not found, inform the sender
            sender.sendMessage(Constants.DEFAULT_CLIENT_ID, "User '" + targetUsername + "' not found.");
        }
    } <- #244-267 if (sender != null && targetUsername != null && !targetUserna...
} <- #242-268 private void processMuteCommand(ServerThread sender, String t...

private void processUnmuteCommand(ServerThread sender, String targetUsername) {  // UCID: mth39, Date: 04/29/24, Milestone 4
    // Check if the sender and targetUsername are valid
    if (sender != null && targetUsername != null && !targetUsername.isEmpty()) {
        // Find the ServerThread representing the target user
        ServerThread targetClient = findClientByUsername(targetUsername);

        // If the target user is found
        if (targetClient != null) {
            // Check if the target user is currently muted
            boolean wasMuted = targetClient.isMuted(sender.getClientName());

            // Unmute the target user by removing the sender's username from their mute list
            targetClient.unmute(sender.getClientName());

            // Check if the mute status changed
            if (wasMuted) {
                // Inform the sender that they have successfully unmuted the target user
                sender.sendMessage(Constants.DEFAULT_CLIENT_ID, "You have unmuted '" + targetUsername + "'.");
                // Inform the target user that they have been unmuted by the sender
                targetClient.sendMessage(Constants.DEFAULT_CLIENT_ID, "You have been unmuted by " + sender.getClientName());
            } <- #285-290 if (wasMuted)
        } else {
            // If the target user is not found, inform the sender
            sender.sendMessage(Constants.DEFAULT_CLIENT_ID, "User '" + targetUsername + "' not found.");
        }
    } <- #272-295 if (sender != null && targetUsername != null && !targetUserna...
} <- #270-296 private void processUnmuteCommand(ServerThread sender, String...
```

Screenshot showing processMuteCommand and processUnmuteCommand methods

Checklist Items (0)

COLLAPSE

Task #2 - Points: 1

Text: Screenshots of the demo

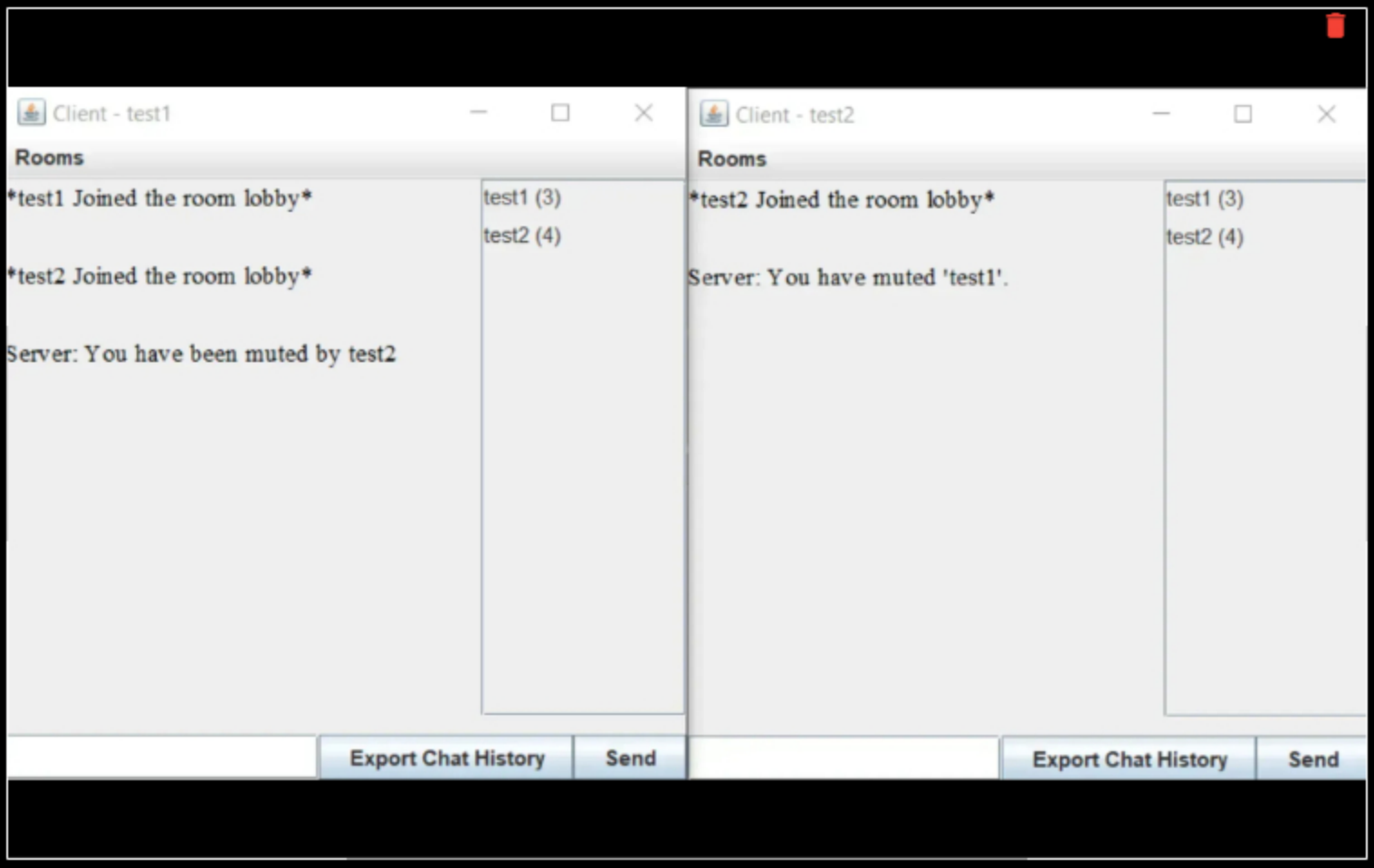
Checklist *The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Show examples of doing /mute twice in succession for the same user only yields one message
☐ #2	1	Show examples of doing /unmute twice in succession for the same user only yields one message
☐ #3	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large



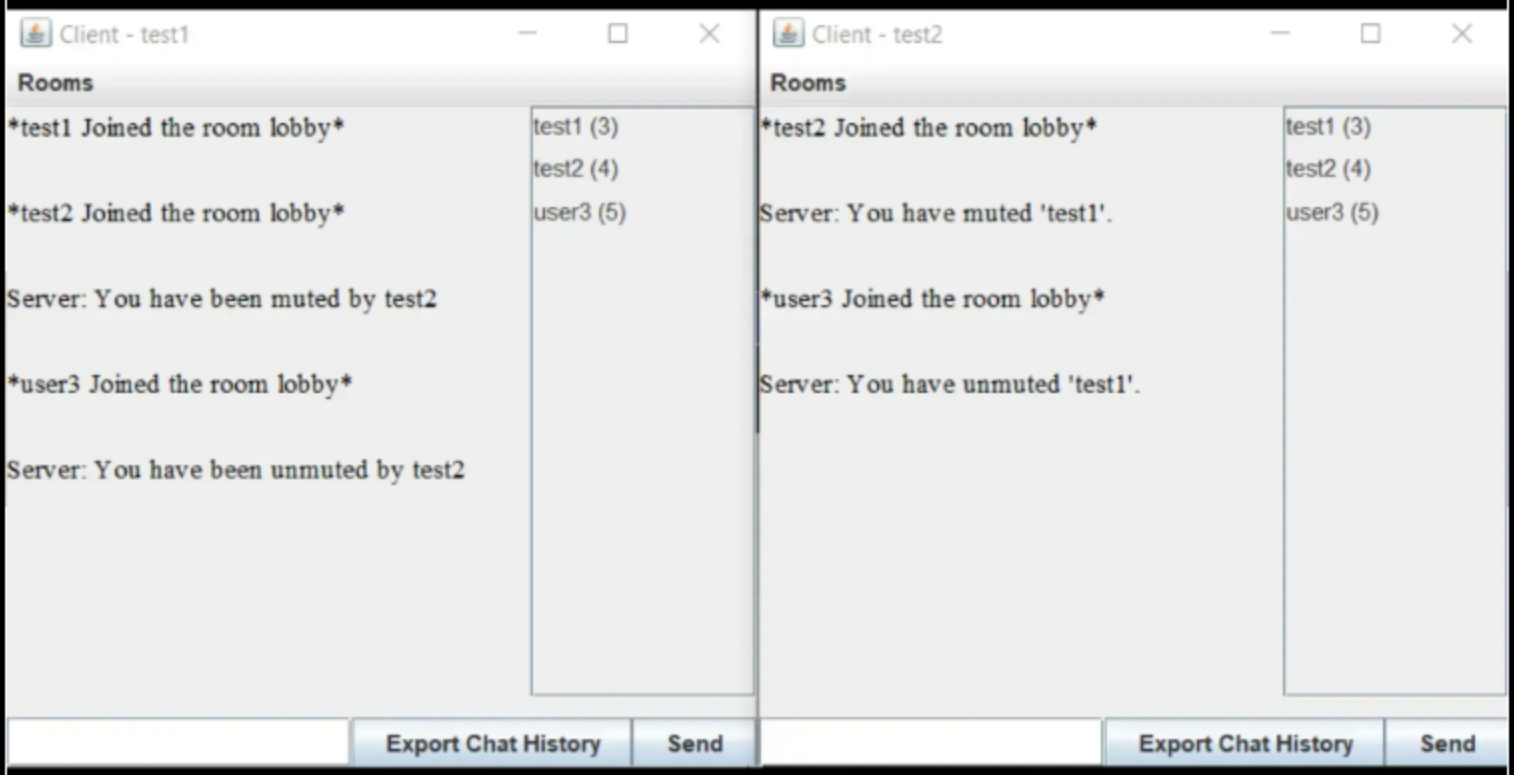
Screenshot of using the /mute command twice in succession for the same user but yielding only one message

Checklist Items (0)

```
Apr 29, 2024 2:55:50 PM Project.server.ServerThread run
INFO: Received from client: Type[MESSAGE],ClientId[0,] ClientName[test2
], Message[/mute test1]
Apr 29, 2024 2:55:50 PM Project.server.ServerThread send
INFO: Sent payload: Type[MESSAGE],ClientId[-1,] ClientName[null], Message[You h
ave muted 'test1'.]
Apr 29, 2024 2:55:50 PM Project.server.ServerThread send
INFO: Sent payload: Type[MESSAGE],ClientId[-1,] ClientName[null], Message[You h
ave been muted by test2]
Apr 29, 2024 2:55:56 PM Project.server.ServerThread run
INFO: Received from client: Type[MESSAGE],ClientId[0,] ClientName[test2], Messa
ge[/mute test1]
```

Screenshot of terminal showing that the /mute command was used twice in succession for the same user

Checklist Items (0)



Screenshot of using the /unmute command twice in succession for the same user but yielding only one message

Checklist Items (0)

```
Apr 29, 2024 2:59:46 PM Project.server.ServerThread run
INFO: Received from client: Type[MESSAGE],ClientId[0,] ClientName[test2], Messa
ge[/unmute test1]
Apr 29, 2024 2:59:46 PM Project.server.ServerThread send
INFO: Sent payload: Type[MESSAGE],ClientId[-1,] ClientName[null], Message[You h
ave unmuted 'test1'.]
Apr 29, 2024 2:59:46 PM Project.server.ServerThread send
INFO: Sent payload: Type[MESSAGE],ClientId[-1,] ClientName[null], Message[You h
ave been unmuted by test2]
Apr 29, 2024 2:59:50 PM Project.server.ServerThread run
INFO: Received from client: Type[MESSAGE],ClientId[0,] ClientName[test2], Messa
ge[/unmute test1]
```

Screenshot of terminal showing that the /mute command was used twice in succession for the same user

Checklist Items (0)

COLLAPSE

Task #3 - Points: 1

Text: Explain solution

Checklist *The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Mention how you limit the messages in each scenario
☐ #2	1	Discuss how you find the correct user to send the message to

Response:

The isMuted method is called on the targetClient with the sender's username. This method checks if the sender's username is already in the mute list of the target user. If the target user is already muted (or unmuted) by the sender, no notification messages are sent and the mute status is simply updated. This prevents duplicate notification messages when muting or unmuting the same user in succession. If the status changes (the user is off of the mute list or vice versa), notification messages are sent to both the user who muted and the user who got muted.

The findClientByUsername method is called to locate the targetUsername in ServerThread. If the target user is found (targetClient != null), notification messages are sent to both the sender and the target user. If the target user is not found, a message is sent to the sender informing them that the target user was not found.

COLLAPSE

Demonstrate user list visual changes (2.25 pts.)

COLLAPSE

Task #1 - Points: 1

Text: Screenshots of the code

Checklist *The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Show the code related to "graying out" muted users and returning them to normal when unmuted
☐ #2	1	Show the code related to highlighting the user who last sent a message (and

		unhighlighting the remainder of the list)
#3	1	Screenshots should include ucid and date comment
#4	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
    protected void updateUserMuteStatus(long clientId, boolean isMuted) {  //  UCID: mth39, Date: 04/30/24, Milestone 4
        JEditorPane textContainer = userMap.get(clientId);
        if (textContainer != null) {
            Font font = textContainer.getFont();
            Map<TextAttribute, Object> attributes = new HashMap<>(font.getAttributes());
            attributes.put(TextAttribute.FOREGROUND, isMuted ? Color.GRAY : Color.BLACK);
            textContainer.setFont(font.deriveFont(attributes));
        } <- #115-120 if (textContainer != null)
    } <- #113-121 protected void updateUserMuteStatus(long clientId, boolean is...

    Codeium: Refactor | Explain | Generate Javadoc | ×
    protected void highlightLastMessageSender(long clientId) {  //  UCID: mth39, Date: 04/30/24, Milestone 4
        if (lastMessageSenderId != -1) {
            JEditorPane lastMessageSender = userMap.get(lastMessageSenderId);
            if (lastMessageSender != null) {
                Font font = lastMessageSender.getFont();
                Map<TextAttribute, Object> attributes = new HashMap<>(font.getAttributes());
                attributes.put(TextAttribute.FOREGROUND, Color.BLACK);
                lastMessageSender.setFont(font.deriveFont(attributes));
            } <- #126-131 if (lastMessageSender != null)
        } <- #124-132 if (lastMessageSenderId != -1)
        JEditorPane currentMessageSender = userMap.get(clientId);
        if (currentMessageSender != null) {
            Font font = currentMessageSender.getFont();
            Map<TextAttribute, Object> attributes = new HashMap<>(font.getAttributes());
            attributes.put(TextAttribute.FOREGROUND, Color.BLUE);
            currentMessageSender.setFont(font.deriveFont(attributes));
            lastMessageSenderId = clientId;
        } <- #134-140 if (currentMessageSender != null)
    } <- #123-141 protected void highlightLastMessageSender(long clientId)
} <- #26-142 public class UserListPanel extends JPanel
```

Screenshot of updateUserMuteStatus (responsible for "graying out") and highlightLastMessageSender (responsible for highlighting the last user who sent a message) methods.

Checklist Items (0)

COLLAPSE

Task #2 - Points: 1

Text: Screenshots of the demo

Checklist

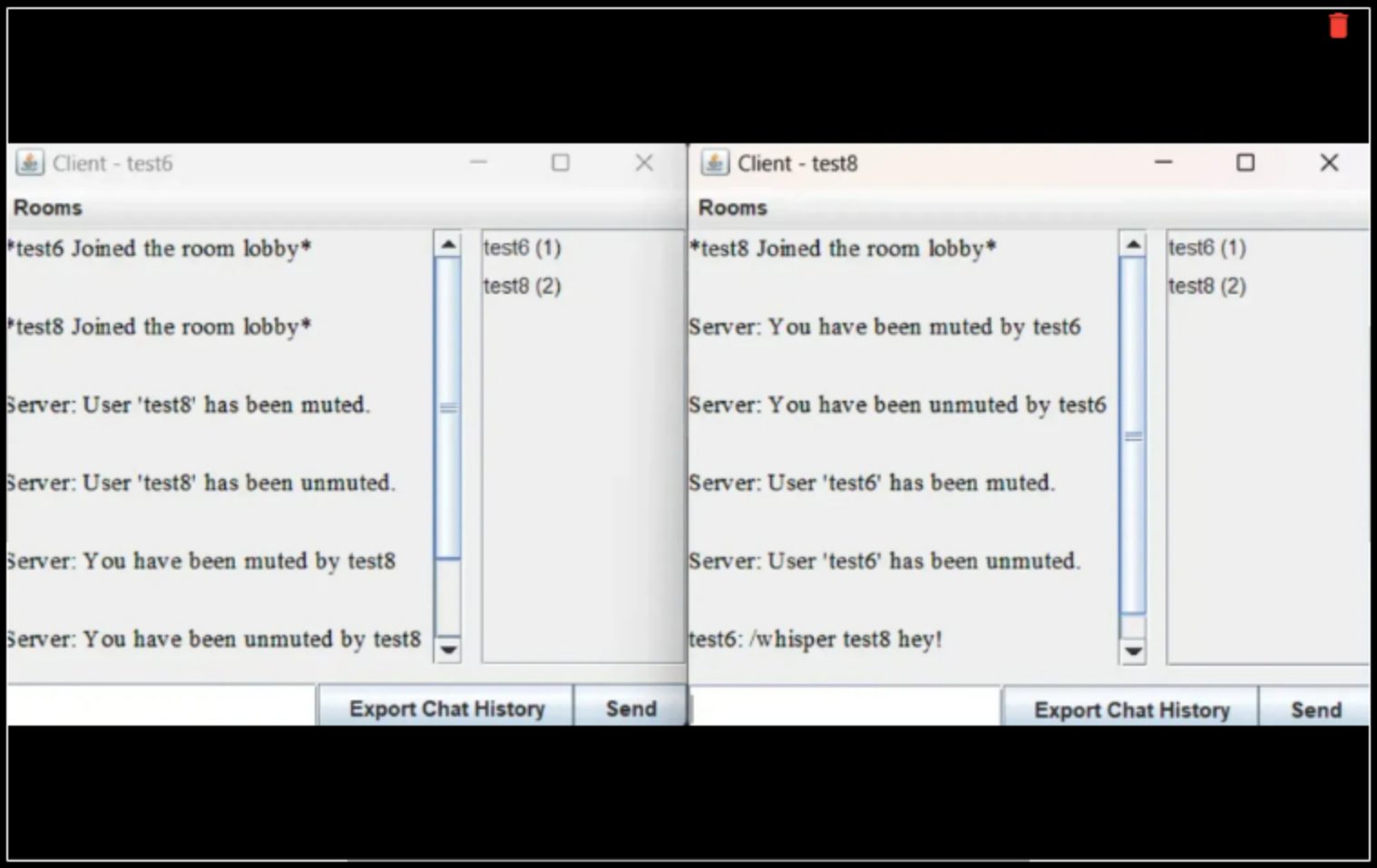
*The checkboxes are for your own tracking

#	Points	Details
#1	1	Show before and after screenshots of the list updating upon mute and unmute
#2	1	Capture variations of "last person to send a message gets highlighted"
#3	1	Each screenshot should be clearly captioned

Task Screenshots:

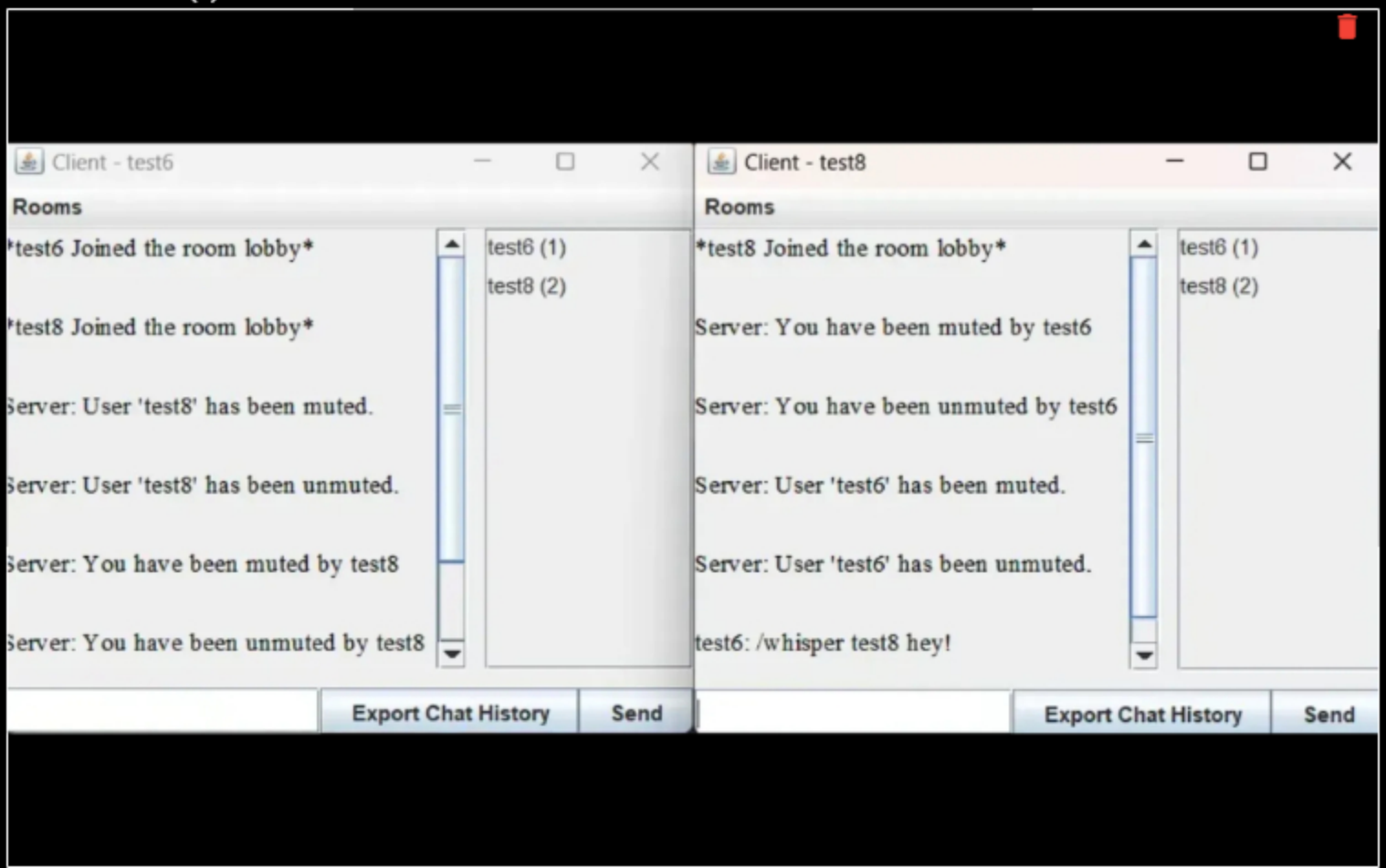
Gallery Style: Large View

Small Medium Large



This screenshot shows a list of muted and unmuted users appearing grayed out.

Checklist Items (0)



This screenshot is supposed to show the highlighted user feature of the last person to send a message. (Doesn't work)

Checklist Items (0)

COLLAPSE

Task #3 - Points: 1

Text: Explain solution

Checklist

*The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Mention how you got the mute/unmute effect implemented
☐ #2	1	Mentioned how you got the highlight effect implemented (including unhighlighting the other users)

Response:

To implement the feature of updating the user list based on the status of each user and highlighting the last person to send a message, several changes were made across multiple files. In Room.java, new PayloadTypes were introduced to relay user status information, and the sendConnectionStatus method was extended to include user status updates. The sendResetUserList method was also changed to convey muted user information specific to each client.

In ServerThread.java, methods for sending user-specific payloads, such as sendConnectionStatus, were enhanced to accommodate the new PayloadTypes. The handling of muted users, including loading and saving the mute list to a file, was integrated into the existing methods.

For the client-side implementation, adjustments were made in Client.java, ClientUI.java, ChatPanel.java, and UserListPanel.java. The processPayload method in Client.java was expanded to handle the new PayloadTypes and update the user list accordingly. For the UI components, the UserListPanel now displays muted users differently (in gray). The updateUserMuteStatus method updates the mute status of a user identified by the clientId. It gets the JEditorPane associated with the clientId from the userMap. It gets its current font and creates a map of text attributes from it. Then, based on "isMuted", it sets the foreground attribute of the text to either gray if the user is muted, or black if the user is not muted. Finally, it sets the font of the JEditorPane with the updated attributes. The last person to send a message is highlighted in the chat window. The highlightLastMessageSender method is supposed to highlight the last message sender and unhighlight the previous one. It's supposed to check if there was a previous message sender (if lastMessageSenderId is not -1). If there was, it retrieves the JEditorPane associated with the previous sender from the userMap. If it exists, it gets its current font and creates a map of text attributes from it. It sets the foreground attribute of the text to black, which unhighlights it. Then, it retrieves the JEditorPane associated with the current message sender (clientId) and does the same steps but highlights it blue. Finally, it updates the lastMessageSenderId to the current sender's clientId, making it the new last message sender.

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

Missing URL

COLLAPSE

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

I was not able to get highlight feature to work unfortunately.

COLLAPSE

Task #3 - Points: 1

Text: WakaTime Screenshot

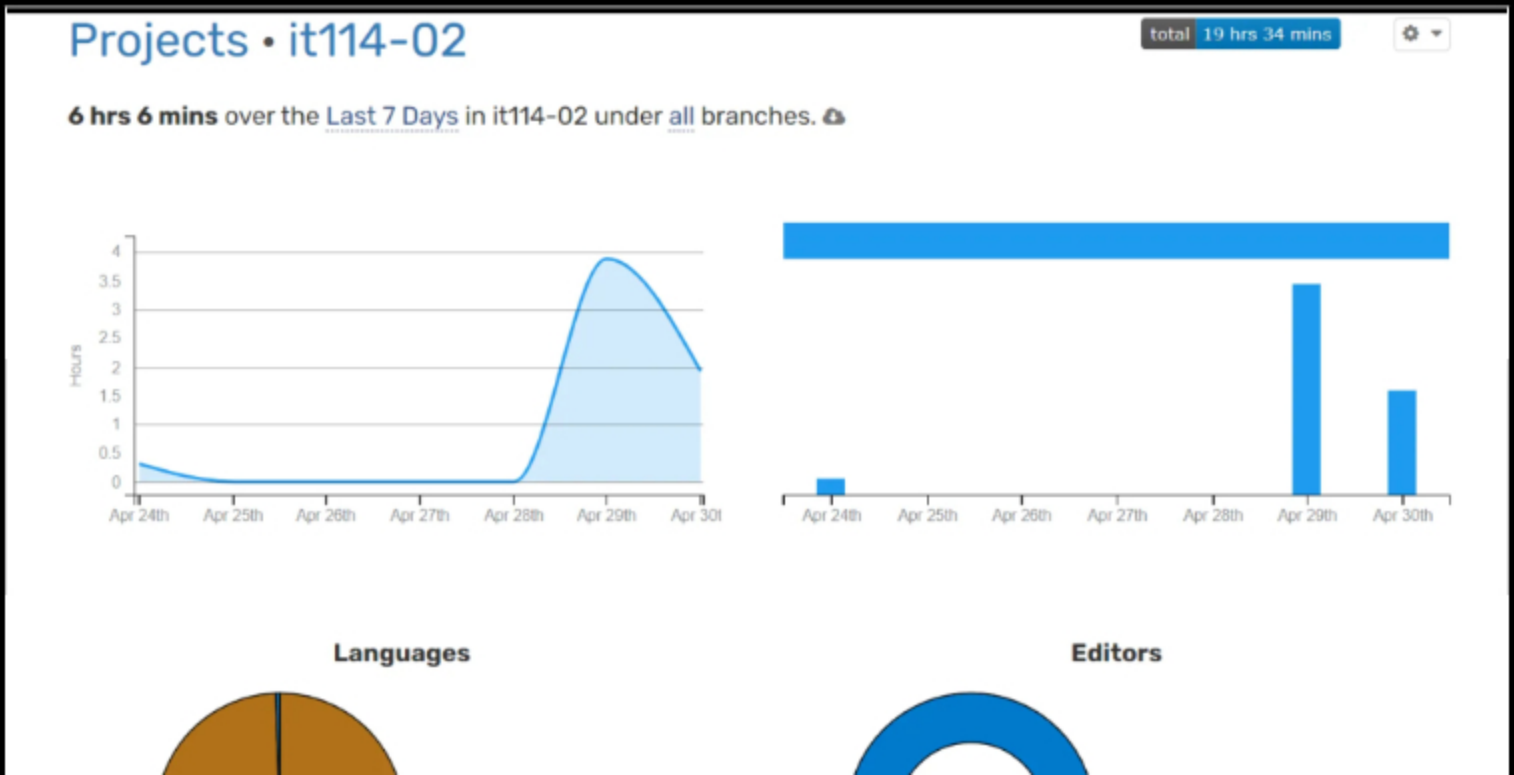
Details:

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 screenshot showing time worked on it114 repository over the past week

End of Assignment