

Submission Worksheet

CLICK TO GRADE

<https://learn.ethereallab.app/assignment/IT114-002-S2024/it114-chatroom-milestone-3-2024/grade/cw72>

IT114-002-S2024 - [IT114] Chatroom Milestone 3 2024

Submissions:

Submission Selection

1 Submission [active] 4/30/2024 10:04:15 PM

Instructions

[^ COLLAPSE ^](#)

Implement the Milestone 3 features from the project's proposal

document: <https://docs.google.com/document/d/10NmvEvel97GTFPGfVwwQC96xSsobbSbk56145X>

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: 14 Points: 10.00

 Basic UI (2 pts.)

[^ COLLAPSE ^](#)

 Task #1 - Points: 1

Text: Screenshots of the following

Checklist

*The checkboxes are for your own tracking

#

Points

Details

#1	1	Connection Panel
#2	1	User Details Panel
#3	1	Chat Panel
#4	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

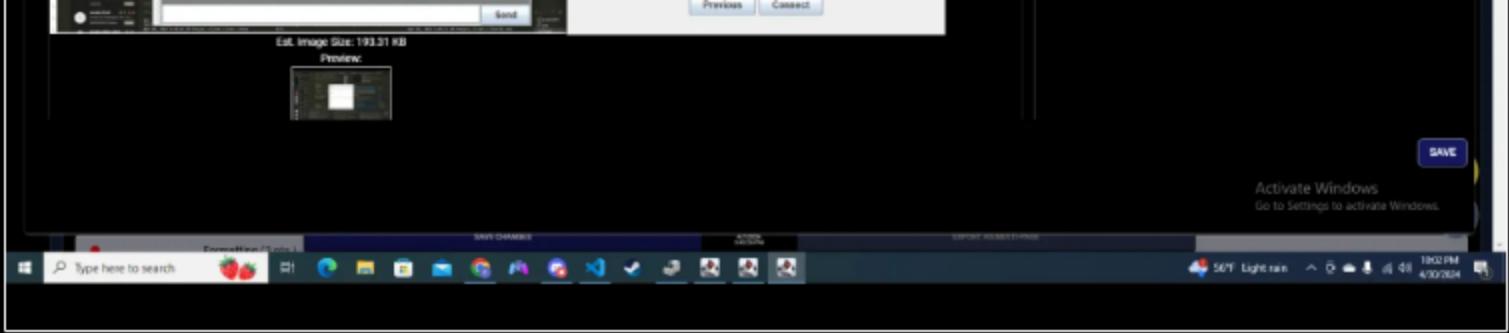
Small Medium Large

This screenshot displays a large gallery view of four panels. At the top, there are four cards with blue icons and labels: #1 (Connection Panel), #2 (User Details Panel), #3 (Chat Panel), and #4 (Clearly caption screenshots). Below these cards is a horizontal navigation bar with three buttons: 'Small', 'Medium', and 'Large'. The main area contains four large, overlapping windows. The first window on the left shows a code editor with Java code related to a Client class. The second window in the middle shows a 'Client' application interface with a 'Rooms' section and a 'Port' input field. The third window on the right shows a 'WakaTime' extension page in the Visual Studio Code Marketplace. The fourth window at the bottom shows a terminal window with Java logs. The overall layout is a grid-based gallery view.

This is the screenshot of the connection panel

Checklist Items (0)

This screenshot shows a checklist interface. On the right, a table titled 'Checklist' lists four items: #1 (Connection Panel), #2 (User Details Panel), #3 (Chat Panel), and #4 (Clearly caption screenshots). Each item has a checkbox next to it. A note says, '*The checkboxes are for your own tracking.' To the left of the checklist, there is a sidebar with the heading 'Screenshots of the following' and a 'Screenshot' button. Below the sidebar, there are two windows: one showing a Java code editor with a 'Client' class and another showing a 'Client' application interface with a 'Rooms' section. A red dashed box highlights the 'Screenshot' button in the sidebar.



This is the screenshot of userDetails panel.

Checklist Items (0)

Checklist		
#	Points	Details
<input checked="" type="checkbox"/> #1	1	Connection Panel
<input checked="" type="checkbox"/> #2	1	User Details Panel
<input checked="" type="checkbox"/> #3	1	Chat Panel
<input checked="" type="checkbox"/> #4	1	Clearly caption screenshots

*The checkboxes are for your own tracking

What checklist items are related to this current image?

Select Related Checklist Items

Caption:

Describe/Highlight what the screenshot is showing

This is the screenshot of the caption panel.

Checklist Items (0)

● Formatting (2 pts.)

▲COLLAPSE ▲

● Task #1 - Points: 1

Text: Screenshots demoing flip and roll commands

Checklist

*The checkboxes are for your own tracking

#	Points	Details
■ #1	1	Flip output in a different format than normal messages

#2	1	Roll # output in a different format than normal messages
#3	1	Roll #d# output in a different format than normal messages
#4	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large

These are the flip and roll commands

Checklist Items (4)

#1 Flip output in a different format than normal messages

#2 Roll # output in a different format than normal messages

#3 Roll #d# output in a different format than normal messages

#4 Clearly caption screenshots

Task #2 - Points: 1

Text: Screenshots demoing custom text formatting

#	Points	Details
#1	1	Custom text formatting for bold working (Part of the message should appear bold)
#2	1	Custom text formatting for italic working (Part of the message should appear italic)
#3	1	Custom text formatting for underline working (Part of the message should appear underline)
#4	1	Custom text formatting for red working (Part of the message should appear red)
#5	1	Custom text formatting for blue working (Part of the message should appear blue)
#6	1	Custom text formatting for green working (Part of the message should appear green)
#7	1	Custom text formatting for combined bold, italic, underline, and a color working (Part of the message should have all 4 formats applied at once)
#8	1	Clearly caption screenshots

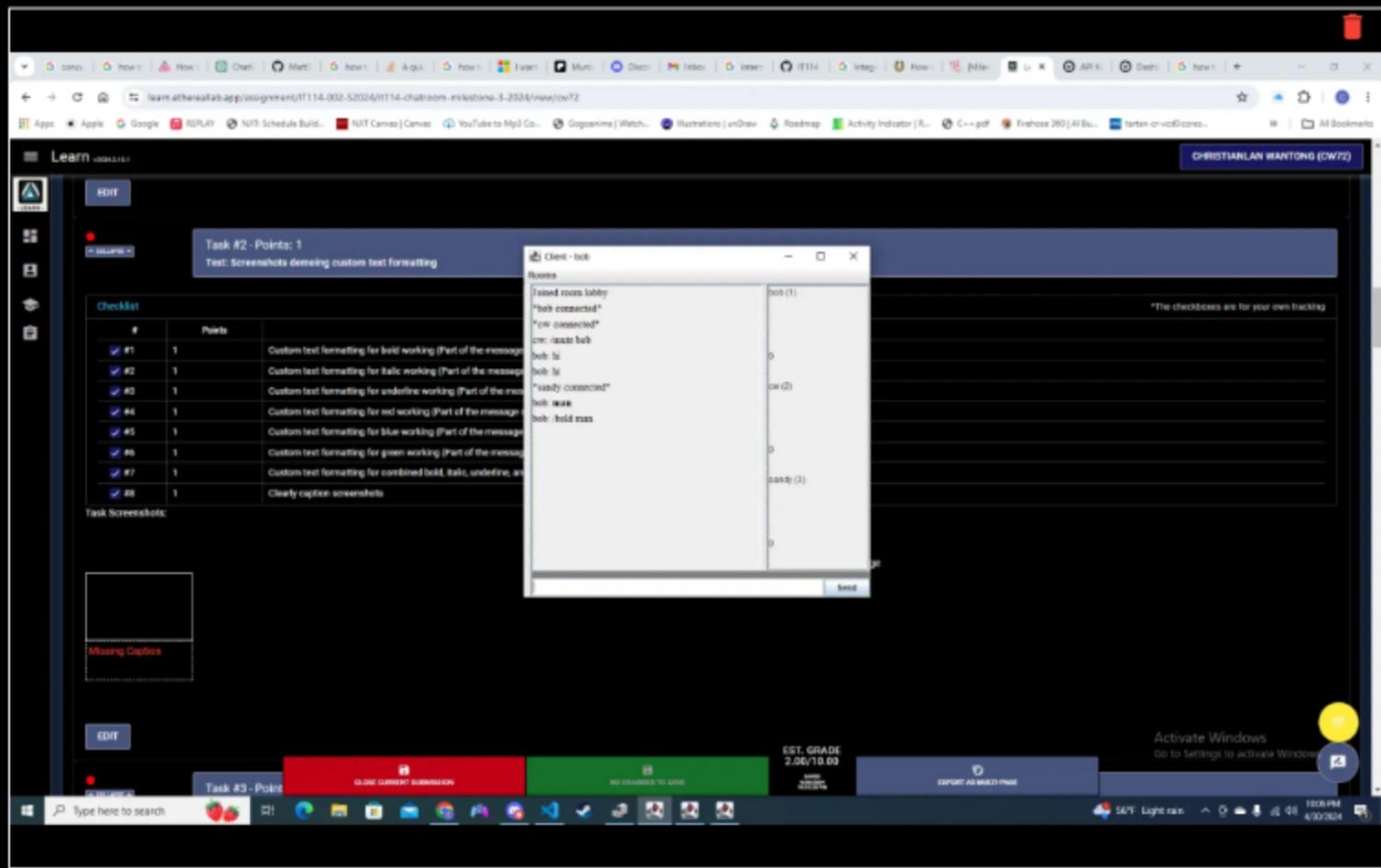
Task Screenshots:

Gallery Style: Large View

Small

Medium

Large



This is the screen of the bold text.

Checklist Items (1)

#1 Custom text formatting for bold working (Part of the message should appear bold)

The screenshot shows a Microsoft Edge browser window with a task assignment in a learning management system. The task is titled "Text: Screenshots demonstrating custom text formatting". The assignment includes a checklist with 8 items, all of which have been completed. One item, "Clearly caption screenshots", has a note: "Missing Caption". Below the checklist is a section for "Task Screenshots", which contains a large empty box and a smaller box labeled "Missing Caption". An "EDIT" button is located below this section.

Task: Screenshots demonstrating custom text formatting

Checklist

#	Points	Description
<input checked="" type="checkbox"/> #1	1	Custom text formatting for bold working (Part of the message)
<input checked="" type="checkbox"/> #2	1	Custom text formatting for italic working (Part of the message)
<input checked="" type="checkbox"/> #3	1	Custom text formatting for underline working (Part of the message)
<input checked="" type="checkbox"/> #4	1	Custom text formatting for red working (Part of the message)
<input checked="" type="checkbox"/> #5	1	Custom text formatting for blue working (Part of the message)
<input checked="" type="checkbox"/> #6	1	Custom text formatting for green working (Part of the message)
<input checked="" type="checkbox"/> #7	1	Custom text formatting for combined bold, italic, underline, and color working (Part of the message)
<input checked="" type="checkbox"/> #8	1	Clearly caption screenshots

Task Screenshots:

Missing Caption

Task #3 - Points: 1
Text: Screenshot of the code solving the formatting display

Client - bob

Roxanne

Joined room: lobby
"bob connected"
"new connected"
"bob: initial bob"
"bob: hi"
"bob: hi"
"bob: ready connected"
"bob: max"
"bob: bold max"
"bob: italic max"
"bob: under max"

Send

EST. GRADE 2.00/10.00

REPORT AS MULTIPAGE

Activate Windows
Go to Settings to activate Windows

Type here to search

CHRISTIANIAN WINTONG (CW72)

10:08 PM 4/30/2024

Italic is working for this text

Checklist Items (1)

#2 Custom text formatting for italic working (Part of the message should appear italic)

Task #3 - Points: 1

Text: Screenshot of the code solving the formatting display

Checklist *The checkboxes are for your own tracking

#	Points	Details
<input type="checkbox"/> #1	1	Show each relevant file this was done in (may be one or more)
<input type="checkbox"/> #2	1	Include ucid and date comment
<input type="checkbox"/> #3	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Task Screenshots:

Gallery Style: Large View

Small Medium Large

Contains screen of relevant work done in file.

Checklist Items (3)

#1 Show each relevant file this was done in (may be one or more)

#2 Include ucid and date comment

#3 Clearly caption screenshots

Task #4 - Points: 1

Text: Explain how the formatting was made to be visible/rendered in the UI

Details:

Note each scenario

Response:

In order to make the text formatting appear in the UI. I wrapped the text in an html element and, JPanel I made sure that the textarea was on html so it was able to get the formatting.

Private Message with @ (2 pts.)

Collapse

Task #1 - Points: 1

Collapse

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Should have 3 clients in the same room
#2	1	Demo a private message where only the sender and target see the message
#3	1	Clearly caption screenshots

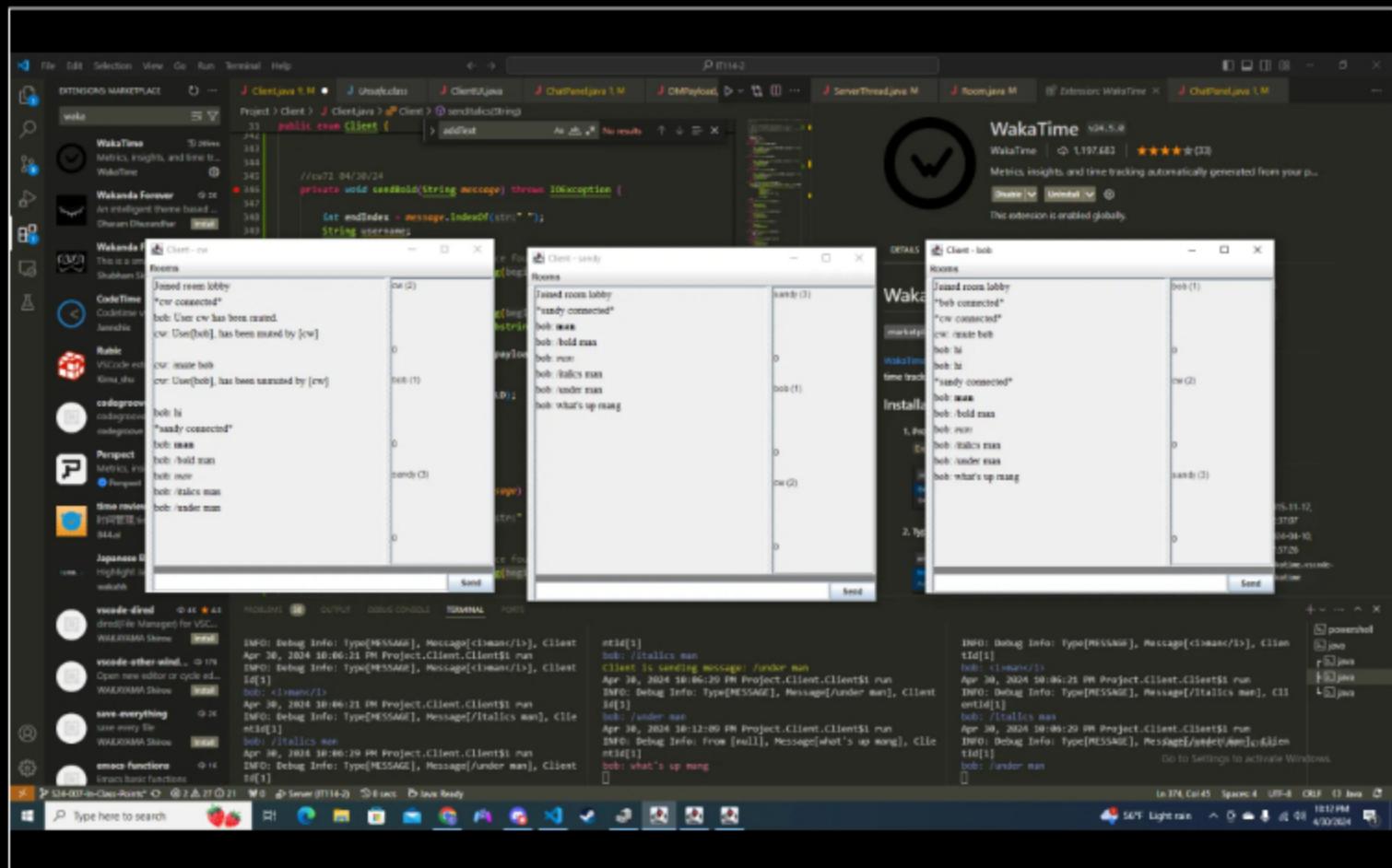
Task Screenshots:

Gallery Style: Large View

Small

Medium

Large



Shows the that there is private message functionality.

Checklist Items (3)

#1 Should have 3 clients in the same room

#2 Demo a private message where only the sender and target see the message

#3 Clearly caption screenshots

Task #2 - Points: 1

Text: Screenshots of the related code

Checklist

***The checkboxes are for your own tracking**

#	Points	Details
#1	1	Show what code processes and handles the private message
#2	1	The message should only be sent to the receiver and the target
#3	1	The client should be targeting the username and the server side should be fetching the correct recipient
#4	1	Include ucid and date comment
#5	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The top navigation bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar. Below the navigation bar are several tabs representing different files: J ClientJava 9.0.M, J UncaughtException, J ClientLog.java, J ChatThread.java 1.0.M, J CM/ayload.java M, J ... (partially visible), J ServerThread.java M, J Room.java M, J Detectors.WakeTime, and J ChatThread.java 1.0.M. The left sidebar displays a 'EXTENSIONS MARKETPLACE' section with various extensions like 'WakaTime', 'Wakanda Forever', 'CodeTime', 'Public', 'codenotes', 'Perfext', 'time review - 99%', 'Japanese Bracket HI.', 'wcode-dired', 'wcode-other-wind.', 'wcode-everything', 'amacs-Functions', and 'WAKAMIMA Status'. The main workspace contains two large code editor panes. The left pane shows a Java file with code related to sending messages and handling clients. The right pane shows another Java file with code for sending messages to rooms. A terminal window at the bottom displays log messages from the application, including entries for 'Client\$1 run', 'Client is sending message: /under man', and 'Client\$1 run'. The status bar at the bottom shows file paths like 'D:\Java\src\com\waka\chat\J ChatThread.java', 'File 1', and 'File 2', along with other system information such as 'CPU: i7-8700K', 'GPU: RTX 2080 Super', 'RAM: 16GB', and 'OS: Windows 10 Pro'. The bottom right corner shows a 'Go to Settings to activate Windows' link.

Shows the logic in the Room.java that handles the private sending message functionality.

Checklist Items (5)

#1 Show what code processes and handles the private message

#2 The message should only be sent to the receiver and the target

#3 The client should be targeting the username and the server side should be fetching the correct recipient.

#4 Include ucid and date comment

#5 Clearly caption screenshots

Part two of the screenshots that show private messaging functionality.

Checklist Items (5)

#1 Show what code processes and handles the private message

#2 The message should only be sent to the receiver and the target

#3 The client should be targeting the username and the server side should be fetching the correct recipient

#4 Include ucid and date comment

#5 Clearly caption screenshots

```

// processes where are you
    setMethodName +=" message_id"+id);
    string username;
    string previousMessage;
    if (username == null) {
        previousMessage = "No user found, or the whole string is the username";
        username = "message_username"+logId);
        previousMessage = "message_username"+id);
    }
    else {
        previousMessage = "message_username"+logId);
        previousMessage = "message_username"+id);
    }

    if (previousMessage != null) {
        previousMessage = previousMessage.substring(0, previousMessage.length() - 1);
        previousMessage += " " + message);
        previousMessage += " " + message);
    }
    else {
        previousMessage = "Client is sending message: " + message);
        previousMessage += " " + message);
    }

    System.out.println(previousMessage);
    System.out.println("Client is sending message: " + message);
}

protected synchronized void handleUserInput(Message message) {
    String logId = message.getMessageType();
    String messageContent = message.getMessageContent();
    String targetUsername = message.getTargetUsername();
    String receiverUsername = message.getReceiverUsername();
    String senderUsername = message.getSenderUsername();

    if (logId.equals("CLIENT")) {
        if (targetUsername != null) {
            if (targetUsername.equals(senderUsername)) {
                System.out.println("Client is sending message to itself");
            }
            else {
                User user = users.get(targetUsername);
                if (user != null) {
                    user.receiveMessage(message);
                }
                else {
                    System.out.println("User not found: " + targetUsername);
                }
            }
        }
        else {
            System.out.println("No target user specified");
        }
    }
    else if (logId.equals("SERVER")) {
        if (targetUsername != null) {
            if (targetUsername.equals(senderUsername)) {
                System.out.println("Client is sending message to itself");
            }
            else {
                User user = users.get(targetUsername);
                if (user != null) {
                    user.receiveMessage(message);
                }
                else {
                    System.out.println("User not found: " + targetUsername);
                }
            }
        }
        else {
            System.out.println("No target user specified");
        }
    }
}

```

Missing Caption

Checklist Items (0)

```

public class Client {
    public void sendPrivateMessage(String message) throws IOException {
        User user = getUsers().get("Client");
        user.sendMessage(message);
        System.out.println("Client is sending message: " + message);
    }
}

private void sendMessage(String message) throws IOException {
    String logId = message.getMessageType();
    String messageContent = message.getMessageContent();
    String targetUsername = message.getTargetUsername();
    String receiverUsername = message.getReceiverUsername();
    String senderUsername = message.getSenderUsername();

    if (logId.equals("CLIENT")) {
        if (targetUsername != null) {
            if (targetUsername.equals(senderUsername)) {
                System.out.println("Client is sending message to itself");
            }
            else {
                User user = users.get(targetUsername);
                if (user != null) {
                    user.receiveMessage(message);
                }
                else {
                    System.out.println("User not found: " + targetUsername);
                }
            }
        }
        else {
            System.out.println("No target user specified");
        }
    }
    else if (logId.equals("SERVER")) {
        if (targetUsername != null) {
            if (targetUsername.equals(senderUsername)) {
                System.out.println("Client is sending message to itself");
            }
            else {
                User user = users.get(targetUsername);
                if (user != null) {
                    user.receiveMessage(message);
                }
                else {
                    System.out.println("User not found: " + targetUsername);
                }
            }
        }
        else {
            System.out.println("No target user specified");
        }
    }
}

```

Logic in the client that explains how to send the private messages to the ServerThread

Checklist Items (5)

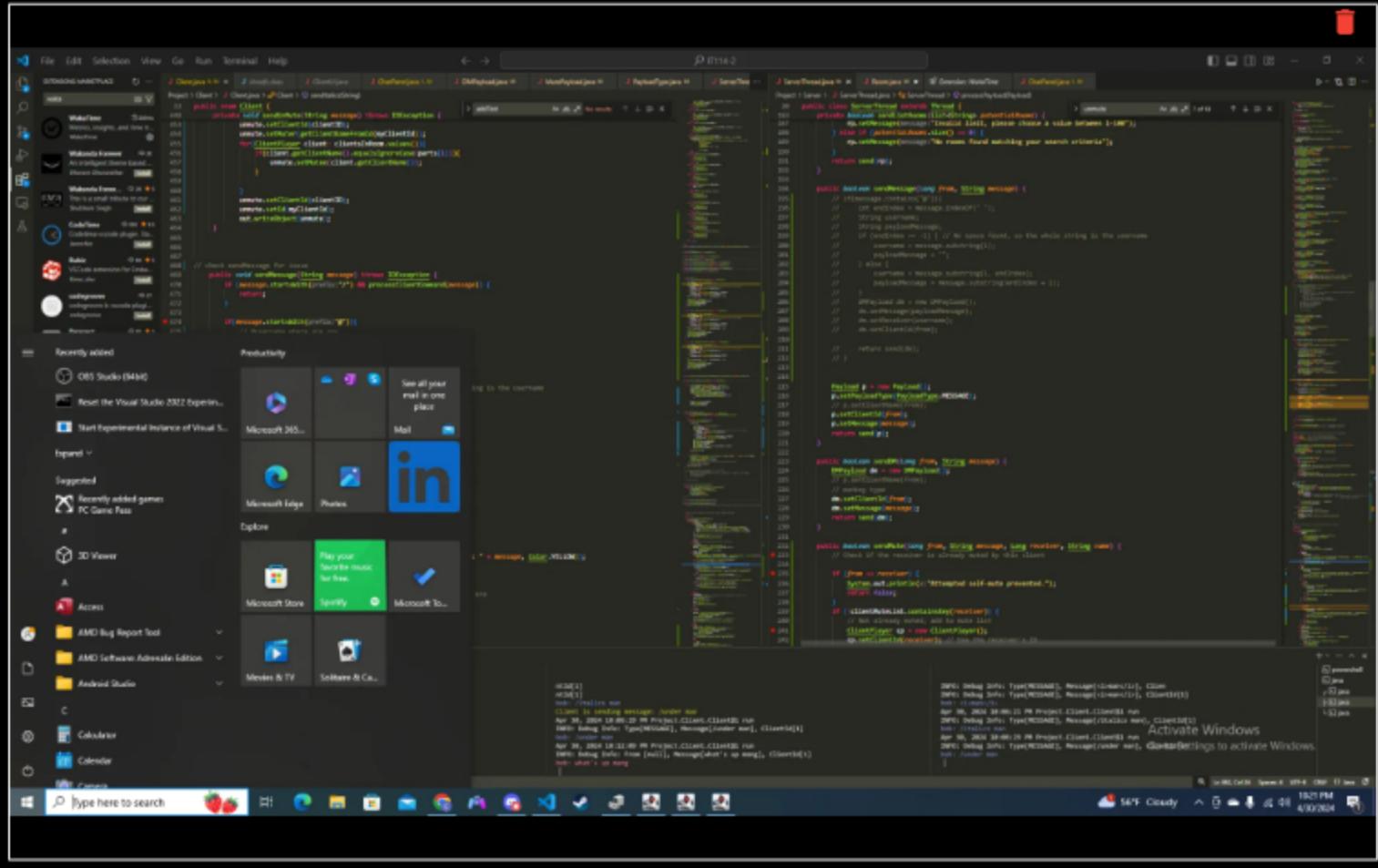
#1 Show what code processes and handles the private message

#2 The message should only be sent to the receiver and the target

#3 The client should be targeting the username and the server side should be fetching the correct recipient

#4 Include uid and date comment

#5 Clearly caption screenshots



Shows the function being called in the ServerThread.java a

Checklist Items (5)

#1 Show what code processes and handles the private message

#2 The message should only be sent to the receiver and the target

#3 The client should be targeting the username and the server side should be fetching the correct recipient

#4 Include ucid and date comment

#5 Clearly caption screenshots

Task #3 - Points: 1

Text: Explain how private message works related to the code above

Checklist

*The checkboxes are for your own tracking

<input checked="" type="checkbox"/> #1	1	Include how the sender and receiver are handled
<input checked="" type="checkbox"/> #2	1	Include how the username is used to get the proper id

Response:

It was a very complicated process. But essentially a payload was sent with PayloadType DM. That payload was sent from Client.java ui, and it checks for "@" symbol. In ServerThread it processes the payload and calls the Room.java sendDM function, there I take information from I extract the client ID from which was set in the Client.java sendMessage function by looping through the clientsInRoom and I check if the receiver is comparing client.getClientId() to the receiver, if it is I then send the message to that client.

Mute/Unmute Users (3 pts.)

[COLLAPSE](#)

Task #1 - Points: 1

Text: Screenshots demoing feature working

Checklist

*The checkboxes are for your own tracking

#	Points	Details
<input checked="" type="checkbox"/> #1	1	Should have 3 clients in the same room
<input checked="" type="checkbox"/> #2	1	Demo mute preventing messages between the muter and the target
<input checked="" type="checkbox"/> #3	1	Demo mute also being accounted for with private messages
<input checked="" type="checkbox"/> #4	1	Demo unmute allowing the messages again from the target to the unmuter

Task Screenshots:

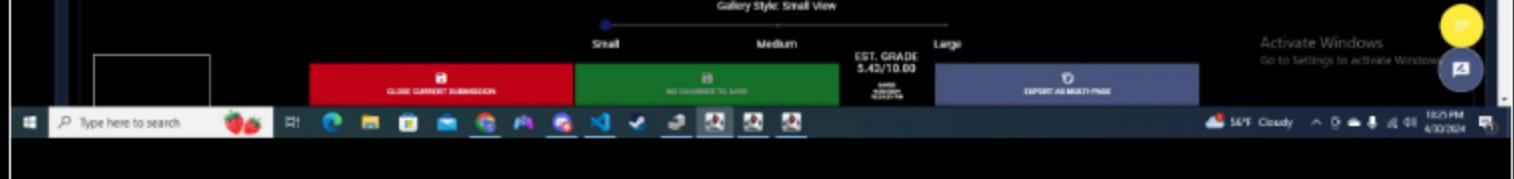
Gallery Style: Large View

Small

Medium

Large

<input checked="" type="checkbox"/> #1	1	Room should handle the mute list when receiving the appropriate payloads
<input checked="" type="checkbox"/> #2	1	Room should check the mute list during send message and private messages
<input checked="" type="checkbox"/> #3	1	include user and date comment
<input checked="" type="checkbox"/> #4	1	Clearly explain screenshots



Shows the mute and unmute function being used.

Checklist Items (4)

#1 Should have 3 clients in the same room

#2 Demo mute preventing messages between the muter and the target

#3 Demo mute also being accounted for with private messages

#4 Demo unmute allowing the messages again from the target to the unmuter

Task #2 - Points: 1

Text: Screenshots of the related code

Checklist

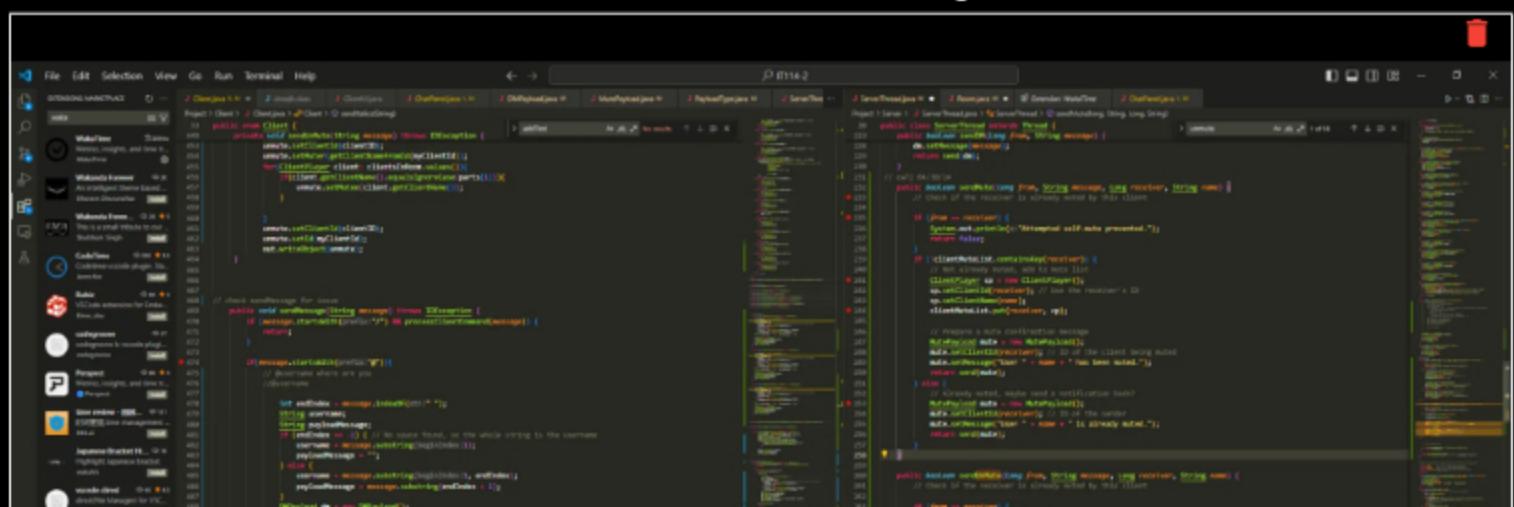
*The checkboxes are for your own tracking

#	Points	Details
#1	1	ServerThread should have a list of who they muted
#2	1	ServerThread should expose and add, remove, and is muted check to room
#3	1	Room should handle the mute list when receiving the appropriate payloads
#4	1	Room should check the mute list during send message and private messages
#5	1	Include ucid and date comment
#6	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large



```
private void send(string message)
{
    var client = new Client();
    client.Send(message);
}

private void receive()
{
    var client = new Client();
    string message = client.Receive();
}

private void StartListening()
{
    var server = new Server();
    server.StartListening();
}

private void StopListening()
{
    var server = new Server();
    server.StopListening();
}
```

Shows the list of clients that are muted, How that room should handle payloads.

Checklist Items (0)

A screenshot of a Windows 10 desktop with two Microsoft Visual Studio windows open side-by-side. The left window displays the 'Client' project, which includes Java code for sending and receiving messages via a socket connection. The right window displays the 'Server' project, which includes Java code for managing client connections and processing received messages. Both windows show multiple tabs and code editors. The taskbar at the bottom features the Start button, a search bar, pinned icons for File Explorer, File History, Task View, and Edge, and a weather widget indicating '16° Cloudy'. The system tray shows battery level, signal strength, and volume.

Continuation of the first screenshot.

Checklist Items (6)

#1 ServerThread should have a list of who they muted

#2 ServerThread should expose and add, remove, and is muted check to room

#3 Room should handle the mute list when receiving the appropriate payloads

#4 Room should check the mute list during send message and private messages

#5 Include uid and date comment

#6 Clearly caption screenshots

Task #3 - Points: 1

Text: Explain how the mute and unmute logic works in relation to the code

Checklist

*The checkboxes are for your own tracking

#	Points	Details
<input checked="" type="checkbox"/> #1	1	Explain how your mute list is handled
<input checked="" type="checkbox"/> #2	1	Explain how it's handled/processed in send message and private message

Response:

There was a command trigger in the client, in the client function both the muter and mutee clientId was added to the payload. Then in server thread, there was new list called clientsInMuteList and I added functionality to check if the clientID was in the list, if not I added the client Id, as well as the client name to the clientPlayer object in the map. Then when In Room.java, in the send message function, I iterated through the send message function and then iterated through the clientsInMuteList for each client and check if the sender was in the mute list and if so I just continued. For the unmute function it was similar logic but in the ServerThread I check if the client ID was in the mute list and if so I remove the key and the value from the map.

Misc (1 pt.)

ACOLLAPSE

Task #1 - Points: 1

Text: Add the pull request link for the branch

1 Details:

Note: the link should end with /pull/#

URL #1

<https://github.com/cdubbx/IT114/pull/1>

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

I believe the assignment was pretty robust, I feel like if I began the assignment back in February it would've given me and other people more time to complete the assignment. I believe the there was way too much to be able to understand. It is pretty realistic because if you want to become a program you need learn on how to adapt to a code.

understand. It is pretty realistic because if you want to become a program you need learn on how to adapt to a code base.

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.

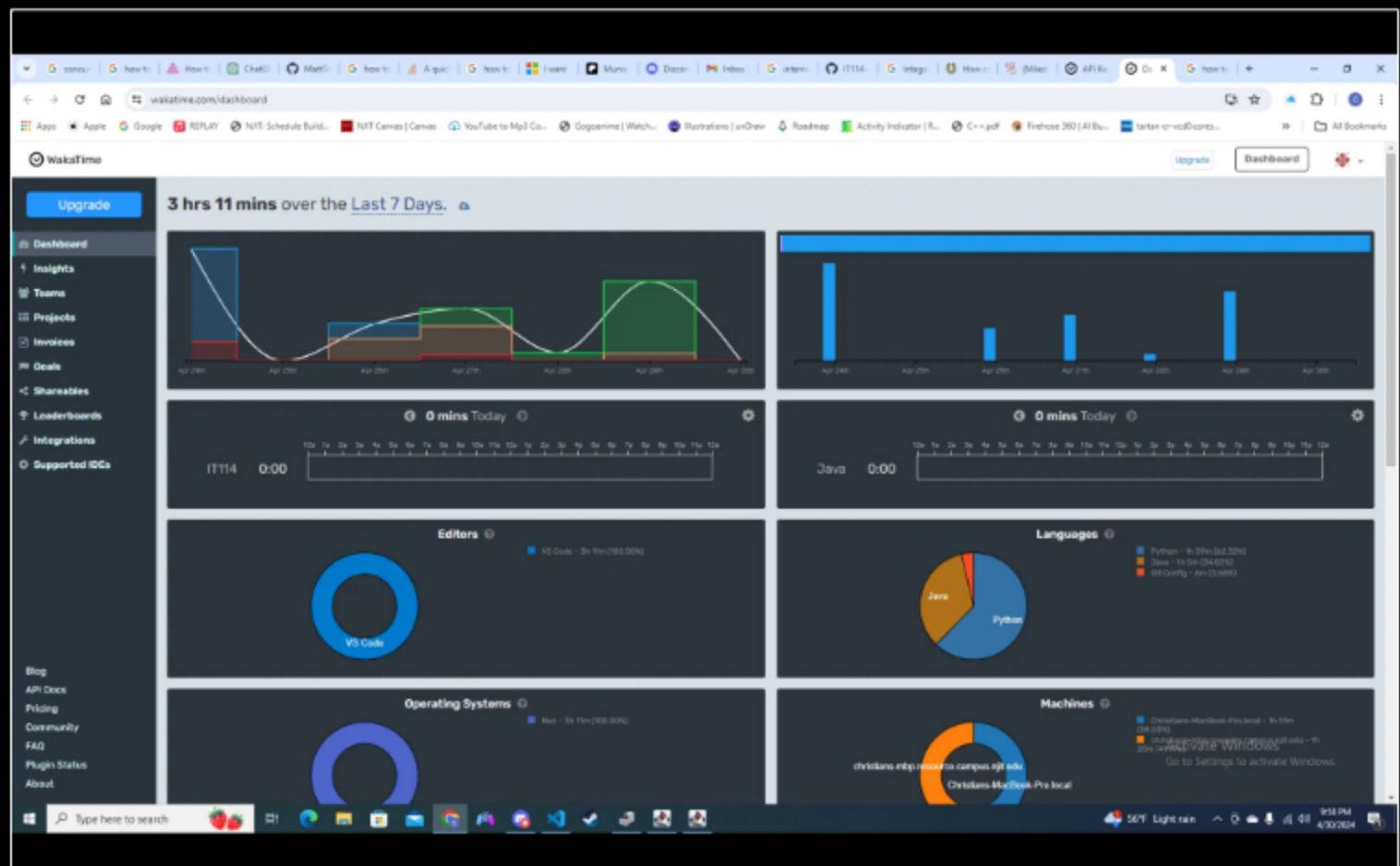
Task Screenshots:

Gallery Style: Large View

Small

Medium

Large



It is not an accurate representation because I did not have waka time on this computer and I have two computers.

End of Assignment