Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-451-M2024/it114-milestone-2-chatroom-2024-m24/grade/st278

IT114-451-M2024 - [IT114] Milestone 2 Chatroom 2024 (M24)

Submissions:

Submission Selection

1 Submission [active] 7/8/2024 11:59:11 PM

Instructions

COLLAPSE

1. Implement the Milestone 2 features from the project's proposal document: https://docs.google.com/document/d/1ONmvEvel97GTFPGfVwwQC96xSsobbSbk56145XizQG4/view
2. Make sure you add your ucid/date as code comments where code changes are done
3. All code changes should reach the Milestone2 branch
4. Create a pull request from Milestone2 to main and keep it open until you get the output PDF from this assignment.
5. Gather the evidence of feature completion based on the below tasks.
6. Once finished, get the output PDF and copy/move it to your repository folder on your local machine.
7. Run the necessary git add, commit, and push steps to move it to GitHub
8. Complete the pull request that was opened earlier
9. Upload the same output PDF to Canvas

Branch name: Milestone2

Tasks: 8 Points: 10.00

Payloads (2 pts.)

COLLAPSE

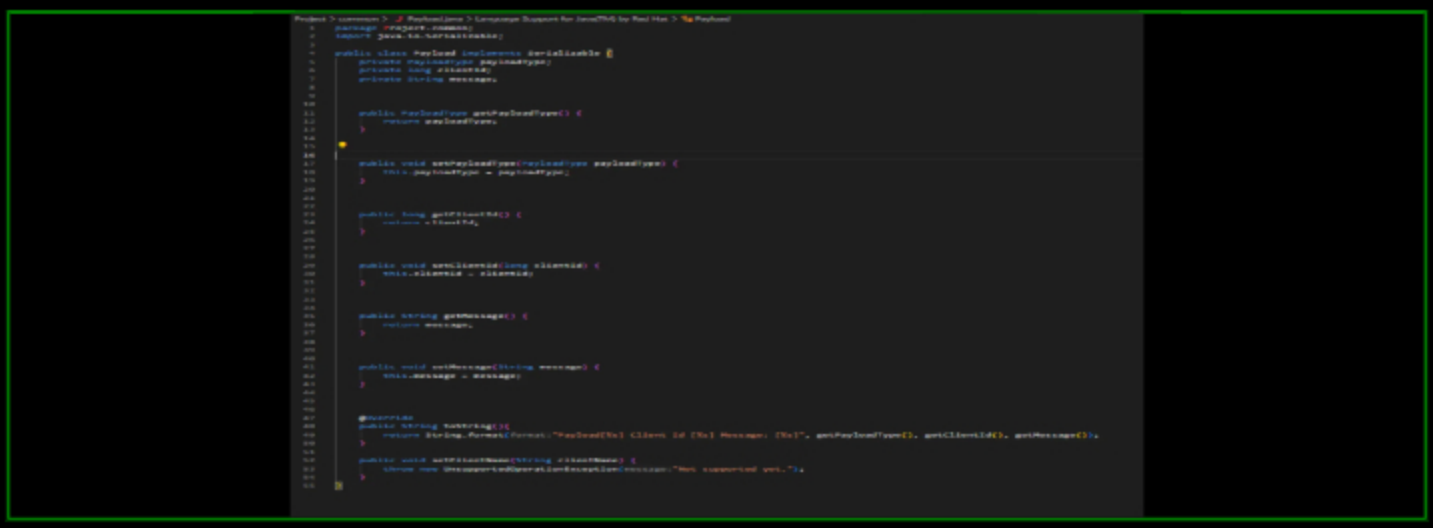
COLLAPSE

Task #1 - Points: 1

Text: Base Payload Class

ℹ️Details:
All code screenshots must have ucid/date visible.

#1) Show screenshot of the Payload.java



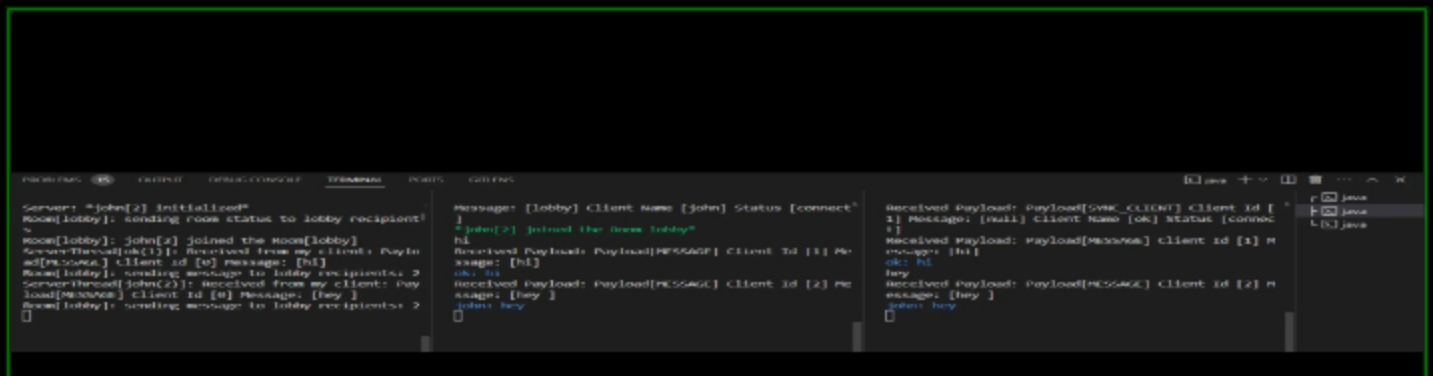
Caption (required) ✓
Describe/highlight what's being shown
Showing screenshot of the Payload.java

Explanation (required) ✓
Briefly explain the purpose of each property and serialization

PREVIEW RESPONSE

The Payload class has three main properties: payloadType to identify the kind of message, clientId to know which client sent it, and message for the actual text content. Serialization allows this object to be turned into a byte stream, making it easy to send over a network or save for later use. This helps in sending detailed messages between clients and servers efficiently.

#2) Show screenshot examples of the terminal output for base Payload objects



Caption (required) ✓
Describe/highlight what's being shown
Base payload options working and giving output.

COLLAPSE

Task #2 - Points: 1

Text: RollPayload Class

Details:
All code screenshots must have ucid/date visible.

#1) Show screenshot of the RollPayload.java (or equivalent)

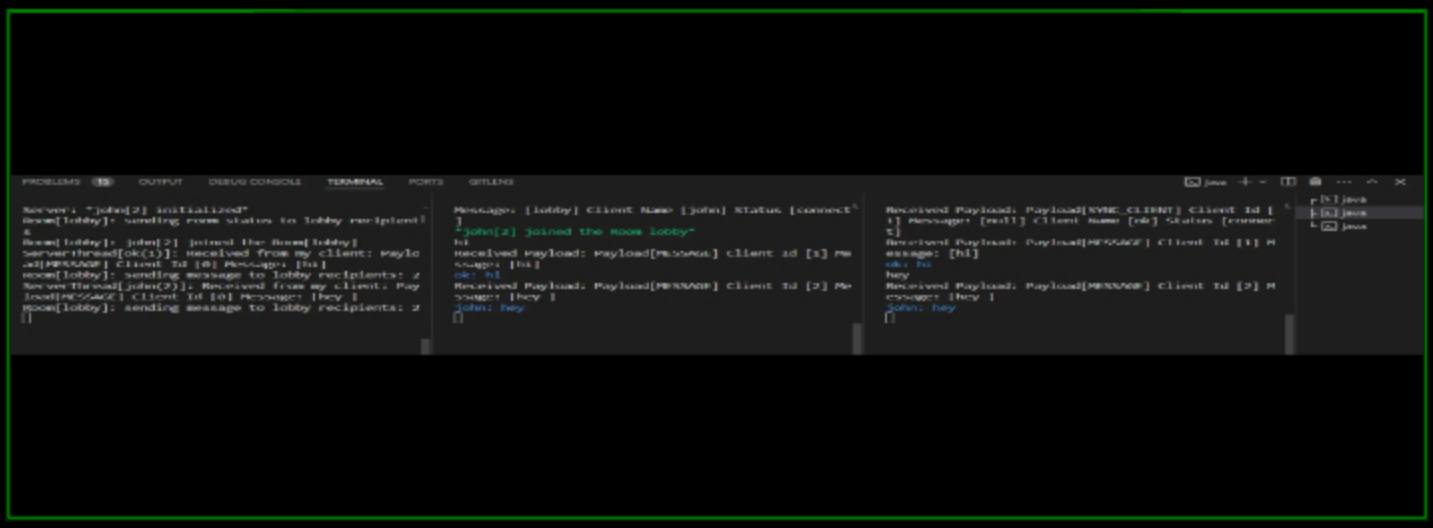
Caption (required) ✓
Describe/highlight what's being shown
Showing screenshot of equivalent of the RollPayload.java

Explanation (required) ✓
Briefly explain the purpose of each property

PREVIEW RESPONSE

This class extends the Payload class and contains the necessary information for a roll command, including the client name, the type of roll, and the result. The processPayload method includes logic to handle the payload, such as printing the result to the console. You can expand the processPayload method to include other functionalities as needed, such as sending the result back to the client or updating the server state.

#2) Show screenshot examples of the terminal output for base RollPayload objects



Caption (required) ✓

Describe/highlight what's being shown

Showing screenshot examples of the terminal output for base RollPayload objects

Client Commands (4 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Roll Command

Details:

All code screenshots must have ucid/date visible.
Any output screenshots must have at least 3 connected clients able to see the output.
All commands must show who triggered it, what they did (specifically) and what the outcome was.

#1) Show the client side code for handling /roll #

```
//st278 and 07/08/2024
        else {
            // Format: #
            rp.setRollType(rollType:"single");
            rp.setMax(Integer.parseInt(command));
        }
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the client side code for handling /roll #

Explanation (required) ✓

Briefly explain the logic

PREVIEW RESPONSE

The code checks if the message contains the command "roll". If the message is just a number, it rolls a single die with that many sides. The results are then sent back to the sender. If there's an error in parsing, it sends an "invalid input" message.

#2) Show the output of a few examples of /roll # (related payload output should be visible)

```
/roll 19
Received Payload: Payload[MESSAGE] Client Id [-1] M
essage: [ok has rolled 19 and got 5]
Room: ok has rolled 19 and got 5
/roll 28
Received Payload: Payload[MESSAGE] Client Id [-1] M
essage: [ok has rolled 28 and got 6]
Room: ok has rolled 28 and got 6
/roll 50
Received Payload: Payload[MESSAGE] Client Id [-1] M
essage: [ok has rolled 50 and got 41]
Room: ok has rolled 50 and got 41
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the output of a few examples of /roll #

#3) Show the client side code for handling /roll #d# (related payload output should be visible)

```
//st278 and 07/08/2024
private void processRollCommand(String command) {
    LoggerUtil.INSTANCE.info("Processing roll command: " + command);
    RollPayload rp = new RollPayload();
    rp.setClientName(myData.getClientName());

    try {
        if (command.contains(s:"d")) {
            // Format: #d#
```

```
            String[] parts = command.split(regex:"d");
            rp.setRollType(rollType:"multiple");
            rp.setNumDice(Integer.parseInt(parts[0]));
            rp.setSides(Integer.parseInt(parts[1]));
        }
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the client side code for handling /roll #d#

Explanation (required) ✓

Briefly explain the logic

PREVIEW RESPONSE

The code checks if the message contains the command "roll". If it does, it initializes a total variable to accumulate the sum of dice rolls. It then tries to parse the command to determine the number of dice and sides per die (format XdY). If successful, it rolls the dice the specified number of times and sums the results.

#4) Show the output of a few examples of /roll #d#

```
/roll 2d6
Received Payload: Payload[MESSAGE] Client Id [-1]
Message: [john has rolled 2d6 and got 7]
Room: john has rolled 2d6 and got 7
/roll 3d9
Received Payload: Payload[MESSAGE] Client Id [-1]
Message: [john has rolled 3d9 and got 12]
Room: john has rolled 3d9 and got 12
/roll 5d20
Received Payload: Payload[MESSAGE] Client Id [-1]
Message: [john has rolled 5d20 and got 28]
Room: john has rolled 5d20 and got 28
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the output of a few examples of /roll #d#

#5) Show the ServerThread code receiving the RollPayload

```
//st278 and 07/08/2024
case ROLL_COMMAND:
```

```
        LoggerUtil.INSTANCE.info("Received ROLL payload: " + payload);
        currentRoom.handleRoll(this, (RollPayload) payload);
        break;
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the ServerThread code receiving the RollPayload

Explanation (required) ✓

Briefly explain the logic

PREVIEW RESPONSE

The ServerThread class processes different types of payloads received from the client. For roll commands, it identifies the payload type as ROLL, casts it to RollPayload, and calls the handleRoll method in the current room to handle the roll logic. This ensures that the command is processed correctly, and the result is sent back to the client.

#6) Show the Room code that processes both Rolls and sends the response

```
//s1278 and 07/08/2024
protected void handleRoll(ServerThread sender, RollPayload payload) {
    LoggerUtil.INSTANCE.info("Room handling roll command: " + payload);
    int result;
    String message;

    if (payload.getRollType().equals(anObject:"single")) {
        result = (int) (Math.random() * payload.getMax()) + 1;
        message = String.format(format:"%s rolled %d and got %d", payload.getClientId(), payload.getMax(), result);
    } else {
        result = 0;
        for (int i = 0; i < payload.getNumDice(); i++) {
            result += (int) (Math.random() * payload.getSides()) + 1;
        }
        message = String.format(format:"%s rolled %dd%d and got %d", payload.getClientId(), payload.getNumDice(), payload.getSides(), result);
    }

    LoggerUtil.INSTANCE.info("Sending roll result: " + message);
    sendMessage(sender, message);
}
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the Room code that processes both Rolls and sends the response

Explanation (required) ✓

Briefly explain the logic

PREVIEW RESPONSE

This code handles special chat commands from clients. When a message starts with "/", it checks for two commands: /flip and /roll. If the command is /flip, it simulates a coin toss and sends a message indicating heads or tails. If the command is /roll, it can handle two formats: /roll # rolls a single die with a specified number of sides, and /roll #d# rolls multiple dice with specified sides. It then sends a message showing the result of the roll. If there's an error in the input, it sends an "invalid input" message.

an error in the input, it sends an "invalid input" message.

^COLLAPSE^

Task #2 - Points: 1

Text: Flip Command

#1) Show the client side code for handling /flip

```java
//st278 and 07/08/2024
private void processFlipCommand() {
    LoggerUtil.INSTANCE.info(message:"Processing flip command");
    Payload p = new Payload();
    p.setPayloadType(PayloadType.FLIP_COMMAND);
    p.setClientName(myData.getClientName());
    LoggerUtil.INSTANCE.info("Sending flip payload: " + p);
    send(p);
}
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the client side code for handling /flip

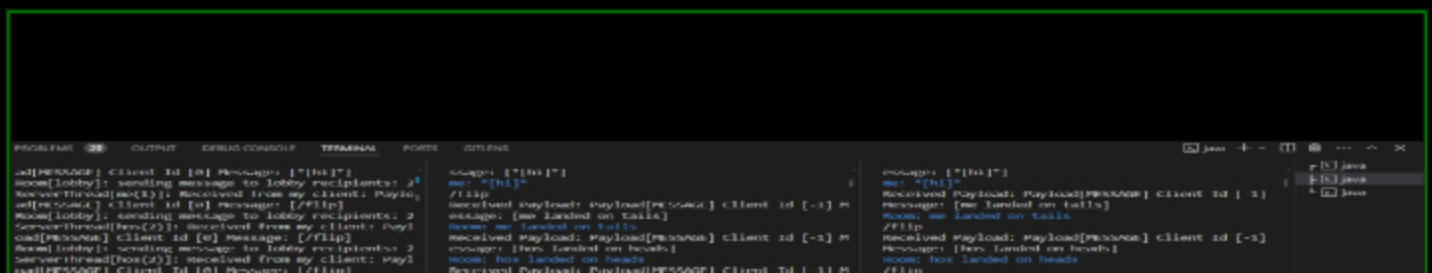
Explanation (required) ✓

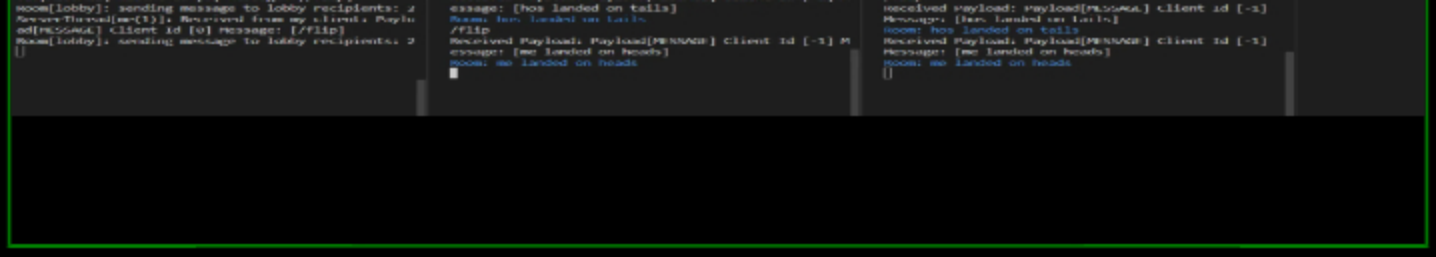
Briefly explain the logic

PREVIEW RESPONSE

This code snippet checks if a chat message contains a specific command triggered by a forward slash (/). If the command is /flip, it proceeds to simulate a coin flip by generating a random integer (either 0 or 1). Depending on the random outcome, it sends a message announcing whether the coin landed on "heads" or "tails," attributing the result to the sender's name. This functionality allows users to interact with the chat system by invoking simple commands for random outcomes, enhancing user engagement and interaction within the chat environment.

#2) Show the output of a few examples of /flip (related payload output should be visible)





Caption (required) ✓

Describe/highlight what's being shown

Showing the output of a few examples of /flip

Text Formatting (3 pts.)

^COLLAPSE ^

^COLLAPSE ^

Task #1 - Points: 1

Text: Text Formatting

ℹ Details:

All code screenshots must have ucid/date visible.
Any output screenshots must have at least 3 connected clients able to see the output.
Note: Having the user type out html tags is not valid for this feature, instead treat it like WhatsApp, Discord, Markdown, etc
Note: Each text trigger must wrap the text that you want to affect
Note: Slash commands are not an accepted solution, the text must be transformed
Note: You do not need to use the same symbols in the below example, it's just an example, also, the below example doesn't show the "correct" output for colors, I'm leaving the proper conversion up to research on your own.
See proposal for an example.

#1) Show the code related to processing the special characters for bold, italic, underline, and colors, and converting them to other characters (should be in Room.java)

Caption (required) ✓

Describe/highlight what's being shown

Showing the code related to processing the special characters for bold, italic, underline, and colors, and converting them to ot

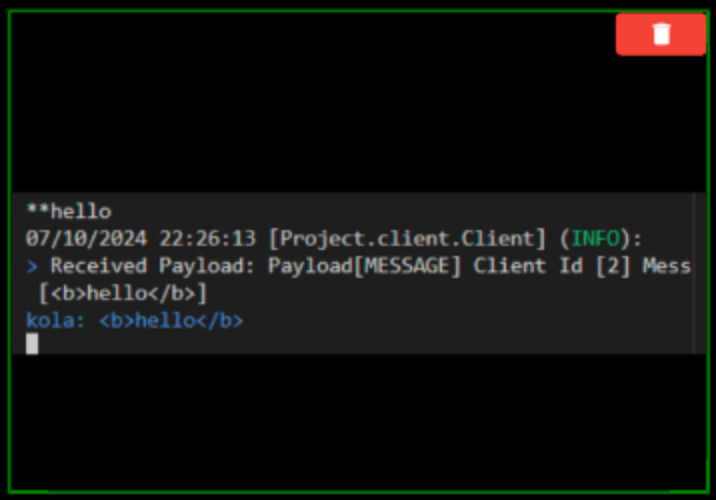
Explanation (required) ✓

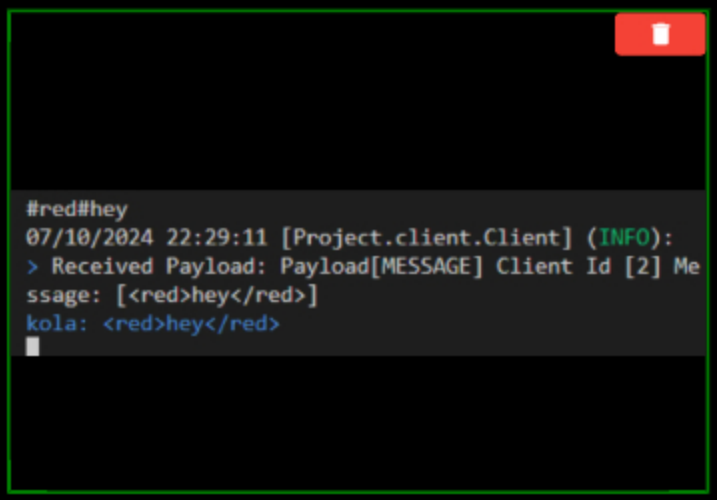
Briefly explain how it works and the choices of the placeholder characters and the result characters

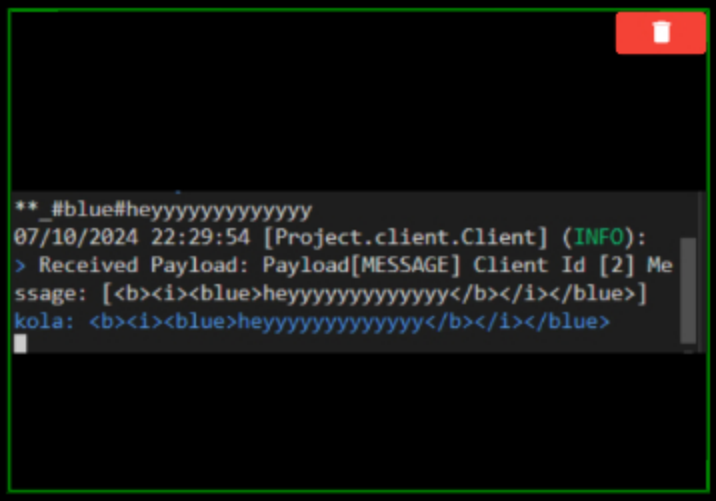
PREVIEW RESPONSE

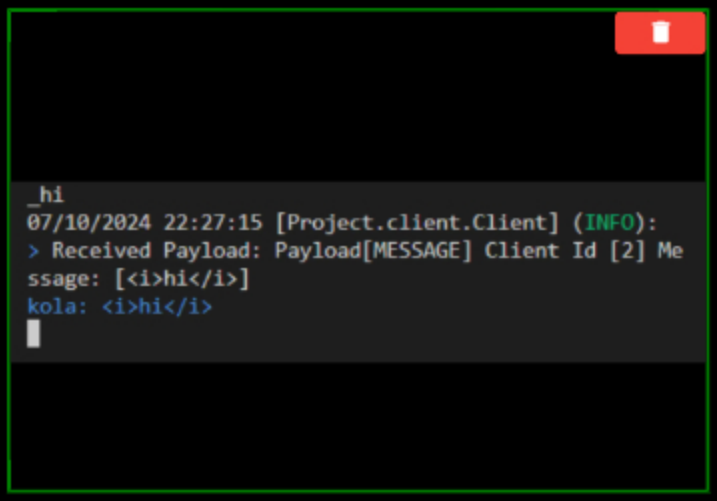
This code transforms a string containing custom formatting symbols into HTML-formatted text. It scans through the input string, looking for special symbols that represent bold, italic, underline, and color formatting. When it finds these symbols, it replaces them with the appropriate HTML tags (like , , ***and color tags). The code keeps track of which formatting is currently active, ensuring that tags are properly opened and closed. At the end, it makes sure all opened tags are closed, resulting in valid HTML that can be displayed with the intended formatting.***

#2) Show examples of each: bold, italic, underline, colors (red, green, blue), and combination of bold, italic, underline and a color









Caption (required) ✓

Describe/highlight what's being shown

Showing examples of each: bold, italic, underline, colors, and combination of bold, italic, underline, and a color

Misc (1 pt.)

COLLAPSE

EXPAND

Task #1 - Points: 1

Text: Add the pull request link for the branch

EXPAND

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

EXPAND

Task #3 - Points: 1

Text: WakaTime Screenshot

End of Assignment