Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-005-F2024/it114-milestone-2-chatroom-2024-m24/grade/kr553

Course: IT114-005-F2024
Assigment: [IT114] Milestone 2 Chatroom 2024 (M24)
Student: Kush R. (kr553)

Submissions:

Submission Selection

1 Submission [submitted] 11/9/2024 11:16:22 AM

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

Group

100%

Group: Payloads
Tasks: 2
Points: 2

COLLAPSE

Task

100%

Group: Payloads
Task #1: Base Payload Class
Weight: ~50%
Points: ~1.00

COLLAPSE

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

Columns: 1

Sub-Task

100%

Group: Payloads
Task #1: Base Payload Class
Sub Task #1: Show screenshot of the Payload.java

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Payload.java

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the purpose of each property and serialization

Response:

Payload is a package of information shared between players and the server. It has details like the message, who sent it, the time, and what kind of message it is. Serialization helps send this package easily over the internet or save it for later use.

Sub-Task

100%

Group: Payloads
Task #1: Base Payload Class
Sub Task #2: Show screenshot examples of the terminal output for base Payload objects

Task Screenshots

Gallery Style: 2 Columns

4 2 1

here is the terminal output for payload object

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

End of Task 1

Task

100%

Group: Payloads
Task #2: RollPayload Class
Weight: ~50%
Points: ~1.00

COLLAPSE

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

Columns: 1

Sub-Task

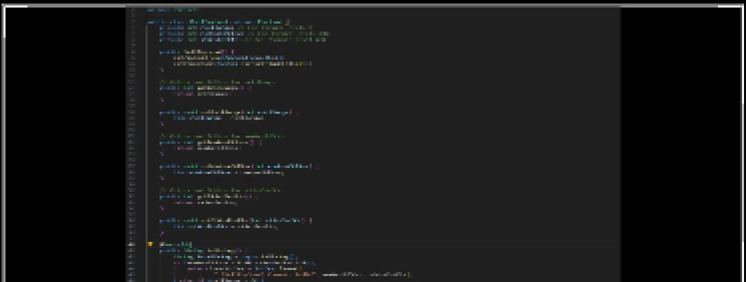
100%

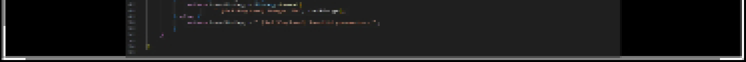
Group: Payloads
Task #2: RollPayload Class
Sub Task #1: Show screenshot of the RollPayload.java (or equivalent)

Task Screenshots

Gallery Style: 2 Columns

4 2 1




RollPayload.java

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the purpose of each property

Response:

`rollRange` sets the limit for a single roll. `numberOfDice` tells how many dice to roll, while `sidesPerDie` shows how many sides each die has. These properties let players roll either a single number or multiple dice.

Sub-Task

100%

Group: Payloads

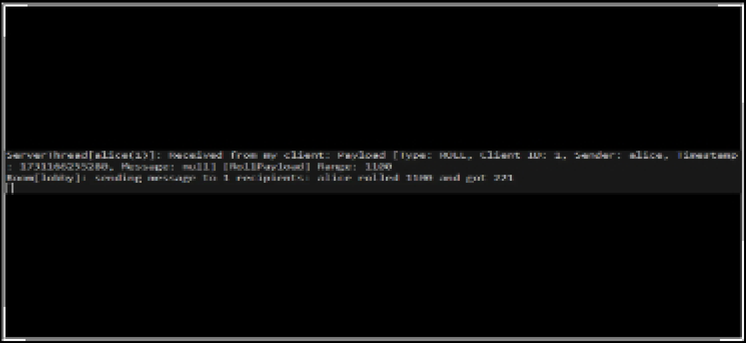
Task #2: RollPayload Class

Sub Task #2: Show screenshot examples of the terminal output for base RollPayload objects

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
ServerThread[alice(1)]: Received from my client: Payload [type: ROLL, Client ID: 1, Sender: alice, Timestamp
: 1731106235200, Message: null] [RollPayload] Range: 1100
Room[lobby]: sending message to 1 recipients: alice rolled 1100 and got 221
```

here is example output for rollpayload

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

End of Task 2

End of Group: Payloads

Task Status: 2/2

Group

100%

Group: Client Commands

Tasks: 2

Points: 4

COLLAPSE

Task

Group: Client Commands

100%

Group: Client Commands
Task #1: Roll Command
Weight: ~50%
Points: ~2.00

COLLAPSE

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.

Columns: 1

Sub-Task

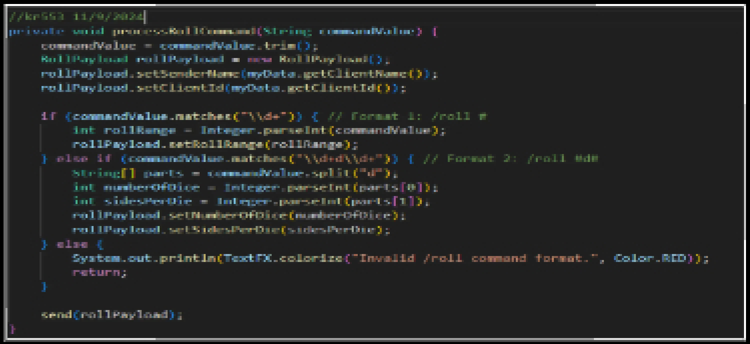
100%

Group: Client Commands
Task #1: Roll Command
Sub Task #1: Show the client side code for handling /roll #

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
//kr553 11/9/2024
private void processRollCommand(String commandValue) {
    commandValue = commandValue.trim();
    RollPayload rollPayload = new RollPayload();
    rollPayload.setSenderName(myData.getClientName());
    rollPayload.setClientId(myData.getClientId());

    if (commandValue.matches("\\d+")) { // Format 1: /roll #
        int rollRange = Integer.parseInt(commandValue);
        rollPayload.setRollRange(rollRange);
    } else if (commandValue.matches("\\d+d\\d+")) { // Format 2: /roll #d#
        String[] parts = commandValue.split("d");
        int numberOfDice = Integer.parseInt(parts[0]);
        int sidesPerDie = Integer.parseInt(parts[1]);
        rollPayload.setNumberOfDice(numberOfDice);
        rollPayload.setSidesPerDie(sidesPerDie);
    } else {
        System.out.println(TextFX.colorize("Invalid /roll command format.", Color.RED));
        return;
    }

    send(rollPayload);
}
```

ProcessRollCommand()

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the logic

Response:

The processRollCommand method handles /roll commands. It checks if the command is to roll a single number (/roll #) or multiple dice (/roll #d#). It then creates a RollPayload with the relevant details, like the range or number of dice, and sends it to the server. Invalid commands show an error message.

Sub-Task

100%

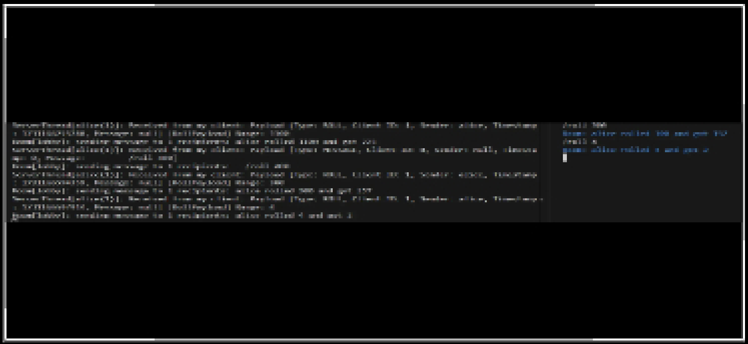
Group: Client Commands
Task #1: Roll Command
Sub Task #2: Show the output of a few examples of /roll # (related payload output should be visible)

Task Screenshots

Gallery Style: 2 Columns

Gallery Style: 2 Columns

4 2 1



```
ServerThread[alice(1)]: Received from my client: Payload [Type: ROLL, Client ID: 1, Sender: alice, Timestamp
: [illegible], Message: null] [RollPayload] Range: 1000
Room[lobby]: sending message to 1 recipients: alice rolled 1000 and got [illegible]
ServerThread[alice(1)]: Received from my client: Payload [Type: MESSAGE, Client ID: 0, Sender: null, Timesta
mp: 0, Message:          /roll 300]
Room[lobby]: sending message to 1 recipients:     /roll 300
ServerThread[alice(1)]: Received from my client: Payload [Type: ROLL, Client ID: 1, Sender: alice, Timestamp
: [illegible], Message: null] [RollPayload] Range: 300
Room[lobby]: sending message to 1 recipients: alice rolled 300 and got 1??
ServerThread[alice(1)]: Received from my client: Payload [Type: ROLL, Client ID: 1, Sender: alice, Timestamp
: [illegible], Message: null] [RollPayload] Range: 4
Room[lobby]: sending message to 1 recipients: alice rolled 4 and got 2
```

```
/roll 300
Room: alice rolled 300 and got 1??
/roll 4
Room: alice rolled 4 and got 2
```

here is the roll commands with the payloads format 1 is regular rolling and 2 is dice

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

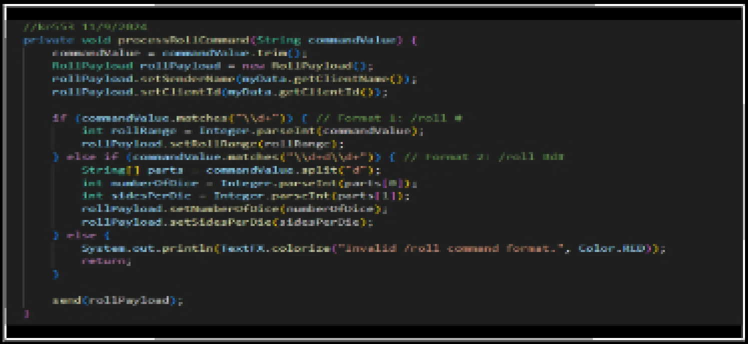
Sub-Task 100%

Group: Client Commands
Task #1: Roll Command
Sub Task #3: Show the client side code for handling /roll #d# (related payload output should be visible)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```java
//kr554 11/9/2024
private void processRollCommand(String commandValue) {
    commandValue = commandValue.trim();
    RollPayload rollPayload = new RollPayload();
    rollPayload.setSenderName(myData.getClientName());
    rollPayload.setClientId(myData.getClientId());

    if (commandValue.matches("\\d+")) { // Format 1: /roll #
        int rollRange = Integer.parseInt(commandValue);
        rollPayload.setRollRange(rollRange);
    } else if (commandValue.matches("\\d+d\\d+")) { // Format 2: /roll #d#
        String[] parts = commandValue.split("d");
        int numberOfDice = Integer.parseInt(parts[0]);
        int sidesPerDie = Integer.parseInt(parts[1]);
        rollPayload.setNumberOfDice(numberOfDice);
        rollPayload.setSidesPerDie(sidesPerDie);
    } else {
        System.out.println(TextFX.colorize("Invalid /roll command format.", Color.RED));
        return;
    }

    send(rollPayload);
}
```

here is code for payloadRollcommand again with other Format 2 of roll ing a dice

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the logic

Response:

The processRollCommand method handles /roll commands. It checks if the command is to roll a single number (/roll #) or multiple dice (/roll #d#). It then creates a RollPayload with the relevant details, like the range or number of dice, and sends it to the server. Invalid commands show an error message.

Sub-Task 100%

Group: Client Commands
Task #1: Roll Command
Sub Task #4: Show the output of a few examples of /roll #d#

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
[illegible]
```

example of rolling a dice

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

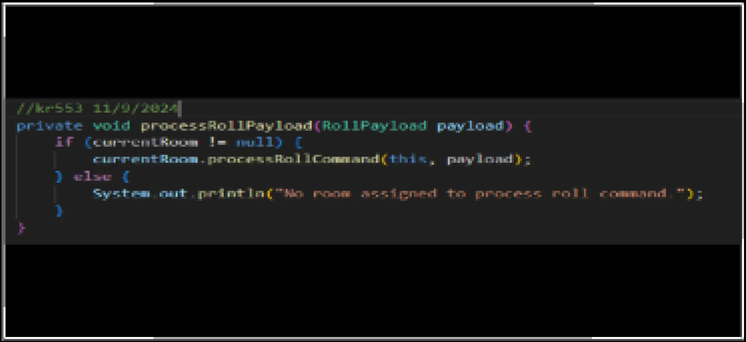
Sub-Task 100%

Group: Client Commands
Task #1: Roll Command
Sub Task #5: Show the ServerThread code receiving the RollPayload

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
//kr553 11/9/2024
private void processRollPayload(RollPayload payload) {
    if (currentRoom != null) {
        currentRoom.processRollCommand(this, payload);
    } else {
        System.out.println("No room assigned to process roll command.");
    }
}
```

Rollpayload in ServerThread

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the logic

Response:

The method checks if the client is in a room. If yes, it processes the roll command for that room; if not, it displays an error saying there's no room assigned.

Sub-Task 100%

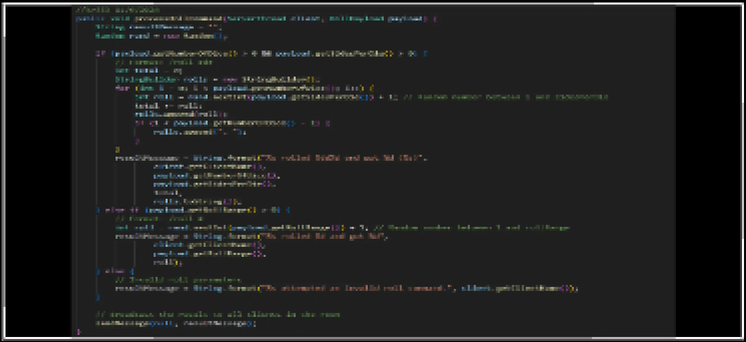
Group: Client Commands
Task #1: Roll Command

Sub Task #6: Show the Room code that processes both Rolls and sends the response

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
//[illegible]
public void processRollCommand(ServerThread client, RollPayload payload) {
    String resultMessage = "";
    Random rand = new Random();

    if (payload.getNumberOfDice() > 0 && payload.getSidesPerDie() > 0) {
        // Format: /roll #d#
        int total = 0;
        StringBuilder rolls = new StringBuilder();
        for (int i = 0; i < payload.getNumberOfDice(); i++) {
            int roll = rand.nextInt(payload.getSidesPerDie()) + 1; // Random number between 1 and [illegible]
            total += roll;
            rolls.append(roll);
            if (i < payload.getNumberOfDice() - 1) {
                rolls.append(", ");
            }
        }
        resultMessage = String.format("%s rolled %dd%d and got %d (%s)",
                client.getClientName(),
                payload.getNumberOfDice(),
                payload.getSidesPerDie(),
                total,
                rolls.toString());
    } else if (payload.getRollRange() > 0) {
        // Format: /roll #
        int roll = rand.nextInt(payload.getRollRange()) + 1; // Random number between 1 and rollRange
        resultMessage = String.format("%s rolled %d and got %d",
                client.getClientName(),
                payload.getRollRange(),
                roll);
    } else {
        // Invalid roll parameters
        resultMessage = String.format("%s attempted an invalid roll command.", client.getClientName());
    }

    // broadcast the result to all clients in the room
    sendMessage(null, resultMessage);
}
```

here is the code for room.java

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the logic

Response:

This method handles rolling dice commands. It checks if the user wants multiple dice (e.g., "2d6") or a simple range (e.g., "1-100"). It generates random rolls accordingly, sums the results, and creates a message. Finally, it sends the result to everyone in the chat room.

End of Task 1

Task

100%

Group: Client Commands
Task #2: Flip Command
Weight: ~50%
Points: ~2.00

COLLAPSE

Columns: 1

Sub-Task

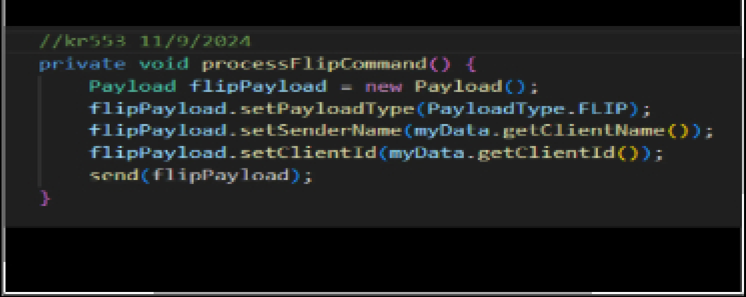
100%

Group: Client Commands
Task #2: Flip Command
Sub Task #1: Show the client side code for handling /flip

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
//kr553 11/9/2024
private void processFlipCommand() {
    Payload flipPayload = new Payload();
    flipPayload.setPayloadType(PayloadType.FLIP);
    flipPayload.setSenderName(myData.getClientName());
    flipPayload.setClientId(myData.getClientId());
    send(flipPayload);
}
```

Flip command in client code

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the logic

Response:

This code creates a "flip a coin" command. It packages details like the sender's name and type of command, then sends this information to the server, which handles the coin flip.

Sub-Task

100%

Group: Client Commands

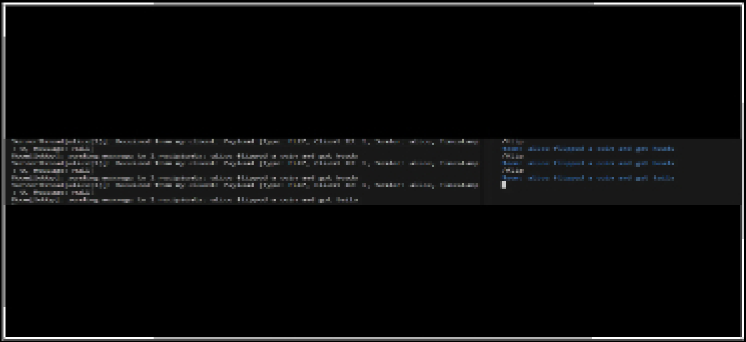
Task #2: Flip Command

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
[illegible]
```

```
/flip
[illegible]
/flip
[illegible]
/flip
[illegible]
```

here are some examp[les with payloads

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

End of Task 2

End of Group: Client Commands

Task Status: 2/2

Group

100%

Group: Text Formatting

Tasks: 1

Points: 3

COLLAPSE

Task

100%

Group: Text Formatting
Task #1: Text Formatting
Weight: ~100%
Points: ~3.00

COLLAPSE

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

Columns: 1

Sub-Task

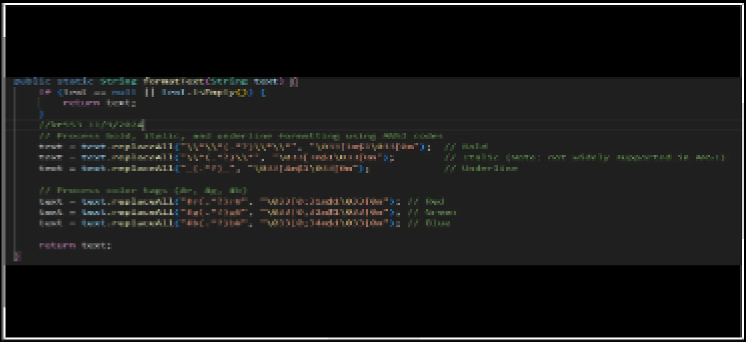
100%

Group: Text Formatting
Task #1: Text Formatting
Sub Task #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)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
public static String formatText(String text) {
    if (text == null || text.isEmpty()) {
        return text;
    }
    //[illegible] 11/4/2024
    // Process bold, italic, and underline formatting using ANSI codes
    text = text.replaceAll("\\*\\*(.*?)\\*\\*", "\033[1m$1\033[0m");  // Bold
    text = text.replaceAll("\\*(.*?)\\*", "\033[3m$1\033[0m");        // Italic (Note: not widely supported in ANSI)
    text = text.replaceAll("_(.*?)_", "\033[4m$1\033[0m");            // Underline

    // Process color tags (#r, #g, #b)
    text = text.replaceAll("#r(.*?)r#", "\033[0;31m$1\033[0m"); // Red
    text = text.replaceAll("#g(.*?)g#", "\033[0;32m$1\033[0m"); // Green
    text = text.replaceAll("#b(.*?)b#", "\033[0;34m$1\033[0m"); // Blue

    return text;
}
```

well i have implemented in TextFX file

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

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

Response:

The code replaces special symbols (**, *, _, #r, etc.) with ANSI codes to style text in the terminal. Symbols represent formatting like bold, italic, underline, or colors, turning plain text into styled output when displayed.

Sub-Task

100%

Group: Text Formatting
Task #1: Text Formatting

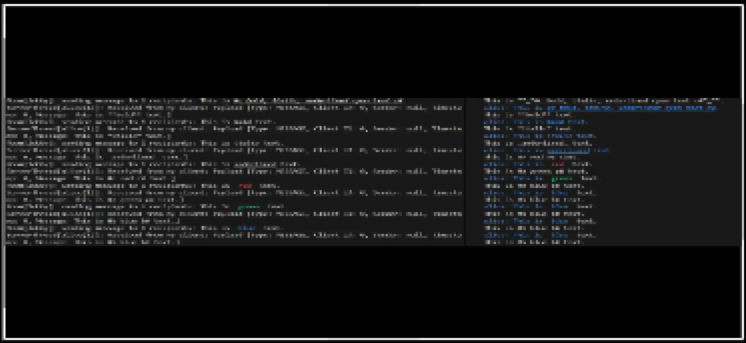
100%

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1


everytype examples with combination

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

End of Task 1

End of Group: Text Formatting
Task Status: 1/1

Group

100%

Group: Misc
Tasks: 3
Points: 1

COLLAPSE

Task

100%

Group: Misc
Task #1: Add the pull request link for the branch
Weight: ~33%
Points: ~0.33

COLLAPSE

Details:
Note: the link should end with /pull/#

Task URLs

URL #1

https://github.com/KushDev19/kr553-IT114-005/pull/7

URL

https://github.com/KushDev19/kr553-IT114-005/

https://github.com/KushDev19/kr553-IT114-005/pull/7

End of Task 1

Task

100%

Group: Misc
Task #2: Talk about any issues or learnings during this assignment
Weight: ~33%
Points: ~0.33

COLLAPSE

Task Response Prompt

Response:

Small issues like error for unknown symbol, undifined reference spent hours to sold that nothing more.

End of Task 2

Task

100%

Group: Misc
Task #3: WakaTime Screenshot
Weight: ~33%
Points: ~0.33

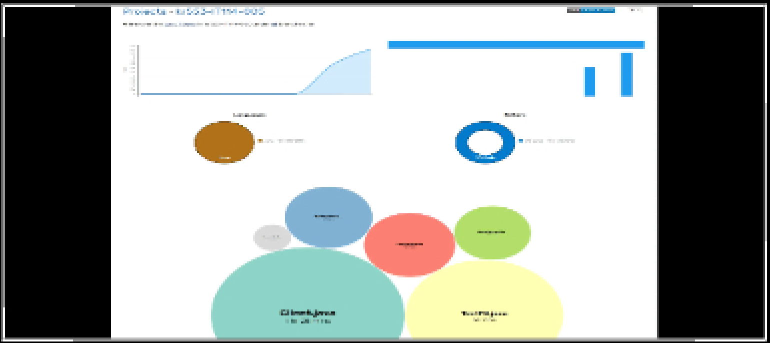
COLLAPSE

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: 2 Columns

4 2 1



Wakatime it is

End of Task 3

End of Task 3

End of Group: Misc
Task Status: 3/3

End of Assignment