Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-002-S2024/it114-milestone-2-chatroom-2024/grade/mth39

IT114-002-S2024 - [IT114] Milestone 2 Chatroom 2024

Submissions:

Submission Selection

1 Submission [active] 4/3/2024 8:07:27 PM

Instructions

COLLAPSE

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

Tasks: 12 Points: 10.00

Demonstrate Usage of Payloads (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots of your Payload class and subclasses and PayloadType

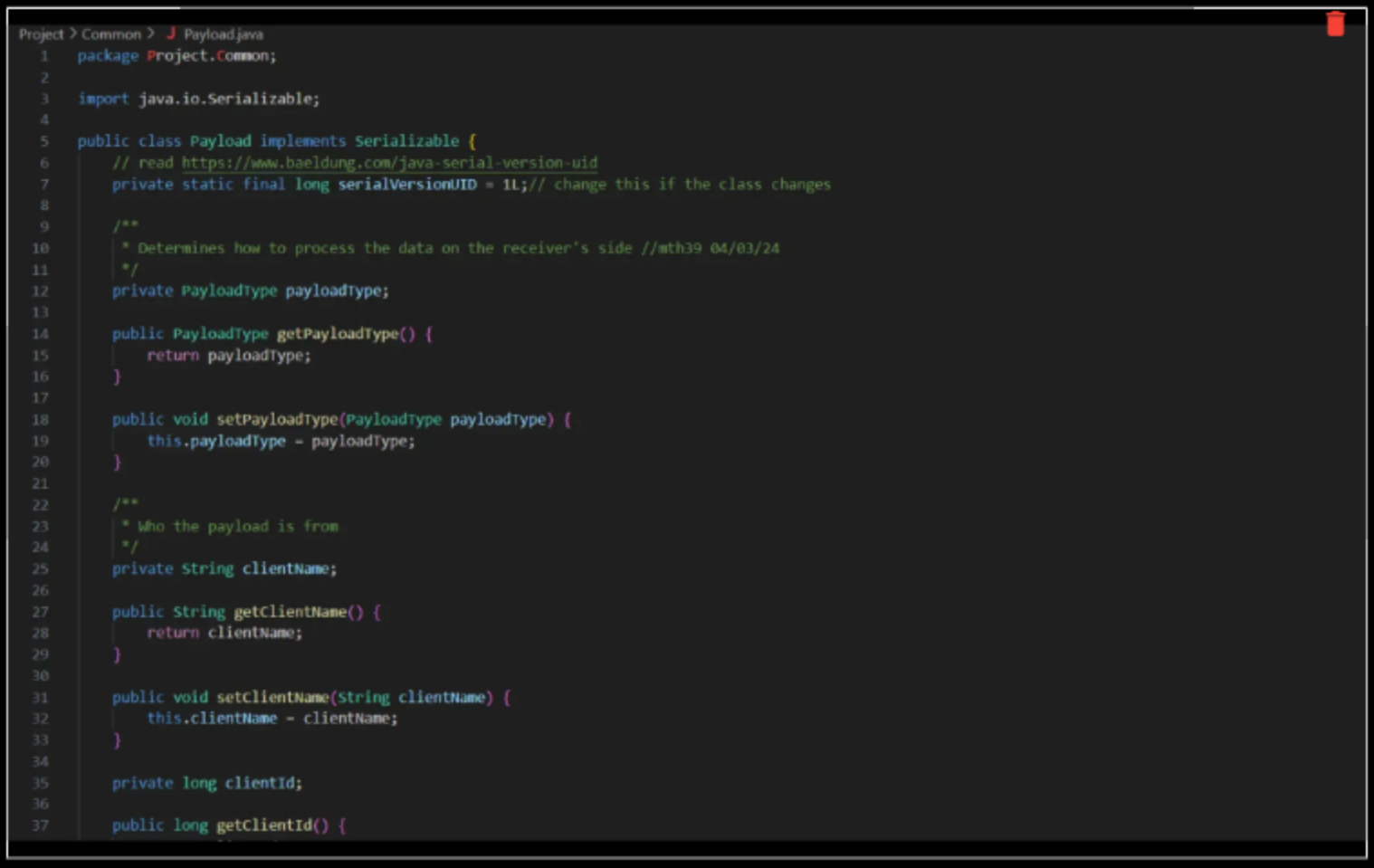
Checklist *The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Payload, equivalent of RollPayload, and any others
☐ #2	1	Screenshots should include ucid and date comment
☐ #3	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

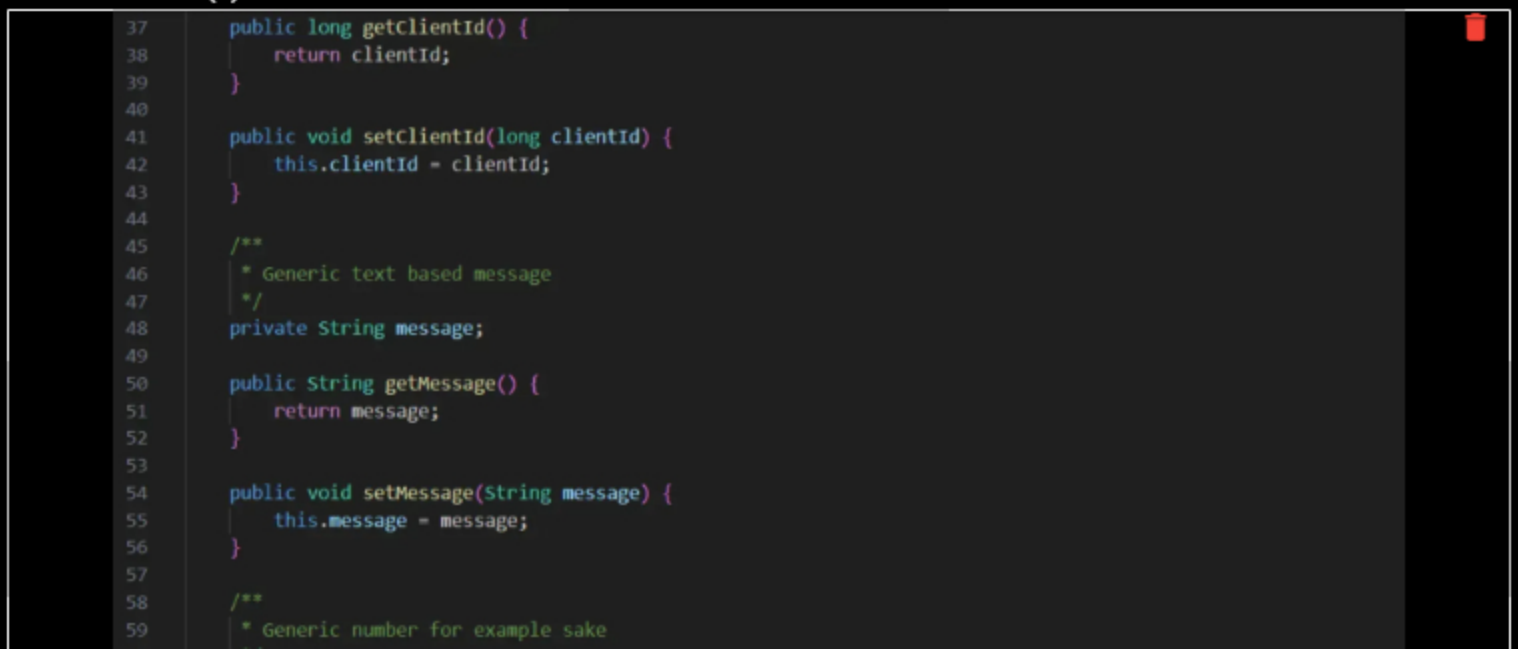
Small Medium Large



```
Project > Common > J Payload.java
1  package Project.Common;
2
3  import java.io.Serializable;
4
5  public class Payload implements Serializable {
6      // read https://www.baeldung.com/java-serial-version-uid
7      private static final long serialVersionUID = 1L;// change this if the class changes
8
9      /**
10     * Determines how to process the data on the receiver's side //mth39 04/03/24
11     */
12     private PayloadType payloadType;
13
14     public PayloadType getPayloadType() {
15         return payloadType;
16     }
17
18     public void setPayloadType(PayloadType payloadType) {
19         this.payloadType = payloadType;
20     }
21
22     /**
23     * Who the payload is from
24     */
25     private String clientName;
26
27     public String getClientName() {
28         return clientName;
29     }
30
31     public void setClientName(String clientName) {
32         this.clientName = clientName;
33     }
34
35     private long clientId;
36
37     public long getClientId() {
```

Payload.java screenshot 1

Checklist Items (0)



```
37     public long getClientId() {
38         return clientId;
39     }
40
41     public void setClientId(long clientId) {
42         this.clientId = clientId;
43     }
44
45     /**
46     * Generic text based message
47     */
48     private String message;
49
50     public String getMessage() {
51         return message;
52     }
53
54     public void setMessage(String message) {
55         this.message = message;
56     }
57
58     /**
59     * Generic number for example sake
```

```java
    */
    private int number;

    public int getNumber() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }

```

Payload.java screenshot 2

Checklist Items (0)

```java

    @Override
    public String toString() {
        return String.format("ClientId[%s], ClientName[%s], Type[%s], Number[%s], Message[%s]", // mth39 04/03/24
                getClientId(), getClientName(), getPayloadType().toString(), getNumber(), // This is the desired format
                                                                                           // the payload can accept
                getMessage());
    } <- #72-77 public String toString()
} <- #5-78 public class Payload implements Serializable
```

Payload.java screenshot 3

Checklist Items (0)

COLLAPSE

Task #2 - Points: 1

Text: Screenshots of the payloads being debugged/output to the terminal

Checklist

*The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Demonstrate flip
☐ #2	1	Demonstrate roll (both versions)
☐ #3	1	Demonstrate formatted message along with any others
☐ #4	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
case ROLL: // mth39 04/03/24
    int max = Integer.parseInt(comm2[1]);
    Random rand = new Random();
    int result = rand.nextInt(max + 1);
    sendMessage(client, "/roll: " + result);
    break;
case FLIP:
    Random rando = new Random();
    int results = rando.nextInt(2);
    if (results == 0) {
        sendMessage(client, "/flip: heads");
    } else {
        sendMessage(client, "/flip: tails");
    }
    break;
```

Code for Flip and Roll commands

Checklist Items (0)

COLLAPSE

Task #3 - Points: 1

Text: Explain the purpose of payloads and how your flip/roll payloads were made

Response:

The purpose of payloads are to encapsulate various types of information, such as messages, commands, and actions that are transferred the server and clients. For making my flip payload, when a client sends the /flip command, the server generates a random number to simulate a coin flip. If the generated number is 0, the server sends a message to the client indicating that the result is "heads". Otherwise, it sends a message indicating "tails". For my roll payload, when a client sends the /roll command followed by a number, the server simulates rolling a dice with the specified number of sides. A random object is instantiated to generate a random integer between 0 and the maximum specified value. The server then sends a message to the client indicating the result of the roll.

Demonstrate Roll Command (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshot of the following items

Checklist

*The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Client code that captures the command and converts it to a RollPayload (or equivalent) for both scenarios /roll # and /roll #d#
☐ #2	1	ServerThread code receiving the payload and passing it to the Room
☐ #3	1	Room handling the roll action correctly for both scenarios (/roll # and /roll #d#) including the message going back out to all clients
☐ #4	1	Code screenshots should include ucid and date comment
☐ #5	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
case ROLL: // mth39 04/03/24
    int max = Integer.parseInt(comm2[1]);
    Random rand = new Random();
    int result = rand.nextInt(max + 1);
    sendMessage(client, "/roll: " + result);
    break;
```

Code for Roll command

Checklist Items (0)

COLLAPSE

Task #2 - Points: 1

Text: Explain the logic in how the two different roll formats are handled and how the message flows

from the client, to the Room, and shared with all other users

Response:

A client initiates the roll command by sending a "/roll" message to the server that can be followed by a number indicating the maximum value of the roll. The server, which is continuously listening for incoming messages from clients, receives the message containing the roll command. After identifying the "/roll" command, the server parses the message to extract the maximum value for the roll. This value indicates the upper limit of the random number that will be generated. The server generates a random result within the specified range, from 0 up to the maximum value provided by the client. This result simulates the outcome of rolling a dice. Based on the random result, the server constructs a message indicating the outcome of the roll. The server broadcasts the message to all clients who are currently in the same room where the roll command was executed. This ensures that all users in the room receive the outcome of the roll. Each client displays the message on their respective interfaces, allowing all users in the room to see the outcome of the roll.

Demonstrate Flip Command (1 pt.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshot of the following items

Checklist

*The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Client code that captures the command and converts it to a payload
☐ #2	1	ServerThread receiving the payload and passing it to the Room
☐ #3	1	Room handling the flip action correctly
☐ #4	1	Code screenshots should include ucid and date comment
☐ #5	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
case FLIP:
    Random rando = new Random();
    int results = rando.nextInt(2);
    if (results == 0) {
        sendMessage(client, "/flip: heads");
```

```
        sendMessage(client, "/flip: heads");
    } else {
        sendMessage(client, "/flip: tails");
    }
    break;
```

Code for Flip command

Checklist Items (0)

COLLAPSE

Task #2 - Points: 1

Text: Explain the logic in how the flip command is handled and processed and how the message flows from the client, to the Room, and shared with all other users

Response:

A client initiates the flip command by sending a message "/flip" to the server. The server listens for incoming messages and identifies the "/flip" command. The server then generates a random result to simulate a coin flip. This involves using a random number generator to produce either a 0 or a 1, representing heads or tails. Based on the random result, the server constructs a message indicating the outcome of the coin flip. For example, if the result is 0, it constructs a message indicating "heads". If the result is 1, it constructs a message indicating "tails". Once the message is generated, the server broadcasts it to all clients who are currently in the same room where the flip command was executed. Each client displays the message on their respective interfaces, allowing all users in the room to see the outcome of the coin flip.

Demonstrate Formatted Messages (4 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshot of Room how the following formatting is processed from a message

Details:

Note: this processing is server-side

Slash commands are not valid solutions for this and will receive 0 credit

Checklist

*The checkboxes are for your own tracking

#	Points	Details

☐ #1	1	Room code processing for bold
☐ #2	1	Room code processing for italic
☐ #3	1	Room code processing for underline
☐ #4	1	Room code processing for color (at least R, G, B or support for hex codes)
☐ #5	1	Show each one working individually and one showing a combination of all of the formats and 1 color from the terminal
☐ #6	1	Must not rely on the user typing html characters, but the output can be html characters
☐ #7	1	Code screenshots should include ucid and date comment
☐ #8	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
case BOLD: // mth39 04/03/24
    String boldMessage = message.replace("*/", "<b>");
    boldMessage = boldMessage.replace("/*", "</b>");
    sendMessage(client, boldMessage);
    break;

case ITALIC:
    String italicMessage = message.replace("&_", "<i>");
    italicMessage = italicMessage.replace("_&", "</i>");
    sendMessage(client, italicMessage);
    break;

case COLOR:
    String color = comm2[1];
    String colorMessage = message.replace(color, "<font color='" + color + "'>");
    colorMessage = colorMessage.replace("#", "</font>");
    sendMessage(client, colorMessage);
    break;
```

Code screenshot for bold, italic, and color

Checklist Items (0)

```
case UNDERLINE:
    String underlineMessage = message.replace("_/", "<u>");
    underlineMessage = underlineMessage.replace("/_", "</u>");
    sendMessage(client, underlineMessage);
```

Code screenshot for underline

Checklist Items (0)

```
Harry: testing
123
Waiting for input
Debug Info: Type[MESSAGE], Number[0],
 Message[123]
Harry: 123
This message is **bold**
Waiting for input
Debug Info: Type[MESSAGE], Number[0],
 Message[This message is **bold**]
Harry: This message is **bold**
Debug Info: Type[MESSAGE], Number[0],
 Message[This message is <b>bold</b>]
```

Screenshot of bold being displayed on terminal

Checklist Items (0)

```
Message[__italic__]
```

```
Harry: __italic__
Debug Info: Type[MESSAGE], Number[0]
 Message[<i>italic</i>]
Harry: <i>italic</i>
```

Screenshot of italic being displayed on terminal

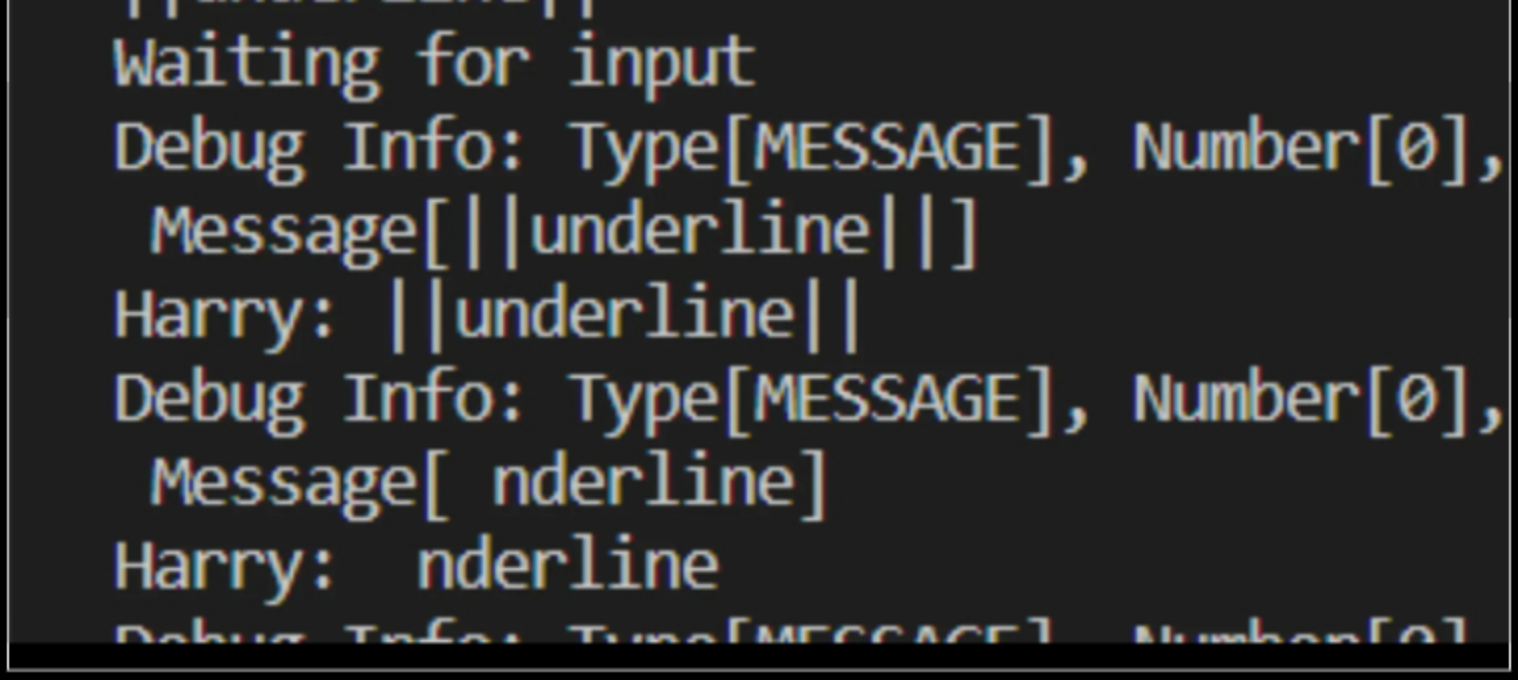
Checklist Items (0)

```
Debug Info: Type[MESSAGE], Number[0],
 Message[red]
Harry: red
#r red r#
Waiting for input
Debug Info: Type[MESSAGE], Number[0],
 Message[#r red r#]
Harry: #r red r#
Debug Info: Type[MESSAGE], Number[0],
 Message[<color red>red</color>]
Harry: <color red>red</color>
```

Screenshot of color being displayed on terminal

Checklist Items (0)

```
 Message[hi]
Harry: hi
||underline||
```



Screenshot of underline being displayed on terminal

Checklist Items (0)

COLLAPSE

Task #2 - Points: 1

Text: Explain the following

Checklist

*The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Which special characters translate to the desired effect
☐ #2	1	How the logic works that converts the message to its final format

Response:

For bold, italics, color, and underline, I used the replace method. This allowed me to put in the text that the user will put in to display their text in either bold, italic, color, or underlined and replace it with HTML tags.

Misc (1 pt.)

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

Missing URL

COLLAPSE

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

For some reason, my flip and roll commands were not correctly working when I tried them in the terminal.

COLLAPSE

Task #3 - Points: 1

Text: WakaTime Screenshot

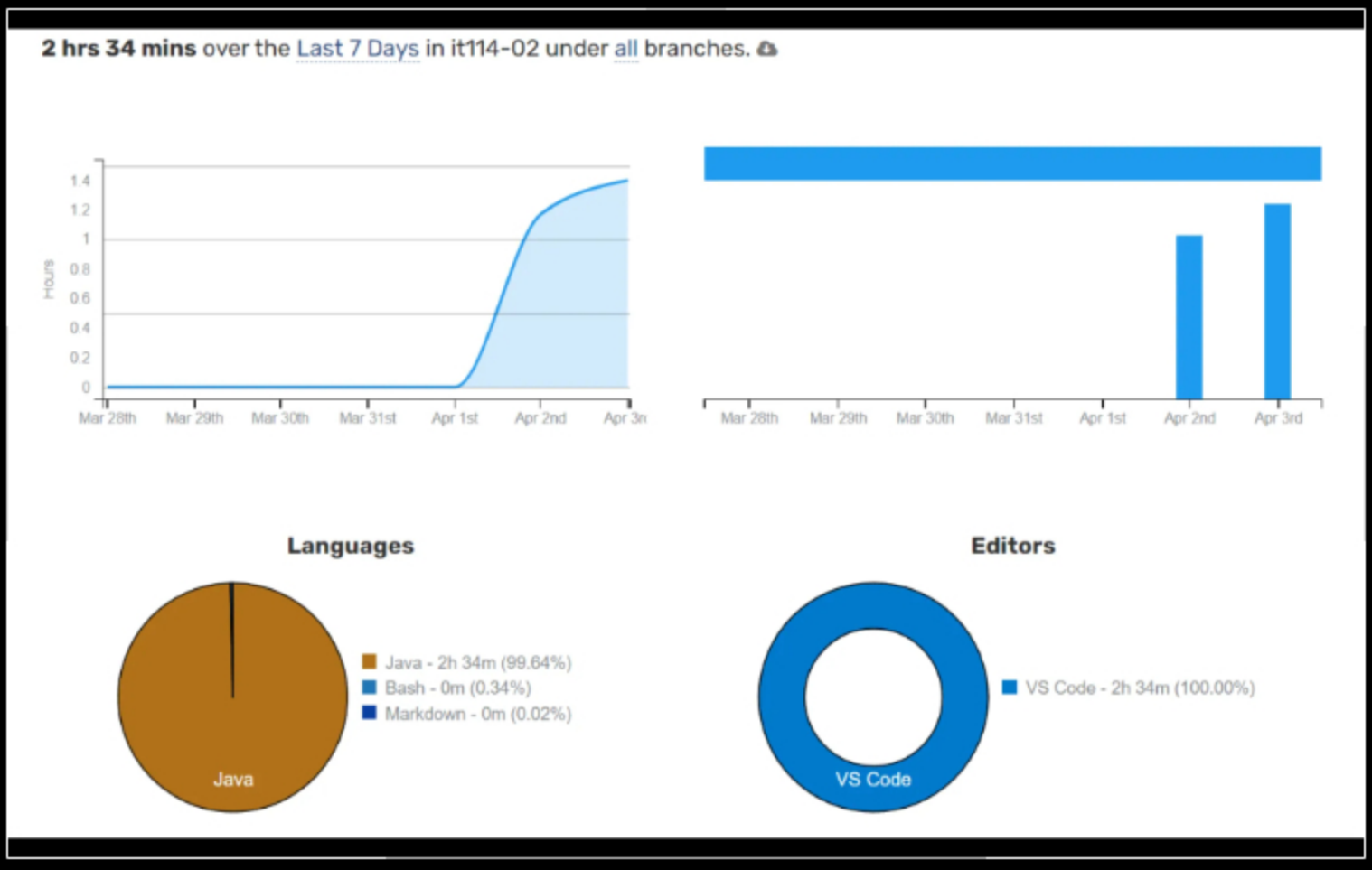
Details:

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 spent on project in the last week

End of Assignment