

Submission Worksheet

CLICK TO GRADE

<https://learn.ethereallab.app/assignment/IT114-003-F2024/it114-milestone-2-trivia-2024-m24/grade/ma2633>

Course: IT114-003-F2024

Assignment: [IT114] Milestone 2 Trivia 2024 (M24)

Student: Mohamad A. (ma2633)

Submissions:

Submission Selection

1 Submission [submitted] 11/13/2024 10:26:45 PM

Instructions

[^ COLLAPSE ^](#)

1. Implement the Milestone 2 features from the project's proposal document:
<https://docs.google.com/document/d/1h2aEWUoZ-etpz1CRI-StaWbZTjkd9BDMq0b6TXK4utl/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

Group: Payloads

Tasks: 3

Points: 1

100%

[^ COLLAPSE ^](#)

Task

Group: Payloads

Task #1: Base Payload Class

Weight: ~33%

Points: ~0.33

100%

▲ COLLAPSE ▲

ⓘ Details:

All code screenshots must have ucid/date visible.

Columns: 1

Sub-Task

Group: Payloads

Task #1: Base Payload Class

Sub Task #1: Show screenshot of the code for this file

100%

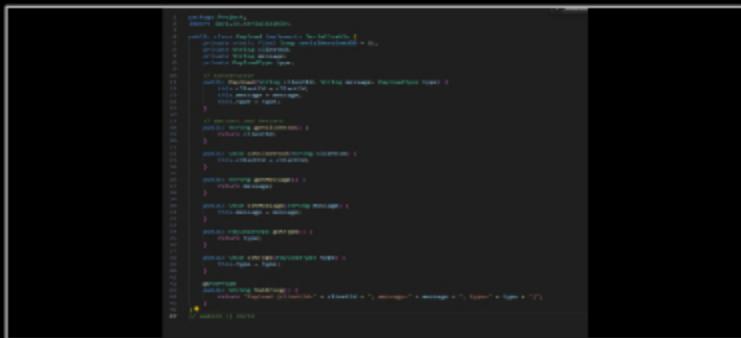
☒ Task Screenshots

Gallery Style: 2 Columns

4

2

1



```
package payloads;
public class BasePayload {
    private String payloadType;
    private String clientId;
    private String message;

    public BasePayload(String payloadType, String clientId, String message) {
        this.payloadType = payloadType;
        this.clientId = clientId;
        this.message = message;
    }

    public String getPayloadType() {
        return payloadType;
    }

    public void setPayloadType(String payloadType) {
        this.payloadType = payloadType;
    }

    public String getClientId() {
        return clientId;
    }

    public void setClientId(String clientId) {
        this.clientId = clientId;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

payload code

Caption(s) (required) ✓

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

≡, Task Response Prompt

Briefly explain the purpose of each property and serialization

Response:

The Payload class has three main properties: payloadType to indicate what kind of data is being sent (like a message or command), clientId to identify which client is sending the data, and message to carry the actual content. Serialization allows this object to be turned into a format that can be sent over a network, so different clients and servers can easily share and understand the information.

Sub-Task

Group: Payloads

Task #1: Base Payload Class

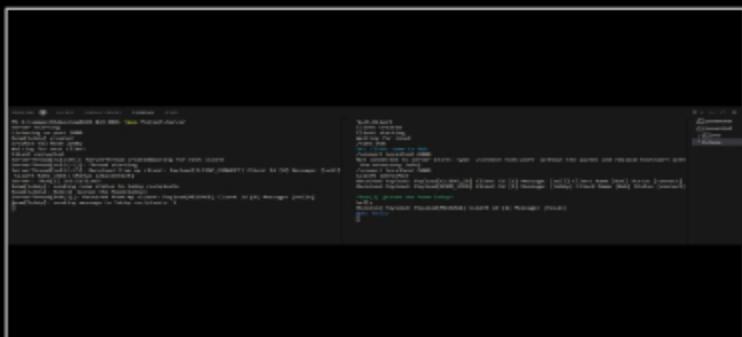
Sub Task #2: Show screenshot examples of the terminal output for this object

100%

Task Screenshots

Gallery Style: 2 Columns

4 2 1



terminal output

Caption(s) (required) ✓

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

End of Task 1

Task



Group: Payloads

Task #2: QAPayload Class

Weight: ~33%

Points: ~0.33

 COLLAPSE 

 Details:

All code screenshots must have ucid/date visible.



Columns: 1

Sub-Task



Group: Payloads

Task #2: QAPayload Class

Sub Task #1: Show screenshot of the code for this file

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
1 package ProjectA
2
3 import java.util.List
4
5 public class Question {
6     public static void main(String[] args) {
7         QAPayload question = new QAPayload();
8
9         question.setQuestion("What is the capital of France?");
10        question.setCategory("Geography");
11        question.setAnswers(new String[]{"A: Berlin", "B: Madrid", "C: Paris", "D: Rome"});
12        question.setCorrectAnswer("C");
13    }
14 }
```

the rest of the code

Caption(s) (required) ✓

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

≡ Task Response Prompt

Briefly explain the purpose of each property

Response:

The QAPayload class has four main properties: question stores the text of the question that is presented to players, while category specifies the category or type of the question, such as Science or History. The answers property is a list of possible answer choices that the players can pick from. Finally, correctAnswer indicates the correct option, such as "A", "B", "C", or "D", which helps in evaluating players' responses.

Sub-Task

Group: Payloads

100%

Task #2: QAPayload Class

Sub Task #2: Show screenshot examples of the terminal output for this object

☒ Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
1 package ProjectA
2
3 import java.util.List
4
5 public class Question {
6     public static void main(String[] args) {
7         QAPayload question = new QAPayload();
8
9         question.setQuestion("What is the capital of France?");
10        question.setCategory("Geography");
11        question.setAnswers(new String[]{"A: Berlin", "B: Madrid", "C: Paris", "D: Rome"});
12        question.setCorrectAnswer("C");
13    }
14 }
```

4 C:\Users\Julian\Documents\GitHub\ProjectA\\$ java ProjectA
5 [QAPayload@1] setQuestion(What is the capital of France)
6 [QAPayload@1] setCategory(Geography)
7 [QAPayload@1] setAnswers([A: Berlin, B: Madrid, C: Paris, D: Rome])
8 [QAPayload@1] setCorrectAnswer(C)

terminal output for QAPayload

Caption(s) (required) ✓

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

End of Task 2

Task

Group: Payloads

Task #3: PointsPayload Class

Weight: ~33%

Points: ~0.33

100%

▲ COLLAPSE ▲



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



Columns: 1

Sub-Task



Group: Payloads

Task #3: PointsPayload Class

Sub Task #1: Show screenshot of the code for this file

Task Screenshots

Gallery Style: 2 Columns

4

2

1

```
// Pointspayload class
class Pointspayload extends Payload {
    private Map<Long, Integer> playerPoints;

    public Map<Long, Integer> getPlayerPoints() {
        return playerPoints;
    }

    public void setPlayerPoints(Map<Long, Integer> playerPoints) {
        this.playerPoints = playerPoints;
    }

    @Override
    public String toString() {
        return String.format("Pointspayload[player points: %s]", playerPoints);
    }
}

// maz653 || 11/12/24
```

pointpayload

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the purpose of each property

Response:

playerPoints (Map<Long, Integer>): This property stores the points of each player in the game. The Map uses the player's ID (of type Long) as the key and the player's points (of type Integer) as the value. This helps keep track of each player's score throughout the game.

Sub-Task



Group: Payloads

Task #3: PointsPayload Class

Sub Task #2: Show screenshot examples of the terminal output for this object

Task Screenshots

Gallery Style: 2 Columns

4

2

1

```
ProjectL3 | 1 File (1 modified) 0.0
1 package Projects;
2 import java.util.Map;
3
4 public class Pointspayload {
5     private Map<Long, Integer> playerPoints;
6
7     public static void main(String[] args) {
8         // Create an instance of Pointspayload
9         Pointspayload pointsPayload = new Pointspayload();
10
11         // Set player points
12         pointsPayload.setPlayerPoints(new Map<Long, Integer>());
13
14         // Print the Pointspayload instance
15         System.out.println(pointsPayload);
16     }
17 }
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Hans\OneDrive\Documents\GitHub\piontpayload [3m, 3r, 3w] PROJECT: piontpayload
Payload.cs(1,1): Error CS0246: The type or namespace name 'Payload' does not exist in the namespace 'piontpayload'. Are you missing an assembly reference?
PS C:\Users\Hans\OneDrive\Documents\GitHub\piontpayload [3m, 3r, 3w]
```

terminal output of piontpayload

Caption(s) (required) ✓

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

End of Task 3

End of Group: Payloads

Task Status: 3/3

Group

Group: Questions

Tasks: 1

Points: 1.25

100%

▲ COLLAPSE ▲

Task

Group: Questions

Task #1: Data Structure

Weight: ~100%

Points: ~1.25

100%

▲ COLLAPSE ▲

i Details:

All code screenshots must have ucid/date visible.



Columns: 1

Sub-Task

Group: Questions

Task #1: Data Structure

Sub Task #1: Show the code related to Questions/Answers (text, category, answers, correct)

100%

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
package org.piontpayload;
import java.util.ArrayList;
public class Question {
    private String ucid;
    private String question;
    private String category;
    private String answers;
    private String correct;
    private String date;
    public Question(String ucid, String question, String category, String answers, String correct, String date) {
        this.ucid = ucid;
        this.question = question;
        this.category = category;
        this.answers = answers;
        this.correct = correct;
        this.date = date;
    }
    public String getUcid() {
        return ucid;
    }
}
```

```
public class Question {
    private String questionText;
    private String category;
    private List<String> answerOptions;
    private String correctAnswer;

    public Question(String questionText, String category, List<String> answerOptions, String correctAnswer) {
        this.questionText = questionText;
        this.category = category;
        this.answerOptions = answerOptions;
        this.correctAnswer = correctAnswer;
    }

    public String getQuestionText() {
        return questionText;
    }

    public String getCategory() {
        return category;
    }

    public List<String> getAnswerOptions() {
        return answerOptions;
    }

    public String getCorrectAnswer() {
        return correctAnswer;
    }

    @Override
    public String toString() {
        return "Question{" +
                "questionText='" + questionText + '\'' +
                ", category='" + category + '\'' +
                ", answerOptions=" + answerOptions +
                ", correctAnswer='" + correctAnswer + '\''
        ;
    }
}
```

Show the code related to Questions/Answers

Caption(s) (required) ✓

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

Task Response Prompt

Explain the class and each property

Response:

The Question class is a simple data container for a trivia game. It has four properties: questionText, category, answerOptions, and correctAnswer. The questionText is the actual question being asked, while the category helps group questions (like "Math" or "History"). The answerOptions is a list of possible answers players can pick from, and correctAnswer is the right answer from those options. There's a constructor to set all these properties, and getters to access them easily. There's also a `toString()` method to neatly print out the question details, which is helpful for seeing what's stored in a question.

Sub-Task

Group: Questions

100%

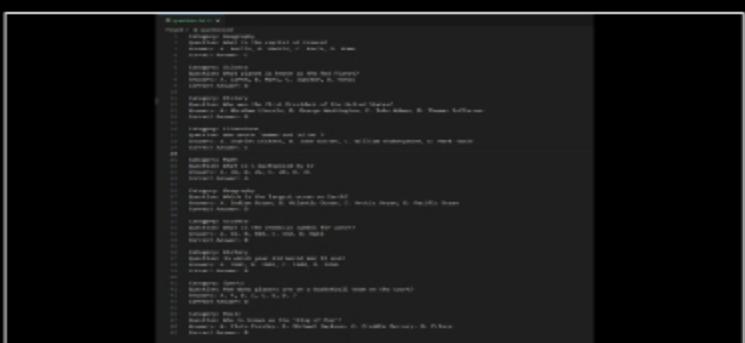
Task #1: Data Structure

Sub Task #2: Show the text file with your sample questions and answers

Task Screenshots

Gallery Style: 2 Columns

4 2 1



text file with your sample questions

Caption(s) (required) ✓

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

End of Task 1

End of Group: Questions

Task Status: 1/1

Group

Group: Session Start

Tasks: 2

100%

Points: 1.25

▲ COLLAPSE ▲

Task

100%

Group: Session Start
Task #1: Question List
Weight: ~50%
Points: ~0.63

▲ COLLAPSE ▲

i Details:

All code screenshots must have ucid/date visible.

**Sub-Task**

100%

Group: Session Start
Task #1: Question List
Sub Task #1: Show the GameRoom loading the question list from a file

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
// Method to load questions from a file
public void loadQuestions() throws IOException {
    try (BufferedReader reader = new BufferedReader(new FileReader("questions.txt"))) {
        String line;
        while ((line = reader.readLine()) != null) {
            String[] parts = line.split(",");
            if (parts.length == 4) {
                String questionText = parts[0];
                String category = parts[1];
                String answer = parts[2];
                String[] options = parts[3].split(",");
                GameRoom question = new GameRoom(questionText, category, Arrays.asList(options), correctAnswer);
                questionList.add(question);
                System.out.println("Added question: " + question);
            } else {
                System.out.println("Invalid question format: " + line); // Debug message for invalid format
            }
        }
        System.out.println("Total questions loaded: " + questionList.size()); // Debug message
    } catch (FileNotFoundException e) {
        System.out.println("Error loading questions: " + e.getMessage());
    }
} // end of class
```

Method to load questions from a file

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

In this method, the code reads questions from a file called "questions.txt". It reads each line, extracts the category, question, answer options, and the correct answer, and then adds the questions to questionList. This helps load all the game questions from an external file, making it easy to update or add new questions.

End of Task 1

Task



Group: Session Start

Task #2: First round

Weight: ~50%

Points: ~0.63

 COLLAPSE 

 Details:

All code screenshots must have ucid/date visible.



Sub-Task



Group: Session Start

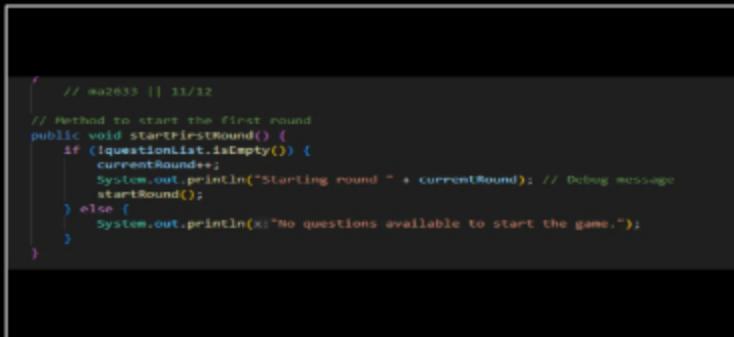
Task #2: First round

Sub Task #1: Show the code that triggers the first round

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
// wa2633 || 11/32
// Method to start the first round
public void startFirstRound() {
    if (questionList.isEmpty()) {
        currentRound++;
        System.out.println("Starting round " + currentRound); // Debug message
        startRound();
    } else {
        System.out.println("No questions available to start the game.");
    }
}
```

code that triggers the first round

Caption(s) (required) ✓

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

End of Task 2

End of Group: Session Start

Task Status: 2/2

Group



Group: Round Start

Tasks: 3

Points: 1.25

 COLLAPSE 

Task

Group: Round Start

100%

Task #1: Picking a question

Weight: ~33%

Points: ~0.42

▲ COLLAPSE ▲

Sub-Task

Group: Round Start

100%

Task #1: Picking a question

Sub Task #1: Show the code related to getting a random question from the list (it should get removed so it's not used again)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
// method to start a round
private void startRound() {
    if (questionList.isEmpty()) {
        currentQuestion = questionList.remove((int) (Math.random() * questionList.size()));
        broadcastQuestionToClients(currentQuestion);
        startRoundTimer();
    } else {
        System.out.println("no more questions available.");
        endSession();
    }
}
```

getting a random question from the list

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

currentQuestion = questionList.remove((int) (Math.random() * questionList.size())); This line gets a random index from the questionList. Math.random() * questionList.size() generates a random index. questionList.remove(...) removes the selected question from the list, so it's not used again.

End of Task 1

Task

Group: Round Start

100%

Task #2: Syncing Question

Weight: ~33%

Points: ~0.42

▲ COLLAPSE ▲

Details:

All code screenshots must have ucid/date visible.
Any terminal output should have at least 3 clients involved.



Columns: 1

Sub-Task

Group: Round Start

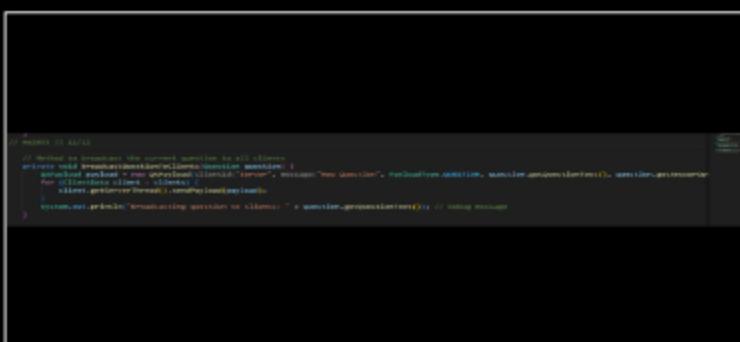
Task #2: Syncing Question

Sub Task #1: Show the code the syncs the question and answer choices to all clients (don't send the correct answer)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



code that syncs the question and answer choices to all clients

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

The broadcastQuestionToClients(Question question) method: QAPayload payload = new QAPayload(...); Creates a payload with the question text and answer choices. The payload does not include the correct answer, ensuring clients only receive the question and possible answer choices. for (ClientData client : clients): Iterates over all connected clients. client.getServerThread().sendPayload(payload): Sends the payload (question and answer choices) to each client.

Sub-Task

Group: Round Start

Task #2: Syncing Question

Sub Task #2: Show the terminal output of all clients receiving it

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Three terminal windows showing MySQL command-line interface interactions between clients and a server.

Client 1:

```
PS C:\Users\Wahidul\Documents\GitHub\aprendizaje03-12&-08>
Connected to server at localhost:3306
Current database: test
SHOW STATUS like '%long_query_time%';
+-----+
| Long_query_time |
+-----+
| 0               |
+-----+
1 row in set (0.00 sec)

SELECT max(score) AS max_score
FROM scores;
+-----+
| max_score |
+-----+
| 100        |
+-----+
1 row in set (0.00 sec)
```

Client 2:

```
PS C:\Users\Wahidul\Documents\GitHub\aprendizaje03-12&-08>
Connected to server at localhost:3306
Current database: test
SHOW STATUS like '%long_query_time%';
+-----+
| Long_query_time |
+-----+
| 0               |
+-----+
1 row in set (0.00 sec)

SELECT max(score) AS max_score
FROM scores;
+-----+
| max_score |
+-----+
| 100        |
+-----+
1 row in set (0.00 sec)
```

Client 3:

```
PS C:\Users\Wahidul\Documents\GitHub\aprendizaje03-12&-08>
Connected to server at localhost:3306
Current database: test
SHOW STATUS like '%long_query_time%';
+-----+
| Long_query_time |
+-----+
| 0               |
+-----+
1 row in set (0.00 sec)

SELECT max(score) AS max_score
FROM scores;
+-----+
| max_score |
+-----+
| 100        |
+-----+
1 row in set (0.00 sec)
```

clients receiving questions

Caption(s) (required) ✓

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

End of Task 2

Task

100%

Group: Round Start

Task #3: Round Timer

Weight: ~33%

Points: ~0.42

▲ COLLAPSE ▲

ⓘ Details:

All code screenshots must have ucid/date visible.
Any terminal output should have at least 3 clients involved.



Columns: 1

Sub-Task

100%

Group: Round Start

Task #3: Round Timer

Sub Task #1: Show the code that triggers the round timer (including the expiry callback)

🖼 Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
// Method to start the round timer
private void startRoundTimer() {
    roundTimer.schedule(new TimerTask() {
        @Override
        public void run() {
            System.out.println("Round timer expired.");
            endRound();
        }
    }, delay:30000); // 30 seconds per round
}
```

Method to start the round timer

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

Timer Initialization: The startRoundTimer() method uses the Timer class to schedule a timer task.

Timer Task: A new TimerTask is created and scheduled to run after 30,000 milliseconds (or 30 seconds). This means that the round will automatically expire after 30 seconds.

Expiry Callback: When the timer expires, the run() method inside the TimerTask is called, and it prints "Round timer expired." to the terminal, then it calls the endRound() method to handle the logic for ending the round.

Sub-Task

Group: Round Start



Task #3: Round Timer

Sub Task #2: Show how the clients are informed of the time (i.e., synced or stopwatch-like)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
// Method to start the round timer with periodic updates to clients
private void startRoundTimer() {
    long updateInterval = 1000; // 1 second per round
    long timeRemaining = 30000; // Update every 5 seconds
    roundTimer.scheduleAtFixedRate(new TimerTask() {
        long timeRemainingLeft = timeRemaining;
        @Override
        public void run() {
            if (timeRemainingLeft > 0) {
                broadcastTimeUpdate();
            } else {
                System.out.println("Round timer expired.");
                endRound();
            }
        }
    }, 0, updateInterval);
}

// Method to broadcast time updates to all clients
private void broadcastTimeUpdate(long timeRemaining) {
    for (ClientObject client : clients) {
        client.broadcastTimeUpdate("server", messageTimeFormat, messageType, time, timeRemaining);
    }
    System.out.println("Broadcasting time update to clients: " + timeRemaining / 1000 + " seconds remaining");
}

// endRound() - decreasing time update by 1000ms / timeRemaining / 1000 = 1 second remaining
}
// endRound() - 11.11/2
```

Method to start the round timer with periodic updates to clients

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

The startRoundTimer() method schedules a repeating task that decreases the remaining time by a fixed interval (updateInterval). The broadcastTimeUpdate() method is called to inform all clients of the time remaining in the round.

Sub-Task

Group: Round Start

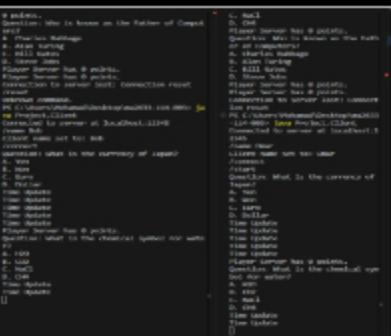


Task #3: Round Timer

Sub Task #3: Show an example from the terminal

Task Screenshots

Gallery Style: 2 Columns

4	2	1
		

Clients are informed of the time

Caption(s) (required) ✓

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

End of Task 3

End of Group: Round Start

Task Status: 3/3

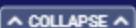
Group



Group: During Round

Tasks: 2

Points: 1.5

 COLLAPSE

Task



Group: During Round

Task #1: Answer Command

Weight: ~50%

Points: ~0.75

 COLLAPSE

ⓘ Details:

All code screenshots must have ucid/date visible.



Columns: 1

Sub-Task

Group: During Round

Task #1: Answer Command

Sub Task #1: Show the code that processes the /answer text



Task Screenshots

4 2 1

```
// Handle answer payload
private void handleAnswerPayload(Payload payload) {
    if (currentRoom != null && currentRoom instanceof GameRoom) {
        ((GameRoom) currentRoom).processAnswer(clientData, payload.getMessage());
    }
} // 102644 | 11/14
```

answer processes

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

`handleAnswerPayload(Payload payload)`: This is a private method that processes an answer payload received from a client.

`if (currentRoom != null && currentRoom instanceof GameRoom)`: This condition checks if the client is currently in a room and if the room is an instance of GameRoom. This ensures that the answer is processed only when the player is in a game room.

Sub-Task

Group: During Round

100%

Task #1: Answer Command

Sub Task #2: Show the code of the GameRoom handling the answer choice of a Player (and how correct answers are handled) Show that the answer is blocked from changing.

Task Screenshots

4 2 1

```
// method to process a clients answer
public void processAnswer(Client client, String answer) {
    if (currentQuestion != null && currentQuestion.getAnswers().contains(answer)) {
        if (currentQuestion.isCorrectAnswer(currentQuestion.getCorrectAnswer())) {
            client.addPoints(client.getConnectionId()); // award points to the player
            GameRoom.printQuestion(client, "Answered correctly and earned 10 points.");
        } else {
            GameRoom.printQuestion(client, "Answered incorrectly.");
        }
        checkAllPlayersAnswered();
    } else {
        System.out.println("Invalid answer provided by " + client.getName());
    }
}

// method to check if all players have answered
private void checkAllPlayersAnswered() {
    boolean allAnswered = true;
    for (Client client : clients) {
        // for real implementation, we would track if each player answered
        // if not simplicity, assuming all players answer is unique for now
        if (!allAnswered) {
            break;
        }
    }
} // 102644 | 11/14
```

GameRoom handling the answer choice of a Player

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Explain in concise steps how this logically works

Response:

processAnswer(ClientData client, String answer): This method processes the answer from the client. if (currentQuestion != null && currentQuestion.getAnswerOptions().contains(answer)): Checks if the current question is available and if the provided answer is valid.

Sub-Task

Group: During Round

Task #1: Answer Command

Sub Task #3: Show the code from the GameRoom that tells all Players a Player locked in an answer

100%

■ Task Screenshots

Gallery Style: 2 Columns

4 2 1

```

// method to notify all players when a player locks in an answer
public void notifyPlayersAnswerLocked(ClientData player) {
    String message = player.getUsername() + " has locked in an answer.";
    Payload payload = new Payload(clientId:"Server", message, PayloadType.NOTIFICATION);
    for (ClientData client : clients) {
        client.getServerThread().sendPayload(payload);
    }
}
//method ||| 11/18

```

code from the GameRoom that tells all Players a Player locked in an answer

Caption(s) (required) ✓

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

≡ Task Response Prompt

Explain in concise steps how this logically works

Response:

Player Submits Answer: When a player provides an answer, the processAnswer() method is called.

Answer Evaluation: The processAnswer() method checks if the provided answer is valid for the current question.

Award Points: If the answer is correct, the player is awarded points, and a message is printed for feedback.

Broadcast Answer Lock-In: Regardless of correctness, a message is broadcasted to all players to inform them that the player has locked in their answer.

Notify Players: The new broadcastPlayerLockedInAnswer() method is invoked, which sends a notification payload to all players, indicating that a player has locked in an answer.

Sub-Task

Group: During Round

Task #1: Answer Command

Sub Task #4: Show at least 3 terminal examples of wrong and correct answers (should be shown on all clients)

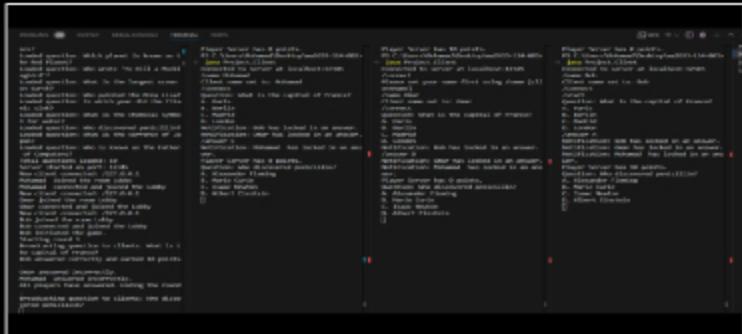
100%

■ Task Screenshots

Task Screenshots

Gallery Style: 2 Columns

4 2 1



3 terminal examples of wrong and correct answers

Caption(s) (required) ✓

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

End of Task 1

Task

100%

Group: During Round

Task #2: Join in progress

Weight: ~50%

Points: ~0.75

[▲ COLLAPSE ▲](#)

Details:

All code screenshots must have ucid/date visible.



Sub-Task

100%

Group: During Round

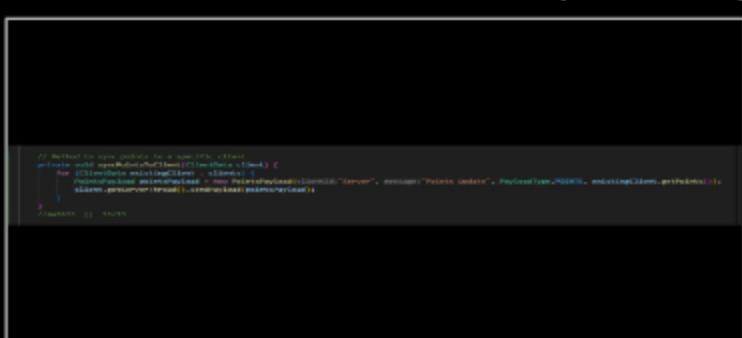
Task #2: Join in progress

Sub Task #1: Show the code related to syncing game state to a joining client if the game is in progress

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Method to sync points to a specific client

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

The method iterates through each client in the list and sends the points update to the newly joined client. It creates a PointsPayload object with relevant data, such as the client ID, message, payload type, and the points value, and sends it using the sendPayload() method.

End of Task 2

End of Group: During Round

Task Status: 2/2

Group

Group: Round End

Tasks: 3

Points: 1.5

100%

▲ COLLAPSE ▲

Task

Group: Round End

Task #1: End Round Conditions

Weight: ~33%

Points: ~0.50

100%

▲ COLLAPSE ▲

i Details:

All code screenshots must have ucid/date visible.



Columns: 1

Sub-Task

Group: Round End

Task #1: End Round Conditions

Sub Task #1: Show the code related to ending a round when all Players answer correctly

100%

Task Screenshots

Gallery Style: 2 Columns

```
// Method to check if all players have answered
private void checkAllPlayersAnswered() {
    if (answeredClients.size() == clients.size()) {
        System.out.println("All players have answered. Ending the round.");
        endRound();
    }
}

// Method to end the round
private void endRound() {
    roundTimer.cancel();
    roundTimer = new Timer(); // Reset the timer for the next round
    syncPointsToClients();
    if (currentRound < 5) { // Example condition for session end after 5 rounds
        startRound();
    } else {
        endSession();
    }
}
// mo2650 || 11/14
```

Show the code related to ending a round when all Players answer correctly

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

processAnswer(): Adds clients to answeredClients after they answer and checks if all players have answered using checkAllPlayersAnswered(). checkAllPlayersAnswered(): Ends the round by calling endRound() if all players have answered. endRound(): Cancels the timer, synchronizes points, and either starts a new round or ends the session after a certain number of rounds.

Sub-Task

Group: Round End

100%

Task #1: End Round Conditions

Sub Task #2: Show the code related to ending a round when the round timer expires

Task Screenshots

Gallery Style: 2 Columns

```
// Method to end the round
private void endRound() {
    roundTimer.cancel();
    roundTimer = new Timer(); // Reset the timer for the next round
    syncPointsToClients();
    if (currentRound < 5) { // Example condition for session end after 5 rounds
        startRound();
    } else {
        endSession();
    }
}
// mo2650 || 11/14
```

Show the code related to ending a round when the round timer expires

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

`startRoundTimer()`: Sets a timer to track the duration of the round. If the timer runs out, it calls `endRound()` to stop the round. `endRound()`: Cancels the timer, resets it, synchronizes points to clients, and either starts a new round or ends the session based on the current round count.

End of Task 1

Task



Group: Round End
Task #2: Points
Weight: ~33%
Points: ~0.50

[▲ COLLAPSE ▲](#)

ⓘ Details:

All code screenshots must have ucid/date visible.
Any terminal output should have at least 3 clients involved.



Columns: 1

Sub-Task

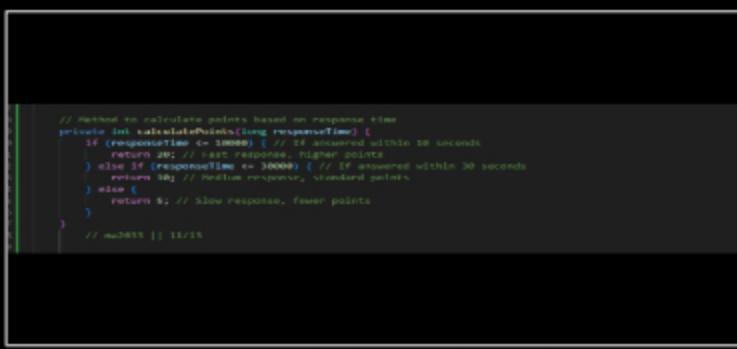


Group: Round End
Task #2: Points
Sub Task #1: Show how points are calculated (fastest = higher score)

🖼 Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
    // Method to calculate points based on response time
    private int calculatePoints(int responseTime) {
        if (responseTime <= 10000) { // If answered within 10 seconds
            return 20; // Fast response, higher points
        } else if (responseTime <= 30000) { // If answered within 30 seconds
            return 10; // Medium response, standard points
        } else {
            return 5; // Slow response, fewer points
        }
    }
    // m02855 ||| 11/11
```

Show how points are calculated (fastest = higher score)

Caption(s) (required) ✓

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

≡ Task Response Prompt

Explain in concise steps how this logically works

Response:

track Response Time: When a player answers, the response time is calculated from when the question was broadcasted. Calculate Points: The `calculatePoints()` method assigns: 20 points if answered within 10 seconds. 10 points if answered between 10 and 30 seconds. 5 points if answered after 30 seconds.

points if answered within 30 seconds. 5 points if answered after 30 seconds.

Sub-Task

Group: Round End

Task #2: Points

Sub Task #2: Show the code related to syncing points to the Clients

100%

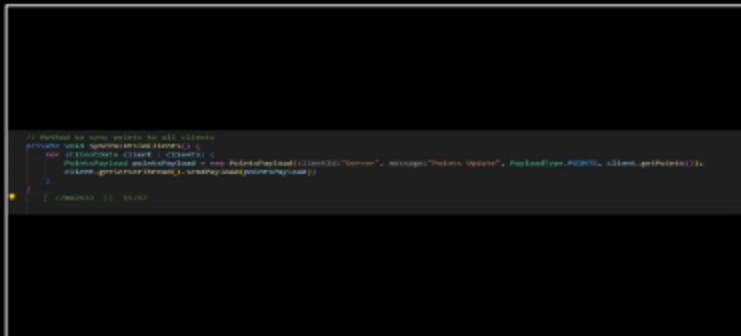
Task Screenshots

Gallery Style: 2 Columns

4

2

1



```
// Iterates through all clients in the game room
private void syncPointsToClients() {
    for (Client client : clients) {
        // Create a new message to send to the client
        PointUpdateMessage message = new PointUpdateMessage(client.getPoints());
        client.sendMessage(message);
    }
}
```

code related to syncing points to the Clients

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

This method loops through all clients in the game room and sends their points to their respective server threads.

Sub-Task

Group: Round End

Task #2: Points

Sub Task #3: Show the code related to the Client updating their local player map

100%

Task Screenshots

Gallery Style: 2 Columns

4

2

1



```
for (Client client : clients) {
    if (client.isLocalPlayer()) {
        String response = client.getResponse();
        if (response != null) {
            String[] parts = response.split(" ");
            if (parts.length == 2) {
                int id = Integer.parseInt(parts[0]);
                int points = Integer.parseInt(parts[1]);
                LocalPlayer localPlayer = localPlayers.get(id);
                if (localPlayer != null) {
                    localPlayer.setPoints(points);
                }
            }
        }
    }
}

// Check for updates every 10 seconds
while (true) {
    Thread.sleep(10000);
    for (Client client : clients) {
        if (client.isLocalPlayer()) {
            String response = client.getResponse();
            if (response != null) {
                String[] parts = response.split(" ");
                if (parts.length == 2) {
                    int id = Integer.parseInt(parts[0]);
                    int points = Integer.parseInt(parts[1]);
                    LocalPlayer localPlayer = localPlayers.get(id);
                    if (localPlayer != null) {
                        localPlayer.setPoints(points);
                    }
                }
            }
        }
    }
}
```

Update the local player points map

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

The client receives a `PointsPayload` from the server, which contains the player's ID (`getClientId()`) and their updated points (`getPoints()`). The client updates its local map (`playerPoints`) using the player ID as the key and the points as the value:

Sub-Task

Group: Round End



Task #2: Points

Sub Task #4: Show the code related to the in-progress scoreboards

Task Screenshots

Gallery Style: 2 Columns

4 2 1

code related to the in-progress scoreboards

code related to the in-progress scoreboards

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

Method to Sync Points to All Clients:Sends updated points to all clients, ensuring everyone has the latest score information. This method is usually called at the end of each round to broadcast scores. **Method to Sync Points to a Specific Client:**Sends the points of all clients to a specific client, usually when a new client joins an in-progress game. This ensures that the newly joined client has the most up-to-date scoreboard. **Updating Local Player Map with Score Information:**When the client receives a PointsPayload from the server, it updates its local player points map (playerPoints). This keeps each client informed of the current scores during the game, allowing for an up-to-date in-progress scoreboard.

Sub-Task

Group: Round End



Task #2: Points

Sub Task #5: Show at least 3 examples of terminal output of the scoreboards

4	2	1
Client Name: Alice Player Server has 4 points. Notifications: Who is known as the Father of Genghis Khan? A. Charlemagne B. Alexander the Great C. Bill Gates D. Steve Jobs Answer: B Notification: Alice has locked in an answer Notification: Alice has locked in an answer Notification: Alice has locked in an answer Notification: Alice has locked in an answer Player Server has 4 points.	Client Name: Bob Player Server has 10 points. Notifications: Who is known as the Father of Genghis Khan? A. Charlemagne B. Alexander the Great C. Bill Gates D. Steve Jobs Answer: C Notification: Bob has locked in an answer Notification: Charlie has locked in an answer Notification: Alice has locked in an answer Notification: Alice has locked in an answer Player Server has 10 points. Notification: Alice has locked in an answer Notification: Alice has locked in an answer Notification: Alice has locked in an answer Notification: Alice has locked in an answer Player Server has 10 points.	Client Name: Charlie Player Server has 8 points. Notifications: What is the currency of Japan? A. Yen B. Mori C. Euro D. Dollar Answer: C Notification: Charlie has locked in an answer Notification: Charlie has locked in an answer Notification: Alice has locked in an answer Player Server has 8 points.

1examples

2examples

3
Player Server has 10 points. Question: What is the largest ocean on Earth? A. Atlantic B. Indian C. Arctic D. Pacific Answer: B Notification: Alice has locked in an answer Notification: Bob has locked in an answer Notification: Charlie has locked in an answer Notification: Alice has locked in an answer Player Server has 10 points. Question: What is the chemical symbol for water? A. H2O

3examples

Caption(s) (required) ✓Caption Hint: *Describe/highlight what's being shown***Task Response Prompt***Explain in concise steps how this logically works*

Response:

in the first picture only one got it right and the score update to 10 the second picture in the next question the also one client got it right score update to 10 the 3rd picture in the next question the also one client got it right score update to 10

each one of them only got one answer right

End of Task 2

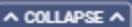
Task

Group: Round End

Task #3: Next Round or Session End

Weight: ~33%

Points: ~0.50

**Details:**

All code screenshots must have ucid/date visible.

Sub-Task

Group: Round End

Task #3: Next Round or Session End

Sub Task #1: Show the code related to triggering the next round or session end (end game condition)

100%

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
// Method to end the round
private void endRound() {
    roundTimer.cancel();
    roundTimer.schedule(new TimerTask() {
        @Override
        public void run() {
            if (currentRound > 5) { // Example condition for session end after 5 rounds
                startRound();
            } else {
                endSession();
            }
        }
    }, 0);
}
```

Method to end the round

Caption(s) (required) ✓Caption Hint: *Describe/highlight what's being shown***Task Response Prompt***Explain in concise steps how this logically works*

Response:

endRound() is called when a round ends. It either starts the next round or ends the session based on the game conditions.

End of Task 3

End of Group: Round End

Task Status: 3/3

Group

Group: Session End

Tasks: 3

Points: 1.25

100%

▲ COLLAPSE ▲

Task

Group: Session End

Task #1: Session End Condition

Weight: ~33%

100%

[▲ COLLAPSE ▲](#)**ⓘ Details:**

All code screenshots must have ucid/date visible.

**Sub-Task**

Group: Session End

100%

Task #1: Session End Condition

Sub Task #1: Show the code related to the session ending when X rounds have passed

☒ Task Screenshots

Gallery Style: 2 Columns

4

2

1

```

// Method to end the session
private void endSession() {
    System.out.println("Ending session round " + round);
    for (Client client : clients) {
        client.sendMessage(new Message("End Round", "Final Score", finalScore, client.getUniqueId()));
        client.getThread().endSession();
    }
}

```

Method to end the session

Caption(s) (required) ✓Caption Hint: *Describe/highlight what's being shown***≡ Task Response Prompt***Explain in concise steps how this logically works*

Response:

When the condition for ending the session is met, the `endSession()` method is called to send the final scores to all clients and reset the game.

End of Task 1

Task

100%

Group: Session End

Task #2: Scoreboards

Weight: ~33%

Points: ~0.42

[▲ COLLAPSE ▲](#)

Details:

All code screenshots must have ucid/date visible.
Any terminal output should have at least 3 clients involved.



Columns: 1

Sub-Task

Group: Session End

Task #2: Scoreboards

Sub Task #1: Show the code related to the final scoreboard

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```

// Method to send rich message
private void sendRichMessage()
{
    System.out.println("Rich message send.");
    // Create a rich message
    for (int i = 0; i < 10000000; i++)
    {
        RichMessage richMessage = new RichMessage();
        richMessage.setPayload("Hello", "World", "Server", "Client", "Java", "RabbitMQ", "POJO", client.getPayload());
        client.sendMessage(richMessage);
    }
    richMessage();
}

```

Send final scoreboard to all clients

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

This uses the PointsPayload to provide each player with their final points, labeled as "Final Score".

Sub-Task

Group: Session End

Task #2: Scoreboards

Sub Task #2: Show the final scoreboard from the terminal

Task Screenshots

Gallery Style: 2 Columns

4 2 1

final scoreboard from the terminal

Caption(s) (required) ✓

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

End of Task 2

Task

Group: Session End



Task #3: Cleanup

Weight: ~33%

Points: ~0.42

[▲ COLLAPSE ▲]

i Details:

All code screenshots must have ucid/date visible.
Any terminal output should have at least 3 clients involved.



Columns: 1

Sub-Task



Group: Session End

Task #3: Cleanup

Sub Task #1: Show the code related to resetting the Player data client-side and server-side (do not disconnect them or move them to the lobby)

☒ Task Screenshots

Gallery Style: 2 Columns

4 2 1

A screenshot of a terminal window displaying Java code. The code defines a private method named 'resetGame' that initializes 'currentRound' to 0 and clears the 'questionList'. A message is printed to the console indicating a new session is ready.

```
//ma2633 || 13/13

// Method to reset the game
private void resetGame() {
    currentRound = 0;
    questionList.clear();
    System.out.println("Game reset. Ready for a new session.");
}
```

Method to reset the game

Caption(s) (required) ✓

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

≡ Task Response Prompt

Explain in concise steps how this logically works

Response:

currentRound is reset to 0. The questionList is cleared, indicating that a new set of questions should be loaded for

the next game. The `answeredClients` list is cleared to remove tracking of answers from previous rounds.

Sub-Task

Group: Session End



Task #3: Cleanup

Sub Task #2: Show any terminal evidence

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Round is reset to 0

terminal evidence

Caption(s) (required) ✓

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

Sub-Task

Group: Session End



Task #3: Cleanup

Sub Task #3: Show the code related to shifting back to the ready Phase

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Method to shift all players back to ready phase

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

This method is used to shift all players back to a ready state, preparing them for a new session after the current game session ends. Reset Player Points: The method iterates through each client in the clients list and resets their points to zero using client.setPoints(0); Send Notification: A Payload object is created with the message "Game is ready for a new session" and a type of PayloadType.NOTIFICATION. This payload is then sent to each client to let them know they are ready for a new session. Log Message: Finally, a log message ("Players are shifted back to the ready phase.") is printed to indicate the operation has been completed.

End of Task 3

End of Group: Session End

Task Status: 3/3

Group

Group: Misc
Tasks: 3
Points: 1

100%

▲ COLLAPSE ▲

Task

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

100%

▲ COLLAPSE ▲

ⓘ Details:

Note: the link should end with /pull/#



🔗 Task URLs

URL #1

<https://github.com/Mohamad4322/ma2633-114-003/>

URL

<https://github.com/Mohamad4322/ma2633-114-003/>

End of Task 1

Task

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

100%

▲ COLLAPSE ▲

Task Response Prompt

Response:

what I found challenging was the processData() function, which involved multiple nested loops and conditional checks. At first, it was hard to follow the logic because each condition affected different variables in subtle ways. I had to carefully go through each line to understand how data flowed through the function. Working through it helped me appreciate the importance of structuring methods clearly, and I learned that breaking down complex logic into smaller helper functions could make code much easier to read and debug.

End of Task 2

Task



Group: Misc

Task #3: WakaTime Screenshot

Weight: ~33%

Points: ~0.33

[▲ COLLAPSE ▲](#)

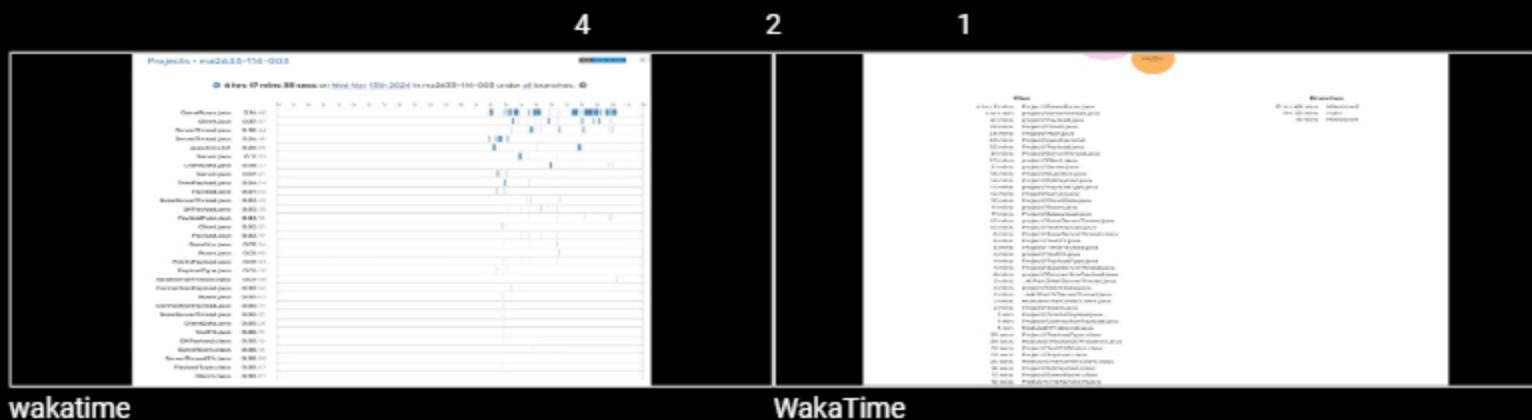
Details:

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



End of Task 3

End of Group: Misc

Task Status: 3/3

End of Assignment

