Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-450-M2024/it114-module-4-sockets-part-1-3/grade/jah89

IT114-450-M2024 - [IT114] Module 4 Sockets Part 1-3

Submissions:

Submission Selection

1 Submission [active] 6/14/2024 11:33:20 AM

Instructions

COLLAPSE

Overview Video: https://youtu.be/5a5HL0n6jek

1. Create a new branch for this assignment
2. If you haven't, go through the socket lessons and get each part implemented (parts 1-3)
    1. You'll probably want to put them into their own separate folders/packages (i.e., Part1, Part2, Part3) These are for your reference
3. Part 3, below, is what's necessary for this HW
    3. https://github.com/MattToegel/IT114/tree/M24-Sockets-Part3
4. Create a new folder called Part3HW (copy of Part3)
5. Make sure you have all the necessary files from Part3 copied here and fix the package references at the top of each file
    1. Add/commit/push the branch
    2. Create a pull request to main and keep it open
6. Implement **two** of the following **server-side** activities for all connected clients (majority of the logic should be processed server-side and broadcasted/sent to all clients if/when applicable)
    1. Simple number guesser where all clients can attempt to guess while the game is active
        1. Have a /start command that activates the game allowing guesses to be interpreted
        2. Have a /stop command that deactivates the game, guesses will be treated as regular messages (i.e., guess messages are ignored)
        3. Have a /guess command that include a value that is processed to see if it matches the hidden number (i.e., */guess 5*)
            1. Guess should only be considered when the game is active
            2. The response should include who guessed, what they guessed, and whether or not it was correct (i.e., Bob guessed 5 but it was not correct)
            3. No need to implement complexities like strikes
    2. Coin toss command (random heads or tails)

    1. Command should be something logical like /flip or /toss or /coin or similar
    2. The result should mention *who* did *what* and got what *result* (i.e., Bob Flipped a coin and got heads)
3. Dice roller given a command and text format of "/roll #d#" (i.e., /roll 2d6)
    1. Command should be in the format of /roll #d# (i.e., /roll 1d10)
    2. The result should mention *who* did *what* and got what *result* (i.e., Bob rolled 1d10 and got 7)
4. Math game (server outputs a basic equation, first person to guess it correctly gets congratulated and a new equation is given)
    1. Have a /start command that activates the game allowing equaiton to be answered
    2. Have a /stop command that deactivates the game, answers will be treated as regular messages (i.e., any game related commands when stopped will be ignored)
    3. Have an answer command that include a value that is processed to see if it matches the hidden number (i.e., */answer 15*)
        1. The response should include who answered, what they answered, and whether or not it was correct (i.e., Bob answered 5 but it was not correct)
5. Private message (a client can send a message targetting another client where only the two can see the messages)
    1. Command can be /pm, /dm followed by the user's name or an @ preceding the users name (clearly note which)
    2. The server should properly check the target audience and send the response to the original sender and to the receiver (no one else should get the message)
    3. Alternatively (make note if you do this and show evidence) you can add support to private message multiple people at once. Evidence should show a larger number of clients than the target list of the private message to show it works. Note to grader: if this is accomplished add 0.5 to total final grade on Canvas
6. Message shuffler (randomizes the order of the characters of the given message)
    1. Command should be /shuffle or /randomize (clearly mention what you chose) followed by the message to shuffle (i.e., /shuffle hello everybody)
    2. The message should be sent to all clients showing it's from the user but randomized
        1. Example: Bob types */command* hello and everyone recevies Bob: lleho
7. Fill in the below deliverables
8. Save the submission and generated output PDF
9. Add the PDF to the Part3HW folder (local)
10. Add/commit/push your changes
11. Merge the pull request
12. Upload the same PDF to Canvas

Branch name: M4-Sockets3-Homework

Tasks: 6 Points: 10.00

Baseline (2 pts.)

^COLLAPSE^

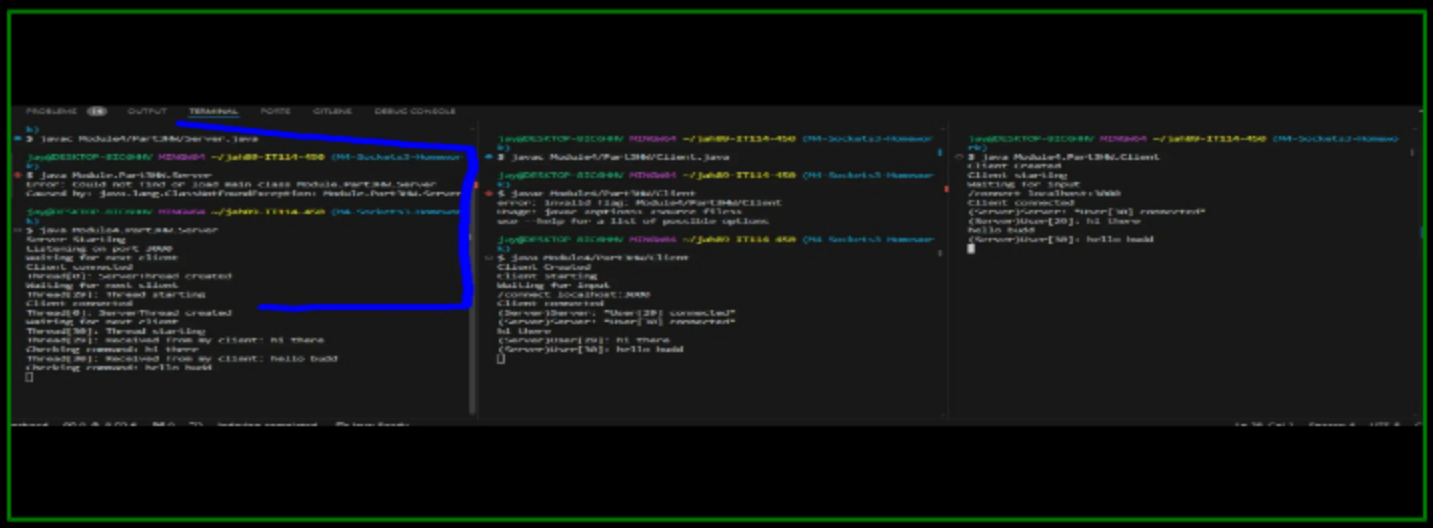
^COLLAPSE^

Task #1 - Points: 1

Text: Demonstrate Baseline Code Working

Details:
This can be a single screenshot if everything fits, or can be multiple screenshots

#1) Show and clearly note which terminal is the Server

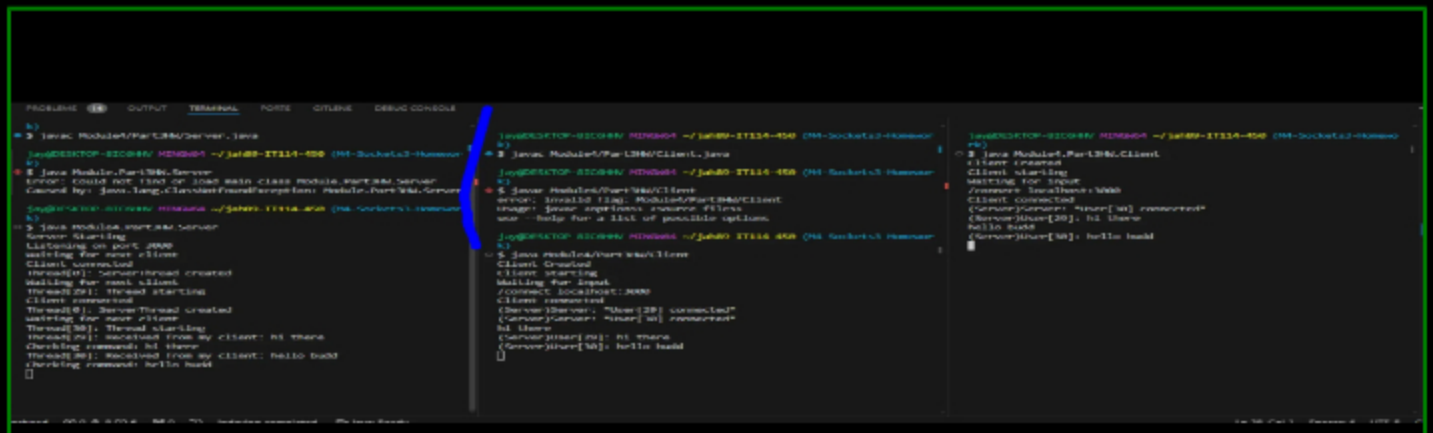


Caption (required) ✓

Describe/highlight what's being shown

Showing the left most terminal as the server

#2) Show and clearly note which terminals are the client

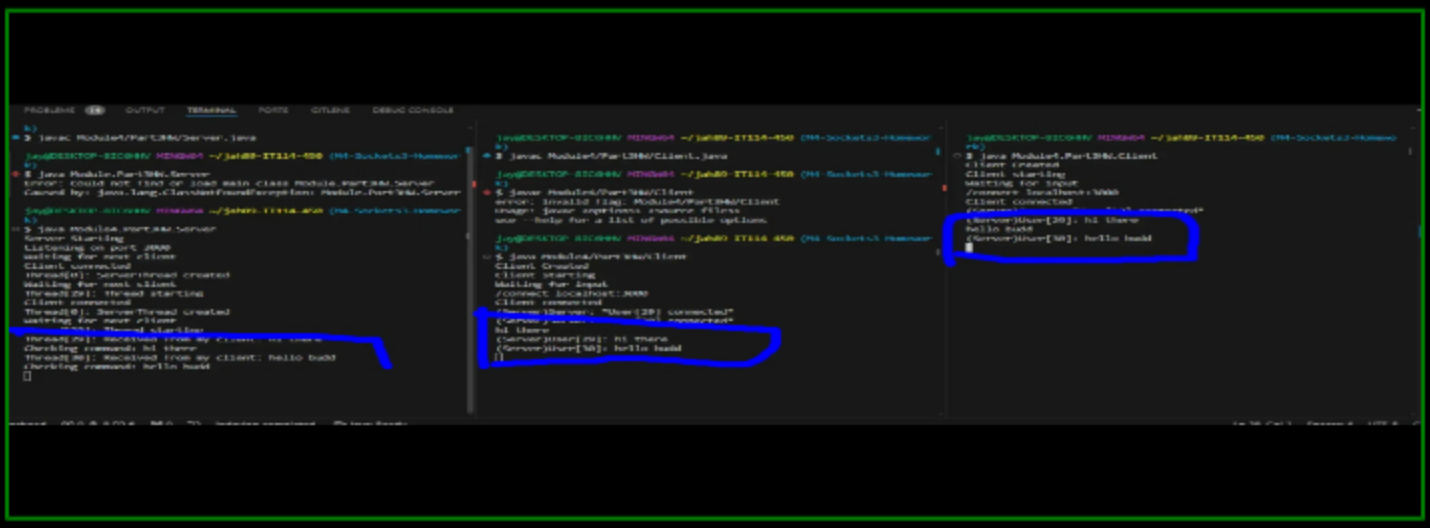


Caption (required) ✓

Describe/highlight what's being shown

Showing the middle and right side as clients

#3) Show all clients receiving the broadcasted/relayed messages



Caption (required) ✓

Describe/highlight what's being shown

Showing messages relayed correctly

#4) Include a screenshot showing you grabbed Parts 1-3 correctly and have them in your repository alongside Part3HW

Caption (required) ✓

Describe/highlight what's being shown

Showing my files

Feature 1 (3 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Solution

#1) Show the code related to the feature (ucid and date must be present as a comment)

Caption (required) ✓

Describe/highlight what's being shown

Showing the 2 sections where I added code to implement coin toss

Explanation (required) ✓

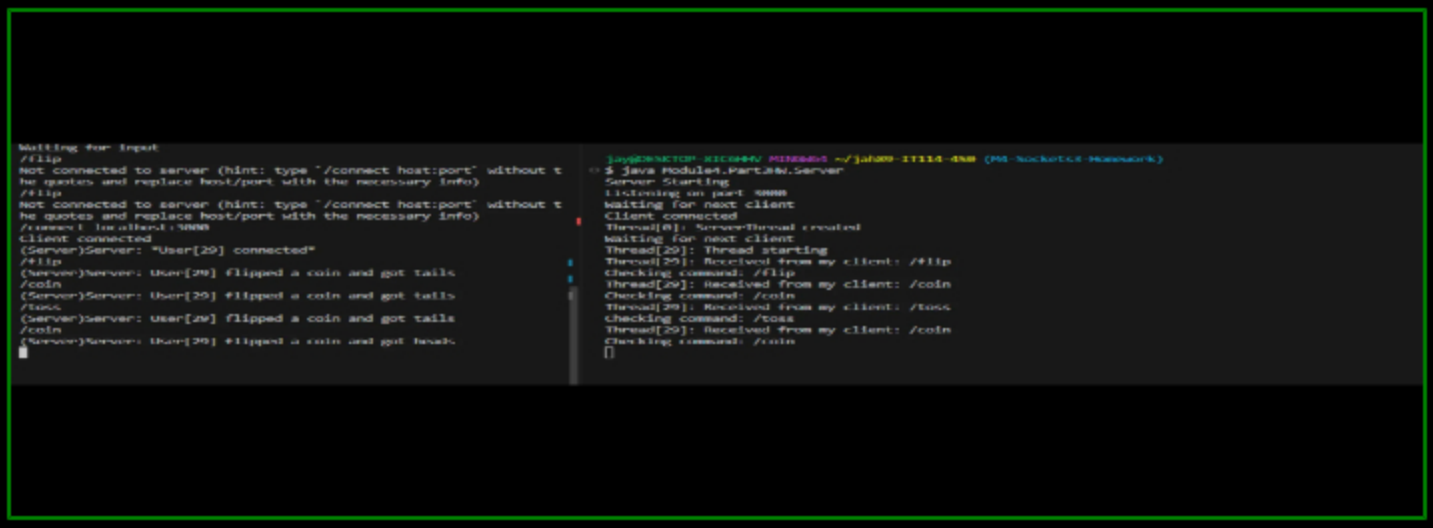
Mention specific feature and explain sufficiently and concisely the implementation (should be aligned with code snippets)

PREVIEW RESPONSE

For the code I added a method called processCoinToss. For this code I used protected void so it can be accessed by everything inside the M4Part3HW package. I have it take one parameter "sender" which was apart of the ServerThread to represent that the client was the one who started the coin toss. I then had to use the random class and used nextBoolean to generate a random true or false result. I used the ? and : operators to assign heads if the random generator brings back true and tails if its false. Then I had to make the message to send out and used %s as a placeholder for the User and the output of true or false. I then used send.Getclientid() to put the client id in the first %s position which is assigned to User[] and then i have "result" to fill in last placeholder with the true or false result. Finally after all this I used the relay method to send the message to the connected clients.

For the second snippet of code I added an else if to the processCommand method. In this method I added it checks if the message the client inputs is /flip or /toss or /coin it will send them to the processCoinToss method where it will go through the code I explained above. I used IgnoreCase so the user doesnt have to worry about typing these in all lowercase etc. I call processCoinToss(sender) because I wanted to make sure I was passing the sender object to show the client sent the command. I then closed off the else if with return true to show it was sucessful.

#2) Show the feature working (i.e., all terminals and their related output)



Caption (required) ✓

Describe/highlight what's being shown

Showing coin being flipped in terminal

Feature 2 (3 pts.)

^COLLAPSE^

^COLLAPSE^

Task #1 - Points: 1

Text: Solution

#1) Show the code related to the feature (ucid and date must be present as a comment)

```
    //jah89 06/17/2024
} private String shuffleMessage(String message) {
    char[] characters = message.toCharArray();
    Random random = new Random();
    for (int i = 0; i < characters.length; i++) {
        int randomIndex = random.nextInt(characters.length);
        char temp = characters[i];
        characters[i] = characters[randomIndex];
        characters[randomIndex] = temp;
    }
    String shuffled = new String(characters);
    System.out.println("Shuffling result: " + shuffled);
    return shuffled;
}
```

```
    //jah89 06/17/2024
} private String shuffleMessage(String message) {
    char[] characters = message.toCharArray();
    Random random = new Random();
    for (int i = 0; i < characters.length; i++) {
        int randomIndex = random.nextInt(characters.length);
        char temp = characters[i];
        characters[i] = characters[randomIndex];
        characters[randomIndex] = temp;
    }
    String shuffled = new String(characters);
    System.out.println("Shuffling result: " + shuffled);
    return shuffled;
}
```

Caption (required) ✓

Describe/highlight what's being shown

Showing code written to shuffle message

Showing code written to shuffle message

Explanation (required) ✓

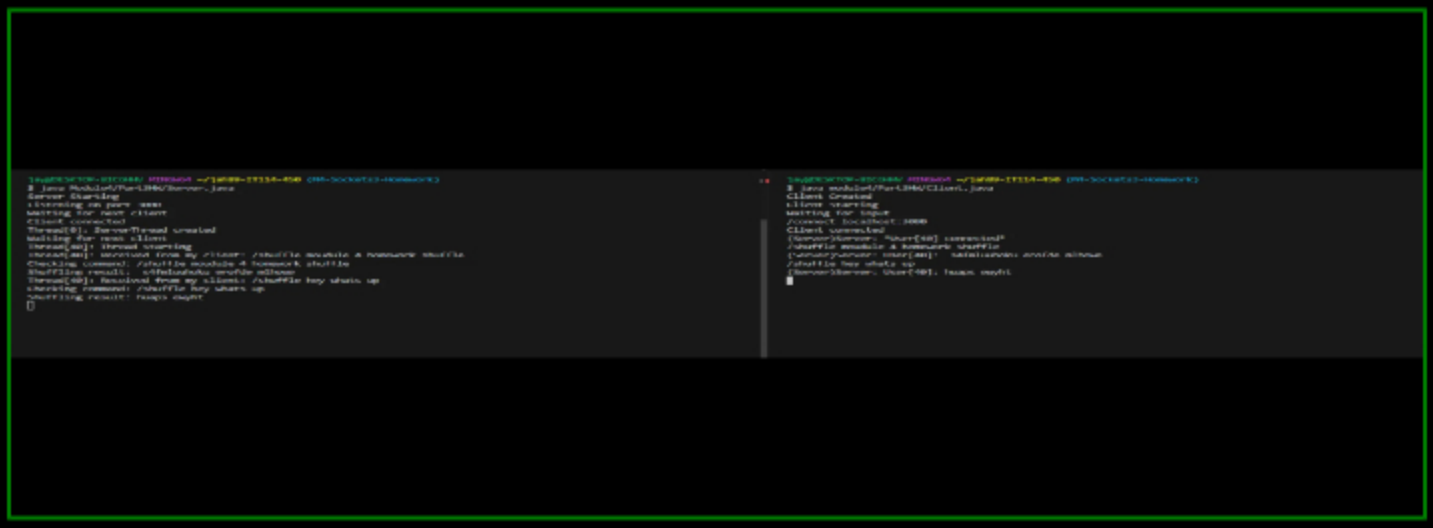
Mention specific feature and explain sufficiently and concisely the implementation (should be aligned with code snippets)s

PREVIEW RESPONSE

For the shuffleMessage method I have it take a string as an input and return a new string with the characters shuffled in a random order. The first thing I did was convert the string into an array using toCharArray. I did this so its easier to see the characters individually and can be changed a lot easier. I added a random object to generate random indexes. I wrote a for-loop to iterate over each character in the array. In this I have it so the current character index is swapped with the random index from the random object. After this is complete I then convert the array back into a string using new String(characters). I then used the relay method to broadcast the message to all clients connected. I added a debug statement because when I was first testing the code it wasn't working correctly and wanted to see where the code was having trouble.

For the next snippet of code I wrote I added to the processCommand method. I split the message into two parts to show command itself used and the message to be shuffled. I did this by using message.split(" ", 2) which will split the string at the first space. I had to do some research on this because I was having trouble at first. Once it is split its put into the array called parts. I then wrote an if statement that checks if parts length is 2 to make sure its formatted correctly and if it isn't they will be sent to the else where an error message displays. I then called the shuffleMessage and set it to parts[1] to prepare it to be shuffled. Then the result of the shuffleMessage is stored in the variable "shuffledMessage" and the orginal message is stored in "originalMessage" variable. I then use the relay method to broadcast the resultMessage to all the clients.

#2) Show the feature working (i.e., all terminals and their related output)



Caption (required) ✓

Describe/highlight what's being shown

Showing output of shuffling message in terminal

Misc (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Reflection

#1) Learn anything new? Face any challenges? How did you overcome any issues?

Explanation (required) ✓

Provide at least a few logical sentences

PREVIEW RESPONSE

I learned a lot of new things in this weeks module. I never dealt with server and clients before so it was very interesting and fun to learn and test the code out. My first challenge came when trying to figure out where to implement my code and in what method to add code to get it to work. I also had an issue when I first wrote the code for the coin flip I was trying to do server.processCoinToss(this). This brought an error because "server" was not recognized from the Server class, and this is because it didn't have a server variable, only the ServerThread.java had a server variable, but since they were in same package I thought it would still be fine and the server.java would still find the server variable but it didn't. This is where I learned that variables declared within a class are not accessible directly by another class, even if they are in the same package. So instead i used this.processCoinToss(sender), by using "this" I explicitly called the processCoinToss method in the current instance of the "Server" class.
I also had trouble with the shuffling the message I had to do a lot of research to understand how to implement it. I had many times where it wouldn't work and after some documentation help I was able to learn how to shuffle characters in a string message. I learned that it's alot easier to split things into an array to change things more indivudually.

COLLAPSE

Task #2 - Points: 1

Text: Pull request link

Details:
URL should end with /pull/# and be related to this assignment

URL #1
https://github.com/jah89/jah89-IT114-450/pull/12

COLLAPSE

Task #3 - Points: 1

Text: Waka Time (or related) Screenshot

Details:
Screenshot clearly shows what files/project were being worked on (the duration of time doesn't correlated with the grade for this item)

Task Screenshots:

Gallery Style: Large View

Small Medium Large

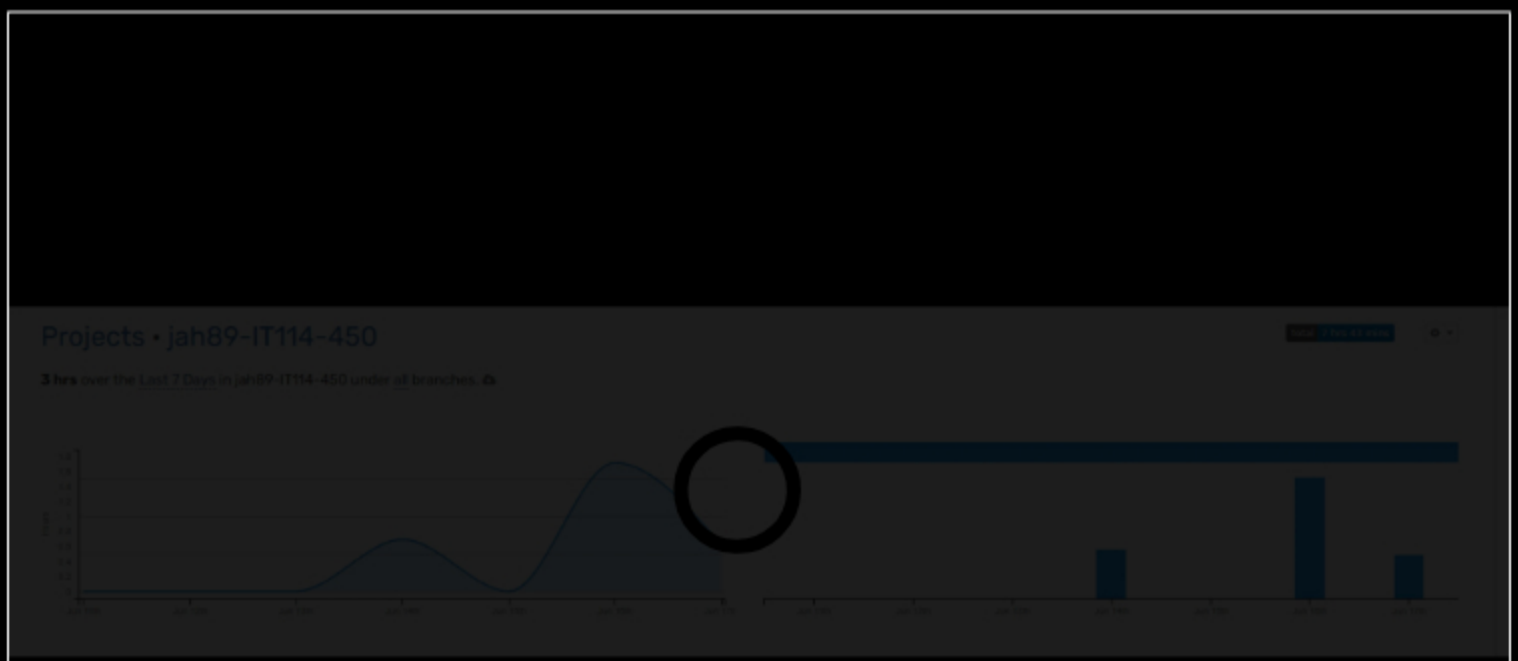
Files

1 hr 46 mins	Module4/Part3HW/Server.java
34 mins	...4/Part3HW/ServerThread.java
17 mins	Module4/Part3HW/Client.java
6 mins	Module4/Client.java
4 mins	Module4/Part1
2 mins	Module4/Part1/Client.java

Branches

3 hrs	M4-Sockets3-Homework
1 sec	Unknown

Showing how long I worked on the files and my m4 branch



Showing time

End of Assignment