Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-002-S2024/it114-sockets-part-1-3-checkpoint/grade/ob75

IT114-002-S2024 - [IT114] Sockets Part 1-3-Checkpoint

Submissions:

Submission Selection

1 Submission [active] 2/18/2024 3:27:01 PM

Instructions

COLLAPSE

1.
2. Create a new branch for this assignment
   Go through the socket lessons and get each part implemented (parts 1-3)
   2. You'll probably want to put them into their own separate folders/packages (i.e., Part1, Part2,
   3. Part3) These are for your reference
3. Part 3, below, is what's necessary for this HW
4. https://github.com/MattToegel/IT114/tree/Module4/Module4/Part3
   Create a new folder called Part3HW (copy of Part3)
   Make sure you have all the necessary files from Part3 copied here and fix the package references at the top of each file
5. Add/commit/push the branch
   Create a pull request to main and keep it open
   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)
   - 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)
     4. No need to implement complexities like strikes
   - 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)
   - 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)
   - 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*)

5. 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)

Private message (a client can send a message targetting another client where only the two can see the messages)

2. Command can be /pm, /dm followed by the user's name or an @ preceding the users name (clearly note which)
3. 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)
6. 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

Message shuffler (randomizes the order of the characters of the given message)

2. Command should be /shuffle or /randomize (clearly mention what you chose) followed by the message to shuffle (i.e., /shuffle hello everybody)
6. The message should be sent to all clients showing it's from the user but randomized
7. Example: Bob types */command* hello and everyone recevies Bob: lleho

8. Fill in the below deliverables
9. Save the submission and generated output PDF
10. Add the PDF to the Part3HW folder (local)
11. Add/commit/push your changes

Merge the pull request

Upload the same PDF to Canvas

Branch name: M4-Sockets3-Homework

Tasks: 7 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

Checklist

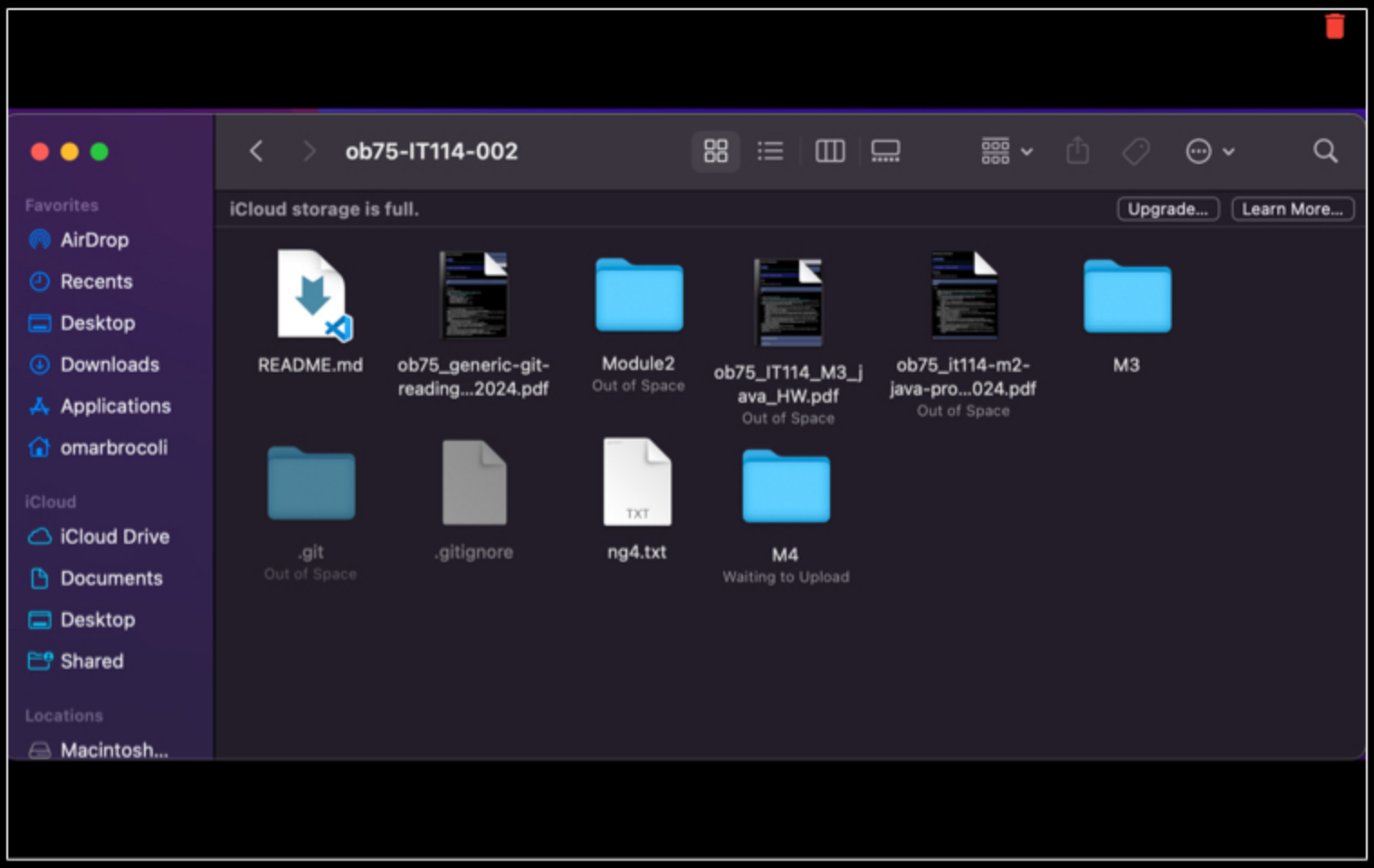
*The checkboxes are for your own tracking

#	Points	Details
#1	1	Server terminal/instance is clearly shown/noted
#2	1	At least 3 client terminals should be visible and noted
#3	1	Each client should correctly receive all broadcasted/shared messages
#4	1	Captions clearly explain what each screenshot is showing
#5	1	Include a screenshot showing you grabbed Parts 1-3 correctly and have them in your repository alongside Part3HW

Task Screenshots:

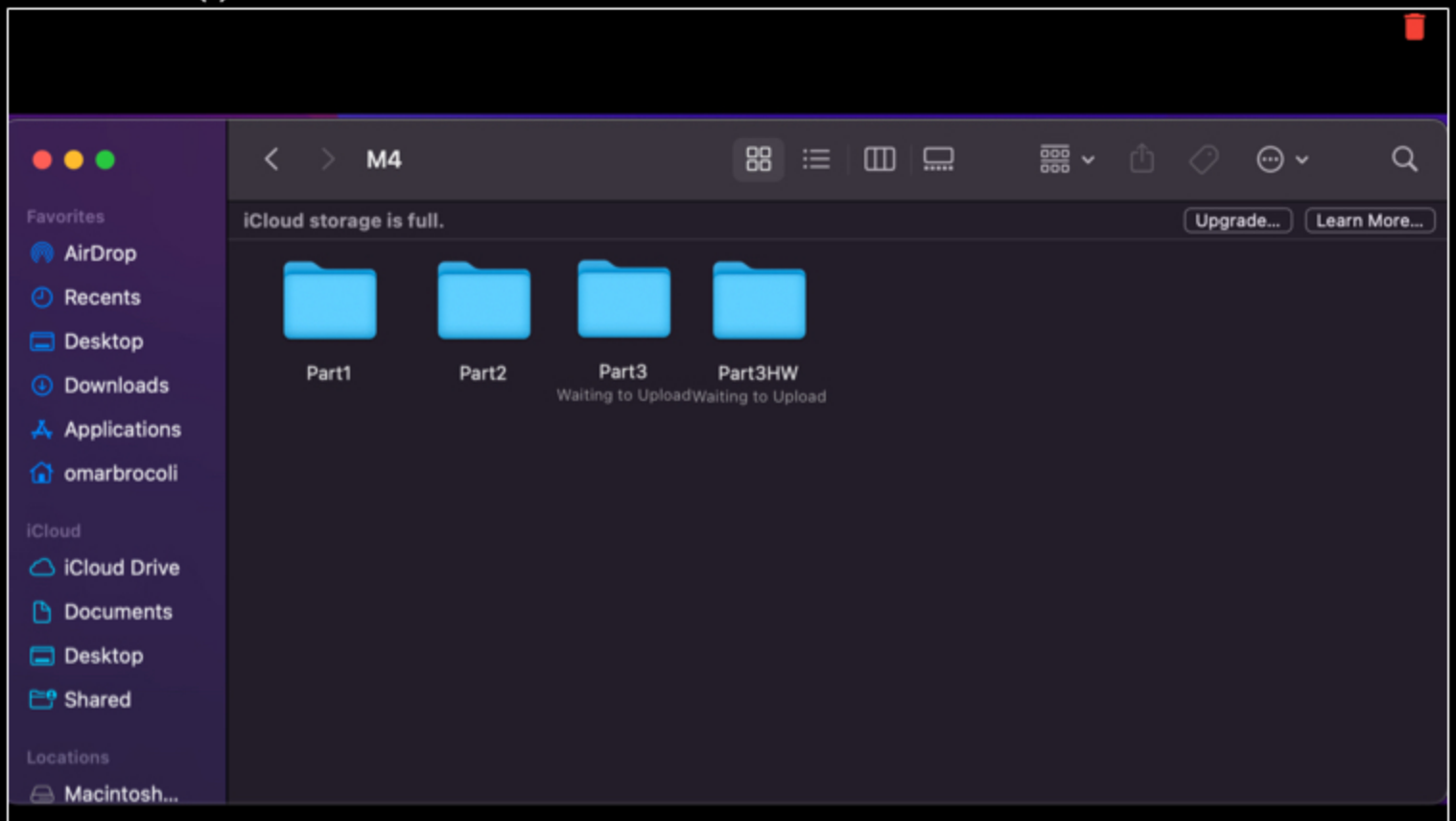
Gallery Style: Large View

Small Medium Large



Showing repository folder with the M4 folder on the bottom right containing parts 1-3 and part3HW.

Checklist Items (0)



showing all part included in the M4 folder which Is in the repository.

Checklist Items (0)

showing baseline code working for part3 and the server as the first terminal window with the rest being the client window all connected and broadcasting messages.

Checklist Items (0)

Feature 1 (3 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: What feature did you pick? Briefly explain how you implemented it

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Feature is clearly stated (best to copy/paste it from above)
#2	1	Explanation sufficiently and concisely describes implementation (should be aligned with code snippets in related task)

Response:

My first implementation was the Coin toss command (random heads or tails). I created a code that

allows the client to type in the command "/toss" and it will start a coin toss and then give the result of the coin toss which is either heads or tails. I made a method first called coinToss that takes in the user id and will then execute the method. I made an if statement with math.random so the value is always random. I made it so if the random value is less than 0.5, then broadcast the message to everyone that they flipped a coin and got tails, else if that number is more, then do the same but instead of tails, its heads. I also included that method into processCommands so that anytime they type the command "/toss", it will start the method.

COLLAPSE

Task #2 - Points: 1

Text: Add screenshot(s) showing the implemented feature working (code and output)

Details:
Add screenshots of the relevant code changes AND relevant output during runtime

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Output is clearly shown and captioned
#2	1	Code shows relevant snippets that accomplish feature, UCID and date are present in all code screenshots. Relevant captions are included for each screenshot of the code.

Task Screenshots:

Gallery Style: Large View

Small Medium Large

My code is inside the processCommands at the end and also my method created under that block with both UCID signifying the start of my code.

Checklist Items (0)

Sever and 3 clients shown connected and tossing the coin when typing the command /toss.

Checklist Items (0)

Feature 2 (3 pts.)

COLLAPSE

Task #1 - Points: 1

Text: What feature did you pick? Briefly explain how you implemented it

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Feature is clearly stated (best to copy/paste it from above)
#2	1	Explanation sufficiently and concisely describes implementation (should be aligned with code snippets in related task)

Response:

My second implementation is Message shuffler (randomizes the order of the characters of the given message). I created code that would take the command "/shuffle" and any words after it, shuffle the words, and broadcast it to everyone in the server. This was done with a method that would convert the message into an array of characters first. Then I had a for loop that go through each character and change their indexes of where they are, thus shuffling them. Once the loop was done, I just converted the characters back to a string but now shuffled from its original state. Lastly I called this method in processCommands taking in "/shuffle" from the user and removing the shuffle command from being included in the shuffle of words. This allowed for the user to input the command and word they wanted to shuffle and would do the method.

COLLAPSE

Task #2 - Points: 1

Text: Add screenshot(s) showing the implemented feature working (code and output)

Details:
Add screenshots of the relevant code changes AND relevant output during runtime

Checklist

*The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Output is clearly shown and captioned
☐ #2	1	Code shows relevant snippets that accomplish feature, UCID and date are present in all code screenshots. Relevant captions are included for each screenshot of the code.

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
91          // ob75 - Febraury 17, 2024
92      } else if (message.startsWith(prefix:"/shuffle")) {
93          String b = message.replace(target:"/shuffle", replacement:"");
94          String c = messageShuffler(b);
95          broadcast(c, clientId);
96          return true;
97      }
98
99      return false;
100 }
101
102 // ob75 - Febraury 17, 2024
103 private String messageShuffler(String message) {
104     String t = message;
105     char[] ch = t.toCharArray();
106
107     for (int i = 0; i < ch.length; i++) {
108         char p = ch[i];
109         int newIndex = (int) Math.floor(Math.random() * (ch.length - 1));
110         ch[i] = ch[newIndex];
111         ch[newIndex] = p;
112     }
113
114     String b = new String(ch);
```

```
115
116         return b;
117
118     }
```

UCID marks the start of my code and included my method as well for messageShuffler. Invoking the method is the top of the screenshot. First goes processing the command with my method then shows my method under.

Checklist Items (0)

shows 3 clients connect to server and typing the command shuffle and the word they want to shuffle. the output is broadcasted and shown as shuffled.

Checklist Items (0)

Misc (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Reflection: Did you have an issues and how did you resolve them? If no issues, what did you learn during this assignment that you found interesting?

Checklist

*The checkboxes are for your own tracking

#	Points	Details

#1	1	An issue or learning is clearly stated
#2	1	Response is a few reasonable sentences

Response:

Main issues came from trying to create my second implementation of shuffling words. I could not figure out how to shuffle the strings inputed so that it comes out different from it original state. I solved this by converting them into an array of characters first, the creating the loop at which they would shuffle and finally convert them back to string. Another issue was with trying to get "/shuffle" to not be included for when shuffling words as it kept shuffling the command as well. I resolved this issue by checking first if the command was typed and then removing the command from the message being shuffled and only allowed for the message after the command to be shuffled.

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/OmarBarrera1/ob75-IT114-002/pull/5

End of Assignment