Submission Worksheet

Submission Data

Course: IT114-005-F2025
Assignment: IT114 Module 4 Sockets Part3 Challenge
Student: Nilkanth D. (nhd5)
Status: Submitted | **Worksheet Progress:** 99%
Potential Grade: 9.82/10.00 (98.20%)
Received Grade: 0.00/10.00 (0.00%)
Started: 10/26/2025 5:48:30 PM
Updated: 10/26/2025 8:04:42 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-005-F2025/it114-module-4-sockets-part3-challenge/grading/nhd5
View Link: https://learn.ethereallab.app/assignment/v3/IT114-005-F2025/it114-module-4-sockets-part3-challenge/view/nhd5

Instructions

- Overview Link: https://youtu.be/_029E_aBTFo

1. Ensure you read all instructions and objectives before starting.
2. Create a new branch from `main` called `M4-Homework`
    1. `git checkout main` (ensure proper starting branch)
    2. `git pull origin main` (ensure history is up to date)
    3. `git checkout -b M4-Homework` (create and switch to branch)
3. Copy the template code from here: GitHub Repository - M4 Homework
    - It includes Sockets `Part1`, `Part2`, and `Part3`. Put all into an `M4` folder or similar if you don't have them yet (adjust `package` reference at the top if you chose a different folder name).
    - Make a copy of `Part3` and call it `Part3HW`
        - [ ] Fix the `package` and `import` references at the top of each file in this new folder (Note: you'll only be editing files in `Part3HW`)
    - Immediately record to history
        - [ ] `git add .`
        - [ ] `git commit -m "adding M4 HW baseline files"`
        - [ ] `git push origin M4-Homework`
        - [ ] Create a Pull Request from `M4-Homework` to `main` and keep it open
4. Fill out the below worksheet
    - Each Problem requires the following as you work
        - [ ] Ensure there's a comment with your UCID, date, and brief summary of how the problem was solved
        - [ ] Code solution (add/commit periodically as needed)
        - [ ] Hint: Note how `/reverse` is handled
5. Once finished, click "Submit and Export"
6. Locally add the generated PDF to a folder of your choosing inside your repository folder and move it to Github
    1. `git add .`
    2. `git commit -m "adding PDF"`

3. `git push origin M4-Homework`
4. On Github merge the pull request from `M4-Homework` to `main`

7. Upload the same PDF to Canvas
8. Sync Local
    1. `git checkout main`
    2. `git pull origin main`

Section #1: (3 pts.) Challenge 1 - Coin Flip

Progress: 100%

Task #1 (3 pts.) - Implement a Coin Flip Command

Progress: 100%

Details:

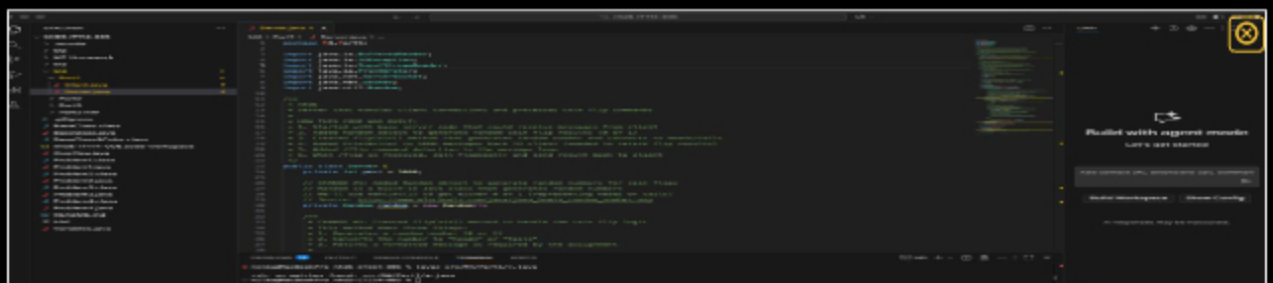
- `Client` must capture the user entry and generate a valid command per the lesson details
    - Command format must be `/flip`
- `ServerThread` must receive the data and call the correct method on `Server`
- `Server` must expose a method for the logic and send the result to everyone
    - The message must be in the format of `<who> flipped a coin and got <result>` and be from the Server
- Add code to solve the problem (add/commit as needed)

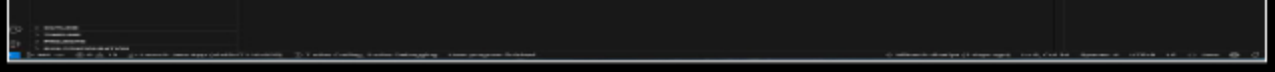
Part 1:

Progress: 100%

Details:
Multiple screenshots are expected

1. Snippet of relevant code showing solution (with ucid/date comment) from `Client`
    - Should only need to edit `processClientCommands()`
2. Snippet of relevant code showing solution (with ucid/date comment) from `ServerThread`
    - Should only need to edit `processCommand()`
3. Snippet of relevant code showing solution (with ucid/date comment) from `Server`
    - Should only need to create a new method and pass the result message to `relay()`
4. Show 5 examples of the command being seen across all terminals (2+ Clients and 1 Server)
    1. This can be captured in one screenshot if you split the terminals side by side

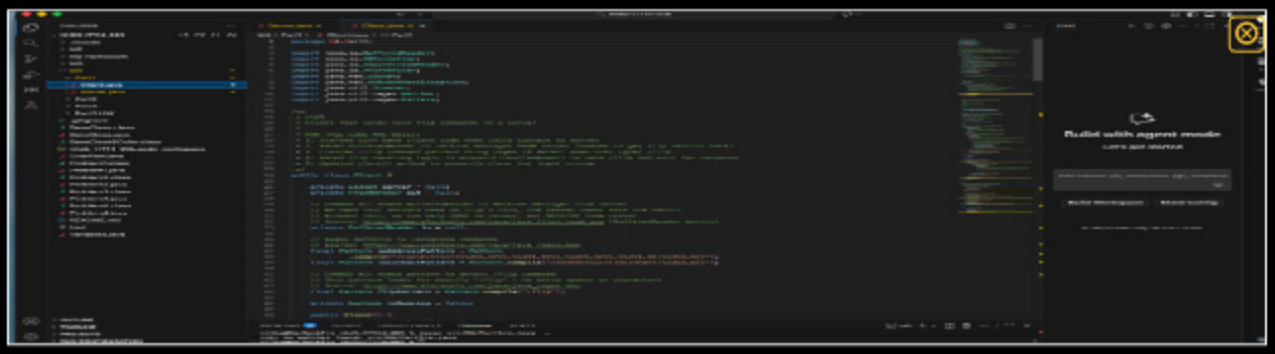




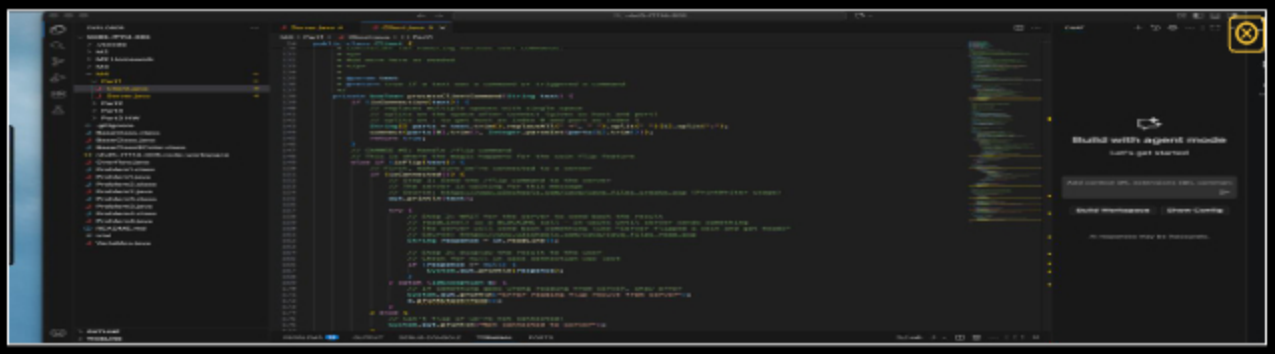
code snippet

code snippet

code snippet



code snippet



code snippet

Saved: 10/26/2025 5:53:40 PM

Part 2:

Progress: 100%

Details:
Direct link to the file in the homework related branch from Github (should end in `.java`)

URL #1
https://learn.ethereallab.app/assignment/v3/IT114-2025/M4/Part1

URL
https://learn.ethereallab.app/assi

Saved: 10/26/2025 5:53:40 PM

Part 3:

Progress: 100%

Details:
Briefly explain `how` the code solves the challenge (note: this isn't the same as `what` the code does)

Your Response:

The code achieves the challenge by having a persistent connection of a client to a server using sockets, whereby the client can enter commands and the server can echo back real-time. When entering /flip, the command is sent over the socket, and the server generates a random result that sends it back over the same channel. The exchange here back-and-forth is what provides interactivity to the coin flip feature.

Saved: 10/26/2025 5:53:40 PM

Section #2: (3 pts.) Challenge 2 - Private Message

Progress: 95%

Task #1 (3 pts.) - Implement a Private Message Command

Progress: 95%

Details:

- `Client` must capture the user entry and generate a valid command per the lesson details
  - Command format must be `/pm <target id> <message>`
- `ServerThread` must receive the data and call the correct method on `Server`
- `Server` must expose a method for the logic
  - The message must be in the format of `PM from <who>: <message>` and be from the Server
  - The result must only be sent to the original sender and to the receiver/target
- Add code to solve the problem (add/commit as needed)

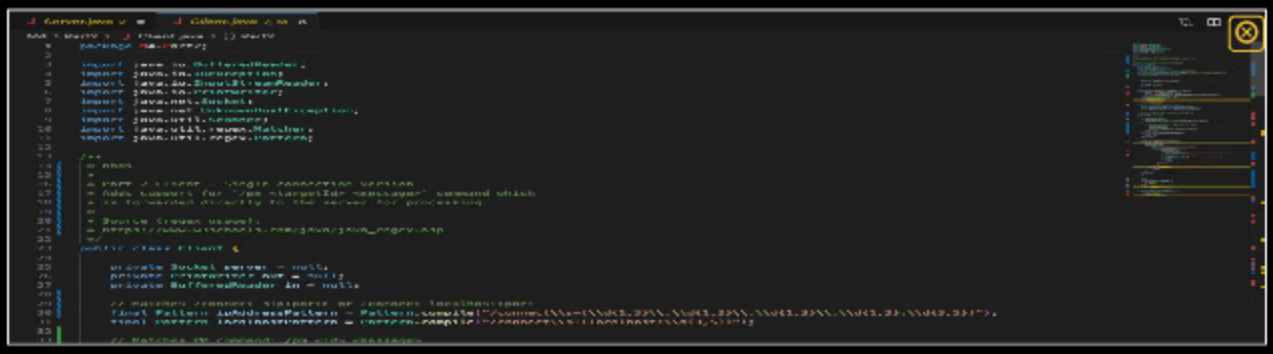
Add code to solve the problem (add/commit as needed)

Part 1:

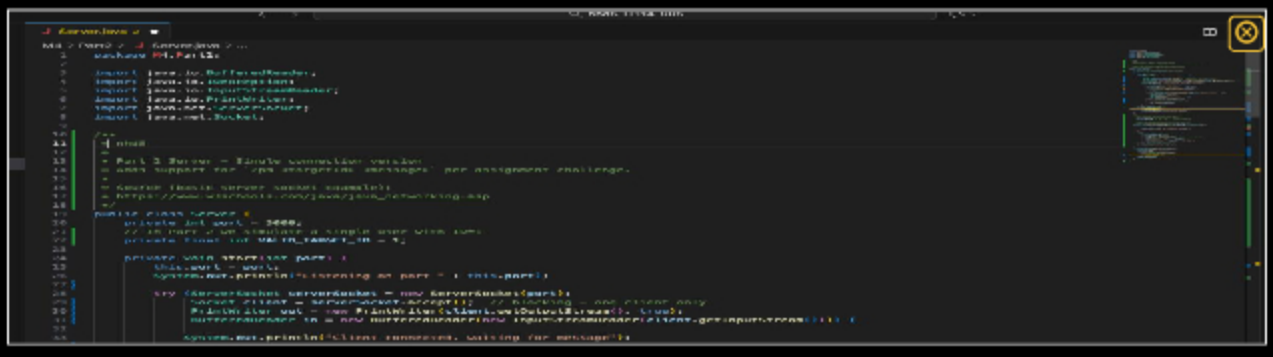
Progress: 85%

Details:
Multiple screenshots are expected

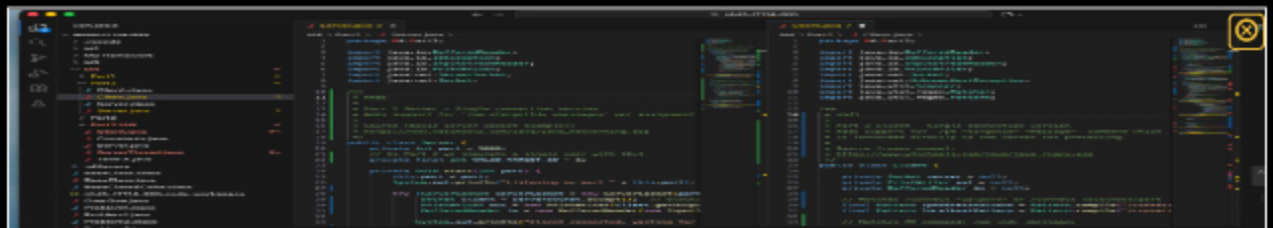
1. Snippet of relevant code showing solution (with ucid/date comment) from `Client`
   - Should only need to edit `processClientCommands()`
2. Snippet of relevant code showing solution (with ucid/date comment) from `ServerThread`
   - Should only need to edit `processCommand()`
3. Snippet of relevant code showing solution (with ucid/date comment) from `Server`
   - Should only need to create a new method and send the result message to just the sender and receiver
4. Show 3 examples of the command being seen across all terminals (3+ Clients and 1 Server)
   1. This can be captured in one screenshot if you split the terminals side by side
   2. Note: Only the sender and the receiver should see the private message (show variations across different users)



code sinppet



code sinppet



code sinppet

code sinppet

code sinppet

Saved: 10/26/2025 6:46:03 PM

Part 2:

Progress: 100%

Details:
Direct link to the file in the homework related branch from Github (should end in `.java`)

URL #1
https://learn.ethereallab.app/assignment/v3/IT11[illegible]/M4/Part2

URL
https://learn.ethereallab.app/assi

Saved: 10/26/2025 6:46:03 PM

Part 3:

Progress: 100%

Details:
Briefly explain `how` the code solves the challenges (note: this isn't the same as `what` the code does)

Your Response:

The code for dealing with the private message challenge is done by defining the /pm command on the client side with regex so that the client can format and send the correct message to the server. Once the server has received the command, it picks up the target ID and the message, and checks whether the ID is valid or not before sending a formatted response back. This flow ensures the message is sent to only the proper recipient (fictionally as ID 1 in Part 2), which aligns with the logic needed for a basic private messaging feature and delivers the resources to add multi-user support later in Part 3.

Saved: 10/26/2025 6:46:03 PM

Section #3: (3 pts.) Challenge 3 - Shuffle Message

Progress: 100%

Task #1 (3 pts.) - Implement a Shuffle Message Command

Progress: 100%

Details:

- `Client` must capture the user entry and generate a valid command per the lesson details
  - Command format must be `/shuffle <message>`
- `ServerThread` must receive the data and call the correct method on `Server`
- `Server` must expose a method for the logic and send the result to everyone
  - The message must be in the format of `Shuffled from <who>: <shuffled_message>` and be from the Server
- Add code to solve the problem (add/commit as needed)

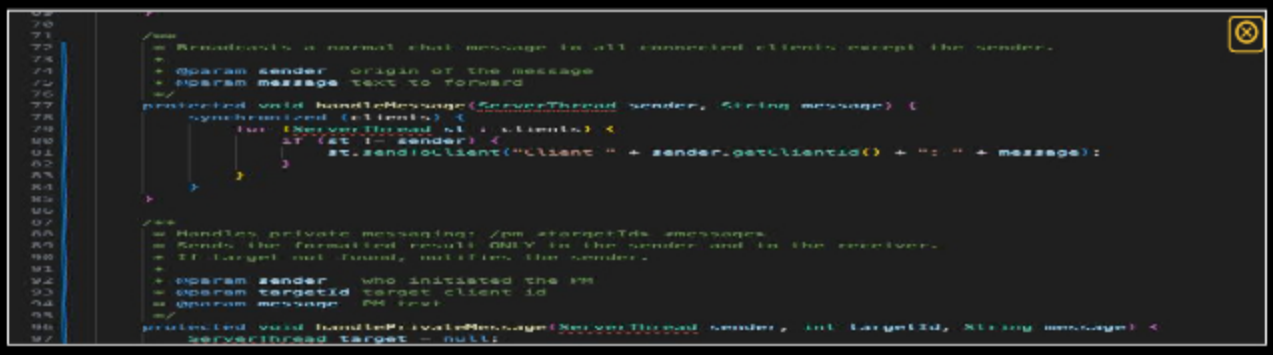
Part 1:

Progress: 100%

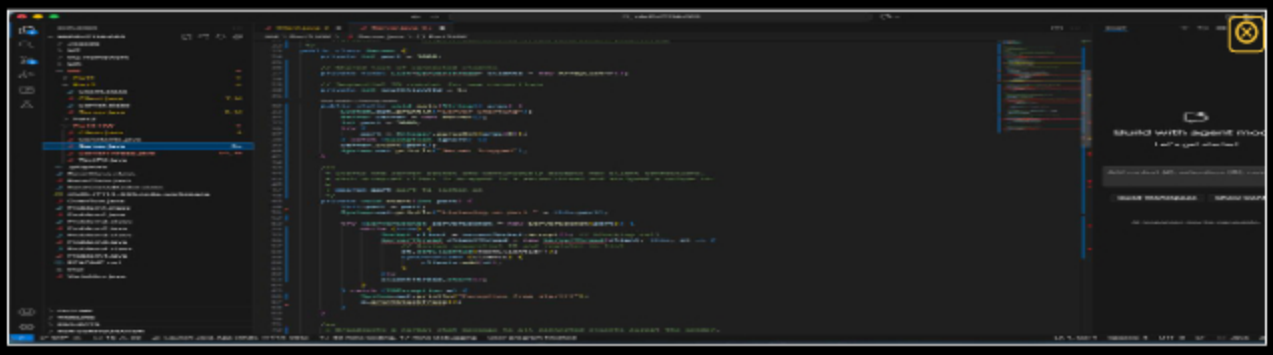
Details:
Multiple screenshots are expected

1. Snippet of relevant code showing solution (with ucid/date comment) from `Client`
   - Should only need to edit `processClientCommands()`
2. Snippet of relevant code showing solution (with ucid/date comment) from `ServerThread`
   - Should only need to edit `processCommand()`
3. Snippet of relevant code showing solution (with ucid/date comment) from `Server`
   - Should only need to create a new method and do similar logic to `relay()`
4. Show 3 examples of the command being seen across all terminals (2+ Clients and 1 Server)
   1. This can be captured in one screenshot if you split the terminals side by side

code snippet



code snippet



code snippet

Saved: 10/26/2025 7:08:44 PM

Part 2:

Progress: 100%

Details:
Direct link to the file in the homework related branch from Github (should end in `.java`)

URL #1
https://learn.ethereallab.app/assignment/v3/IT11[illegible]/M4/Part3%20HW

URL
https://learn.ethereallab.app/assi

Saved: 10/26/2025 7:08:44 PM

Part 3:

Progress: 100%

Details:
Briefly explain `how` the code solves the challenges (note: this isn't the same as `what` the code does)

Your Response:

The code solves the PM issue by giving a unique ID to each connected client and storing them in a shared list so that the server may refer to individual users and not broadcast the message to all of them. If a /pm command is encountered, the ServerThread reads the message and calls the server's private messaging procedure, which forwards the message to the sender and the recipient individually. The /who command enables clients to see all online IDs, therefore being capable of knowing who they can send private messages to.

Saved: 10/26/2025 7:08:44 PM

Section #4: (1 pt.) Misc

Progress: 100%

Task #1 (0.33 pts.) - Github Details

Progress: 100%

Part 1:

Progress: 100%

Details:
From the Commits tab of the Pull Request screenshot the commit history Following minimum should be present

pull request

Saved: 10/26/2025 7:15:35 PM

Part 2:

Progress: 100%

Details:
Include the link to the Pull Request (should end in `/pull/#`)

URL #1
https://github.com/Pixeldepth5/nhd5-IT11[illegible]/

URL
https://github.com/Pixeldepth5/n

Saved: 10/26/2025 7:15:35 PM

Task #2 (0.33 pts.) - WakaTime - Activity

Progress: 100%

Details:

- Visit the WakaTime.com Dashboard
- Click `Projects` and find your repository
- Capture the overall time at the top that includes the repository name
- Capture the individual time at the bottom that includes the file time
- Note: The duration isn't relevant for the grade and the visual graphs aren't necessary

WakaTIme Activity

Saved: 10/26/2025 7:17:06 PM

Task #3 (0.33 pts.) - Reflection

Progress: 100%

Task #1 (0.33 pts.) - What did you learn?

Progress: 100%

Details:
Briefly answer the question (at least a few decent sentences)

Your Response:

Your Response:

In this second part of the assignment, I learned how to make the shift from a simple single-client server to a true multi-client configuration using threads. I also learned how each connected user needs to have its own ServerThread so that the server can maintain clients individually and support functionality like private messaging. The second important thing I learned was how the server can store these clients in a shared list and use their individual IDs to forward messages to specific recipients instead of relaying all messages to all users.

Saved: 10/26/2025 8:01:00 PM

Task #2 (0.33 pts.) - What was the easiest part of the assignment?

Progress: 100%

Details:
Briefly answer the question (at least a few decent sentences)

Your Response:

The easiest part of this assignment was sending the user commands from client to server after connection had been established. Since the networking framework was already in place, integrating /pm and /who only required recognizing the command and passing the string along to the server for processing. Client-side handling was simple compared to the threading code because it was mostly a messenger without heavy logic necessary.

Saved: 10/26/2025 8:02:31 PM

Task #3 (0.33 pts.) - What was the hardest part of the assignment?

Progress: 100%

Details:
Briefly answer the question (at least a few decent sentences)

Your Response:

The most tricky part in the assignment was to manage multiple clients at the same time using threads and make sure that the server was able to send messages to only the intended receiver. Unlike Part 2, where there was only one client, Part 3 entailed managing all active connections and assigning a unique ID to each of them. Providing that the /pm command were sent to the sender and

receiver but not the /who command of all members required understanding of shared state on the server and communication thread-to-thread.

Saved: 10/26/2025 8:04:42 PM