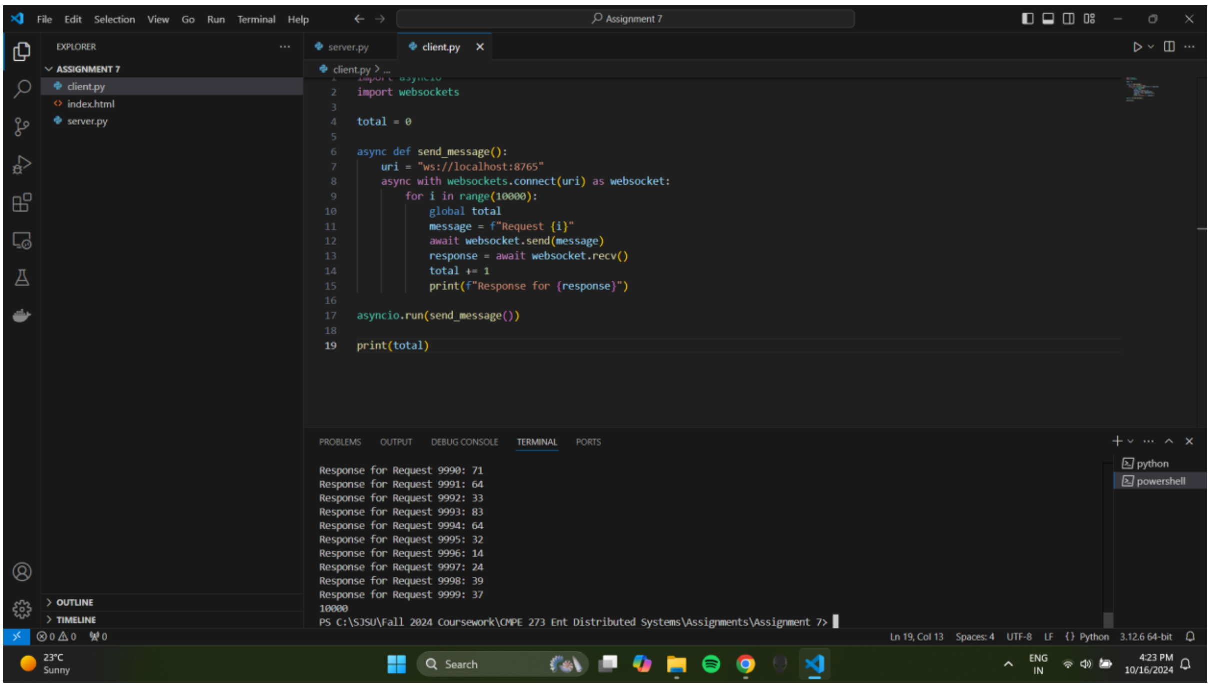
Web Socket

The objective was to implement a websocket-based communication system. There are three components in this project:

1. Server application (server.py): This application receives a request from any of the two clients, and handles them by attaching a random number between 1 and 100 to the end of the request message and returning the response.
2. Client application (client.py): This application generates 10K messages and sends them as requests to the client, and then handle the response by printing it.
3. Client UI (index.html): This is a UI-based application which sends a single request to the server on the press of a button and displays the response on the webpage.

The code for the same is available on Github.

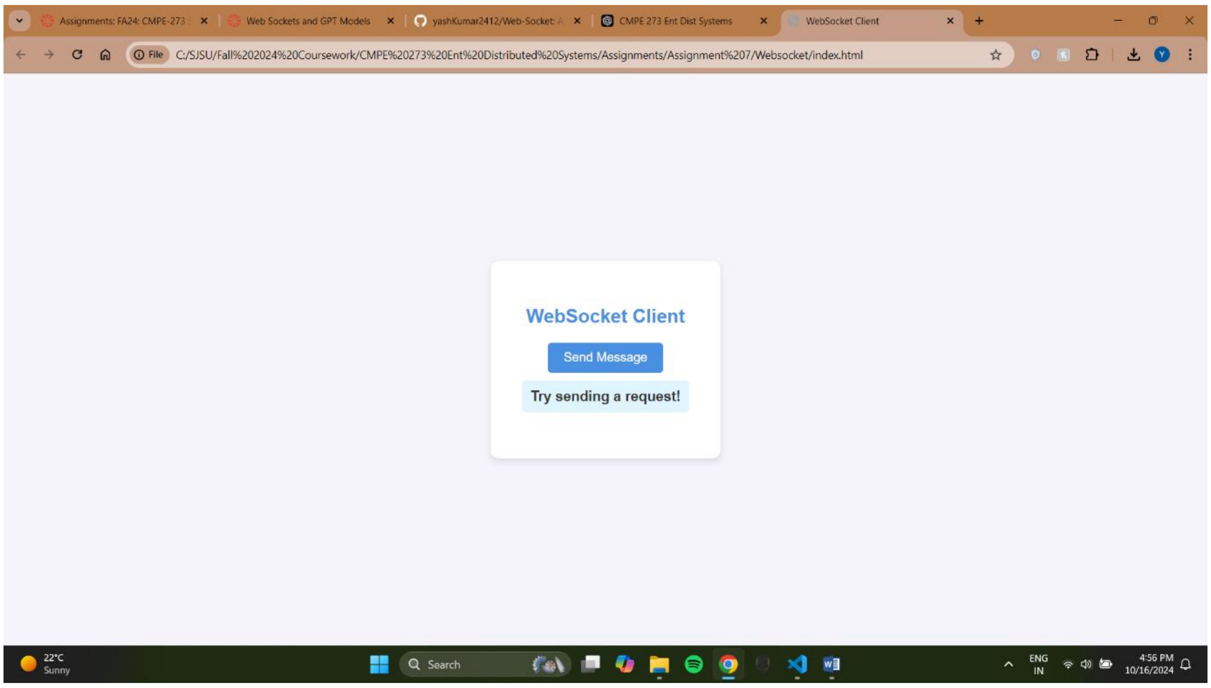
When ran with the client application, we maintained a global count in the client to keep track of the number of responses. It was observed that for all 10K messages, the client sent a request to the server and received the appropriate response for each of them. This can be seen in the image below with the responses being printed to shell along with the final count of 10K at the end.



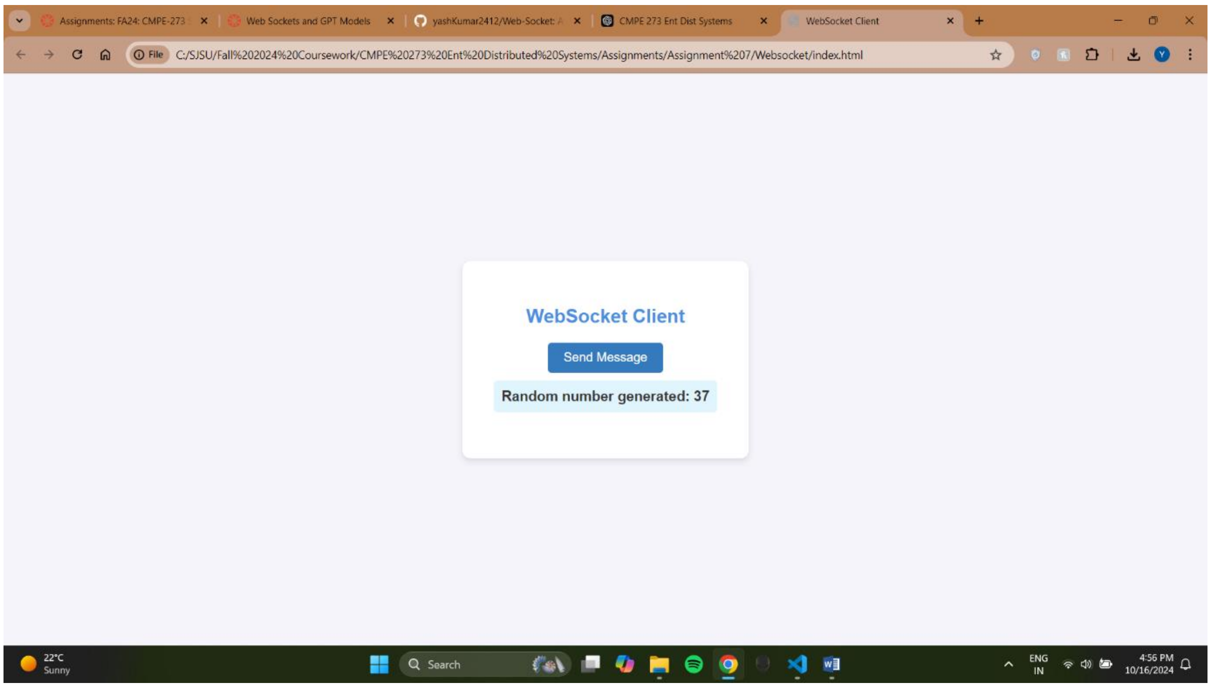
```python
import asyncio
import websockets

total = 0

async def send_message():
    uri = "ws://localhost:8765"
    async with websockets.connect(uri) as websocket:
        for i in range(10000):
            global total
            message = f"Request {i}"
            await websocket.send(message)
            response = await websocket.recv()
            total += 1
            print(f"Response for {response}")

asyncio.run(send_message())

print(total)
```

```
Response for Request 9990: 71
Response for Request 9991: 64
Response for Request 9992: 33
Response for Request 9993: 83
Response for Request 9994: 64
Response for Request 9995: 32
Response for Request 9996: 14
Response for Request 9997: 24
Response for Request 9998: 39
Response for Request 9999: 37
10000
PS C:\SJSU\Fall 2024 Coursework\CMPE 273 Ent Distributed Systems\Assignments\Assignment 7>
```

When ran with the client UI, we pressed the button to generate a random number and it did so each time, replacing the previous response with the new message from the server. This can also be seen in the images below with the default screen (before requests) and the updated display with the randomized number from the server.



Default screen when the client UI is opened.



Updated display with the randomly generated number.

As we have seen, we can correctly implement websocket-based communication between a server and various clients.