This week you will learn how a websocket server works. SocketIO uses websocket so you should research what is websocket and what are the differences between other protocols. This will help you to understand this concept. There are other ways to create a server and maintain it but we use socketio in our projects, you can research others for better learning.

Here is socketio-python doc link: https://python-socketio.readthedocs.io/en/latest/index.html

Project 1: Create a Basic Socket.IO Server & Client

1. Create a **Python Socket.IO server** (server.py) that:

- Listens on port 5000
- Prints "Client connected" when a client connects
- Prints "Client disconnected" when a client disconnects

2. Create a **Python Socket.IO clien**t (client.py) that connects to this server.

3. When the client connects, the server should send back a "welcome" event containing the message: "Welcome to the Socket.IO server!"

4. The client should print this message to the console.

Expected Output:

Client prints the welcome message and server logs connection information.

Project 2: Sending Custom Events & Data

1. The client sends a "login" event with a JSON dict:

{ "username": "Ali", "level": 5 }

2. The server receives this data and responds with:

"Hello Ali! Your level is 5."

using a "login_response" event.

3. The client prints the response.

Expected Output:

Client sees a custom welcome based on the JSON it sent.

Project 3: Implement Real-Time Chat (Broadcasting)

1. Create a server to listen for a "chat_message" event.

2. When a client sends "chat_message": "Hello", the server should broadcast:

"new_message": "Ali says: Hello"

to **all connected clients**, including the sender.

3. Create two client scripts and verify the broadcast works.

Expected Output:

Both clients see each other's messages instantly.

Project 4: Working with Rooms

1. Create two rooms: "roomA" and "roomB".

2. When a client sends:

sio.emit("join_room", {"room": "roomA"})

the server adds that client to the specified room.

3. When a client sends a "room_message" with a message body, the server broadcasts it **only inside that room**.

4. Clients should print messages only from their own room.

Expected Output:

Clients in roomA communicate only with roomA, and same for roomB.

Project 5: Namespaces & Event Separation

1. Create two namespaces:

- `/chat` for chat messages
- `/status` for system notifications

2. On `/chat`, handle events:

- "chat_message"

3. On `/status`, handle event:

- "health_check", server should respond with "ok"

4. Write a client that connects to **both** namespaces and logs received events.

Expected Output:

Chat messages appear on `/chat`, system checks appear on `/status`.

Project 6: Asynchronous Socket.IO Server (aiohttp)

1. Build an **async Socket.IO server** using `socketio.AsyncServer`.

2. Make event handlers asynchronous, e.g.:

```
@sio.event
async def get_time(sid):
    return {"server_time": time.time()}
```

3. Create an async Python client (python-socketio[asyncio]) that:

- Connects
- Sends a "get_time" event every 1 second
- Prints server responses

Expected Output:

Client prints updated time every second; server handles load asynchronously.