Backend Assignment Socket Chat Server

Goal

Build a **simple TCP chat server** (no HTTP, no database) that allows multiple users to connect, log in with a username, and chat with each other in real time.

You can use **any programming language** you're comfortable with (Python, Node.js, Go, Java, etc.).
Use only the **standard library for sockets** — no frameworks or external chat libraries.

Requirements

1. Server Setup

- The server should listen on **port 4000** by default.
- It must handle **multiple clients** at the same time (at least 5–10 users).
- You can make the port configurable through an environment variable or a command-line argument.

2. Login Flow

When a client connects, they must first send:

```
LOGIN <username>
```

If the username is already in use, reply with:

```
ERR username-taken
```

Otherwise, reply with:

```
OK
```

- Once logged in, the user can send and receive chat messages.

3. Messaging

After login, users can send messages in this format:

```
MSG <text>
```

The server should broadcast that message to all connected users as:

```
MSG <username> <text>
```

- Handle newlines and extra spaces gracefully so messages always appear clean.

4. Disconnects

- When a user disconnects or closes their connection:
  Remove them from the active user list.

Notify all remaining users with:
`INFO <username> disconnected`

5. Optional Features (Bonus)

If you want to go further, you can add:

List active users

```
WHO
```

- → Respond with `USER <username>` for each connected user.

Private messages

```
DM <username> <text>
```

- **Idle timeout** — disconnect users after 60 seconds of inactivity.
- **Heartbeat** — respond to `PING` with `PONG`.

Example Interaction

Client 1

```
$ nc localhost 4000
LOGIN Naman
OK
MSG hi everyone!
MSG how are you?
```

Client 2

```
$ nc localhost 4000
LOGIN Yudi
OK
MSG hello Naman!
```

Client 1 sees

```
MSG Yudi hello Naman!
```

Client 2 sees

```
MSG Naman hi everyone!
MSG Naman how are you?
```

When Client 1 disconnects:

```
INFO Naman disconnected
```

What to Deliver

1. **Source Code**
   - Your implementation of the chat server (single or multiple files).
   - Use only standard library socket programming.
2. **README File**
   - How to run the server (commands, dependencies, etc.).
   - Example commands to connect (e.g., using `nc` or `telnet`).
   - Example input/output from at least two users chatting.
3. **Screen Recording (Compulsory)**
   - Record a short video (1–2 minutes) showing your server running and at least **two clients chatting** in real time.
   - You can use any screen recorder (e.g., Loom, OBS, QuickTime, or Windows Game Bar).
   - Include the video link in your README or submission.
4. **(Optional) Deployment Link**
   - If hosted online, include the IP address or hostname and port number.
   - If not hosted, show in the screen recording that it runs locally.