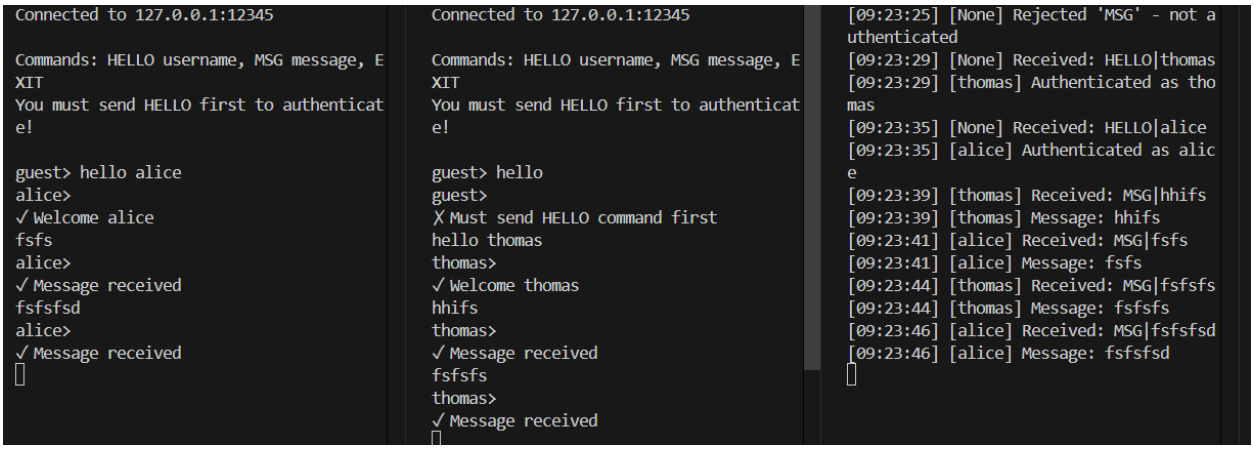
Part 5 – Reflection Questions

1. What happens if the server crashes while the client is connected?
    a. When the server is crash, the client try to send the message is gonna fail to send the message.
2. How can the server handle multiple clients?
    a. Create the loop in main server to accepts connection is gonna creates new thread. Each thread independently handles one client. All clients can connect simultaneously. Session state stored in dictionary: sessions[conn]




3. Why can recv(1024) split messages unexpectedly?
    a. TCP is a stream-based protocol, not message-based. The problems with fixed buffer size:
        i. recv() returns whatever data is available
        ii. A large message might be split across multiple packets
        iii. Multiple small messages might arrive in one recv() call
4. How would you add basic security (authentication, encryption) to this chat application?
    a. Require authenticated HELLO (username+password checked against securely stored salted hashes or a token issued after login) and wrap all sockets with TLS while adding input validation, session timeouts, and rate-limiting to reduce abuse.

Part 6 – Security Analysis

In 300–400 words, analyze the security implications of your design. Your analysis must reference specific parts of your implementation and address:

New attack surfaces introduced by protocol and state

Potential abuse scenarios

Why TCP does not provide security guarantees

Proposed mitigations for a real deployment

The implementation in *server.py* and c*lient.py* introduces a significant parsing attack surface via the COMMAND|DATA framing. Because TCP is a byte-stream protocol, not message-oriented, *recv()* may deliver partial commands. Although the code includes MAXLENGTH checks and searches for the " | " delimiter, an attacker can exploit this by sending fragmented packets to desynchronize the server's buffer or inject unauthorized command sequences.

Additionally, managing state via a dict and an authenticated flag introduces resource exhaustion risks. Since the server spawns a new thread for every connection, an attacker can flood the service to exhaust file descriptors or memory, leading to a Denial of Service (DoS).

**Abuse Scenarios and TCP Limitations**

The design is highly vulnerable to several abuse scenarios:

- **Impersonation:** The HELLO command is accepted without verification, allowing for identity theft or "username squatting."
- **Information Leakage:** The log() function records raw payloads, potentially exposing sensitive data in production logs.
- **Packet Sniffing:** Because TCP provides no security guarantees, focusing only on reliability and ordering, it lacks confidentiality and integrity. All traffic is sent in plaintext, leaving it vulnerable to eavesdropping and Man-in-the-Middle attacks.

**Proposed Mitigations for Deployment**

To harden the system for real-world use, the following mitigations are essential:

- Wrap all sockets with ssl.SSLContext (TLS) to provide robust encryption and endpoint authentication.
- Replace simple handshakes with salted, iterated hashes and issue short-lived session tokens post-login.
- Use length-prefixed framing instead of delimiters. This allows the server to know exactly how many bytes to read, preventing buffer desynchronization.
- Use Asynchronous I/O or a fixed thread pool to manage connections, alongside per-IP rate limiting to mitigate spam and DoS attacks.