**Project Deadline: 14.11.2025**

# Chat Project

## 1. Project Objective

The purpose of this project is to provide students with an opportunity to develop a **multi-user chat system** composed of **three separate Python programs**:

1. **Chat Server** – Manages connections, enforces nicknaming rules, broadcasts public messages, and handles private messages.
2. **Chat Client** – Connects to the server (directly or via relay) and allows user interaction.
3. **Chat Relay** (optional) – Acts as a pass-through server that prepends '*' to any client nickname and forwards traffic to the main server.

Students will practice **socket programming**, **basic concurrency**, **message logging**, **user interfaces** (GUI or console), and **spam protection**. By extending the Echo or Simple Chat module from Project 01, they gain deeper insights into real-world chat application features.

---

## 2. Required Tools and Prerequisites

- **Programming Language: Python 3**
- **Potentially Required Libraries:**
  - socket, threading or select (for concurrency)
  - time or datetime (for timestamps)
  - Optional: GUI frameworks (e.g., tkinter) or additional logging modules
- **Development Environment:**
  - Windows, Mac, or Linux with Python 3 installed
  - May use virtual environments to keep dependencies isolated

---

## 3. Project Structure and Modules

This project consists of three primary components: a server, a client, and a (non-mandatory) relay.

### A. Chat Server

Objective: Accept multiple client connections, enforce unique nicknames, relay messages, and log chat activity.

**Core Tasks:**

1. **Server Options**
    - Server should listen to selected, available IP address(es)
    - Server should listen to spesific port(s) (with a default value)
        - Chat server port
        - HTTP port
        - Websocket port
2. **Nickname Assignment**
    - Ensure each new client's requested nickname is unique.
    - If it's taken, add a random suffix (e.g., User123).
    - Block nicknames starting with character: '*' (reserved for relay).
3. **Public Message Handling**
    - Broadcast messages to all connected clients in real time.
    - Include timestamps in each broadcasted message.
    - Each message should also show source nickname.
4. **Private Messaging**
    - When a client requests a private message to another user, forward it only to that target.
    - If the target is offline, optionally save message for late delivery, discard or return an error (implementation-specific).
5. **Client Management**
    - Keep track of connected clients and update all client lists when someone joins or leaves.
    - Remove a departing user upon "Exit" command/button.
6. **Message Logging**
    - Record all messages (public or private) in a simple log file (text, CSV, etc.).
    - For private messages, include info on sender, recipient, timestamp.
7. **Simple Performance Monitoring**
    - Maintain a small counter or stats window: how many messages processed, how many clients connected etc.

- Print or display periodic status updates.

8. **Rate Limiting**
    - If a client sends messages too rapidly, temporarily mute or disconnect them to prevent spam.
9. **Web Server**
    - Server should also have http and websocket server functionality.
    - All messages (including private messages) and/or logs should be available on a web page in real time with websockets.
        - Same view with logging.

**B. Chat Client**

Objective: Provide an interface (GUI or console) for a user to join the chat, exchange messages, and engage in private conversations.

**Core Tasks:**

1. **Nickname & Connection**
    - Prompt user for a nickname at first step.
    - Connect directly to the server or to the optional relay server.
    - If connecting via relay, the nickname is prepended with '*' at the server level (automaticaly).
        - This step should be implemented in relay server (nex section).
2. **Main Chat Window**
    - Display all public messages with timestamps and nicknames.
    - Show a list of all currently connected users.
    - Remove a user from the list upon disconnection.
3. **Private Messaging Window**
    - If the user double-clicks a nickname (or some other action), open a dedicated private chat window.
    - Similarly, if the user receives a private message, pop up a separate window.
4. **Exit Handling**
    - On “Exit” button/command, cleanly disconnect from the server.
    - The server should remove this user from its internal list.

**C. Chat Relay (Optional)**

**Objective: Sits between the client and the main server, acting as a pass-through or "proxy" that appends '*' to the user's nickname.**

**Core Tasks:**

1. **Relay Connections**
    - Listen on a local port, accept client connections.
    - Establish a corresponding connection to the main server.
2. **Nickname Rewriting**
    - When the client sends its initial nickname, rewrite it as '*nickname' before sending to the main server.
3. **Transparent Forwarding**
    - For all chat data, pass bytes unmodified in both directions.
    - If you want advanced logging, you can also keep a local log of relayed traffic.

---

**4. Project Integration**

1. **Menu / Launch Scripts**
    - You may have separate scripts for chat_server.py, chat_client.py, and chat_relay.py, or a single integrated script with a menu.
2. **Network Flow**
    - Client → Server: Standard chat usage.
    - Client → Relay → Server: Relay usage scenario (nickname prepended with '*').
3. **Data Exchange and Timestamps**
    - Ensure you embed or prepend timestamps to each message string or packet.
    - Log the same in the server's log file.

---

**5. Deliverables**

1. **Code Files**
    - Python scripts for server, client, and optional relay (plus any helper modules).
2. **Documentation**

- Inline comments and/or a short README explaining how to run each program and any dependencies.

3. **Execution Guide**
    - Detailed instructions on starting the server, launching clients, and optionally using the relay.
    - Mention any optional command-line arguments (e.g., specifying ports or addresses).
4. **Test Outputs**
    - Screenshots or console logs showing:
        - Multiple clients connecting and exchanging public messages.
        - Private messages in separate windows or console prompts.
        - Server logs with timestamps and message details.
        - Rate limiting in action (if triggered).

---

**6. Evaluation Criteria**

1. **Functionality**
    - Correctly enforces unique nicknames, broadcast logic, private messaging, etc.
    - Chat relay works if implemented (prefixing with '*').
2. **Code Quality**
    - Well-organized and readable code with meaningful function/class names.
    - Proper concurrency or event handling (thread-based or select-based).
3. **Error Management**
    - Graceful handling of denied nickname collisions or spam rate limits.
    - Clear messages/logs for user errors or connection failures.
4. **Logging and Timestamps**
    - All messages show correct date/time.
    - Server log file captures relevant events.
5. **Performance Monitoring & Rate Limiting**
    - Server accurately updates total message counts and connected client stats.

- Rate limit logic effectively prevents spam from flooding the chat.

6. **Creativity**
    - Additional features like emoticons, command-line arguments, small animations in GUI, etc., are welcome.
    - Overall user-friendliness and innovative touches.

---

**7. Additional Suggestions (Non-Mandatory)**

1. **Incremental Development**
    - Start by creating the core server with a single threaded or select-based approach and improve.
    - Then implement the client basics (nickname, messages).
    - Gradually add private messaging and GUI elements if desired.
2. **Testing**
    - Test with multiple console clients. Ensure robust concurrency.
    - Use logging to confirm private messages are not sent to everyone.
3. **Relay Deployment**
    - Treat the relay as a separate optional exercise. If time allows, integrate it and test prefix logic.
4. **Source Control**
    - Use Git or another version control system to manage changes.
    - Commit frequently to track progression.
5. **User Interface**
    - Whether GUI or console, ensure it’s intuitive: clearly labeled prompts and well-formatted messages.