Phase 1: Requirements Analysis

- **Project Type:** Server and client chat application.
- **Protocol Specifications:** Detailed message types and formats for communication between server and clients.
- **Key Features:**
    - User Identification and Authentication.
    - Status Management (ACTIVE, BUSY, AWAY).
    - Private and Public Messaging.
    - Room Creation and Management.
    - User Invitations to Rooms.
    - Robust Error Handling adhering to protocol responses.
- **Constraints:**
    - Username limit: 8 characters.
    - Room name limit: 16 characters.
    - Enforce SOLID principles and clean code practices.

Phase 2: UML Design (Object-Oriented)

Class Diagram Overview

1. **User**
    - **Attributes:**
        - `Username` (string, max 8 chars)
        - `Status` (enum: ACTIVE, BUSY, AWAY)
        - `ConnectedRooms` (List<Room>)
    - **Methods:**
        - `Identify()`
        - `ChangeStatus()`
        - `SendMessage()`
        - `JoinRoom()`
        - `LeaveRoom()`
2. **Room**
    - **Attributes:**
        - `RoomName` (string, max 16 chars)
        - `Users` (List<User>)
    - **Methods:**
        - `CreateRoom()`
        - `InviteUsers()`
        - `BroadcastMessage()`
        - `AddUser()`

        - `RemoveUser()`

3. **Server**
    - **Attributes:**
        - `ConnectedUsers` (Dictionary<string, User>)
        - `ActiveRooms` (Dictionary<string, Room>)
    - **Methods:**
        - `HandleIdentify()`
        - `HandleStatus()`
        - `HandleText()`
        - `HandlePublicText()`
        - `HandleNewRoom()`
        - `HandleInvite()`
        - `HandleRoomText()`
        - `HandleLeaveRoom()`
        - `HandleDisconnect()`
        - `ValidateMessage()`
4. **Client**
    - **Attributes:**
        - `Username` (string)
        - `Status` (enum)
        - `JoinedRooms` (List<Room>)
    - **Methods:**
        - `ConnectToServer()`
        - `Identify()`
        - `ChangeStatus()`
        - `SendPrivateMessage()`
        - `SendPublicMessage()`
        - `CreateRoom()`
        - `InviteUsers()`
        - `JoinRoom()`
        - `LeaveRoom()`
        - `Disconnect()`

Relationships

- **Server** manages multiple **Users** and **Rooms**.
- **Users** can join multiple **Rooms**.
- **Rooms** contain multiple **Users**.

Phase 3: Pseudocode Plan

1. Server-Side Implementation

- **Initialize Server:**
    - Start listening on a specified port.
    - Initialize `ConnectedUsers` and `ActiveRooms` dictionaries.
- **Handle Incoming Connections:**
    - Accept new client connections.
    - Instantiate a `User` object upon connection.
- **Message Handling:**
    - Parse incoming JSON messages.
    - Validate message structure and required fields.
    - Route messages to appropriate handler methods (`HandleIdentify`, `HandleStatus`, etc.).
    - Implement error responses as per protocol.
- **User Management:**
    - Ensure unique usernames.
    - Manage user statuses and notify other clients on status changes.
- **Room Management:**
    - Create and delete rooms based on user actions.
    - Handle user invitations and room memberships.
    - Broadcast room-specific messages to members.

2. Client-Side Implementation

- **Initialize Client:**
    - Connect to the server using specified IP and port.
    - Prompt user for identification (`IDENTIFY` message).
- **User Interface:**
    - Provide options for changing status, sending messages, creating rooms, etc.
    - Display incoming messages and updates in real-time.
- **Message Sending:**
    - Construct JSON messages adhering to the protocol for various actions.
    - Handle user inputs and trigger appropriate message dispatch.
- **Message Receiving:**
    - Listen for incoming messages from the server.
    - Update UI based on `NEW_USER`, `NEW_STATUS`, `TEXT_FROM`, etc.

3. Error Handling & Validation

- **Server:**
    - Validate all incoming messages for completeness and correctness.
    - Respond with appropriate `RESPONSE` messages on errors.
    - Disconnect clients on critical errors.
- **Client:**
    - Handle server responses and display error messages to the user.
    - Manage reconnection or graceful shutdown on disconnections.

Phase 4: Confirmation

Ready to proceed with the UML Class Diagram creation and move into the C# implementation? Let me know if you'd like to adjust any part of the plan or if you're good to go!