Joshua Van Deren, Jacob Lai, Grant Savage, Joy Tan

April 23, 2018

SE310

Siewert

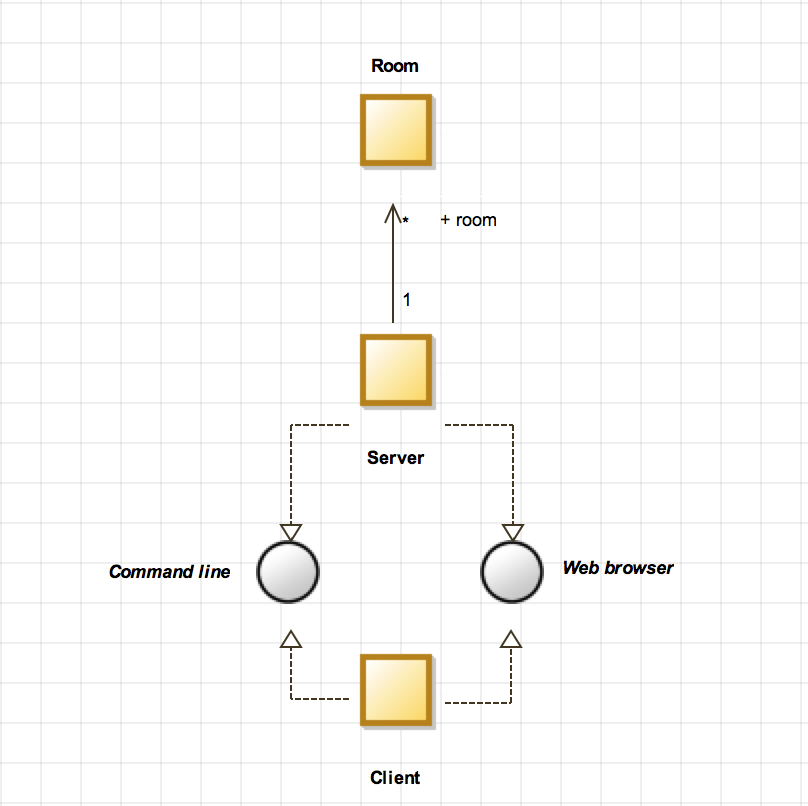
**SE310, Analysis and Design of Software Systems**

**Exercise #6 – Project Design**

1. Proposal and block diagram

For our project, we decided to develop a chatting medium for users through a command line interface. The first and most important requirement is having our server running and listening on a specified port. Users can then connect to the server and send messages to each other either privately or in a group. We performed a stress test to verify how many users can be connected to the server simultaneously, and our server was able to support 142 requests per second. Once connected to the server, users must enter a nickname to be recognized by other users. One of our requirements were that all the connected users must have a unique nickname, if a user enters a nickname that is currently being used by another user, they are prompted to enter in a different nickname. Although we were able to implement nicknames in our prototype, we decided to defer the unique nickname requirement since each user already has a universally unique identifier (UUID) which is a randomly generated combination of letters and numbers. In our initial stages, we planned to implement colored nicknames as part of our requirements for users to easily differentiate between which users are sending messages, but due to the time constraint and its low priority, we decided to defer that requirement as well.

After choosing a nickname, users are immediately prompted with a list of available actions they can perform or request from the program such as: list the current online users, create a room, join a room, leave a room, send a message, and quit the program. For private messaging, all a user has to do is select a currently connected user to talk to by specifying their nickname and send a message to them. Private messages can only be seen by the two users involved. We used rooms to implement group messaging. Users who join a room can send and receive messages to/from all other users currently in the same room while users who are not in the same room cannot receive those messages. Additionally, users can only be in one room or in no rooms at any given moment. Part of our requirements regarding group messaging included the implementation of an administrator of a room. The user who creates a room automatically becomes the assigned administrator of that room and gains the ability to block users from joining the room, unblock the blocked users, lock the room, and delete the room. Once the administrator of the room leaves, the room is automatically deleted. The administrator requirement is included in our design, but we were not able to develop it in our prototype because we focused on improving the already existing features instead.



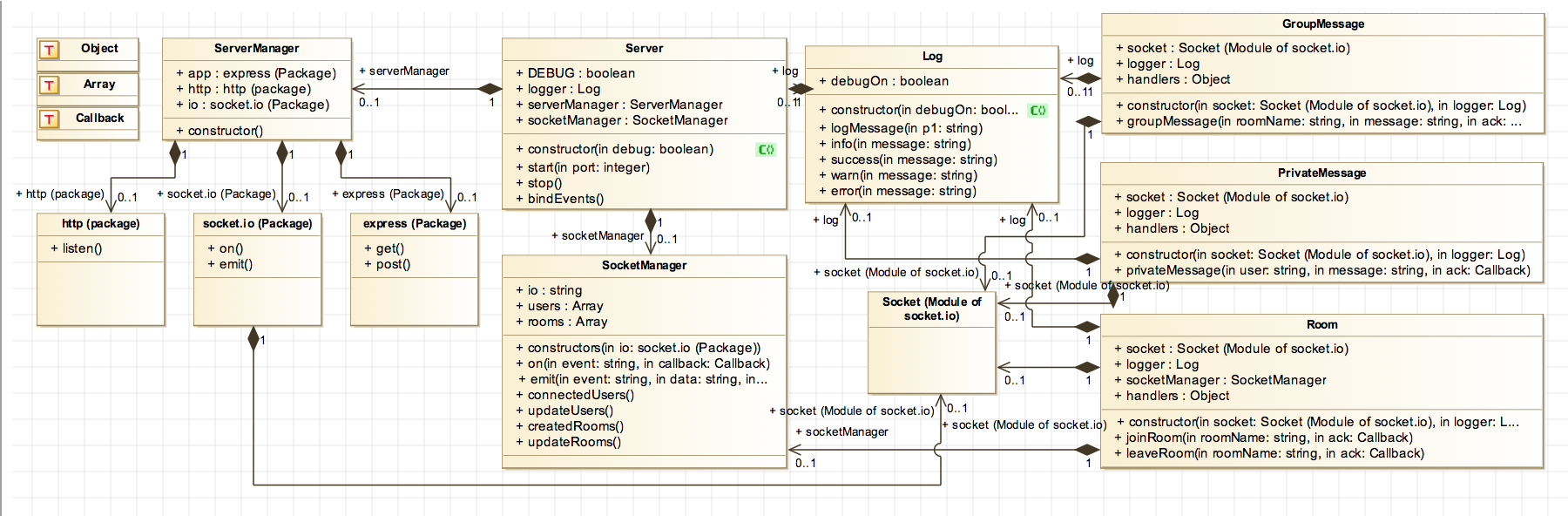
**Figure 1**: Block Diagram

Reading the above block diagram from bottom to top, the client must connect to the server via either the command line or the web browser. (We require that there must always be a server running). Additionally, the server interfaces to the command line and the web browser. Once the user is connected, they can create chat rooms. The user can create multiple chat rooms on one server.

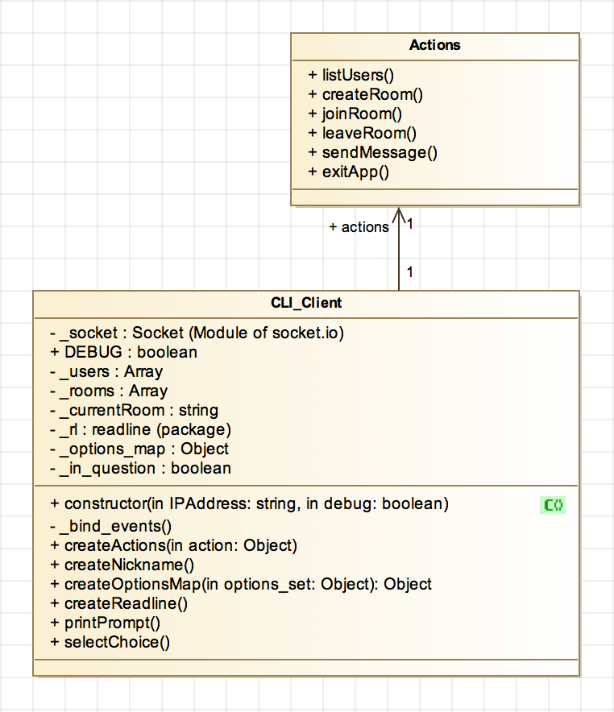
1. At least 3 structural and 3 behavioral models

\*NOTE\*: To better and more clearly view the following models, please refer to our repository on GitHub: jvanderen1/cli\_chat. All models and diagrams can be found and downloaded in the cli\_chat/models folder.

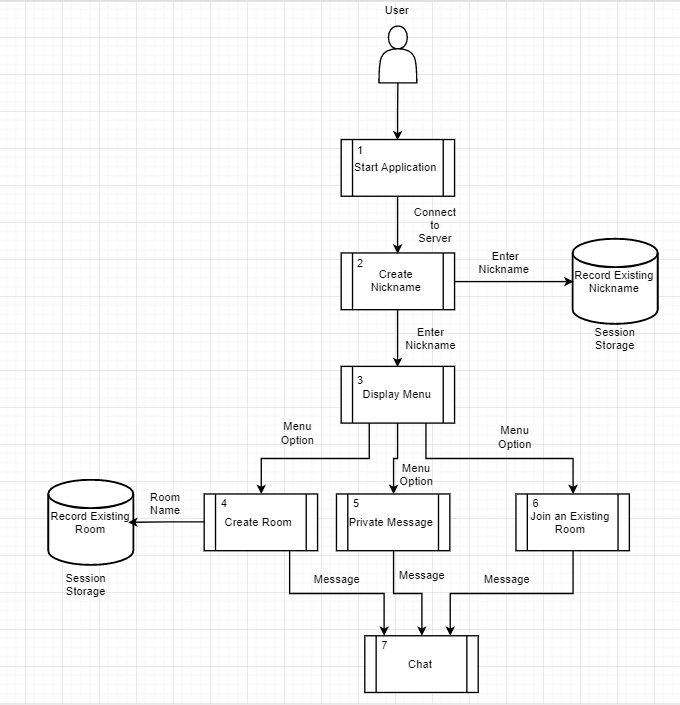
**Structural:**



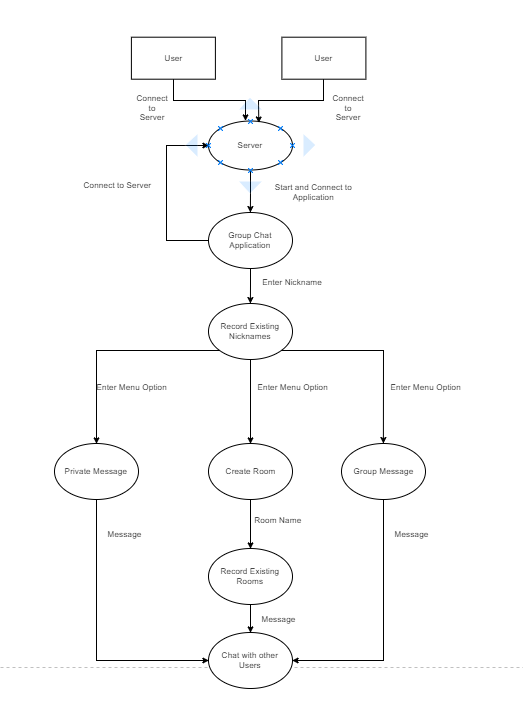
**Figure 2:** Server Class Diagram



**Figure 3**: Client Class Diagram

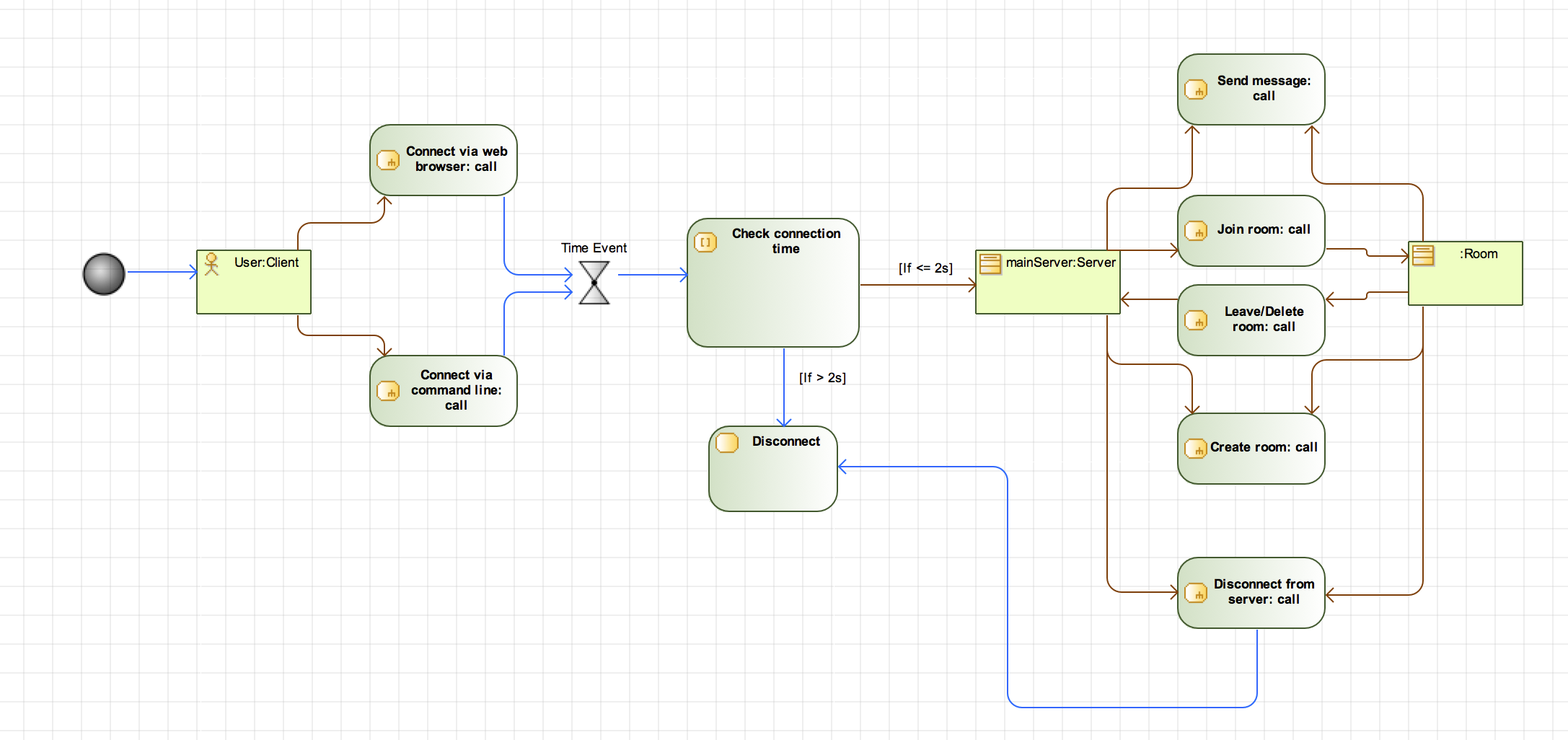
****

**Figure 3:** Data Flow Diagram

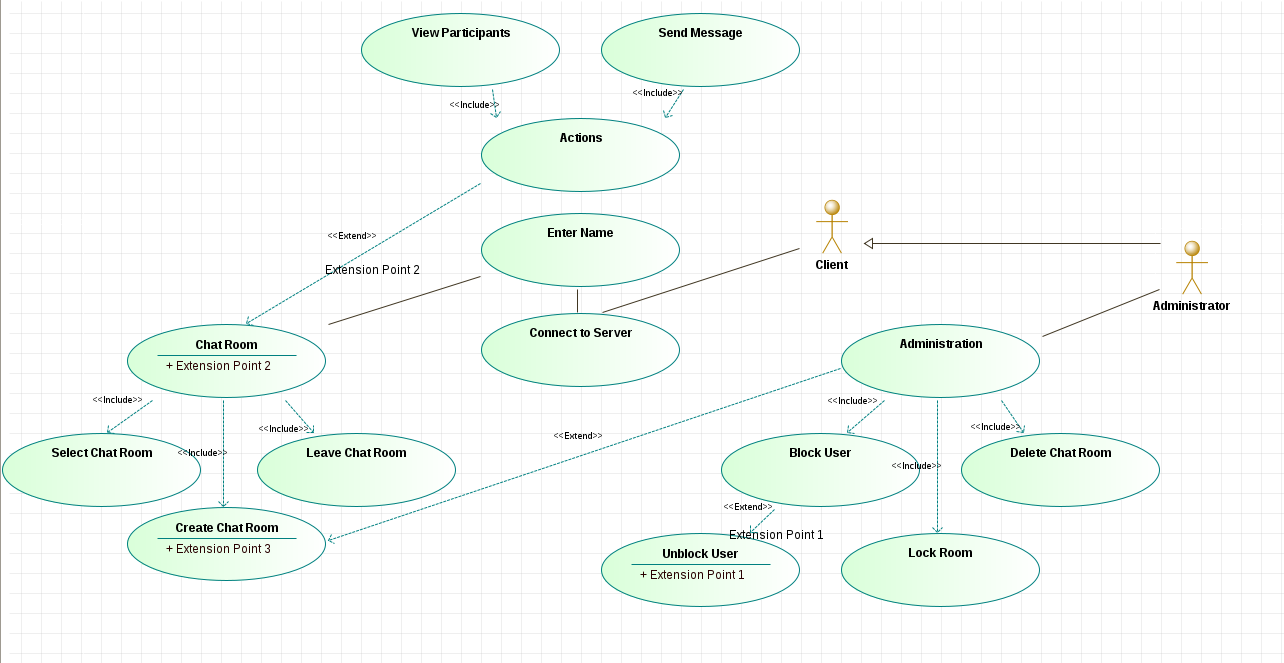


**Figure 4**: Control Flow Diagram

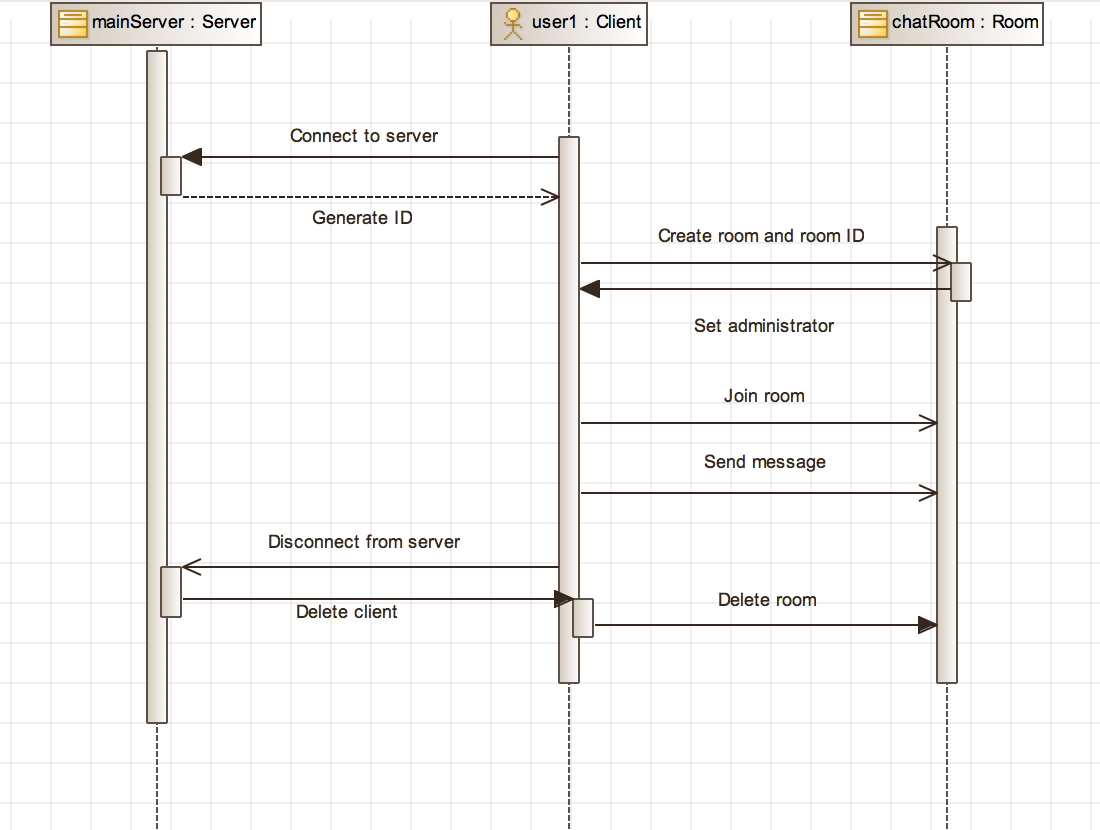
**Behavioral:**



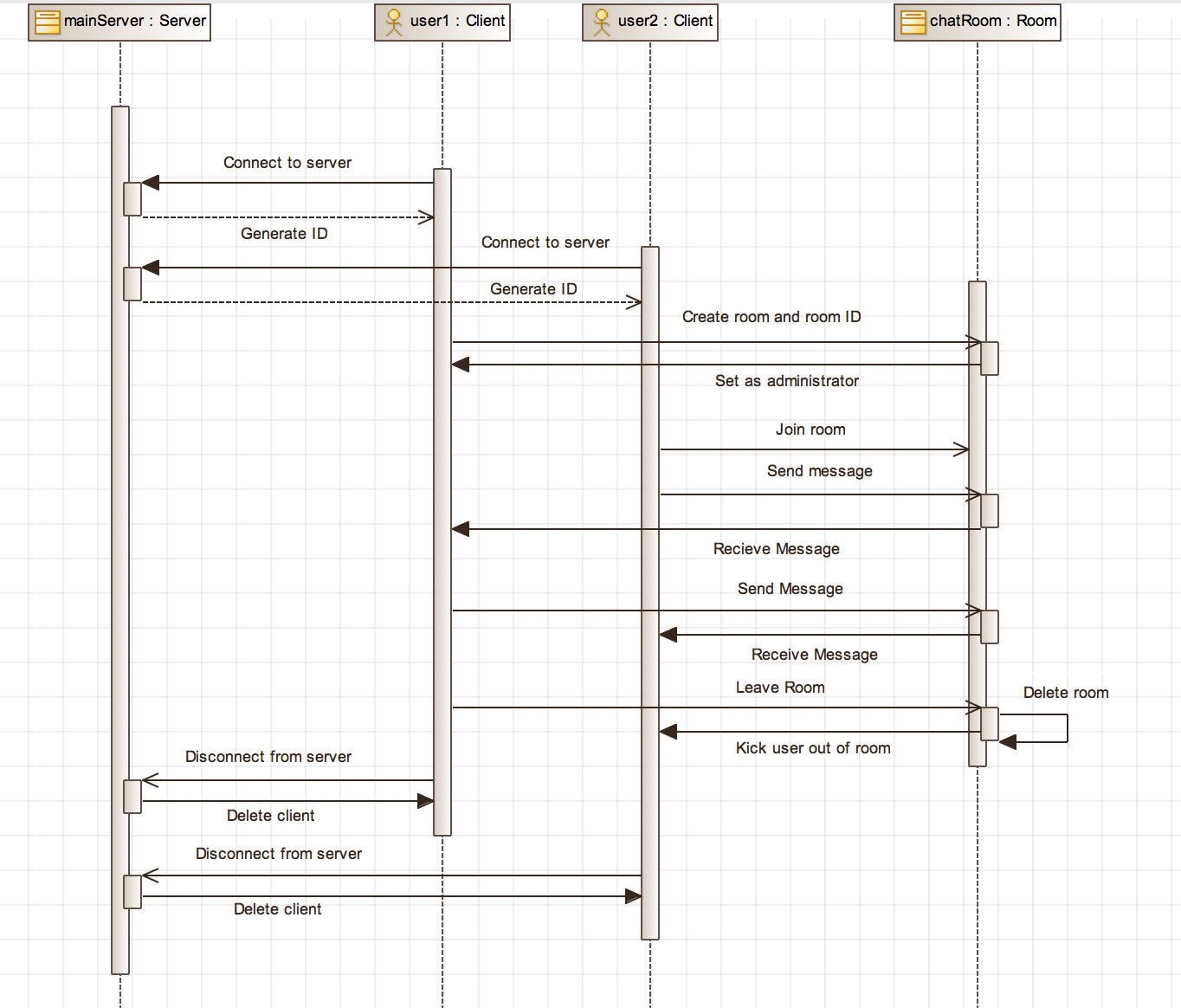
**Figure 5:** Activity Diagram

****

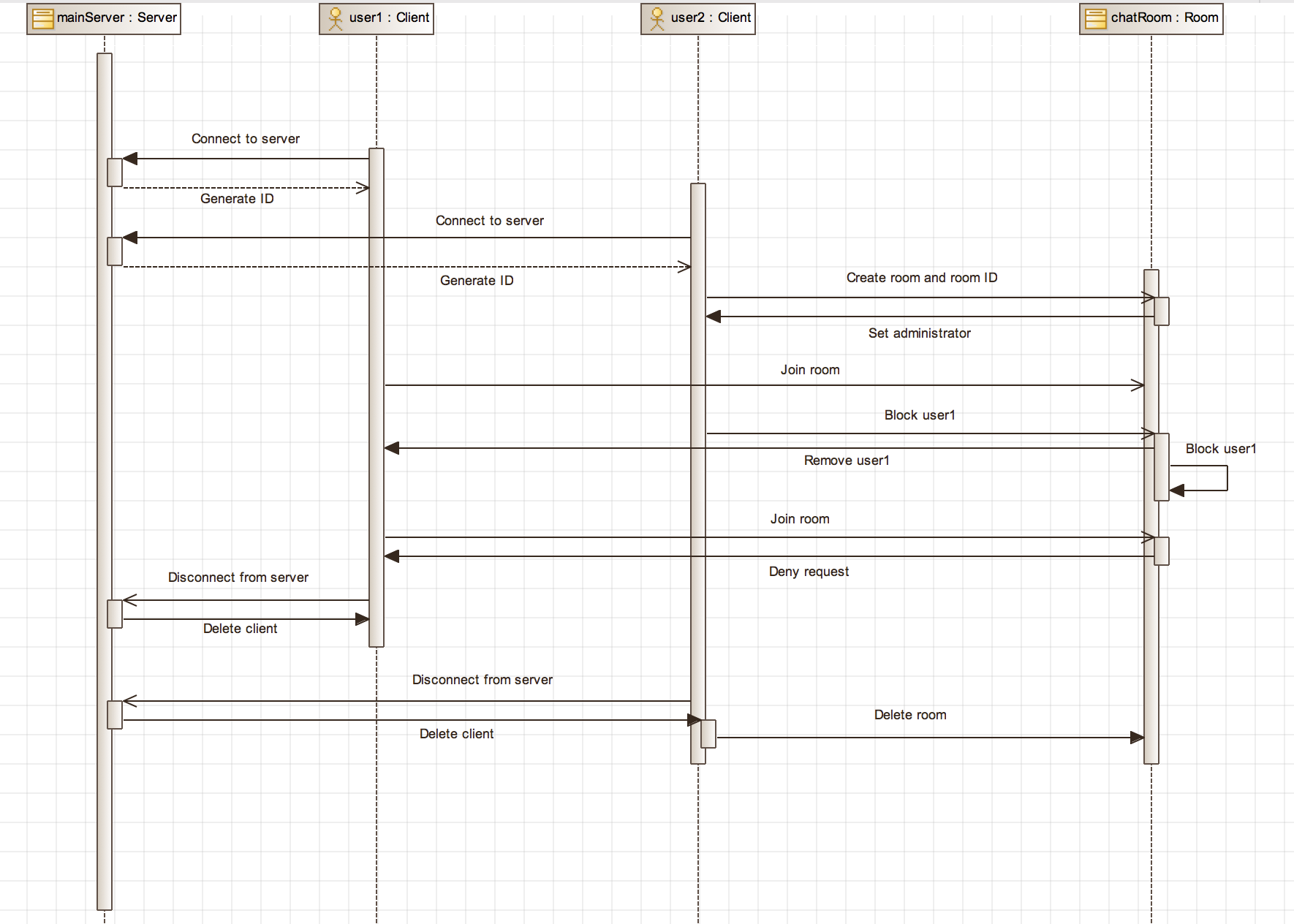
**Figure 6:** Use Case Model



**Figure 7:** Sequence Diagram 1

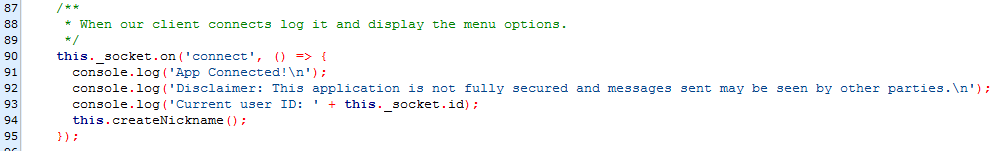


**Figure 8:** Sequence Diagram 2



**Figure 9:** Sequence Diagram 3

1. Refine requirements, analysis, and design based on walkthrough feedback
   1. Requirements
      1. Make a requirement declaring that server must be running first or test if client can wait for server to start
         1. Server should always be running first, but we will implement a test to verify that client can wait for the server to start in case the server crashes when client is already connected
      2. Save user ID
         1. Deferred feature, may be too time consuming for the scope of this class and requires a database
      3. Log conversations
         1. Deferred feature, may be too time consuming for the scope of this class and requires a database
      4. Implement style guide for consistency
         1. Halfway implemented Google style guide. (Client code follows style guide, but server code does not at the moment.)
      5. Need more detailed comments to help those unfamiliar with JavaScript, specifically, comment on “require” function and “mocking” a client
         1. In the process of implementing
         2. “require” pulls in and reads the file specified just like #include does
         3. When we “mock” a client, we create a fake client so that server can perform as if there is a real client
      6. Prevent hardcoding server port
         1. Server port is now separated into a different module for easier and less intrusive modification
         2. Unsure as how to read input from user for specific port to connect to because client has to be already connected to port in order to input answers that server can read
      7. Implement a disclaimer to notify users about the lack of privacy/open chat



* + 1. Creating clients but not deleting them – memory issue
       1. Clients are created when they connect and deleted as soon as they disconnect
    2. What happens if client port does not match server port
       1. Client waits until there is a server running on the same port then connects to server
       2. Server listens for clients to connect to port, does not timeout
  1. Analysis model and diagram
     1. What UML notation is the high-level diagram?
        1. Block Diagram under SysML Architect
     2. Describe diagrams with English words
        1. Currently in the process of completing
     3. Use Case Model – Is extend in correct direction?
        1. Yes
     4. Use Case Model – Provide user hierarchy
        1. User hierarchy has been implemented
  2. Design model and diagram
     1. Clarify block diagram notations
        1. In the process of implementing
     2. Label all flows in DFD
        1. All flows have been labeled
     3. DFD – Existing nickname is not persistent – should session data be stored persistently; persistent stores need to be changed to session based stores
        1. Changed persistent stores to session based stores
     4. DFD – Display menu should not lead back to user block
        1. Display menu is now a separate process and does not lead back to user block
     5. Use Case Model – Consider adding chat room admin actor and breaking up Use Case Model accordingly
        1. Administrator actor has been added

1. Prototype/unit test cases
   1. Please refer to our repository on GitHub: jvanderen1/cli\_chat. The code provided can be pulled and compiled to run our prototype. Additionally, all of our tests and test cases can be found on the repository as well under cli\_chat/test/client and cli\_chat/test/server.
2. Use-case model to requirements tracing analysis

R1: Send Message  
R2: Join Room  
R3: Leave Room  
R4: Colored Name  
R5: Create Nickname  
  
U1: Peer-to-Peer Chat  
U2: Peer-to-Room Chat  
U3: Customize Nickname  
U4: Block User from Room

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Requirements | Priority Weights | U1 | U2 | U3 | U4 |
| R1 | 4 | X | X |  |  |
| R2 | 3 |  | X |  |  |
| R3 | 3 |  | X |  |  |
| R4 | 5 | X | X | X | X |
| R5 | 1 |  |  | X |  |
| R6 | 1 |  |  | X |  |
| Score | | 9 | 15 | 7 | 5 |

Therefore, we will work on U2 first, then U1 second, etc.

Roles and responsibilities for this sprint:

1. Joy
   1. Refine requirements
   2. Fix bugs
   3. Proposal and descriptions
   4. Deferred features
   5. Walkthrough feedback
   6. SE310 Assignment #6 writeup
2. Jacob
   1. CFD
   2. DFD
3. Josh
   1. Update tests and test strategy
   2. Update Block Diagram
   3. Created Class Diagram for client
4. Grant
   1. DCD
   2. Fix bugs
   3. Refine requirements
   4. SE420 Assignment #6 writeup
5. Everyone
   1. Use case model to requirements tracing analysis
   2. Review and clean up of code