Use Case Realizations Server Side

- User starts server. Server ensures it is capable of connecting to the proper port. If not it throws an error message; otherwise it moves to a resting state to wait for response from a client.
- Client sends request to log in. Server used internal computation to validate the login. If it works, it allows the client to connect and echos that it was successful; otherwise it denies the request and echos the problem.
- Client sends request to create an account. The server does the same thing as above and ensures that it meets the requirements. Stores account and sends an echo if it does work, otherwise it echos the failure.
- Client successfully logs in. Server checks to see if they have any friends and if those friends are online. If so, it sends a message to those friends letting them know that that person is online.
- Client requests to add a new contact. Server checks to see if contact is held on the server. If so, adds contact to client's list; otherwise it refutes the request.
    - *Elaboration:* The user A presumably has access to the name of user B. User A sends a request to the server to add B to their contact list. The server receives the request and checks its storage information (model) for user B. If user B is found, it adds B to A's contact info in storage (model) and allows A to see B on his contact list by sending A an updated contact list (sort of a view - communicating to the client), however, B must add A as a friend before they can chat.
- Client requests to delete contact. Server checks and confirms the contact's existence. Removes the contact if it exists, refutes the request otherwise. Server responds with success or failure status.
- Client A requests to chat with client B. Since A has to have B as a contact for them to chat, the server does need to check for that. Server opens a communication window between the two and sends responses to both clients. If A or B add C to the chat, C receives the entire chat log between A and B
- Client logs off. Server informs contact list of the client that the client is offline.
    - *Elaboration:* The user closes the client. This sends a message to the server. The server receives the message, marks them as offline in the storage (model). Then the server checks the user's contact list for anyone that's online. If so, the server sends that user a message that they're friend is offline.

JSON Contact Data:

- List of “Contact” objects

Contact

- Name (must be unique, case sensitive)
- Status (online true : offline false)

Contacts (“data”) or <contact data> will be delimited strings as the following:
<username>\0<status>\n<username>\0<status>\n<username>\0<status>

IE:

“user1\0online\0user2\0of fline\0user3\0 online”

Room History (when logging in) will be delimited strings as the folowing:
<room>\0<user>:<message>\n<user>:<message>\n<user>:<message>\0\0<room>\0<user>:<message>\n<user>:<message>

room histories separated by \0\0 (split this first)

<room history 1>\0\0<room history 2>\0\0<room history 3...>

history data is split by \0, separating room id and the chat messages

<roomid>\0<message history>

messages are split by \n (and split by : between user and message, but this parsing isn’t necessary)

<user>:<message>\n<user>:<message>

Therefore, the large and complex example below:

00a1\0user1: Hi there! How are you?\nuser2: I am good, how about you?\nuser3: What about me???\0\055a2\0user1: this is the beginning of a new chat history\nuser2: yes, yes it is, user 3 isn’t here now\nuser4: But I am here!\0\0442a\0user1: im alone

Transmission data will always be a Dictionary<string, string> encoded in JSON.
Dictionaries do not include the quotation marks surrounding the data below.

Chat room id’s are just called “room”

API:

Client Sends...	**Server to client...**	**Server to others...**	**Notes**
“request”:”login” “user”:”johndoe” “password”:”abc123”	“request”:”login” “result”:”false”	N/A	username doesn’t exist or password is wrong or already logged in
	“request”:”login” “result”:”true” “data”:<contact data> “roomhistory”:”00a1\0name1: hi there\nname2: oh hello	“request”:”contactupdate” “data”:<contact data>	correct information See above the API table for how the room history is delimited.

	there\0\055b2\0user1: im here\nuser2: so am i”		
“request”:”signup” “user”:”johndoe” “password”:”abc123”	“request”:”signup” “result”:”false”	N/A	user name already exists
	“request”:”signup” “result”:”true”	N/A	successfully created new user - user will be logged in No info to others because new user - no friends yet
“request”:”addcontact” “user”:”friendname”	“request”:”addcontact” “result”:”false”	N/A	user does not exist or is already a contact friend
	“request”:”addcontact” “result”:”true” “data”:<contact data>	N/A	we could notify the user that was added that “johndoe” added him as a friend - maybe?
“request”:”removecontact” “user”:”enemyname”	“request”:”removecontact” “result”:”false”	N/A	user wasn’t a friend or doesn’t exist
	“request”:”removecontact” “result”:”true” “data”:<contact data>	N/A	could notify enemy user of removed friendship
“request”:”logout”	N/A	“request”:”contactupdate” “data”:<contact data>	New contact data for all other users incase they knew the logged out user - their status needs updated No need to send anything to the logged out user - they are gone
“request”:”leavechat” “room”:”00a1”	N/A	“request”:”chatuserleft” “room”:”00a1” “user”:”johndoe” “validusers”:”name1\0name2\0name3\0...\0...”	Remove self from chat session, everyone else can keep talking. Tell everyone else that johndoe left. Note: if johndoe was the 2nd to last person in the chat, the last person should be notified that everybody left, and their chat window should close. Room 00a1 would be disbanded in this case. The valid users is the list of users that can be added to the chat (mutual friends of everybody in the room).
“request”:”startchat” “user”:”myfriend”	“request”:”startchat” “result”:”true” “room”:”00a1” “users”:”johndoe\0myfriend” “validusers”:”name1\0name2\0name3\0...\0...” “history”:””		Create new chat session - opens on both ends. Both users notified of a “startchat” with the other person listed under user. **Note: this could be an already existing chat if the user wanted to start a new chat with a user that already has a chat.** IE - 00a1 already exists as a room only with johndoe and myfriend. This would return 00a1, not a new room. The valid users is the list of users that can be added to the chat

			(mutual friends of everybody in the room). History is always blank (here at least).
	“request”:”startchat” “result”:”false”	N/A	The given user wasn’t a two-way friend, you can’t start a chat with them :P
“request”:”sendchat” “room”:”00a1” “message”:”Hello friends!”	“request”:”receivechat” “room”:”00a1” “user”:”johndoe” “message”:”Hello friends!”		Message sent to everyone. Client who sent the message only displays it from the server’s response (incase they get disconnected, sending messages should not show up if you aren’t connected to the server)
“request”:”addtochat” “room”:”00a1” “user”:”friendtwo”	“request”:”addtochat” “room”:”00a1” “user”:”friendtwo” “result”:”false”	N/A	User was not on everybody’s friend list. This usually will not show up because the client should only display users to add that are in the same group as everyone in the room.
	[*to the person requesting the added person*] “request”:”addtochat” “room”:”00a1” “user”:”friendtwo” “result”:”true” [*to all people in the current chat*] “request”:”chatuserjoined” “room”:”00a1” “user”:”friendtwo” “validusers”:”name1\0name2\0name3\0...\0...” [*to the added user*] “request”:”startchat” “room”:”00a1” “users”:”johndoe\0myfriend\0friendtwo” “validusers”:”name1\0name2\0name3\0...\0...” “history”:”name1: Hi there\nname2: oh hello there”		First block is only sent to the sender. This is just a confirmation message. You shouldn’t do anything with it other than that. Second block is sent to all people in the current room. Third block is sent to the added user (telling them to join the room). The valid users is the list of users that can be added to the chat (mutual friends of everybody in the room).