The program design,
The whole assignment was implemented with python3 ‘Python 3.9.1’, and based on tutor Wei Song’s starter code. This code helped with much of the threading portion of the assignment, and was a blessing. I also used version control in the form of a private GitHub repository to store the code. I also used lecture code that worked to create a UDP server client relationship.

The server is designed so that upon initiation, there are no previous logs of data, and userlog.txt and messagelog.txt for broadcasted messages are started from scratch. Note that if the server is exited, the logs still remain and it only gets deleted upon loading of the server again. Server.py deals with the multithreading in python, with an instance of the class ‘ClientThread’ created, with self attributes such as the address, socket, and when the client was last active. That way it can handle the blocking of clients after the 10 second period. Once the user logs out, the client thread is set to not active and terminated that specific thread.

With the design, there are also many data structures that need to be defined for user of the application. These are initialised upon server starting.These variables are global so that they can be used within class functions. The list of them includes:

```
global clientStatus, activeUsers, messages, rooms
clientStatus = {}
activeUsers = []
messages = []
rooms = []
```

- ClientStatus is a dictionary of users, with the key being the username, and the value being the time that they last logged in. This allows for checking whether the client has waited 10 seconds after account blocking.
- ActiveUsers is a simple list of usernames, and upon logging in, the username is appended to the list. If the user logs out, they are removed from the list. This data structure continually updates whenever a client logs in or logs out.
- Messages is a list for broadcasted messages. When a client broadcasts, it is appended to this list, and then written out to messagelog.txt with the appropriate format.
- Rooms is a list of tuples, where the tuple is defined as:
    - ```
      (roomID, separateRoomUsers)
      ```
    - RoomID being the ID created sequentially, and separateRoomUsers being a list of users that are part of the room.
    - Messages for rooms are added to separate files:
        - ```
          'SR_{roomID}_messagelog.txt'
          ```
    - It is therefore handled via the file whenever users requests for messages.

The application layer message format,
The application layer message format is simple, whereupon a command from the client, a dictionary is sent through the socket through JSON dumps. There is always a field for the command called ‘type’, and the server analyses the type of command it is, and can therefore perform the correct action.

Error handling of the client's input is done directly through the client side, where string splits of python are used to determine whether there are the correct number of arguments, and the arguments are in the correct format e.g. correct timestamp format.

A brief description of how your system works.

Login - Login is required from the beginning when the client starts their connection with the server. Therefore it is out of the first loop in the attempt to connect to server. The client send their username, password, IP, and udp port to the server. The server authenticates whether the username exists, and then whether the password is matching correctly. If so, the user is added to the active list, and the user enters the main loop.

Blocking of user- Upon a failed login attempt, the server sends the fail attempts number back, to the client, and if the client's login attempts are over that number, they are blocked. This is confirmed within the clientStatus data structure, where there is a timestamp of their failed login after the amount of attempts. The user is then blocked, and this is noted via the key and boolean value when they reattempt to login.

Logout- A user enters the command OUT, and then their username is sent to the server. The server sets their client thread to not active, and removes the username from the list of active users. The individual client thread stops running and the user is successfully logged out.

Broadcast Message- A user enters the command BCM, error checking is done to ensure a valid message input is entered, and a message is included in the message to be sent to the server. The server handles the appending of messages to the message log and prints it out to the terminal.

Display Active Users- A user enters the ATU command, and the request is sent to the server through the same method of a dictionary, which contains the message type and user who entered the command. The server then handles checking active users, and returns a list back to the client, and the client.py code then handles displaying the active users in order of when they were active. If no other users exists, then no users are shown.

Separate Room Service- When the client wants to create a separate room building, a room is appended to the list of rooms, unless there is a room with the same users in it already. Messages are sent in the same way, where the server checks if the user is in that room, otherwise it will not be able to send a message.

Read Message - Reads messages directly from the messagelog, and if required, looks through to see which rooms the user are part of, and finds all messages that are after the timestamp in the command. Data structure looks like a tuple with roomID, and a list of missed messages.

P2p- For this part, I attempted to make an individual UDP server for each client, that attempts to listen for messages when another user makes a request to send a file over. Firstly, the user gets the user they want to send a video file to's IP address and unique UDP port. The process for which the presenter is a thread is created where they are able to listen for incoming files from the presenter, then they receive it. There is some unusual functionality, so the command after the p2p transfer can be done on a new line.

Describe possible improvements and extensions to your program and indicate how you could realize them.

Upon completion of the assignment, I think a few improvements that could be made are a greater emphasis on object oriented programming, allowing for different classes to operate in parallel, and more error raising usage since it is all in python. This would make the files shorter, and possibly improve readability.

```
→  CS3331-Assignment git:(feature/UDP) ✗ python3 client.py 127.0.0.1 4000 80
0
> Username: hans
> Password: falcon*solo
> Welcome to the TOOM!
> Enter one of the following commands (BCM, ATU, SRB, SRM, RDM, OUT): OUT
Bye hans!
```

Login and logout

```
→  CS3331-Assignment git:(feature/UDP) ✗ python3 client.py 127.0.0.1 4000 80
0
> Username: hans
> Password: fawgawg
> Invalid Password. Please try again
> Password: gawfeag
Invalid Password. Your account has been blocked. Please try again later
→  CS3331-Assignment git:(feature/UDP) ✗ python3 client.py 127.0.0.1 4000 80
0
> Username: hans
> Password: ga
> Your account is blocked due to multiple login failures. Please try again l
ater
```

Blocking of user

```
→  CS3331-Assignment git:(feature/UDP) ✗ python3 client.py 127.0.0.1 4000 80
01
> Username: b
> Password: b
> Welcome to the TOOM!
> Enter one of the following commands (BCM, ATU, SRB, SRM, RDM, OUT): ATU
    > a, 127.0.0.1; 800; active since 04 Aug 2022 18:03:44
```

Displaying active users

```
> Welcome to the TOOM!
> Enter one of the following commands (BCM, ATU, SRB, SRM, RDM, OUT): RDM b
03 Aug 2022 00:00:00
Messages broadcasted since 03 Aug 2022 00:00:00:
    #1; b: hello at 04 Aug 2022 18:04:28
> Enter one of the following commands (BCM, ATU, SRB, SRM, RDM, OUT):
```

Reading of broadcasted messages

```
> Enter one of the following commands (BCM, ATU, SRB, SRM, RDM, OUT): SRB b
> Separate chat room has been created, room ID: 1, users in this room: b, a
> Enter one of the following commands (BCM, ATU, SRB, SRM, RDM, OUT): SRM 1
hello
#1; 04 Aug 2022 18:05:33; a; hello
```

Creating room and sending msg

```
> Enter one of the following commands (BCM, ATU, SRB, SRM, RDM, OUT): RDM s
04 Aug 2022 00:00:00
room-1:
    ##1; a: hello at 04 Aug 2022 18:05:33
```

Reading msg from room

```
> Welcome to the TOOM!
> Enter one of the following commands (BCM, ATU, SRB, SRM, RDM, OUT or UDP):
> Received example1.mp4 from b
> Enter one of the following commands (BCM, ATU, SRB, SRM, RDM, OUT or UDP):
```

```
> Enter one of the following commands (BCM, ATU, SRB, SRM, RDM, OUT or UDP): UDP a non_existent

File not accessible
> Enter one of the following commands (BCM, ATU, SRB, SRM, RDM, OUT or UDP): UDP a example1.mp4
```

The user can currently see a text response, however I did not implement the actual downloading of video. There is a p2p connection established.