Basic Info:

Environment: Linux, Python 3
Library: socket, threading, ssl, hashlib, json, pyCrypto
IP, PORT: localhost, 8082(server)
Source Code: https://github.com/StevChen/SecureMessageChannel

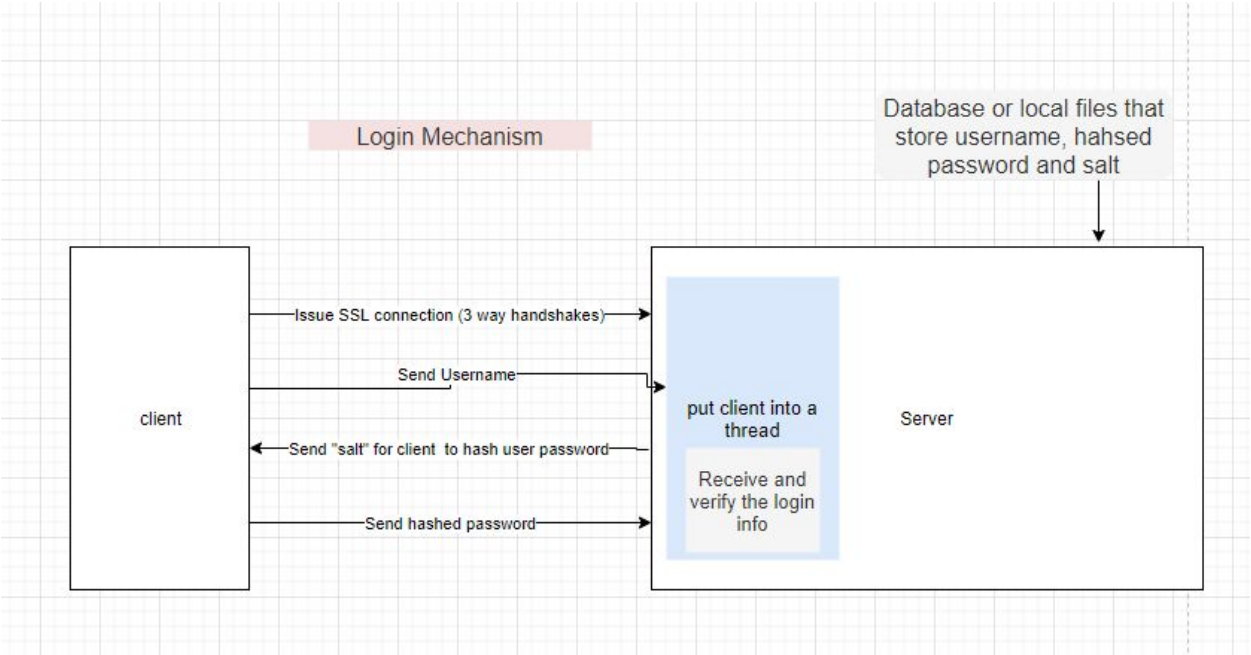
Implementation:

1. SSL/TLS/login mechanism
   Using the openssl to generate the self-signed certificate(public key) and private key pair. And then using the module python3 built in module to wrap the socket into ssl.
   - Server has its certificate and private key pair, while client has server certificate in its local file.
   - See design diagram below and appendix for detail implementation and screenshoot

2. User authentication and username and password management
   - The user profile, username and user password is saved in the local json file in json format
   - There are two groups of user, admin and regular user. They have different accessibility base on their user group. (eg, admin can see all the connected user; regular user can see friend list and chat with friend)

3. Communication module
   - There are two clients. Sender and Receiver, the receiver start the wait for chat connection first, and then sender send the chat request.
     - In first round, they do a key exchange. Send send his public key with chat request and then receiver return a packet with receiver's public key to the sender
     - In second round, sender type in the message and send the package with ciphertext. Receiver receive the packet and print out the packet as project request (see appendix for screen shot)

4. Deal with offline client
   - See appendix



Login mechanism:

- Client login with username and server send the corresponding salt
- Client input password and server verify the hashed password


```
{
    "0": {
        "username": "admin",
        "password": "c91e58179c6bd209fefe66e3d14364ba7b3513c846b14e4b3b4bfde4d255cf80",
        "salt": "XQb5XbnCJiNU154z",
        "group": "admin",
        "metadata": {
            "friends": [
                ""
            ],
            "group": "admin"
        }
    },
    "1": {
        "username": "user1",
        "password": "17ee9cbc8771f5e42299d5e3409139dbd8271bab97afda26e0c1fcdb5a888c26",
        "salt": "ZAt5SbynJnGDcZFy",
        "group": "regular",
        "metadata": {
            "friends": [
                "user2"
            ],
            "group": "regular"
        }
    },
    "2": {
        "username": "user2",
        "password": "f17ebd74274d2932fbe0699d1b276384cf74330cc6eb6c32edb6ebd6a61ec0ef",
        "salt": "HCuHZA2h6jnBnsAT",
        "group": "regular",
        "metadata": {
            "friends": [
                "user1"
            ],
            "group": "regular"
        }
    }
}
```


* User profile stored as json format in .json file

```
Waiting for client......
connection established for client: ('127.0.0.1', 61285)
strat a new thread to handle client requests
Waiting for client......
{'username': 'user1', 'password': '17ee9cbc8771f5e42299d5e3409139dbd
8271bab97afda26e0c1fcdb5a888c26', 'salt': 'ZAt5SbynJnGDcZFy', 'group
': 'regular', 'metadata': {'friends': ['user2'], 'group': 'regular'}
}
ZAt5SbynJnGDcZFy
user:  user1  connected
waiting for password...
user:  user1  login successfully
```

```
Connection Success
Username: user1
Recevied Salt:  ZAt5SbynJnGDcZFy
Password:
login success
Thank you for using the secure chat
Pleas pick one of the options below, eg. '0' for exit
 [1] Show friend list
 [2] Chat with friend
 [3] wait for chat connection
 [0] Disconnect from the server
Please enter your request (1, 2, 3...without brackets)
Enter your request:
```

User login

SSL/TLS:

```python
#create context
context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
#context.verify_mode = ssl.CERT_REQUIRED
context.load_cert_chain(certfile=server_cert, keyfile=server_key)
#context.load_verify_locations(cafile=client_certs)

#create socket
bindsocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
bindsocket.bind((listen_addr, listen_port))
bindsocket.listen(5)
```

Server side ssl creation

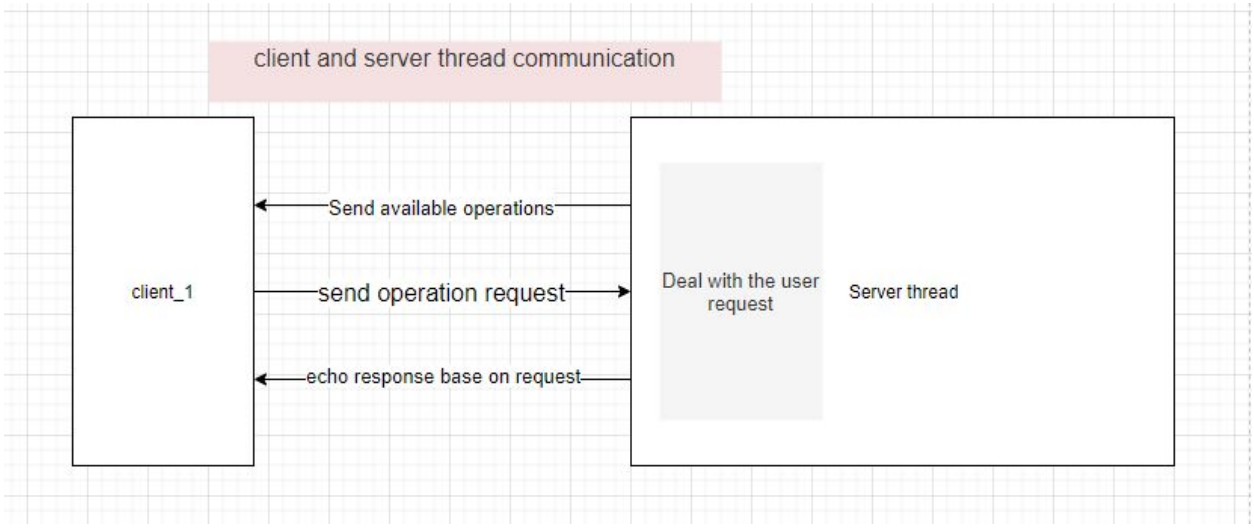
- Server has self-signed certificate(server_cert) and server_key

```python
def init_client(self):
    # create ssl context
    self.context = ssl.create_default_context(ssl.Purpose.SERVER_AUTH, cafile=self.server_cert)
    # self.context.load_cert_chain(certfile=self.client_cert, keyfile=self.client_key)

    # create socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.sslsock = self.context.wrap_socket(sock, server_side=False, server_hostname=self.server_hostname)

    try:
        self.sslsock.connect((self.host_ip, self.port))
    except Exception as e:
        print("connection fail: ", e, " client exit.")
        sys.exit()
```

Client side ssl creation

- Where server_cert is the certificate save in local at client machine.

Client/Server Communication:




```
client.py
Connection Success
Username: user2
Password:
login success
Thank you for using the secure chat
Pleas pick one of the options below, eg. '0' for exit
 [1] Show friend list
 [2] Chat with friend
 [3] wait for chat connection
 [0] Disconnect from the server
Please enter your request (1, 2, 3...without brackets)
Enter your request:
```

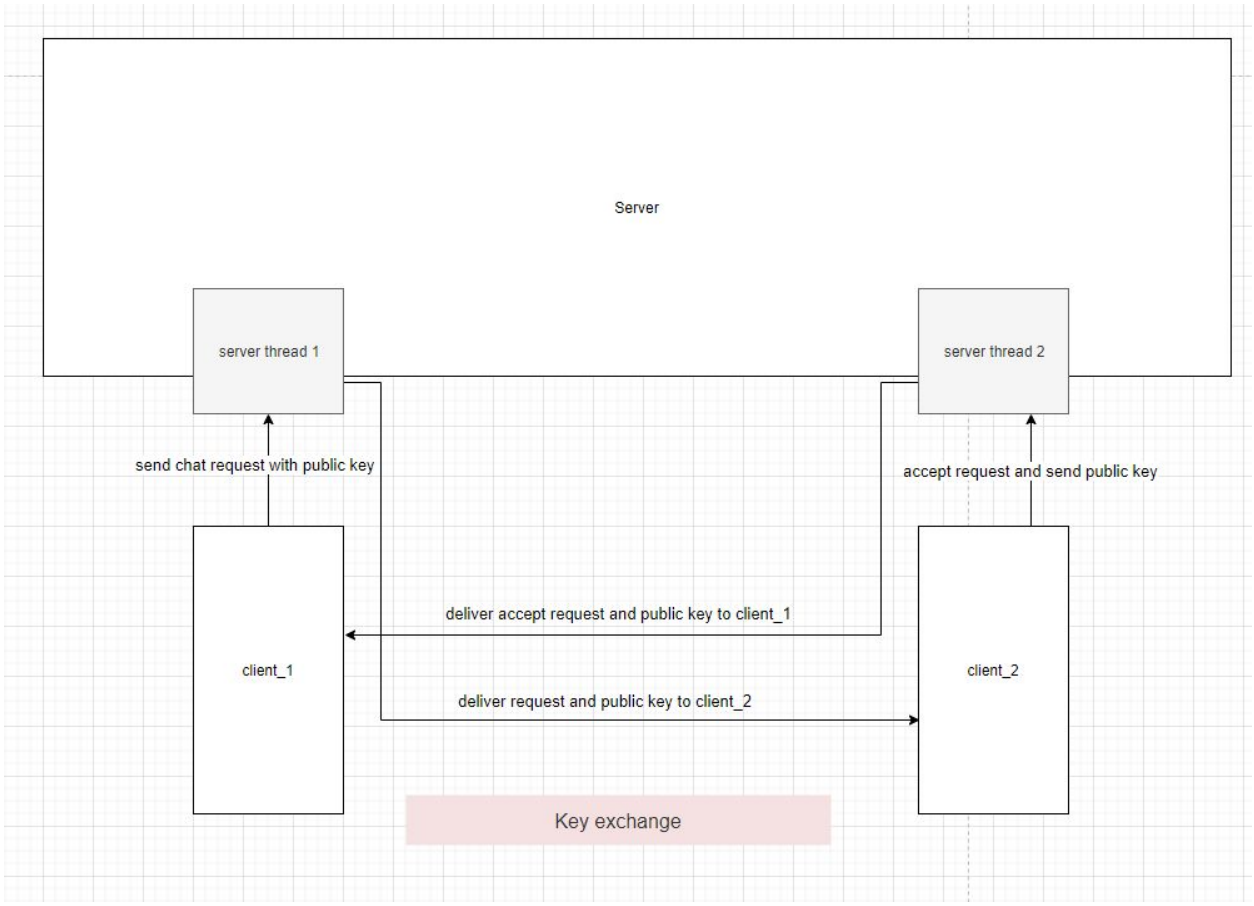
Login as regular user

```
Connection Success
Username: admin
Password:
login success
Welcome admin
Pleas pick one of the options below, eg. '0' for exit
 [1] Show connected users
 [2] add new user
 [0] Disconnect from the server
Please enter your request (1, 2, 3...without brackets)
Enter your request: 1
["user1", "admin"]
Enter your request: 0
Closing connection
```

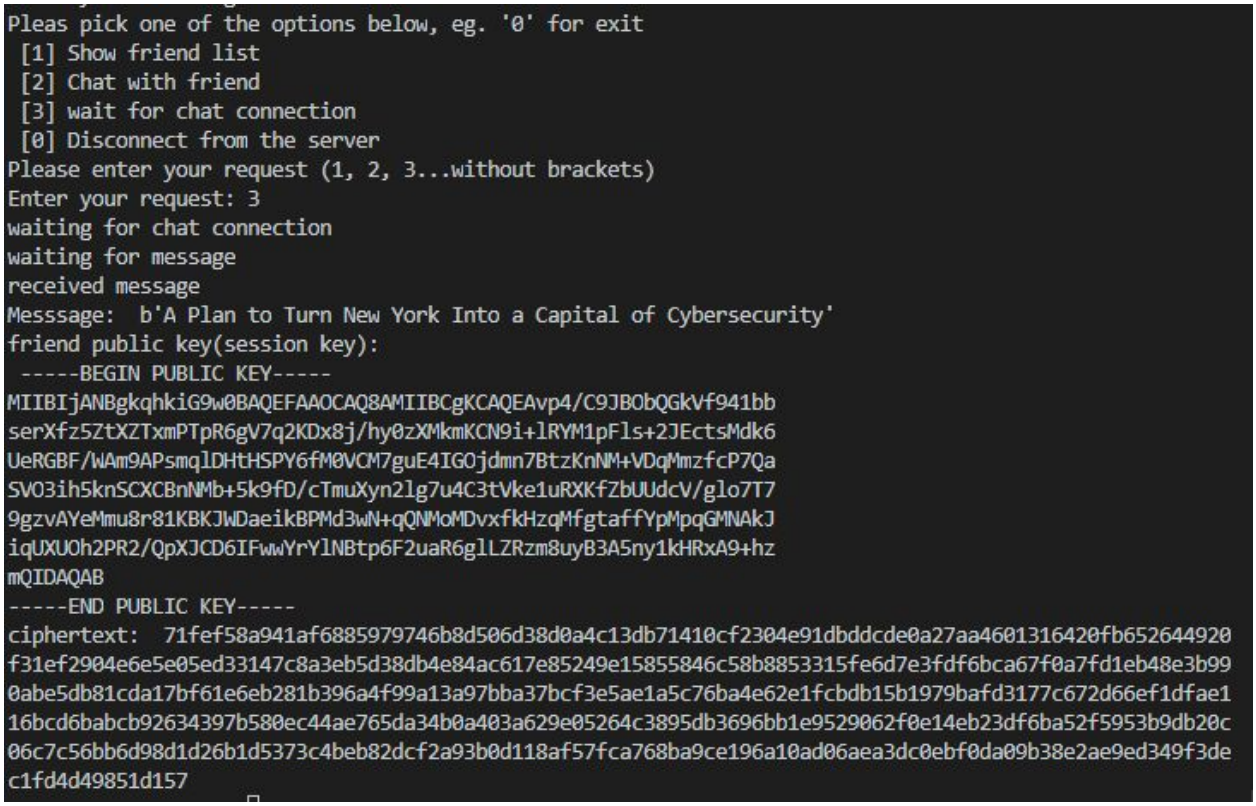
Login as admin user

- User has different accessibility base on their user group, add new user is not implemented

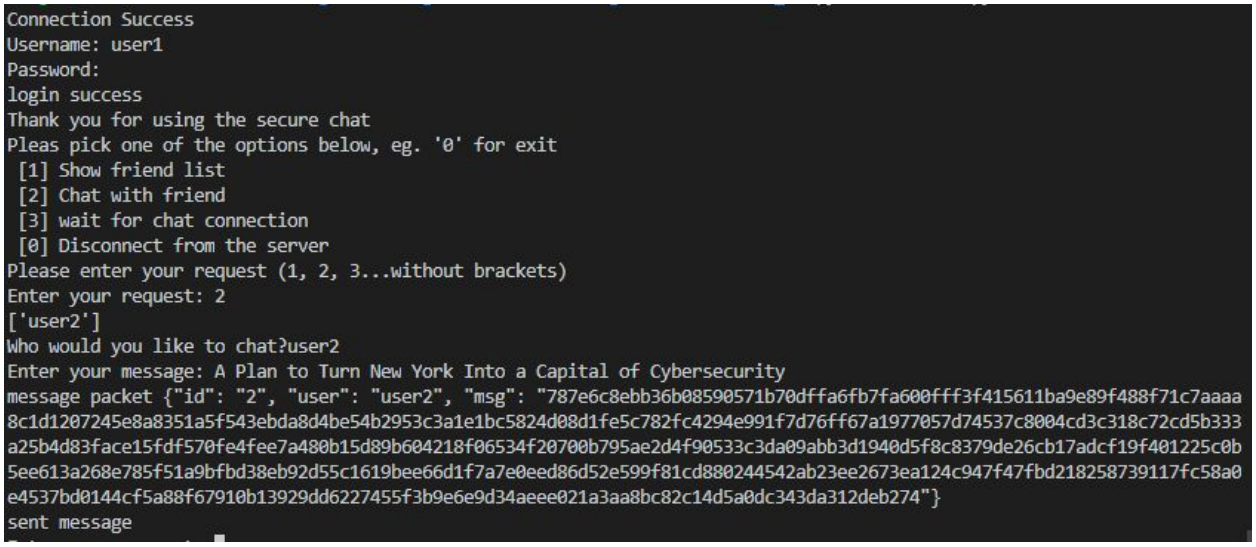
Key exchange and communication module:



- Client communication following the same idea above, client can send encrypted message to the server thread and server thread will be responsible to deliver the encrypted message to the target client_user.

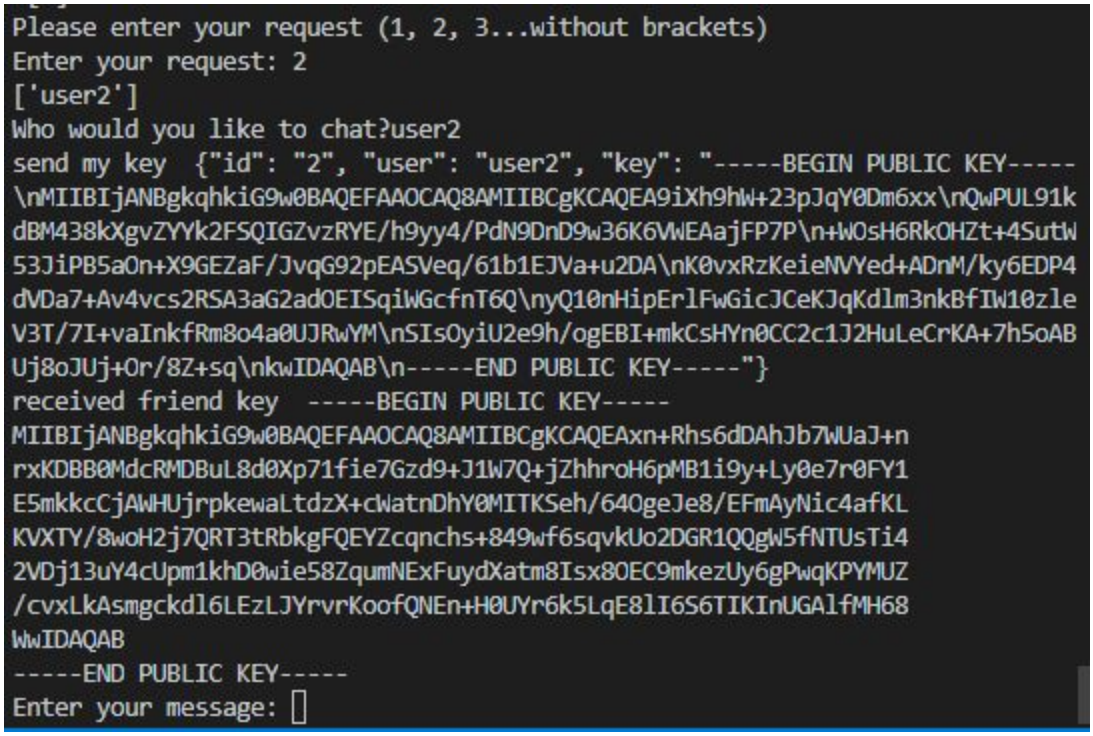


1. Receiver choose option 3 to wait for chat connection
2. Receiver receiver sender's public key and then, waiting for message
3. And display the result as project requested, the order above, decrypted message, session key and ciphertext

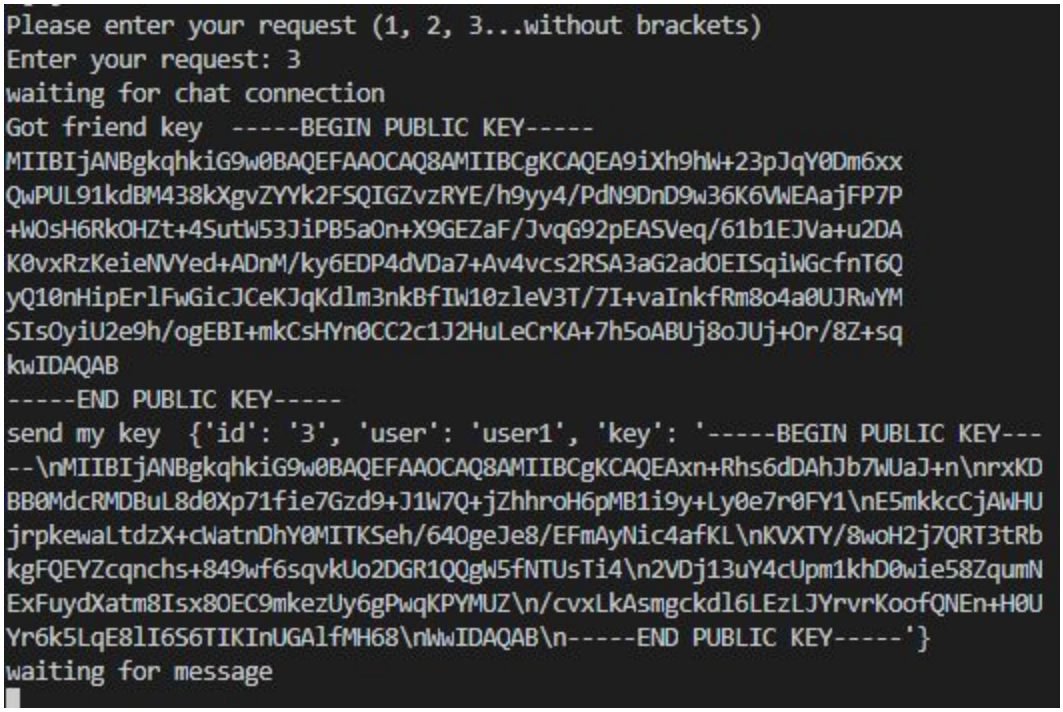


1. Sender choose friend to chat
2. Sender received receiver's public key
3. Sender input the message and send cipher text

Screenshot with key exchange:



Sender above, receiver below



Key pair generation:

```
# private and public key generation
private_key = RSA.generate(2048)
public_key = private_key.publickey()
session_key = public_key
```

Deal with offline client:

```
.py
Connection Success
Username: user1
Password:
login success
Thank you for using the secure chat
Pleas pick one of the options below, eg. '0' for exit
 [1] Show friend list
 [2] Chat with friend
 [3] wait for chat connection
 [0] Disconnect from the server
Please enter your request (1, 2, 3...without brackets)
Enter your request: 2
['user2']
Who would you like to chat?user2
user is not connected
Enter your request:
```

1. If user is not on the connected user list, send the message user is not online