Assignment Report

Program Design

This instant messaging application is based on a multiple Clients - Server model using TCP transport layer protocol. The Server uses multi-threading to handle each connecting client. The client uses a sub thread to receive response from the Server in the background while the main process handles the user commands and subsequently sends the requests to the Server.

Program Language and Environment

Python 3.8

Data Structure Design

- A global Python dictionary data structure is used working as the ‘database’ of the Server to store users’ info.

```
data = {
    'users': [ {
                'username':             str
                'password':             str
                'time_last_login':      int
                'time_last_logout':     int
                'time_frozen':          int
                'time_last_active':     int
                'soc':                  object
                'blocker':              set()
                'blockee':              set()
                'recvMsg':              []
                } ]
```

- Server interacts with the global data (db.py) through a list of APIs.

```
def loadTxtFile():…
def loadTxtUser(name, psw):…
def addNewUsrToTxtFile(username, psw):…
def getUserInfo(name):…
def addNewUsrIntoDB(name, psw, socket):…
def freezeLogin(name):…
def logInUser(name, socket):…
def logOutUser(name):…
def updateActiveTime(name):…
def isUsrOnline(name):…
def getOnlineUsr(name):…
def getSinceUsr(name, sinceTime):…
def getNonBlockedReciSock(sender):…
def getNonBlockedSenderSock(sender):…
def getMsgRecipSock(name):…
def setBlock(ber, bee):…
def setUnBlock(ber, bee):…
def isOnBlacklist(sender, receiver):…
def isBlocked(name):…
def addOfflineMsg(name, message):…
def getOfflineMsg(name):…
def getAllUsers():…
```

Application Layer Message Format

Client			Server	
Command	**Action**		**Action**	**Condition**
Login	SEND 'username'	->		
		<-	SEND 'Sign-up'	New user
			SEND 'Log-in'	Existing user
	SEND 'password'	->		
		<-	SEND 'Login Success'	Password is correct
			SEND 'Login Blocked'	3 password attempts failure
			SEND 'Login Failed'	1 password attempt failure
Logout	SEND 'logout'	->		
			PRINT '{user} Logout'	
whoelse	SEND 'whoelse'	->		
		<-	SEND '[NO ONE CURRENTLY ONLINE]'	No user currently online
			SEND {user}	At least one user online
whoelsesince <time>	SEND 'whoelsesince'	->		
		<-	SEND '[NO ONE ONLINE SINCE]'	No user online since
			SEND {user}	At least one user online since
block <user>	SEND 'block'	->		
		<-	SEND '{user} is blocked'	Block succeed
unblock <user>	SEND 'unblock'	->		
		<-	SEND '{user} is unblocked'	Unblock succeed
			SEND 'Error. {user} was not blocked'	Unblock Non-blocked user
message <user> <message>	SEND 'message'	->		
		<-	SEND 'Error. Invalid user'	The recipient is not existed
			SEND '[SENDER BLOCKED]Your message could not be delivered as the recipient has blocked you'	The recipient has blocked the sender
			SEND '{user}: {msg}'	Message sent
broadcast <message>	SEND 'broadcast'	->		
		<-	SEND 'Your message could not be delivered to some recipients'	Broadcaster is blocked by the recipients
read	SEND 'read'	->		
		<-	SEND '[NO OFFLINE MESSAGE]'	No offline message for the user
			SEND {offline_message}	Offline message sent

System Manual

- Step1: Run '**python3 server.py** <**server_port**> <**block_duration**> <**timeout**>' command.
- Step2: Run '**python3 client.py** <**server_port**>' command.
- Step3: Type in the username only if new to the application then password, or the correct username and password to proceed to the next phase of interaction.
- Step4: Once the authentication completes, type in the supported commands (**whoelse**/**whoelsesince**/**message**/**broadcast**/**block**/**unblock**/**read**) to start interacting with the Server and other Clients.
- Step5: Once finished using the app, type in command(**logout**) to exit the app.

Design trade-offs

- Reading offline messages in client.py.

  The specs require the server to send all offline messages to the user immediately once the user has logged into the system. In this scenario, prior to inputting any command, the user will be 'forced' to read the whole offline messages as soon as they are into the application. However, I do not think this is a user-friendly design for the Client application. I believe the user should have some level of control over what they want to view on their end once they are logged in. To be able to control what a user wants to view is a better user experience design.

  To implement this concept into code, I have introduced a command '**read**' for client.py. A user can just simply type in '**read**' to read any offline messages from other users.