Python Project: Internet Chat Room Using

Kypros Tsolakis

(N0950765)

NOTTINGHAM TRENT UNIVERSITY

COMP 10082

Foundation of Computing &Technology-Programming

Dr. Tawfik Al-Hadhrami

December 6, 2020

Internet Chat Room Project Using Python

The main objective of the project is development of Internet Chatroom with the following functionality:

1. Client authentication
2. Client registration
3. Sending broadcast messages (to everyone)
4. Sending private messages (to a particular user)

In addition to that, I have been considering the following not functional requirements:

1. Improvement of user interface. Since it was not feasible to develop GUI application I have decided to use ANSI codes (supported by Mac and Windows 10 (as discussed in https://stackoverflow.com/questions/12492810/python-how-can-i-make-the-ansi-escape-codes-to-work-also-in-windows)) to set font attributes and to manage screen.
2. Authentication implementation requires a permanent storage. I have decided to use SQLite (https://www.sqlite.org/index.html) that does not require installing a special database server software having a database as a file from one hand and being able to benefit from SQL from the other.
3. In order to improve application usability I decided to avoid having such things like IP, port, etc., in the core code and implement configuration file.
4. In order to improve system usability I had to implement several times error handling. Due to the client-server nature of the system, errors should be communicated to the client.
5. In addition to the above the server part of the system should maintain a log.

Client-Server is a suggested application architecture. In order to implement the above functionality client and server should be able to communicate using TCP/IP. Moreover, in order to handle multiple simultaneous connections I had decided to use a threading model. The idea and implementation of core functionality I have adopted from https://github.com/TomPrograms/Python-Internet-Chat-Room

In order to meet functional and not functional requirements the following improvements were done:

1. Implemented a configuration file. The following article https://martin-thoma.com/configuration-files-in-python/ discussed different ways of configuration files implementation in Python. I have decided to use the easiest, i.e. Python Configuration File, which, in fact, defines a dictionary with necessary configuration parameters like:

```
config = {
   "host" : "0.0.0.0",
   "port" : "9000",
   "dbfilename": r"c:\Users\ADMIN\Desktop\N0950765 Python Project
\usersdb321.db",
   "logfolder": r"c:\Users\ADMIN\Desktop\N0950765 Python Project\server\logs",
   "ansi" : "no"
}
```

2. In order to improve a look of end-user interface I decided to use ANSI codes. The following article discusses ANSI activation for Windows 10 as well as how to implement ANSI in Python code. Finally I came to the following ‘colors’ class definition:

```
class colors:                    #   ANSII Escape Codes definition

   if client_config.config["ansi"] == "yes":
      fBlack      = '\u001b[30;1m'
      fRed        = '\u001b[31;1m'
      fGreen      = '\u001b[32;1m'
      fYellow     = '\u001b[33;1m'
      fBlue       = '\u001b[34;1m'
      fMagenta    = '\u001b[35;1m'
      fCyan       = '\u001b[36;1m'
      fWhite      = '\u001b[37;1m'

      bBlack      = '\u001b[40;1m'
      bRed        = '\u001b[41;1m'
      bGreen      = '\u001b[42;1m'
      bYellow     = '\u001b[43;1m'
      bBlue       = '\u001b[44;1m'
      bMagenta    = '\u001b[45;1m'
      bCyan       = '\u001b[46;1m'
      bWhite      = '\u001b[47;1m'
      Clear       = '\u001b[2J'
      Reset       = '\u001b[0m'
   else:
      fBlack      = ''
      fRed        = ''
      fGreen      = ''
      fYellow     = ''
      fBlue       = ''
      fMagenta    = ''
      fCyan       = ''
      fWhite      = ''

      bBlack      = ''
      bRed        = ''
      bGreen      = ''
      bYellow     = ''
      bBlue       = ''
      bMagenta    = ''
      bCyan       = ''
      bWhite      = ''
      Clear       = ''
      Reset       = ''
```

The above class depending on ‘ansi’ configuration parameter produces relevant ANSI codes if ‘ansi’ is ‘yes’ and empty strings otherwise

Switching fonts and managing screen is as simple as printing relevant code like:

```
print (colors.Clear)
print (colors.fYellow)
```

3. The implementation of client and server TCP sockets is a standard:
   Server:

```
self.s.bind((host,port))
self.s.listen(100)
```

   where ‘host’ and ‘port’ are IP address and port of the server socket taken from the configuration file and 100 is a max number of supported connect requests.

   For the client socket, I am using the following code:

```
self.s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
```

   I have got the above from the following article:
   https://docs.python.org/3/howto/sockets.html

4. Client authentication requires using a database on a server side. In order to work with the database I am using ‘code first’ approach, i.e. if the database file cannot be found, my server script creates the database:

```
if not c_file.is_file():       # Create Database

        sql = """ CREATE TABLE "users" (
                "username"  TEXT,
                "FirstName" TEXT,
                "LastName"  TEXT,
                "Password"  TEXT,
                PRIMARY KEY("username")
        );"""

        try:
            conn = None                              # Connect to SQLite
            conn = sqlite3.connect(db_file)
            cur = conn.cursor()
            cur.execute(sql)
            conn.commit()
            conn.close()
            print("Created new database file '" + colors.fYellow + db_file + colors.Rese
t + "'.")
            self.write_log("Created new database file '" + db_file + "'.")
        except:
            print(colors.fRed + "Cannot create database. Quitting." + colors.Reset)
            self.write_log("Cannot create database. Quitting.")
            quit()
```

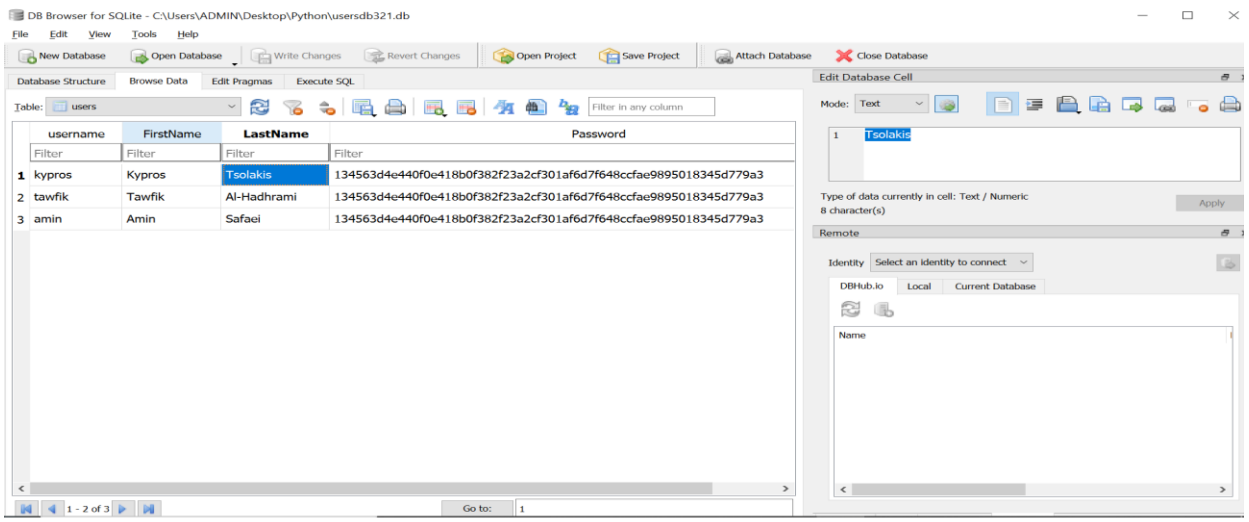
In order to check existence of the file I am using the following code (the database file name is defined in the configuration file)

```
    db_file = config.config["dbfilename"]

# Check if database exists, if not then create one
c_file = Path(db_file)
if not c_file.is_file():        # Create Database
```

As discussed in https://stackoverflow.com/questions/82831/how-do-i-check-whether-a-file-exists-without-exceptions

The database data looks like (I am using SQLiteDatabaseBrowserPortable_3.12.0_English browser):



DB Browser for SQLite - C:\Users\ADMIN\Desktop\Python\usersdb321.db

Table: users

	username	FirstName	LastName	Password
1	kypros	Kypros	Tsolakis	134563d4e440f0e418b0f382f23a2cf301af6d7f648ccfae9895018345d779a3
2	tawfik	Tawfik	Al-Hadhrami	134563d4e440f0e418b0f382f23a2cf301af6d7f648ccfae9895018345d779a3
3	amin	Amin	Safaei	134563d4e440f0e418b0f382f23a2cf301af6d7f648ccfae9895018345d779a3

Edit Database Cell — Mode: Text — 1 Tsolakis

Type of data currently in cell: Text / Numeric
8 character(s)

In order to improve security of the system I am storing in the database password hashes. Password is hashed on the client side using hashlib (https://docs.python.org/3/library/hashlib.html):

```
    h = hashlib.sha256()
h.update(self.password.encode())
self.password = h.hexdigest()
```

Then authentication evolves looking up user’s record and comparing password hashes.

```
cur.execute("select password from users where username=?",(username,)) # Get user's
 Password Hash
                rows = cur.fetchall()

                if len(rows)==0:                # Zero result -> wrong username
                    c.send("#iBad".encode())        # Send back bad authentication
                else:
                    row = rows[0]
                    if row[0]==pwd:                # Password OK
                        c.send("#iOK".encode())
                        break
                    else:
                        c.send("#iBad".encode())    # Send back bad authentication
```

I've got general SQLite information as well as code snippets from https://www.sqlitetutorial.net/sqlite-python

5. Client requires hiding passwords when they are being typed. I am using 'getpass' discussed in https://stackoverflow.com/questions/9202224/getting-command-line-password-input-in-python

```
self.password=getpass.getpass(colors.fGreen+'Enter password --> '+colors.Reset)
```

Then password is being hashed and sent to the server:

```
h = hashlib.sha256()
            h.update(self.password.encode())
            self.password = h.hexdigest()
            self.s.send(("#l&"+self.username + "&" + self.password).encode())     # Sen
d password to the server
```

'#l' (l - login) is a flag that instructs the server for the need of authentication. The request will be processed by the server as follows:

```
if cmd == "#l":          ##################################### SIGN IN

                cur.execute("select password from users where username=?",(username
,)) # Get user's Password Hash
                rows = cur.fetchall()

                if len(rows)==0:                # Zero result -> wrong username
                    c.send("#iBad".encode())        # Send back bad authentication
                else:
                    row = rows[0]
                    if row[0]==pwd:                # Password OK
                        c.send("#iOK".encode())
                        break
                    else:
                        c.send("#iBad".encode())    # Send back bad authentication
```

'&' is a separator that I have chosen for the protocol. The message received from the socket is parsed using split () function:

```
mess = c.recv(1024).decode().split('&')          # Get username & Password Hash from
Client and parse the response
```

Alternatively client can generate and send to the server #r (r – register) registration request:

```
 ########################################   LOGIN (LOOP UNTIL SUCCESSF
UL LOGIN) ############################
        self.username = input(colors.fGreen+'Enter username --
> '+colors.Reset)       # Get username
        self.firstname = input(colors.fGreen+'Enter First Name --
> '+colors.Reset)       # Get first name
        self.lastname = input(colors.fGreen+'Enter Last Name --
> '+colors.Reset)       # Get last name

        while True:                                             # Get password. Loop
until they match
          self.password=getpass.getpass(colors.fGreen+'Enter password --
> '+colors.Reset)
          self.password1=getpass.getpass(colors.fGreen+'Confirm password --
> '+colors.Reset)
          if self.password == self.password1:
            break
          else:
            print(colors.fRed+"Passwords do not match!"+colors.Reset)

        h = hashlib.sha256()
        h.update(self.password.encode())
        self.password = h.hexdigest()
        self.s.send(("#r&"+self.username + "&" + self.password + "&" + self.firstname
+ "&" + self.lastname).encode())    # Send credentials to the server
        resp = self.s.recv(1204).decode()
        if resp[0:5]=="ERROR":
          print(colors.fRed + resp + colors.Reset)
          input(colors.fRed + "Press ENTER to continue..." + colors.Reset)
        else:
          print(colors.fGreen + resp + colors.Reset)
          break
```

In such a case server creates new user record:

```
if cmd == "#r":         ######################################## SIGN ON
  firstname = mess[3]
  lastname = mess[4]
  try:
    cur.execute("insert into users (username, FirstName, LastName, Password) values (
?,?,?,?)", (username, firstname, lastname, pwd,))
    conn.commit()
    print("Created new user '" + colors.fYellow + username + colors.Reset + "'")
    self.write_log("Created new user '" + username +  "'")
    c.send(("New user '" + username + "' created.").encode())
    break
  except:
    print("Cannot create new user '" + colors.fYellow + username + colors.Reset + "'. Us
er exists.")
    self.write_log("Cannot create new user '" + username +  "'. User exists.")
    c.send(("ERROR creating new user '" + username + "'. User exists.").encode())
```

6. By default server broadcasts received message to all connected users except the sender:

```
for connection in self.clients:
                        if connection != c:                # Send to everybody except self
                            connection.send(msg)
```

Where ‘self.clients’ is a list of connections. List is maintained by

```
self.clients.append(c)
```

Additionally in order to map username to a connection and vice versa I am using two dictionaries ‘username_lookup’ and ‘connection_lookup’. Those dictionaries are maintained by the following code:

```
self.username_lookup[c] = username                                             self.connection_look
up[username] = c
```

And can be used to send personal messages.

The personal messages begins with @<*username*>*.* ‘@’ is an indication to the server that personal message should be processed:

```
 pm = m.split("@")
               if len(pm)>1:   # There is '@'. Send personal message
                   pma = pm[1].split(' ')
                   au = pma[0] # addressee
                   try:
                       cc = self.connection_lookup[au]
                       cc.send(mm.encode())
                       print("Personal message sent to user '" + colors.fYellow + au + colors.Re
set + "'")
                       self.write_log("Personal message sent to user '" + au + "'")
                   except:
                       print("User '" + colors.fYellow + au + colors.Reset + "' not found. Messag
e '" + colors.fYellow + mm + colors.Reset + "' not sent.")
                       self.write_log("User '" + au + "' not found. Message '" + mm + "' not sent."
)
                       c.send(("User '" + au + "' not found. Message not sent").encode())
```

Server retrieves from the message user name

```
                pma = pm[1].split(' ')
                au = pma[0] # addressee
```

Looks up connection:

```
cc = self.connection_lookup[au]
```

And, finally, sends the message

```
cc.send(mm.encode())
```

7. The last feature is logging. Server creates a new log file every time it starts. In order to make log file name unique I am using a timestamp as discussed in https://stackoverflow.com/questions/2961509/python-how-to-create-a-unique-file-name

   I have optimized the suggested code using strftime() method (https://www.programiz.com/python-programming/datetime/strftime). Thus, the code looks like:

```
self.log_filename = 'chatroom' + datetime.datetime.now().strftime('%Y_%m_%d_%H%M%S') + '.log'     # Datetimestamp based filename
self.log_file = open(config.config["logfolder"] + "\\" + self.log_filename, "w")
```

The first row in the log I am writing manually as:

```
 self.log_file.write(datetime.datetime.now().strftime('**** Chatroom server started on %Y.%m.%d at %H:%M:%S.%f') + "\n")
```

Then, in order to write to the log I am using a method defined like:

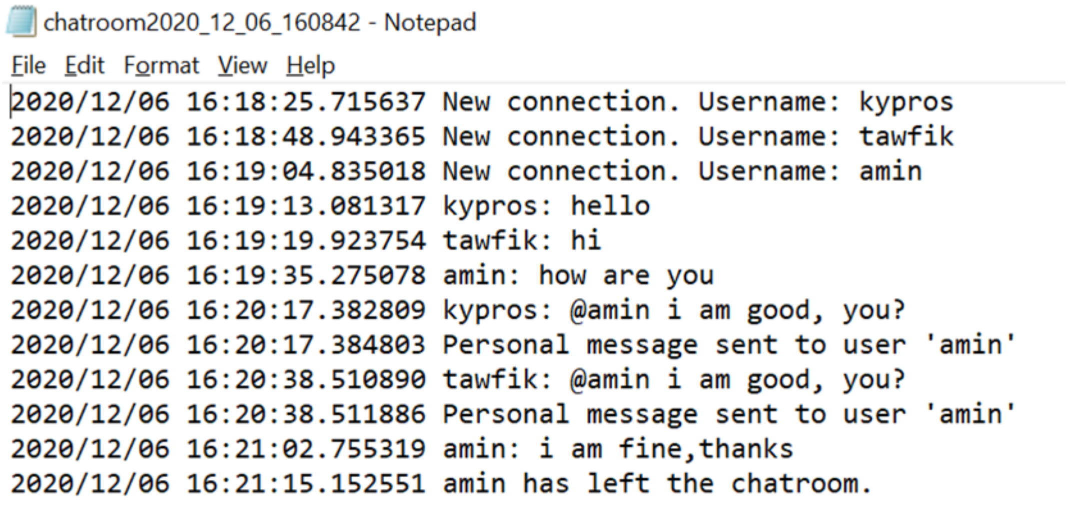
```
 def write_log(self,msg):
        self.log_file = open(config.config["logfolder"] + "\\" + self.log_filename, "a")
            # Opening/Closing a file in order to avoid losing data
        self.log_file.write(datetime.datetime.now().strftime("%Y/%m/%d %H:%M:%S.%f") + " " + msg + "\n")
        self.log_file.close()
```

I am opening and closing the log file every time we write a message in order to avoid any data loss.

8. In order to terminate client end-user must enter #Q command which will be processed as follows:

```
            if si[1].capitalize()=="Q":
                self.s.close()
                break
```

Below is test report from server logs file.


chatroom2020_12_06_160842 - Notepad
File Edit Format View Help

```
2020/12/06 16:18:25.715637 New connection. Username: kypros
2020/12/06 16:18:48.943365 New connection. Username: tawfik
2020/12/06 16:19:04.835018 New connection. Username: amin
2020/12/06 16:19:13.081317 kypros: hello
2020/12/06 16:19:19.923754 tawfik: hi
2020/12/06 16:19:35.275078 amin: how are you
2020/12/06 16:20:17.382809 kypros: @amin i am good, you?
2020/12/06 16:20:17.384803 Personal message sent to user 'amin'
2020/12/06 16:20:38.510890 tawfik: @amin i am good, you?
2020/12/06 16:20:38.511886 Personal message sent to user 'amin'
2020/12/06 16:21:02.755319 amin: i am fine,thanks
2020/12/06 16:21:15.152551 amin has left the chatroom.
```

Future improvements:

1. **Security**

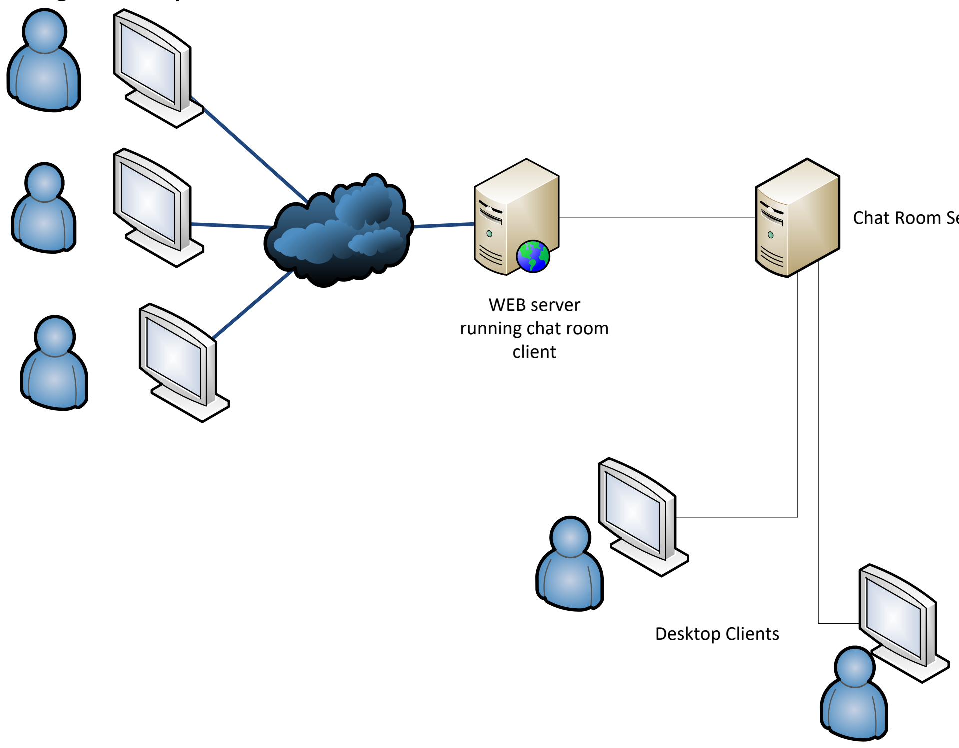
   The chatroom is not secure. One of the features that I would like to add is encryption. Symmetric encryption is pretty well discussed in the following article: https://devqa.io/encrypt-decrypt-data-python/ . Keys can be defined in the configuration file and communicated to clients using different channels.

2. **GUI**

   The current implementation is not very usable. In particular, sending private messages requires end-user to memorize other users' names. It would be much easier if, for example, user names could be selected from the drop-down list. However, drop-down list implementation is problematic within CLI paradigm. At the same time it could be easily implemented using GUI.

3. **WEB**

   The usability could be improved even higher using WEB interface for clients. WEB server could also be implemented in Python using 'flask' (https://flask.palletsprojects.com/en/1.1.x/) platform. In such a case we would need to reconsider client architecture significantly for it will be presented by the WEB server where clients can connect from WEB browsers. The modification of the server part is minimal. It will also be possible to connect to the server using desktop client software:



Installation notes

Server installation

Server software is located in the ‘Server’ folder. Copy the folder to a desired location on your machine. The content of the folder is as follows:

Server.py
Config.py

In order to run the software, you have to modify config.py file.

The content of the file is as follows:

```
config = {
   "host" : "0.0.0.0",
   "port" : "9000",
   "dbfilename": r"c:\Users\Admin\Desktop\server\usersdb.db",
   "logfolder": r" c:\Users\Admin\Desktop\server \logs",
   "ansi" : "no"
}
```

‘host’ - IP address that should be bound to the server socket. If client and server are running on the same computer it can be loopback IP, i.e. 127.0.0.1. If server socket should be bound to *all* IP addresses configured on the host, it should be 0.0.0.0.
‘port’ - desired port. Can be any non-privileged, unused port.
‘dbfilename’ – user database file name. You have to specify fully qualified file name including drive and path. If database does not exist it will be created automatically. Then all users should sign on before using the chat room
‘logfolder’ – specify fully qualified path to the file where you want to keep system logs. Server creates a log file each time it starts. Datetime stamp is used to create unique file names like chatroom *<year>_<month>_<day>_<HHMMSS>.*
‘ansi’ – ‘yes’ or ‘no’. Use ‘yes’ if your display is configured for ANSI codes (default for MAC, Windows 10), otherwise, use ‘no’

Running the server.

1. Run command prompt (Windows) or terminal (Mac)
2. Change folder to the folder that contains server software. In my case it was
   cd c:\users\ADMIN\Desktop\N0950765 Python Project\server
3. Run server.py script either from command prompt (Windows) or terminal (Mac) like

python server.py

Upon successful start the following message will be displayed:

```
Running on host: 0.0.0.0
Running on port: 9000
```

Optionally system can inform you about creating a new user database file like:\
Created new database file 'c:\users\ADMIN\Desktop\N0950765 Python Project\server\usersdb.db'
When you run the server on Windows for the first time you have to allow using the server port:

Client installation

Client software is located in the 'Client' folder. Copy the folder to a desired location on your machine. The content of the folder is as follows:

client.py
client_config.py

In order to run the software you have to modify config.py file. The content of the file is as follows:

```
config = {
   "host" : "localhost",
   "port" : "9000",
   "ansi" : "yes"
}
```

'host' - Name or IP address of the computer running server. If you are running client and server on the same machine it can be either '127.0.0.1' or 'localhost'
'port' - server's port. You must use the same port specified for the server (see above)
'ansi' – 'yes' or 'no'. Use 'yes' if your display is configured for ANSI codes (default for MAC, Windows 10), otherwise, use 'no'

Running the client

1. Run command prompt (Windows) or terminal (Mac)
2. Change folder to the folder that contains server software. In my case it was

```
cd c:\users\ADMIN\Desktop\ N0950765 Python Project\client
```

3. Run client.py script either from command prompt (Windows) or terminal (Mac) like

```
python client.py
```

Upon successful start the following message will be displayed:

```
Connected to host: localhost
Port: 9000
Press Enter to continue...
```

Server script must be running. Otherwise you are going to get error connecting to the server!

4. Press ‘Enter’. The system will display the following menu:

```
Sign in to the chatroom -------------------- 1
Sign on to the chatroom -------------------- 2
Quit --------------------------------------- Q

Select your option and press Enter >
```

If you are running the system for the first time you have to choose the second option to register a new user

5. Choose ‘2’. Enter username, first name, last name, password and confirm the password:

```
Select your option and press Enter > 2
Enter username --> kypros
Enter First Name --> Kypros
Enter Last Name --> Tsolakis
Enter password -->
Confirm password -->
```

6. The system will display the following message:

```
New user '<username>' created.
You are connected to the chatroom. Type your message.
```

7. You can start using the system

System will notify you if other users are connected to the system