

**Project Report on**  
**CSN-503 Advanced Computer Networks**

**Group Chat Application**

**Submitted by :-**

**Group - 10**

17114004 Abhishek Rathod

17118007 Ajay Neeti Kannan

17114013 Anshuman Shakya

17114030 Suresh Babu Gangavarapu

17114008 Aman Jaiswal

17114009 Aman Verma

## **Contribution of Group members**

The server code using socket programming in server.py is implemented by : Ajay Neeti Kannan and Suresh Gangavarupu, Page No. - 3, 4, 5, 6

The client code using socket programming and GUI of application in clients.py is implemented by : Abhishek Rathod and Anshuman Shakya Page No. - 3, 4, 5, 6, 7

The database, SQL Queries and authentication of the application in server.py commands.py tables.py is implemented by : Aman Verma and Aman Jaiswal Page No. - 3, 8, 9, 10

## **Problem Statement**

To build a Group Chat Application using Socket Programming which can help students to communicate with each other. An authentication feature also needs to be implemented so that only the students of the hostel have the access.

## **Introduction**

We implemented a desktop application for the Group Chat App using python programming language.

Technologies used in the application are :-

1. Python 3.
2. Tkinter - The tkinter package ("Tk interface") is the standard Python interface to the Tk GUI toolkit.

3. Sqlite3 - SQLite is a C library that provides a lightweight disk-based database and allows accessing the database using a nonstandard variant of the SQL query language.

## **Source Code Files**

- 1) **server.py** - This is the server code file, with which all the clients connect. The server creates the socket for the TCP interaction. It performs the functions of authentication of the client. Also, it broadcasts the messages sent by one client to all the other clients. It stores all the messages received using the SQLite3 database for sending them whenever some client connects late with the server. It also maintains the users database table whenever the passwords are changed by the students, which are initially set by the admin.
- 2) **client.py** - This is the client code file which can be used by different students to connect to the server to access the chat application. The GUI of the chat interface is also implemented in this file. The functionality of login for users is also implemented in this file.
- 3) **commands.py** - This is the command line utility file. It has various commands like createsuperuser (that creates an admin account), register student (which is used by the admin to register a student account), userlist (prints the list of users present in the database), etc.
- 4) **table.py** - This is the Sqlite3 database management file. It provides the functionality of creating the database tables, authorizing the user, adding and removing users, updating passwords, adding and listing the messages, etc.

## **How to execute**

To run server - python3 server.py

To run client - python3 client.py

**NOTE - Always run the server file before initiating the connection.**

**[The following commands can be used by the admin]**

To create admin user - python3 commands.py createsuperuser

To register a user - python3 commands.py registerstudent

To delete a user - python3 commands.py deleteuser

To get the users list - python3 commands.py userlist

To get the list of messages - python3 commands.py messagelist

## **Features Implemented**

- Admin can add and delete students in the database.
- Students have to connect to the server as clients.
- Students can login to the app using credentials given by the admin.
- Students can change the initial password which was created by the admin.
- Students can then chat with each other using the chat room.
- The Admin Officials can post different hostel level notices for the students to view.
- Students can then discuss their views with other students and the admin.
- The students can also view other clients who are active on their screen.

## **MODULES USED**

### **1) Socket Programming**

Sockets and the socket API are used to send messages across a network. They provide a form of inter-process communication (IPC). The network can be a logical, local network to the computer, or one that's physically connected to an external network, with its own connections to other networks.

The primary socket API functions and methods we have used in this module are:

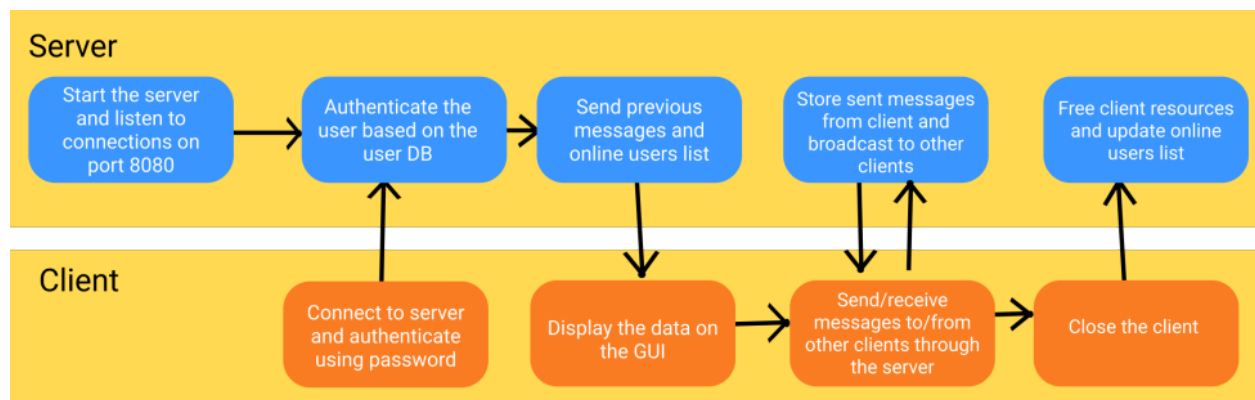
- `socket()` – To create a socket, you must use the `socket.socket()` function available in `socket` module, which has the general syntax –

`s = socket.socket (socket_family, socket_type, protocol=0)`

Here is the description of the parameters –

- `socket_family` – This is either `AF_UNIX` or `AF_INET`
- `socket_type` – This is either `SOCK_STREAM` or `SOCK_DGRAM`.
- `protocol` – This is usually left out, defaulting to 0.
- `bind()` – This method binds address (hostname, port number pair) to `socket`.
- `listen()` – This method sets up and starts a TCP listener.
- `accept()` – This passively accepts TCP client connection, waiting until connection arrives (blocking).
- `connect()` – This method actively initiates TCP server connection.
- `send()` – This method transmits TCP message
- `recv()` – This method receives TCP messages.
- `close()` – This method closes socket.

In the diagram below, we have explained the flow of server client communication using the socket programming in the application.



Diagrammatic representation of the application flow

- To meet our purpose of group chat, the connection of a client is set up with the server and then the server waits to receive messages from each client in different threads.
- When a client sends a message to the server, the server broadcasts the message to all the other clients.
- In the client, threading is done for receiving messages from the server and for running the GUI.

## 2) GUI implementation - Tkinter

Python has a lot of GUI frameworks, but **Tkinter** is the only framework that's built into the Python standard library. Tkinter has several strengths:-

- It's cross-platform, so the same code works on Windows, macOS, and Linux.
- Visual elements are rendered using native operating system elements, so applications built with Tkinter look like they belong on the platform where they're run.

The foundational element of a Tkinter GUI is the **window**. Windows are the containers in which all other GUI elements live.

These other GUI elements, such as text boxes, labels, and buttons, are known as **widgets**. Widgets are contained inside of windows.

- Label - A widget used to display text on the screen.
- Button - A button that can contain text and can perform an action when clicked.
- Entry - A text entry widget that allows only a single line of text.
- Text - A text entry widget that allows multiline text entry.
- Frame - A rectangular region used to group related widgets or provide padding between widgets.

Text widgets are used for entering text, just like Entry widgets. The difference is that Text widgets may contain multiple lines of text. With a Text widget, a user can input a whole paragraph or even several pages of text! Just like Entry widgets, there are three main operations you can perform with Text widgets:

1. Retrieve text with `.get()`
2. Delete text with `.delete()`
3. Insert text with `.insert()`

### 3) Database Used - SQLite3

SQLite is a C library that provides a lightweight disk-based database that doesn't require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language.

We chose to go with SQLite since it is already a builtin library available with python3 and is very easy to setup, as it uses a file to store the database. This suited our development needs well.

We used SQL to store persistent data about the students and the messages sent in the group chat.

For this project the database required two tables:-

- Users - to store different users and their information.
- Messages - to store all the messages of group chat.

**Database Design is as follows :-**

```
CREATE TABLE users (  
    pid            INTEGER PRIMARY KEY NOT NULL,  
    name           TEXT NOT NULL,  
    enr_no        INT,  
    password       CHAR(50) NOT NULL,  
    is_admin       BIT  
);
```

```
CREATE TABLE messages (  
    mid           INTEGER PRIMARY KEY,  
    sender        INT NOT NULL,  
    date          DATETIME DEFAULT CURRENT_TIMESTAMP,  
    data          CHAR(256),
```



```
FOREIGN KEY (sender) REFERENCES users(pid) ON  
DELETE CASCADE  
);
```

The admins who have access to the server, can use the terminal to register new students. This can be done by running the command

```
> python commands.py registerstudent
```

Users can be removed as well using the removestudent command

```
> python commands.py removestudent
```

Other command line functions for the admins include ability to get all the messages, get all the users, create new admins, etc.

We handle the authentication aspect by verifying whether the user and password given is present in the users table.

We also store all the messages sent by the user in the database, and storing the user info as well, using a foreign key.

We handle all the errors caused while accessing the Database smoothly and pass proper output to the server.

Some important functions written in the table.py are:

Authorise\_user: This checks whether the username logs in with his/her password, to allow the socket to connect to the chat application.

Add\_user: This function can be used both in the server as well as the command line, to create new students and new admins

Update\_password: Allows users to change their password, which was set by default using the admin while creating the students accounts.

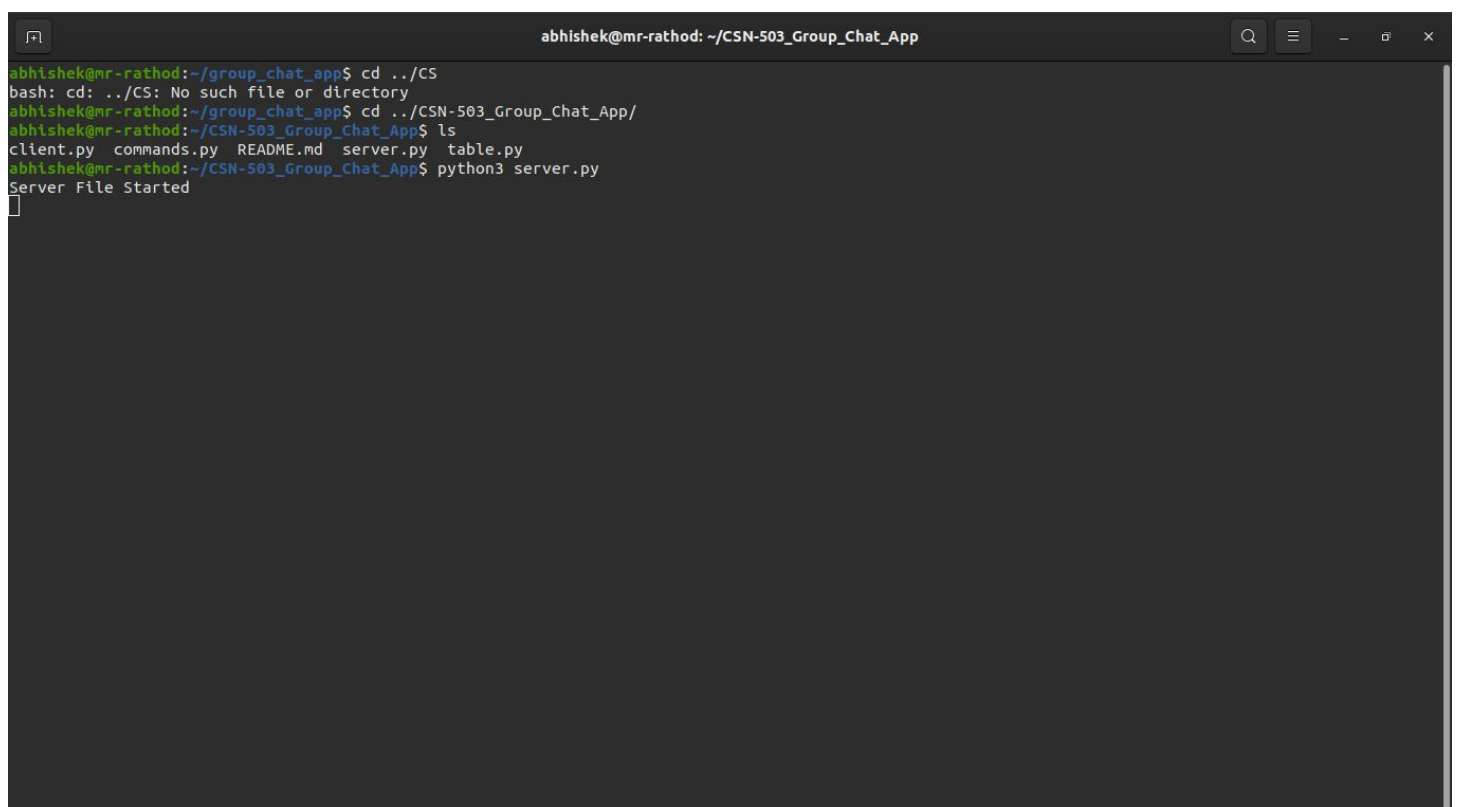
List\_messages: This is used to list all the messages in order, which were entered in chat.

Add\_message: This is used to store new messages sent by user and store in the database.

List\_users: This is used to list all the users, used by the server to get more information about the users such as username, is\_admin, which is used while broadcasting messages to the connected clients.

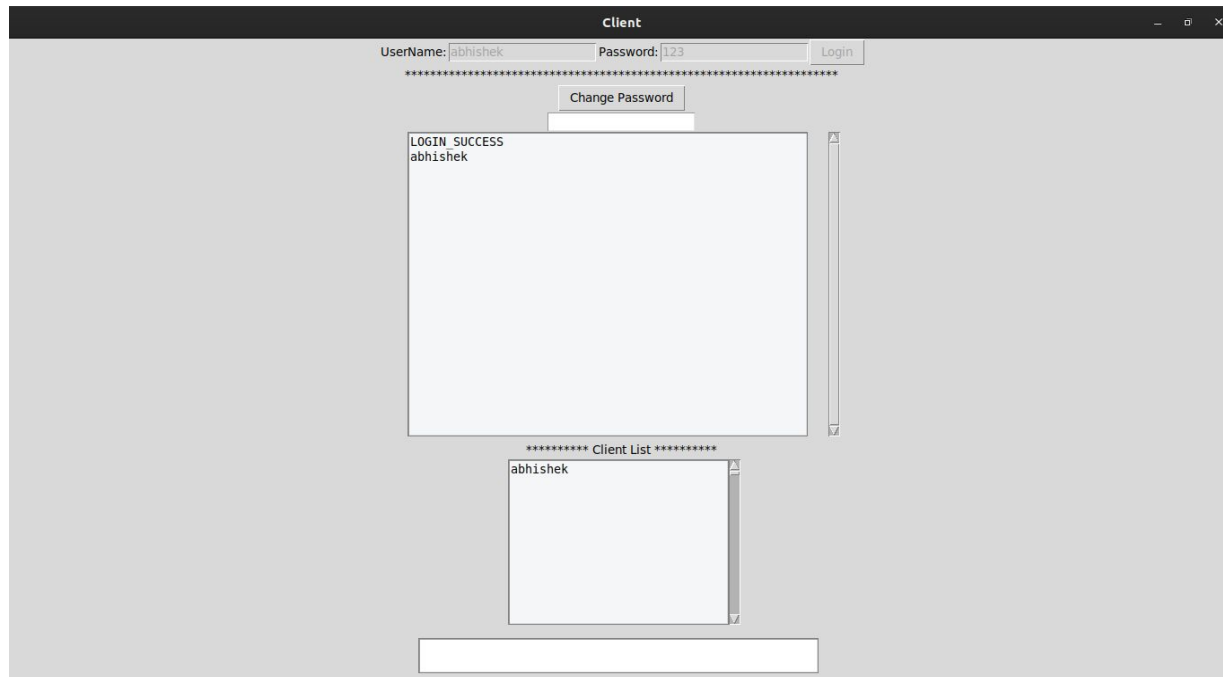
## **Screenshots of the Application**

### 1) Starting the Server Code File.

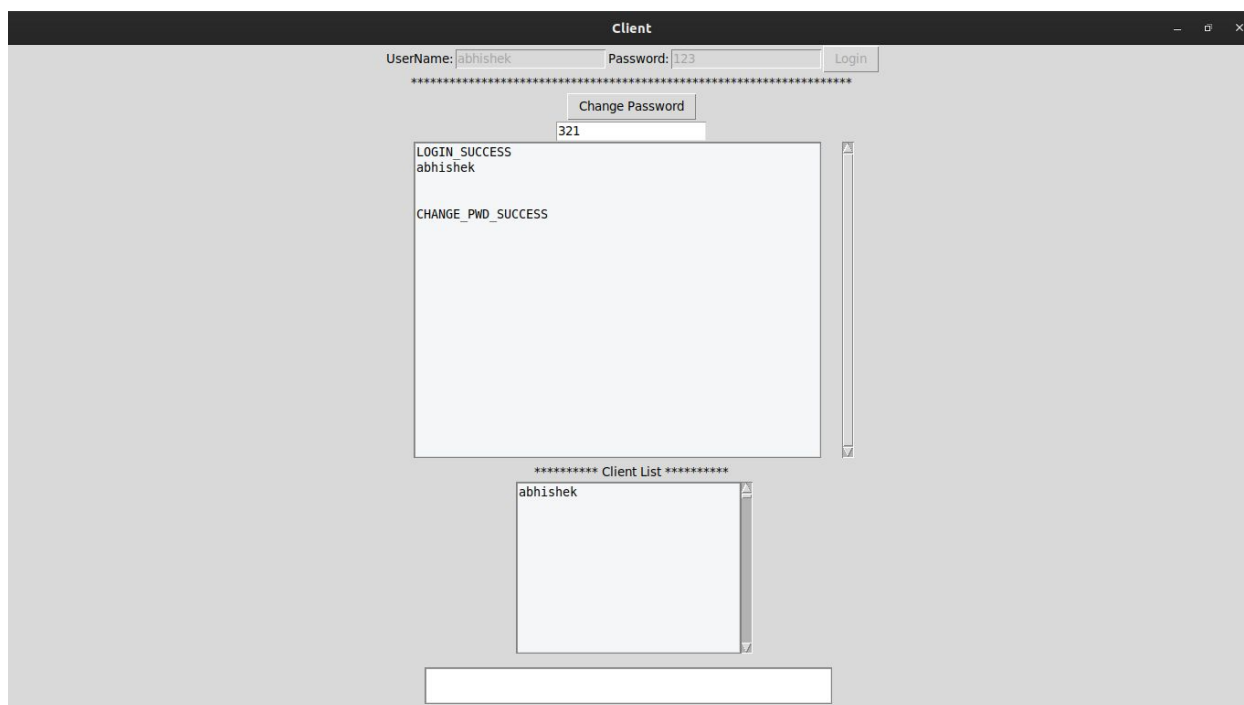


```
abhishek@mr-rathod: ~/CSN-503_Group_Chat_App
abhishek@mr-rathod:~/group_chat_app$ cd ../CS
bash: cd: ../CS: No such file or directory
abhishek@mr-rathod:~/group_chat_app$ cd ../CSN-503_Group_Chat_App/
abhishek@mr-rathod:~/CSN-503_Group_Chat_App$ ls
client.py  commands.py  README.md  server.py  table.py
abhishek@mr-rathod:~/CSN-503_Group_Chat_App$ python3 server.py
Server File Started
█
```

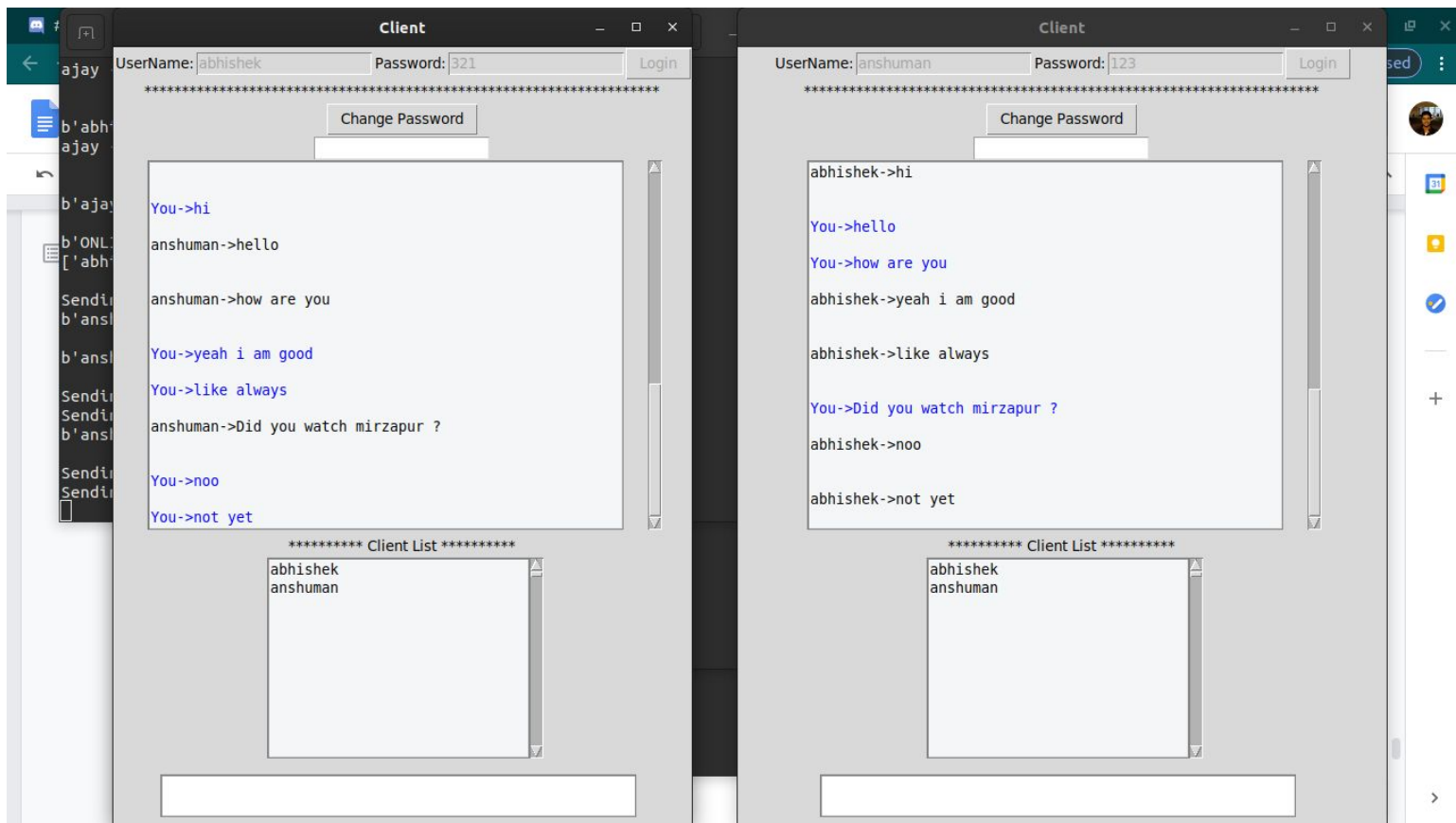
## 2) The After Login Screen.



## 3) After changing the password Screen.



#### 4) The Chat Screen.



## **REFERENCES**

<https://docs.python.org/3/howto/sockets.html>

<https://docs.python.org/3/library/tkinter.html>

<https://docs.python.org/3/library/sqlite3.html>