# CHAT APPLICATION

MINOR PROJECT REPORT

By

**TUMATI OMKAR CHOWDARY (RA2211026010188)**
**AKULA SATYA SESHA SAI (RA2211026010168)**
**PALAMETI REDDY LAKSHMI MANOJ (RA2211026010179)**

Under the guidance of

**Mrs. V. Indumathi**

*In partial fulfilment for the Course*

of

**21CSC203P – ADVANCED PROGRAMMING PRACTICE**

in Department Of Computational Intelligence

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**SCHOOL OF COMPUTING**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR**

**NOVEMBER  2023**

# SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

**(Under Section 3 of UGC Act, 1956)**

## BONAFIDE CERTIFICATE

Certified that this minor project report for the course **21CSC203P ADVANCED PROGRAMMING PRACTICE** entitled in "**ONLINE E-COMMERCE CLOTHING STORE** " is the bonafide work of **Anish Khadamkar (RA2211033010169), Aayush Doshi (RA2211033010171) and Samyak Mutha (RA2211033010173)** who carried out the work under my supervision.

**SIGNATURE**

**PROJECT GUIDE**

Mrs. V. Indumathi  
**Teaching Associate**  
Department of Computational Intelligence  
SRM Institute of Science and Technology  
Kattankulathur

**SIGNATURE**

**HEAD OF THE DEPARTMENT**

Dr. Annie Uthra  
**Professor & Head**  
Department of Computational Intelligence  
SRM Institute of Science and Technology  
Kattankulathur

# ABSTRACT

Chat refers to the process of communicating, interacting and/or exchanging messages over the Internet. It involves two or more individuals that communicate through a chat-enabled service or software. Chat may be delivered through text, audio or video communication via the Internet.

Chat applications are computer programs that allow users to communicate with each other in real time. They typically consist of a client application, which is installed on the user's computer, and a server application, which is hosted on a remote server. The client application connects to the server application, and the two applications then exchange messages back and forth.

Chat applications can be used for a variety of purposes, including personal communication, business communication, and customer support. They are often used to communicate with people who are located in different parts of the world, and they can be a valuable tool for staying in touch with friends and family.

The chat application we are going to make will be more like a chat room, rather than a peer-to-peer chat. So, this means that multiple users can connect to the chat server and send their messages. Every message is broad casted to every connected chat user.

skdhvbs

## ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V. Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman,** for imparting confidence to complete my course project.

We wish to express my sincere thanks to **Course Audit Professors Dr. Vadivu. G, Professor, Department of Data Science and Business Systems and Dr. Sasikala. E Professor, Department of Data Science and Business Systems** and **Course Coordinators** for their constant encouragement and support.

We are highly thankful to our Course project Faculty **Mrs. V. Indumathi, Teaching Associate, Department of Computational Intelligence,** for her assistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to our **HoD, Dr. Annie Uthra, Professor & Head, Department of Computational Intelligence** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

# INTRODUCTION

In a world increasingly connected by technology, the need for seamless and instant communication has never been more prevalent. Introducing [Chat mingle], your gateway to effortless conversations, bridging distances and fostering connections with just a few taps.

[Chat mingle] is more than just a messaging app; it's a platform where ideas ignite, friendships blossom, and meaningful interactions thrive. With its user-friendly interface and a host of innovative features, [Chat mingle] redefines the way you connect with the world.

Your privacy is our priority. [Chat mingle] employs industry-leading encryption protocols to safeguard your conversations and personal information.

# MOTIVATION

Chat applications have become an integral part of our daily lives, facilitating communication and connection across geographical boundaries. The motivations behind using chat applications are diverse and stem from various personal and professional needs. Here are some of the key motivations driving the widespread adoption of chat applications:

Real-time communication: Chat applications enable instant, real-time communication, mimicking face-to-face interactions without the constraints of physical distance. This immediacy fosters a sense of connection and allows for spontaneous conversations and collaborations.

Convenience and accessibility: Chat applications are readily accessible on various devices, including smartphones, tablets, and computers. This convenience allows users to connect with others anytime, anywhere, making it an ideal tool for busy individuals and those with limited mobility

Community building: Chat applications often serve as online communities where individuals with shared interests can connect and engage. These communities provide a sense of belonging and foster meaningful interactions.

Multimedia support: Chat applications often support the sharing of multimedia content, including images, videos, and audio. This feature enhances communication by allowing users to express themselves more vividly and share their experiences with others.

# OBJECTIVE

Chat applications serve a variety of objectives, catering to different user needs and scenarios. Here's a comprehensive overview of the key objectives of chat applications:

Enhanced communication and collaboration: Chat applications provide a platform for enhanced communication and collaboration, particularly in workplace settings. They enable project discussions, task coordination, and efficient team communication, streamlining workflows and improving productivity.

Information sharing and dissemination: Chat applications can serve as a valuable tool for information sharing and dissemination. They allow users to share news, updates, and resources with their connections, facilitating the spread of knowledge and keeping individuals informed.

Distance learning and education: Chat applications are being integrated into educational platforms to facilitate distance learning and enhance the learning experience. They enable online classes, group discussions, and student-teacher interactions, providing a virtual classroom environment.

In summary, chat applications serve a diverse range of objectives, catering to various user needs and scenarios. Their ability to facilitate real-time communication, enhance collaboration, and foster community building has made them an indispensable tool in our personal and professional lives. As technology continues to evolve, chat applications are likely to play an even more prominent role in our communication and interaction with the world around us.

# PROBLEM STATEMENT

In today's increasingly interconnected world, the need for efficient and secure communication tools is more prevalent than ever. While traditional methods like phone calls and email remain useful, they often lack the real-time interactivity and flexibility that modern communication demands. Chat applications have emerged as a popular solution, offering a convenient and accessible platform for instant messaging between individuals and groups. However, despite their widespread adoption, existing chat applications face several challenges that hinder their effectiveness and user satisfaction.

Scalability: As the number of users grows, chat applications must be able to handle the increasing volume of messages and maintain low latency for real-time communication.

Security: Chat applications must ensure the confidentiality and integrity of user data, protecting against unauthorized access and data breaches.

End-to-end encryption: Users should have the option to enable end-to-end encryption for their messages, ensuring that only the intended recipient can decrypt and view the content.

Group chat capabilities: Effective group chat features are essential for collaboration and community building, allowing users to engage in discussions with multiple participants simultaneously.

File sharing: Chat applications should enable users to share files, images, and other documents, facilitating collaboration and information exchange.

# CHALLENGES

Chat applications have become an integral part of our daily lives, providing a convenient and efficient way to communicate with friends, family, and colleagues. However, developing and maintaining a successful chat application presents a number of challenges.

## Technical Challenges:

Real-time communication: Chat applications need to handle real-time communication between users, ensuring that messages are delivered and displayed instantly. This requires a robust and scalable infrastructure that can handle a large volume of messages and concurrent users.

Scalability: As the user base of a chat application grows, it needs to be able to scale to accommodate the increased traffic and data load. This involves optimizing the application's architecture, using efficient data storage solutions, and employing cloud-based infrastructure.

## User Experience Challenges:

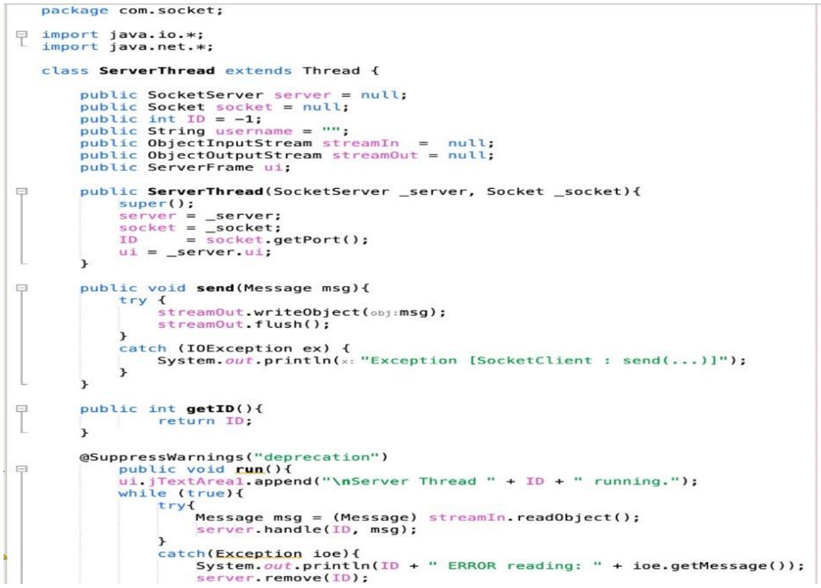
User engagement: Attracting and retaining users is crucial for the success of a chat application. This requires providing a user-friendly interface, offering valuable features, and ensuring a seamless communication experience.

Differentiation: The chat application market is saturated with competitors. Standing out from the crowd requires offering unique features, innovative design, and a strong brand identity.

User privacy and data protection: Users are increasingly concerned about their privacy and data protection. Chat applications need to be transparent about their data collection practices, implement robust privacy controls, and comply with relevant data protection regulations.

# JAVA IMPLEMENTATION

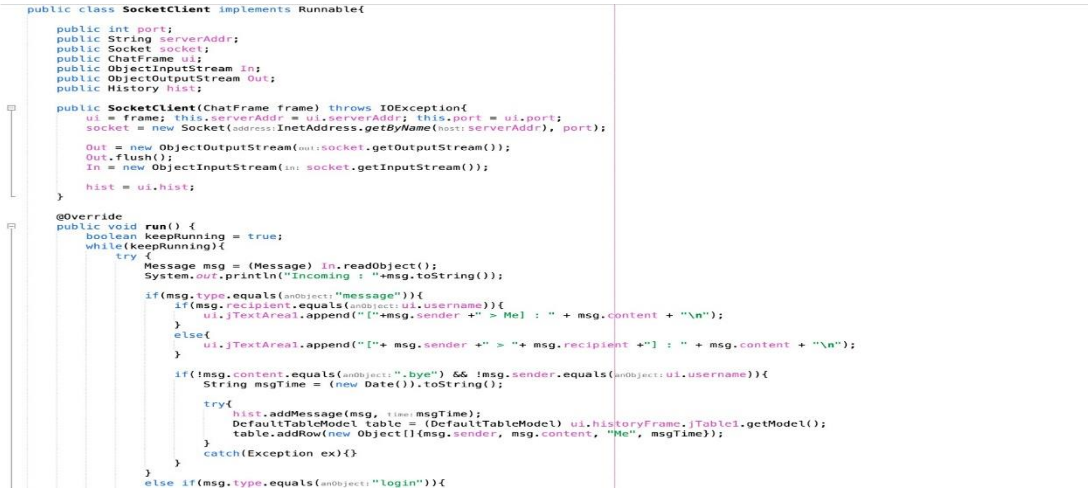
## STEP1: creating a socket-based Multi-client Server



```java
package com.socket;

import java.io.*;
import java.net.*;

class ServerThread extends Thread {

    public SocketServer server = null;
    public Socket socket = null;
    public int ID = -1;
    public String username = "";
    public ObjectInputStream streamIn  =  null;
    public ObjectOutputStream streamOut = null;
    public ServerFrame ui;

    public ServerThread(SocketServer _server, Socket _socket){
        super();
        server = _server;
        socket = _socket;
        ID     = socket.getPort();
        ui = _server.ui;
    }

    public void send(Message msg){
        try {
            streamOut.writeObject(obj:msg);
            streamOut.flush();
        }
        catch (IOException ex) {
            System.out.println(x: "Exception [SocketClient : send(...)]");
        }
    }

    public int getID(){
            return ID;
    }

    @SuppressWarnings("deprecation")
        public void run(){
        ui.jTextArea1.append("\nServer Thread " + ID + " running.");
        while (true){
            try{
                Message msg = (Message) streamIn.readObject();
                server.handle(ID, msg);
            }
            catch(Exception ioe){
                System.out.println(ID + " ERROR reading: " + ioe.getMessage());
                server.remove(ID);
```

The code defines a server in Java for a basic chat application using sockets. It handles multiple client connections, manages message broadcasting, and tracks connected clients. The server listens on a specified port and creates a separate thread for each connected client, allowing them to send and receive messages.

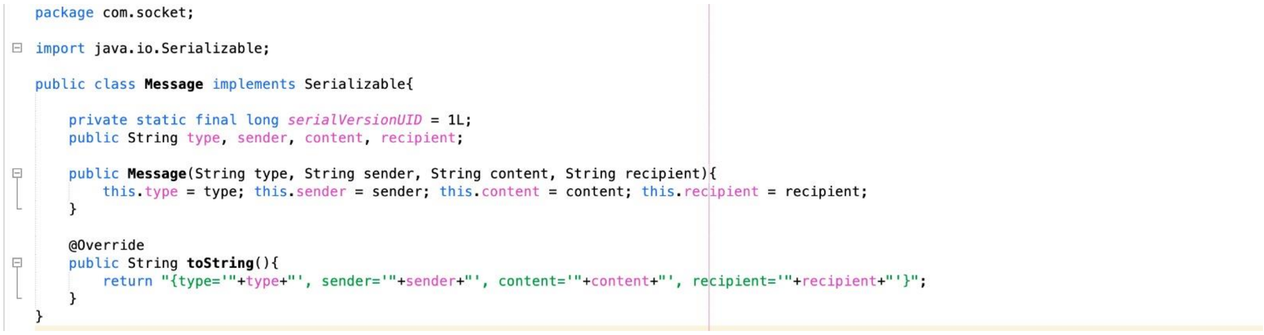
## STEP2: Java Socket-based Chat Client Implementation with File Transfer



```java
public class SocketClient implements Runnable{

    public int port;
    public String serverAddr;
    public Socket socket;
    public ChatFrame ui;
    public ObjectInputStream In;
    public ObjectOutputStream Out;
    public History hist;

    public SocketClient(ChatFrame frame) throws IOException{
        ui = frame; this.serverAddr = ui.serverAddr; this.port = ui.port;
        socket = new Socket(address:InetAddress.getByName(host: serverAddr), port);

        Out = new ObjectOutputStream(out:socket.getOutputStream());
        Out.flush();
        In = new ObjectInputStream(in: socket.getInputStream());

        hist = ui.hist;
    }

    @Override
    public void run() {
        boolean keepRunning = true;
        while(keepRunning){
            try {
                Message msg = (Message) In.readObject();
                System.out.println("Incoming : "+msg.toString());

                if(msg.type.equals(anObject:"message")){
                    if(msg.recipient.equals(anObject: ui.username)){
                        ui.jTextArea1.append("["+msg.sender +" > Me] : " + msg.content + "\n");
                    }
                    else{
                        ui.jTextArea1.append("["+ msg.sender +" > "+ msg.recipient +"] : " + msg.content + "\n");
                    }

                    if(!msg.content.equals(anObject:".bye") && !msg.sender.equals(anObject: ui.username)){
                        String msgTime = (new Date()).toString();

                        try{
                            hist.addMessage(msg, time: msgTime);
                            DefaultTableModel table = (DefaultTableModel) ui.historyFrame.jTable1.getModel();
                            table.addRow(new Object[]{msg.sender, msg.content, "Me", msgTime});
                        }
                        catch(Exception ex){}
                    }
                }
                else if(msg.type.equals(anObject:"login")){
```

This Java code defines a client-side component for a chat application using sockets. It handles message communication, file uploads, and user interactions, connecting to a server and running in a separate thread. The code enables chat features, file transfers, and user login/signup, integrated with GUI.

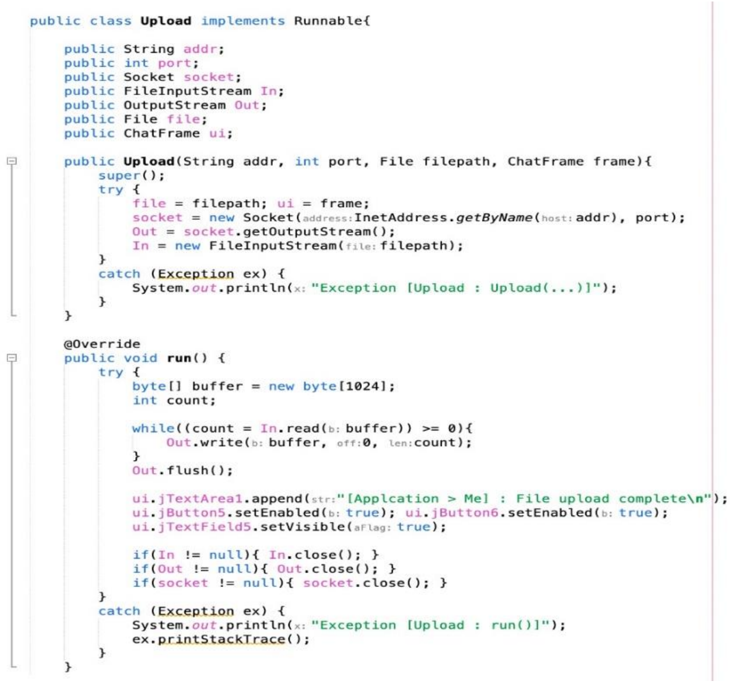
## STEP3: Create a Serializable Message Class



```
package com.socket;

import java.io.Serializable;

public class Message implements Serializable{

    private static final long serialVersionUID = 1L;
    public String type, sender, content, recipient;

    public Message(String type, String sender, String content, String recipient){
        this.type = type; this.sender = sender; this.content = content; this.recipient = recipient;
    }

    @Override
    public String toString(){
        return "{type='"+type+"', sender='"+sender+"', content='"+content+"', recipient='"+recipient+"'}";
    }
}
```

This Java code defines a Message class that implements the Serializable interface for object serialization. It represents messages exchanged in a chat application and includes attributes like message type, sender, content, and recipient. The class allows message objects to be transmitted easily between the client and server over a network.

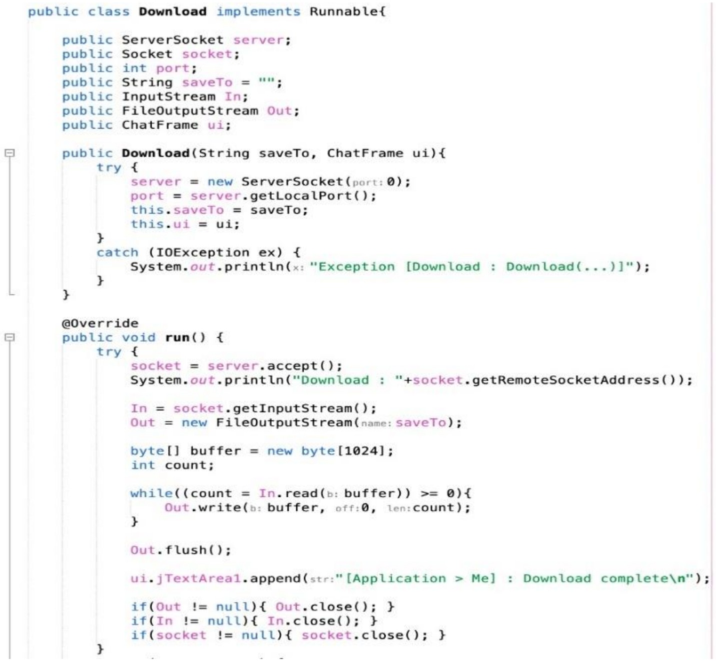
## STEP4: Creating a File Upload Class for Chat Application Client



```
public class Upload implements Runnable{

    public String addr;
    public int port;
    public Socket socket;
    public FileInputStream In;
    public OutputStream Out;
    public File file;
    public ChatFrame ui;

    public Upload(String addr, int port, File filepath, ChatFrame frame){
        super();
        try {
            file = filepath; ui = frame;
            socket = new Socket(address:InetAddress.getByName(host:addr), port);
            Out = socket.getOutputStream();
            In = new FileInputStream(file:filepath);
        }
        catch (Exception ex) {
            System.out.println(x:"Exception [Upload : Upload(...)]");
        }
    }

    @Override
    public void run() {
        try {
            byte[] buffer = new byte[1024];
            int count;

            while((count = In.read(b:buffer)) >= 0){
                Out.write(b:buffer, off:0, len:count);
            }
            Out.flush();

            ui.jTextArea1.append(str:"[Applcation > Me] : File upload complete\n");
            ui.jButton5.setEnabled(b:true); ui.jButton6.setEnabled(b:true);
            ui.jTextField5.setVisible(aFlag:true);

            if(In != null){ In.close(); }
            if(Out != null){ Out.close(); }
            if(socket != null){ socket.close(); }
        }
        catch (Exception ex) {
            System.out.println(x:"Exception [Upload : run()]");
            ex.printStackTrace();
        }
    }
```

This Java code defines an Upload class for uploading files from a client to a server in a chat application. It establishes a network connection, reads a file (filepath), and sends it to the server using an output stream. The class handles file uploads and enables the user to interact with the chat interface.

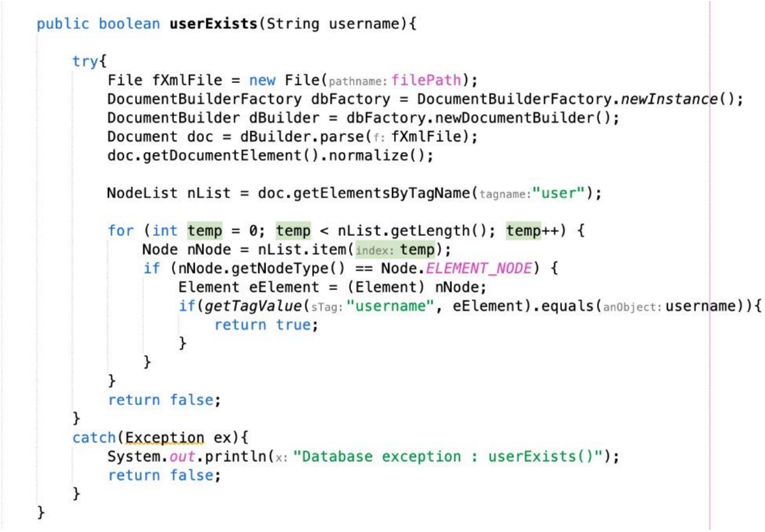
## STEP5: Creating a Server-Side File Download Handler for Chat Application



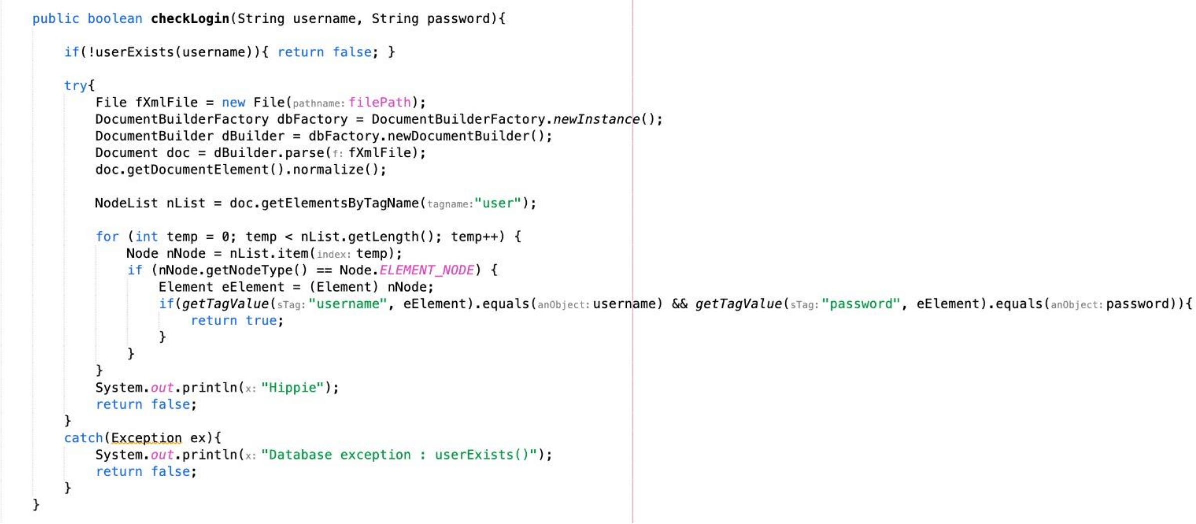
```java
public class Download implements Runnable{

    public ServerSocket server;
    public Socket socket;
    public int port;
    public String saveTo = "";
    public InputStream In;
    public FileOutputStream Out;
    public ChatFrame ui;

    public Download(String saveTo, ChatFrame ui){
        try {
            server = new ServerSocket(port: 0);
            port = server.getLocalPort();
            this.saveTo = saveTo;
            this.ui = ui;
        }
        catch (IOException ex) {
            System.out.println(x: "Exception [Download : Download(...)]");
        }
    }

    @Override
    public void run() {
        try {
            socket = server.accept();
            System.out.println("Download : "+socket.getRemoteSocketAddress());

            In = socket.getInputStream();
            Out = new FileOutputStream(name: saveTo);

            byte[] buffer = new byte[1024];
            int count;

            while((count = In.read(b: buffer)) >= 0){
                Out.write(b: buffer, off:0, len:count);
            }

            Out.flush();

            ui.jTextArea1.append(str:"[Application > Me] : Download complete\n");

            if(Out != null){ Out.close(); }
            if(In != null){ In.close(); }
            if(socket != null){ socket.close(); }
        }
```

This Java code defines a Download class for the server-side handling of file downloads in a chat application. It sets up a server socket to listen on a dynamically allocated port and receives files from clients. The class reads incoming data, writes it to a file, and notifies the user interface upon download completion.

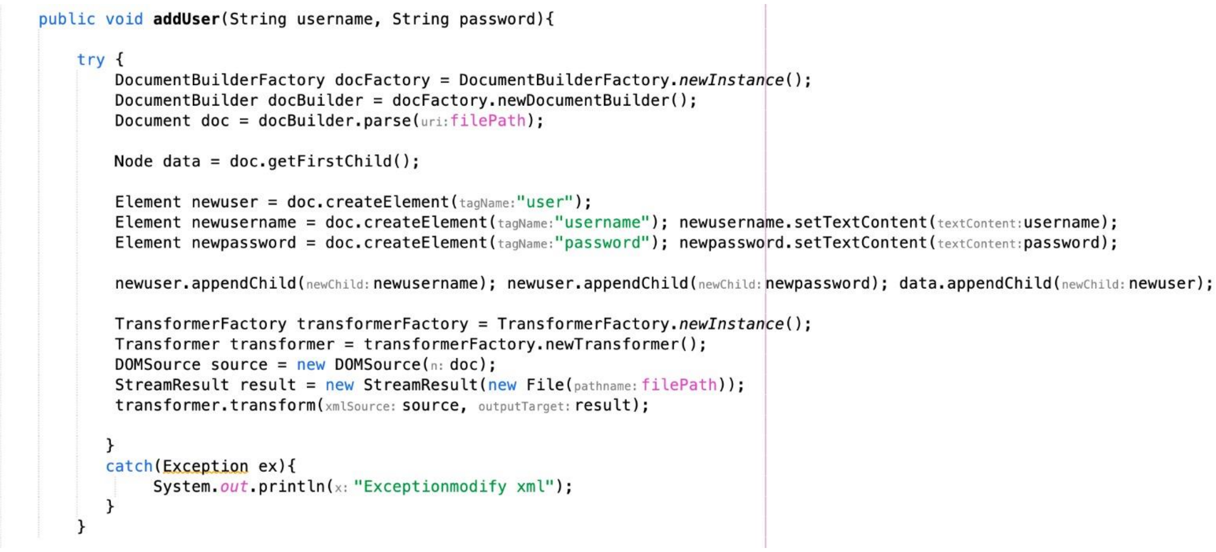
## STEP6: Creating a User Database Manager with XML



```java
public boolean userExists(String username){

    try{
        File fXmlFile = new File(pathname: filePath);
        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
        Document doc = dBuilder.parse(f: fXmlFile);
        doc.getDocumentElement().normalize();

        NodeList nList = doc.getElementsByTagName(tagname:"user");

        for (int temp = 0; temp < nList.getLength(); temp++) {
            Node nNode = nList.item(index: temp);
            if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                Element eElement = (Element) nNode;
                if(getTagValue(sTag: "username", eElement).equals(anObject: username)){
                    return true;
                }
            }
        }
        return false;
    }
    catch(Exception ex){
        System.out.println(x: "Database exception : userExists()");
        return false;
    }
}
```



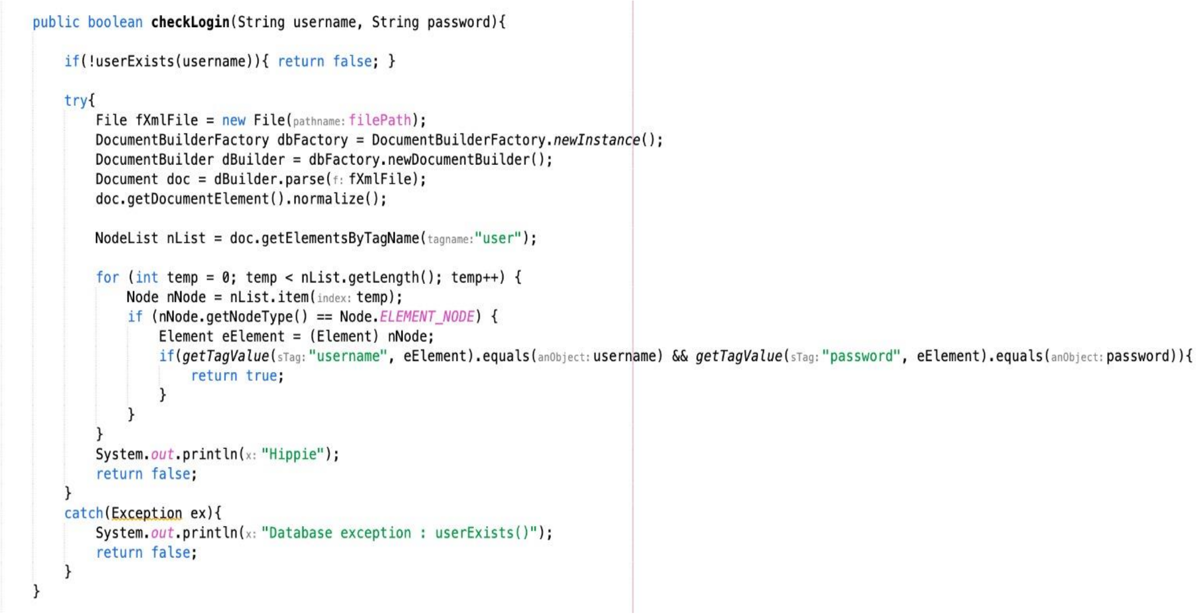
```
public boolean checkLogin(String username, String password){

    if(!userExists(username)){ return false; }

    try{
        File fXmlFile = new File(pathname: filePath);
        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
        Document doc = dBuilder.parse(f: fXmlFile);
        doc.getDocumentElement().normalize();

        NodeList nList = doc.getElementsByTagName(tagname:"user");

        for (int temp = 0; temp < nList.getLength(); temp++) {
            Node nNode = nList.item(index: temp);
            if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                Element eElement = (Element) nNode;
                if(getTagValue(sTag: "username", eElement).equals(anObject: username) && getTagValue(sTag: "password", eElement).equals(anObject: password)){
                    return true;
                }
            }
        }
        System.out.println(x: "Hippie");
        return false;
    }
    catch(Exception ex){
        System.out.println(x: "Database exception : userExists()");
        return false;
    }
}
```



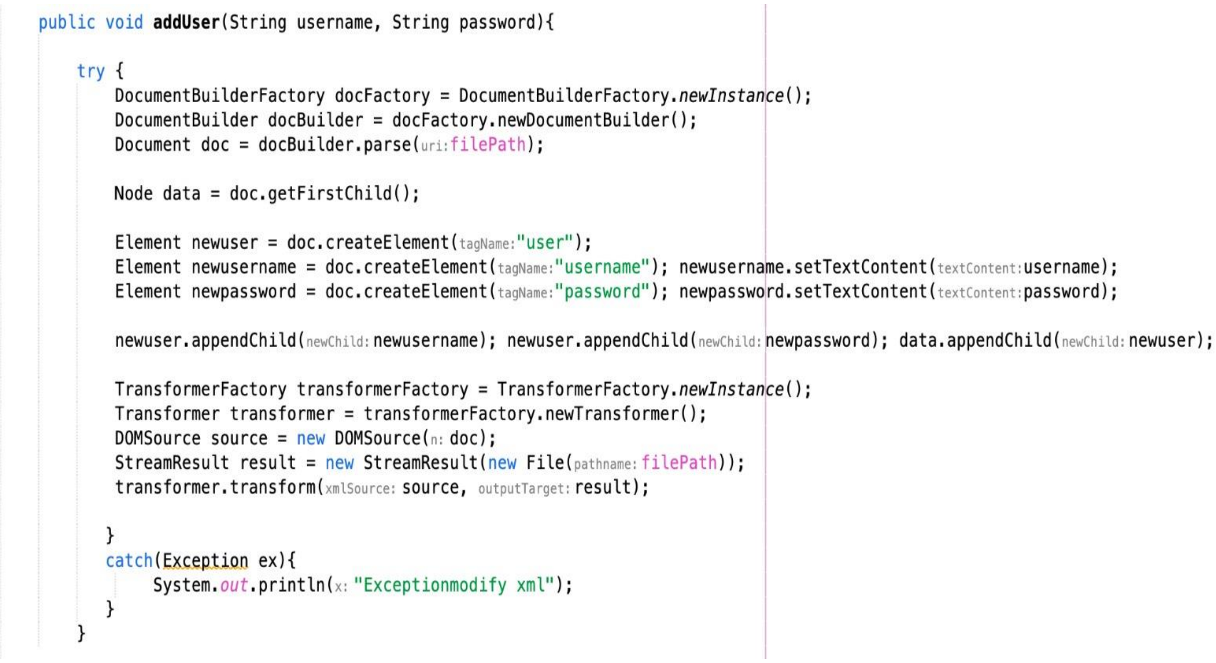
```
public void addUser(String username, String password){

    try {
        DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
        Document doc = docBuilder.parse(uri:filePath);

        Node data = doc.getFirstChild();

        Element newuser = doc.createElement(tagName:"user");
        Element newusername = doc.createElement(tagName:"username"); newusername.setTextContent(textContent:username);
        Element newpassword = doc.createElement(tagName:"password"); newpassword.setTextContent(textContent:password);

        newuser.appendChild(newChild: newusername); newuser.appendChild(newChild: newpassword); data.appendChild(newChild: newuser);

        TransformerFactory transformerFactory = TransformerFactory.newInstance();
        Transformer transformer = transformerFactory.newTransformer();
        DOMSource source = new DOMSource(n: doc);
        StreamResult result = new StreamResult(new File(pathname: filePath));
        transformer.transform(xmlSource: source, outputTarget: result);

       }
       catch(Exception ex){
            System.out.println(x: "Exceptionmodify xml");
       }
    }
```

This Java code defines a Database class for managing user information in XML files for a chat application. It includes methods to check if a user exists, validate login credentials, and add a new user to the XML database. The class handles XML parsing and manipulation to manage user data.

# STEP7: Creating a Message History Manager with XML



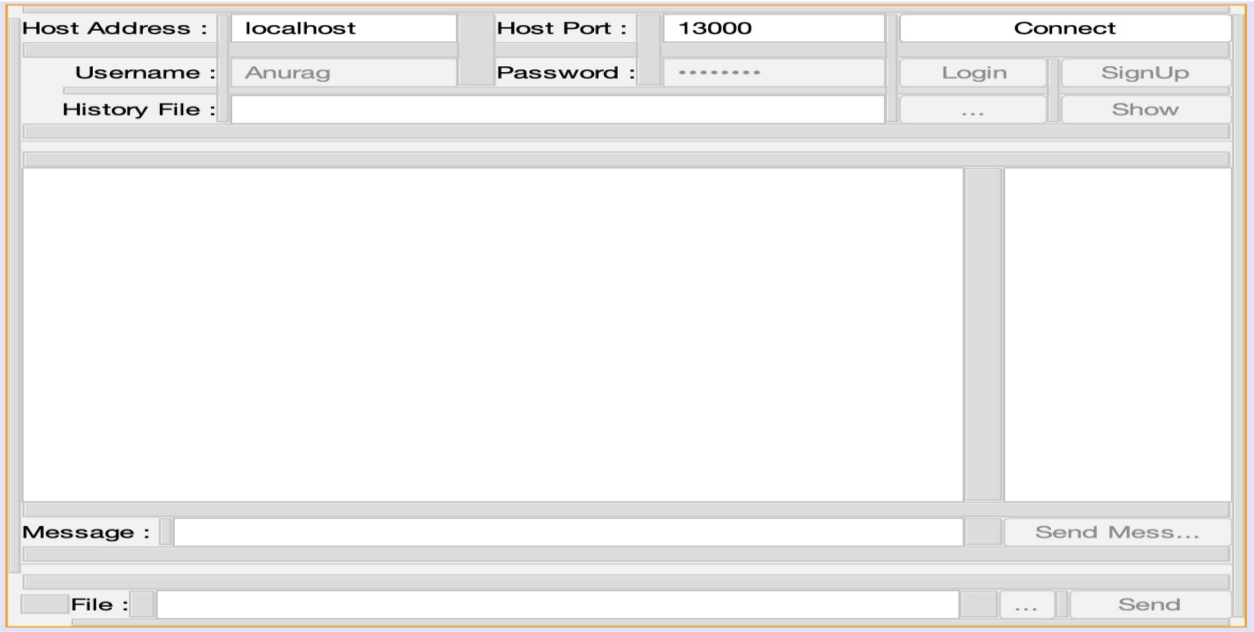
```
public boolean checkLogin(String username, String password){

    if(!userExists(username)){ return false; }

    try{
        File fXmlFile = new File(pathname: filePath);
        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
        Document doc = dBuilder.parse(f: fXmlFile);
        doc.getDocumentElement().normalize();

        NodeList nList = doc.getElementsByTagName(tagname:"user");

        for (int temp = 0; temp < nList.getLength(); temp++) {
            Node nNode = nList.item(index: temp);
            if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                Element eElement = (Element) nNode;
                if(getTagValue(sTag: "username", eElement).equals(anObject: username) && getTagValue(sTag: "password", eElement).equals(anObject: password)){
                    return true;
                }
            }
        }
        System.out.println(x: "Hippie");
        return false;
    }
    catch(Exception ex){
        System.out.println(x: "Database exception : userExists()");
        return false;
    }
}
```



```
public void addUser(String username, String password){

    try {
        DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
        Document doc = docBuilder.parse(uri:filePath);

        Node data = doc.getFirstChild();

        Element newuser = doc.createElement(tagName:"user");
        Element newusername = doc.createElement(tagName:"username"); newusername.setTextContent(textContent:username);
        Element newpassword = doc.createElement(tagName:"password"); newpassword.setTextContent(textContent:password);

        newuser.appendChild(newChild: newusername); newuser.appendChild(newChild: newpassword); data.appendChild(newChild: newuser);

        TransformerFactory transformerFactory = TransformerFactory.newInstance();
        Transformer transformer = transformerFactory.newTransformer();
        DOMSource source = new DOMSource(n: doc);
        StreamResult result = new StreamResult(new File(pathname: filePath));
        transformer.transform(xmlSource: source, outputTarget: result);

      }
      catch(Exception ex){
           System.out.println(x: "Exceptionmodify xml");
      }
  }
```

This Java code defines a history class for managing and storing message history in an XML file for a chat application. It provides methods to add chat messages to the history and populate a table with message data for display. The class handles XML parsing and updating the chat history.

## STEP8: Creating GUI using Java Swing



Host Address : localhost

Host Port : 13000

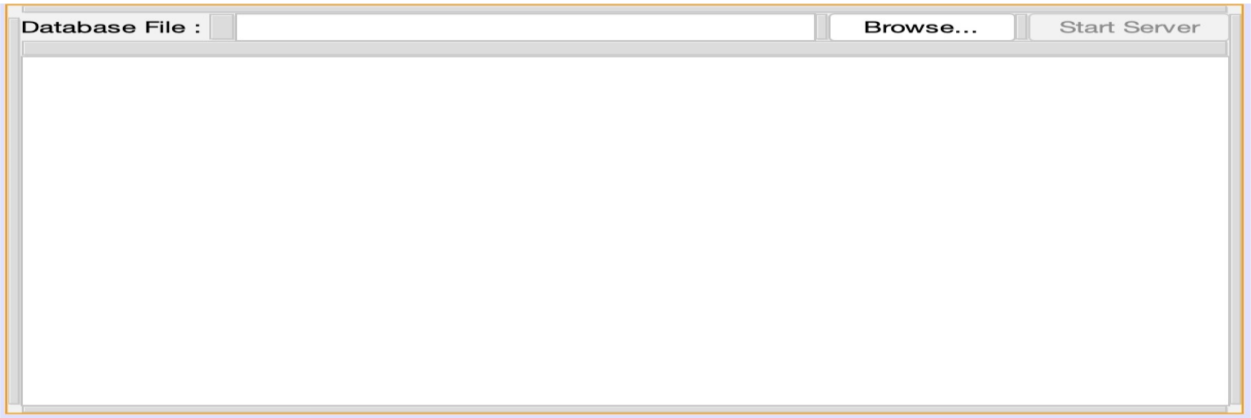
Username : Anurag

Password : ••••••••

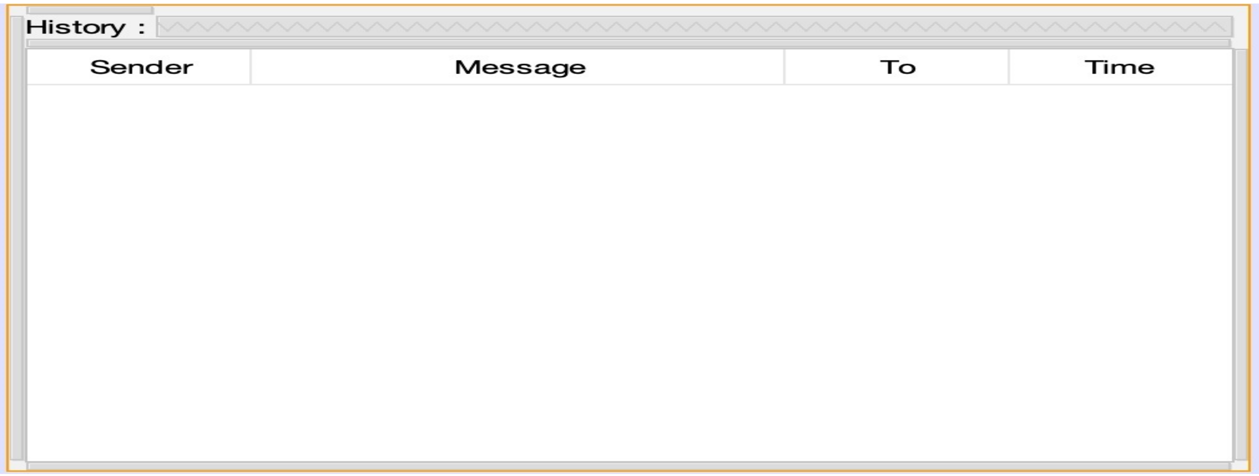
History File :

Message :

File :



Database File :

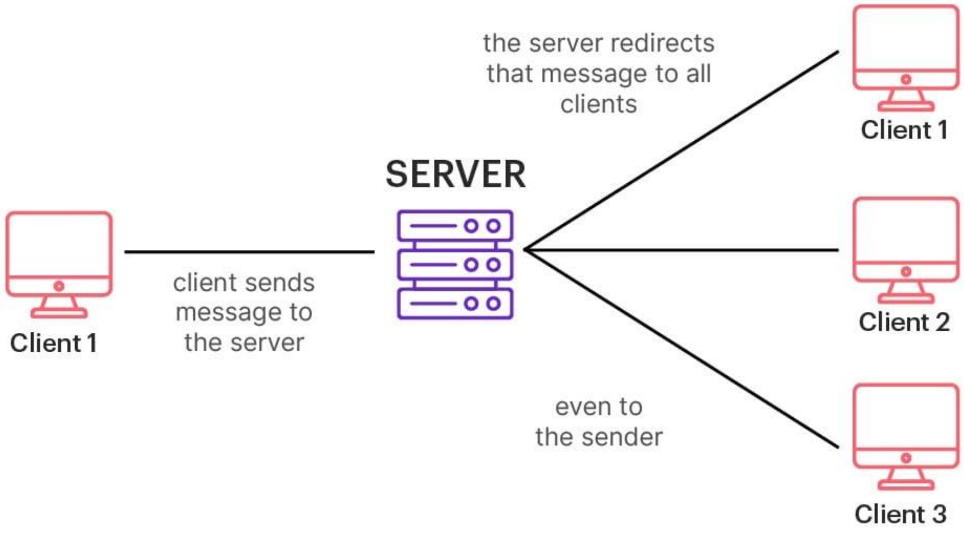


History :

| Sender | Message | To | Time |
|---|---|---|---|

Created a Server frame, Chat frame and History frame using Java swing

# ARCHITECTURE & DESIGNS


the server redirects that message to all clients
SERVER
Client 1
client sends message to the server
Client 1
Client 2
even to the sender
Client 3

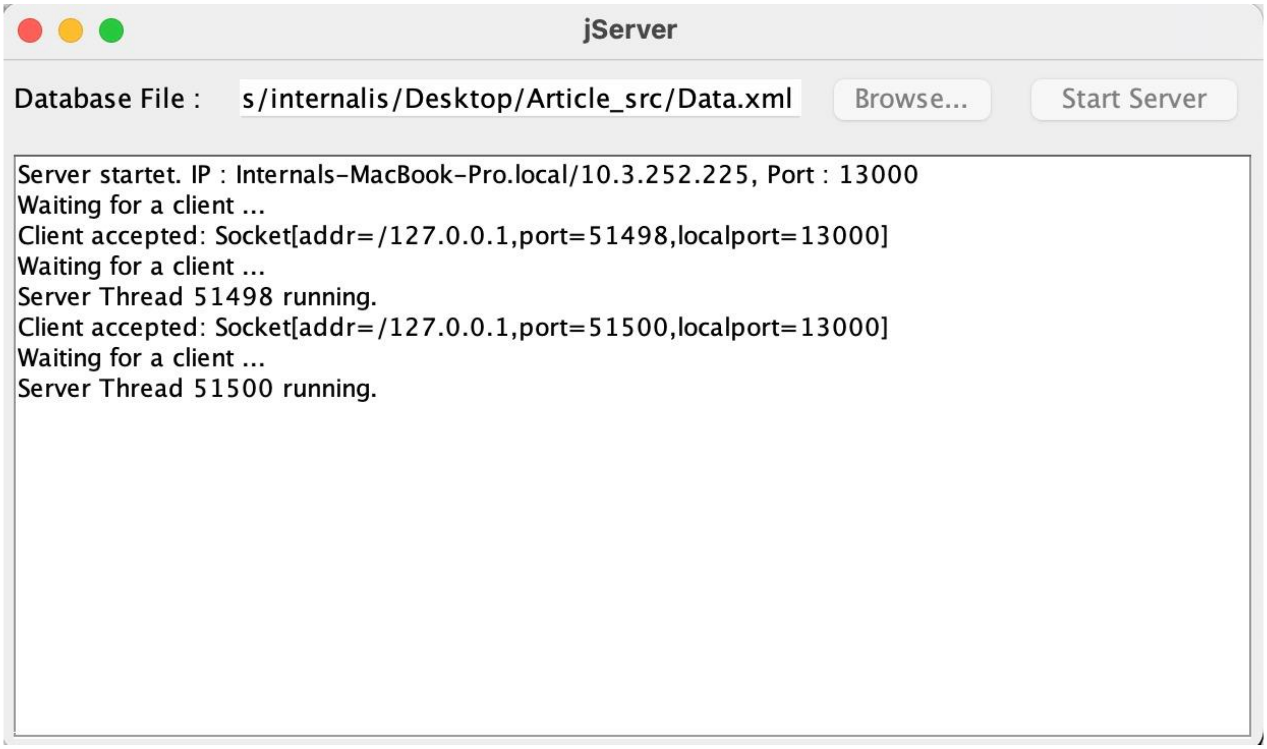
## Chat Client :

The chat client is what the user experiences. A desktop, web or smartphone chat application, the chat client is responsible for interacting with the operating system (i.e., your computer, browser, or smartphone). Interactions include sending push notifications, displaying data to the user, and storing messages and files. When you type a message and hit send, the chat client transmits that message to the other major component: the chat server.

## Chat Server:

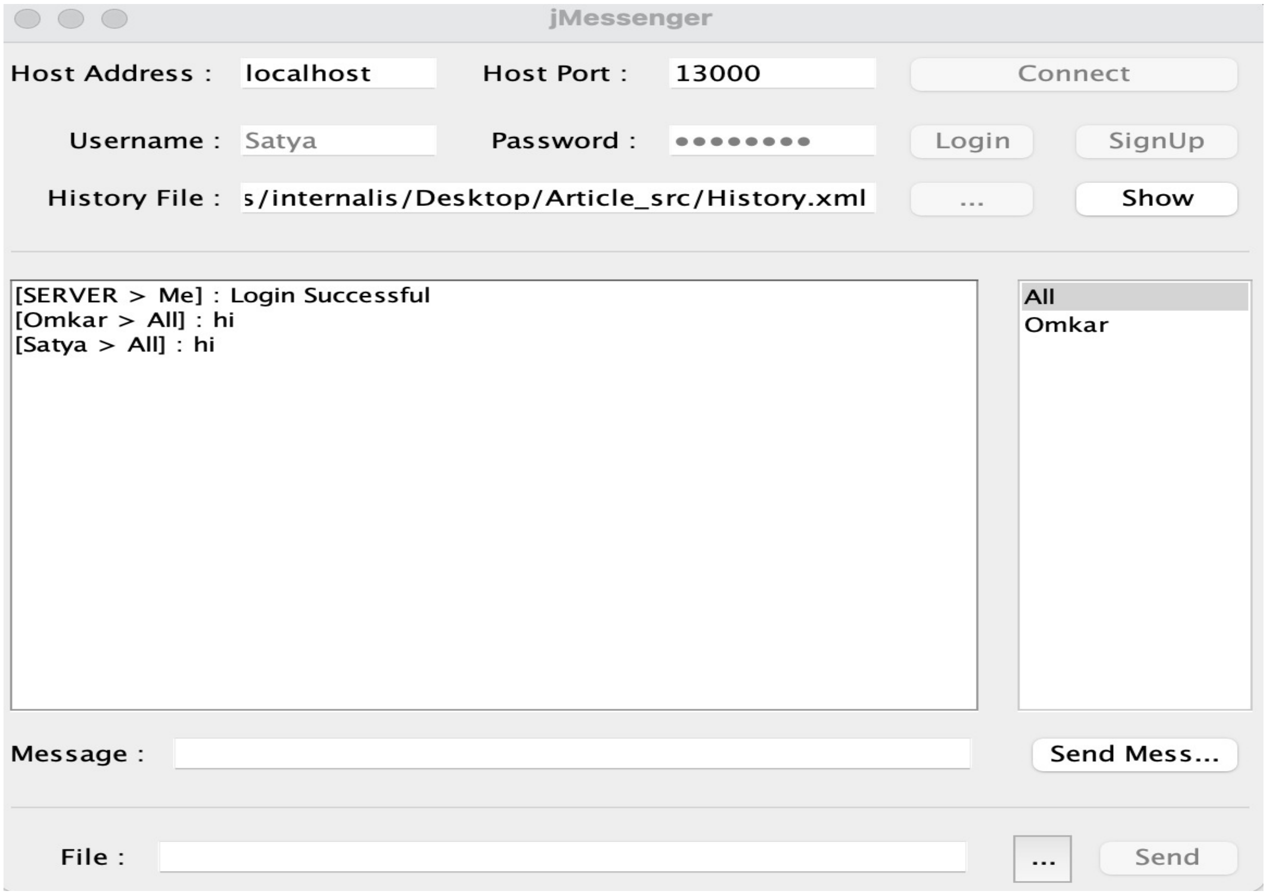
The chat server is just that, a server (or usually many servers) that hosts all the software, frameworks and databases necessary for the chat app to operate. This server, or pool of servers, is responsible for securely receiving a message, identifying the correct recipient, queuing for the message and then forwarding the message to the recipient's chat client. The chat server’s resources can include a REST API, a WebSocket server, an AWS instance for media storage, etc.

# JAVA RESULTS



Database File : s/internalis/Desktop/Article_src/Data.xml

```
Server startet. IP : Internals-MacBook-Pro.local/10.3.252.225, Port : 13000
Waiting for a client ...
Client accepted: Socket[addr=/127.0.0.1,port=51498,localport=13000]
Waiting for a client ...
Server Thread 51498 running.
Client accepted: Socket[addr=/127.0.0.1,port=51500,localport=13000]
Waiting for a client ...
Server Thread 51500 running.
```



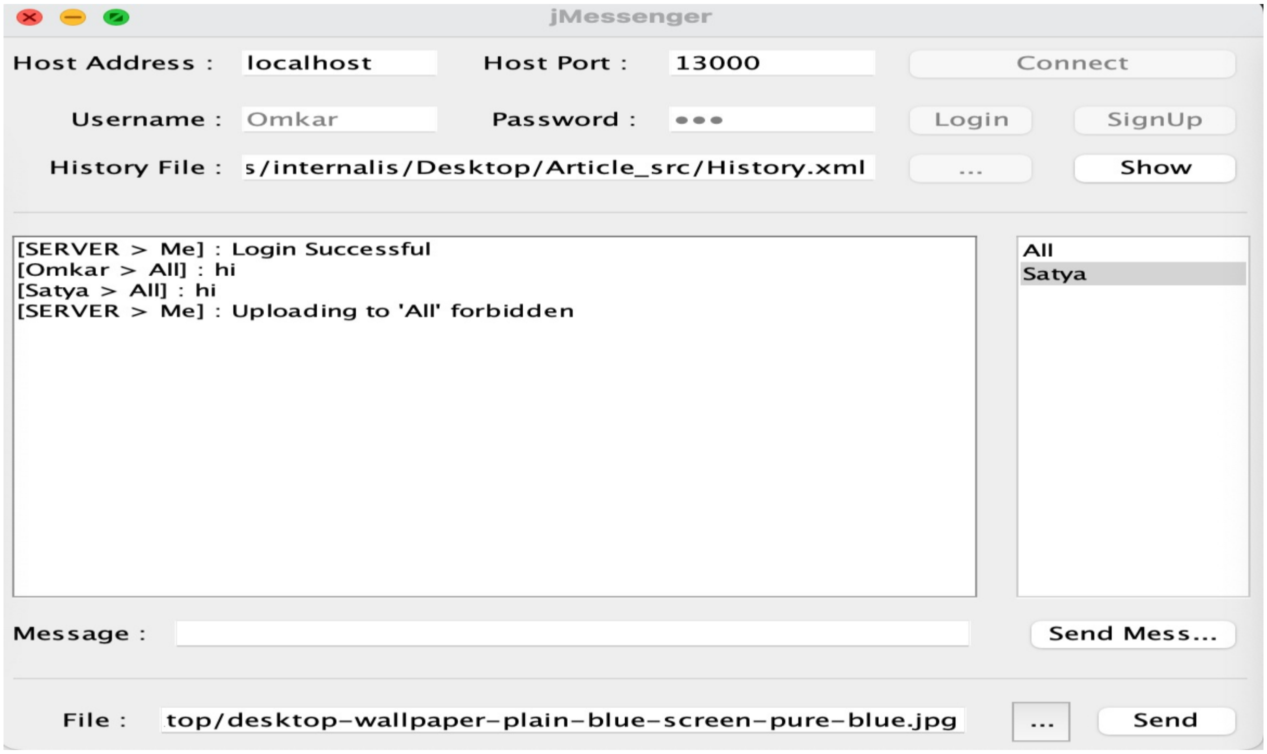
Host Address : localhost    Host Port : 13000

Username : Satya    Password : ●●●●●●●●●

History File : s/internalis/Desktop/Article_src/History.xml

```
[SERVER > Me] : Login Successful
[Omkar > All] : hi
[Satya > All] : hi
```

All
Omkar



Host Address : localhost

Host Port : 13000

Username : Omkar

Password : •••

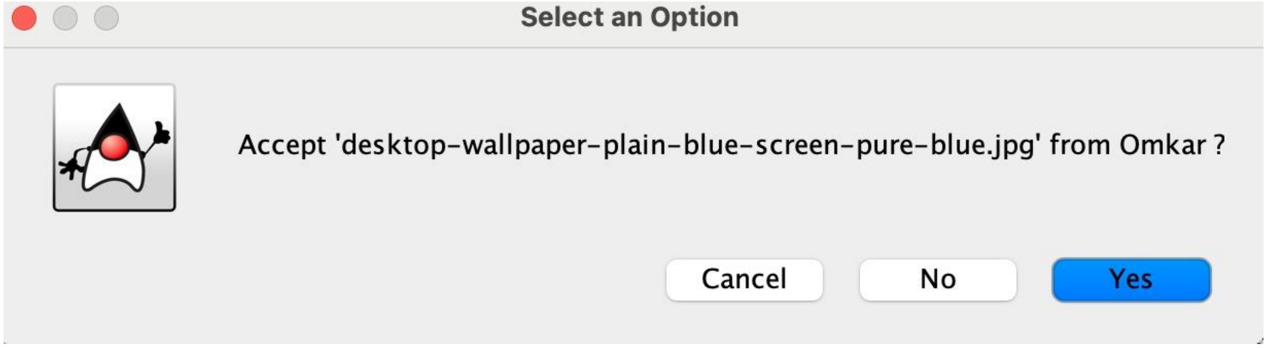
History File : s/internalis/Desktop/Article_src/History.xml

```
[SERVER > Me] : Login Successful
[Omkar > All] : hi
[Satya > All] : hi
[SERVER > Me] : Uploading to 'All' forbidden
```

All
Satya

Message :

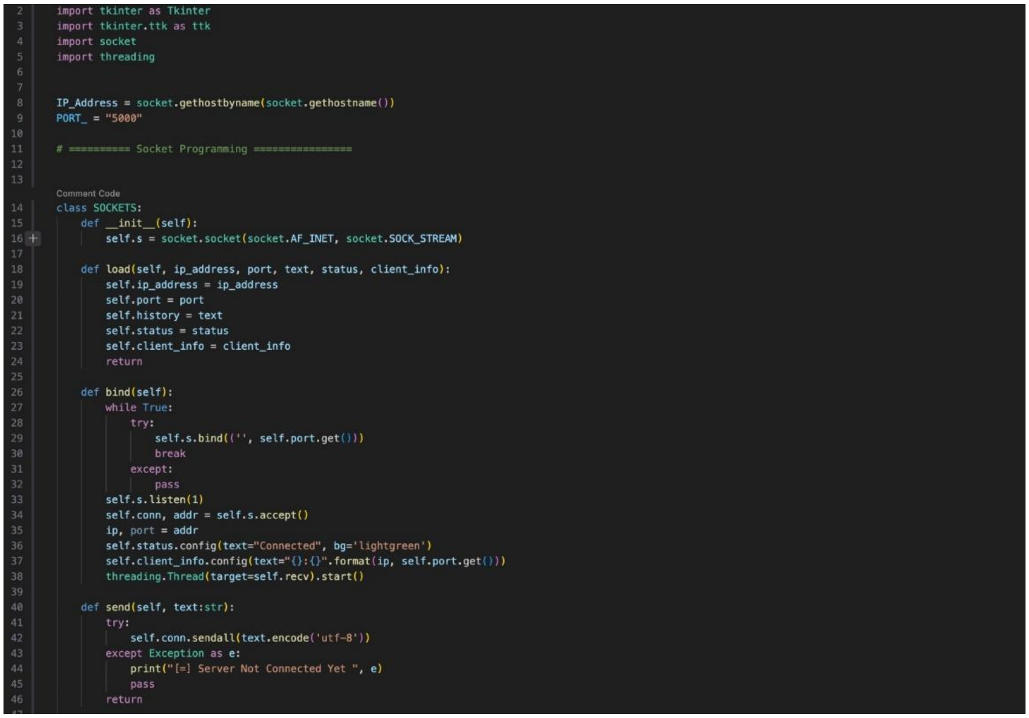
File : top/desktop-wallpaper-plain-blue-screen-pure-blue.jpg



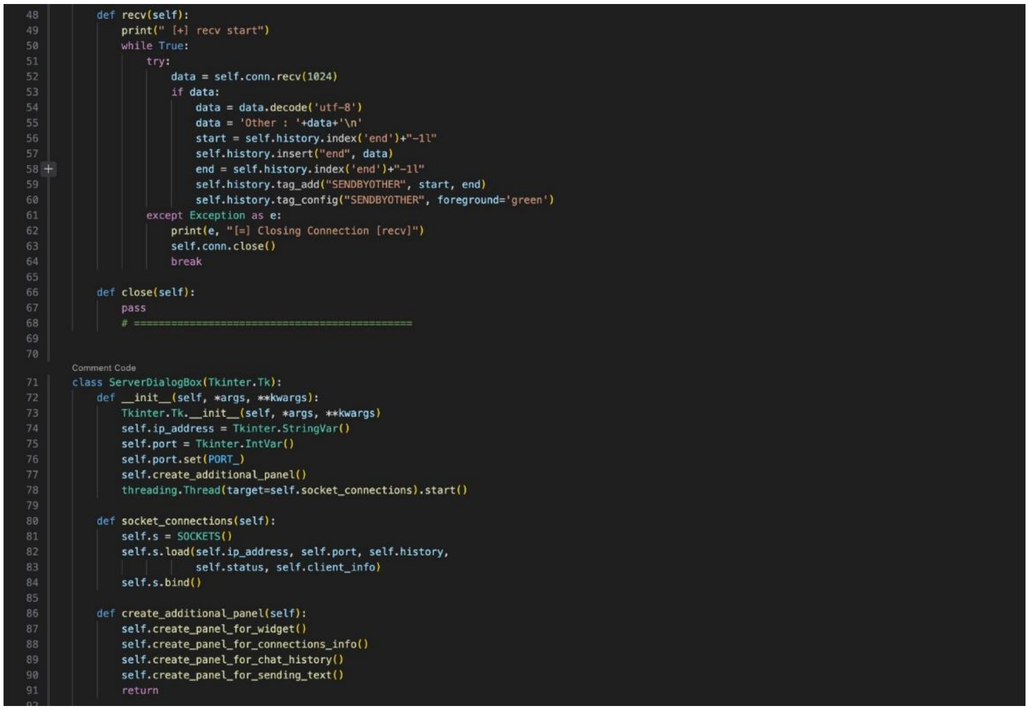
Accept 'desktop-wallpaper-plain-blue-screen-pure-blue.jpg' from Omkar ?

# PYTHON IMPLEMENTATION

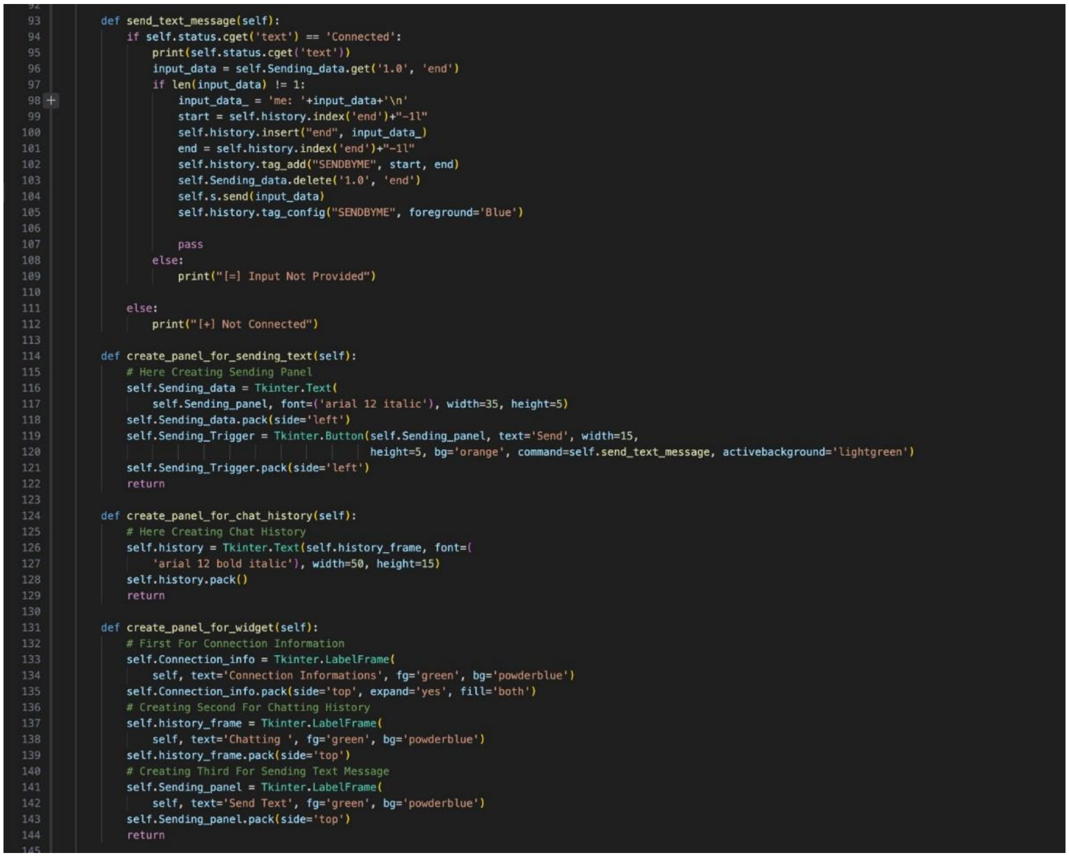
## Server Module



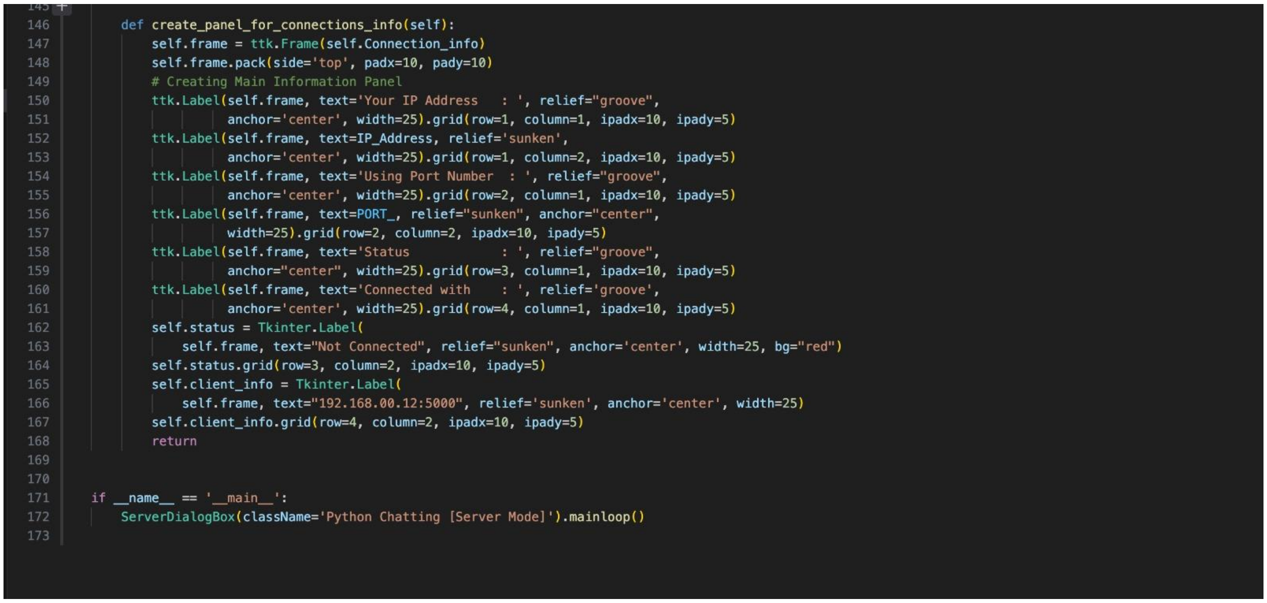
```
import tkinter as Tkinter
import tkinter.ttk as ttk
import socket
import threading


IP_Address = socket.gethostbyname(socket.gethostname())
PORT_ = "5000"

# ========== Socket Programming ================


Comment Code
class SOCKETS:
    def __init__(self):
        self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    def load(self, ip_address, port, text, status, client_info):
        self.ip_address = ip_address
        self.port = port
        self.history = text
        self.status = status
        self.client_info = client_info
        return

    def bind(self):
        while True:
            try:
                self.s.bind(('', self.port.get()))
                break
            except:
                pass
        self.s.listen(1)
        self.conn, addr = self.s.accept()
        ip, port = addr
        self.status.config(text="Connected", bg='lightgreen')
        self.client_info.config(text="{}:{}".format(ip, self.port.get()))
        threading.Thread(target=self.recv).start()

    def send(self, text:str):
        try:
            self.conn.sendall(text.encode('utf-8'))
        except Exception as e:
            print("[=] Server Not Connected Yet ", e)
            pass
        return
```



```
    def recv(self):
        print(" [+] recv start")
        while True:
            try:
                data = self.conn.recv(1024)
                if data:
                    data = data.decode('utf-8')
                    data = 'Other : '+data+'\n'
                    start = self.history.index('end')+"-1l"
                    self.history.insert("end", data)
                    end = self.history.index('end')+"-1l"
                    self.history.tag_add("SENDBYOTHER", start, end)
                    self.history.tag_config("SENDBYOTHER", foreground='green')
            except Exception as e:
                print(e, "[=] Closing Connection [recv]")
                self.conn.close()
                break

    def close(self):
        pass
        # ============================================


Comment Code
class ServerDialogBox(Tkinter.Tk):
    def __init__(self, *args, **kwargs):
        Tkinter.Tk.__init__(self, *args, **kwargs)
        self.ip_address = Tkinter.StringVar()
        self.port = Tkinter.IntVar()
        self.port.set(PORT_)
        self.create_additional_panel()
        threading.Thread(target=self.socket_connections).start()

    def socket_connections(self):
        self.s = SOCKETS()
        self.s.load(self.ip_address, self.port, self.history,
                    self.status, self.client_info)
        self.s.bind()

    def create_additional_panel(self):
        self.create_panel_for_widget()
        self.create_panel_for_connections_info()
        self.create_panel_for_chat_history()
        self.create_panel_for_sending_text()
        return
```

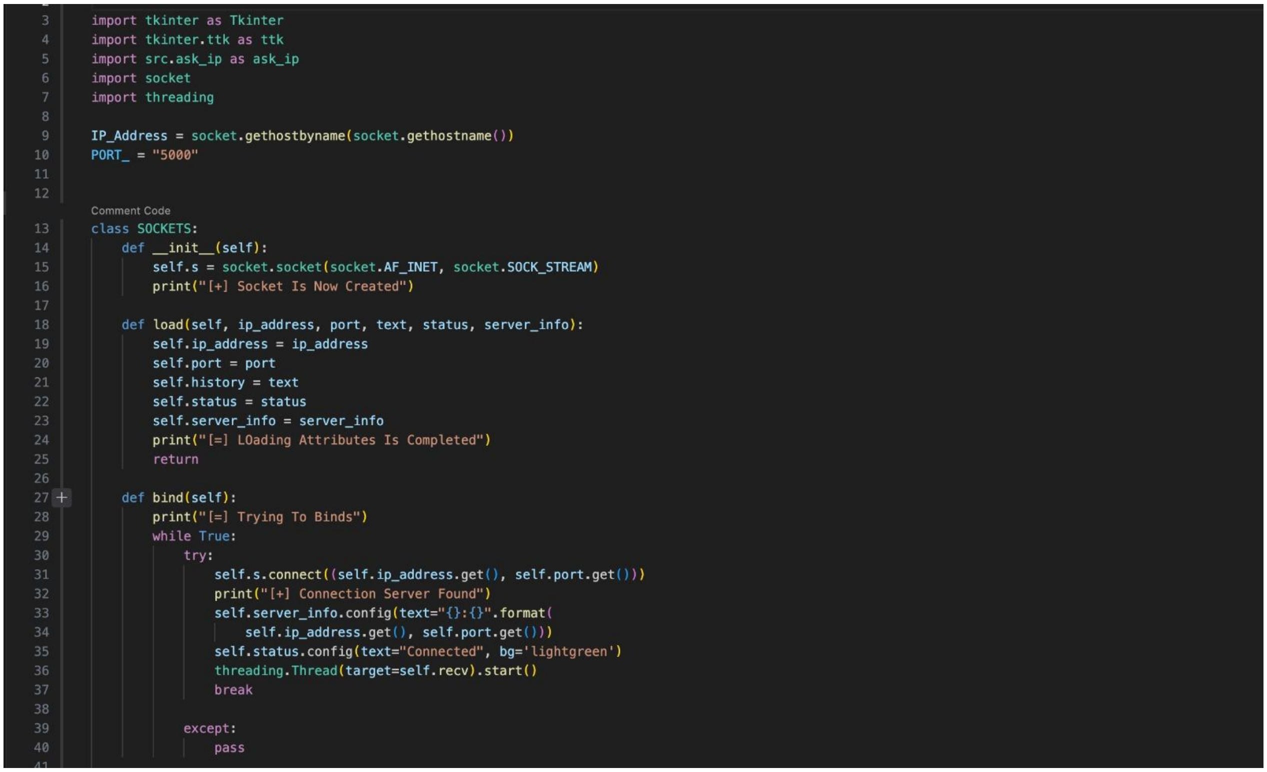


```
    def send_text_message(self):
        if self.status.cget('text') == 'Connected':
            print(self.status.cget('text'))
            input_data = self.Sending_data.get('1.0', 'end')
            if len(input_data) != 1:
                input_data_ = 'me: '+input_data+'\n'
                start = self.history.index('end')+"-1l"
                self.history.insert("end", input_data_)
                end = self.history.index('end')+"-1l"
                self.history.tag_add("SENDBYME", start, end)
                self.Sending_data.delete('1.0', 'end')
                self.s.send(input_data)
                self.history.tag_config("SENDBYME", foreground='Blue')

                pass
            else:
                print("[=] Input Not Provided")

        else:
            print("[+] Not Connected")

    def create_panel_for_sending_text(self):
        # Here Creating Sending Panel
        self.Sending_data = Tkinter.Text(
            self.Sending_panel, font=('arial 12 italic'), width=35, height=5)
        self.Sending_data.pack(side='left')
        self.Sending_Trigger = Tkinter.Button(self.Sending_panel, text='Send', width=15,
                                              height=5, bg='orange', command=self.send_text_message, activebackground='lightgreen')
        self.Sending_Trigger.pack(side='left')
        return

    def create_panel_for_chat_history(self):
        # Here Creating Chat History
        self.history = Tkinter.Text(self.history_frame, font=(
            'arial 12 bold italic'), width=50, height=15)
        self.history.pack()
        return

    def create_panel_for_widget(self):
        # First For Connection Information
        self.Connection_info = Tkinter.LabelFrame(
            self, text='Connection Informations', fg='green', bg='powderblue')
        self.Connection_info.pack(side='top', expand='yes', fill='both')
        # Creating Second For Chatting History
        self.history_frame = Tkinter.LabelFrame(
            self, text='Chatting ', fg='green', bg='powderblue')
        self.history_frame.pack(side='top')
        # Creating Third For Sending Text Message
        self.Sending_panel = Tkinter.LabelFrame(
            self, text='Send Text', fg='green', bg='powderblue')
        self.Sending_panel.pack(side='top')
        return
```

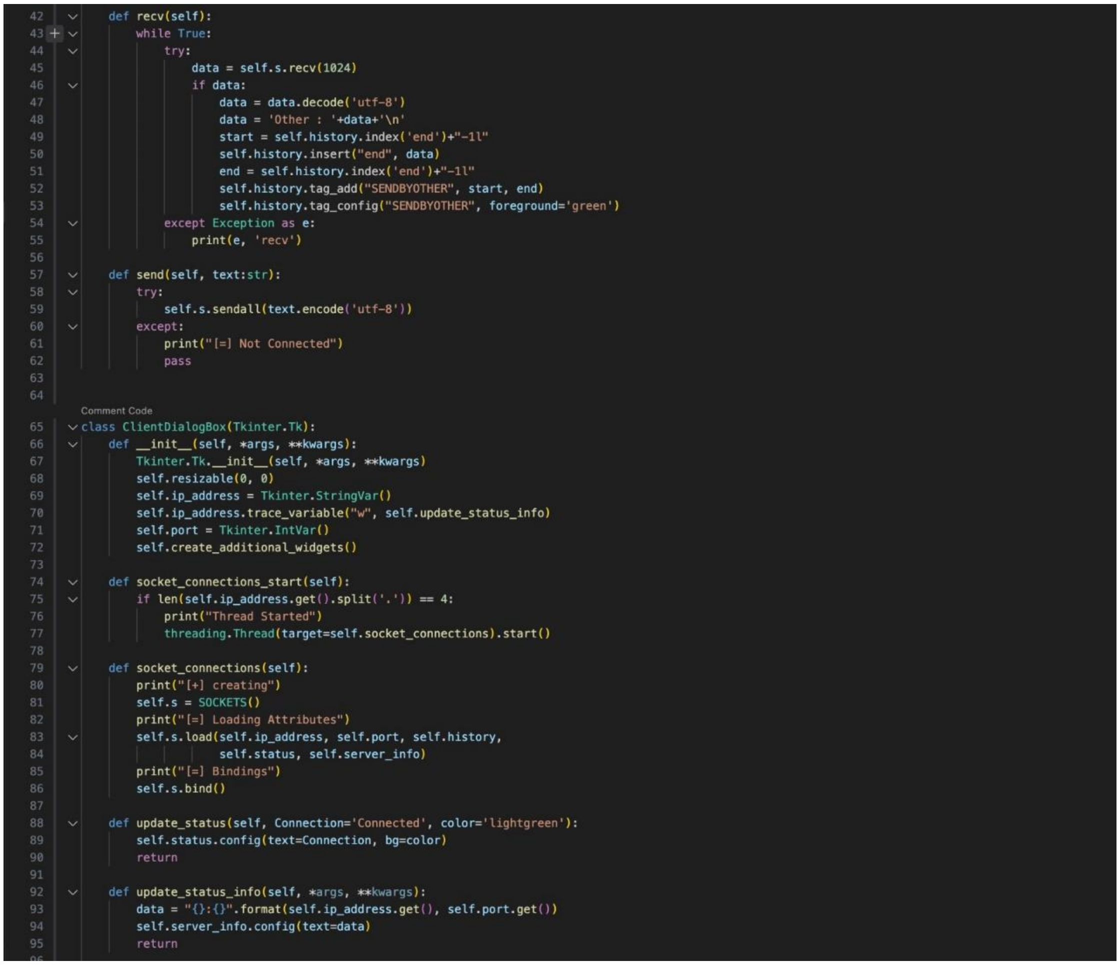


```
    def create_panel_for_connections_info(self):
        self.frame = ttk.Frame(self.Connection_info)
        self.frame.pack(side='top', padx=10, pady=10)
        # Creating Main Information Panel
        ttk.Label(self.frame, text='Your IP Address   : ', relief="groove",
                  anchor='center', width=25).grid(row=1, column=1, ipadx=10, ipady=5)
        ttk.Label(self.frame, text=IP_Address, relief='sunken',
                  anchor='center', width=25).grid(row=1, column=2, ipadx=10, ipady=5)
        ttk.Label(self.frame, text='Using Port Number  : ', relief="groove",
                  anchor='center', width=25).grid(row=2, column=1, ipadx=10, ipady=5)
        ttk.Label(self.frame, text=PORT_, relief="sunken", anchor="center",
                  width=25).grid(row=2, column=2, ipadx=10, ipady=5)
        ttk.Label(self.frame, text='Status            : ', relief="groove",
                  anchor="center", width=25).grid(row=3, column=1, ipadx=10, ipady=5)
        ttk.Label(self.frame, text='Connected with    : ', relief='groove',
                  anchor='center', width=25).grid(row=4, column=1, ipadx=10, ipady=5)
        self.status = Tkinter.Label(
            self.frame, text="Not Connected", relief="sunken", anchor='center', width=25, bg="red")
        self.status.grid(row=3, column=2, ipadx=10, ipady=5)
        self.client_info = Tkinter.Label(
            self.frame, text="192.168.00.12:5000", relief='sunken', anchor='center', width=25)
        self.client_info.grid(row=4, column=2, ipadx=10, ipady=5)
        return


if __name__ == '__main__':
    ServerDialogBox(className='Python Chatting [Server Mode]').mainloop()
```

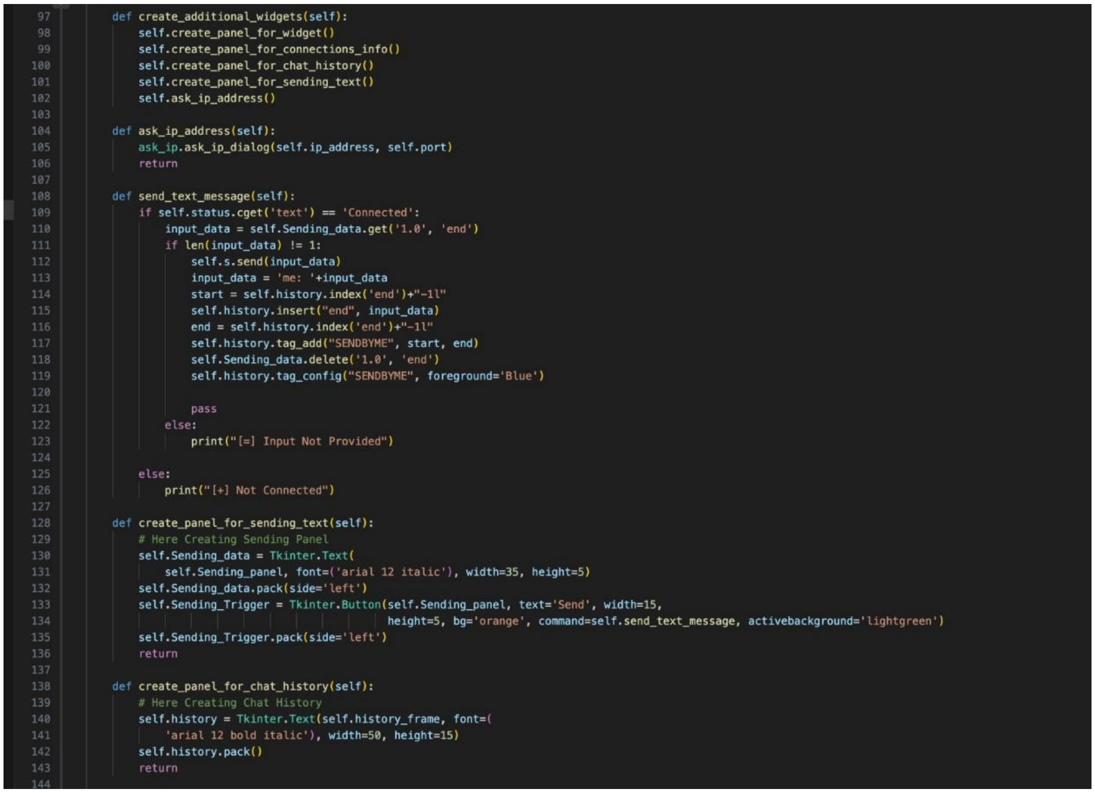
# Client Module



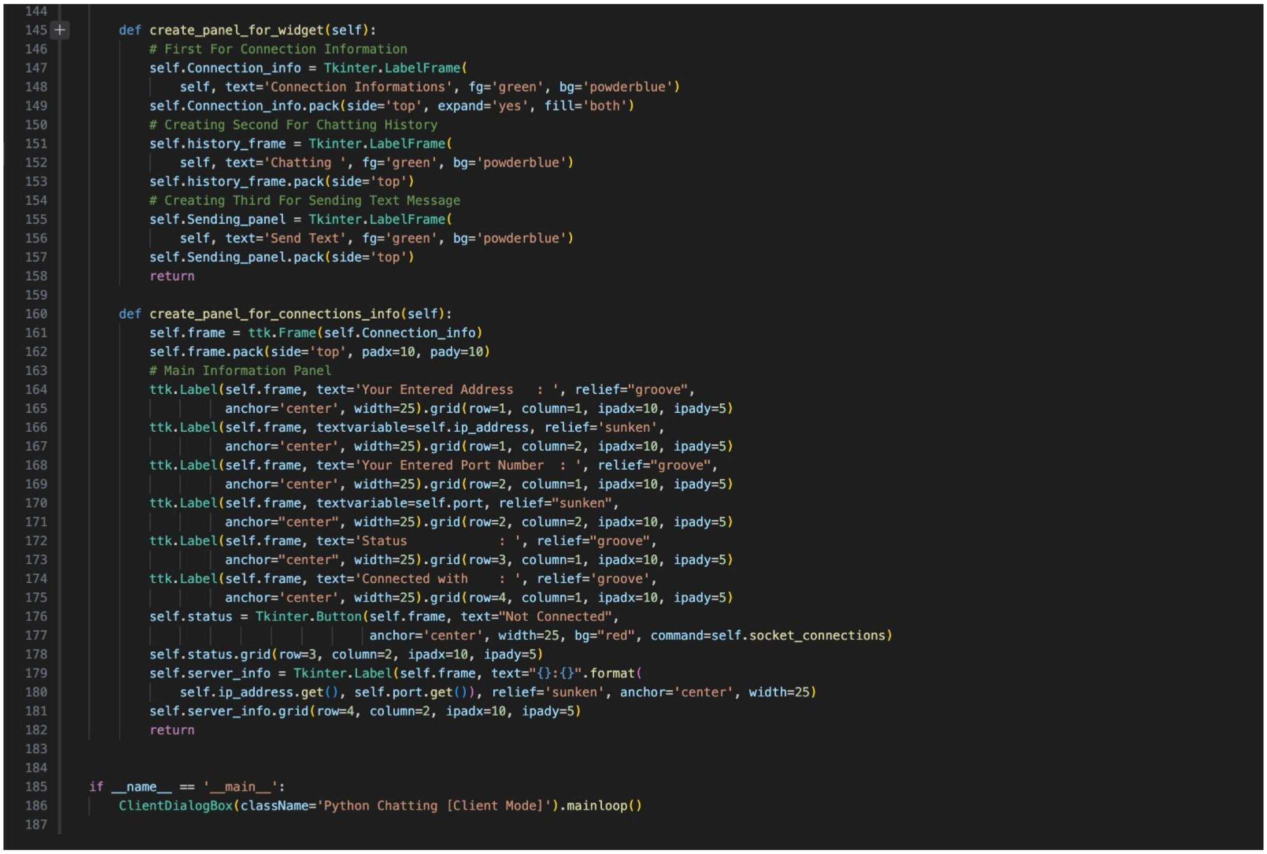
```python
import tkinter as Tkinter
import tkinter.ttk as ttk
import src.ask_ip as ask_ip
import socket
import threading

IP_Address = socket.gethostbyname(socket.gethostname())
PORT_ = "5000"


class SOCKETS:
    def __init__(self):
        self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        print("[+] Socket Is Now Created")

    def load(self, ip_address, port, text, status, server_info):
        self.ip_address = ip_address
        self.port = port
        self.history = text
        self.status = status
        self.server_info = server_info
        print("[=] LOading Attributes Is Completed")
        return

    def bind(self):
        print("[=] Trying To Binds")
        while True:
            try:
                self.s.connect((self.ip_address.get(), self.port.get()))
                print("[+] Connection Server Found")
                self.server_info.config(text="{}:{}".format(
                    self.ip_address.get(), self.port.get()))
                self.status.config(text="Connected", bg='lightgreen')
                threading.Thread(target=self.recv).start()
                break

            except:
                pass
```



```python
    def recv(self):
        while True:
            try:
                data = self.s.recv(1024)
                if data:
                    data = data.decode('utf-8')
                    data = 'Other : '+data+'\n'
                    start = self.history.index('end')+"-1l"
                    self.history.insert("end", data)
                    end = self.history.index('end')+"-1l"
                    self.history.tag_add("SENDBYOTHER", start, end)
                    self.history.tag_config("SENDBYOTHER", foreground='green')
            except Exception as e:
                print(e, 'recv')

    def send(self, text:str):
        try:
            self.s.sendall(text.encode('utf-8'))
        except:
            print("[=] Not Connected")
            pass


class ClientDialogBox(Tkinter.Tk):
    def __init__(self, *args, **kwargs):
        Tkinter.Tk.__init__(self, *args, **kwargs)
        self.resizable(0, 0)
        self.ip_address = Tkinter.StringVar()
        self.ip_address.trace_variable("w", self.update_status_info)
        self.port = Tkinter.IntVar()
        self.create_additional_widgets()

    def socket_connections_start(self):
        if len(self.ip_address.get().split('.')) == 4:
            print("Thread Started")
            threading.Thread(target=self.socket_connections).start()

    def socket_connections(self):
        print("[+] creating")
        self.s = SOCKETS()
        print("[=] Loading Attributes")
        self.s.load(self.ip_address, self.port, self.history,
                    self.status, self.server_info)
        print("[=] Bindings")
        self.s.bind()

    def update_status(self, Connection='Connected', color='lightgreen'):
        self.status.config(text=Connection, bg=color)
        return

    def update_status_info(self, *args, **kwargs):
        data = "{}:{}".format(self.ip_address.get(), self.port.get())
        self.server_info.config(text=data)
        return
```

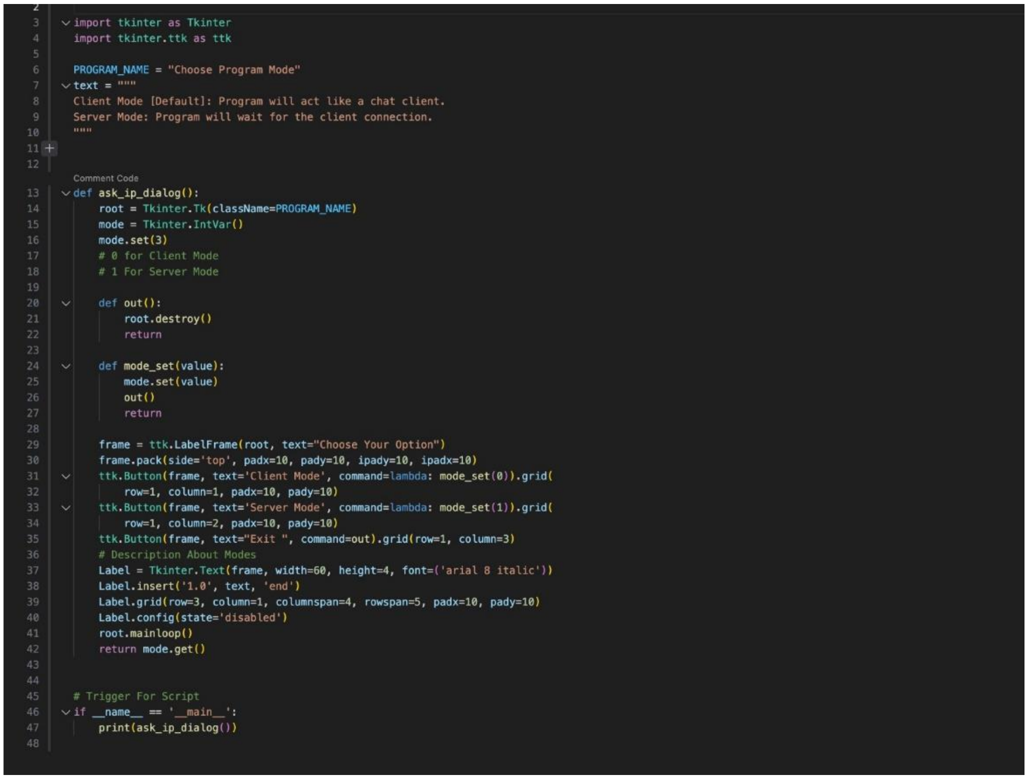


```python
    def create_additional_widgets(self):
        self.create_panel_for_widget()
        self.create_panel_for_connections_info()
        self.create_panel_for_chat_history()
        self.create_panel_for_sending_text()
        self.ask_ip_address()

    def ask_ip_address(self):
        ask_ip.ask_ip_dialog(self.ip_address, self.port)
        return

    def send_text_message(self):
        if self.status.cget('text') == 'Connected':
            input_data = self.Sending_data.get('1.0', 'end')
            if len(input_data) != 1:
                self.s.send(input_data)
                input_data = 'me: '+input_data
                start = self.history.index('end')+"-1l"
                self.history.insert("end", input_data)
                end = self.history.index('end')+"-1l"
                self.history.tag_add("SENDBYME", start, end)
                self.Sending_data.delete('1.0', 'end')
                self.history.tag_config("SENDBYME", foreground='Blue')

                pass
            else:
                print("[=] Input Not Provided")

        else:
            print("[+] Not Connected")

    def create_panel_for_sending_text(self):
        # Here Creating Sending Panel
        self.Sending_data = Tkinter.Text(
            self.Sending_panel, font=('arial 12 italic'), width=35, height=5)
        self.Sending_data.pack(side='left')
        self.Sending_Trigger = Tkinter.Button(self.Sending_panel, text='Send', width=15,
                                              height=5, bg='orange', command=self.send_text_message, activebackground='lightgreen')
        self.Sending_Trigger.pack(side='left')
        return

    def create_panel_for_chat_history(self):
        # Here Creating Chat History
        self.history = Tkinter.Text(self.history_frame, font=(
            'arial 12 bold italic'), width=50, height=15)
        self.history.pack()
        return

```



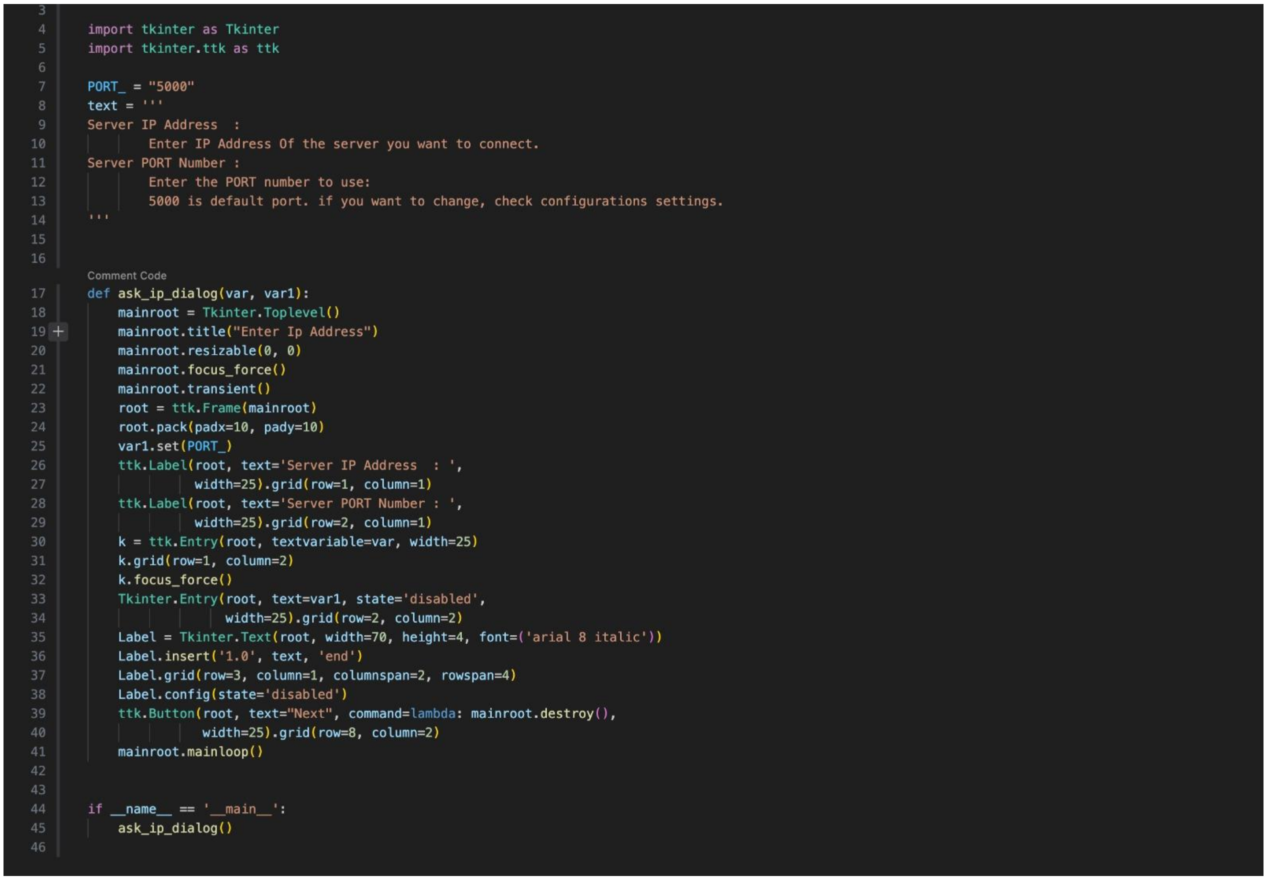
```python

    def create_panel_for_widget(self):
        # First For Connection Information
        self.Connection_info = Tkinter.LabelFrame(
            self, text='Connection Informations', fg='green', bg='powderblue')
        self.Connection_info.pack(side='top', expand='yes', fill='both')
        # Creating Second For Chatting History
        self.history_frame = Tkinter.LabelFrame(
            self, text='Chatting ', fg='green', bg='powderblue')
        self.history_frame.pack(side='top')
        # Creating Third For Sending Text Message
        self.Sending_panel = Tkinter.LabelFrame(
            self, text='Send Text', fg='green', bg='powderblue')
        self.Sending_panel.pack(side='top')
        return

    def create_panel_for_connections_info(self):
        self.frame = ttk.Frame(self.Connection_info)
        self.frame.pack(side='top', padx=10, pady=10)
        # Main Information Panel
        ttk.Label(self.frame, text='Your Entered Address   : ', relief="groove",
                  anchor='center', width=25).grid(row=1, column=1, ipadx=10, ipady=5)
        ttk.Label(self.frame, textvariable=self.ip_address, relief='sunken',
                  anchor='center', width=25).grid(row=1, column=2, ipadx=10, ipady=5)
        ttk.Label(self.frame, text='Your Entered Port Number  : ', relief="groove",
                  anchor='center', width=25).grid(row=2, column=1, ipadx=10, ipady=5)
        ttk.Label(self.frame, textvariable=self.port, relief="sunken",
                  anchor="center", width=25).grid(row=2, column=2, ipadx=10, ipady=5)
        ttk.Label(self.frame, text='Status            : ', relief="groove",
                  anchor="center", width=25).grid(row=3, column=1, ipadx=10, ipady=5)
        ttk.Label(self.frame, text='Connected with    : ', relief='groove',
                  anchor='center', width=25).grid(row=4, column=1, ipadx=10, ipady=5)
        self.status = Tkinter.Button(self.frame, text="Not Connected",
                                     anchor='center', width=25, bg="red", command=self.socket_connections)
        self.status.grid(row=3, column=2, ipadx=10, ipady=5)
        self.server_info = Tkinter.Label(self.frame, text="{}:{}".format(
            self.ip_address.get(), self.port.get()), relief='sunken', anchor='center', width=25)
        self.server_info.grid(row=4, column=2, ipadx=10, ipady=5)
        return


if __name__ == '__main__':
    ClientDialogBox(className='Python Chatting [Client Mode]').mainloop()

```

## Switch Module



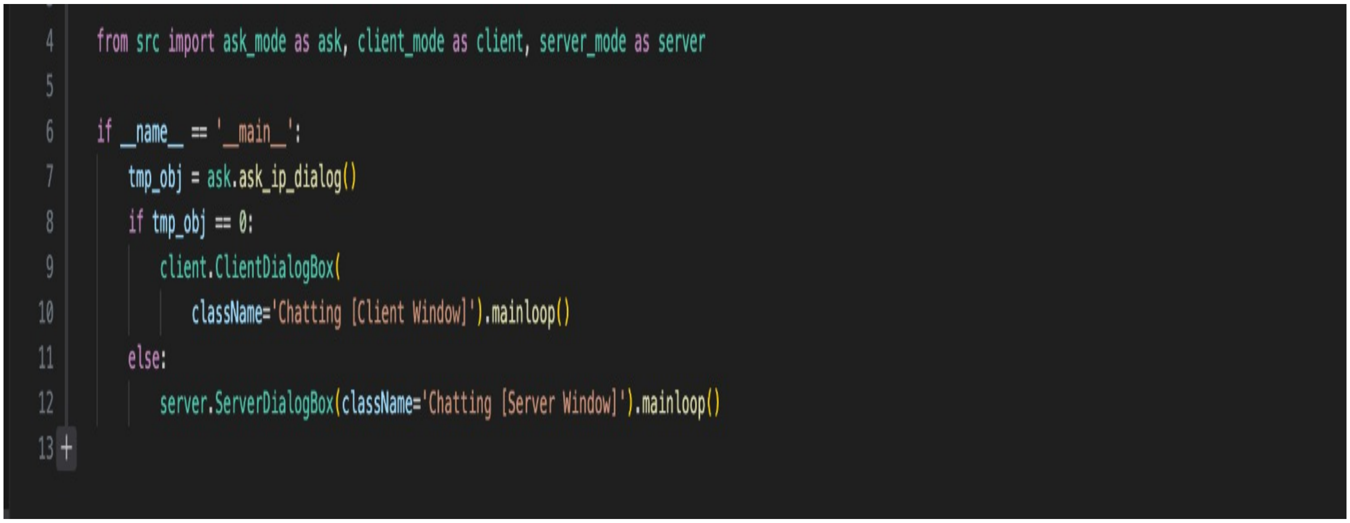
```python
import tkinter as Tkinter
import tkinter.ttk as ttk

PROGRAM_NAME = "Choose Program Mode"
text = """
Client Mode [Default]: Program will act like a chat client.
Server Mode: Program will wait for the client connection.
"""


def ask_ip_dialog():
    root = Tkinter.Tk(className=PROGRAM_NAME)
    mode = Tkinter.IntVar()
    mode.set(3)
    # 0 for Client Mode
    # 1 For Server Mode

    def out():
        root.destroy()
        return

    def mode_set(value):
        mode.set(value)
        out()
        return

    frame = ttk.LabelFrame(root, text="Choose Your Option")
    frame.pack(side='top', padx=10, pady=10, ipady=10, ipadx=10)
    ttk.Button(frame, text='Client Mode', command=lambda: mode_set(0)).grid(
        row=1, column=1, padx=10, pady=10)
    ttk.Button(frame, text='Server Mode', command=lambda: mode_set(1)).grid(
        row=1, column=2, padx=10, pady=10)
    ttk.Button(frame, text="Exit ", command=out).grid(row=1, column=3)
    # Description About Modes
    Label = Tkinter.Text(frame, width=60, height=4, font=('arial 8 italic'))
    Label.insert('1.0', text, 'end')
    Label.grid(row=3, column=1, columnspan=4, rowspan=5, padx=10, pady=10)
    Label.config(state='disabled')
    root.mainloop()
    return mode.get()


# Trigger For Script
if __name__ == '__main__':
    print(ask_ip_dialog())

```

## Module to collect IP Address



```python
import tkinter as Tkinter
import tkinter.ttk as ttk

PORT_ = "5000"
text = '''
Server IP Address  :
        Enter IP Address Of the server you want to connect.
Server PORT Number :
        Enter the PORT number to use:
        5000 is default port. if you want to change, check configurations settings.
'''


def ask_ip_dialog(var, var1):
    mainroot = Tkinter.Toplevel()
    mainroot.title("Enter Ip Address")
    mainroot.resizable(0, 0)
    mainroot.focus_force()
    mainroot.transient()
    root = ttk.Frame(mainroot)
    root.pack(padx=10, pady=10)
    var1.set(PORT_)
    ttk.Label(root, text='Server IP Address  : ',
              width=25).grid(row=1, column=1)
    ttk.Label(root, text='Server PORT Number : ',
              width=25).grid(row=2, column=1)
    k = ttk.Entry(root, textvariable=var, width=25)
    k.grid(row=1, column=2)
    k.focus_force()
    Tkinter.Entry(root, text=var1, state='disabled',
                  width=25).grid(row=2, column=2)
    Label = Tkinter.Text(root, width=70, height=4, font=('arial 8 italic'))
    Label.insert('1.0', text, 'end')
    Label.grid(row=3, column=1, columnspan=2, rowspan=4)
    Label.config(state='disabled')
    ttk.Button(root, text="Next", command=lambda: mainroot.destroy(),
               width=25).grid(row=8, column=2)
    mainroot.mainloop()


if __name__ == '__main__':
    ask_ip_dialog()

```

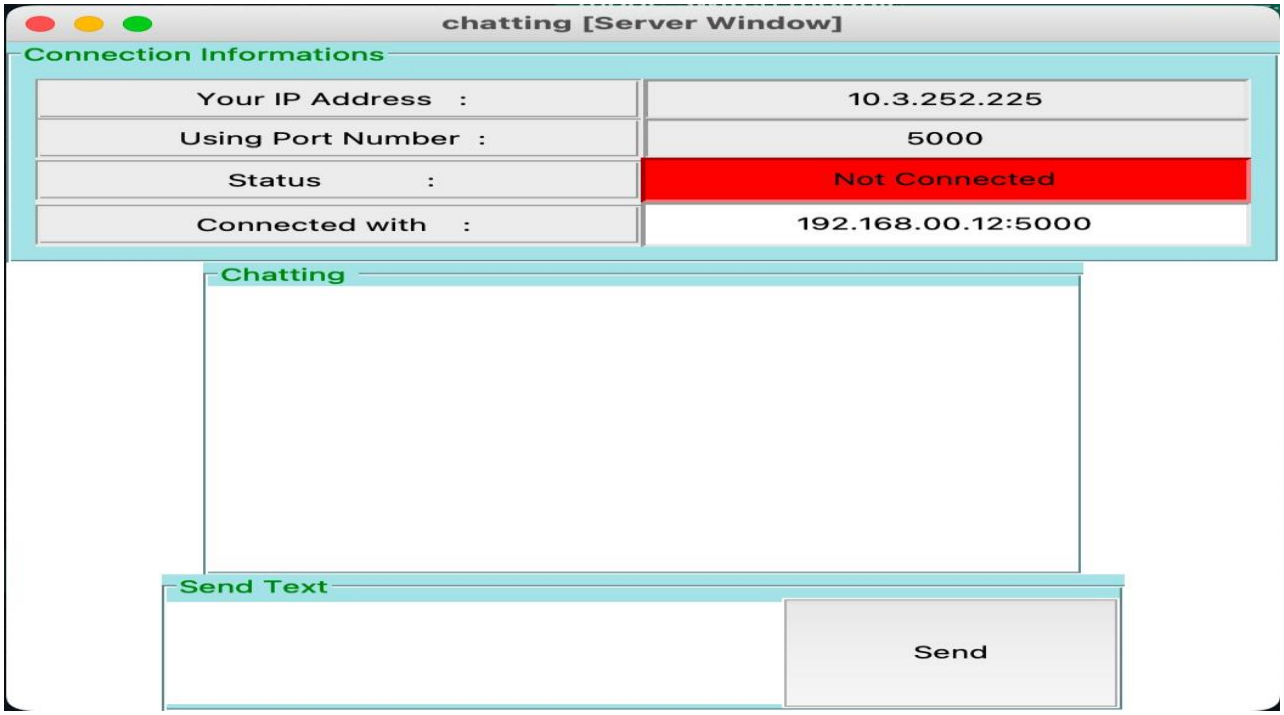
## Main Module



```
from src import ask_mode as ask, client_mode as client, server_mode as server

if __name__ == '__main__':
    tmp_obj = ask.ask_ip_dialog()
    if tmp_obj == 0:
        client.ClientDialogBox(
            className='Chatting [Client Window]').mainloop()
    else:
        server.ServerDialogBox(className='Chatting [Server Window]').mainloop()
```

# PYTHON RESLUTS

## Server



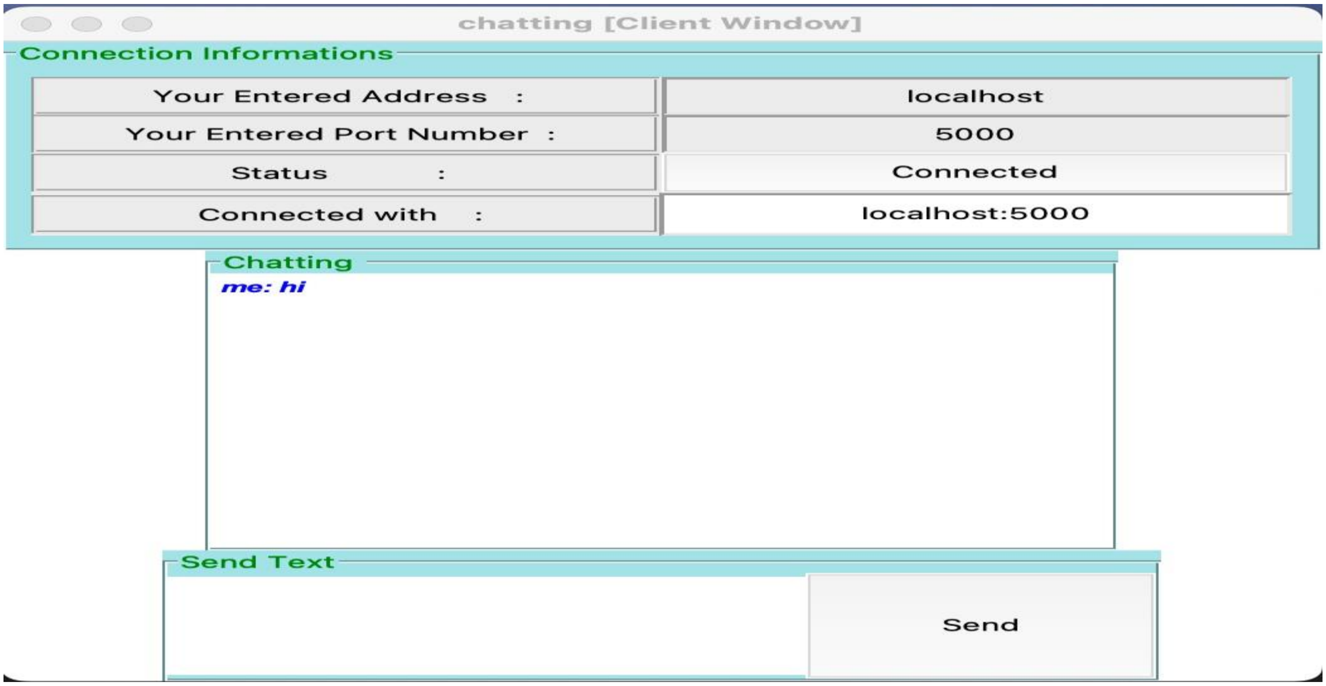
Your IP Address : 10.3.252.225

Using Port Number : 5000

Status : Not Connected

Connected with : 192.168.00.12:5000

## Client



Your Entered Address : localhost

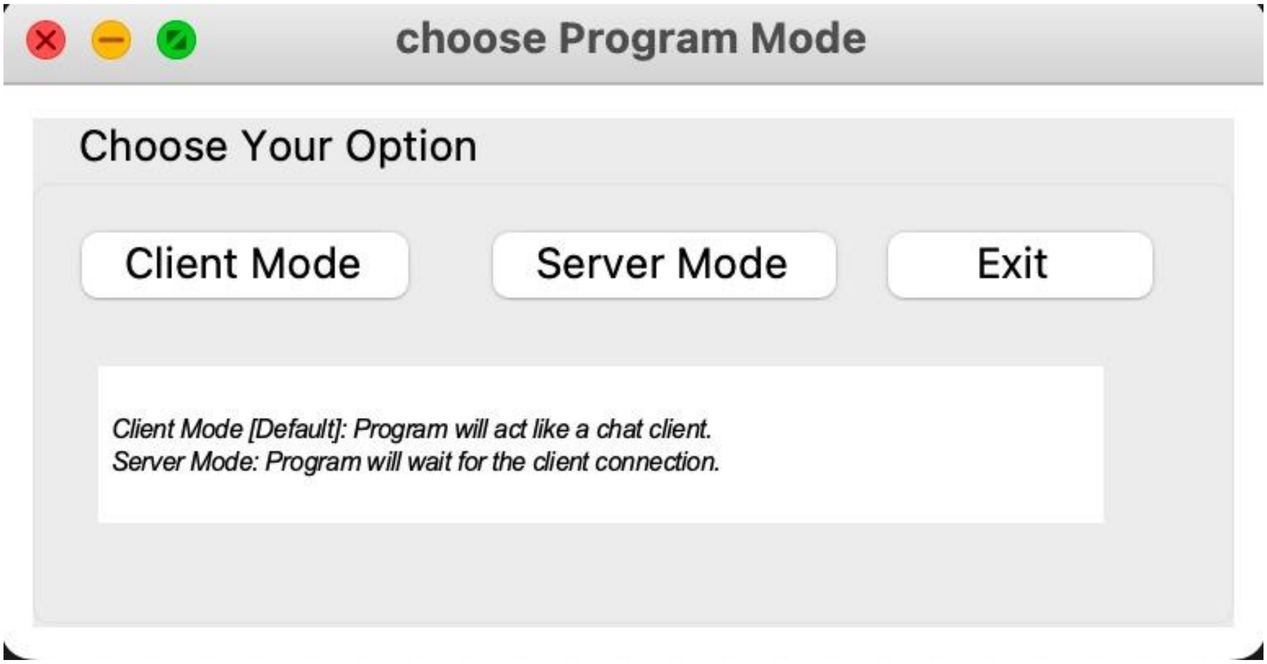
Your Entered Port Number : 5000

Status : Connected

Connected with : localhost:5000

*me: hi*

**Mode selection**



*Client Mode [Default]: Program will act like a chat client.*
*Server Mode: Program will wait for the client connection.*

# CONCLUSION

Chat applications have become an integral part of our daily lives, revolutionizing the way we communicate and connect with others. These applications have evolved from simple text-based messaging platforms to feature-rich tools that enable real-time communication, file sharing.

The development of chat applications has brought about numerous benefits, including:
Enhanced Communication, Global Connectivity, Improved Productivity, Enhanced Social Engagement, Accessibility and Convenience.

Despite their widespread adoption, chat applications also face certain challenges, including:
Security and Privacy Concerns, Information Overload, Misinterpretation and Misunderstandings, Cyberbullying and Harassment

Despite these challenges, chat applications remain indispensable tools for communication and connection. As technology advances, chat applications are likely to evolve further, incorporating new features, enhancing security measures, and addressing the challenges of information overload and digital addiction.

# REFERENCES

Professional chat application based on natural language processing" by K. Annamalai, E. Kharbat, and C. Goplakrishnan (2018)

Chat Reference: A Guide to Live Virtual Reference Services by Jana Ronan (2002

"Development of Chat Application" by Jhalak Mittal, Arushi Garg, Shivani Sharma (2020)

"Application of Reference Guidelines in Chat Reference Interactions" by Susan A. Bonits and Melissa S. Gross (2014)

Chat Message Reference" by OutSystems (2023)