

JAVA MUSIC PLAYER THAT CONNECTS PEOPLE WITH SIMILAR MUSIC TASTES USING SOCKET PROGRAMMING

*Report submitted to the SASTRA Deemed to be University
as the requirement for the course*

CSE302: COMPUTER NETWORKS

Submitted by

HARSHITH TEJA DONTUJU
(Reg no: 124003113, CSE)

December 2022



SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)



THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA - 613 401



SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)



THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SCHOOL OF COMPUTING THANJAVUR - 613401

Bonafide Certificate

This is to certify that the report titled “**Java Music Player that connects people with similar music tastes using Socket Programming**” submitted as a requirement for the course, **CSE302: COMPUTER NETWORKS** for B.Tech is a bonafide record of the work done by **Mr. Harshith Teja Dontoju (Reg no. 124003113, CSE)** during the academic year 2022-23, in the school of computing.

Project Based Work Viva voce held on _____

Examiner 1

Examiner 2

ACKNOWLEDGEMENTS

First of all, I would like to thank God Almighty for his endless blessings.

I would like to express my sincere gratitude to **Dr. S. Vaidyasubramaniam, Vice-Chancellor** for his encouragement during the span of my academic life at SASTRA Deemed University.

I would forever remain grateful and I would like to thank **Dr. A. Umamakeswari, Dean, School of Computing, and R. Chandramouli, Registrar** for their overwhelming support provided during my course span at SASTRA Deemed University.

I am extremely grateful to **Dr. Shankar Sriram, Associate Dean, School of Computing** for his constant support, motivation and academic help extended for the past three years of my life in the School of Computing. I also thank him for providing me with an opportunity to do this project and for his guidance in successfully completing the project.

I also thank all the Teaching and Non-teaching faculty, and all other people who have directly or indirectly helped me through their support, encouragement, and all other assistance extended for the completion of my project and for the successful completion of all courses during my academic life at SASTRA Deemed University.

Finally, I thank my parents and all others who help me acquire this interest in the project and aided me in completing it within the deadline without much struggle

ABBREVIATIONS

ACK	Acknowledgment
TCP	Transmission Control Protocol
IP	Internet Protocol
JDK	Java Development Kit
JRE	Java Runtime Environment

ABSTRACT

Java Music Player is a classic music player that allows users to play, pause and switch between music. Besides the music player connects various people with similar tastes in music. It is an application of multiple layers of computer networks that explores various operations and dimensions from playing an audio file to texting strangers who have familiar music taste like you with the help of client-server architecture. This music player acts as an icebreaker among people. It doesn't allow you to rack your brain on how to start a conversation with a stranger. Because as soon as a user gets connected to the server, the server stores the playlist of that user and it scans the playlist of a user against every other user connected to the server at that particular time and provides a percentage match in music taste from one user to all other users connected to the server. A user can use it as a starter for a conversation with someone who is completely new to them.

In this project, TCP/IP protocols are implemented with the help of socket programming to connect various users to a single server and make users send texts to each other.

In a nutshell, the music player allows a user to

- Play Music
- Pause music
- Switch between music
- Identify the percentage of music taste match a user has with others
- Text other users

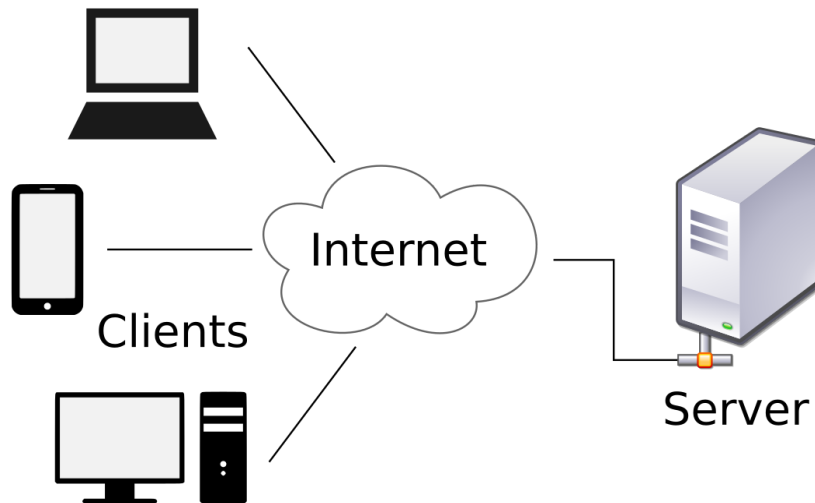
KEY WORDS - Socket Programming, Client-Server architecture, Percentage of Music match, TCP, IP

TABLE OF CONTENTS

S.no	TOPIC	Page no
1.	Bonafide Certificate	ii
2.	Acknowledgments	iii
3.	Abbreviations	iv
4.	Abstract	v
5.	Table of Contents	vi
6.	Introduction	1
7.	Prerequisites	5
8.	Objective	7
9.	Literature Survey	8
10.	Methodology	8
11.	Source Code	9
12.	Output Snapshots	29
13.	Conclusion	37
14.	Future Works	37
15.	References	37

INTRODUCTION

CLIENT-SERVER ARCHITECTURE

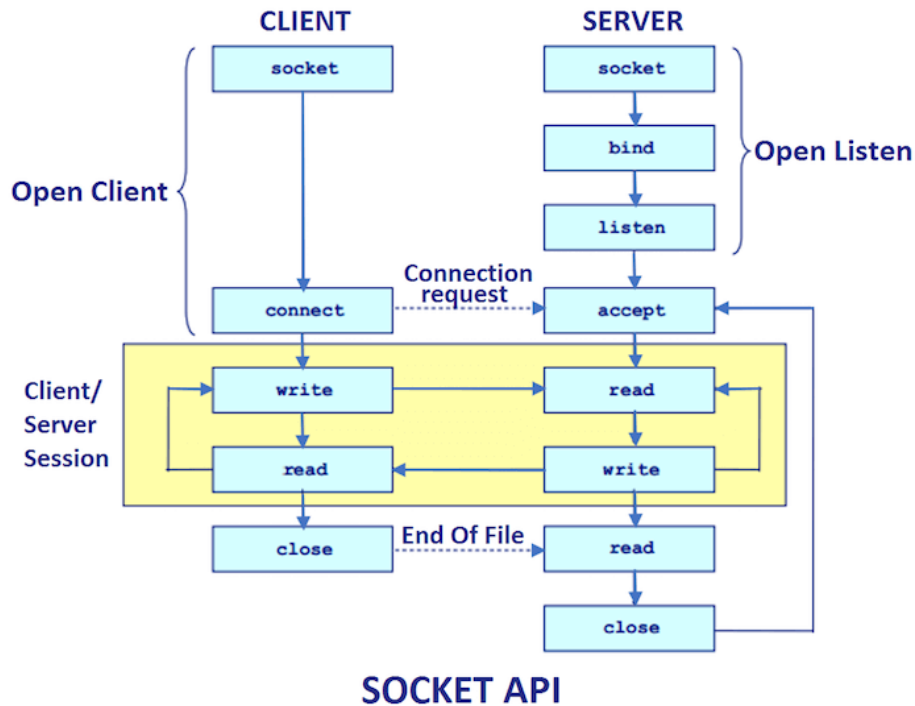


Internet Communication is an analogy to telephone or postal communication. Each communication is initiated by one party, if a party sends a letter or makes a telephone call, then the other party responds to the initiator's contact by sending a reply letter or by picking up the phone and talking.

Client-Server architecture works pretty much the same. The client initiates communication, while the server waits passively and then responds to clients that contact it. The client and server together compose the application. The server's job is to provide the service to the clients, and the client's job is to receive that service from the server. And the communication between the client and server starts only the initiated connection from the client is successfully accepted by the server.

The Client-Server architecture uses the Socket API to connect with each other. That's where socket programming comes in handy. To establish a connection the client needs to know the server's IP address and port number initially but not vice versa. This is analogous to a telephone call - in order to make a call, you must know the phone number of a person, but in order to be called, a person need not know the telephone number of the caller.

SOCKET PROGRAMMING



Socket programming is an abstraction through which an application may send or receive data. Sockets are used in socket programming one each on the server side and client side. A socket allows an application to plug into the network and communicate with other applications that are plugged into the same network.

Different types of sockets use different protocols to carry out communication between applications. One such protocol is the TCP/IP protocol. The main types of sockets in TCP/IP are stream sockets and datagram sockets. Stream sockets use TCP and are reliable. Datagram sockets use UDP and are unreliable. A TCP/IP protocol is uniquely identified by an internet address and a port number.

The `java.net.Socket` class represents a socket. To open a socket we normally follow the below procedure:

```
Socket socket = new Socket("localhost",8888);
```


- Here the first parameter is the IP address of the Server. As the client and server are running on the same machine, “localhost” has been mentioned as the IP address. If both the server and client are running on different machines then the server's IP address must be passed as the first argument (Ex: 192.72.2.67).
- The second parameter is the port number. It's just a number representing which application to run on a server. Port numbers vary from 0 to 65535

Creating a Server

To create a server, we must first create an instance of `ServerSocket` class. On the server side, we use a port number for client-server communication. The `accept()` method passively waits for the clients who are trying to connect to the server. If the client connects with the given port number and server's IP address. The `accept()` method accepts the connection and returns an instance of `Socket`.

Example:

```
ServerSocket ss = new ServerSocket(8888);
Socket server = ss.accept();
```

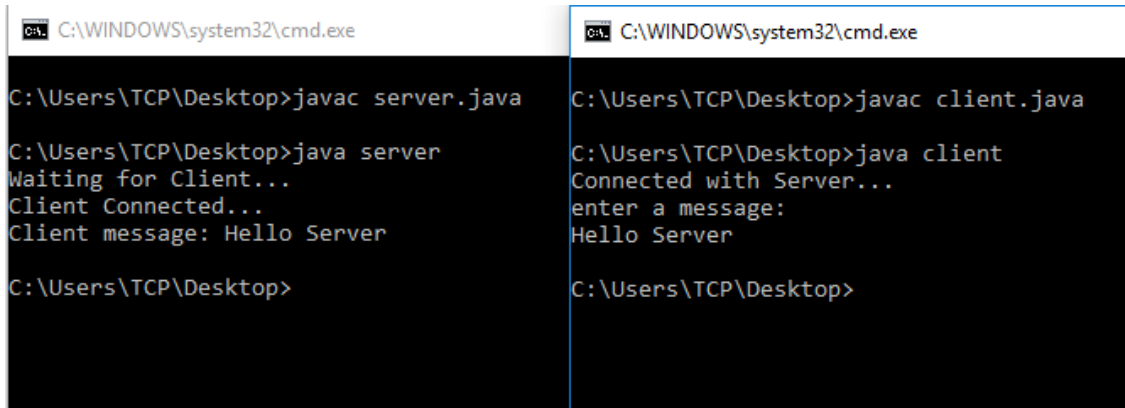
Creating a Client

To create a client, we must first create an instance of the `Socket` class, and we need to pass the IP address of the server, and the port number on which the server is waiting as the parameter.

Example:

```
Socket client = new Socket("192.72.2.67",8888);
```

To establish a connection the server's code must run first, and then when the server is waiting for a connection, the client's code should start running. Then we get something like this



The image shows two side-by-side Windows command prompt windows. Both windows have a title bar that reads 'C:\WINDOWS\system32\cmd.exe'. The left window shows the execution of a Java server program. The user enters 'javac server.java' and 'java server'. The program outputs 'Waiting for Client...', 'Client Connected...', and 'Client message: Hello Server'. The right window shows the execution of a Java client program. The user enters 'javac client.java' and 'java client'. The program outputs 'Connected with Server...' and prompts 'enter a message:'. The user enters 'Hello Server'.

```
C:\WINDOWS\system32\cmd.exe
C:\Users\TCP\Desktop>javac server.java
C:\Users\TCP\Desktop>java server
Waiting for Client...
Client Connected...
Client message: Hello Server
C:\Users\TCP\Desktop>

C:\WINDOWS\system32\cmd.exe
C:\Users\TCP\Desktop>javac client.java
C:\Users\TCP\Desktop>java client
Connected with Server...
enter a message:
Hello Server
C:\Users\TCP\Desktop>
```

TCP/IP PROTOCOL

TCP/IP stands for Transmission Control Protocol/Internet Protocol. The TCP/IP protocol is the bread and butter of today's Internet. These communication protocols are used to connect network devices to the internet.

TCP/IP provides end-to-end encryption for data exchanges on the internet. It handles how data is broken into packets, addressed, transmitted, routed, and received at its destination.

TCP is said to be connection-oriented because before one application process can start sending data to another application, the two processes must first “handshake” with each other—This means that some preliminary segments have to be sent to each other to set the parameters to ensure data transfer. As part of TCP connection establishment, both sides of the connection will initialize a number of TCP state variables

The concept of how applications can create communication channels over a network is defined by TCP. It also defines how a message is broken into small packets before they are transmitted over the internet and reassembled in the same order at the destination.

IP defines how to address and route each packet to ensure that it reaches the correct destination. Each gateway computer on the network checks this IP address to determine where to forward messages. The IP protocol specifies the format of the packets sent and received among routers and end systems. Currently, two versions of the IP are in use today: IPv4 and IPv6. Version 6 is the latest version of IP and offers many advantages like longer bit address length, complexity, and efficiency.

PREREQUISITES

JAVA JDK

Java JDK(Java Development Kit) is a key platform component for building Java applications. The JDK is one of the 3 core technology packages used in java programming along with JVM(Java Virtual Machine) and JRE(Java Runtime Environment)

The main components of JDK are:

- Java Runtime Environment(JRE)
- An interpreter/loader(Java)
- A compiler(javac)
- An archiver(jar)

The Java Runtime Environment in JDK is usually called Private Runtime because it is separated from the regular JRE and has extra content. The Private Runtime in JDK contains a JVM and all the class libraries present in the production environment, as well as additional libraries useful to developers, e.g, internationalization libraries and the IDL libraries.

IntelliJ IDE

IntelliJ is one of the most powerful and popular Integrated Development Environments (IDEs) for Java, developed and maintained by JetBrains, it is available as the Community and Ultimate editions. This feature-rich IDE enables rapid development and improves code quality.

IntelliJ IDEA has some highly productive Java code completion features. Its predictive algorithm can intelligently assume what a coder is trying to type, and it completes it automatically for the programmer.

Some useful features of IntelliJ are

- Intelligent code completion
- Duplicates detection
- Smart and Useful Shortcuts
- Built-in debugger

Java Packages

The Java packages used in the code are as follows:

- **java.io:** Java I/O (input and output) are used to process the input and generate the output. Java uses the concept of a stream to speed I/O operations. The java.io package contains all the classes required for input/output operations. Java I/O API can be used to perform file handling in java
- **java.util:** This includes the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and various utility classes (a string tokenizer, a random-number generator, and a bit array).
- **javax.net:** This Provides the classes needed to create an applet and the classes an applet uses to communicate with its applet context. It Contains all of the classes for creating user interfaces and for drawing graphics and images.
- **javax.swing:** Java Swing tutorial is a part of Java Foundation Classes (JFC) which is used to create window-based applications. It is built on top of AWT API(Abstract Windowing Toolkit) and entirely written in java. Unlike AWT, Java Swing provides lightweight, platform-independent components. The javax.swing package provides classes for java Swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser, etc
- **javax.sound:** The main sub-packages of javax.sound namely javax.sound.sampled and javax.sound.sampled.spi provides interfaces and classes for retrieving, processing, and playing sampled audio data and it supplies abstract classes for service providers to subclass when offering new audio devices, sound file readers and writers, or audio format converters respectively.
- **java.awt:** The java.awt package contains all of the classes for creating user interfaces and for drawing graphics and images, such as TextFields, Labels, TextAreas, RadioButtons, etc... This makes Java GUI programming easier for the users, and this awt package is platform independent.

OBJECTIVE

To design a Java Music Player application that connects people with similar music tastes using Socket Programming

The goal is to design a new unique feature using COMPUTER NETWORKS concepts that excites social media users and gives them a fresh feeling. There are a lot of Music Players on social media that allow users to play and enjoy free music but no Music Player allows a user to chat with other users using the same application. The idea is to establish a connection between users who use the Music Player which can be done using Socket Programming. But what makes the connection possible is the interesting fact that this project talks about.

As mentioned earlier, connecting Music Player users is done based on their music tastes. As per the programmer's point of view, this is done with the help of the user's playlist. Whenever a user connects to the server with the username, the server takes the user's playlist as the input implicitly(without the knowledge of the user and without manual inputting). The server now tries to match the songs from the user's playlist with the playlists of other users(who are connected to the server) and returns a percentage match along with that user's name and displays it on the client's console.

With the help of the percentage match in music taste provided, the users can initiate a chat with each other and the percentage match in music taste can actually act as an ice breaker among users.

Finally, when the user feels to take a break, he/she can exit the application by clicking on the exit button provided on the interface. Clicking on this button will send a message to all connected users that this user has disconnected from the server

LITERATURE SURVEY

There are a multitude of Music Player applications in the market, and almost all of them provide great music and awesome features like music downloads, and music suggestions. All the algorithms and techniques that have been used are, to provide a better user experience but there aren't many Music Players that allow users using the Music Player to interact with other.

After a lot of brainstorming sessions, the idea to connect users of the Music Player based on their own music tastes has been developed. This idea is actually implemented on the server side, the server has the information of all connected users and it matches the playlist of each user to all other users connected to the server and returns a percentage match in music taste for every user

In the hunt to connect users using Music Player with the help of Socket Programming, the book **"TCP/IP Sockets in Java"** by **Kenneth L. Calvert** and **Michael J. Donahoo** has been referred to and studied thoroughly. And referring to online websites, articles, and queries related to the Socket Programming concept made it easier in dealing with the problems pertaining to Sockets while building this project. The concepts mentioned in the books were applied and executed and debugged successfully which resulted in the success of this project

METHODOLOGY

Socket Programming played a significant role in building this project. The connection between users has been established with the help of client-server architecture using TCP/IP protocols where multiple clients try and connect to the single server that provides the service.

In matching the music percentages of the users a method/function is used which takes the user's playlist as a parameter and maps it with the other user's playlist and for every song match a count variable is increased, with which percentage is calculated and returned to the client's console

SOURCE CODE

Server.java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Vector;

import static java.lang.System.out;

public class ChatServer {

    Vector<String> users = new Vector<String>(); //users list
    Vector<HandleClient> clients = new Vector<HandleClient>(); //Connected clients

    String[][] playlist = new String[30][30]; //to store playlist's of users
    int[] playlistSizeArr = new int[30]; //to store each playlist size
    int index =0; //keeps track of index

    public void process() {

        try(ServerSocket server = new ServerSocket(4444)) { //waits for connection on specified
port no

            out.println("Server Started...");
            while (true) {
                try{
                    Socket client = server.accept(); //accept connection
                    //add incoming client to connected clients vector.
                    HandleClient c = new HandleClient(client);
                    clients.add(c);
                }catch(Exception e)
                {
                    out.println("can't accept");
                }
            }
        }
```

```

    } // end of while
    } catch (Exception e) {
        out.println(e);
    }
}

public static void main(String... args) {
    new ChatServer().process();
} // end of main

public void broadcast(String user, String message) { //broadcasts message to all connected
users
    // send message to all connected users
    for (HandleClient c : clients) {
        c.sendMessage(user, message);
        c.updateOnlineUsers(c.name);
    }
}

/*
 * Inner class, it is responsible for handling incoming clients.
 * Each connected client will set its own thread.
 */
class HandleClient extends Thread {

    String name = "", plist; // client name/username
    BufferedReader input; //gets input from client
    PrintWriter output; //sends output to client

    public HandleClient(Socket client) throws Exception {
        // get input and output streams
        input = new BufferedReader(new InputStreamReader(client.getInputStream()));
        output = new PrintWriter(client.getOutputStream(), true);

        // read name
        name = input.readLine();
        users.add(name); // adding unames to users vector
    }
}

```



```

    plist = input.readLine();
    if(plist.equals("playlist"))
        getPlaylist();

    broadcast(name, " Has connected!");

    getOnlineUsers();

    start();
}

public void sendMessage(String uname, String msg) { //sends grp message
    output.println("[Grp Msg] " + uname + " : " + msg); //sending the grp msg to all clients
}

public void sendPvtMessage(String uname, String msg) { //sends private message
    output.println("[PvtMsg] "+uname + " : " + msg); //sends the pvt msg to only targeted
client
}

public void getOnlineUsers() { //get the list of online users
    output.println("Users List"); //informing the client that server is sending users list
    output.println(users.size()-1); //sending usersSize data

    int skippedIndex = users.size()-1; //skipping user's index
    for(int i=0 ; i<users.size()-1 ; i++)
        output.println(users.get(i) + " : " + getOnlinePercentage(i,skippedIndex) + "%");
}

public void updateOnlineUsers(String uname) //update the list of online users
{

    output.println("Users List"); //informing the client that server is sending users list
    output.println(users.size()-1); //sending usersSize data

    int skippedIndex=0;
    for(int i=0 ; i<users.size() ; i++)
    {
        if(users.get(i).equals(uname))

```

```

        {
            skippedIndex = i;
            break;
        }
    }
    for(int i=0 ; i<users.size() ; i++)
    {
        if(users.get(i).equals(uname))
        {
            continue; //user's own details are not needed
        }

        output.println(users.get(i) + " : " + getUpdatePercentage(i,skippedIndex) + "%");
    }
}

public void getPlaylist() throws Exception //gets the playlist of the user
{
    String songs;
    songs = input.readLine();
    playlistSizeArr[index] = Integer.parseInt(songs); //songsSize[k]

    while(index <users.size())
    {
        for(int j = 0; j< playlistSizeArr[index] ; j++)
            playlist[index][j] = input.readLine();

        index++;
    }
    index--;

}

public String getOnlinePercentage(int i, int skippedIndex) //gets percentage match a user
has with other user

```

```

{
    float percentage;
    int count = 0;
    int songsSize1 = playlistSizeArr[skippedIndex];
    int songsSize2 = playlistSizeArr[i];
    int m,p;

    p = i;
    m=skippedIndex;

    for(int j=0 ; j<songsSize1 ; j++)
    {
        for(int n=0 ; n<songsSize2 ; n++)
        {
            if(playlist[m][j].equals(playlist[p][n])) //matching songs from each user
                count++; //incrementing count if it matches
        }
    }

    percentage = ((float)count/songsSize1)*100; //finding percentage according to main user

    return String.format("%.2f",percentage); //returns roundup value of percentage
}

public String getUpdatePercentage(int i , int skippedIndex)
{
    float percentage;
    int count = 0;
    int songsSize1 = playlistSizeArr[skippedIndex];
    int songsSize2 = playlistSizeArr[i];
    int m,p;

    p = i;
    m=skippedIndex;

    for(int j=0 ; j<songsSize1 ; j++)
    {
        for(int n=0 ; n<songsSize2 ; n++)

```

```

        {
            if(playlist[m][j].equals(playlist[p][n])) //matching songs from each user
                count++; //incrementing count if it matches
        }
    }

    percentage = ((float)count/songsSize1)*100; //finding percentage acc to main user

    return String.format("%.2f",percentage);
}

public void run() { //starts the thread
    String line;
    try {
        while (true) {
            line = input.readLine();
            if (line.equals("!end")) {
                //notify all for user disconnection
                broadcast(name, " Has disconnected!");
                clients.remove(this);
                users.remove(name);
                break;
            }
            else if(line.equals("pvtMsg")) //if the msg is private
            {
                String targetUserName = input.readLine(); //reads name of that user to initiate
                private chat
                String msgToBeSent = "";
                for(HandleClient c : clients)
                {
                    if(c.name.equals(targetUserName)) //if the name matches with any client name
                    .equals(line)
                    {
                        targeted user
                        msgToBeSent = input.readLine(); //the msg which needs to be sent to
                        c.sendPvtMessage(name,msgToBeSent); //sends pvt msg only to that user
                        break;
                    }
                }
            }
        }
    }
}

```

```

        String senderUserName = name;
        for(HandleClient c : clients)
        {
            if(c.name.equals(senderUserName)) //if the name matches with any client name
.equals(line)
            {
                c.sendPvtMessage("You to "+ targetUserName,msgToBeSent); //sends what
msg the user has sent to other's (user's own msg to user)
                break;
            }
        }

        }
        else {
            broadcast(name, line); // method of outer class - send messages to all

        }

    } // end of while
} // try
catch (Exception ex) {
    System.out.println(ex.getMessage());
}
} // end of run()
} // end of inner class
} // end of Server

```

Client.java

```
import java.io.*;
import java.net.*;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import static java.lang.System.out;

import java.util.*;
import java.util.Timer;

import javax.sound.sampled.*;
import javax.swing.border.Border;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.filechooser.FileSystemView;

public class ChatClient extends JFrame implements ActionListener, ListSelectionListener {
    String uname, selectedUserListView;
    PrintWriter pw;
    BufferedReader br;
    public JTextArea taMessages;
    JList<String> usersList;
    DefaultListModel<String> listModel;

    JTextField tfInput;
    JButton btnSend, btnExit;
    Socket client;

    JLabel musicPlayer, musicTable, musicClients, songName, selectASong;
    JButton play, pause, previous, next;
    JButton openDirectory;
    JPanel mainPanel, mplayerPanel, mplayerPanelUp, mplayerPanelCenter, mplayerPanelDown,
    mTablePanel, clientDetailsPanel;
```

```

Border greenBorder, blueBorder, blackBorder;
JProgressBar songBar;
Timer timer;//pgbar timer
TimerTask task;//pgbar timer task

//song related declarations
File file;
AudioInputStream audioStream;
Clip clip;

//prv, nxt song
File directory;
File[] filesInTheDirectory;
ArrayList<File> songs;
int songNumber; //tracks song number

File song;
JFileChooser j;

public ChatClient(String uname, String servername) throws Exception
{
    super("Connected as: " + uname); // set title for the JFrame

    client = new Socket(InetAddress.getByName(servername),4444); //requests for a
connection
    this.uname = uname;
    br = new BufferedReader(new InputStreamReader(client.getInputStream()));
    pw = new PrintWriter(client.getOutputStream(), true);
    pw.println(uname); // send name to server

    //bring up the chat interface
    buildInterface();

    pw.println("playlist");
    pw.println(songs.size());
    for (File value : songs)
        pw.println(value);

```

```

        new MessagesThread().start(); // create thread that listens for messages
    }

    public void buildInterface() throws UnsupportedOperationException, IOException,
    LineUnavailableException{

        //GUI

        //retrieving songs from the user
        songs = new ArrayList<File>();
        directory = new File("D:\\Songs(.wav files)");
        filesInTheDirectory = directory.listFiles();
        if(filesInTheDirectory != null) //adding all the user's song to songs arraylist
        {
            for(File files: filesInTheDirectory)
                songs.add(files);
            // songs.addAll(Arrays.asList(filesInTheDirectory));
        }

        //Panels and Labels
        //music player panel

        musicPlayer = new JLabel("MUSIC PLAYER",SwingConstants.CENTER); //MUSIC
        PLAYER label
        musicPlayer.setForeground(Color.GREEN);
        musicPlayer.setFont(new Font(null,Font.BOLD,30));

        mplayerPanelUp = new JPanel(new BorderLayout()); //top panel in music player panel
        mplayerPanelUp.setBackground(Color.BLACK);
        mplayerPanelUp.add(musicPlayer,"North");
        add(mplayerPanelUp);

        //label and progress bar for center panel
        songName = new JLabel("Song Name",SwingConstants.CENTER); //song name label
        songName.setForeground(Color.CYAN);
        songName.setFont(new Font(null,Font.BOLD,30));

        songBar = new JProgressBar(); //progress bar

```



```

songBar.setValue(0);
songBar.setBackground(Color.GREEN);
songBar.setForeground(Color.LIGHT_GRAY);
songBar.setStringPainted(true);

mplayerPanelCenter = new JPanel(); //center panel in music player panel
mplayerPanelCenter.setLayout(new
BoxLayout(mplayerPanelCenter,BoxLayout.X_AXIS));
mplayerPanelCenter.setBackground(Color.BLACK);
mplayerPanelCenter.add(Box.createHorizontalGlue());
mplayerPanelCenter.add(songName);
mplayerPanelCenter.add(Box.createHorizontalGlue());
mplayerPanelCenter.add(songBar);
mplayerPanelCenter.add(Box.createHorizontalGlue());

//Buttons for bottom of music player panel

play = new JButton("play");
pause = new JButton("pause");
previous = new JButton("previous");
next = new JButton("Next");

mplayerPanelDown = new JPanel(); //bottom panel in music player panel
mplayerPanelDown.add(previous);
mplayerPanelDown.add(play);
mplayerPanelDown.add(pause);
mplayerPanelDown.add(next);

mplayerPanelDown.setLayout(new GridLayout(1,4)); //all buttons get displayed in a single
row

play.addActionListener(this); //adding functionalities to the button
pause.addActionListener(this);
previous.addActionListener(this);
next.addActionListener(this);

greenBorder = BorderFactory.createLineBorder(Color.GREEN,5);

```

```

mplayerPanel = new JPanel(new BorderLayout()); //main music player panel
mplayerPanel.setBounds(10,10,500,500);
mplayerPanel.setBorder(greenBorder);
mplayerPanel.add(mplayerPanelUp,BorderLayout.NORTH);
mplayerPanel.add(mplayerPanelDown,BorderLayout.SOUTH);
mplayerPanel.add(mplayerPanelCenter,BorderLayout.CENTER);
add(mplayerPanel);

//Music Table Panel
musicTable = new JLabel("MUSIC TABLE",SwingConstants.CENTER); //MUSIC TABLE
label
musicTable.setForeground(Color.BLUE);
musicTable.setFont(new Font(null,Font.BOLD,30));

openDirectory = new JButton("Open Songs List"); //open directory button
openDirectory.addActionListener(this);

selectASong = new JLabel("Select a song you would like to
play",SwingConstants.CENTER); //select a song label
selectASong.setForeground(Color.BLUE);
selectASong.setFont(new Font(null,Font.BOLD,25));

blueBorder = BorderFactory.createLineBorder(Color.BLUE,5);

mTablePanel = new JPanel(new BorderLayout()); //Music Table panel
mTablePanel.setBounds(530,10,500,500);
mTablePanel.setBackground(Color.yellow);
mTablePanel.setBorder(blueBorder);
mTablePanel.add(musicTable,BorderLayout.NORTH);
mTablePanel.add(selectASong,BorderLayout.CENTER);
mTablePanel.add(openDirectory,BorderLayout.SOUTH);
add(mTablePanel);

//Music Clients Panel
musicClients = new JLabel("MUSIC CLIENTS",SwingConstants.CENTER); //MUSIC
CLIENTS label
musicClients.setFont(new Font(null,Font.BOLD,30));

```

```

blackBorder = BorderFactory.createLineBorder(Color.BLACK,5);

clientDetailsPanel = new JPanel(new BorderLayout()); //Main client details panel
clientDetailsPanel.setBounds(1050,10,450,500);
clientDetailsPanel.setBackground(Color.CYAN);
clientDetailsPanel.setBorder(blackBorder);
clientDetailsPanel.add(musicClients,"North");


//Chat application GUI - adding it to the clientDetailsPanel
btnSend = new JButton("Send");
btnExit = new JButton("Exit");


//chat area
taMessages = new JTextArea();
taMessages.setRows(25);//28
taMessages.setColumns(29);//33
taMessages.setEditable(false);


//online users list
usersList = new JList<>();
listModel = new DefaultListModel<>();
usersList.setModel(listModel);


//Scroll panel (chat area and online users list)
JScrollPane chatPanel = new JScrollPane(taMessages,
JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
JScrollPane onlineUsersPanel = new JScrollPane(usersList,
JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);


//top panel (chatPanel, onlineUsersPanel)
JSplitPane tp = new JSplitPane();
tp.setBackground(Color.CYAN);
tp.setLeftComponent(onlineUsersPanel);
tp.setRightComponent(chatPanel);
clientDetailsPanel.add(tp, "Center");

```

```

    tfInput = new JTextField(30); //user input field

    //bottom panel (input field, send and exit)
    JPanel bp = new JPanel(new FlowLayout());
    bp.setBackground(Color.CYAN);
    bp.add(tfInput);
    bp.add(btnSend);
    bp.add(btnExit);
    clientDetailsPanel.add(bp, "South");

    btnSend.addActionListener(this);
    tfInput.addActionListener(this); //allow user to press Enter key in order to send message
    btnExit.addActionListener(this);

    userList.addListSelectionListener(this); //allows users to click on any value on JList

    add(clientDetailsPanel);

    setLayout(null);
    setTitle("MUSIC PLAYER");
    setSize(1550,800);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setVisible(true);
}

@Override
public void actionPerformed(ActionEvent evt) {
    if (evt.getSource() == btnExit) {
        pw.println("!end"); // send end to server so that server know about the termination
        System.exit(0);
    }
    else if(evt.getSource() == play)
    {
        clip.start(); //song starts playing
    }
    else if(evt.getSource() == pause)
    {
        clip.stop(); //song stops playing
    }
}

```

```

}
else if(evt.getSource() == previous) //navigating to previous song
{
    if(songNumber > 0)
    {
        songNumber--;
        clip.stop();
        try
        {
            audioStream = AudioSystem.getAudioInputStream(songs.get(songNumber));
            clip = AudioSystem.getClip();
            clip.open(audioStream);
            clip.start();
            songName.setText(songs.get(songNumber).getName());
        } catch (Exception e)
        {
            out.println(e);
        }
    }
}
else
{
    songNumber = songs.size() - 1;
    clip.stop();
    try
    {
        audioStream = AudioSystem.getAudioInputStream(songs.get(songNumber));
        clip = AudioSystem.getClip();
        clip.open(audioStream);
        clip.start();
        songName.setText(songs.get(songNumber).getName());
    } catch (Exception e)
    {
        out.println(e);
    }
}
}
else if(evt.getSource() == next) //navigating to next song
{

```

```

if(songNumber < songs.size() - 1)
{
    songNumber++;
    clip.stop();
    try
    {
        audioStream = AudioSystem.getAudioInputStream(songs.get(songNumber));
        clip = AudioSystem.getClip();
        clip.open(audioStream);
        clip.start();
        songName.setText(songs.get(songNumber).getName());
    } catch (Exception e)
    {
        out.println(e);
    }
}
else
{
    songNumber = 0;
    clip.stop();
    try
    {
        audioStream = AudioSystem.getAudioInputStream(songs.get(songNumber));
        clip = AudioSystem.getClip();
        clip.open(audioStream);
        clip.start();
        songName.setText(songs.get(songNumber).getName());
    } catch (Exception e)
    {
        out.println(e);
    }
}
}
else if(evt.getSource() == openDirectory) //pops up select a song directory
{
    j = new JFileChooser("D:\\Songs(.wav files)" , FileSystemView.getFileSystemView());
    j.showDialog(mTablePanel,"Play the song");

    song = j.getSelectedFile();
}

```

```

songNumber = songs.indexOf(j.getSelectedFile());
songName.setText(j.getSelectedFile().getName());

try{
    file = new File(String.valueOf(song));
    audioStream = AudioSystem.getAudioInputStream(file);
    clip = AudioSystem.getClip();
    clip.open(audioStream);
    clip.start();

    beginTimer();
} catch (Exception e)
{
    out.println("connected "+e);
}

}
else if(selectedUserListValue.equals("Group Chat")) //if grp chat is selected, it sends a msg
to the server, to activate grp chat
{
    pw.println(tfInput.getText());
}
else{
    // sends private message to server
    initiatePrivateChat(selectedUserListValue);
    pw.println(tfInput.getText());
}
}

public static void main(String[] args) {

    // take username from user
    String name = JOptionPane.showInputDialog(null, "Enter your name: ", "Username",
JOptionPane.PLAIN_MESSAGE);
    String servername = "localhost";
    // String servername = "172.22.8.67";
    try {
        new ChatClient(name, servername);
    } catch (Exception ex) {

```

```

        out.println("Unable to connect to server.\nError: " + ex.getMessage());
    }

} // end of main

public void beginTimer() //begins timer in the progress bar
{
    timer = new Timer();
    songBar.setValue(100);
    task = new TimerTask() {

        public void run() {
            try //in case of division by zero
            {

songBar.setValue((int)(100*clip.getMicrosecondPosition()/clip.getMicrosecondLength()));

songBar.setString((int)(clip.getMicrosecondPosition()/(1e6*60))+":"+((int)((clip.getMicrosecond
Position()/1e6)%60)+" /
"+(int)(clip.getMicrosecondLength()/(1e6*60))+":"+((int)((clip.getMicrosecondLength()/1e6)%6
0)));
                } catch(ArithmeticException e)
                {
                    out.println(e);
                }

            }

        };
    timer.scheduleAtFixedRate(task,1000,1000);

}

public void initiatePrivateChat(String str) //initiate private chat
{
    int temp = str.length()-9; //getSelectedValue returns name with percentage
    str = str.substring(0,temp); //converting 'Kailash : 88.88%' to 'Kailash'

    pw.println("pvtMsg"); //indication that msgs are private

```



```

        pw.println(str);
    }

    @Override
    public void valueChanged(ListSelectionEvent e) { //JList selected value

        if(usersList.isSelectionEmpty())
            selectedUserListValue = "Group Chat";
        else
            selectedUserListValue = usersList.getSelectedValue();

    }

    // inner class for Messages Thread
    class MessagesThread extends Thread {

        @Override
        public void run() {
            String line;
            int usersSize;
            try {

                while(true) {
                    line = br.readLine();
                    if(line.equals("Users List")) //if the incoming data is users list, all the users data
must be displayed only on userlist textarea
                    {
                        listModel.removeAllElements(); //clears the previous list

                        usersSize = Integer.parseInt(br.readLine()); //getting usersSize
                        for(int i=0 ; i<usersSize ; i++)
                        {
                            line = br.readLine(); //getting the name of each user
                            listModel.addElement(line); //adding names to the JList
                        }
                        listModel.addElement("Group Chat"); //adding Group Chat option at the end of
the JList

```

```

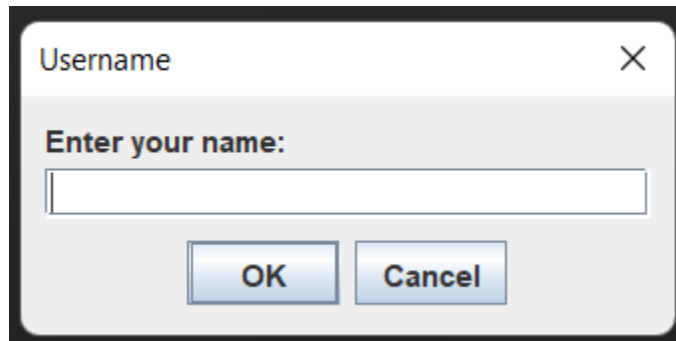
    }
    else {
        taMessages.append(line + "\n");
        taMessages.setCaretPosition(taMessages.getDocument().getLength()); //auto scroll
to last message
    }

    } // end of while
} catch (Exception ex) {
    out.println("nconnected"+ex);
}
}
}
} // end of client

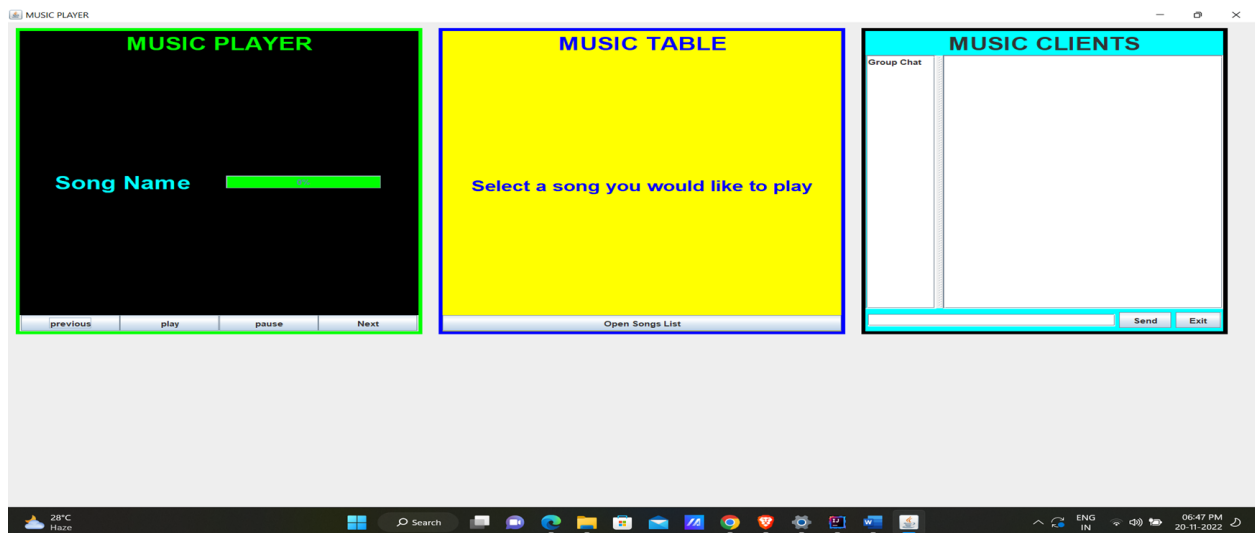
```

OUTPUT SNAPSHOTS

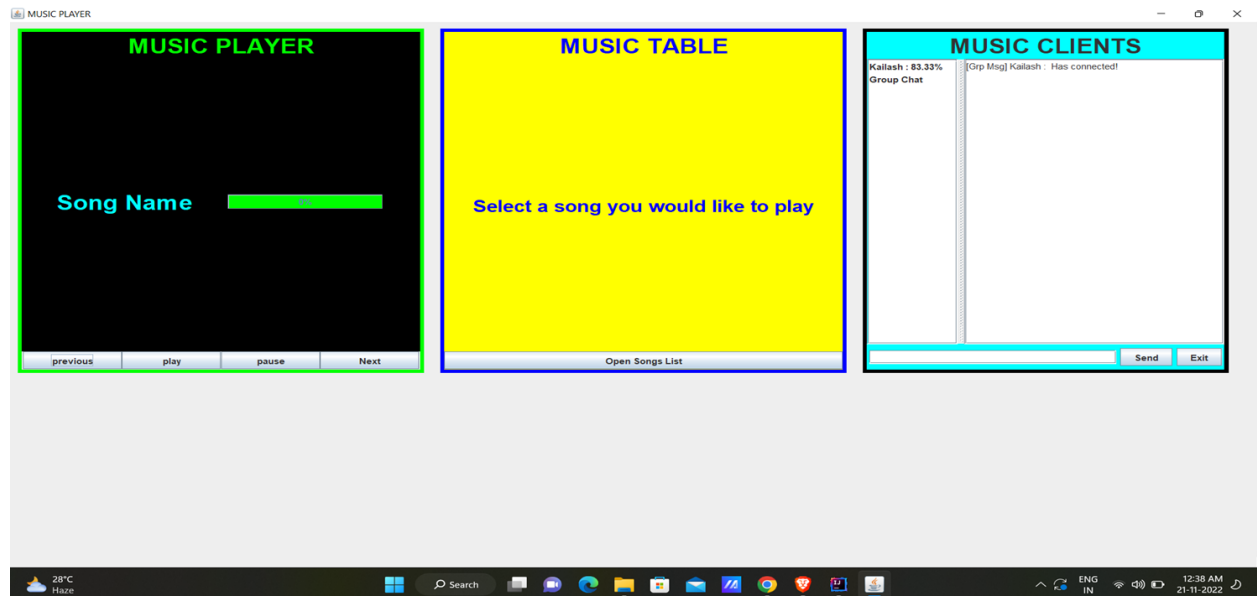
Initial Interface



When the user enters username



When another user gets connected to the network



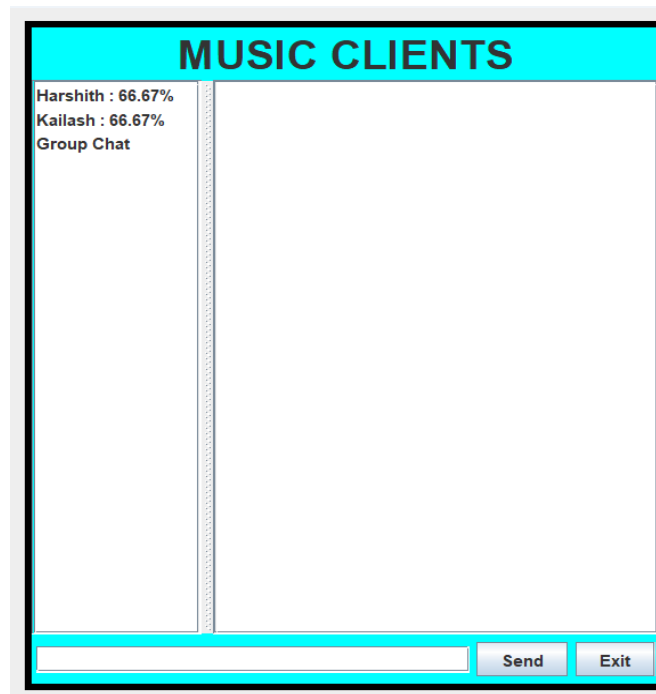
One more user has connected



POV from User2, When User2 and User3 gets connected

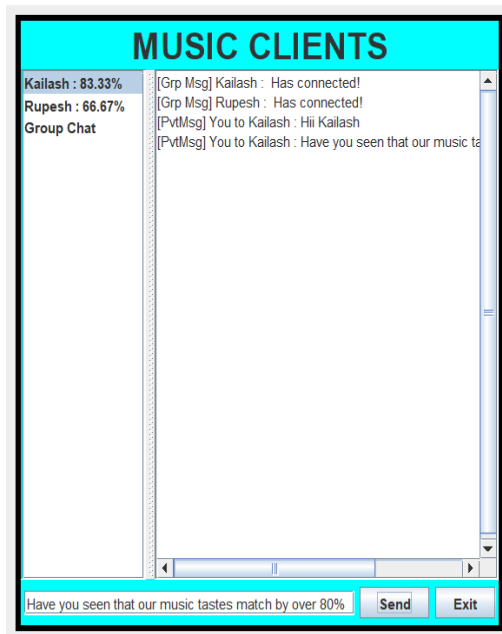


POV from User3, When User3 gets connected

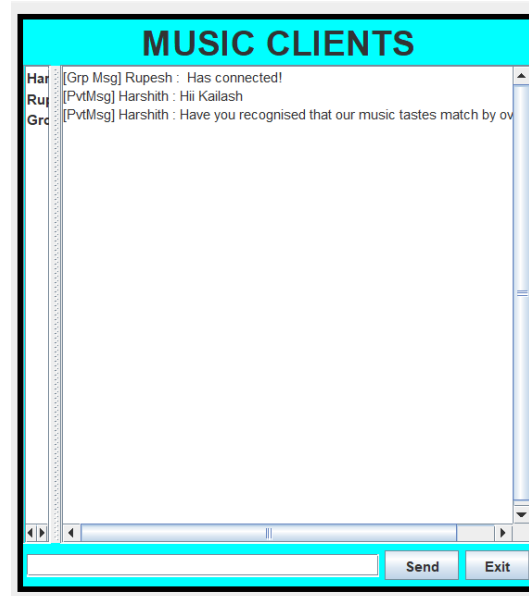


User1 sends a private message to User2

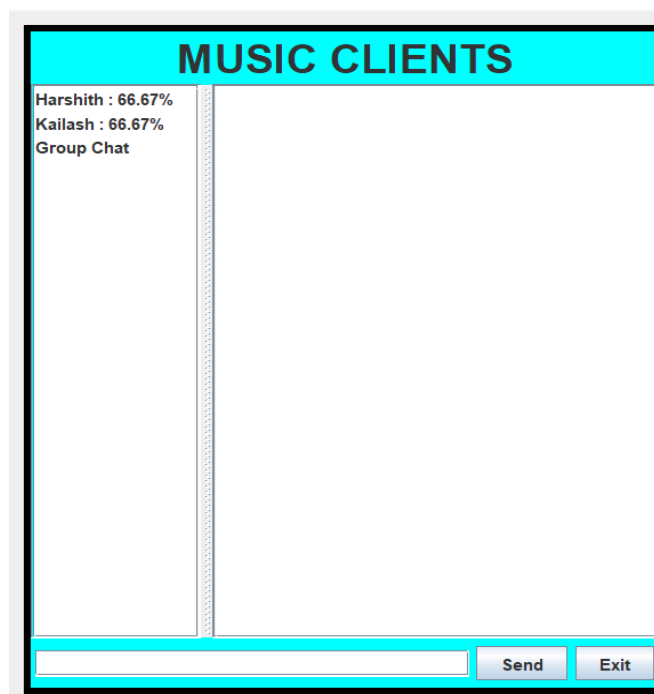
POV: User1(Harshith)



POV: User2(Kailash)

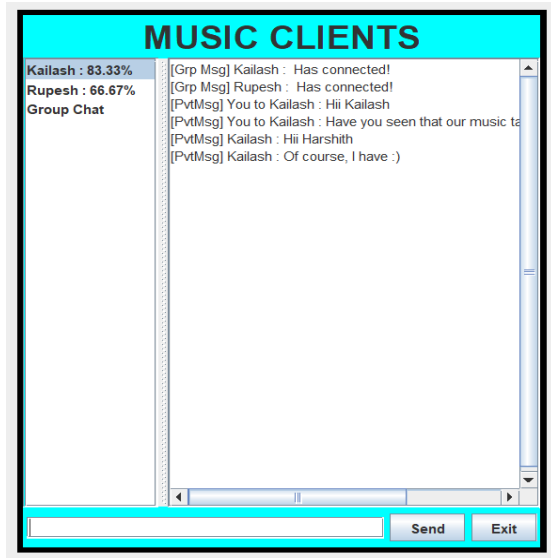


POV: User3(Rupesh)

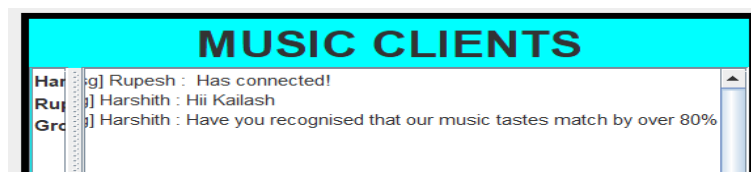
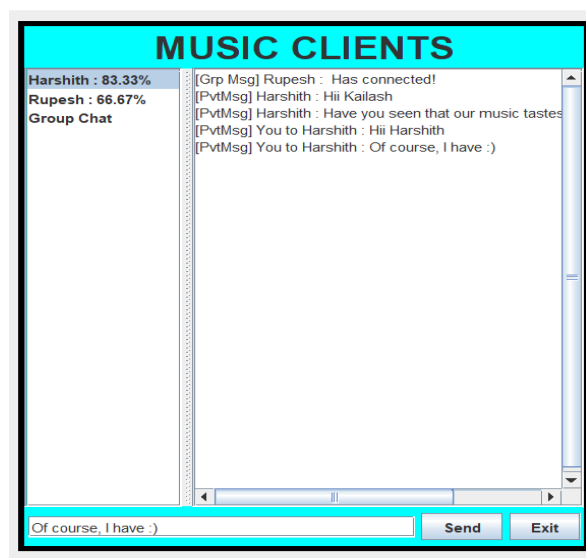


User2 responds to User1's private message

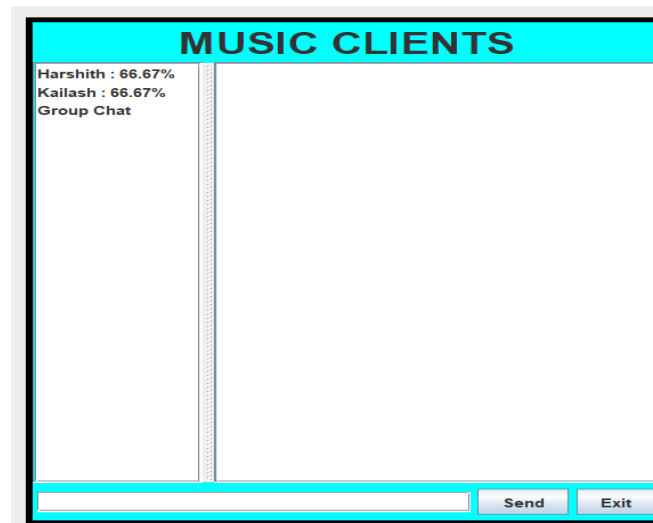
POV: User1(Harshith)



POV: User2(Kailash)

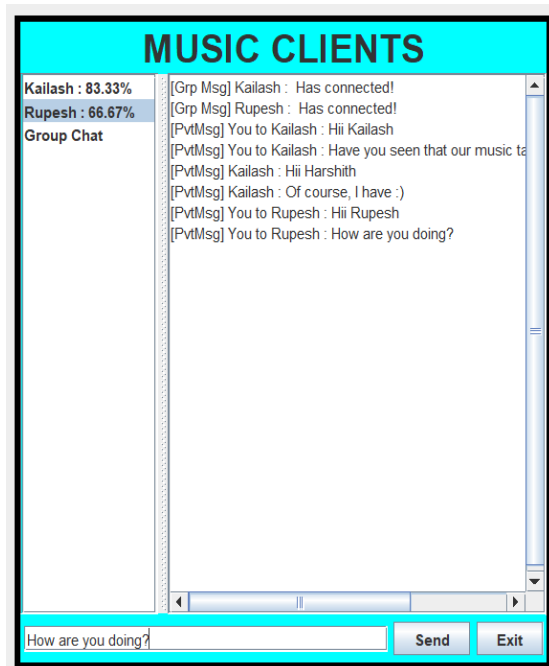


POV: User3(Rupesh)

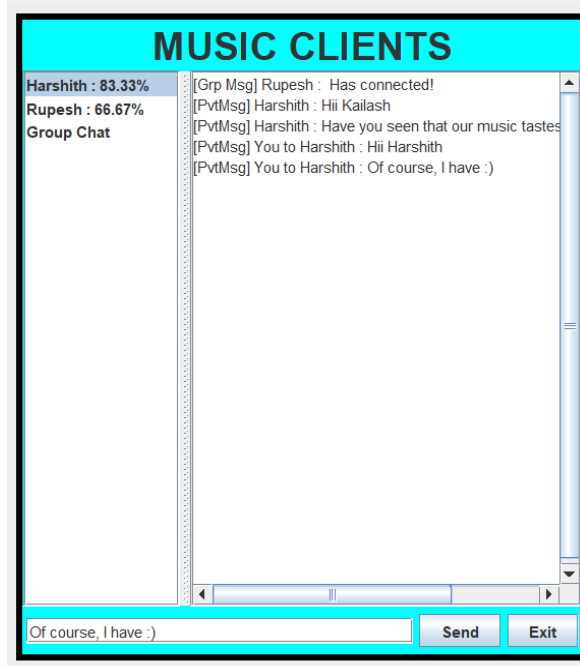


User1 sends a private message to User3

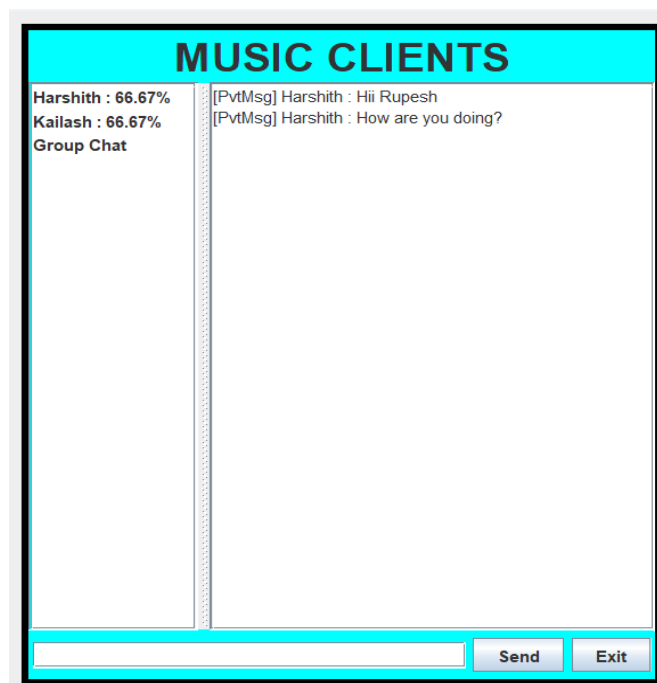
POV: User1(Harshith)



POV: User2(Kailash)

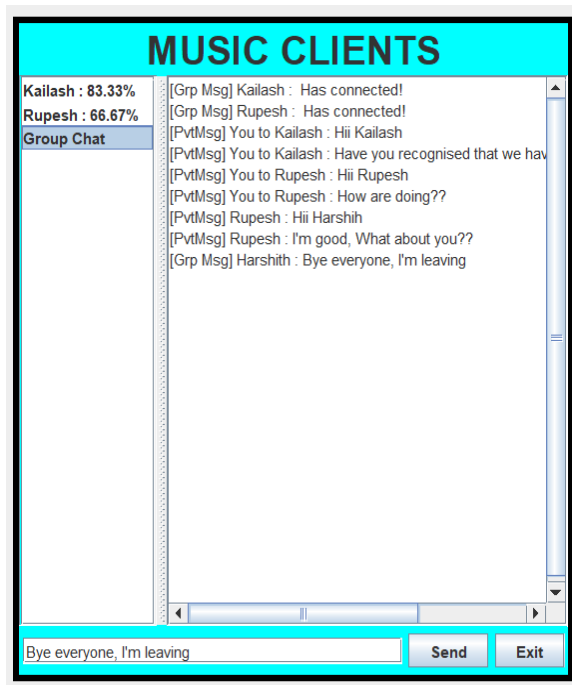


POV: User3(Rupesh)

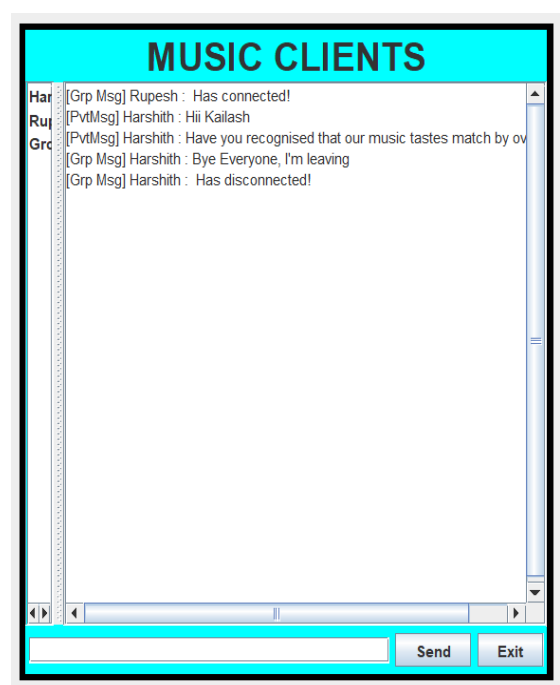


User1 sends a group message to all users

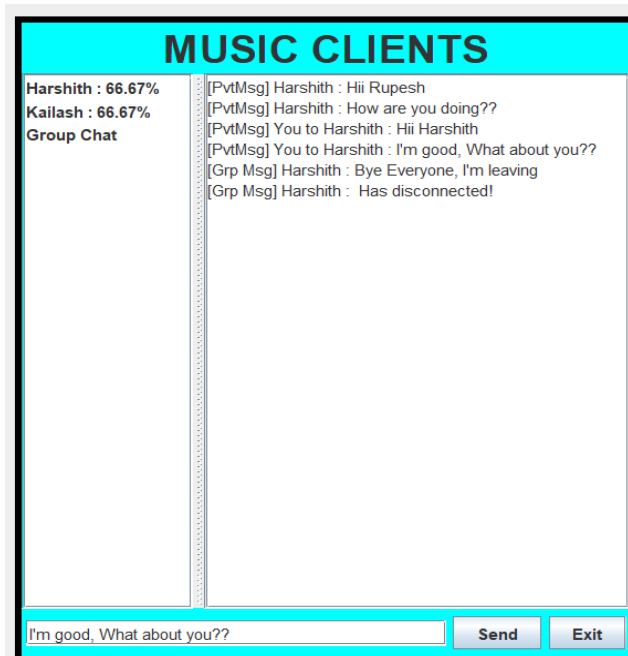
POV: User1(Harshith)



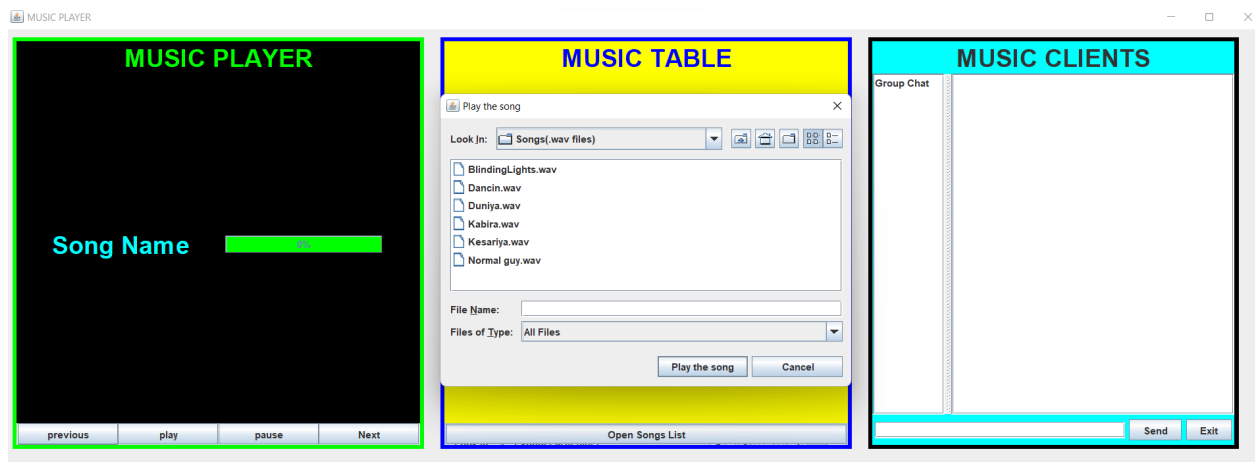
POV: User2(Kailash)



POV: User3(Rupesh)



Choosing a song from the playlist



Playing a song from the playlist



CONCLUSION

A real Java Music Player that connects people with Music tastes using Socket Programming has been developed, which allows a user to play, pause, switch between songs and connect to other users.

Socket Programming made the application connect multiple users to a single server and allowed users to send and receive chats from each other. JRE provided all the necessary features like Swing and awt packages to develop an interactive and elegant interface in java. The Sockets and client-server architecture made the chats more reliable and efficient.

FUTURE WORKS

- This project can be extended by providing a database for the chats of the users
- The GUI can be made more elegant and interactive
- Features like calls and video calls can be inserted into the chat area
- A feature which allows one or many users to listen to the same song at the same time by creating separate rooms in the chat for them, can be implemented

REFERENCES

- TCP/IP Sockets in Java book by Kenneth L. Calvert and Michael J. Donahoo
- Wikipedia
- Geeks for Geeks
- Javatpoint
- StackOverflow