Report for Distributed Systems Application project

Lalagkos Lysandros Konstantinos

l.lalagkos@mc-class.gr

Department of Computing, Computer Science

Contents

Introduction

The application made for the Distributed Systems assessment is a two-part program. The first part is the server side of the application which is a server that listen for clients and accepts them, implementing a primitive distribution architecture in order to keep the load balanced. The server is capable of knowing usernames and passwords for users to log in, asks the users to log in to the application, can check whether a user is already logged in to prevent someone logging in with the same account twice, new users are able to register. Finally made according to the assessment criteria, the server is able to either send a .wav file to the client or stream the .wav file to the client. Last but not least the server is able to show to users what other users listen to using client-server-client communication.

The second part of the program is the client part. The client is able to enter an IP address to connect to the server, then an automated handshake initiates to establish connection. The user is able to manually respond a request services from the server according to the automated explanatory messages the server sends. Lastly the client is able to receive .wav files upon request from the server and either save them to machine or playback the file as it streams in a temporary memory. By having a music player, the client can play saved .wav files directly from the app and the streamed files are also played from the app. The audio player is capable of pausing the song, stopping the song, skip to a specific time of the song. Source code is commented in order to be understandable and most of the variables have specific names according to what are used for.

Server

The server has a number of classes each responsible for various networking or servicing operations. In this part of the report, I will show and explain the parts that the server is made of. The program is able to handle wrong log in attempts, users that exited unexpectedly, problems with sending and receiving files, wrong inputs (not in all cases, will be explained later).

Access Server

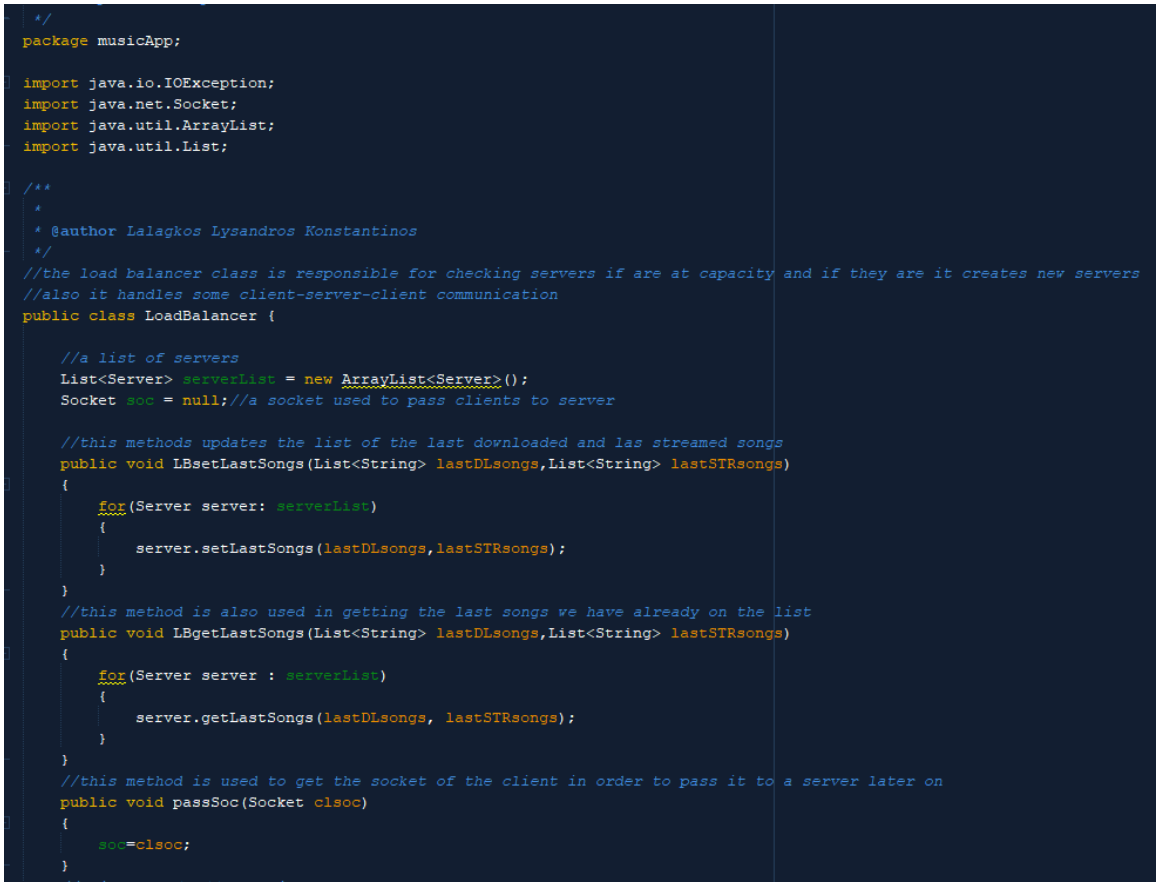
The first file is the file named “AccessServer”, as the name indicates is the server that listens for incoming requests and accepts the requests. The Access Server calls another class that we will see next that is responsible for the balancing of the serving servers. Lastly in order to achieve client-server-client communication even across different servers of the application, in the access server is where the system checks for the last downloaded .wav file from the users and the last streamed file.

```
//otherwise the load balancer is called and if necessery it creates a new server instance to service new clients
public class AccessServer {

    /**
     *
     * @param args
     * @throws IOException
     */
    public static void main(String args[]) throws IOException {
        // TODO code application logic here
        //instance of load balancer class
        LoadBalancer LB = new LoadBalancer();

        int port = 42069;//port number of access server
        //new server socket is created at specified port
        ServerSocket servSoc = new ServerSocket(port);
        //server prints the ip address so it is known in case we run the files of the client in different machine than the server
        System.out.println(servSoc.getInetAddress().getLocalHost().getHostAddress());
        //a new empty socket to connect the client to
        Socket soc2serv = null;
        // message in server console in order to know if it is runnung or not
        System.out.println("Server up and running");
        //this lists are for the "what are other users listen to, they store the last songs downloaded or streamed
        List<String> lastDownloaded = new ArrayList<String>();
        List<String> lastStreamed = new ArrayList<String>();
        //duplicate checker is a class where it if last song list have duplicates in order to remove them
        Check4DuplicateSongs dc = new Check4DuplicateSongs();

        while(true)//this while is for constant listening for client requests
        {

            //the server accepts the connection and pass it to a socket in the server class
            soc2serv = servSoc.accept();
            System.out.println("accepting client");//message to know that a new client has been accepted
            LB.passSoc(soc2serv);//load balancer class calls its pass socket method in order to pass it to the appropriate server
            System.out.println("sending client to server");//message to let us know that the client has been passed in to a server
            //the load balancer class calls its balancer servers method in order to check if the servers are full or not and if needed to create a new server
            LB.balanceServers();
            System.out.println("client sent to appropriate server");

            LB.LBsetLastSongs(lastDownloaded, lastStreamed);//the load balancer tracks and stores the last songs that where streamed or downloaded
            lastDownloaded = dc.checkListForDuplicates(lastDownloaded);//list checked for duplicates
            lastStreamed = dc.checkListForDuplicates(lastStreamed);//list checked for duplicates
            LB.LBgetLastSongs(lastDownloaded, lastStreamed);//list updated


        }
```

Picture 1. The “AccessServer” file

Load Balancer

The next file on the server side is the “LoadBalancer” file. Load balancer as the name suggests is mainly responsible for balancing how many clients are in each server and if it is needed it creates a new instance of the server to serve new clients. It is made in a way that theoretically it can expand for ever as needed if the system resources are capable of handling such a load. Lastly it is able to remove inactive clients, meaning clients that have not taken any action for more than 15 minutes are removed from the server. Also some operations happen for keeping updated the “what are users listen to” service. The load balancer every 5 clients that are on e single server will created a new server to accumulate new clients, in the case of clients leaving the previously full server the load balancer is aware and if a new client request is accepted, the client will be put in the vacant spot on the previously full server. In case of server being empty then the load balancer will delete the instance of the server that does not serve any clients.



```
 */
package musicApp;

import java.io.IOException;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author Lalagkos Lysandros Konstantinos
 */
//the load balancer class is responsible for checking servers if are at capacity and if they are it creates new servers
//also it handles some client-server-client communication
public class LoadBalancer {

    //a list of servers
    List<Server> serverList = new ArrayList<Server>();
    Socket soc = null;//a socket used to pass clients to server

    //this methods updates the list of the last downloaded and las streamed songs
    public void LBsetLastSongs(List<String> lastDLsongs,List<String> lastSTRsongs)
    {
        for(Server server: serverList)
        {
            server.setLastSongs(lastDLsongs,lastSTRsongs);
        }
    }
    //this method is also used in getting the last songs we have already on the list
    public void LBgetLastSongs(List<String> lastDLsongs,List<String> lastSTRsongs)
    {
        for(Server server : serverList)
        {
            server.getLastSongs(lastDLsongs, lastSTRsongs);
        }
    }
    //this method is used to get the socket of the client in order to pass it to a server later on
    public void passSoc(Socket clsoc)
    {
        soc=clsoc;
    }
```

Picture 2. Part 1. Of the “LoadBalancer” File

```
//this method adds a client to a server
public void AddClient2Server(Server server)
{

    server.addClient2handler(soc);

}
//this method is used to create new servers when called
public Server CreateServer()
{
    Server servingServer = new Server();
    return servingServer;
}
//this method contains the balancing logic for the servers
public void balanceServers() throws IOException
{
    if(serverList.isEmpty())//if there is no server i create a new one
    {

        System.out.println("first instance of serving Server is created");
        serverList.add(CreateServer());
    }
    int fullServers=0;
    for (Server i: serverList)//cannot modify while in foreach
    {

        i.RemoveInactiveClientHandlers();//this method is from the class server and it removes inactive clients
        i.RemoveExitedClients();//this method also from class server removes clients that have chose to exit the app

        if(i.isAvailable())//if server is available then add client to available server
        {
            AddClient2Server(i);
            System.out.println("adding client to serving server "+serverList.indexOf(i) +" " +soc);
        }
        else if (i.isFull())
        {
            fullServers++;


        }

    }
```

Picture 2. Part 2. Of the “LoadBalancer” file.

```
if(fullServers==serverList.size())//if all servers existing are full then create a new server and add client there
{
            serverList.add(CreateServer());
            AddClient2Server(serverList.get(serverList.size()-1));
            System.out.println("servers are full, creating new server to add client");
            System.out.println("adding client to serving server "+serverList.indexOf(serverList.get(serverList.size()-1)) +" " +soc);
}
```

Picture 3. Part 3. Of the “LoadBalancer” file.

Server

For the next part of the server side of the application is the “Server” file. The “Server” file is responsible for serving clients by opening a new thread each time a new client requests service. This is the “Server” that is created from the load balancer whenever is deemed necessary, for testing purposes I made the number of clients that each server can handle simultaneously 5. The “Server” checks itself if has available spots for new clients or if it full, it removes inactive clients, it removes clients that requested to exit the application, and last but not least is handling the last downloaded/streamed songs of the clients.

```java
package musicApp;

import java.io.IOException;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author Lalagkos Lysandros Konstantinos
 */

//this class is handling the connection of the clients and is the server that when gets a client it will open a thread to service him
public class Server {

    //list of clienthandler class/threads meaning a list of clients
    List<ClientHandler> clHandList = new ArrayList<ClientHandler>();

    //this method creates a new thread client handler and adds a client there in order to serve them the app
    public void addClient2handler(Socket soc)
    {
        ClientHandler clHand = new ClientHandler();

        clHand.addclient(soc);
        clHand.start();
        clHandList.add(clHand);

    }
```

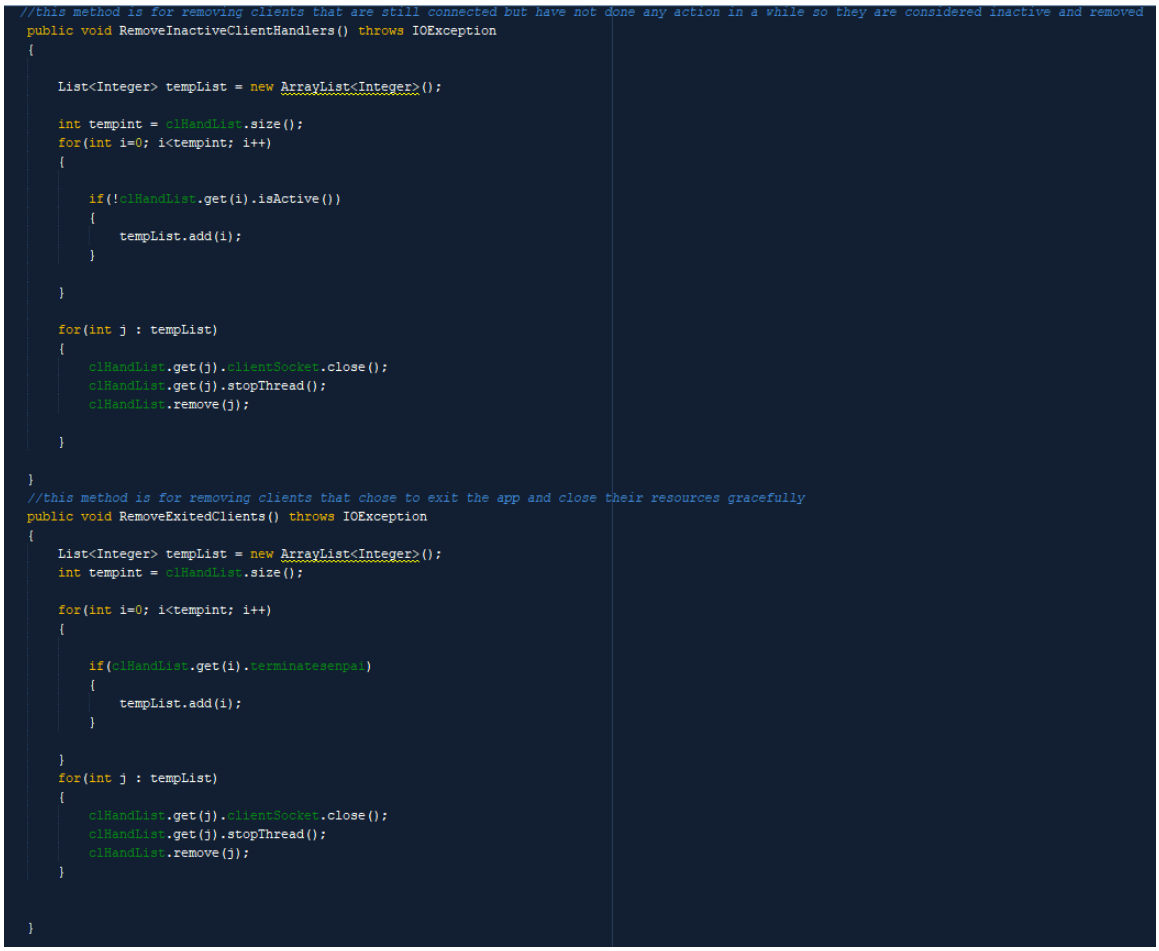
Picture 4. Part 1. “Server” file

```java
//this method returns a boolean true if a server is full or false if it can accept clients
public boolean isFull()
{
    boolean full=false;
    int maxClients = 5;//this number represents the max number of clients a server can have, it can be changed, it is five for testing purposes, depending on the machine i
    if(clHandList.size()==maxClients)
    {
        full=true;
    }
    else
    {
        full=false;
    }
    return full;

}
//this method returns a boolean if a server is available to accept another client
public boolean isAvailable()
{
    if(!isFull())
    {
        return true;
    }
    else
    {
        return false;

    }

}
//this method connects to client handler in order to get the last song streamed or downloaded and return to the loadbalance to achieve client server client communication
```

Picture 5. Part 2. “Server” file

```java
//this method connects to client handler in order to get the last song streamed or downloaded and return to the loadbalance to achieve client server client communication
public void getLastSongs(List<String> lastDLsongs,List<String> lastSTRsongs)
{
    for(ClientHandler ch : clHandList)
    {
        for(String dl : lastDLsongs)
        {

            ch.otherUsersLastDownloaded.add(dl);
        }
        ch.otherUsersLastDownloaded.clear();

        for(String str : lastSTRsongs)
        {

            ch.otherUsersLastStreamed.add(str);
        }
        ch.otherUsersLastStreamed.clear();
    }
}
//like the above method but this time for adding songs to the list and not retrieving
public void setLastSongs(List<String> lastDLsongs,List<String> lastSTRsongs)
{
    for(ClientHandler ch : clHandList)
    {
        lastDLsongs.add(ch.lastDownloaded);
        lastSTRsongs.add(ch.lastStreamed);
    }
}
```

Picture 6. Part 3. “Server” file



```
//this method is for removing clients that are still connected but have not done any action in a while so they are considered inactive and removed
public void RemoveInactiveClientHandlers() throws IOException
{

    List<Integer> tempList = new ArrayList<Integer>();

    int tempint = clHandList.size();
    for(int i=0; i<tempint; i++)
    {

        if(!clHandList.get(i).isActive())
        {
            tempList.add(i);
        }

    }

    for(int j : tempList)
    {
        clHandList.get(j).clientSocket.close();
        clHandList.get(j).stopThread();
        clHandList.remove(j);

    }

}
//this method is for removing clients that chose to exit the app and close their resources gracefully
public void RemoveExitedClients() throws IOException
{
    List<Integer> tempList = new ArrayList<Integer>();
    int tempint = clHandList.size();

    for(int i=0; i<tempint; i++)
    {

        if(clHandList.get(i).terminatesenpai)
        {
            tempList.add(i);
        }

    }
    for(int j : tempList)
    {
        clHandList.get(j).clientSocket.close();
        clHandList.get(j).stopThread();
        clHandList.remove(j);
    }


}
```

Picture 7. Part 4. “Server” file

Client Handler

The “ClientHandler” file is responsible for handling requests for services from the client, is has a menu that the client can interact with, it does and responds to handshakes such as connection handshake, file send/receive handshake, disconnect handshake. It uses files that will see later for different actions such as sending and receiving files and messages and it handles the user registration or log in, it shows to the client the available songs list either for downloading or streaming, it is capable of sending the .wav files to the client with custom packets of 1024 bytes, it handles the exit request of the client, it has a custom search by term for songs in order the client to search a word or phrase for a song and finally it handles the “what other users listen to” request.

```java
package musicApp;

import java.io.IOException;
import java.net.Socket;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

/**
 *
 * @author Lalagkos Lysandros Konstantinos
 */

//this class/ thread is used to serv the connected client, each client has his own handler in order to serve all clients in the same time
public class ClientHandler extends Thread {

    Socket clientSocket=null;
    ServerCommunicator com;
    boolean ClientHandlerRunCondition = true;
    public boolean terminatesenpai=false;

    String lastDownloaded = "";
    String lastStreamed = "";

    List<String> otherUsersLastDownloaded = new ArrayList<>();
    List<String> otherUsersLastStreamed = new ArrayList<>();

    //this method is to aquire the socket of the client
    public void addclient(Socket soc)
    {
        clientSocket=soc;
    }
    //this method is to connect the communicator class to the client streams
    public void initcom() throws IOException
    {
        com = new ServerCommunicator(clientSocket);
    }
```

Picture 8. Part 1. “ClientHandler” file

```java
//this method returns a boolean, it checks if a client is active or has not taken an action for more than 15 minutes
public boolean isActive() throws IOException
{
   boolean active = true;

   if((System.currentTimeMillis()-com.lastAction.getTime())>(15*60*1000)) //time of server not reading anything from client 15mins
   {
       System.out.println("client timed out "+clientSocket);
       com.closeCommunicator();
       active = false;
   }

   return active;
}

//in order to stop the client handler i change the condition of the loop so it will exit the thread when reaches the end according to the documents by oracle for java threads
public void stopThread()
{
    ClientHandlerRunCondition = false;
}
```

Picture 9. Part 2. “ClientHandler” file

```java
//the thread runs here
@Override
public void run()
{

    boolean runningCondition;
    HashMap<String,String> userpass = new HashMap<String,String>();
    HashMap<String,String> newEntries = new HashMap<String,String>();
    FileManipulation fileManip =new FileManipulation();
    String w8="w8";
    boolean loggedin=false;
    List<String> songlist = new ArrayList<String>();
    String choice="";
    boolean filemanipulation = false;
    String pathname="";
    String username="";
    fileManip.removeLoggedInUser("=empty=");



    try
    {
       initcom();//initalize communication

       //System.out.println("i started trying");
       runningCondition = com.initialHandshake();//handshake is done here
       //System.out.println("finished HS");
       int counter=0;

       com.sendMessage("Welcome to THE MUSIC APP (tm Distributed Systems) \n "
                    + "Press the number of the choice you want and press enter to validate");//message for client

       fileManip.ReadFromFile("userpass.txt", userpass);
```

Picture 10. Part 3. “ClientHandler” file

```java
while(runningCondition&&ClientHandlerRunCondition)//this loop keeps the thread alive and runs for as long the conditions are met
{

    if(!loggedin)//if client not logged in we prompt him to log in or register if he does not have an account
    {
        com.sendMessage("1.Register\n"+"2.Log In\n"+"0.Exit"+w8);//menu for client
        choice = com.recieveMessage();
        switch(choice)
        {
            case "1":
            {
                com.sendMessage("please give a username longer than 4 characters"+w8);
                username = com.recieveMessage();
                if(username.length()<=4)
                {
                    com.sendMessage("username too short, try another username");//check if username standards are met
                }
                else if(userpass.containsKey(username)||newEntries.containsKey(username))//check if username already exists
                {
                    com.sendMessage("username already exists");
                }
                else
                {
                    com.sendMessage("username accepted \n Please give a password, longer than 4 characters"+w8);//accepting username, waiting for password
                    String password = com.recieveMessage();
                    if(password.length()<=4)//check if password conditions are met
                    {
                        com.sendMessage("invalid password, too short");
                    }
                    else
                    {
                        newEntries.put(username, password);//creating new user, using hashmap we keep track of usernames and passwords
                        com.sendMessage("Account Created");
                    }
                }

                break;
            }
```

Picture 11. Part 4. “ClientHandler” file

```
case "2"://if account exists and client wants to log in
{
    com.sendMessage("Write your Username :"+w8);
    username = com.recieveMessage();

    if(fileManip.isUserAlreadyLoggedIn(username))//check if user is already logged in
    {
        com.sendMessage("User is already logged in");
        break;
    }


    if(userpass.containsKey(username)||newEntries.containsKey(username))//check if username exists
    {
        com.sendMessage("username correct\n insert password"+w8);
        String password = com.recieveMessage();
        if(password.equals(userpass.get(username))||password.equals(newEntries.get(username)))//check if password matches username
        {
            com.sendMessage("correct password\n Welcome "+username);
            loggedin=true;
            //next menu switch case starts here or use booleans to if() and there the switch if true
        }
        else
        {
            com.sendMessage("wrong password");
        }
    }
    else
    {
        com.sendMessage("username does not exist");
    }

    break;
}
```

Picture 12. Part 5. “ClientHandler” file

```
case "0"://if clients chose to exit
{

    com.sendMessage("If you really want to exit type EXIT else type no"+w8);//check if client pressed exit by mistake

    if(com.terminationHandshake())
    {

        terminatesenpai=true;//flag to let the server know that client wants to exit
        runningCondition = false;//kill thread by stoping the loop
        com.closeCommunicator();//close streams of communication
    }

    break;
}
default :
    if(com.SocExep)//if something wrong with the socket close communication
     {
         runningCondition = false;

     }


    com.sendMessage("invalid choice, please try again");
    break;
```

Picture 13. Part 5. “ClientHandler” file

```
if(loggedin)//if the user is logged in
{

    //menu for user to chose from
    com.sendMessage("1.List of Song \n 2.Search \n 3.other users listen to\n 4.Listen to Downloaded songs \n"+"0.EXIT"+w8);


    choice=com.recieveMessage();
    switch(choice)
    {
        case "1"://show list of songs to client to chose action
        {
            //read txt list, show to client

            fileManip.ReadFromFile("list of songs.txt", songlist);//read and show available songs to client

            for(String i: songlist)
            {
                com.sendMessage(songlist.indexOf(i)+". "+i);
            }
            //at this point we wain for a response from the client
            com.sendMessage("end of list"+w8);
            choice = com.recieveMessage();
            while(Integer.parseInt(choice)<0||Integer.parseInt(choice)>=songlist.size())//check client input
            {
                com.sendMessage("invalid choice, please select again"+w8);
                choice = com.recieveMessage();

                 if(com.SocExep)//if something wrong with socket close communication
                 {
                     runningCondition = false;
                     break;
                 }
            }
            pathname = songlist.get(Integer.parseInt(choice));
            filemanipulation =true;//flag to prompt new menu for client



            songlist.clear();
            break;
        }
```

Picture 14. Part 6. “ClientHandler” file

```
case "2"://custom search for term in order to show all entries that contain term
{
    fileManip.ReadFromFile("list of songs.txt", songlist);

    List<String> searchResults = new ArrayList<String>();
    com.sendMessage("enter a search term"+w8);//waiting for client response
    choice=com.recieveMessage();
    choice=choice.toLowerCase();
    for(String i: songlist)
    {

        String comparator=i.toLowerCase();
        if(comparator.contains(choice))
        {
            searchResults.add(i);
        }
    }
    if(searchResults.isEmpty())//if no result tell client
    {
        com.sendMessage("no results found");

    }
    else
    {
        for(String i: searchResults)//show list of results to client for the searched term
        {
            com.sendMessage(searchResults.indexOf(i)+". "+i);
        }
        com.sendMessage("end of list"+w8);
        choice = com.recieveMessage();
        while(Integer.parseInt(choice)<0||Integer.parseInt(choice)>=searchResults.size())
        {
             com.sendMessage("invalid choice, please select again"+w8);
             choice = com.recieveMessage();

             if(com.SocExep)//again handling socket exceptions
             {
                 runningCondition = false;
                 break;
             }

        }
        pathname = searchResults.get(Integer.parseInt(choice));
        filemanipulation =true;
    }

    songlist.clear();
    break;
```

Picture 15. Part 7. “ClientHandler” file

```
case "3"://what other users listen to
{

    com.sendMessage("Other users recently downloaded :");
    for(String dl : otherUsersLastDownloaded)//list of the last song each connected user has downloaded
    {
        com.sendMessage(dl);
    }

    com.sendMessage("Other users recently streamed :");//list of the last song each connected user has streamed
    for(String str : otherUsersLastStreamed)
    {
        com.sendMessage(str);
    }

    break;
}
case "0"://again exit confirmation
{
    com.sendMessage("If you really want to exit type EXIT else type no"+w8);

    if(com.terminationHandshake())
    {
        terminatesenpai=true;
        runningCondition = false;
        com.closeCommunicator();
    }
    break;
}
case "4"://client chose to listen to already downloaded songs so server does nothing and resends the menu to client
    break;
default:

     if(com.SocExep)//again socket exception handling
     {
         runningCondition = false;
     }

    System.out.println("CHOICE :  "  + choice);
    com.sendMessage("invalid choice");
    break;
```

Picture 16. Part 8 “ClientHandler” file

```
else
{
    com.sendMessage("If you dont have an account please Register \n"+"otherwise log in correctly \n");
}

if(filemanipulation)//if client chose to recieve or stream a file
{

    String fileReceived = "";
    com.sendMessage("1. download file \n"+"2. Stream file \n"+w8);//menu of choices for client
    choice = com.recieveMessage();
    switch(choice)
    {
        case "1"://client chose to download a file
        {
            //download
            com.sendMessage("sending file" + pathname+".wav");//a handshake in order to begin transmition
            com.SendFile("songs/"+pathname+".wav");//may work may not, it is what it is

            lastDownloaded = pathname;


            while(true)
            {
                fileReceived = com.recieveMessage();

                if(fileReceived.equals("OK"))//waiting for ok from client
                {
                    break;
                }
                else
                {
                    com.sendMessage("Operation will continue only if server receives OK"+w8);//tell client to send ok after transmition is finished
                    //fileReceived = com.recieveMessage();
                }

                if(com.SocExep)
                {
                    runningCondition = false;
                    break;
                }

             }

            break;
        }
```

Picture 17. Part 9 “ClientHandler” file

```
case "2"://client chose to stream a song
{
    //stream
    com.sendMessage("Starting Stream");
    com.SendFile("songs/"+pathname+".wav");//sending the file

    lastStreamed = pathname;

    while(true)
    {

        try
        {
            fileReceived = com.recieveMessage();
        }
        catch(Exception e)
        {
            break;
        }



        if(fileReceived.equals("OK"))//again waiting for ok from client
        {
            break;
        }
        else
        {
            com.sendMessage("Operation will continue only if server receives OK");
        }

        if(com.SocExep)
        {
            runningCondition = false;
            break;
        }

    }


    break;
}
```

Picture 18. Part 10 “ClientHandler” file

```
                    default :
                        com.sendMessage("invalid input");
                        break;

                }
                choice = "";
                filemanipulation = false;
            }



            counter++;

        }


    }
    catch(Exception e)
    {
        System.out.println(e.toString());
    }
    finally
    {
        if(!username.equals(""))
        {
            fileManip.removeLoggedInUser(username);
            System.out.println("User " + username + " logged out");
        }

        fileManip.Write2File("userpass.txt", newEntries);
    }
```

Picture 19. Part 11 “ClientHandler” file

Server Communicator

Server Communicator file is the file responsible for sending and receiving messages, sending files, handshakes and keeping track of the time of the last action made by the client in order if it is more than 15 minutes to remove client and close connection. This file is mainly used by the “ClientHandler” file. Mutual exclusion or mutex is used for the actions needed in order for the communication with the client to be smooth and everything to happen in order.

```java
/**
 *
 * @author Lalagkos Lysandros Konstantinos
 */

//this class is made to handle communication between client and server, it also has all the handshakes used for smooth communication
//every method is mutexed in order to achieve smooth communication
//files are send in the form of packets and buffers are used for this, the packet size is 1024
//also this class sends files to client
public class ServerCommunicator {


    String input = "";
    String output = "";
    DataInputStream dis ;
    DataOutputStream dos;
    BufferedInputStream bis;
    BufferedOutputStream bos;
    FileInputStream fis;
    FileOutputStream fos;
    Timestamp lastAction ;//= new Timestamp(System.currentTimeMillis());
    ReentrantLock lock = new ReentrantLock(true);//this is the way i chose to do mutex
    String pathName;
    Integer endoffile = new Integer(129);
    byte[] EndOfFile = new byte[1024];
    boolean SocExep = false;


    //constractor
    public ServerCommunicator(Socket comSoc) throws IOException
    {
        this.dis = new DataInputStream(comSoc.getInputStream());
        this.dos = new DataOutputStream(comSoc.getOutputStream());
        comSoc.setSoTimeout(2*60*1000);


        for(int i = 0; i< EndOfFile.length;i++)
        {
            EndOfFile[i] = endoffile.byteValue();
        }

    }
```

Picture 20. Part 1. “ServerCommunicator”

```java
//method for closing data strems
public void closeCommunicator() throws IOException
{
    //flag for exception and close from handler
    if(lock.isHeldByCurrentThread())
    {
        lock.unlock();
    }
    dis.close();
    dos.close();
}

//method for handshake to establish communication with client, it returns true if handshake successfull
public boolean initialHandshake() throws IOException
{
    //System.out.println("i should HS now");
    input = recieveMessage();
    //System.out.println("i rcvd : "+input);
    if(input.equals("HELLO SERVER"))
    {
        output="HELLO CLIENT";
        sendMessage(output);
        //System.out.println("i sent : "+output);
        input = recieveMessage();
        //System.out.println("i rcvd : "+input);
        if(input.equals("OK SERVER"))
        {
            System.out.println("Handshake is done");
            return true;

        }
        else
        {
            return false;
        }

    }
    else
    {
        return false;
    }
}
```

Picture 21. Part 2. “ServerCommunicator”

```java
//this method sends a msg to the client
public void sendMessage(String toSend) throws IOException
{

    if(!lock.isHeldByCurrentThread())
    {
        lock.lock();

        try
        {

            dos.writeUTF(toSend);
            System.out.println("com wrote : " + toSend);
            dos.flush();
            System.out.println("flushed");
        }
        catch(SocketException se)
        {
            SocExep = true;
            dis.close();
            dos.close();
            System.out.println("Communicator-> connection terminated unexpectedly");
            //close stuff
        }
        catch(Exception e)
        {
            System.out.println("communicator -> sendMessage failed");
            System.out.println(e.toString());
        }
        finally
        {
            lock.unlock();
        }
    }
```

Picture 22. Part 3. “ServerCommunicator” file

```java
//this method is responsible for recieving messages from the client
public String recieveMessage() throws IOException
{
    //System.out.println("check if lock is locked");
    if(!lock.isHeldByCurrentThread())
    {
        //System.out.println("lock is free ill take it");
        lock.lock();

        try
        {
            //System.out.println("im supposed to rcv");
            output = dis.readUTF();
            setTime();
            System.out.println("i got from client :"+output);

            return output;
        }
        catch(SocketException se)
        {
            SocExep = true;
            dis.close();
            dos.close();
            return "";
        }
        catch(Exception e)
        {
            System.out.println(e.toString());
            return "";
        }
        finally
        {
            lock.unlock();
        }
    }
    else
    {
        throw new IOException("problem with rcving messages");
    }
```

Picture 23. Part 4. “ServerCommunicator” file

```java
//method sets the time that the client last did something in order to keep track if active or not
public void setTime()
{
    lastAction = new Timestamp(System.currentTimeMillis());
}

//handshake for sending file and getting client in a state to recieve the file, if client ready then returns true
public boolean fileHandshake() throws IOException
{

    output="SEND FILE";
    sendMessage(output);
    input=recieveMessage();

    if(input.equals("FILE OK"))
    {
        sendMessage("SENDING FILE");
        return true;
    }
    else
    {
        return false;
    }
}


//method/handshake to terminate client after clients request, returns boolean in order to close resources gracefully
public boolean terminationHandshake() throws IOException
{
    input = recieveMessage();
    if(input.equals("EXIT"))
    {
        sendMessage("OK EXIT");
        input=recieveMessage();
        if(input.equals("OK BYE"))
        {
            return true;
        }
        else
        {
            return false;
        }

    }
    else
    {
        return false;
    }
}
```

Picture 24. Part 5. “ServerCommunicator” file

```java
//this method is responsible for sending a file to the client, using bytes made it able to send any type of file
public void SendFile(String pathName) throws FileNotFoundException, IOException
{

    if(!lock.isHeldByCurrentThread())//mutex
    {

        ObjectOutputStream oos;

        lock.lock();

        try
        {
            File file2send = new File(pathName);
            System.out.println(file2send.length());



            long fileSize = file2send.length();
            dos.writeLong(fileSize);
            dos.flush();

            byte[] bytearray = new byte[1024];
            fis = new FileInputStream(file2send);

            bos = new BufferedOutputStream(dos);
            int counter;

            while((counter=fis.read(bytearray))>0)
            {

                System.out.println(counter);

                bos.write(bytearray, 0, counter);
            }

            bos.flush();
            fis.close();

        }
```

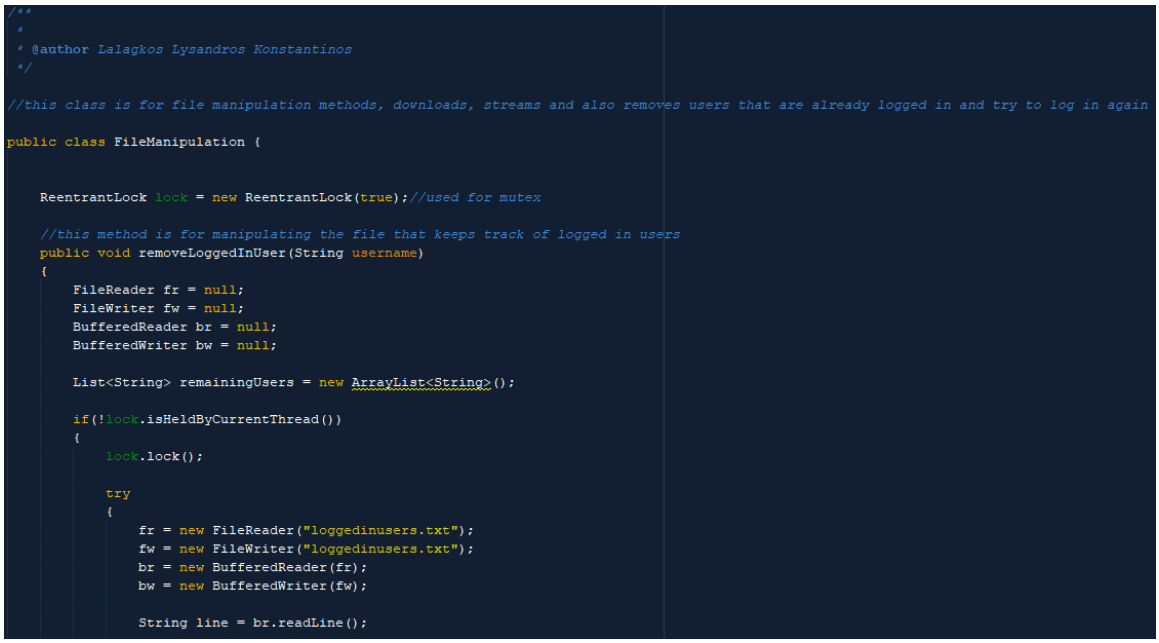
Picture 25. Part 5 “ServerCommunicator” file

```java
    catch(SocketException se)
    {
        SocExep = true;
        dis.close();
        dos.close();
    }
    catch(Exception e)
    {
        System.out.println("communicator -> sendfile");
        System.out.println(e.toString());
    }
    finally
    {
        lock.unlock();
    }

}
else
{
    throw new IOException("problem with sending file");
}
```

Picture 26. Part 6. “ServerCommunicator” file

File Manipulation

The next file is called “FileManipulation” and as the name indicates is for doing various manipulations to files. Specifically, this file is made for reading from a file, writing to a file and handling the file of the logged in users and if someone already logged in tries to log in again with the same account it rejects the request from the “ClientHandler” file. The file uses mutex where needed.

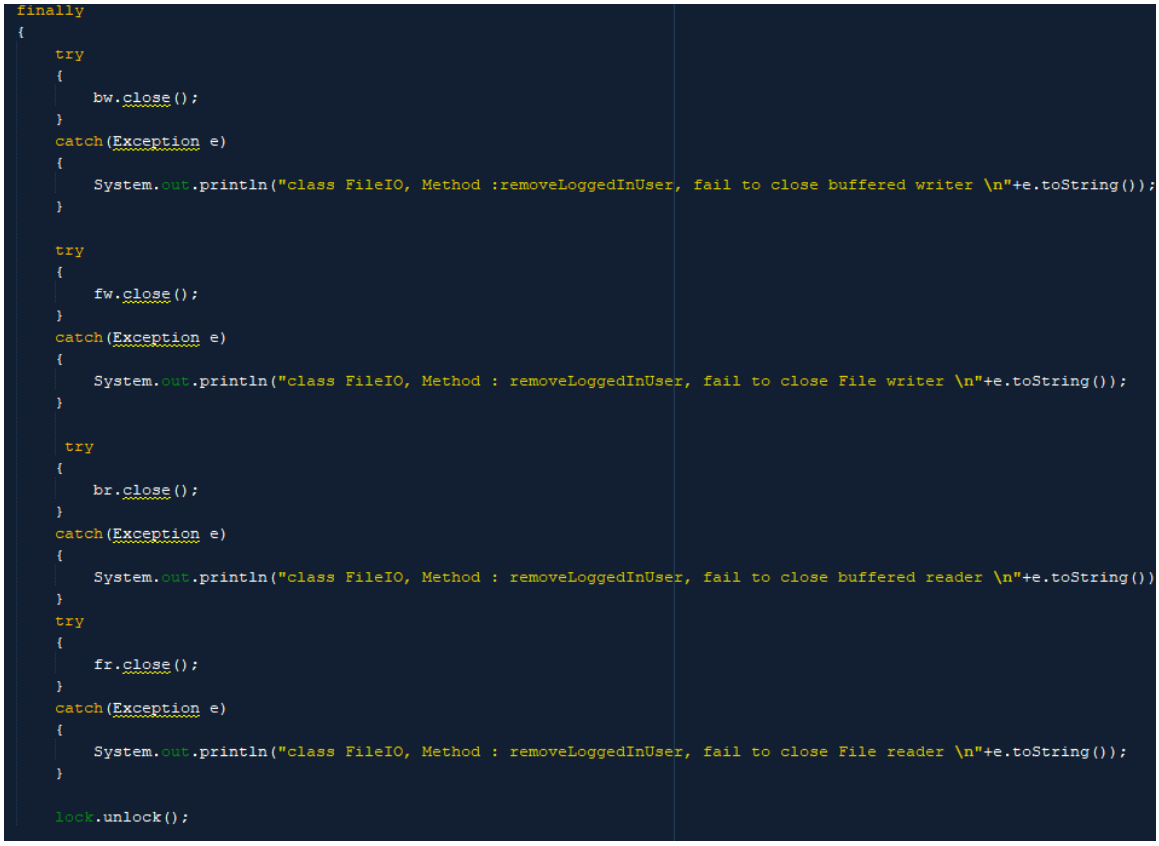


```
/**
 *
 * @author Lalagkos Lysandros Konstantinos
 */

//this class is for file manipulation methods, downloads, streams and also removes users that are already logged in and try to log in again

public class FileManipulation {


    ReentrantLock lock = new ReentrantLock(true);//used for mutex

    //this method is for manipulating the file that keeps track of logged in users
    public void removeLoggedInUser(String username)
    {
        FileReader fr = null;
        FileWriter fw = null;
        BufferedReader br = null;
        BufferedWriter bw = null;

        List<String> remainingUsers = new ArrayList<String>();

        if(!lock.isHeldByCurrentThread())
        {
            lock.lock();

            try
            {
                fr = new FileReader("loggedinusers.txt");
                fw = new FileWriter("loggedinusers.txt");
                br = new BufferedReader(fr);
                bw = new BufferedWriter(fw);

                String line = br.readLine();
```

Picture 27. Part 1 “FileManipulation” file

```
    while(line != null)
    {
        if(!username.equals(line))
        {
            remainingUsers.add(line+"/n");
        }
    }

    if(username.equals("=empty="))
    {
        remainingUsers.clear();
    }

    bw.write("");
    bw.flush();
    fw = new FileWriter("loggedinusers.txt",true);
    bw = new BufferedWriter(fw);

    for(String user : remainingUsers)
    {
        bw.write(user);
    }

}
catch(Exception e)
{}
```

Picture 28. Part 2. “FileManipulation” file



```
finally
{
    try
    {
        bw.close();
    }
    catch(Exception e)
    {
        System.out.println("class FileIO, Method :removeLoggedInUser, fail to close buffered writer \n"+e.toString());
    }

    try
    {
        fw.close();
    }
    catch(Exception e)
    {
        System.out.println("class FileIO, Method : removeLoggedInUser, fail to close File writer \n"+e.toString());
    }

     try
    {
        br.close();
    }
    catch(Exception e)
    {
        System.out.println("class FileIO, Method : removeLoggedInUser, fail to close buffered reader \n"+e.toString())
    }
    try
    {
        fr.close();
    }
    catch(Exception e)
    {
        System.out.println("class FileIO, Method : removeLoggedInUser, fail to close File reader \n"+e.toString());
    }

    lock.unlock();
```

Picture 29. Part 3. “FileManipulation” file

```java
//this method checks if a user is already tracked as logged in and returns a boolean true or false
public boolean isUserAlreadyLoggedIn(String username)
{
    FileReader fr = null;
    FileWriter fw = null;
    BufferedReader br = null;
    BufferedWriter bw = null;

    boolean result = true;
    if(!lock.isHeldByCurrentThread())
    {
        lock.lock();

        try
        {
            fr = new FileReader("loggedinusers.txt");
            fw = new FileWriter("loggedinusers.txt",true);
            br = new BufferedReader(fr);
            bw = new BufferedWriter(fw);

            String line = br.readLine();

            int counter = 0;

            while(line != null)
            {
                if(username.equals(line))
                {
                    counter++;
                }
                line = br.readLine();
            }

            if(counter == 0)
            {
                bw.write(username + "\n");
                bw.flush();
                result = false;
            }

        }
```

Picture 30. Part 4. “FileManipulation” file

```
catch(Exception e)
{}
finally
{
    try
    {
        bw.close();
    }
    catch(Exception e)
    {
        System.out.println("class FileIO, Method :isUserAlreadyLoggedIn, fail to close buffered writer \n"+e.toString());
    }

    try
    {
        fw.close();
    }
    catch(Exception e)
    {
        System.out.println("class FileIO, Method : isUserAlreadyLoggedIn, fail to close File writer \n"+e.toString());
    }

     try
    {
        br.close();
    }
    catch(Exception e)
    {
        System.out.println("class FileIO, Method : isUserAlreadyLoggedIn, fail to close buffered reader \n"+e.toString());
    }
    try
    {
        fr.close();
    }
    catch(Exception e)
    {
        System.out.println("class FileIO, Method : isUserAlreadyLoggedIn, fail to close File reader \n"+e.toString());
    }

    lock.unlock();

}
```

Picture 31. Part 5. “FileManipulation” file

```java
//this method is for writing in to a file and it keeps track of usernames and passwords of users
public void Write2File(String filename, HashMap<String, String> hm)
{

    if(!lock.isHeldByCurrentThread())
    {

        lock.lock();

        FileWriter fw = null;
        BufferedWriter bw = null;


        try
        {
            fw = new FileWriter(filename, true);
            bw = new BufferedWriter(fw);

            for(String username : hm.keySet())
            {
                bw.write(username + "::" + hm.get(username));
                System.out.println( "WROTE : " +username + "::" + hm.get(username));
                bw.write("\n");
            }

        }
        catch(Exception e)
        {
            System.out.println("class FileIO, Method : Write2File \n"+e.toString());

        }
        finally
        {
            try
            {
                bw.close();
            }
            catch(Exception e)
            {
                System.out.println("class FileIO, Method : Write2File, fail to close buffered writer \n"+e.toString());
            }
```

Picture 32. Part 6. “FileManipulation” file

```java
        try
        {
            fw.close();
        }
        catch(Exception e)
        {
            System.out.println("class FileIO, Method : Write2File, fail to close File writer \n"+e.toString());
        }

        lock.unlock();
    }
```

Picture 33. Part 7. “FileManipulation” file

```
//this method reads from a file and it is for retrieving the usernames with the passwords of the users
public void ReadFromFile(String filename, HashMap<String, String> hm)
{


    if(!lock.isHeldByCurrentThread())
    {



        lock.lock();
        FileReader fr = null;
        BufferedReader br = null;

        try
        {
            fr = new FileReader(filename);
            br = new BufferedReader(fr);

            String line = br.readLine();



            String[] brokenline = new String[2];

            while(line != null)
            {
                System.out.println("READ LINE IS : " + line);
                brokenline[0] = line.substring(0, line.indexOf("::"));
                System.out.println("USERNAME IS : " + brokenline[0]);
                brokenline[1] = line.substring(line.indexOf("::") + 2, line.length());
                System.out.println("PASSWORD IS : " + brokenline[1]);

                hm.put(brokenline[0], brokenline[1]);

                line = br.readLine();
            }
```

Picture 34. Part 8. “FileManipulation” file

```
catch(Exception e)
{
    System.out.println("class FileIO, Method : ReadFromFile (hashmap) \n"+e.toString());
}
finally
{
    try
    {
        br.close();
    }
    catch(Exception e)
    {
        System.out.println("class FileIO, Method : ReadFromFile(hashmap), fail to close buffered reader \n"+e.toString());
    }
    try
    {
        fr.close();
    }
    catch(Exception e)
    {
        System.out.println("class FileIO, Method : ReadFromFile(hashmap), fail to close File reader \n"+e.toString());
    }

    lock.unlock();
```

Picture 35. Part 9. “FileManipulation” file

```
//this method reads the list of songs from a file
public void ReadFromFile(String filename,List<String> songlist )
{

    if(!lock.isHeldByCurrentThread())
    {
       lock.lock();
       FileReader fr = null;
       BufferedReader br = null;

       try
       {
           fr = new FileReader(filename);
           br = new BufferedReader(fr);

           String line = br.readLine();
           songlist.clear();
           while(line != null)
           {


               songlist.add(line);

               line = br.readLine();
           }


       }
       catch(Exception e)
       {
           System.out.println("class FileIO, Method : ReadFromFile(List) \n"+e.toString());
       }
       finally
       {
           try
           {
               br.close();
           }
           catch(Exception e)
           {
               System.out.println("class FileIO, Method : ReadFromFile(List), fail to close buffered reader \n"+e.toString());
           }
           try
           {
               fr.close();
           }
```

Picture 36. Part 10. “FileManipulation” file

```
catch(Exception e)
{
    System.out.println("class FileIO, Method : ReadFromFile(List), fail to close File reader \n"+e.toString());
}

lock.unlock();
```

Picture 37. Part 11. “FileManipulation” file

Checking for duplicate entries in last song lists

The last file on the server side is named “Check4DuplicateSongs” and it is for checking in the last downloaded song list and last streamed list if there are any duplicate titles in order to remove them because the author did not find necessary to show to the client more than once a song if it is listened to by more than one user.

```java
/**
 * @author Lalagkos Lysandros Konstantinos
 */

//the entire class is made to check if in the last downloaded/streamed songs list are any duplicates and remove them in order to show to client
public class Check4DuplicateSongs {
    public void checkForDuplicates(List<String> lastDLsongs,List<String> lastSTRsongs)
    {
        List<String> tempDL = new ArrayList<String>();
        List<String> tempSTR = new ArrayList<String>();

        for(String dl : lastDLsongs)
        {
            if(!tempDL.contains(dl))
            {
                tempDL.add(dl);
            }
        }

        for(String str : lastSTRsongs)
        {
            if(!tempSTR.contains(str))
            {
                tempSTR.add(str);
            }
        }

        lastDLsongs = tempDL;
        lastSTRsongs = tempSTR;
    }

    public List<String> checkListForDuplicates(List<String> stringList)
    {
        List<String> temp = new ArrayList<>();

        for(String st : stringList)
        {
            if(!temp.contains(st))
            {
                temp.add(st);
            }
        }

        return temp;

    }
```

Picture 38. “Check4DuplicateSongs” file

Client side

The client side of this application has two files, the client file and the audio player file. The client file is responsible for the communication with the server and it gives the ability to the user to enter numerical or other input for requesting a variety of services from the server. The audio player file is for playing the requested .wav files either if the request was for stream or if the client decided to listen to already downloaded songs.

Client

The client file prompts at start the user to enter the servers IP address, it is designed this way in order to run from different machines on the same network. If both the server and the client are running on the same machine, then the client can enter the local address 127.0.0.1 to connect to the server without any problems. The client file also has mutual exclusion for smooth communication with the server. It has methods for sending and receiving messages, receiving files, receiving streams and calls the audio player when needed.

```
 * @author Lalagkos Lysandros Konstantinos
 */

//client class, contains all logic for client
public class Client {

    InetAddress ip ;//= InetAddress.getByName("localhost");
    int port; //= 42069;
    DataInputStream dis;
    DataOutputStream dos;
    Socket soc;
    ReentrantLock lock = new ReentrantLock(true);//used for mutual exclusion
    Scanner scn = new Scanner(System.in);
    String input;
    String output;

    BufferedInputStream bis;
    FileManipulation fileManip = new FileManipulation();
    Integer endoffile = new Integer(129);
    byte[] EndOfFile = new byte[1024];

    public void connect() throws UnknownHostException, IOException
    {
        System.out.println("Provide Server IP Address, if server is on same machine, give 127.0.0.1");//the client must provide the access server ip or if on same machine
        ip = InetAddress.getByName(scn.nextLine());
        port = 42069;

        soc = new Socket(ip,port);//socket created

        dis = new DataInputStream(soc.getInputStream());//creating communication streams
        dos = new DataOutputStream(soc.getOutputStream());

        soc.setSoTimeout(2*60*1000);//internal time out of 2 minutes for the client

        bis = new BufferedInputStream(dis);

        for(int i = 0; i< EndOfFile.length;i++)
        {
            EndOfFile[i] = endoffile.byteValue();
        }

    }
```

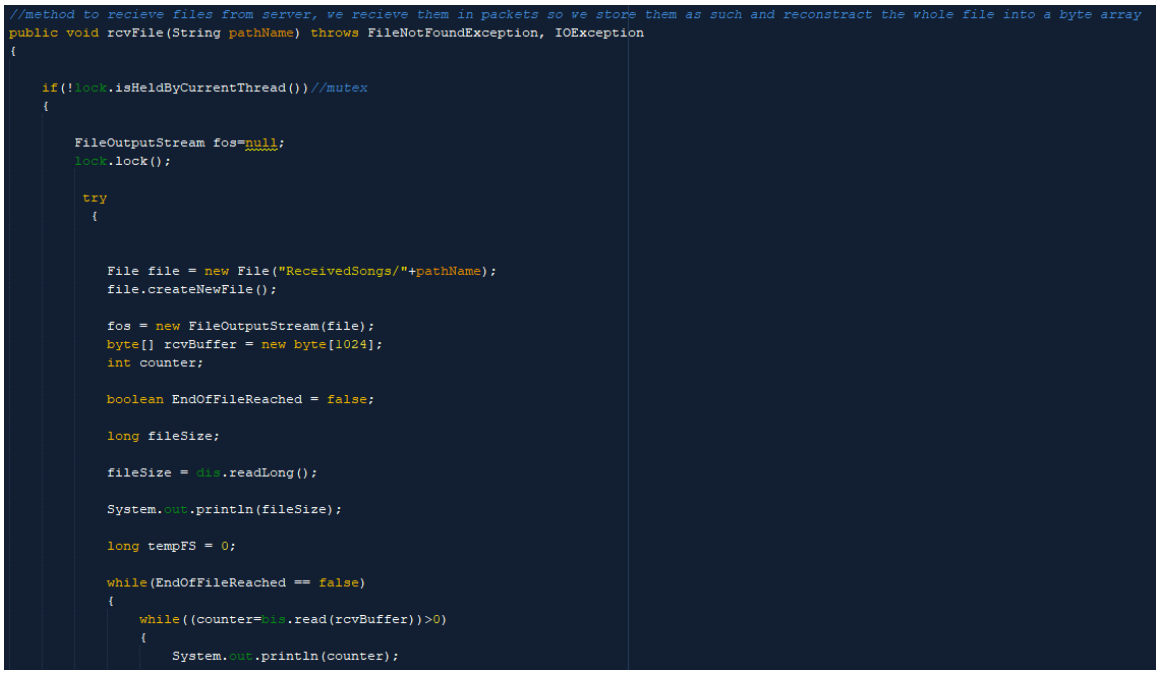
Picture 39. Part 1. “Client” file

```
//handshake made to establish connection with server, server is waiting for something like this to establish connection
public boolean clInitHS() throws IOException
{
    output="HELLO SERVER";
    sendMessage(output);
    //System.out.println("i sent hello server");
    input=recieveMessage();
    //System.out.println("i rcvd :"+input);
    if(input.equals("HELLO CLIENT"))
    {
        output="OK SERVER";
        sendMessage(output);
        //System.out.println("i sent :"+output);
        return true;
    }
    else
    {
        return false;
    }

}

//method to send messages to server through the stream
public void sendMessage(String toSend)
{
    if(!lock.isHeldByCurrentThread())//mutex
    {
        lock.lock();

        try
        {
            dos.writeUTF(toSend);
            dos.flush();

        }
        catch(Exception e)
        {
            System.out.println(e.toString());
        }
        finally
        {
            lock.unlock();
        }
    }
}
```

Picture 40. Part 2. “Client” file

```java
//method to recieve messages from server
public String recieveMessage() throws IOException
{

    String recieveMessage = "";

    if(!lock.isHeldByCurrentThread())//mutex
    {
        lock.lock();

        try
        {

            recieveMessage = dis.readUTF();

        }
        catch(Exception e)
        {
            System.out.println("client -> rcvmsg problem");
            System.out.println(e.toString());
        }
        finally
        {
            //System.out.println("unlock lock");
            lock.unlock();
        }

    }
    else
    {
        //System.out.println("Thread was occupied");
        throw new IOException("problem with sending messages");
    }
    return recieveMessage;
}
```

Picture 41. Part 3. “Client” file



```java
//method to recieve files from server, we recieve them in packets so we store them as such and reconstract the whole file into a byte array
public void rcvFile(String pathName) throws FileNotFoundException, IOException
{

    if(!lock.isHeldByCurrentThread())//mutex
    {

        FileOutputStream fos=null;
        lock.lock();

         try
          {


            File file = new File("ReceivedSongs/"+pathName);
            file.createNewFile();

            fos = new FileOutputStream(file);
            byte[] rcvBuffer = new byte[1024];
            int counter;

            boolean EndOfFileReached = false;

            long fileSize;

            fileSize = dis.readLong();

            System.out.println(fileSize);

            long tempFS = 0;

            while(EndOfFileReached == false)
            {
                while((counter=bis.read(rcvBuffer))>0)
                {
                    System.out.println(counter);
```

Picture 42. Part 4. “Client” file

```
        System.out.println(counter);
        try
        {

            tempFS+=counter;

            System.out.println(tempFS);

            if(tempFS==fileSize)
            {
                fos.write(rcvBuffer, 0, counter);
                System.out.println("reached end of file");
                EndOfFileReached = true;
                break;
            }
            else
            {
                fos.write(rcvBuffer, 0, counter);
            }
        }


        catch(EOFException e)
        {
            System.out.println("problem");
            break;
        }
        catch(Exception ex)
        {
            System.out.println("eofe was here");
            break;

        }
    }

    FileWriter myWriter = new FileWriter("ReceivedSongsList.txt",true);

    myWriter.write(pathName+"\n");
    myWriter.close();

    System.out.println("Server stopped transmition,type OK to continue");

}
```

Picture 43. Part 5. “Client” file

```
            fos.close();

        }
        catch(EOFException eofer)
        {
            System.out.println("eofe exception in rcvFile");
        }
        catch(IOException e)
        {
            System.out.println(e.toString());
            System.out.println("IO exception in rcvFile");

        }
        finally
        {

            lock.unlock();
        }

    }
else
{
    throw new IOException("problem with receving file");
}
```

Picture 44. Part 6. “Client” file

```
//this method is for recieving a song stream from the server
//as the stream comes from the server we recive each packet an
public void rcvStream()
{
    if(!lock.isHeldByCurrentThread())//mutex
    {
        lock.lock();

        AudioPlayer audioplayer ;

        try
        {

            long fileSize;

            fileSize = dis.readLong();
            System.out.println("fileSize : " +fileSize);

            boolean endoffile = false;

            byte[] wholesong = new byte[(int)fileSize];

            byte[] rcvBuffer = new byte[1024];

            int counter;

            long tempcounter = 0;

            long tempsize = 0;

            while(!endoffile)
            {

                while((counter = bis.read(rcvBuffer))>0)
                {
                    tempsize += counter;
```

Picture 45. Part 7. “Client” file

```
            if(tempsize!= fileSize)
            {
                for(byte b : rcvBuffer)
                {
                    wholesong[(int)tempcounter] = b;
                    tempcounter++;
                }
            }
            else
            {
                System.out.println("Last buffer operation");
                System.out.println(counter + "#" + rcvBuffer.length);
                for(byte b : rcvBuffer)
                {

                    if(tempcounter<fileSize)
                    {
                        System.out.println(tempcounter);
                        wholesong[(int)tempcounter] = b;
                        System.out.println(tempcounter);
                        tempcounter++;
                    }
                }
                endoffile = true;
                break;

            }

        }

    }

System.out.println("Containing array is size : " + wholesong.length);

audioplayer = new AudioPlayer();

audioplayer.musicMenu(wholesong,scn);
```

Picture 46. Part 8. “Client” file

```
        catch(Exception e)
        {
            System.out.println(e.toString());

        }
        finally
        {
            lock.unlock();
        }

    }
    else
    {
        System.out.println("Thread was occupied");
    }
}

//this is the main method of the client
public static void main(String args[]) throws IOException, UnsupportedAudioFileException, LineUnavailableException {

   Client client = new Client();
   client.connect();//we connect to the server through socket
   boolean keepwhiling = client.clInitHS();//getting results of handshake
   List<String> songList = new ArrayList<>();


   if(keepwhiling)//checking results
   {
       System.out.println("Handshake done successfully");
       client.fileManip.ReadFromFile("ReceivedSongsList.txt",songList);
   }
   else
   {
       System.out.println("Server not responding to handshake");
       client.soc.close();
   }
```

Picture 47. Part 9. “Client” file

```
String servermsg="";

while(keepwhiling)//main while for keeping client open and responsive
 {

    try
    {
        servermsg = client.recieveMessage();
    }
    catch(Exception e)
    {
        servermsg="";
        System.out.println("exception in client main after rcv msg");
    }

    if(servermsg.contains("w8"))//server said is waiting for response
    {
        //System.out.println("rcv had w8");
        servermsg=servermsg.replace("w8","");
        servermsg=servermsg.trim();
        System.out.println(servermsg);
        client.output=client.scn.nextLine();

        while(servermsg.contains("4.Listen to Downloaded songs") && client.output.equals("4"))//client chose to listen to already downloaded songs
        {
            do
            {
                System.out.println("Choose a song");//song list of downloaded songs appear and client choose one
                for(String song : songList)
                {
                   System.out.println(songList.indexOf(song)+". "+song);

                }

                client.output = client.scn.nextLine();

                String songPath = songList.get(Integer.parseInt(client.output));

                AudioPlayer audioplayer = new AudioPlayer();

                audioplayer.musicMenu(songPath, client.scn);

                System.out.println("Want (1)another song, or (2)go to server?");//keep listening to songs or return to server

                client.output = client.scn.nextLine();
```

Picture 48. Part 10. “Client” file

```
        while(client.output.equals("1"));

        System.out.println(servermsg);
        client.output=client.scn.nextLine();

    }



    System.out.println(client.output);
    client.sendMessage(client.output);
}
else if(servermsg.contains("sending file"))//server sais that he is sending a file so clients prepare for recieving a file
{

    try
    {
        servermsg=servermsg.replace("sending file","");
        client.rcvFile(servermsg);
        client.fileManip.ReadFromFile("ReceivedSongsList.txt", songList);
        client.output = client.scn.nextLine();
        client.sendMessage(client.output);
    }
    catch(Exception e)
    {
        System.out.println("exception in client rcv file");
        break;
    }
}
else if(servermsg.equals("OK EXIT"))//exit handshake and closing resources and socket
{
    client.sendMessage("OK BYE");
    keepwhiling = false;
    client.dis.close();
    client.dos.close();
    client.soc.close();

}
else if(servermsg.equals("Starting Stream"))//ready to recieve stream
{

    client.rcvStream();

    System.out.println("intermediate");
```

Picture 49. Part 11. “Client” file

```
                client.sendMessage("OK");


            }
            else
            {

                System.out.println(servermsg);
            }


        }

    }
}
```

Picture 50. Part 12. “Client” file

The Audio Player

The last file of the client side and the last file of the whole application is the “AudioPlayer” file. The audio player is called from the “Client” file and is used to play the .wav

files regardless if are from a stream request or already downloaded from the server. Audio player has it is own menu and is capable to pause, stop, play on specific time, and reset a song.

```
//class for playing audio for the client, this class belong to the client and works for the client
public class AudioPlayer
{

        // to store current position
        Long currentFrame;
        Clip clip;

        // current status of clip
        String status;

        AudioInputStream audioInputStream;
        static String filePath;

        byte[] audio;
        InputStream inputStream;

        boolean downloadedSong=false;
        boolean streaming = false;


        // constructor
        public AudioPlayer() throws UnsupportedAudioFileException, IOException, LineUnavailableException
        {

        }
        //constractor that takes path for file
        public AudioPlayer(String pathname) throws UnsupportedAudioFileException, IOException, LineUnavailableException
        {
            filePath = pathname;
            downloadedSong=true;
            audioInputStream = AudioSystem.getAudioInputStream(new File(pathname).getAbsoluteFile());
            clip = AudioSystem.getClip();
            clip.open(audioInputStream);
            clip.loop(Clip.LOOP_CONTINUOUSLY);
        }
```

Picture 51. Part 1. “AudioPlayer” file

```java
//constractor that takes audio input stream for playing streamed audio and not downloaded
//method audio player, used to play downloaded and streamed audios from server or from client
public AudioPlayer(byte[] fullAudio)throws UnsupportedAudioFileException,IOException, LineUnavailableException
{
    streaming = true;
    audio = fullAudio;
    inputStream = new ByteArrayInputStream(audio);

    audioInputStream = AudioSystem.getAudioInputStream(inputStream);
    //AudioFormat format = audioInputStream.getFormat();
    clip = AudioSystem.getClip();
    clip.open(audioInputStream);
    clip.loop(Clip.LOOP_CONTINUOUSLY);
}

//menu that the client sees in order to play, pause, stop or go to a specific time in a STREAMED audio
public void musicMenu(byte[] fullAudio,Scanner sc)
{
        AudioPlayer audioPlayer;

        try
        {


                audioPlayer = new AudioPlayer(fullAudio);


                audioPlayer.play();

                boolean isPlaying =true;

                while (isPlaying)
                {
                        System.out.println("1. pause");
                        System.out.println("2. resume");
                        System.out.println("3. restart");
                        System.out.println("4. stop");
                        System.out.println("5. Jump to specific time");

                        String c;

                        c = sc.nextLine();
```

Picture 52. Part 2. “AudioPlayer” file

```
                    audioPlayer.gotoChoice(c,sc);
                    if (c.equals("4"))
                    {
                        isPlaying = false;
                        break;
                    }

                }
            }

            catch (Exception ex)
            {

                //sc.nextLine();
                System.out.println("Error with playing sound.");
                ex.printStackTrace();

            }
    }
    //menu that the client sees in order to play, pause, stop or go to a specific time in a DOWNLOADED audio
    public void musicMenu(String filepath,Scanner sc)
    {
            AudioPlayer audioPlayer;

            try
            {

                audioPlayer = new AudioPlayer("ReceivedSongs/"+filepath);


                audioPlayer.play();

                boolean isPlaying =true;

                while (isPlaying)
                {
                    System.out.println("1. pause");
                    System.out.println("2. resume");
                    System.out.println("3. restart");
                    System.out.println("4. stop");
                    System.out.println("5. Jump to specific time");

                    String c;
```

Picture 53. Part 3. “AudioPlayer” file

```
                c = sc.nextLine();

                audioPlayer.gotoChoice(c,sc);
                if (c.equals("4"))
                {
                    isPlaying = false;
                    break;
                }

            }

        }

        catch (Exception ex)
        {

            System.out.println("Error with playing sound.");
            ex.printStackTrace();

        }
}

// Work as the user enters his choice

private void gotoChoice(String c,Scanner sc) throws IOException, LineUnavailableException, UnsupportedAudioFileException
{
        switch (c)
        {
                case "1":
                    pause();
                    break;
                case "2":
                    resumeAudio();
                    break;
                case "3":
                    restart();
                    break;
                case "4":
                    stop();
                    break;
                case "5":
                    System.out.println("Enter time (" + 0 +
                    ", " + clip.getMicrosecondLength() + ")");
```

Picture 54. Part 4. “AudioPlayer” file

```
                    long c1 = sc.nextLong();
                    sc.nextLine();
                    jump(c1);
                    break;

            }

    }

    // Method to play the audio
    public void play()
    {

        FloatControl gainControl = (FloatControl) clip.getControl(FloatControl.Type.MASTER_GAIN);

        gainControl.setValue(6.0f);

        //start the clip
        clip.start();

        System.out.println("started clip");

        status = "play";
    }

    // Method to pause the audio
    public void pause()
    {
            if (status.equals("paused"))
            {
                    System.out.println("audio is already paused");
                    return;
            }
            this.currentFrame =
            this.clip.getMicrosecondPosition();
            System.out.println("set mseconds" + currentFrame);


            clip.stop();
            System.out.println("stopped clip");

            status = "paused";
    }
```

Picture 55. Part 5. “AudioPlayer” file

```
// Method to resume the audio
public void resumeAudio() throws UnsupportedAudioFileException, IOException, LineUnavailableException
{
        if (status.equals("play"))
        {
                System.out.println("Audio is already "+
                "being played");
                return;
        }


        clip.close();

        System.out.println("closed clip");

        resetAudioStream();

        System.out.println("reset stream");


        System.out.println("mseconds are " + currentFrame);
        clip.setMicrosecondPosition(currentFrame);

        System.out.println("got mseconds");

        this.play();
}

// Method to restart the audio
public void restart() throws IOException, LineUnavailableException, UnsupportedAudioFileException
{
        clip.stop();
        clip.close();
        resetAudioStream();
        currentFrame = 0L;
        clip.setMicrosecondPosition(0);
        this.play();
}
```

Picture 56. Part 6. “AudioPlayer” file

```
// Method to stop the audio
public void stop() throws UnsupportedAudioFileException,
IOException, LineUnavailableException
{
        currentFrame = 0L;
        clip.stop();
        clip.close();
}

// Method to jump over a specific part
public void jump(long c) throws UnsupportedAudioFileException, IOException, LineUnavailableException
{
        if (c > 0 && c < clip.getMicrosecondLength())
        {
                clip.stop();
                clip.close();
                resetAudioStream();
                currentFrame = c;
                clip.setMicrosecondPosition(c);
                this.play();
        }
}

// Method to reset audio stream
public void resetAudioStream() throws UnsupportedAudioFileException, IOException, LineUnavailableException
{


    if(downloadedSong)
    {
        audioInputStream = AudioSystem.getAudioInputStream(new File(filePath).getAbsoluteFile());

        System.out.println("reset audioInputStream");

        clip.open(audioInputStream);

        System.out.println("open clip");

        clip.loop(Clip.LOOP_CONTINUOUSLY);

        System.out.println("set loop true");
```

Picture 57. Part 7. “AudioPlayer” file

```
        try
        {
            audioInputStream = AudioSystem.getAudioInputStream(new File(filePath).getAbsoluteFile());
        }
        catch(Exception e)
            {}
    }
    else if(streaming)
    {
        inputStream = new ByteArrayInputStream(audio);
        audioInputStream = AudioSystem.getAudioInputStream(inputStream);

        System.out.println("reset audioInputStream");

        clip.open(audioInputStream);

        System.out.println("open clip");

        clip.loop(Clip.LOOP_CONTINUOUSLY);

        System.out.println("set loop true");
        try
        {
            audioInputStream = AudioSystem.getAudioInputStream(inputStream);
        }
        catch(Exception e)
            {}
    }
```

Picture 58. Part 8. “AudioPlayer” file

Testing

The testing of the application happened on the same network, it was tested on a single machine where it works as intended, and it was tested on two machines in the same network as stated above meaning that the server was running on a machine and the client connected to the server from the second that was running on. Due to the pandemic I was not able to do further tests on different networks or machines so I have no knowledge if it works under different circumstances.

The distribution part was tested by opening 16 clients in order to make sure that when the server reaches 5 clients capacity, on the sixth client it checks and creates a new instance of server in order to serve the client and then again I tested another 5 clients and another till I reached 4 servers and 16 clients. My machine resources could not handle the load but in theory if I had more resources, I could keep adding clients without a problem.

The design of the distribution has one flaw in my opinion, in order to refresh the lists of clients that left, clients that went inactive and last streamed or downloaded song, a new client must be accepted to the server. I was not able to a find a way to fix this flaw mainly due to time restrictions.

The application was created step by step, the engineering plan was to create any file, component or method was used first and then keep creating until all or most of the requirements are met.

The application on its higher level was not tested intensively on entering wrong input or doing anything out of the ordinary, the tests were about asking for a service and watch if the request is served successfully.

Overall the application as it is stated already it can successfully accept multiple client from the same or different machines in the same network, it explains to the user with appropriate messages what the user must do exactly, it can handle users logging in and creating new accounts which stores externally, it can send a .wav file to the client and the client play that file from his side of the application, it streams songs successfully, it can handle unexpected disconnection from the clients, it handles exit requests from the clients and last but not least is able to do client-server-client communication in order to show to client upon request what other users listen to.

Everything was done on Netbeans IDE 12.0.

In conclusion, testing was done to an extent, in all cases was successful but it was not tested on not expected behavior except for client disconnection, it has practically vertical scaling but in my opinion is created in a way to simulate how horizontal scale would be done, again due to the pandemic I had limited resources and I could not try or test further, for academic purposes I hope is enough.

Problems in the App

The application is capable of serving multiple clients even from different machines on the same network, it can send files to clients either for streaming or downloading, clients can log in successfully and create new accounts.

There are some things in the app that may cause it to crash such as, entering an alphabetical input where a numerical is asked for, when file manipulation is done if something goes wrong it may be possible that the client would have to stop and rerun the “Client” file. There are no other problems with the application that I know off.

Problems in development process

During the development of the application, I came across many problems and errors. The hardest error was to make the application send the .wav file without saving it in the machine and have it streamed directly to the client. Some minor issues with the load balancer where new servers where not created, I could not remove clients and keep count of them, many logical problems in the “ClientHandler” file in order to keep the server in a state where does nothing until the client sends something.

In some point when I tried to make some .mp3 file to .wav for the application, the files where not recognized and could not be played from the application, the reason was that the site I tried to do it was messing up the format, if anyone wants to add manually songs to the server I have to warn you that this problem may occur if the format is not correct.

Finally, in my opinion it was a challenging process that required a lot of hours to make it like it is in the current version and I believe it works as intended and does almost everything that is required for the assessment.

Instructions

In order to use the application or test it you need to extract the files and using an appropriate IDE, I used netbeans IDE 12.0 to create and test it, you open the project named “Distributed Systems Music App” and from the file explorer in the file “source packages” find the only package in, named “musicApp” and collapse to view the files inside. The file that must run are with that order, “AccessServer” and then “Client”, on the console of the server are some debugging messages and the servers IP address and after it outputs general messages for the state of the communication. In the clients console is the menu and everything the client needs to request or instructions or for input purposes.

You can create new accounts if you want but there are already three accounts for testing purposes. The accounts are the following:

1. Username: superman, Password: lovesbatman
2. Username: spiderman, Password: webslinger
3. Username: takis7tsouk, Password: antegeia

The server has already some songs and a list of them, in order to add songs one should put the .wav file in the directory named “ReceivedSongs” and in the file “list of songs.txt” add the name of the .wav file exactly as it is.