DHIRUBHAI AMBANI INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY

IT 304 - COMPUTER NETWORKS

Autumn 2014-2015

PROJECT: STUDENT DESK

By: ADITI BHATNAGAR

201201164

CONTENT

1.OVERVIEW: [What is the project all about?]

STUDENT DESK: The java based application following client server architecture aims at providing the students or for that matter, all the people who are connected via the same network with the following facilities:

1. Chat Hub
2. Share Box
3. Resource Pool

1. Chat Hub:

This feature allows the user to join in live chat hub. The feature imitates IRC [Internet Relay Chat] in functionality i.e. the user is able to see the conversations after entering the hub. The conversations are not stored anywhere. Whenever any new user enters the hub or any current user leaves the hub, all other clients are notified. Message entered by any user is sent to all other online users. Simultaneously, one can also send message to a single person by addressing the person's name as @<username> before typing the message. This would ensure targetting the message to that particular person.

2. Share Box:

A pre-allocated space on server stores all the files uploaded by different users i.e. users can upload the files/resources they want to share in this resource pool. The feature share box allows user to select any file from their computer and upload it on server drive space. The destination address is predefined in this application. All the files uploaded would go to servers' Deskop/clientUpload folder. If such a folder doesn't exist on Server, it will be created and then the files would be transferred from client to server.

3. Resource Pool:

This feature helps download any file uploaded on server by any user. As you select this feature a table gets loaded with all existing files uploaded on server. The table exhibits all the information such as file name, file size, socket information of client who uploaded that file and date-time of upload.

As one selects any particular row of the table, a dialog box appears offering option to download that file. On clicking OK the file gets downloaded on client's computer. In this application the location of download is predefined and hardcoded (/Desktop/serverDownload). Similar to share box, this directory gets created if doesn't exists previously on client machine.
The file server is backed up with My SQL database to save the information related to uploads.

2. Softwares used for development:

(i) JAVA Netbeans IDE

(ii) JDBC driver for SQL Connection

(iii) MySQL 5.1

(iv) OS: Ubuntu 12.04 LTS

(v) Language: JDK 1.6

3. Surfing through the code: [How is it implemented?]

1. Dashboard: This is the main class of the application. As the class is initialised, the connection to FileServer and ChatServer is established.

Source Code: [Constructor of Dashboard]

```
public Dashboard() {
      initComponents();
      cl=new FileClient();
      cl.connect();
      cr=new ChatRoom();
     dcr=new DAChatRoomApp(cr);
   }
```

2. Chat Hub:
A dedicated server is used for chat hub.
Threads are used to connect multiple clients on hub. Threads communicate primarily by sharing access to fields and the objects reference fields refer to. This form of

communication is extremely efficient, but gives scope to errors. The tool needed to prevent these errors is synchronization.Synchronisation is being applied for smooth running.

SourceCode:ChatServer: creation of new threads on accept

```
/*
* Create a client socket for each connection and pass it to a new client
* thread.
*/
while (true) {
try {
clientSocket = serverSocket.accept();
System.out.println("connected");
int i = 0;
for (i = 0; i < maxClientsCount; i++) {
if (threads[i] == null) {
(threads[i] = new clientThread(clientSocket, threads)).start();
break;
}
}
if (i == maxClientsCount) {
PrintStream os = new PrintStream(clientSocket.getOutputStream());
os.println("Server too busy. Try later.");
os.close();
clientSocket.close();
}
} catch (IOException e) {
System.out.println(e);
}
```

Source Code: Chat Server (clientThread)

```
class clientThread extends Thread {
private String clientName = null;
```

```
private DataInputStream is = null;
private PrintStream os = null;
private Socket clientSocket = null;
private final clientThread[] threads;
private int maxClientsCount;
public clientThread(Socket clientSocket, clientThread[] threads) {
this.clientSocket = clientSocket;
this.threads = threads;
maxClientsCount = threads.length;
}
public void run() {
int maxClientsCount = this.maxClientsCount;
clientThread[] threads = this.threads;
try {
/*
* Create input and output streams for this client.
*/
is = new DataInputStream(clientSocket.getInputStream());
os = new PrintStream(clientSocket.getOutputStream());

String name;
while (true) {

    os.println("Enter your name.");
name = is.readLine().trim();
if (name.indexOf('@') == -1) {
break;
} else {
os.println("The name should not contain '@' character.");
}
}
/* Welcome the new the client. */
os.println("Welcome " + name
+ " to our chat room.\nTo leave enter /quit in a new line.");
synchronized (this) {
for (int i = 0; i < maxClientsCount; i++) {
if (threads[i] != null && threads[i] == this) {
clientName = "@" + name;
break;
}
```

```
}
for (int i = 0; i < maxClientsCount; i++) {
if (threads[i] != null && threads[i] != this) {
threads[i].os.println("*** A new user " + name
+ " entered the chat room !!! ***");
}
}
}
/* Start the conversation. */
while (true) {
String line = is.readLine();
if (line.startsWith("/quit")) {
break;
}
/* If the message is private sent it to the given client. */
if (line.startsWith("@")) {
String[] words = line.split("\\s", 2);
if (words.length > 1 && words[1] != null) {
words[1] = words[1].trim();
if (!words[1].isEmpty()) {
synchronized (this) {
for (int i = 0; i < maxClientsCount; i++) {
if (threads[i] != null && threads[i] != this
&& threads[i].clientName != null
&& threads[i].clientName.equals(words[0])) {
threads[i].os.println("<" + name + "> " + words[1]);
/*
* Echo this message to let the client know the private
* message was sent.
*/
this.os.println(">" + name + "> " + words[1]);
break;
}
}
}
}
}
} else {
/* The message is public, broadcast it to all other clients. */
synchronized (this) {
```

```
for (int i = 0; i < maxClientsCount; i++) {
if (threads[i] != null && threads[i].clientName != null) {
threads[i].os.println("<" + name + "> " + line);
}
}
}
}
}
synchronized (this) {
for (int i = 0; i < maxClientsCount; i++) {
if (threads[i] != null && threads[i] != this
&& threads[i].clientName != null) {
threads[i].os.println("*** The user " + name
+ " is leaving the chat room !!! ***");
}
}
}
os.println("*** Bye " + name + " ***");
/*
* Clean up. Set the current thread variable to null so that a new client
* could be accepted by the server.
*/
synchronized (this) {
for (int i = 0; i < maxClientsCount; i++) {
if (threads[i] == this) {
threads[i] = null;
}
}
}
/*
* Close the output stream, close the input stream, close the socket.
*/
is.close();
os.close();
clientSocket.close();
} catch (IOException e) {
}
}
```

Source Code: Chat Client [Continuous printing of messages from server]

```
public void run() {
```

```
/*
* Keep on reading from the socket till we receive "Bye" from the
* server. Once we received that then we want to break.
*/
   System.out.println("haan haan run ho rela h");
String responseLine;
try {
while ((responseLine = is.readLine()) != null) {
   System.out.println("rp:"+responseLine);

cr.getTa().append(responseLine+"\n");
if (responseLine.indexOf("*** Bye") != -1)
break;
}
closed = true;
} catch (IOException e) {
System.err.println("IOException:  " + e);
}
}
```

Source Code:Chat Client: [SEnding message to server on button click]

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
   buttonClicked=true;
   dcr.os.println(getInputText().trim());  //Getting the output stream of current client and printing the text entered
   jTextArea2.setText("");  //Clearing the text area, for further message }
```

3. Share Box:

A dedicated FileServer is created to manage uploads and downloads.

On clicking on upload file button, a file chooser dialog box appears, from which you can choose file. The connection made to the server in Dashboard is now put to use. DEpending on whether the user wishes to upload or download , accordingly the code is executed via the same connection.

Source Code: [Upload Button]

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
final JFileChooser fc = new JFileChooser();
    int returnVal = fc.showOpenDialog(jDialog1);
     if (returnVal == JFileChooser.APPROVE_OPTION) {
           File file = fc.getSelectedFile();
           cl.sendFile(file.getAbsolutePath());
          System.out.println("Sending: " + file.getAbsolutePath() );
            jDialog1.dispose();
            JOptionPane.showMessageDialog(null, "Yoo! Your file has made to the Resource Pool! :D");

     }
     else {
           System.out.println("Open command cancelled by user.");
       }
 }
```

The file server is also multi-threaded to allow multiple clients to connect with ease.

```
class CLIENTConnection implements Runnable {

    private Socket clientSocket;
    private ObjectInputStream in = null;
```

```
    private ObjectOutputStream os = null;
    private FileEvent fileEvent;
    private File dstFile = null;
    private FileOutputStream fileOutputStream = null;
    private String destinationPath="/home/infinite/Desktop/ServerDownload/";  //hard-coded destination
    public CLIENTConnection(Socket client) {
        this.clientSocket = client;
    }

    @Override
    public void run() {
        try {
            os = new ObjectOutputStream(clientSocket.getOutputStream());

            in = new ObjectInputStream(
            clientSocket.getInputStream());
            Object clientSelection = null;
            while(true)
            {
                System.out.println("Keep sending!");
                try {
                clientSelection = in.readObject();
            } catch (ClassNotFoundException ex) {

Logger.getLogger(CLIENTConnection.class.getName()).log(Level.SEVERE, null, ex);
            }
```

```
        /*Depending on what the client wants to do, according to option string written
on it's output stream server makes a decisive call */
                if(clientSelection.equals("1"))  {
                    downloadFile();   //Client to server file transfer
                } else if(clientSelection.equals("2"))
                {
                 try {
                    getUploads();
                      }
                 catch (SQLException ex) {

Logger.getLogger(CLIENTConnection.class.getName()).log(Level.SEVERE,  null,
ex);
                 }
                    }
                else if(clientSelection.equals("3"))
                {
                 try {
                    sendFile((String)in.readObject());
                 } catch (ClassNotFoundException ex) {

Logger.getLogger(CLIENTConnection.class.getName()).log(Level.SEVERE,  null,
ex);
                 }
                }

Logger.getLogger(CLIENTConnection.class.getName()).log(Level.SEVERE,  null,
ex);


        }
```

```
        }
        catch (IOException ex) {
            Logger.getLogger(CLIENTConnection.class.getName()).log(Level.SEVERE,
null, ex);
        }
}
```

Source Code:[For server to download file sent by client, MySQL connection, downloadFile() method at Server Side, sendFile() method at client side]

```
 public void downloadFile() {
        String UID="root";
        String PWD="123";
        String
DB="jdbc:mysql://localhost:3306/uploadFiles?user="+UID+"&password="+PWD;




        try {
            java.sql.Connection con = DriverManager.getConnection(DB);
             java.sql.Statement stmt= con.createStatement();



            fileEvent  = (FileEvent)  in.readObject();
            if (fileEvent.getStatus().equalsIgnoreCase("Error"))  {
                System.out.println("Error occurred ..So exiting");
                System.exit(0);
            }
```

```
        String outputFile = fileEvent.getDestinationDirectory() +
fileEvent.getFilename();
        if (!new File(fileEvent.getDestinationDirectory()).exists()) {
           new File(fileEvent.getDestinationDirectory()).mkdirs();
        }
        dstFile = new File(outputFile);
        fileOutputStream = new FileOutputStream(dstFile);
        fileOutputStream.write(fileEvent.getFileData());
        fileOutputStream.flush();
        fileOutputStream.close();
        System.out.println("Output file : " + outputFile + " is successfully saved ");
        Thread.sleep(3000);
         System.out.println(""+Calendar.getInstance().getTime());
        String query="INSERT INTO UFiles
VALUES('"+fileEvent.getFilename()+"',"+fileEvent.getFileSize()+",'"+clientSocket+"
','"+Calendar.getInstance().getTime()+"');";
        stmt.executeUpdate(query);
        con.close();
      } catch (IOException e) {
        e.printStackTrace();
      } catch (ClassNotFoundException e) {
        e.printStackTrace();
      } catch (Exception e) {
        e.printStackTrace();
      }
       return;
    }
```

```
public void sendFile(String sourceFilePath) {
    System.out.println("o fenny re");
    try {
        outputStream.writeObject("1");
    } catch (IOException ex) {
        Logger.getLogger(FileClient.class.getName()).log(Level.SEVERE, null, ex);
    }
    fileEvent = new FileEvent();
    String fileName = sourceFilePath.substring(sourceFilePath.lastIndexOf("/") + 1,
sourceFilePath.length());
    String path = sourceFilePath.substring(0, sourceFilePath.lastIndexOf("/") + 1);
    fileEvent.setDestinationDirectory(destinationPath);
    fileEvent.setFilename(fileName);
    fileEvent.setSourceDirectory(sourceFilePath);
    File file = new File(sourceFilePath);
    if (file.isFile()) {
        try {
            DataInputStream diStream = new DataInputStream(new
FileInputStream(file));
            long len = (int) file.length();
            byte[] fileBytes = new byte[(int) len];
            int read = 0;
            int numRead = 0;
            while (read < fileBytes.length && (numRead = diStream.read(fileBytes,
read,
                fileBytes.length - read)) >= 0) {
                read = read + numRead;
            }
```

```
            fileEvent.setFileSize(len);
            fileEvent.setFileData(fileBytes);
            fileEvent.setStatus("Success");
        } catch (Exception e) {
            e.printStackTrace();
            fileEvent.setStatus("Error");
        }
    } else {
        System.out.println("path specified is not pointing to a file");
        fileEvent.setStatus("Error");
    }
    //Now writing the FileEvent object to socket
    try {
        outputStream.writeObject(fileEvent);
        System.out.println("Done...Going to exit");
        Thread.sleep(3000);
        //socket.close();
       // System.exit(0);
    } catch (IOException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

}
```

---

4. Resource Pool:

This uses the pre-established connection to server, for doing two things:

1. Fetching the records of uploads made by clients from UFile database managed at server end, and finally displaying them into tabular form.

[getUploads() method of FileServer, fetchRecords() method at client side.]

2. On clicking the tabular row, and choosing download option , it uses the same connection to download the file from server to client.

[sendFile() method of FileServer, receiveFile() method at Client side]

Source Code:[getUploads() Server Side]

```
//We simply write array list of object[] to output stream.
public void getUploads() throws SQLException
   {
      String UID="root";
      String PWD="123";
      String
DB="jdbc:mysql://localhost:3306/uploadFiles?user="+UID+"&password="+PWD;
      ResultSet rs = null;
      java.sql.Connection con = null;
      try {
         con = DriverManager.getConnection(DB);
          java.sql.Statement stmt= con.createStatement();



         String query="SELECT * FROM UFiles ;";
        rs=stmt.executeQuery(query);



      }
```

```
        catch (Exception e) {
            e.printStackTrace();
        }
        int p=0;



        List<Object[]> records=new ArrayList<Object[]>();
while(rs.next()){
    p++;
    int cols = rs.getMetaData().getColumnCount();
    Object[] arr = new Object[cols];
    for(int i=0; i<cols; i++){
      arr[i] = rs.getObject(i+1);
    }
    records.add(arr);
}
            try {
                os.writeObject(records);
            } catch (IOException ex) {

Logger.getLogger(CLIENTConnection.class.getName()).log(Level.SEVERE,  null,
ex);
            }



   con.close();
```

```
            return;
        }
```

Source Code:[Send file:server side]

```
public void sendFile(String fileName) {
            fileEvent = new FileEvent();
        String sourceFilePath = "/home/infinite/Desktop/ClientUpload/"+fileName;
        fileEvent.setDestinationDirectory(destinationPath);
        fileEvent.setFilename(fileName);
        fileEvent.setSourceDirectory(sourceFilePath);
        File file = new File(sourceFilePath);
        if (file.isFile()) {
            try {
                DataInputStream diStream = new DataInputStream(new
FileInputStream(file));
                long len = (int) file.length();
                byte[] fileBytes = new byte[(int) len];
                int read = 0;
                int numRead = 0;
                while (read < fileBytes.length && (numRead = diStream.read(fileBytes,
read,
                        fileBytes.length - read)) >= 0) {
                    read = read + numRead;
                }
                fileEvent.setFileSize(len);
                fileEvent.setFileData(fileBytes);
                fileEvent.setStatus("Success");
```

```
            } catch (Exception e) {
                e.printStackTrace();
                fileEvent.setStatus("Error");
            }
        } else {
            System.out.println("path specified is not pointing to a file");
            fileEvent.setStatus("Error");
        }
        //Now writing the FileEvent object to socket
        try {

            os.writeObject(fileEvent);
            System.out.println("name:"+fileEvent.getFilename());

            System.out.println("Sent to Client...Going to exit");
            Thread.sleep(3000);
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

    return;
    }
```

Source Code: [Receive File: Client Side]

```
public void receiveFile(String fn)
{
                    try {
                outputStream.writeObject("3");
                outputStream.writeObject(fn);



            } catch (IOException ex) {

                Logger.getLogger(FileClient.class.getName()).log(Level.SEVERE,
null, ex);
            }
            try {

                fileEvent = (FileEvent) inputStream.readObject();
                if (fileEvent.getStatus().equalsIgnoreCase("Error")) {
                    System.out.println("Error occurred ..So exiting");
                    System.exit(0);
                }
                String outputFile = fileEvent.getDestinationDirectory() +
fileEvent.getFilename();
                if (!new File(fileEvent.getDestinationDirectory()).exists()) {
                    new File(fileEvent.getDestinationDirectory()).mkdirs();
                }
                dstFile = new File(outputFile);
                fileOutputStream = new FileOutputStream(dstFile);
                fileOutputStream.write(fileEvent.getFileData());
```

```
                    fileOutputStream.flush();
                    fileOutputStream.close();
                    System.out.println("File  received : " + outputFile + " is
successfully saved ");
                  }
              catch (IOException ex) {

                Logger.getLogger(FileClient.class.getName()).log(Level.SEVERE,
null, ex);
              }          catch (ClassNotFoundException ex) {
                Logger.getLogger(FileClient.class.getName()).log(Level.SEVERE,
null, ex);
              }



    }
```

5. **FileEvent :[To manage the file specifications]**

SourceCode:[File Event Class used at both server and client end]

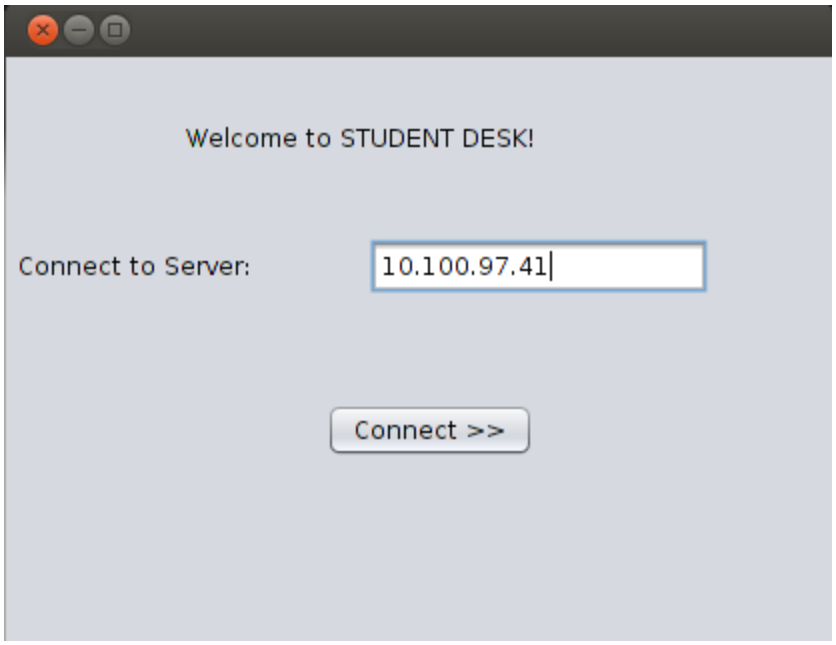
```
public class FileEvent  implements  Serializable  {

   public FileEvent()  {
   }

   private static final long serialVersionUID  = 1L;

   private String destinationDirectory;
   private String sourceDirectory;
```

```
private String filename;
private long fileSize;
private byte[] fileData;
private String status;

public String getDestinationDirectory() {
   return destinationDirectory;
}

public void setDestinationDirectory(String destinationDirectory) {
   this.destinationDirectory = destinationDirectory;
}

public String getSourceDirectory() {
   return sourceDirectory;
}

public void setSourceDirectory(String sourceDirectory) {
   this.sourceDirectory = sourceDirectory;
}

public String getFilename() {
   return filename;
}

public void setFilename(String filename) {
   this.filename = filename;
```

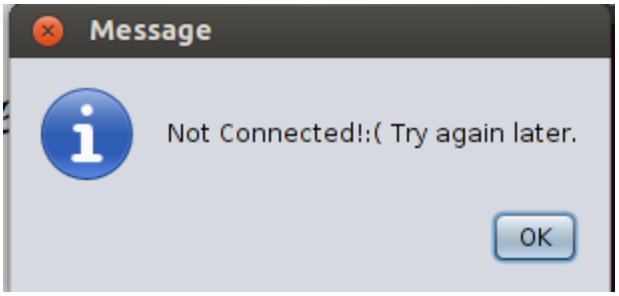
```
    }

    public long getFileSize()  {
        return fileSize;
    }

    public void setFileSize(long  fileSize)  {
        this.fileSize  = fileSize;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public byte[] getFileData()  {
        return fileData;
    }

    public void setFileData(byte[]  fileData)  {
        this.fileData  = fileData;
    }
}
```

4. Surfing through the code: [How is it implemented?]

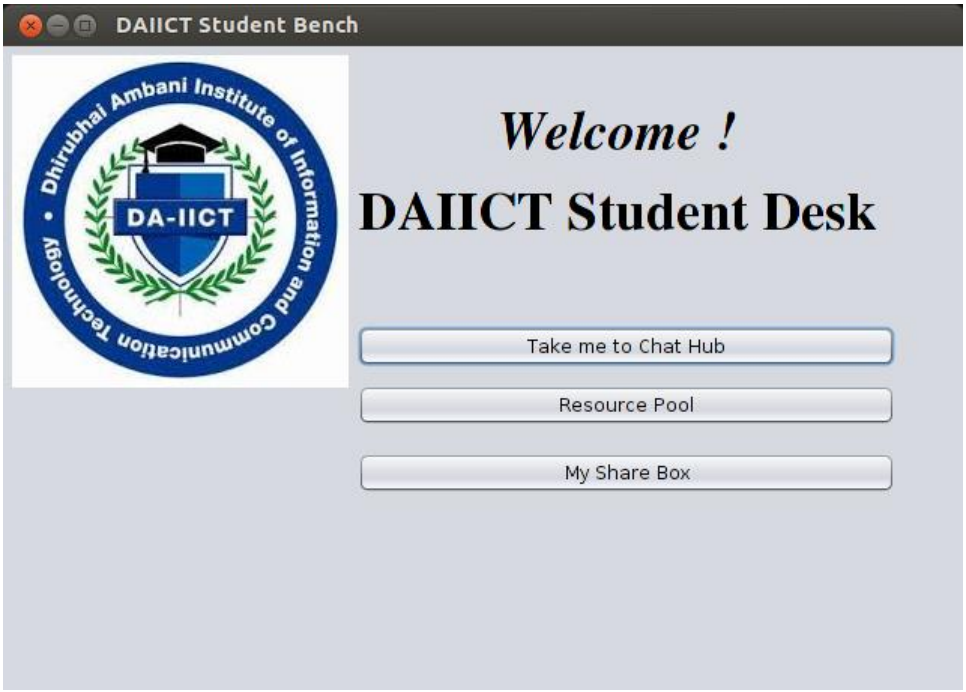
The chat server and the file server would be running on same machine. So the first screen one gets on running the application is:



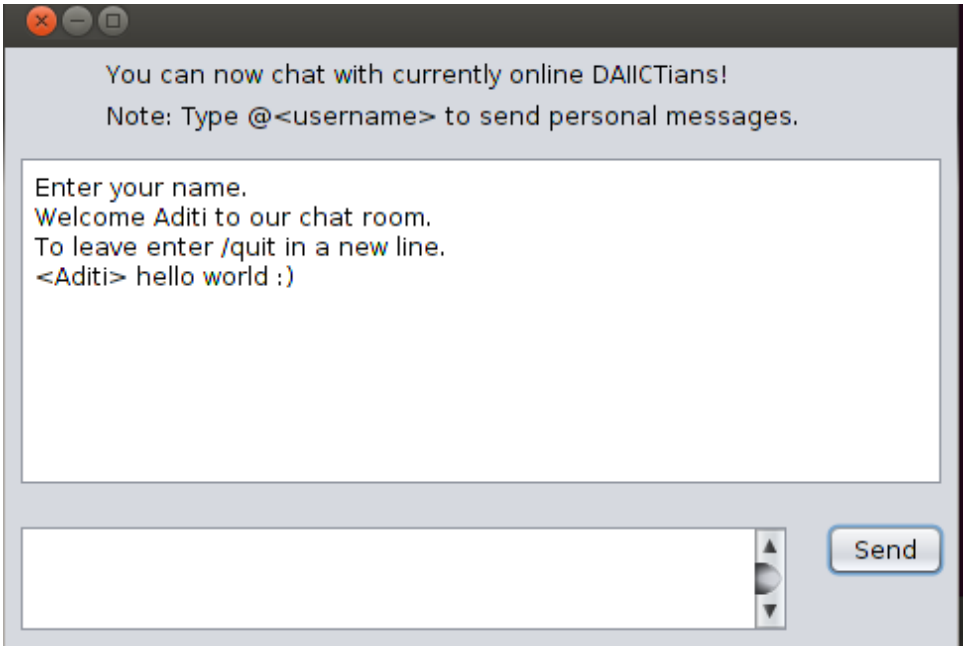
If server is down, may due to too many clients or its not presently running, then user gets this option:



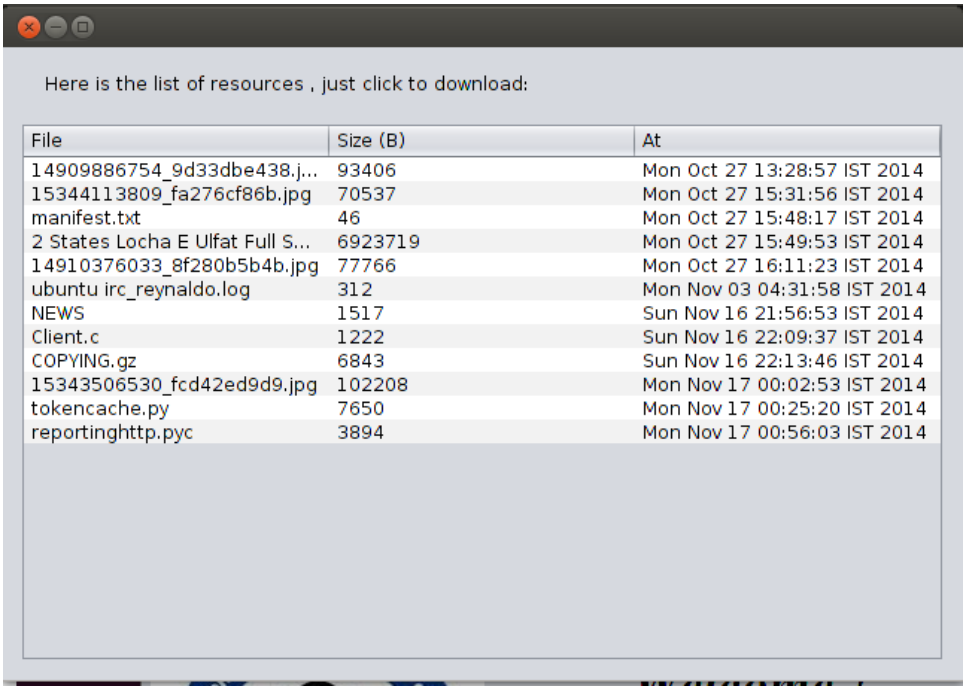
If connected, the Student Desk appears, offering the three functionalities:



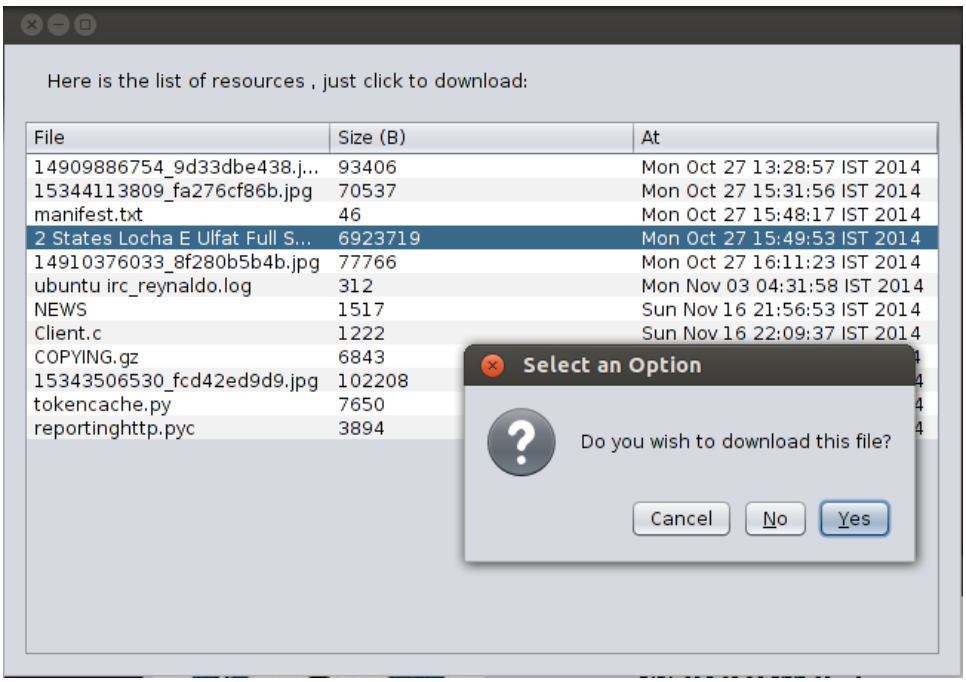
Clicking on the first one takes me to Chat Hub:



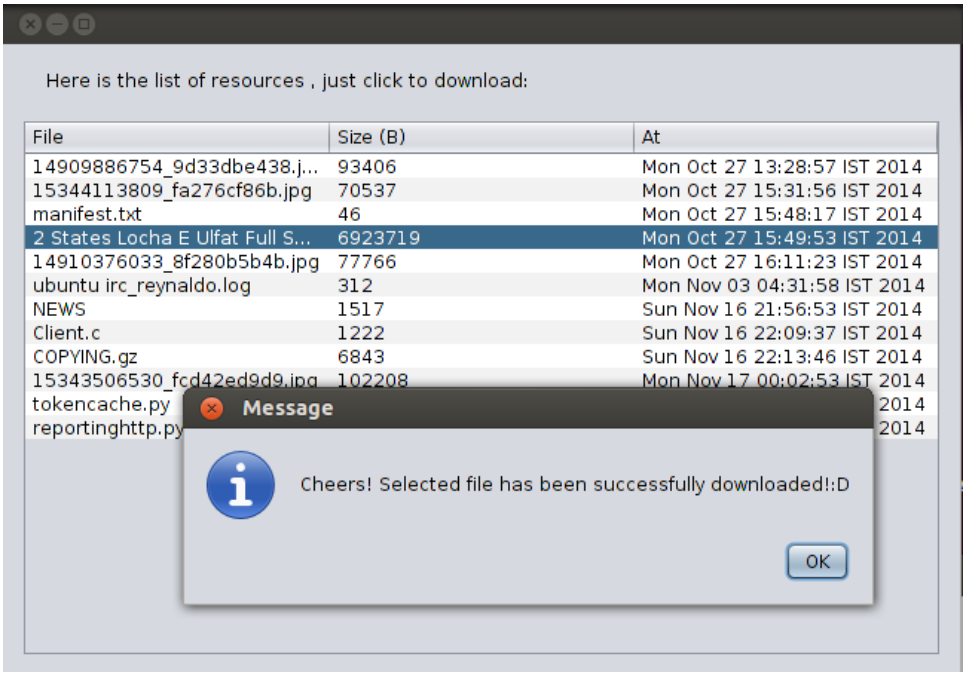
If I wish to download some file from the server, I will navigate to Resource Pool:



It will contain an exhaustive list of files uploaded on the server space. Clicking on any one of these will provide me with a download option as follows:

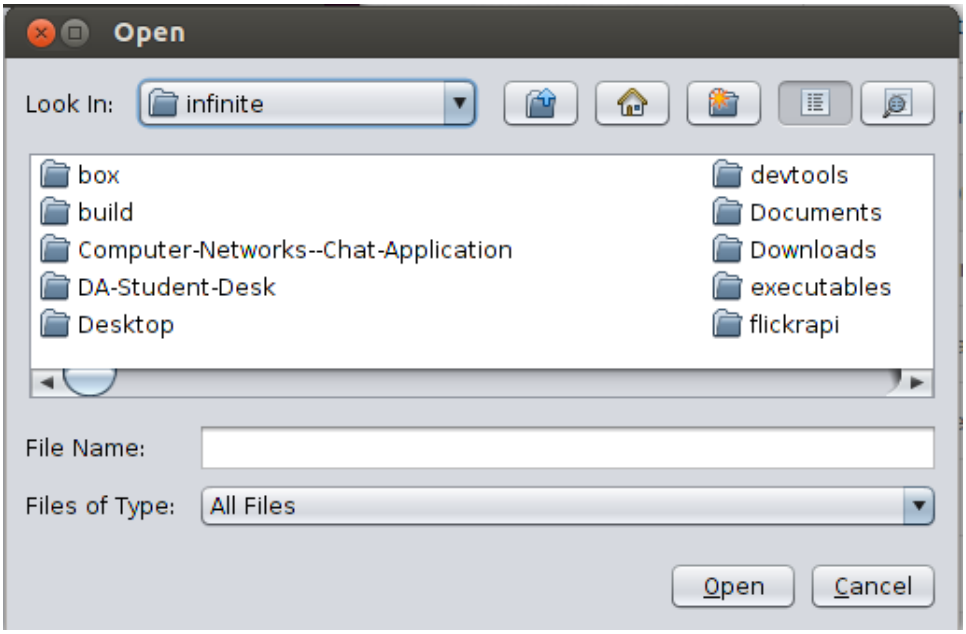


Clicking on yes, would initiate the download. After receiving the file, you get a reassurance saying:

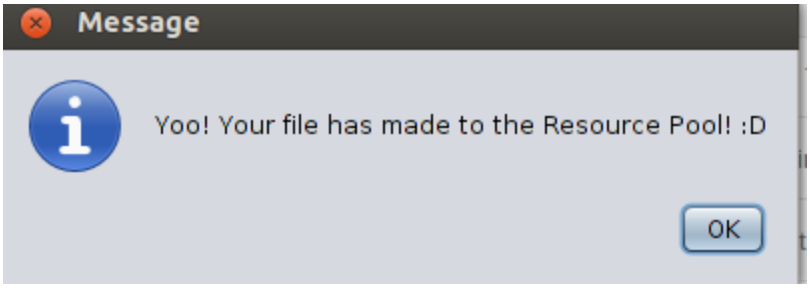


Let's navigate to Share Box. This feature helps me upload my files on server:

Clicking on Upload File would provide me with the File Chooser Dialog Box:



Select the file and click open. The upload would start. Once successfully uploaded, the user would get this message:



That's all. More features can be added accordingly to handle all possible test cases.

5. Application and Future Scope:

The application provides an integrated platform to exploit the LAN connection to the fullest. Not only are you able to transfer files and share resources conveniently on network but also you can shat with all the online users. This would be really helpful for workplaces, offices, institute campuses and every other area where collaborative work exists.

The future scope may include improving upon the existing features and adding on more features:

Improvement on existing features:

1. Add a list to display online users in chat.
2. File/Image transfer via chat portal.
3. Customized download and upload directories on server and client side.
4. Improvement on User Interface.
5. Provide option to choose location of file download.
6. Access to user wise uploads.
7. Manage server space.
8. User registration and login.

Some features that can be added:

1. Collaborative doc creation and manipulation [Like Google Doc]
2. Facilitating desktop sharing etc.

6.Instructions to make the application run on any system:

1. Establish ChatServer, simply by running the ChatServer executable jar file from one terminal.

2. Establish FileServer, by running executable FileServer jar file from other terminal. Make sure to create SQL Database namely uploadFiles and create a table named UFiles having file name, size, uploadedby and time-date fields

3. Now run the dashboard executable jar file from different terminal. Rest is evident.

Note: The locations of file upload and download are hardcoded in the source code.