

Multi Client Single Server Architecture Implementation



Project Setup Manual

Prepared by:

Rakesh Chintha

TABLE OF CONTENTS

1. OBJECTIVE	2
2. SOFTWARE REQUIRED FOR THE SETUP	2
3. FEATURES OF THE PROGRAM	2
4. ARCHITECTURE OF THE CODE	2
5. EXECUTION STEPS AND RESULTS	4
6. CODE DOCUMENTATION	17

PROJECT OBJECTIVE

This is a simple Network Programming based project focusing on how to create a Multi-Client-Single-Server architecture model implementation of an Appointment scheduler. The manual does not guide you on how to develop the code rather it explains how to setup the already existing project code that can be downloaded from the following URL <https://code.google.com/p/multiclientserver-rccreations/>

SOFTWARE REQUIRED FOR THE SETUP

- Java Development Kit (JDK) – preferably 1.5 or above.
- The code can be executed from the command line. Make sure that your Java path is properly set in the environment variables. This manual explains commands assuming that the code is run on a Windows-based platform. If you are using a Mac or Unix based environment, change your commands that suits your platform.
- Eclipse/Net Beans IDE if you require one for further development.

FEATURES OF THE PROGRAM

- ✓ Interactive
- ✓ GUI oriented
- ✓ Checks the availability of any user input appointment.
- ✓ Adds a new appointment.
- ✓ Edits an existing appointment.
- ✓ Deletes the existing appointment.
- ✓ Supports Multiple Clients. If a client is performing a write operation on the appointment scheduler, the other clients would be provided only the read access by activating “lock” feature which is implemented in the code.
- ✓ Reports the status and error messages to the UDPServer which in turn writes these messages to the ‘syslog.txt’ file.

ARCHITECTURE OF THE CODE

Below image shows the architecture of the code

(scroll down to the next page)

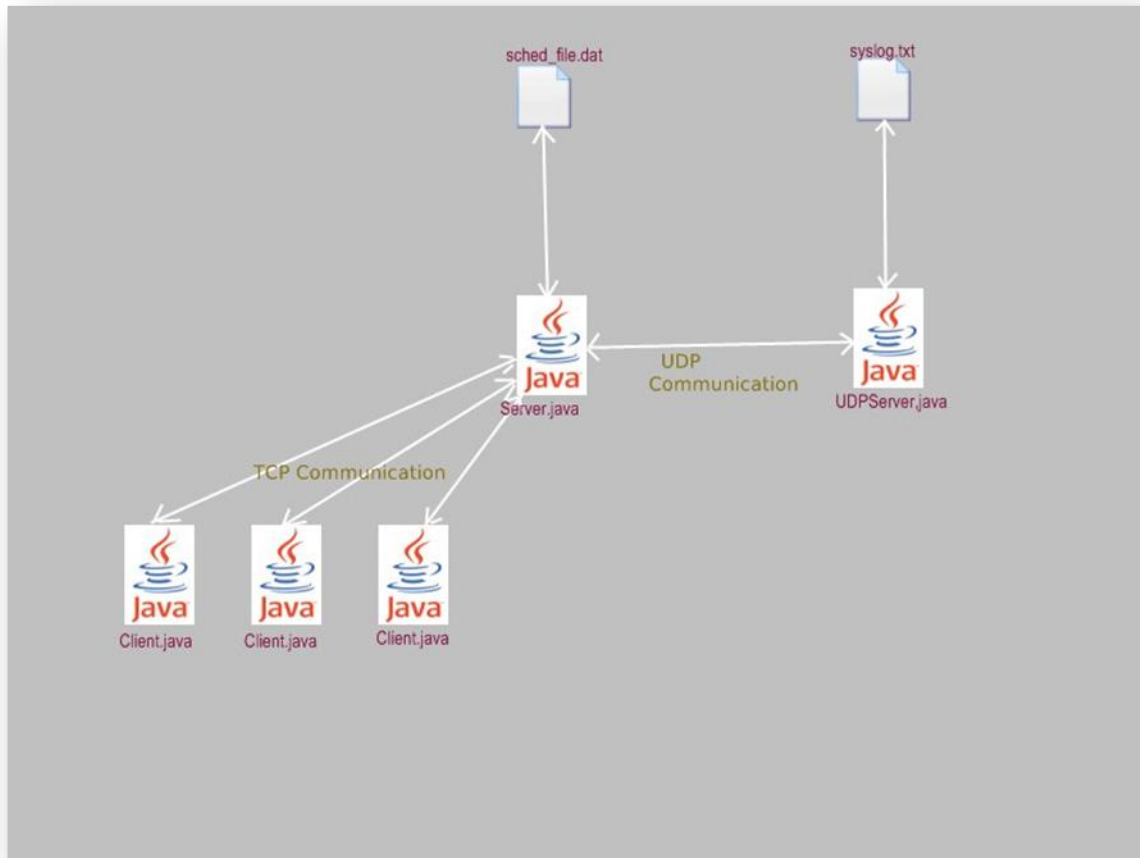


FIGURE 1: ARCHITECTURE OF THE CODE

EXECUTION STEPS AND RESULTS

1. Open three command line windows.
2. In one command line window, execute the following command at the appropriate folder structure as shown.

```
CHECKED-OUT-FOLDER/trunk/UDPServer>javac UDPServer.java
```

Similarly, execute the following commands at the respective folder structures as shown,

```
CHECKED-OUT-FOLDER/trunk/Server>javac Server.java
CHECKED-OUT-FOLDER/trunk/ClientUI>javac ClientUI.java
```

Ignore any warnings which say about deprecation. If you see any serious exceptions means that you did something wrong and need to do this again correctly. If you don't see any exceptions, your

command executed successfully. To verify, you can go and check for the class files in the respective folders wherever you executed the above commands.

3. Then execute the following commands one at appropriate folder structure as shown. Execute each of the below commands in a separate command windows.

```
CHECKED-OUT-FOLDER/trunk/UDPServer>java UDPServer
```

```
CHECKED-OUT-FOLDER/trunk/Server>java Server
```

Then you will see the following messages in the windows where your executed the above commands.

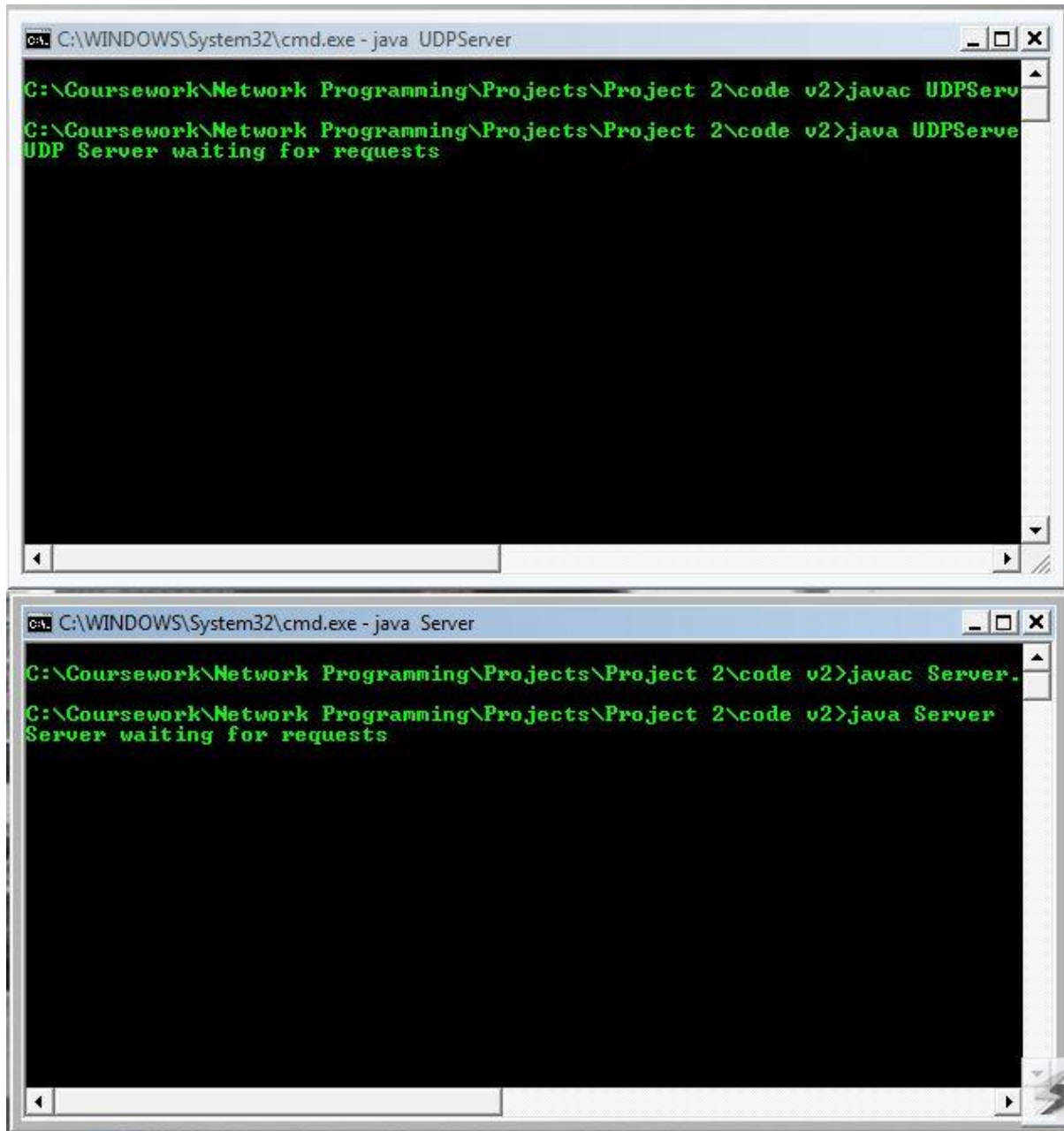


FIGURE 2: TERMINAL WINDOWS SHOWING MESSAGES AFTER EXECUTING THE RUN COMMAND ON COMPILED CLASS FILES

4. Then execute the following command in the third terminal window.

```
CHECKED-OUT-FOLDER/trunk/ClientUI>java ClientUI
```

Then you will see the following GUI which pops up on the screen. This is Appointment Manager window. Initially the appointment table looks blank as it is not connected to the server yet.

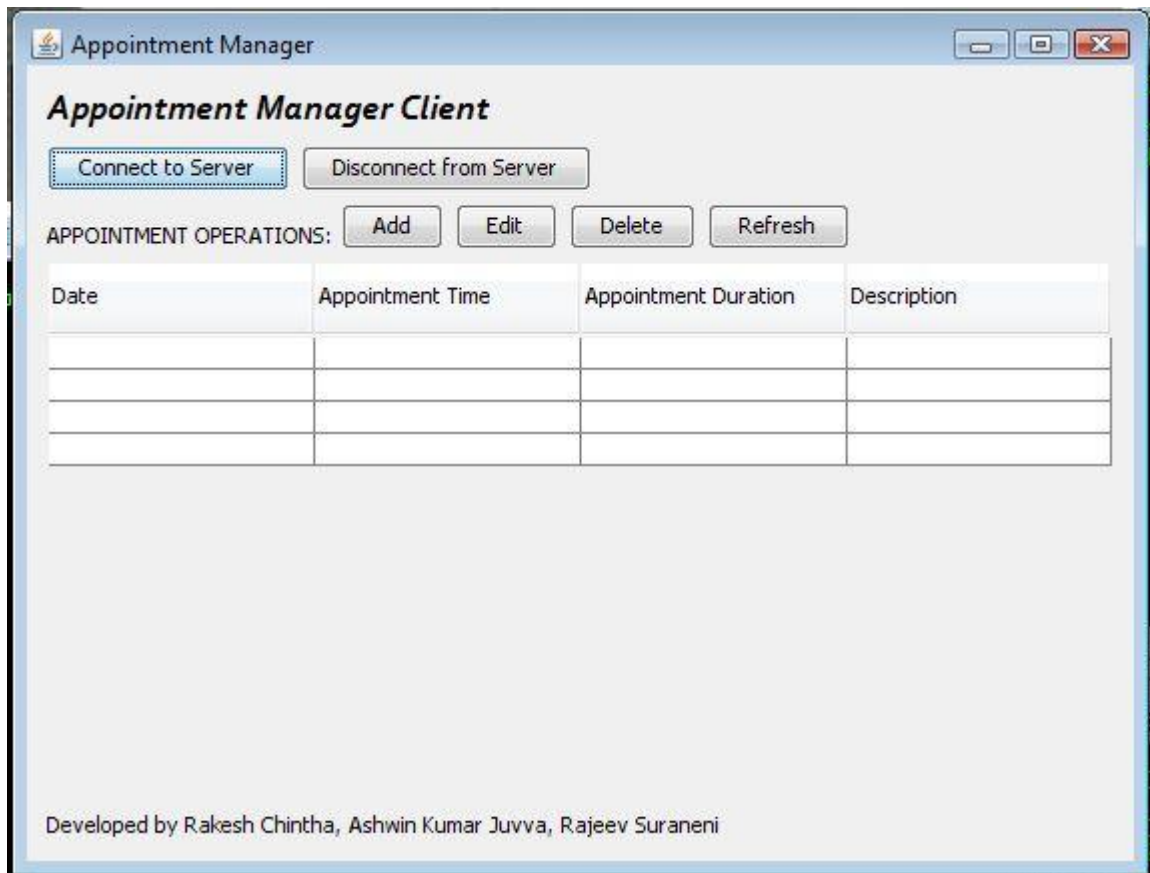


FIGURE 3: APPOINTMENT MANAGER CLIENT GUI

This GUI allows the client to perform operations like adding an appointment to the database, editing an appointment in the database and deleting an appointment from the database. The database that we are talking about here is simply a .DAT file that can be accessed only by the server.

- Now click the 'Connect to Server' button. As you click this button, appointment table gets populated with the current appointment on the database at the server side as shown in the image below.

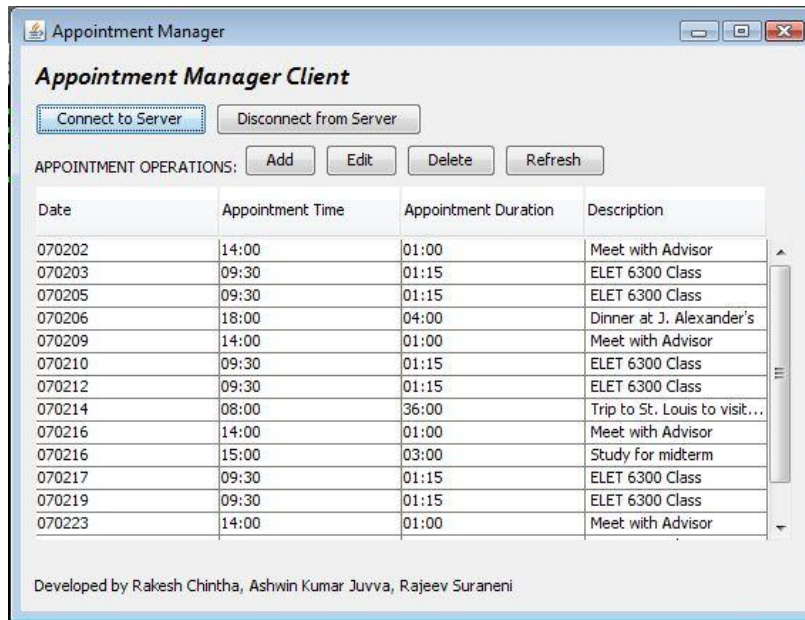


FIGURE 4: CLIENT GUI SHOWING THE ACTION OF 'CONNECT TO SERVER' BUTTON

- Now click 'Add' Button. As you click this button, a new dialog window titled 'Add Appointment' pops up as shown in the image below.

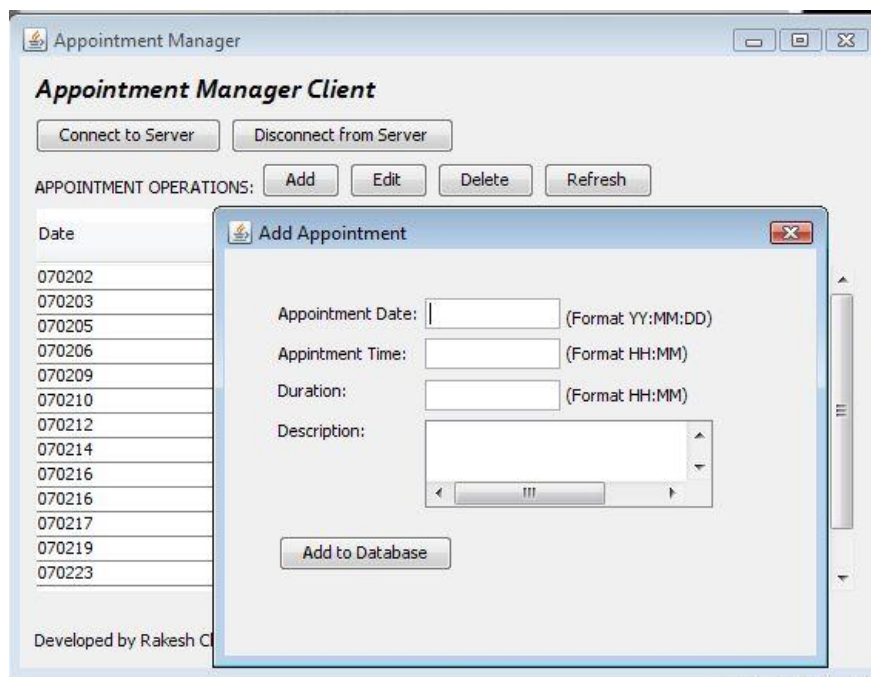


FIGURE 5: CLIENT GUI SHOWING THE ACTION OF 'ADD' BUTTON

7. If the user tries to enter the new appointment information which clashes with the already existing appointment, then a warning message window pops up with some message as shown in the images below,

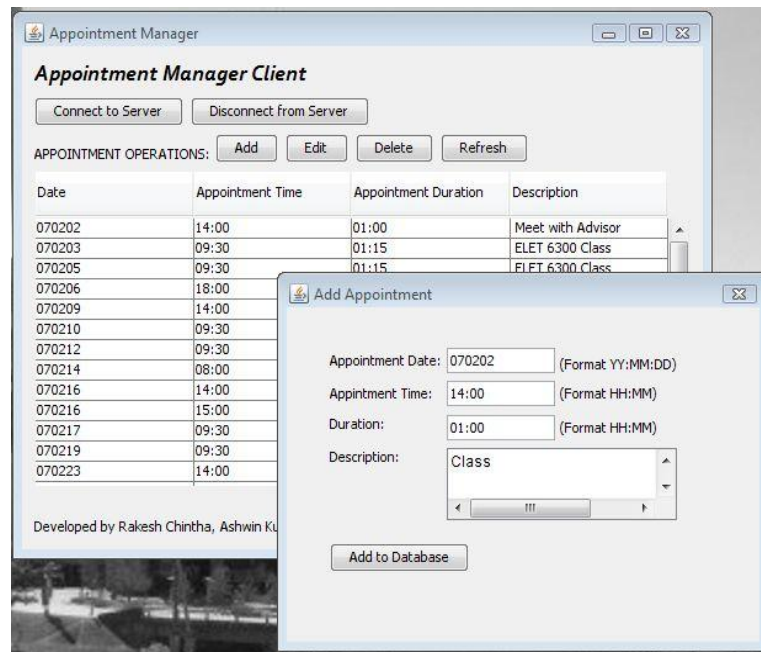


FIGURE 6: CLIENT GUI WITH NEW ADD APPOINTMENT INFORMATION WHICH ALREADY EXISTS

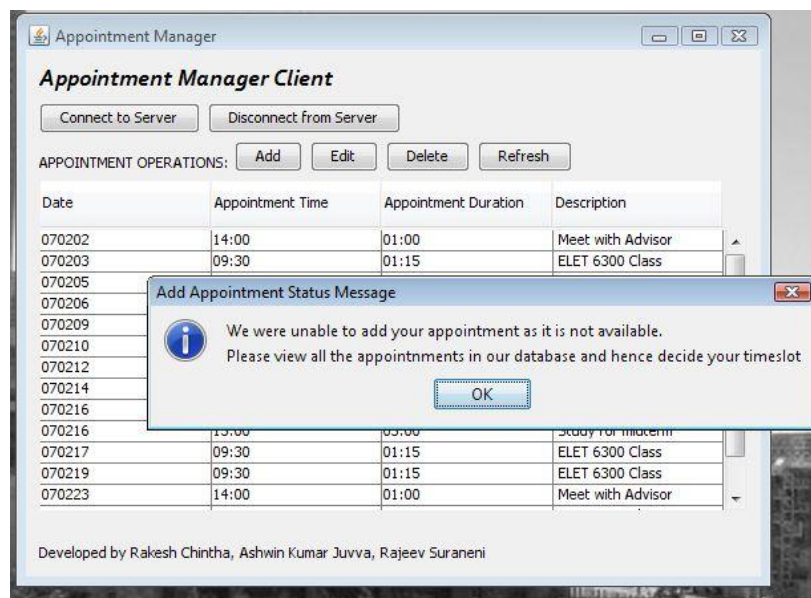


FIGURE 7: CLIENT GUI SHOWING A WARNING MESSAGE THAT THE REQUESTED APPOINTMENT CANNOT BE ADDED

8. If the user tries to enter a new appointment date information in wrong format, then a warning message window pops up with some message as shown in the images below,

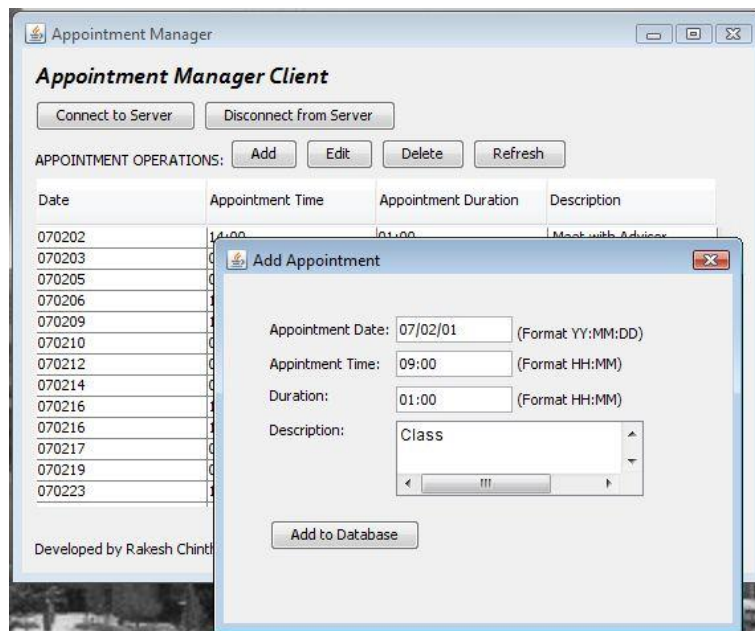


FIGURE 8: CLIENT GUI SHOWING THE NEW APPOINTMENT INFORMATION WITH DATE IN INVALID FORMAT

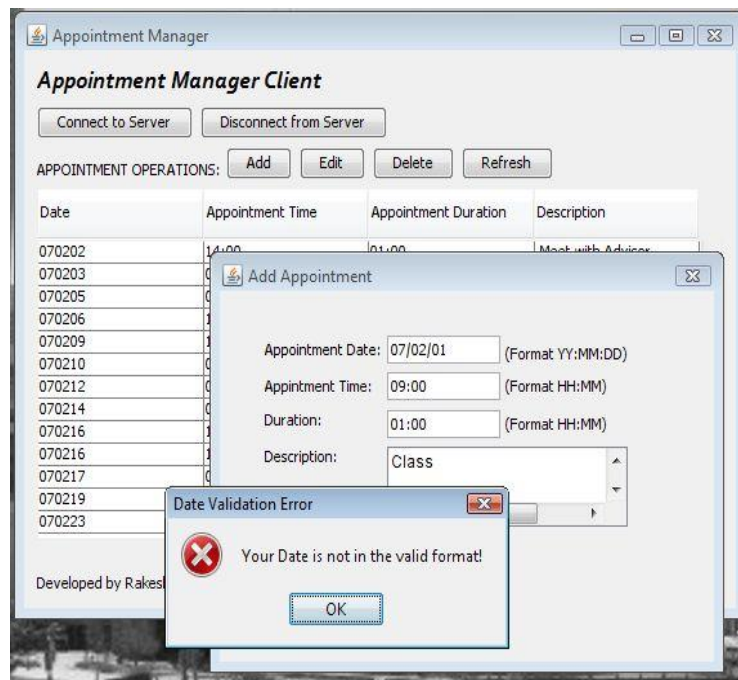


FIGURE 9: CLIENT GUI SHOWING A WARNING MESSAGE THAT THE USER ENTERED DATE VALUE IS INVALID

9. If the user tries to enter a new appointment time information in wrong format, then a warning message window pops up with some message as shown in the images below,

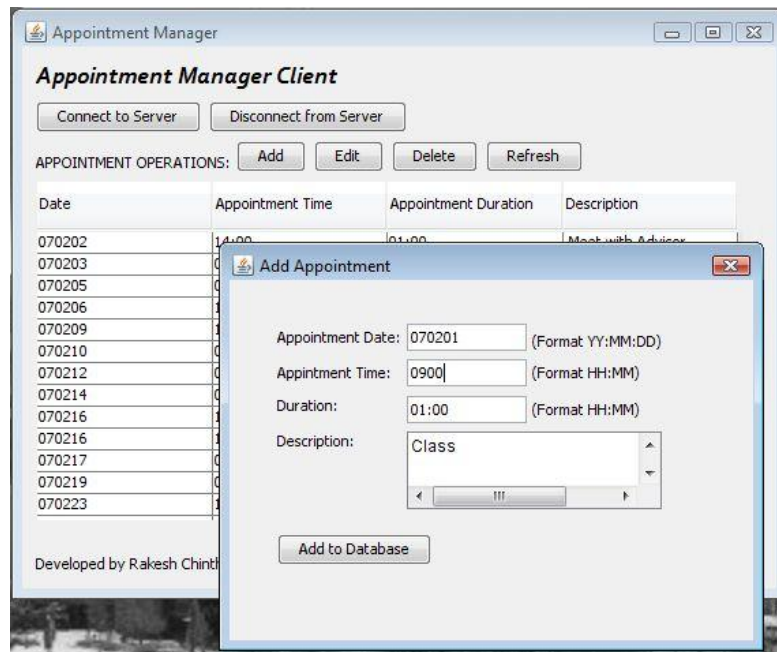


FIGURE 10: CLIENT GUI SHOWING THE NEW APPOINTMENT INFORMATION WITH TIME IN INVALID FORMAT

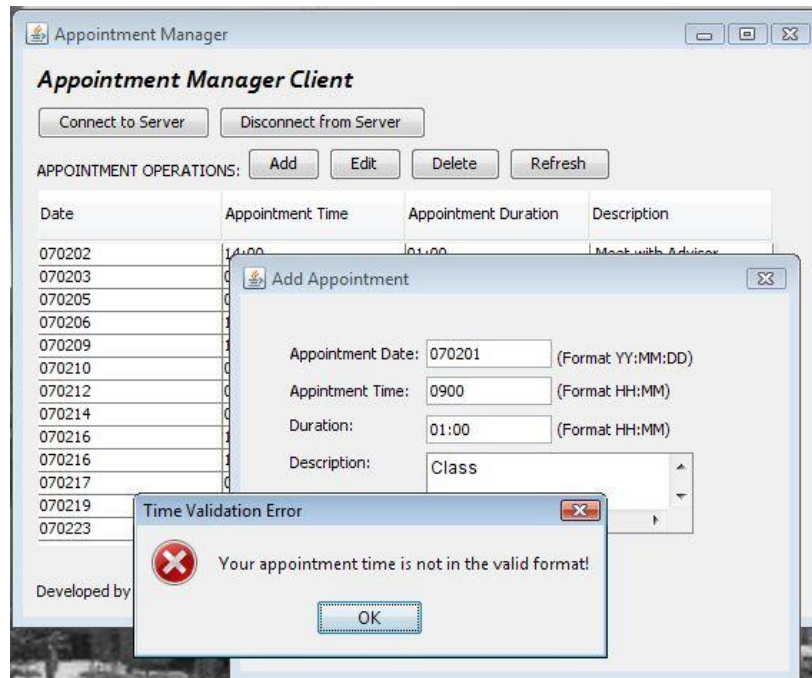


FIGURE 11: CLIENT GUI SHOWING A WARNING MESSAGE THAT THE USER ENTERED TIME VALUE IS INVALID

10. If the user tries to enter a new appointment duration information in wrong format, then a warning message window pops up with some message as shown in the images below,

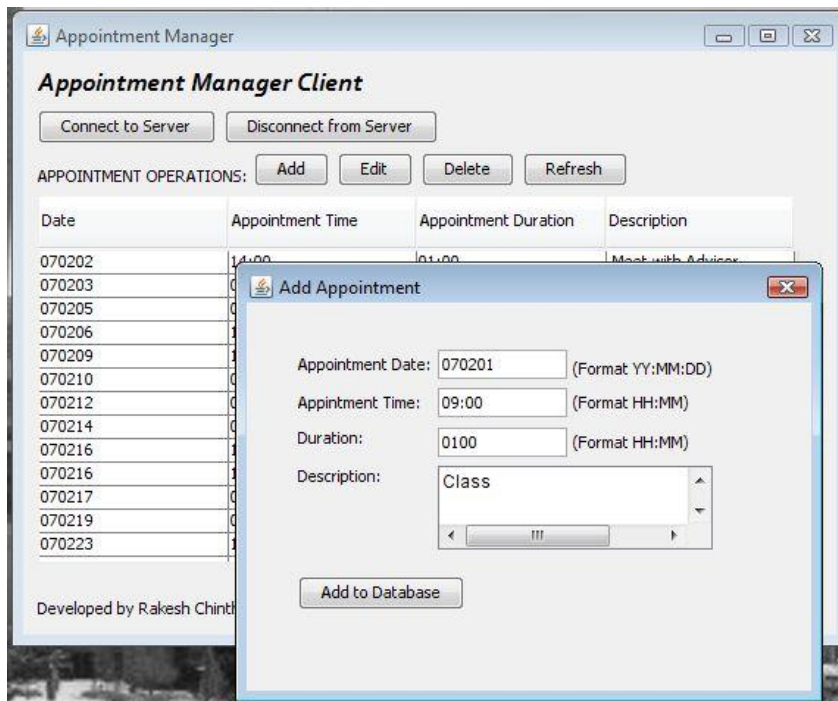


FIGURE 12: CLIENT GUI SHOWING THE NEW APPOINTMENT INFORMATION WITH DURATION IN INVALID FORMAT

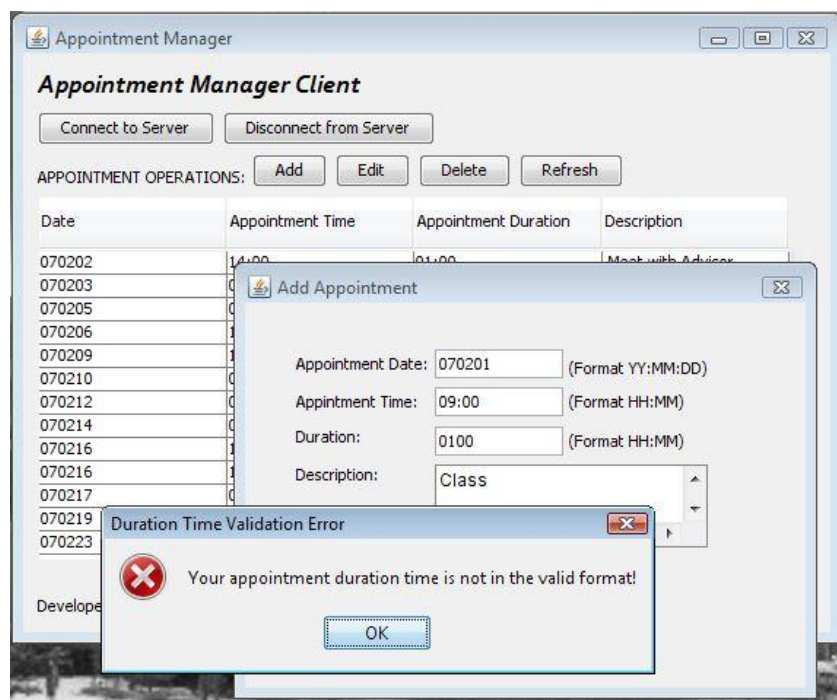


FIGURE 13: CLIENT GUI SHOWING A WARNING MESSAGE THAT THE USER ENTERED DURATION VALUE IS INVALID

11. If the user enters all the appointment information in the correct specified format and if it does not coincide with any other existing appointment, then it will be added and a small information message pops up saying that the operation was successful (shown in images below).

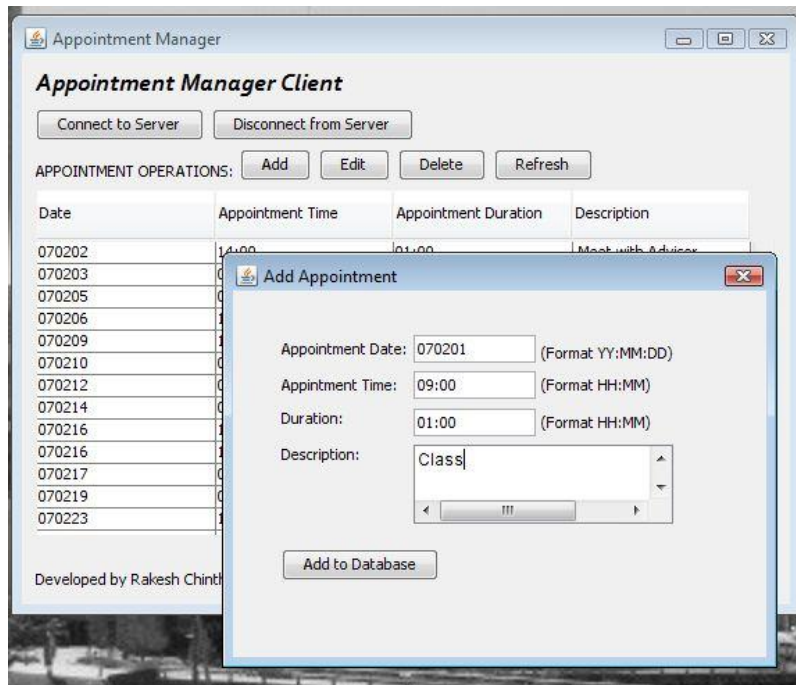


FIGURE 14: CLIENT GUI SHOWING THE CORRECT FOMRAT USER INPUT FOR ADDING AN APPOINTMENT

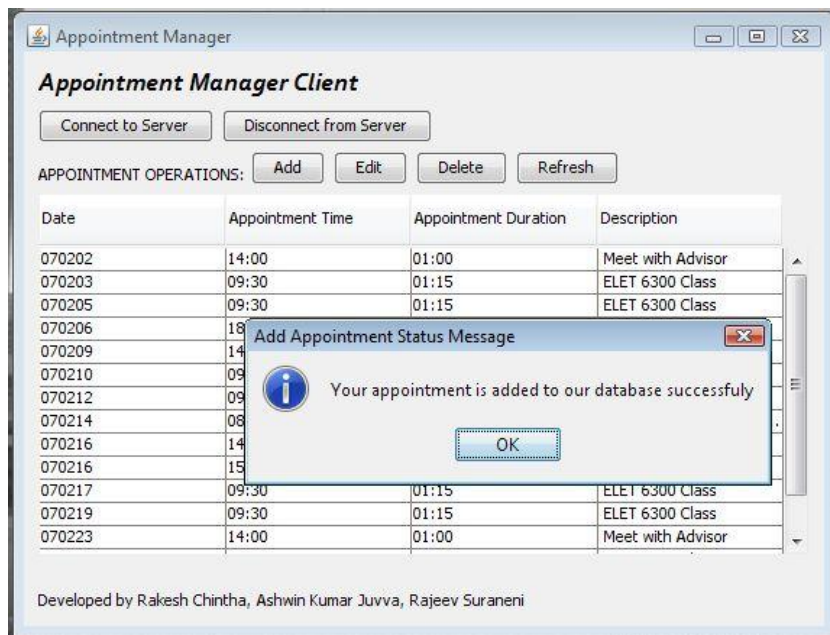


FIGURE 15: CLIENT GUI SHOWING THAT THE ADD APPOINTMENT OPERATION WAS SUCCESSFUL

12. If a user clicks 'edit' button even without selecting an appointment to edit, then an error pops up as shown in image below,

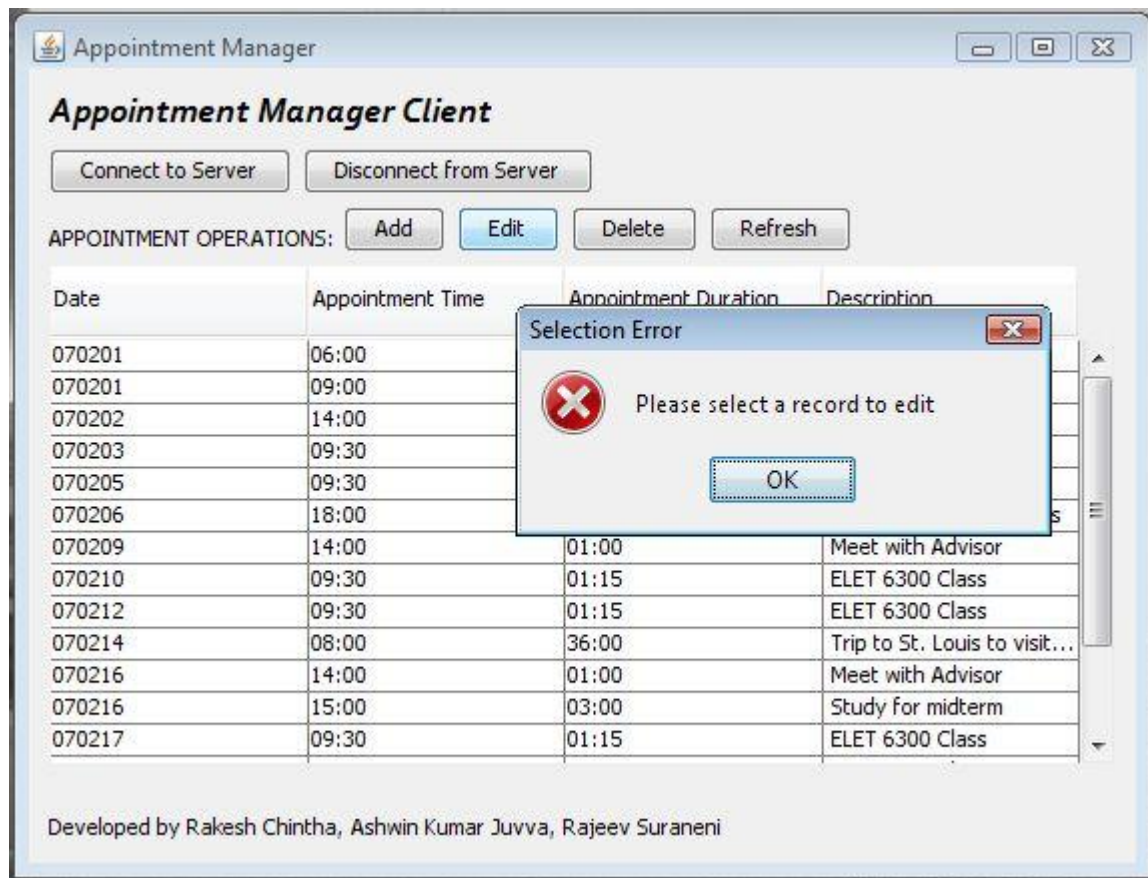


FIGURE 16: CLIENT GUI SHOWING AN ERROR MESSAGE WHILE EDIT OPERATION

13. If a user edits an appointment in such a way that it might coincide with another existing appointment, then such edit operation will be aborted with a warning message being popped up as shown in images below,

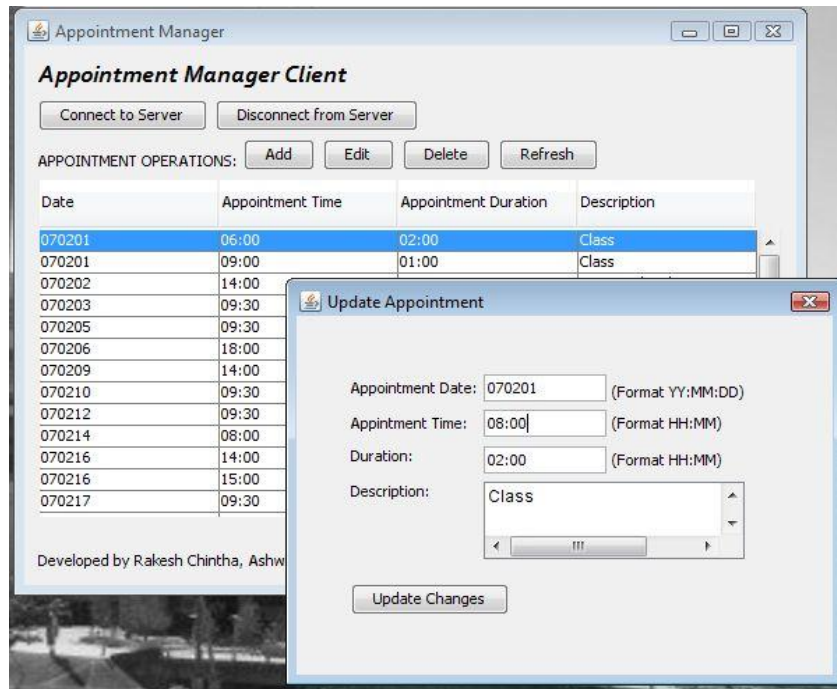


FIGURE 1: CLIENT GUI SHOWING THE EDIT OPERATION WHICH IS COINCIDING WITH THE EXISTING APPOINTMENT

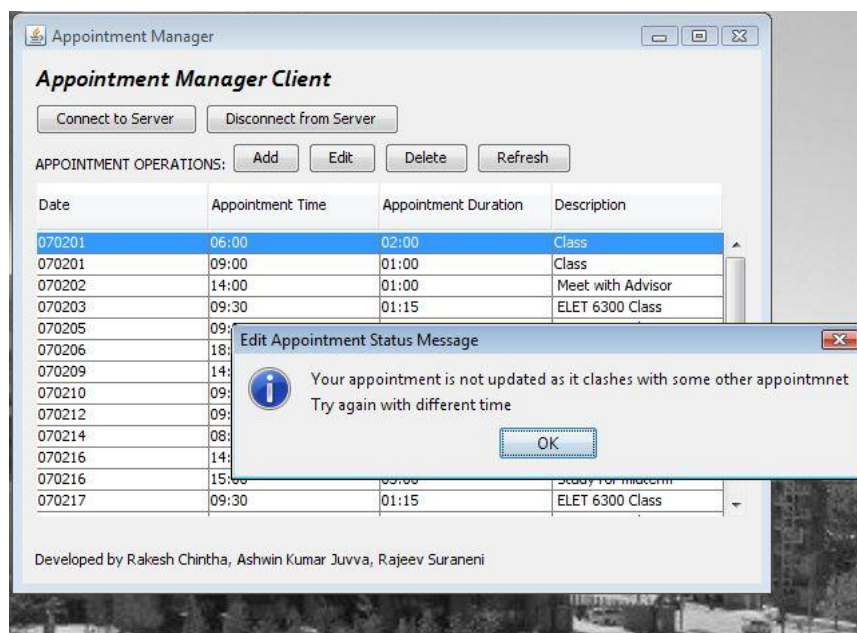


FIGURE 2: CLIENT GUI SHOWING A MESSAGE THAT THE EDIT OPERATION IS ABORTED

14. If a user selects an appointment and then clicks 'delete' button, then the selected appointment will be deleted from the database on the server side and after that a message pops up saying that the operation was successful.

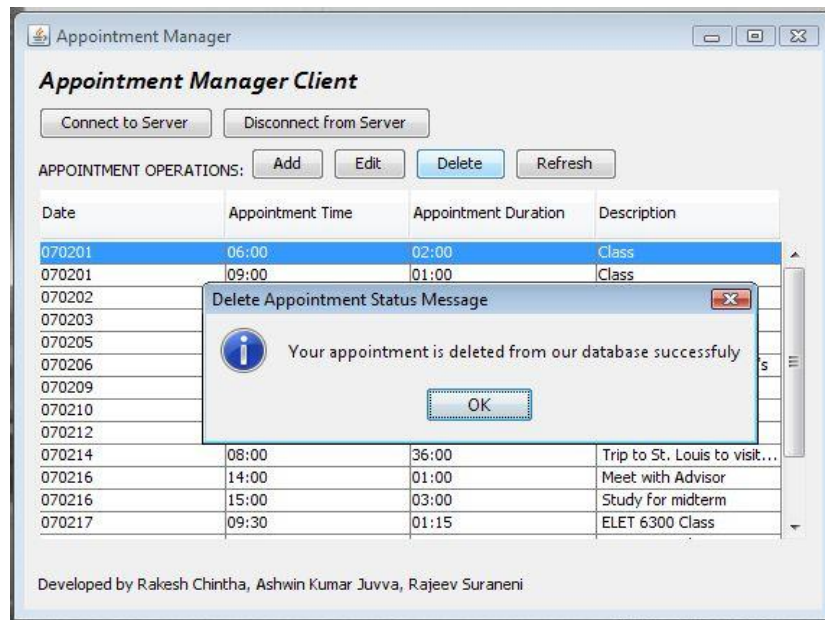


FIGURE 3: CLIENT GUI SHOWING THE DELETE OPERATION SUCCESS MESSAGE

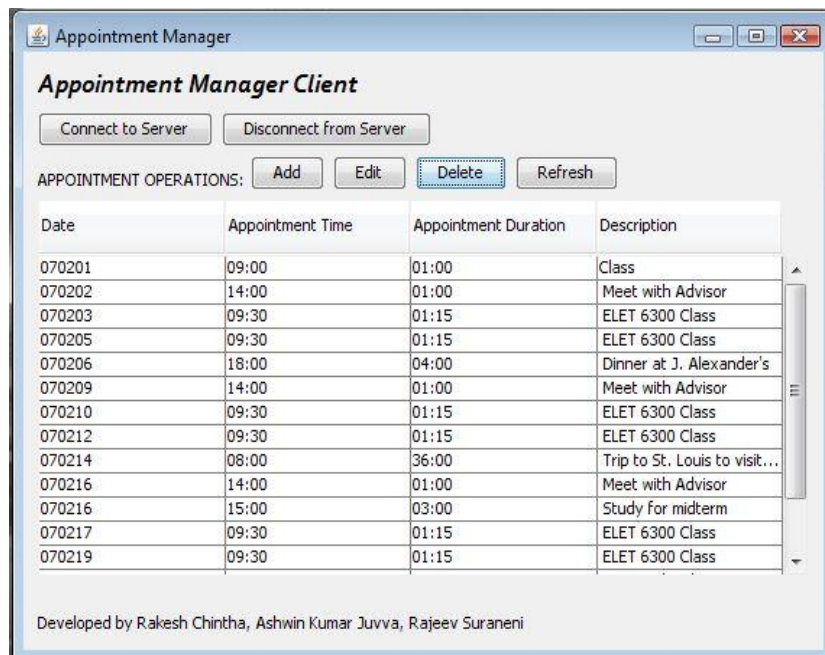


FIGURE 4: CLIENT GUI SHOWING THE UPDATED RESULTS AFTER THE DELETE OPERATION

15. If the user clicks 'Disconnect from Server' button, then client GUI disconnects from the server and the appointment table gets emptied as shown in the image below. At the server side, the thread allotted for the client services will be terminated (shown in the image below).

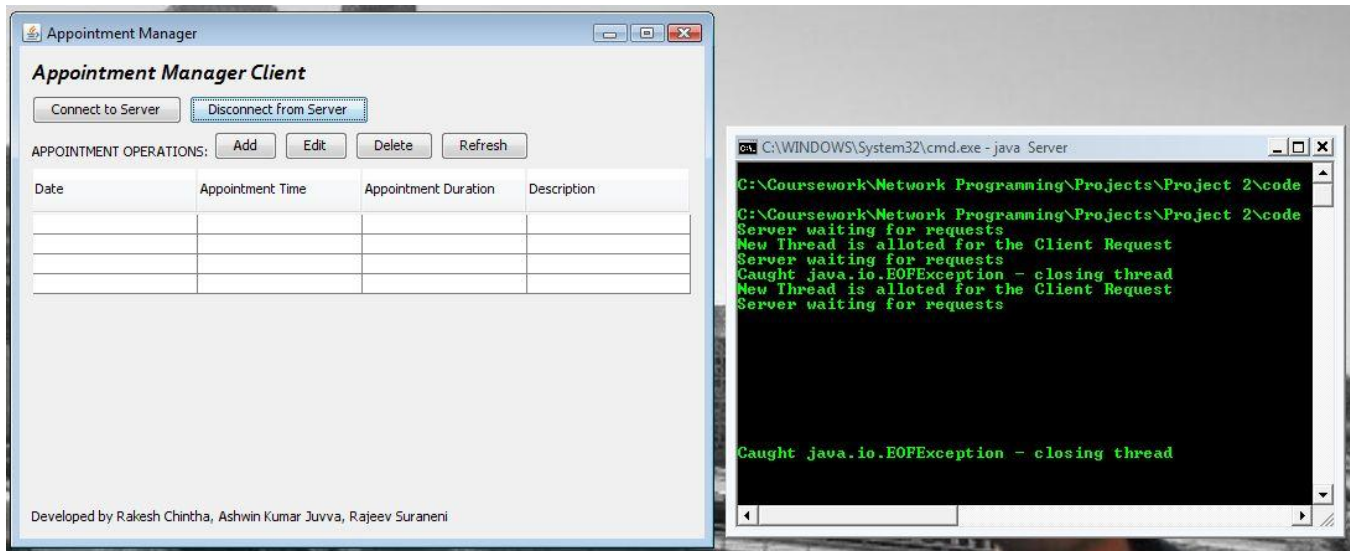


FIGURE 5: IMAGE SHOWING THE CLIENT GUI AND THE SERVER TERMINAL AFTER CLIENT GETS DISCONNECTED FROM THE SERVER

16. Apart from providing add, delete, edit operations on appointments for the client, Server also does the error reporting and console message reporting to the UDP Server which runs as a daemon service independent of the Server. The UDP Server writes the messages reported by the Server onto a syslog.txt file which is shown in the image below,

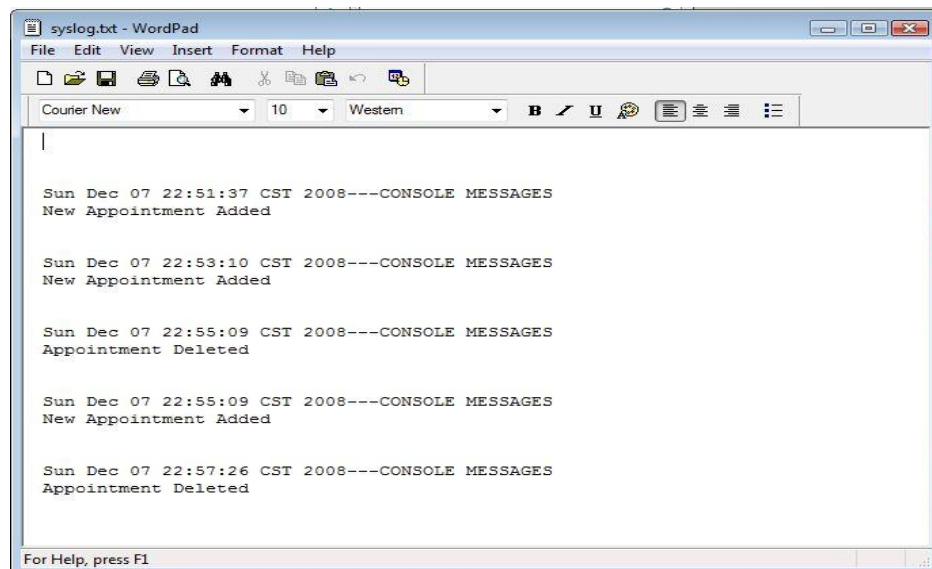


FIGURE 6: SYSLOG FILE AT THE UDP SERVER SIDE

CODE DOCUMENTATION

ClientUI.java is organized into following methods

```
public ClientUI()
```

This method creates new form ClientUI.

```
private void initComponents()
```

This method is called from within the constructor to initialize the form

```
private void  
addAppointmentButtonActionPerformed(java.awt.event.ActionEvent evt)
```

This method describes the action performed by the 'addAppointmentButton'. When this button is clicked, 'addAppointmentDialogPane' pops up which lets the user to input the appointment values

```
private void  
addToDatabaseButtonActionPerformed(java.awt.event.ActionEvent evt)
```

This method describes the action performed by the 'addToDatabaseButton'. When this button is clicked, the user entered new appointment information is sent to the Server side.

```
private void  
connectToServerButtonActionPerformed(java.awt.event.ActionEvent evt)
```

This method describes the action performed by the 'connectServerButton'. When this button is clicked, the Client connects to the Server running at port 4444. Also the appointment table is filled with the values.

```
private void  
deleteAppointmentButtonActionPerformed(java.awt.event.ActionEvent evt)
```

This method describes the action performed by the 'deleteAppointmentButton'. When this button is clicked, the selected appointment is deleted on the server side.

```
private void  
disconnectFromServerButtonActionPerformed(java.awt.event.ActionEvent  
evt)
```

This method describes the action performed by the 'disconnectFromServerButton'. When this button is clicked, the Client disconnects from the Server and also the appointment table is emptied.

```
private void  
editAppointmentButtonActionPerformed(java.awt.event.ActionEvent evt)
```

This method describes the action performed by 'editAppointmentButton'. When this button is clicked, 'editAppointmentDialogPane' pops up which lets the user to edit the selected appointment information.

```
private void refreshButtonActionPerformed(java.awt.event.ActionEvent evt)
```

This method describes the action performed by the 'refreshButton'. When this button is clicked, the updated results are displayed in the appointment table on the Client side.

```
private void  
updateDatabaseButtonActionPerformed(java.awt.event.ActionEvent evt)
```

This method describes the action performed by the 'updateDatabaseButton'. When this button is clicked, it the user updated information of the selected appointment is sent to the Server side.

```
public void updateTable(String dataFromServer)
```

This method updates the appointment table on the client side.

```
public boolean validateDate(String dateToBeValidated)
```

This method validates the user entered date value input in the form of string.

```
public boolean validateTime(String timeToBeValidated)
```

This method validates the user entered time value input in the form of string.

Server.java is organized into following methods.

```
public Server(int port) throws Exception
```

Constructor for the Server class.

ServerThread class extends Thread class and it is the inner class for the Server.java, It is organized into following methods.

```
public ServerThread(Socket socket) throws Exception
```

Constructor for the ServerThread class.

```
public boolean addAppointment(String userData)
```

This method takes the user input data as an argument, adds the appointment to the database and returns the boolean 'true' if added successfully.

```
public boolean checkAppointment(String userData)
```

This method takes the user input data as an argument, finds out if the user requested appointment is available or not and returns a boolean result. Depending on the appointment availability

```
public boolean deleteAppointment(String userData)
```

This method takes the user input data as an argument, deletes the record as requested by the user.

```
public String getRecord(int recordNumber) throws Exception
```

This method takes in an index as an argument and returns the record (appointment) at that index from the sorted database.

```
public String[] getRecordItems(String record) throws Exception
```

This method takes record(appointment) as an argument splits it into its individual items and returns it in the form of a String Array.

```
public String[] getRecords()
```

This method returns all the records(appointments) in the form of a String Array.

```
public int getRecordsNumber()
```

This method counts the number of records(appointments) in the database and returns the count.

```
public String matchRecord(String userData)
```

This method takes the user input data as an argument, matched the exact record and returns a match String which has an indication that a match was found and also the index at which it was found.

```
public int reportToUDPServer(String message) throws Exception
```

This method sends console and error messages to the UDP Server which is running as a separate program.

```
public void run()
```

Execution of the ServerThread takes place here.

```
public String sortRecords(String newRecord)
```

This method takes in a String of new data (after adding new appointment) and sorts the records.

```
public void updateDatabase(String newData)
```

This method takes a string of new data after adding appointments and the after the data has been sorted this method takes a string of new data after adding appointments and the after

the data has been sorted updates the file contents and returns the boolean result accordingly. Whenever updateDatabase method is used by a ServerThread instance, it checks if the lock variable is '1'. If it is 1, that means, some other ServerThread instance is updating the database. So, it waits till the lock becomes 0. Whenever lock is '0', the current ServerThread will have the write access to the database.

```
public String viewAppointments()
```

This method returns all the appointments in the database in the form of a String.

UDPServer.java is organized into following methods,

```
public UDPServer(int port) throws Exception
```

Constructor for the UDPServer

UDPServerThread class extends Thread class and it is organized into methods.

```
public UDPServerThread(String data) throws Exception
```

Constructor for UDPServerThread

```
public void run()
```

Execution of the UDPServerThread takes place here

```
public int consoleMessageReporter(Date current, String message)
```

This method takes in the current date with time and the console message as parameters and writes them into the syslog.txt.

```
public int errorMessageReporter(Date current, String message)
```

This method takes in the current date with time and the error message as parameters and writes them into the syslog.txt