



King Abdul-Aziz university
Faculty of Computing and Information Technology
Computer Science Department

February 13th, 2023

CPCS-371: Computer Networks



Simulating a Remote Health Monitoring System (RHMS) for elderly patients using client-server TCP Sockets

Instructors: Dr. Etimad Fadel and Dr.Ohoud Alzamzami

ID	Name	Section
2005230	Hasna Othman Bukhari	B1
2006870	Amani Khalid Biraik	B1
2006207	Jetana Bassim Abudawood	B1
2005887	Shaden Dhafer Alhashysh	B1
2005341	Logain Ezzat Sendi	B2
2007057	Wajd Bandar Alharbi	B1

Table of Contents

1	Introduction.....	4
1.1	Client-server applications	4
1.2	Java TCP sockets	4
1.3	Selected GUIs elements	4
1.4	RHMS application	5
1.5	The role of the clients and servers in RHMS application	5
1.6	Overview.....	5
2	Patient Monitoring Application interaction diagram	6
2.1	Interaction diagram	6
3	RHMS implementation	7
3.1	Sensor (Client) Application	7
3.2	Personal Server	9
3.3	Medical Server	12
3.4	Where are TCP sockets used?.....	14
3.5	Where are threads used? For what reason are they used?.....	14
3.6	What are the functions used for sending and receiving information using sockets?	14
3.7	What are the GUIs elements used in your application?	14
4	RHMS Application runs snapshots.....	15
4.1	run of the program on one machine	15
4.2	run of the program on different machines.....	17
5	Teamwork and Lessons learned	20
6	Conclusion	20
7	References.....	21

Table of Figures

Figure 1: run in GUI 3 classes	15
Figure 2: run in Sensor console	15
Figure 3:run in Personal Server console	16
Figure 4:run in Medical Server console.....	16
Figure 5:run in Medical Server console on the first device	17
Figure 6:run in Personal Server console on the first device	18
Figure 7:run in Sensor console on the second device	18
Figure 8:run in Personal and Medical Servers GUI on the first device	19
Figure 9:run in Sensor GUI on the second device	19

1 Introduction

1.1 Client-server applications

The client-server application consists of two parts which involve a client and a server. The client sends a message request, and the server responds with a message containing the services that were requested. This procedure is accomplished using a request-response pattern. Additionally, a communication protocol between the client and server is required to ensure the success of this procedure. Beyond that, the client and server can either run on the same computer or other computers and connect through a network.

1.2 Java TCP sockets

Transmission Control Protocol (TCP) is a transport layer protocol that provides a connection-oriented reliable data transfer. The sensor client application functions as a client in this program and uses TCP sockets to transfer data to the personal server. The personal server, however, has both a client and a server role. Prior to sending the data to the medical server, the personal server works as a server and accepts the data from the sensor client application. If the data was unusual, it would be forwarded to the medical server, where the personal server would act as the client, open a new TCP socket, and make the connection.

1.3 Selected GUIs elements

In our application, we used the java swing GUI application to create the interfaces for all applications (sensor client application, personal server, and medical server). We used labels, text fields, text areas, and a button.

1.4 RHMS application

A Remote health monitoring system (RHMS) is a system that is thoroughly used in the health department to solve medical issues, mostly used with elderly patients. Three distinct sensors in this device can detect oxygen saturation, heart rate, and temperature. Every five seconds, the sensor senses data, and if a problem or abnormal state arises, the personal server must alert the medical server so that it may take the necessary steps to save the patient's life.

1.5 The role of the clients and servers in RHMS application

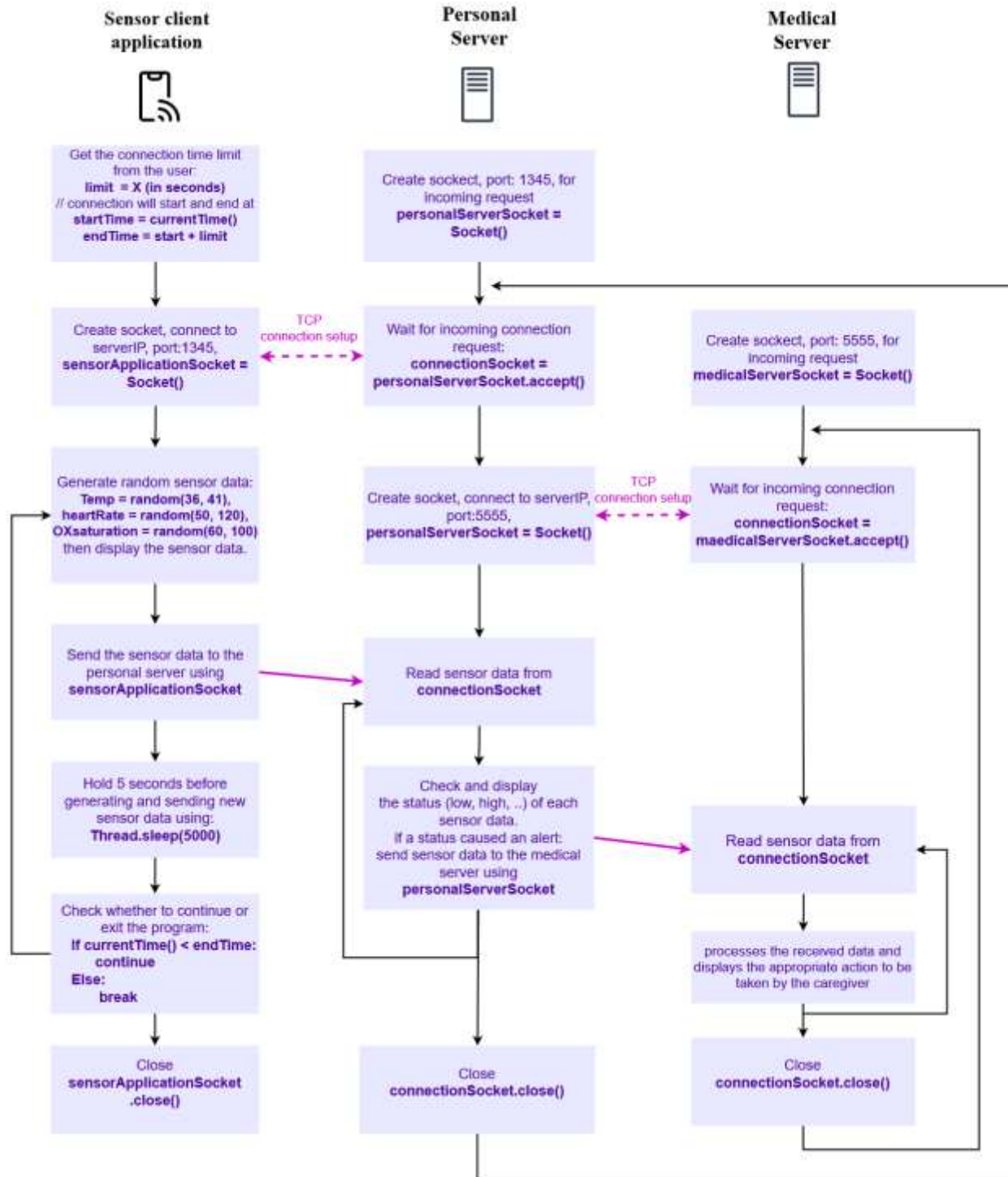
The sensor client application acts as a client in the RHMS application. It generates and collects the user's medical data every five seconds through sensors and sends them to the personal server using a TCP connection that is open for a minimum of 60 seconds. The personal server acts as both a client and a server. Its role is to receive the data from the sensor client application and process it to figure out whether the data sent is normal or abnormal (here, the personal server acts as a server). If the data is anomalous, a message is sent to the medical server (here, the personal server acts as a client). The medical server acts as a server only. Its role is to receive the messages sent from the personal server and display them to the caregiver while also processing the message to decide what action needs to be taken.

1.6 Overview

Section two presents the interaction diagram and pseudocode for the RHMS application. In section three, the implementation of our projection is displayed and explained. Section four, snapshots of the RHMS application being run on one machine and two machines, will be shown with details explaining each. Section five is about how the project was divided, what each girl did, and the lessons learned are mentioned. And finally, in section six, the conclusion will be presented.

2 Patient Monitoring Application interaction diagram

2.1 Interaction diagram



3 RHMS implementation

3.1 Sensor (Client) Application

```
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.net.Socket;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Scanner;
import java.util.concurrent.ThreadLocalRandom;

public class Sensor_client_application {

    // initiate variables
    static int heartRateSensor = 0;
    static double TempSensorDouble = 0;
    static String TempSensorString = "";
    static int O2Sensor = 0;
    static int limit = 0;
    static Socket socket = null;
    static long startTime = 0;
    static long endTime = 0;

    public static void main(String[] args) throws IOException, InterruptedException {
        Scanner input = new Scanner(System.in);

        // ask the user to enter the desired running time of the program
        System.out.print("Enter connection time wanted in seconds, 60s minimum: ");
        limit = input.nextInt();

        // the minimum time is 60 seconds, therefore, check if the user entered a time less than 60s
        if (limit < 60) {
            limit = 60; }

        // determine the start and end time of the program
        startTime = System.currentTimeMillis();
        endTime = startTime + limit * 1000;

        // create the client TCP socket
        socket = new Socket("localhost", 1345);

        // for sending the data to the personal server
        OutputStreamWriter output = new OutputStreamWriter(socket.getOutputStream());
        BufferedWriter writer = new BufferedWriter(output);

        while (System.currentTimeMillis() < endTime) {
```

```
// generate random data
// Heart rate sensor is between 50 and 120 heart beats each minute
heartRateSensor = Data(50, 120);
// Temperature sensor data is between 36 and 41
TempSensorDouble = DataDouble(36, 41);
// Oxygen level in the blood data is sent by the sensor with values between 60 to 100
O2Sensor = Data(60, 100);
// formatting the date and time
SimpleDateFormat formatter = new SimpleDateFormat("dd MMM yy 'time' HH:mm:ss");
Date date = new Date();

// converting the double value of temperature to a string value
TempSensorString = String.valueOf(TempSensorDouble);
// take only one digit after the decimal point
TempSensorString = TempSensorString.substring(0, 4);

// print data
System.out.println("");
System.out.println("At date: " + formatter.format(date) + ",sensed temperature is " + TempSensorString);
System.out.println("At date: " + formatter.format(date) + ",sensed heart rate is " + heartRateSensor);
System.out.println("At date: " + formatter.format(date) + ",sensed oxygen saturation is " + O2Sensor);

// send the data to the personal server
writer.write(heartRateSensor);
writer.write(O2Sensor);
writer.write(TempSensorString);
writer.newLine();
writer.flush();

// the program must hold for a 5s before the next iteration
Thread.sleep(5000); }

socket.close(); }

// this method generates an integer random number between the two values min and max
public static int Data(int min, int max) {

int random_int = (int) Math.floor(Math.random() * (max - min + 1) + min);

return random_int; }

// this method generates a double random number between the two values min and max
public static double DataDouble(int min, int max) {
return (ThreadLocalRandom.current().nextDouble() * (max - min)) + min; } }
```


3.2 Personal Server

// personal server: receives sensed data from the client application, then checks the user's medical status and send data to medical server if needed.

```
package personal_server;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Personal_server {

    public static void main(String[] args) throws IOException, InterruptedException {

        // create the personal server TCP socket
        ServerSocket serverSocket = new ServerSocket(1345);
        SimpleDateFormat formatter = new SimpleDateFormat("dd MMM yy 'time' HH:mm:ss");

        //1st loop
        while (true) { // the server always on

            // accept the socket CONNECTION
            Socket socket = serverSocket.accept();

            // medical server socket
            Socket medicalSocket = new Socket("localhost", 5555);

            // to write data to medical server
            OutputStreamWriter medicalOutput = new OutputStreamWriter(medicalSocket.getOutputStream());
            BufferedWriter medicalWriter = new BufferedWriter(medicalOutput);

            // initiate variables
            double receiveTemp;
            String TempString = "";
            int receiveHeartRate;
            int receiveOxygenSaturation;

            // an alert variable is used to determine whether to send data to medical server or not
            boolean alert = false;

            // 2nd loop
            while (true) {
```

```
// to read from the client application class
```

```
InputStreamReader input = new InputStreamReader(socket.getInputStream());
BufferedReader reader = new BufferedReader(input);
```

```
// read data from client application class
```

```
receiveHeartRate = reader.read();
receiveOxygenSaturation = reader.read();
TempString = reader.readLine();
```

```
// if the client application is terminated then the default data to be send is -1, so this method will check whether
to end the connection or not
```

```
if (receiveHeartRate == -1) {
```

```
    alert = true; break; }
```

```
// make sure to take only one digit after the decimal point of the temperature value
```

```
TempString = TempString.substring(0, 4);
```

```
// convert the String value of temperature to a double value
```

```
receiveTemp = Double.valueOf(TempString);
```

```
Date date = new Date();
```

```
// check temperature
```

```
if (checkTemp(receiveTemp)) {
```

```
    System.out.println("At date: " + formatter.format(date) + ", Temperature is normal");
```

```
    } else {
```

```
        alert = true;
```

```
        System.out.println("At date: " + formatter.format(date) + ", Temperature is high " + receiveTemp + ". An alert
message is sent to the Medical Server."); }
```

```
//-----
```

```
// check heart rate
```

```
String heartRate = checkHeartRate(receiveHeartRate);
```

```
if (heartRate.equalsIgnoreCase("Above")) {
```

```
    alert = true;
```

```
    System.out.println("At date: " + formatter.format(date) + ", Hear rate is above normal " + receiveHeartRate +
". An alert message is sent to the Medical Server.");
```

```
    } else if (heartRate.equalsIgnoreCase("Below")) {
```

```
        alert = true;
```

```
        System.out.println("At date: " + formatter.format(date) + ", Hear rate is below normal " + receiveHeartRate +
". An alert message is sent to the Medical Server.");
```

```
    } else {
```

```
        System.out.println("At date: " + formatter.format(date) + ", Hear rate is normal"); }
```

```
//-----  
// check oxygen saturation  
if (checkOxygenSaturation(receiveOxygenSaturation)) {  
  
System.out.println("At date: " + formatter.format(date) + ", Oxygen Saturation is normal");  
  
} else {  
  
alert = true;  
  
System.out.println("At date: " + formatter.format(date) + ", Oxygen Saturation is low " +  
receiveOxygenSaturation + ". An alert message is sent to the Medical Server."); }  
//-----  
  
System.out.println("");  
// check if the alert is true to send data to medical server  
if (alert) {  
medicalWriter.write(1);  
medicalWriter.write(receiveHeartRate);  
medicalWriter.write(receiveOxygenSaturation);  
medicalWriter.write(TempString);  
medicalWriter.newLine();  
medicalWriter.flush(); }  
// the program must hold for a 5s before the next iteration  
Thread.sleep(5000); }//end of 2nd loop  
  
medicalWriter.write(1); socket.close(); }// end of 1st loop }  
  
public static boolean checkTemp(double temp) {  
  
if (temp > 37) {  
return false;  
} else {  
return true; } }  
  
// check heart rate method  
public static String checkHeartRate(int heartRate) {  
  
if (heartRate > 100) {  
return "Above";  
} else if (heartRate < 60) {  
return "Below";  
} else {  
return "Normal"; } }  
  
// check oxygen saturation method  
public static boolean checkOxygenSaturation(int oxygenSaturation) {  
  
if (oxygenSaturation < 75) {  
return false;  
} else {  
return true; } } }
```

3.3 Medical Server

// medical server: receives the data from the personal server to process abnormal cases.

```
package medical_server;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.ServerSocket;
import java.net.Socket;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Medical_server {

    public static void main(String[] args) throws IOException, InterruptedException {

        // create the medical server TCP socket
        ServerSocket MedicalServerSocket = new ServerSocket(5555);

        SimpleDateFormat formatter = new SimpleDateFormat("dd MMM yy 'time' HH:mm:ss");
        //1st loop
        while (true) { //the server always on

            Socket socket = MedicalServerSocket.accept(); // accept the socket

            // initiate the variables
            int alert = -1;
            double receiveTemp;
            String TempString = "";
            int receiveHeartRate;
            int receiveOxygenSaturation;

            while (true) {

                // to read from the personal server class
                InputStreamReader medicalInput = new InputStreamReader(socket.getInputStream());
                BufferedReader medicalReader = new BufferedReader(medicalInput);

                // if there is an alert from the personal server this value must be read
                alert = medicalReader.read();

                // if the personal server detect any abnormal condition the alert will be send to the medical server (this server)
                if (alert == 1) {

                    // read the temperature, heart rate, and oxygen saturation that sent by the personal server
                    receiveHeartRate = medicalReader.read();
                    receiveOxygenSaturation = medicalReader.read();
                    TempString = medicalReader.readLine();
                }
            }
        }
    }
}
```

```
// if the client application is terminated then the default data to be send is -1, so this method will check whether to end the connection or not
```

```
if (receiveHeartRate == -1) {  
    break;  
}
```

```
//take only one digit after the decimal point of the temperature value
```

```
TempString = TempString.substring(0, 4);
```

```
// convert the String value of temperature to a double value
```

```
receiveTemp = Double.valueOf(TempString);  
Date date = new Date();
```

```
// If the temperature exceeds 39 and heart rate is above 100 and oxygen is below 95
```

```
if ((receiveTemp > 39) && (receiveHeartRate > 100) && (receiveOxygenSaturation < 95)) {  
    System.out.println("At date: " + formatter.format(date) + ", Temperature is high " + receiveTemp + ".");  
    System.out.println("At date: " + formatter.format(date) + ", Hear rate is above normal " + receiveHeartRate + ".");  
    System.out.println("At date: " + formatter.format(date) + ", Oxygen Saturation is low" +  
receiveOxygenSaturation + ".");  
    System.out.println("ACTION: Send an ambulance to the patient!");  
    System.out.println("");
```

```
} // If the temperature is between 38 and 38.9, and the heart rate is between 95 and 98, and oxygen is below 80
```

```
else if (((receiveTemp >= 38) && (receiveTemp <= 38.9)) && ((receiveHeartRate >= 95) &&  
(receiveHeartRate <= 98)) && (receiveOxygenSaturation < 80)) {  
    System.out.println("At date: " + formatter.format(date) + ", Temperature is high " + receiveTemp + ".");  
    System.out.println("At date: " + formatter.format(date) + ", Hear rate is normal " + receiveHeartRate + ".");  
    System.out.println("At date: " + formatter.format(date) + ", Oxygen Saturation is low" +  
receiveOxygenSaturation + ".");  
    System.out.println("ACTION: Call the patient's family!");  
    System.out.println("");
```

```
} // default message for the caregiver
```

```
else {  
    System.out.println("At date: " + formatter.format(date) + ", Temperature is " + receiveTemp + ".");  
    System.out.println("At date: " + formatter.format(date) + ", Hear rate is " + receiveHeartRate + ".");  
    System.out.println("At date: " + formatter.format(date) + ", Oxygen Saturation is " + receiveOxygenSaturation  
+ ".");  
    System.out.println("ACTION: Warning, advise patient to make a checkup appointment!");  
    System.out.println(""); } }
```

```
// the program must hold for a 5s before the next iteration
```

```
Thread.sleep(5000); }
```

```
// closing the connections
```

```
socket.close(); } }
```

3.4 Where are TCP sockets used?

TCP sockets are used in the client sensor application, to connect to the servers. TCP sockets are also used in the personal and medical servers to listen to the connection, as well as request and send messages. Personal server sends messages to the medical server through the TCP socket, and the medical server receives those messages through its TCP socket. (colored blue in the code comments above)

3.5 Where are threads used? For what reason are they used?

A program's single sequential control flow is known as a thread. A single thread is how most programs operate. We may run many processes simultaneously by using Java's multithreading feature, which enables the server to support multiple clients, by accepting a client connection, opening a thread for that conversation, and keeping an eye out for requests from other clients.

3.6 What are the functions used for sending and receiving information using sockets?

(Colored pink in code comments above)

1) OutputStreamWriter and BufferedWriter 2) writer.write(data)	In client sensor for sending the data to the personal server, and in personal for sending the data to the medical server
1) InputStreamReader and BufferedReader 2) reader.read(data)	in personal server for reading data from sensor client application class, and in medical server for reading data from personal server

3.7 What are the GUIs elements used in your application?

- Labels
- Text field
- Button
- Text Area

// this methos is responsible for taken the action of the input after the user clicks it

private void EnterActionPerformed(java.awt.event.ActionEvent evt)

// this line is used to get the entered input from the user

limit = Integer.parseInt(jTextField1.getText());

if (limit < 60) {

limit = 60; }

// is to print the result into the text area as it appears in Image1

object.append

4 RHMS Application runs snapshots.

4.1 run of the program on one machine

1. the user will enter the required time in the text field.
2. The Sensor will start to send the information every 5 seconds to Personal server as it's appearing in Image 1
3. The personal server will print the result and send the alert to the medical server.
4. The Medical server will output the result and print it on the screen.

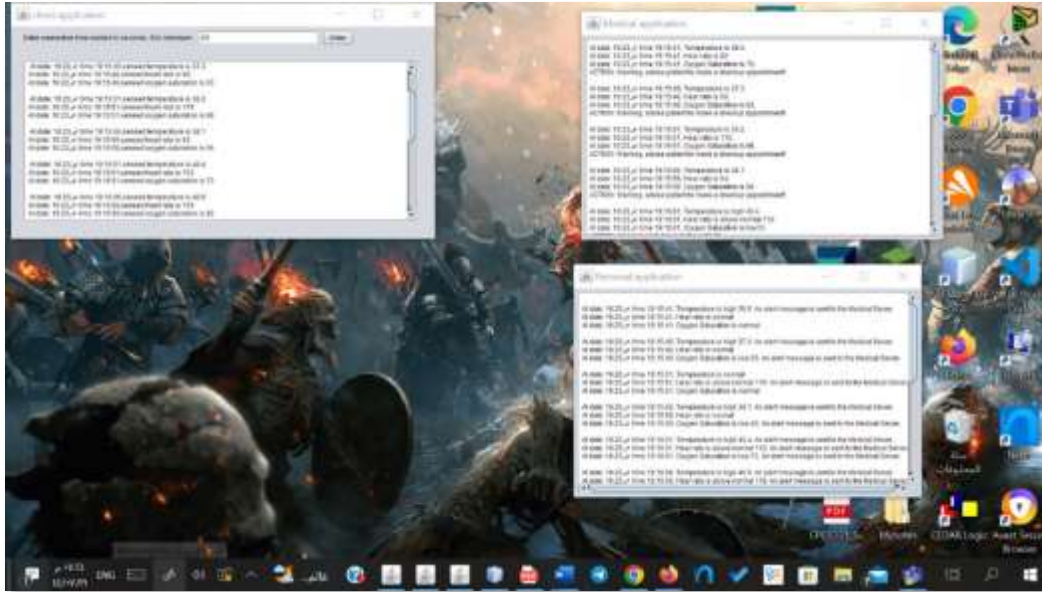


Figure 1: run in GUI 3 classes.

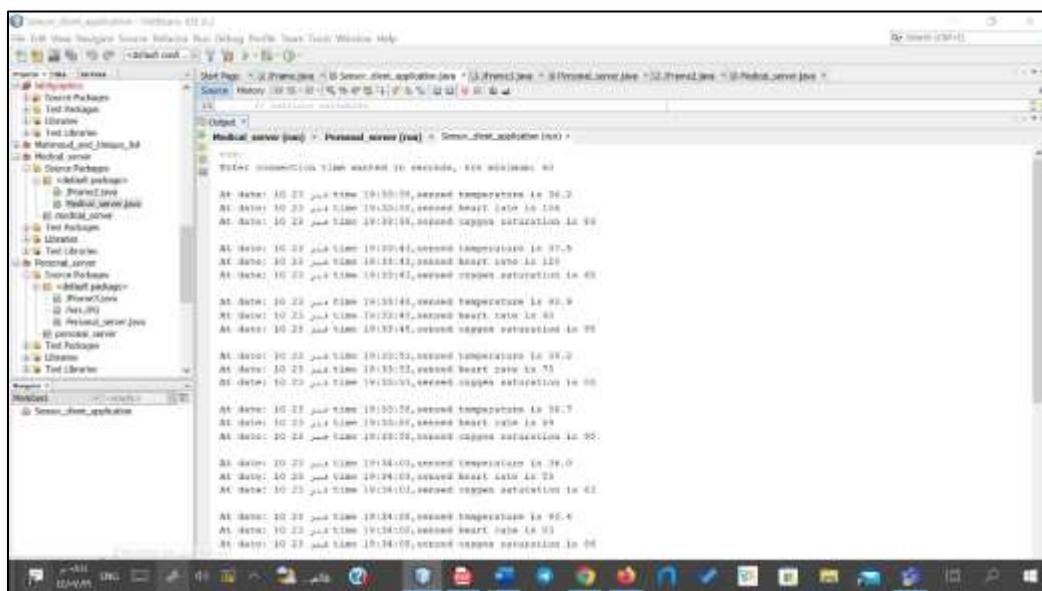
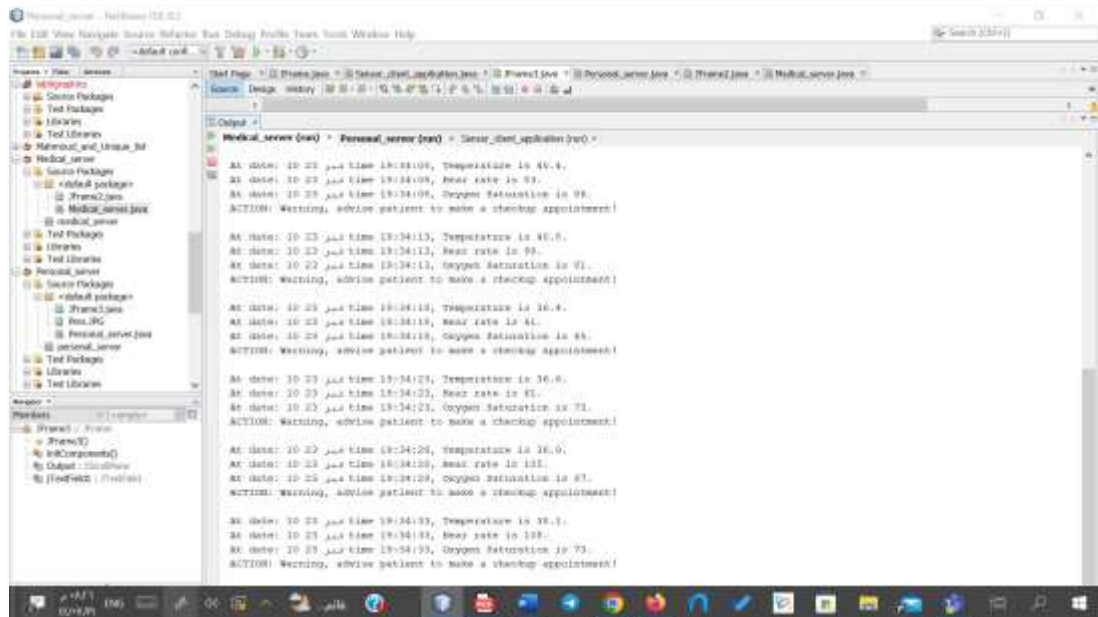
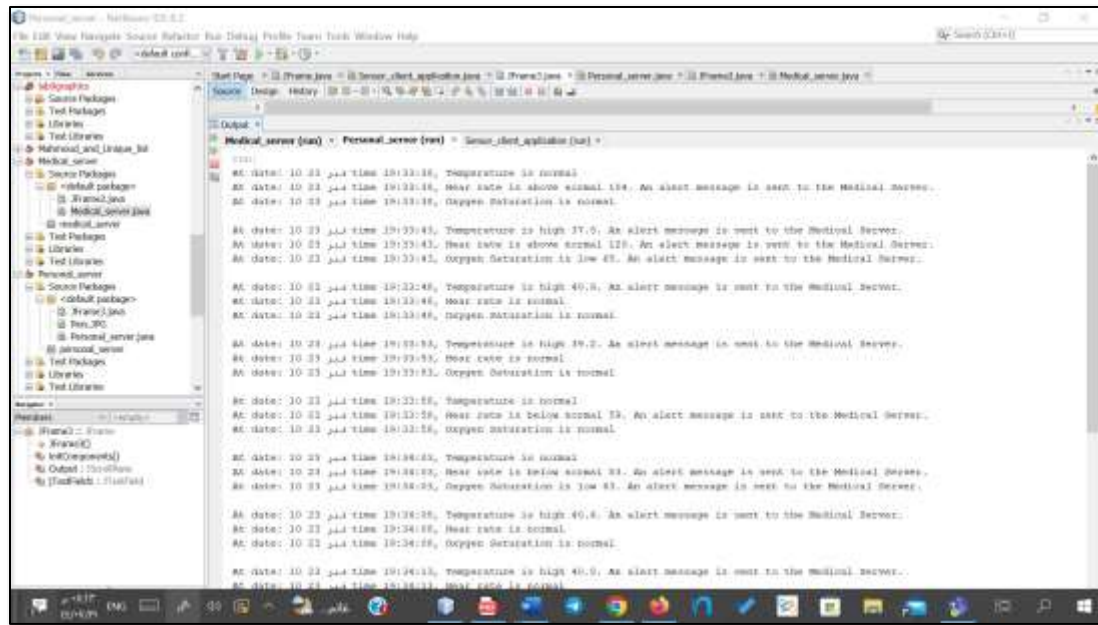


Figure 2: run in Sensor console.



4.2 run of the program on different machines.

Using the code below

```
// to get the address from another device
```

```
InetAddress addresses = InetAddress.getByName("x.x.x.x");
```

```
// save the address in a String variable
```

```
String hostName = addresses.getHostByName();
```

```
// create the socket based on the specific port and address
```

```
Socket socket = new Socket(hostName, port);
```

We could run the code on two different machines based on its IP address.

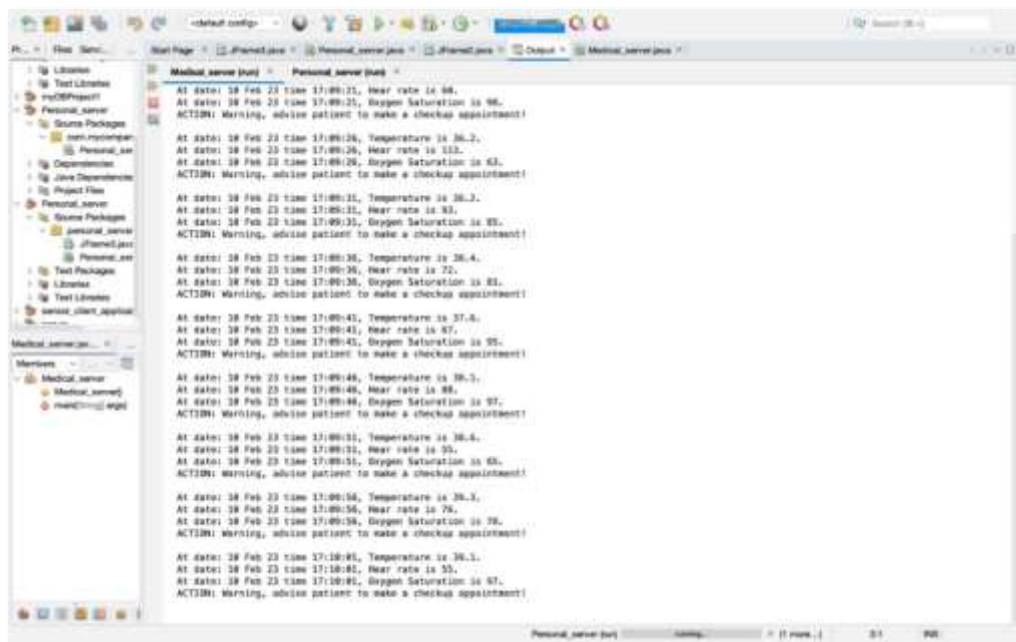


Figure 5:run in Medical Server console on the first device.

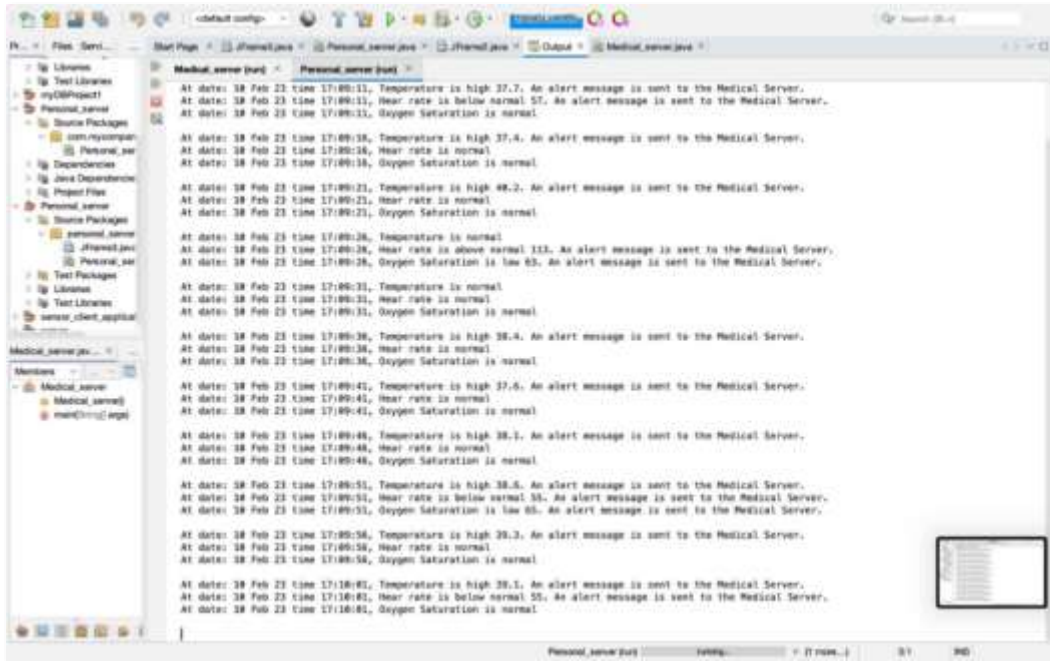


Figure 6: run in Personal Server console on the first device.

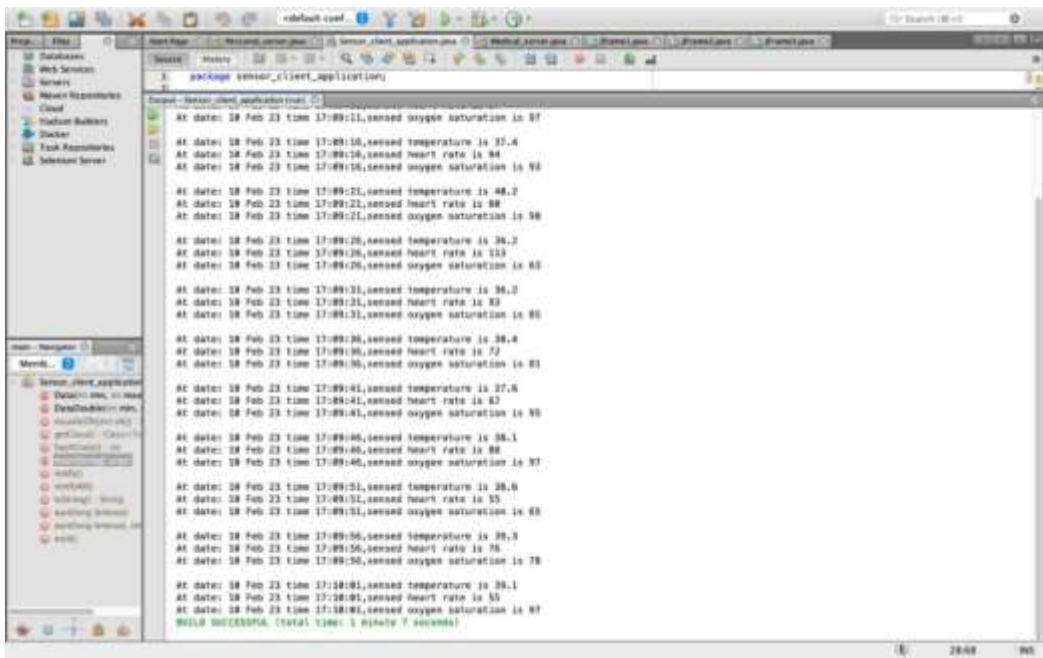


Figure 7: run in Sensor console on the second device.

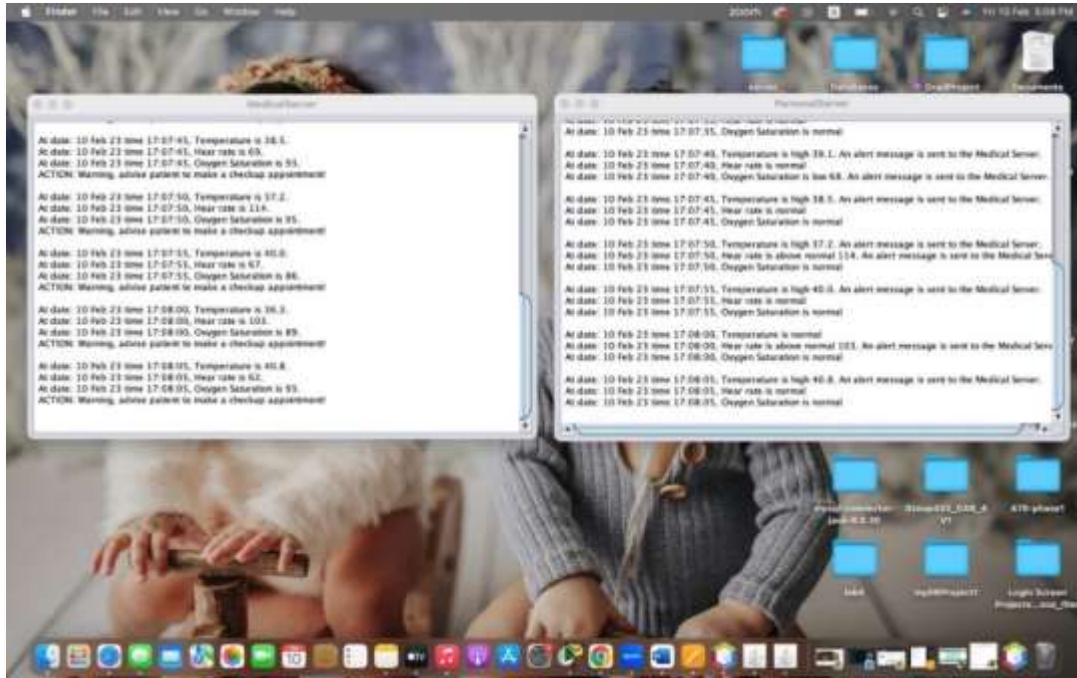


Figure 8:run in Personal and Medical Servers GUI on the first device.



Figure 9:run in Sensor GUI on the second device.

5 Teamwork and Lessons learned.

We divided our group of 6 girls into three pairs using an online spinner to decide which pair works on which class. Shaden and Logain worked on the sensor client application, Amani and Jetana on the personal server, Hasna and Wajd worked on the medical server. Amani debugged the final application code and fixed all minor details that were causing some issues with the connection of projects together. As for the report, the introduction and teamwork and lesson learned were done by Logain and Jetana, the patient monitoring application interaction diagram was done by Amani, the RHMS implementation was done by Wajd, the RHMS application run snapshots were done by Shaden, the word file template and the conclusion were done by Hasna.

As for the lessons learned, this project was new to all group members, and it explained the client-server architecture and taught us how it's implemented. We also learned how to manage and divide a big group to get the most out of everyone's time and effort.

6 Conclusion

To sum up, this project is based on the client-server architecture and how to implement an application to inspect elderly patients' health using TCP sockets. Additionally, the application consists of one client and two servers. So, the messages will flow throughout the 3 TCP sockets. One of the main ideas in this application is that the program could be run on up to three end systems. Therefore, the whole report was done by 6 group members cooperatively.

7 References

James F. Kurose, Keith W. Ross,, "Computer Networking", Pearson Education; 6 edition (2012)

Java Socket Programming (Java Networking Tutorial) - javatpoint. (n.d.). Retrieved January 22, 2023 from [www.javatpoint.com](https://www.javatpoint.com/socket-programming). <https://www.javatpoint.com/socket-programming>

P. (2022, November 22). *Thread.sleep() in Java - Java Thread sleep*. Retrieved January 25, 2023 from DigitalOcean. <https://www.digitalocean.com/community/tutorials/thread-sleep-java>

NetBeans, A. (n.d.). *Designing a Swing GUI in NetBeans IDE*. <https://netbeans.apache.org/kb/docs/java/quickstart-gui.html>