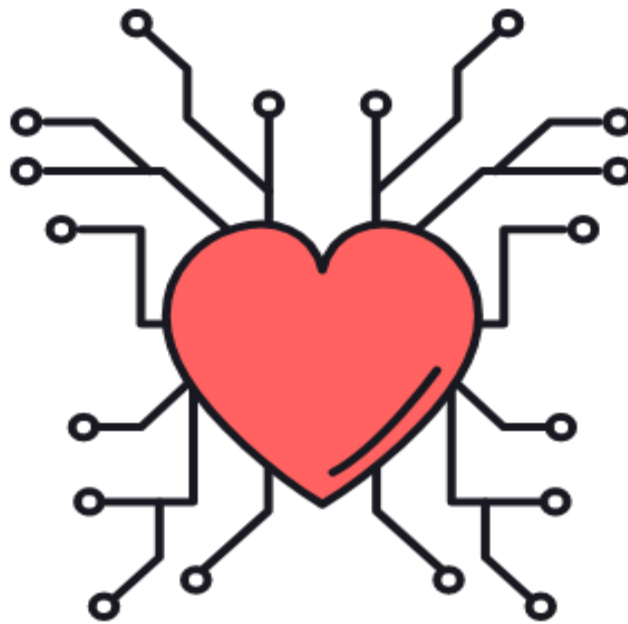


Course Project Report

Simulating a Remote Health Monitoring System for Elderly Patients Using Client-Server TCP Sockets

CPCS-371: Computer Networks I

Group #1



Group Member	Section
Anfal Sultan Alshehri	B1
Danah Saleh Almalki	
Hadeel Hassan Althbiti	
Layla Zuhair Alsulaimani	
Leen Hosiki	
Maysoon AlMalawi	

Table of Contents:

1. Introduction.....	3
1.1. Client-Server Applications	3
1.2. Java TCP Sockets	3
1.3. GUI Elements.....	3
1.4. RHMS applications	4
1.5. Role of Sensor Application Client.....	4
1.6. Role of Personal Client/Server	4
1.6.1. Connection with the sensor application as a server	4
1.6.2. Connection with the medical client as a client.....	5
1.7. Role of Medical Server	5
1.8. Overview.....	5
2. Patient Monitoring Application Interaction Diagram	6
3. RHMS Implementation	7
3.1. Medical Server	7
3.2. Personal Server	11
3.3. Sensor Client.....	17
4. Application Run Snapshots	24
4.1. Program Run on One Machine.....	24
4.1.1. The Welcome GUI:	24
4.1.2. The GUI to get the time from the user:.....	24
4.1.3. The output display GUI:.....	24
4.1.4. The content of output display GUI:	25
.....	25
4.2. Program Run on Different Machines.....	26
5. Teamwork and Lessons Learned	29
5.1. Work Distribution Among Group Members	29
5.2. Work Coordination Among Group Members	30
5.3. Difficulties Encountered and How They Were Overcome	30
5.4. Lessons Learned.....	30
6. Conclusion	31
7. References.....	31

1. Introduction

1.1. Client-Server Applications

The three basic elements to establish an effective human interaction are the sender, receiver, and message. In the context of a network, those three elements are equivalent to a client, server, and data respectively. A client-server application is one where a client starts a connection by sending the server some request. The server, in turn, analyzes the received data and gives feedback. In this type of application two clients cannot contact each other directly—such communication can only happen with a server as an intermediary. The server must always be on to be able to host any client at any time. Moreover, the server has a fixed IP address, so the clients always know where to reach.

1.2. Java TCP Sockets

When a process is running from different end systems, all the data needed for the communication must be sent from a socket and received through a socket. Before any data can be sent or received, the two devices need to perform a handshake. As the name suggests, a handshake is a signal between the two devices that establishes the parameters of the connection; a TCP (Transmission Control Protocol) connection uses a three-way handshake, where three segments are exchanged to acknowledge the connection. Due to this signal, the sockets are now connected and can send data without the need to refer to the address of the destination in a packet.

1.3. GUI Elements

This project utilizes the Swing Java toolkit to build the GUI (Graphical User Interface) which specializes in designing window-based applications using many components—some are simple elements such as frames, buttons, and text fields; and others are complex. Furthermore, the usage of pluggable items makes the Swing an easy and straightforward toolkit. Some of the notable features that were used in this application are:

For displaying output, a JFrame object was used and added to it was the JTextArea object for displaying Strings and the JScrollPane object that provide scrolling to enable the user to view all frame contents even when they exceed the limits of the frame size.

Welcome GUI and the GUI that gets the time are built using JFrame Form. Some elements have been dragged and dropped in the frame, such as labels, textArea, and buttons. Also, the

elements were edited by double clicking on the component to go into the method in the source code.

1.4. RHMS applications

Elderly patients usually require a lot of monitoring as their conditions can be somewhat unstable and prone to sudden declines in health state. Thus, the Remote Health Monitoring System (RHMS) serves to have an ongoing connection to record temperature, heart rate, and oxygen saturation from the patient. These measurements are taken by the Sensor Client Application and then get sent to be processed to the Personal Server, which acts as a server upon receiving the data. If The Personal Server detected that any of the readings happened to be abnormal, it then takes the role of a client and sends those readings for the Medical Server to decide and display an action.

1.5. Role of Sensor Application Client

The role of the personal sensor application is to connect to the personal server, start generating random readings for elderly patients for as long as the user specifies, displaying the data on the personal application GUI, and sending all the data to the personal server.

1.6. Role of Personal Client/Server

The personal application program works as an intermediate server between the medical server and the sensor application. Its role starts with establishing two different TCP connections: one with the medical server as a client and the other with the sensor application as a server. In addition, it outputs data in the desired format, as will be discussed in 1.6.2.

1.6.1. Connection with the sensor application as a server

In this connection, the personal program works as server for the sensor application. It must stay on all the time. Once a connection is requested and then established between the two applications, the personal server starts to receive data from the sensor application. It then processes the data, categorizes them based on the specified rules. It finally outputs the data with their categorizations, in addition to a warning message if the data needed to be sent to the medical server. Finally, the connection ends when there is no more data to be sent from the sensor application.

1.6.2. Connection with the medical client as a client

In this connection, the personal program works as a client that sends data received from the sensor to the medical server. It is important to mention that it only sends emergency cases as specified in 1.4. The role of the personal program in this connection ends when it is done sending data. No data is expected to be sent back from the medical server.

1.7. Role of Medical Server

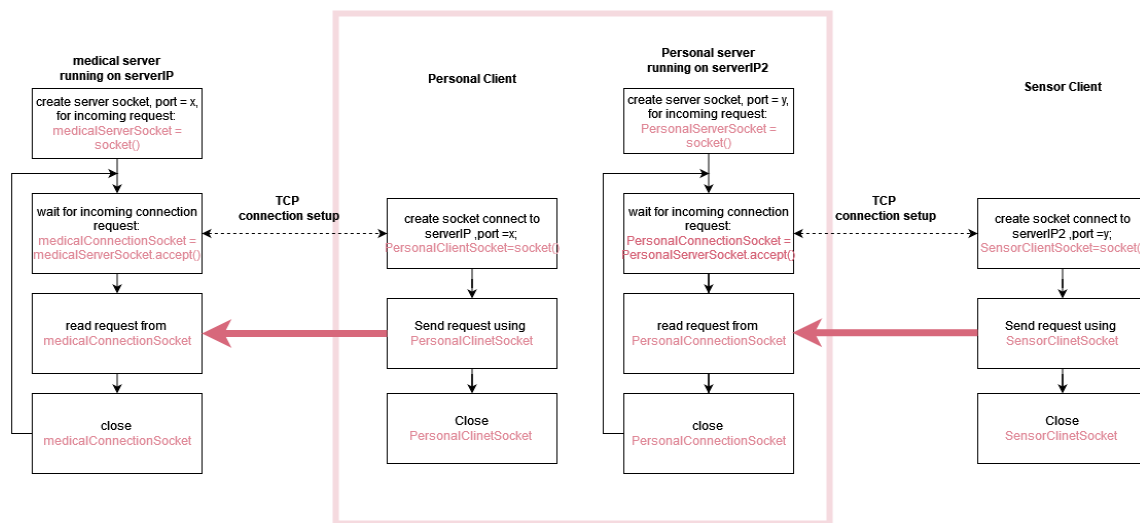
The role of the Medical Server is to receive abnormal readings from the Personal Client, processes those readings accordingly to decide the appropriate action, and then displays only the concerning measurements along with the decided action on the caregiver interface.

1.8. Overview

In section 2, the interaction diagram of this project is presented. In section 3, Java code implementation of the main classes is shown, with a clear description of the workflow of the program. In section 4, two run snapshots of the program are displayed: first on one machine, and then on different machines. Finally, section 5 will be a discussion of teamwork management throughout this project, the obstacles that were encountered and the methods used to overcome them, and the lessons learned from this project.

2. Patient Monitoring Application Interaction Diagram

Figure 1 illustrates the connections between the three applications: Medical Server, Personal Server/Client, and the Sensor Client. We can see that there are 2 TCP connections, one between the Medical Server and the Personal Client, the other between the Personal Server and the Sensor Client. The personal application had been divided into a server and a client despite being implemented as one program for easier understanding of the diagram and the flow of data between those three applications.



RHMS interaction diagram

Figure 1:Interaction Diagram

3. RHMS Implementation

3.1. Medical Server

Starting with the methods implemented and used in the medical server class:

set_up_gui(): this method is used to create and customize the GUI components used to display the output. Customizations include: the frame title, font size and type for the text area, background and foreground(font) color for the text area, adding a scroll pane to the text area, setting the size and the location of the frame on the screen, and finally setting the default close operation to hide on close.

```
// set up gui frame and textArea
static void set_up_gui() {
    // create the frame object
    frame = new JFrame(title: "Medical Server");

    // create text area
    textArea = new JTextArea(
        | | text: ""
    );
    // customize the text area
    textArea.setFont(new Font(name: "DialogInput", Font.PLAIN, size: 25));
    textArea.setLineWrap(wrap: true);
    textArea.setWrapStyleWord(word: true);
    textArea.setBackground(new Color(r: 245, g: 255, b: 250));
    textArea.setForeground(new Color(r: 105, g: 105, b: 105));
    textArea.setEditable(b: false);

    // creat a scroll pane to allow scrolling
    JScrollPane scrollPane = new JScrollPane(textArea);

    //Add the text area to the frame
    frame.add(scrollPane);
    frame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);

    // set the frame size
    frame.setPreferredSize(new Dimension(width: 700, height: 400));

    frame.pack();
    // set the frame location on the screen
    Point point = new Point(610 - 350, 340 + 200);
    frame.setLocation(point);
    frame.setAlwaysOnTop(alwaysOnTop: true);
}
```

connect(): This is a simple method to set up the connection input stream and create a buffered reader for the stream. No output stream or buffered writer is necessary since the medical server doesn't need to write a reply to the personal application client

```
static void connect() throws IOException {  
    // get connection_socket's input and output stream  
    inputStream = new InputStreamReader(connection_socket.getInputStream());  
    // construct buffered reader and buffered writer to read/write from/to client  
    reader = new BufferedReader(inputStream);  
}
```

display(String msg): this method is an extension of the GUI of our program, it is used to append any output string to the text area created in the set_up_gui(). The setVisible() method is invoked here and given the boolean parameter true so the output frame created in the set_up_gui() only appear on the screen when there is an output.

```
// use gui to display the medical server output  
static void display(String msg) {  
    frame.setVisible(true);  
    // add to the text area  
    textArea.append(  
        msg  
    );  
}
```


process_and_display_reading(): here, readings are processed and appropriate messages are created only for abnormal measurements that caused the personal server to send to the medical server in the first place. The display() method is then used to display those messages.

```
static void process_and_display_reading() {
    if (temp >= 38) {
        temp_msg = "At date :" + date + ",time " + time + ", Temperature is high " + temp + ".\n";
        display(temp_msg);
    }
    if (rate >= 100 || rate <= 60){
        if (rate >= 100){
            rate_msg = "At date :" + date + ",time " + time + ", Heart rate is above normal " + rate + ".";
        } else {
            rate_msg = "At date :" + date + ",time " + time + ", Heart rate is below normal " + rate + ".";
        }
        display(rate_msg);
    }
    if (oxygen < 75){
        oxygen_msg = "At date :" + date + ",time " + time + ", Oxygen saturation is low " + oxygen + ".";
        display(oxygen_msg);
    }
}
```

decide_action(): this is the method that provide the main functionality of the medical server, that is to process measurements sent by the personal server and decide on the action to be displayed to the caregiver accordingly.

```
static String decide_action() {
    // decide the action according to the temperature, heart rate, and oxygen saturation
    if (temp >= 39 && rate >= 100 && oxygen <= 95) {
        return "\n\n ACTION: Send an ambulance to the patient!\n\n\n";
    } else if (temp >= 38 && temp <= 38.9 && rate >= 95 && rate <= 98 && oxygen <= 80) {
        return "\n\n ACTION: Call the patient's family!\n\n\n";
    } else {
        return "\n\n ACTION: Warning, advise patient to make a checkup appointment!\n\n\n";
    }
}
```

close_connection(): this is the last method, and it is used to close the connection socket, its input stream object, and its buffered reader.

Finally, we have the main method where we invoke all the methods mentioned above. After the **set_up_gui()** method is used, a TCP server socket is created for the server. Inside an infinite loop, the server waits for requests and accept them through the **accept()** method that return a connection socket and the **connect()** method is invoked to create the input stream and buffered reader objects to read input from the connected client. Another loop is used to keep reading input as long as the client is sending data through the connection, inside that loop each of the measurements is read into a double variable along with the date and time of those readings, then the **process_and_display_reading()** method process those reading to display only the abnormal measurements. After that and before the end of the inner loop, an action is decided through **decide_action()** and displayed with the **display(String msg)**. if there is no more input, we break out the inner loop and use **close_connection()** to close the current connection socket while the outer loop is continued, and the server socket is still open and waiting for new requests.

```
Run | Debug
public static void main(String[] args) throws Exception {
    set_up_gui();
    try {
        // creating TCP server socket "welcoming socket"
        server_socket = new ServerSocket(port);
        // wait for incoming connection request
        while (listening) {
            // new connection socket returned by accept() when server recieve a request from the client
            connection_socket = server_socket.accept();
            // setting up input stream and buffer reader
            connect();
            // keep connection while the client is sending
            input = reader.readLine();
            while ((input) != null) {
                date = input;
                time = reader.readLine();
                temp = Double.parseDouble(reader.readLine());
                rate = Double.parseDouble(reader.readLine());
                oxygen = Double.parseDouble(reader.readLine());
                action = "";
                // display appropriate message for readings
                process_and_display_reading();
                // decide on the action to be displayed
                action = decide_action();
                // display action
                display(action);
                input = reader.readLine();
            }
            close_connection();
        }
    } catch (IOException e) {
    }
}
```

3.2. Personal Server

set_up_gui(): as described in the medical server

Line 32 –47 describes the connection, set up the connection input stream and create a buffered reader for the stream to work as a server. Also, create output stream and buffered writer to act like a client to the medical server.

```
set_up_gui();
    // creating TCP server socket "welcoming socket"
    ServerSocket startSocket = new ServerSocket(port);
    // infinite loop to accept connection request from sensor
    while (true) {
        // new connection socket returned by accept() when personal server
        // recieve a request from the sensor
        connectionSocket = startSocket.accept(); //Handshaking

        // data steam obj to make the server read client input through
        // connection socket
        SensorOutput = new
        InputStreamReader(connectionSocket.getInputStream());

        // connection socket with medical server
        personal2MedicalConnection = new Socket("localhost", 1204);

        //data steam obj to make the server able to send masseges to thr
        // client
        personal2Medical = new
        OutputStreamWriter(personal2MedicalConnection.getOutputStream());

        // construct buffered reader and buffered writer
        reader = new BufferedReader(SensorOutput);
        writer = new BufferedWriter(personal2Medical);
```

This loop- second while loop- works as long as the client (the sensor in our case) is sending input.

checkTemp(Date date, Date time, double temp), **checkHeart**(Date date, Date time, double heartRate), **checkOxygen**(Date date, Date time, int oxygen) these methods check the read input from the client -which is its parameters- and displays the appropriate message according to the values.

```
while ((temp = reader.readLine()) != null) {  
    //read values from the client  
    temperature = Double.valueOf(temp);  
    date = reader.readLine();  
    time = reader.readLine();  
    // display appropriate message for tempreature readings  
    tempCaseMsg = checkTemp(date, time, temperature);  
  
    heartRate = Double.valueOf(reader.readLine());  
    date = reader.readLine();  
    time = reader.readLine();  
    // display appropriate message for heart readings  
    heartCaseMsg = checkHeart(date, time, heartRate);  
  
    oxygenSaturation = Integer.valueOf(reader.readLine());  
    date = reader.readLine();  
    time = reader.readLine();  
    // display appropriate message for oxygen readings  
    oxgCaseMsg = checkOxygen(date, time, oxygenSaturation);  
  
    // only send reading if at least one reading is abnormal  
    if (send2Medical) {  
        // send to medical server
```

The if statement works when one of the readings of the patient is abnormal, that is, when the send2Medical variable evaluates to true. The writer buffer sends the data to the medical server one by one. Then, it sets the send2Medical to false for the next reading and, finally, calls the display method.

```
// only send reading if at least one reading is abnormal
    if (send2Medical) {
        // send to medical server
        writer.write(date);
        writer.newLine();
        writer.flush();

        writer.write(time);
        writer.newLine();
        writer.flush();

        writer.write(temperature + "");
        writer.newLine();
        writer.flush();

        writer.write(heartRate + "");
        writer.newLine();
        writer.flush();

        writer.write(oxygenSaturation + "");
        writer.newLine();
        writer.flush();

        // set send2medical to false for the next reading
        send2Medical = false;
    }
    // display output through gui
    display();
```

The **checkTemp** method takes the temperature as input, as well as the time and date of the reading. If the temperature is 38 or above, it sets the send2Medical variable to TRUE, forms a message that says the temperature is high and that an alert has been sent to the medical server and returns the message. However, the temperature is normal, it forms a message that says so and returns it.

The **checkHeart** method takes the heart rate as input, as well as the time and date of the reading. If the heart rate is 100 or above or 60 or below, it sets the send2Medical variable to TRUE, forms a message that says the heart rate is above normal or below normal and that an alert message has been sent to the medical server and returns the message. On the other hand, if the heart rate is normal, it forms a message that says so and returns it.

The **checkOxygen** method takes the oxygen saturation as input, as well as the time and date of the reading. If the oxygen saturation is 75 or less, it sets the send2Medical server to TRUE, forms a message that says that the heart rate is below normal and that an alert message has been sent to the medical server and returns the message. However, if the oxygen saturation is normal, it forms a message that says so and returns it.

```
public static String checkTemp(String date, String time, double temperature)
throws IOException {
    // displys a message depending on the parameters values
    if (temperature >= 38) {
        send2Medical = true;
        String msg = "At date :" + date + ",time " + time + ", Temperature is
high " + temperature + ".";
        return msg + "\n An alert message is sent to the Medical Server";
    } else {
        return "At date :" + date + ",time " + time + ", Temperature is
normal " + temperature + ".";
    }
} //method end
```

```

public static String checkHeart(String date, String time, double heartRate)
throws IOException {
    // displys a message depending on the parameters values
    if (heartRate >= 100) {
        send2Medical = true;
        String msg = "At date :" + date + ",time " + time + ", Heart rate is
above normal " + heartRate + ".";
        return msg + "\n An alert message is sent to the Medical Server";

    } else if (heartRate <= 60) {
        send2Medical = true;
        String msg = "At date :" + date + ",time " + time + ", Heart rate is
below normal " + heartRate + ".";
        return msg + "\n An alert message is sent to the Medical Server";

    } else {
        return "At date :" + date + ",time " + time + ", Heart rate is normal
" + heartRate + ".";
    }
} //method end

```

```

public static String checkOxygen(String date, String time, int oxygenSaturation)
throws IOException {
    // displys a message depending on the parameters values
    if (oxygenSaturation <= 75) {
        send2Medical = true;
        String msg = "At date :" + date + ",time " + time + ", Oxygen
saturation is low " + oxygenSaturation + ".";
        return msg + "\n An alert message is sent to the Medical Server";
    } else {
        return "At date :" + date + ",time " + time + ", Oxygen saturation is
normal " + oxygenSaturation + ".";
    }
} //method end

```

The **close_connection** method is responsible for closing the two connections of the personal server/client application and everything related to it. It returns void and throws an exception. It closes the following:

- Connection with the sensor client.
- Connection with the medical server.
- Reader buffer that is responsible for reading from the sensor client.
- Writer buffer that is responsible for writing to the medical server.
- The input stream reader for the sensor client.
- The output stream writer for the medical server.

```
- static void close_connection() throws Exception {  
-     connectionSocket.close();  
-     personal2MedicalConnection.close();  
-     reader.close();  
-     writer.close();  
-     personal2Medical.close();  
-     SensorOutput.close();  
- }
```

The **display** method is responsible for the GUI element in the application. It displays the GUI frame by setting its variable to TRUE and appending the static messages variables to the text area of the frame.

```
static void display() {  
    frame.setVisible(true);  
    // add to the text area  
    textArea.append(  
        " " + tempCaseMsg + "\n " + heartCaseMsg + "\n " + oxgCaseMsg +  
        "\n\n\n"  
    );  
}
```


3.3. Sensor Client

First, the main code, like the method's calls, has been placed in the function of the button because it must start compiling the other lines of code after the user clicks on the button (Enter).

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) { //GEN-FIRST:event_jButton1ActionPerformed
    /*Main code*
    // get the time from the user
    t = Long.parseLong(seconds.getText());
    if (t < 60) {
        JOptionPane.showMessageDialog(null, "The time is less than 60 Seconds, Please try again.");
        return;
    }
    //After Start
    // 1-Connection with the Server
    try {
        connectToServer();
    } catch (IOException ex) {
        Logger.getLogger(Sensor_Client.class.getName()).log(Level.SEVERE, null, ex);}

    long user_time=t;
    user_time *= 1000; //2- convert the time from seconds to milliseconds
    user_time += System.currentTimeMillis(); //add user_time in the current time

    this.setVisible(false);
    set_up_gui();

    while (user_time > System.currentTimeMillis()) {
        try {
            //3- get random data
            temperature = randomTem();
            heartRate = randomHeart();
            oxygen = randomOxygen();

            printAndSendData(); //4- send data ro tha personal server and print data
            Thread.sleep(5000); //5- stop the code 5 seconds

        } catch (IOException ex) {
            Logger.getLogger(Sensor_Client.class.getName()).log(Level.SEVERE, null, ex);
        } catch (InterruptedException ex) {
            Logger.getLogger(Sensor_Client.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    JOptionPane.showMessageDialog(null, "The " + t + " Seconds is ended");
    frame.setVisible(true);

    try {
        close_connection();
    } catch (Exception ex) {
        Logger.getLogger(Sensor_Client.class.getName()).log(Level.SEVERE, null, ex);
    }
} //GEN-LAST:event_jButton1ActionPerformed
```

Then check if the time was less than 60 seconds. If it's less than 60 seconds, show the message window to tell the user to try again (Figure 3).

```
// get the time from the user
t = Long.parseLong(seconds.getText());
if (t < 60) {
    JOptionPane.showMessageDialog(null, "The time is less than 60
Seconds, Please try again.");
    return;
}
```

connectToServer(): This function sets up the TCP connection with the personal server.

```
public static void connectToServer() throws IOException {

    socket = new Socket("localhost", 2000);

    sendToServer = new OutputStreamWriter(socket.getOutputStream());

    buf_SendToServer = new BufferedWriter(sendToServer);
}
```

The setting of the time to end the run of the program.

```
long user_time=t;
    user_time *= 1000;//2- convert the time from seconds to milliseconds

    user_time += System.currentTimeMillis();//add user_time in the current time
```

set_up_gui(): This method to set up the display output through GUI, it is -the same as the medical server's method described in the previous subsection.

```
// use gui to display the Sensor server output
static void displayTemp() {

    // add to the text area
    textArea.append("At date: " + date + ", time " + time + ", sensed
temperature is " + temperature + "\n"
    );
}

static void displayHeart() {
    // add to the text area
    textArea.append("At date: " + date + ", time " + time + ", sensed heart
rate is " + heartRate + "\n"
    );
}
static void displayOxygen() {
    // add to the text area
    textArea.append("At date: " + date + ", time " + time + ", sensed oxygen
saturation is " + oxygen + "\n\n"
    );
}
```

In this While loop, random values are given to the variables through the method. Also, the call of the function prints the data and sends it to the server. Last, use threads to stop the loop for 5 seconds.

```
while (user_time > System.currentTimeMillis()) {
    try {
        //3- get random data
        temperature = randomTem();
        heartRate = randomHeart();
        oxygen = randomOxygen();

        printAndSendData();//4- send data ro tha personal server and print data

        Thread.sleep(5000); //5- stop the code 5 seconds

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

randomTem(), randomHeart(), randomOxygen(): These methods generate random numbers within the constraints.

```
//To generate a random number for the Temperature sensor between 36C and 41C.
public static double randomTem() {
    double temperature = 36 + (Math.random() * ((41 - 36) + 1));
    if (temperature > 41) {
        temperature = 41.0;
    }

    temperature = Double.parseDouble(new
DecimalFormat("##.##").format(temperature)); //Get one digit after the decimal
point
    return temperature;
}
```

```

//To generate a random number for the Heart rate sensor between 50 and 120.
public static double randomHeart() {
    double heartRate = 50 + (Math.random() * ((120 - 50) + 1));
    if (heartRate > 120) {
        heartRate = 120.0;
    }
    heartRate = Double.parseDouble(new
DecimalFormat("##.##").format(heartRate)); //Get one digit after the decimal point
    return heartRate;
}

//To generate a random number for the Oxygen Level sensor between 60% to 100%.
public static int randomOxygen() {
    int oxygen = 60 + (int) (Math.random() * ((100 - 60) + 1));
    return oxygen;
}

```

In these two methods, formatting for time and date is like the sample output.

```

// get the date
public static String date() {
    Date date = new Date();
    SimpleDateFormat DateFor = new SimpleDateFormat("dd MMMM yy");
    String stringDate = DateFor.format(date);
    return stringDate;
}

// get the time
public static String time() {
    Date time = new Date();
    SimpleDateFormat TimeFor = new SimpleDateFormat("HH:mm:ss");
    String stringTime = TimeFor.format(time);
    return stringTime;
}

```

DoubleToString(double d): This function converts the double number to a string because the `BufferedWriter.write()` doesn't accept the double type.

```

public static String DoubleToString(double d) {
    String str = Double.toString(d);
    return str;
}

```

printAndSendData(): Prints the data in the GUI and sends it to the personal server using a buffered writer.

```
public static void printAndSendData() throws IOException {
    //Send and print the *temperature* to the Personal Server
    date = date();
    time = time();

    System.out.println("At date: " + date + ", time " + time + ", sensed
temperature is " + temperature);
    displayTemp();
    buf_SendToServer.write(DoubleToString(temperature));
    buf_SendToServer.newLine();
    buf_SendToServer.write(date);
    buf_SendToServer.newLine();
    buf_SendToServer.write(time);
    buf_SendToServer.newLine();

    //Send and print the *heart rate* to the Personal Server

    System.out.println("At date: " + date + ", time " + time + ", sensed
heart rate is " + heartRate);
    displayHeart();
    buf_SendToServer.write(DoubleToString(heartRate));
    buf_SendToServer.newLine();
    buf_SendToServer.write(date);
    buf_SendToServer.newLine();
    buf_SendToServer.write(time);
    buf_SendToServer.newLine();

    //Send and print the *oxygen saturation* to the Personal Server
    System.out.println("At date: " + date + ", time " + time + ", sensed oxygen
saturation is " + oxygen);
    displayOxygen();
    buf_SendToServer.write(Integer.toString(oxygen));
    buf_SendToServer.newLine();
    buf_SendToServer.write(date);
    buf_SendToServer.newLine();
    buf_SendToServer.write(time);
    buf_SendToServer.newLine();
    buf_SendToServer.flush();

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

In the while loop, these two lines of the code, the first line shows a message to the user to inform them that the time set for sending has ended(Figure 5), then the second line shows the output GUI(Figure 6).

```
JOptionPane.showMessageDialog(null, "The " + t + " Seconds is ended");  
  
frame.setVisible(true);
```

close_connection(): In the end, it closes the connection with the personal server.

```
static void close_connection() throws Exception{  
    socket.close();  
  
    sendToServer.close();  
  
    buf_SendToServer.close(); }  
}
```

4. Application Run Snapshots

4.1. Program Run on One Machine

4.1.1.The Welcome GUI:



Figure 2

4.1.2.The GUI to get the time from the user:

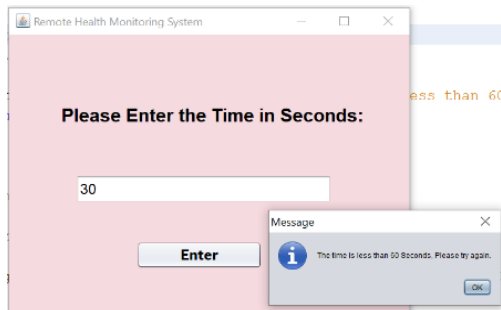


Figure 3

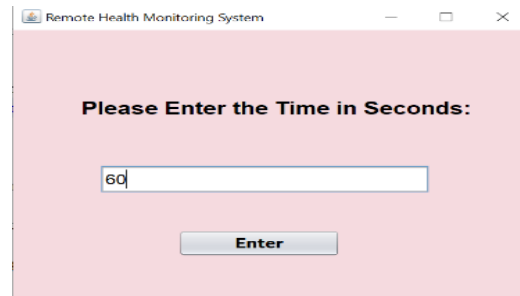


Figure 4

4.1.3.The output display GUI:

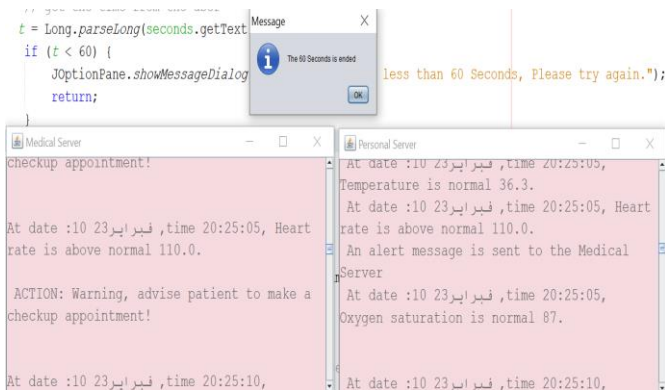


Figure 5

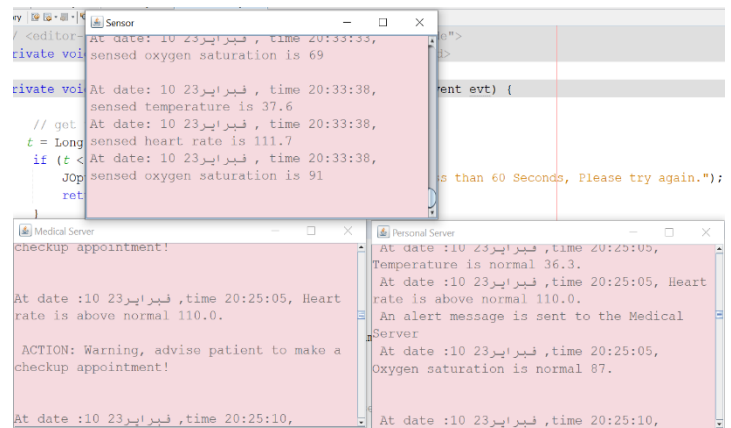



Figure 6

4.1.4. The content of output display GUI:


 **Sensor**

```
At date: 10 فبرایر 23 , time 20:32:43, sensed temperature is 41.0
At date: 10 فبرایر 23 , time 20:32:43, sensed heart rate is 92.7
At date: 10 فبرایر 23 , time 20:32:43, sensed oxygen saturation is 71

At date: 10 فبرایر 23 , time 20:32:48, sensed temperature is 36.6
At date: 10 فبرایر 23 , time 20:32:48, sensed heart rate is 100.2
At date: 10 فبرایر 23 , time 20:32:48, sensed oxygen saturation is 88

At date: 10 فبرایر 23 , time 20:32:53, sensed temperature is 40.6
At date: 10 فبرایر 23 , time 20:32:53, sensed heart rate is 87.3
At date: 10 فبرایر 23 , time 20:32:53, sensed oxygen saturation is 83
```

Figure 7


 **Personal Server**

```
At date :10 فبرایر 23 ,time 20:20:57, Temperature is high 41.0.
An alert message is sent to the Medical Server
At date :10 فبرایر 23 ,time 20:20:57, Heart rate is above normal 116.5.
An alert message is sent to the Medical Server
At date :10 فبرایر 23 ,time 20:20:57, Oxygen saturation is normal 89.

At date :10 فبرایر 23 ,time 20:21:02, Temperature is high 39.4.
An alert message is sent to the Medical Server
At date :10 فبرایر 23 ,time 20:21:02, Heart rate is above normal 113.4.
An alert message is sent to the Medical Server
At date :10 فبرایر 23 ,time 20:21:02, Oxygen saturation is low 65.
An alert message is sent to the Medical Server

At date :10 فبرایر 23 ,time 20:21:07, Temperature is normal 36.3.
At date :10 فبرایر 23 ,time 20:21:07, Heart rate is normal 78.7.
At date :10 فبرایر 23 ,time 20:21:07, Oxygen saturation is low 74.
An alert message is sent to the Medical Server
```

Figure 8

 **Medical Server**

```
At date :10 فبرایر 23 ,time 20:20:57, Temperature is high 41.0.
At date :10 فبرایر 23 ,time 20:20:57, Heart rate is above normal 116.5.

ACTION: Send an ambulance to the patient!

At date :10 فبرایر 23 ,time 20:21:02, Temperature is high 39.4.
At date :10 فبرایر 23 ,time 20:21:02, Heart rate is above normal 113.4.
At date :10 فبرایر 23 ,time 20:21:02, Oxygen saturation is low 65.0.

ACTION: Send an ambulance to the patient!

At date :10 فبرایر 23 ,time 20:21:07, Oxygen saturation is low 74.0.

ACTION: Warning, advise patient to make a checkup appointment!
```

Figure 9

4.2. Program Run on Different Machines

Running the program will first display the main window in *Figure 10* which has a welcome message and a button to START the application. The screen then moves to *Figure 11* which prompts the user to enter a time. Invalid input is any time that is less than a minute, thus it will result in the error message shown in *Figure 12*. Valid input, shown in *Figure 14* will allow the user to proceed.

There are five readings that can be seen in *Figure 17* and then *Figure 18* shows the corresponding response. These readings indicate the following cases:

- 1- The first four readings all fall under the third case which displays the message: “Warning, advise patient to make checkup appointment!”
- 2- The fifth reading falls under the first case which displays the message: “Send an ambulance to the patient!”

Finally, there is the second case which displays the message: “Call the patient’s family!” This action gets triggered when the temperature and the heart rate high and the oxygen saturation is low. However, this case is not available in the current output set. The following code segment is used to enable the applications to run on different machines. Specifically in the client code where the "x.x.x.x" string represent the IP address of the device that run the servers for each client, the obtained hostname is then used in constructing the connection socket object instead of "localhost" that is used when running on one device.

```
InetAddress addresses = InetAddress.getByName("x.x.x.x");  
String hostName = addresses.getHostName();
```

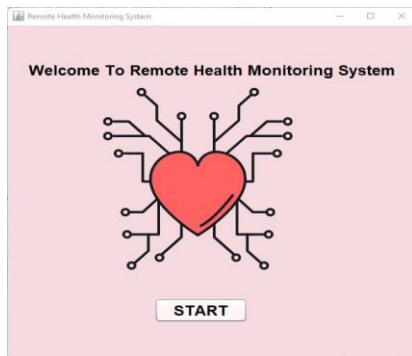


Figure 10:Welcome RHMS GUI Machine 1



Figure 11:user input(if the time less than 60 sec)
Machine 1



Figure 12:Error Message Machine 1



Figure 13:Welcome RHMS GUI Machine 1



Figure 14: user input(if the time 60 sec) Machine 1

```

Personal Server
At date :13 23فراير ,time 22:20:36,
Temperature is normal 36.4.
At date :13 23فراير ,time 22:20:36, Heart
rate is above normal 108.2.
An alert message is sent to the Medical
Server
At date :13 23فراير ,time 22:20:36,
Oxygen saturation is low 67.
An alert message is sent to the Medical
Server

```

Figure 15:output in Machine 2

```

Medical Server
At date :13 23فراير ,time 22:20:36, Heart
rate is above normal 108.2.At date :13
23فراير ,time 22:20:36, Oxygen saturation
is low 67.0.

ACTION: Warning, advise patient to make a
checkup appointment!

At date :13 23فراير ,time 22:20:41,
Temperature is high 41.0

```

Figure 16:output in Machine 3

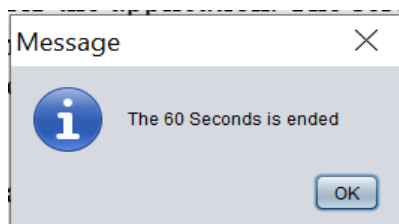


Figure 17:output in Machine 1

```

Sensor
At date: 13 23فراير , time 22:20:36,
sensed temperature is 36.4
At date: 13 23فراير , time 22:20:36,
sensed heart rate is 108.2
At date: 13 23فراير , time 22:20:36,
sensed oxygen saturation is 67

At date: 13 23فراير , time 22:20:41,
sensed temperature is 41.0
At date: 13 23فراير , time 22:20:41,
sensed heart rate is 82.6
At date: 13 23فراير , time 22:20:41,
sensed oxygen saturation is 71

```

Figure 18:output in Machine 1

```

Personal Server
At date :13 23فراير ,time 22:20:36, Temperature is normal 36.4.
At date :13 23فراير ,time 22:20:36, Heart rate is above normal 108.2.
An alert message is sent to the Medical Server
At date :13 23فراير ,time 22:20:36, Oxygen saturation is low 67.
An alert message is sent to the Medical Server

At date :13 23فراير ,time 22:20:41, Temperature is high 41.0.
An alert message is sent to the Medical Server
At date :13 23فراير ,time 22:20:41, Heart rate is normal 82.6.
At date :13 23فراير ,time 22:20:41, Oxygen saturation is low 71.
An alert message is sent to the Medical Server

```

Figure 19:output in Machine 2

```

Medical Server
At date :13 23فراير ,time 22:20:36, Heart rate is above normal 108.2.At date :13 23فراير ,time 22:20:36, Oxygen saturation
is low 67.0.

ACTION: Warning, advise patient to make a checkup appointment!

At date :13 23فراير ,time 22:20:41, Temperature is high 41.0.
At date :13 23فراير ,time 22:20:41, Oxygen saturation is low 71.0.

ACTION: Warning, advise patient to make a checkup appointment!

```

Figure20:output in Machine 3

```

Sensor
At date: 13 23فراير , time 22:20:36, sensed temperature is 36.4
At date: 13 23فراير , time 22:20:36, sensed heart rate is 108.2
At date: 13 23فراير , time 22:20:36, sensed oxygen saturation is 67

At date: 13 23فراير , time 22:20:41, sensed temperature is 41.0
At date: 13 23فراير , time 22:20:41, sensed heart rate is 82.6
At date: 13 23فراير , time 22:20:41, sensed oxygen saturation is 71

```

Figure21:output in Machine 1

5. Teamwork and Lessons Learned

5.1. Work Distribution Among Group Members

Group Member	Tasks
Anfal Alshehri	<ul style="list-style-type: none">– Interaction Diagram– Personal server code– Snap Shots on different Machine
Danah Almalki	<ul style="list-style-type: none">– Interaction Diagram– Medical Server Code– Output GUI– Role of the medical server– Lessons learned
Hadeel Althbiti	<ul style="list-style-type: none">– Sensor Client Code– GUI for Sensor Client– Snapshots on one Machine
Layla Alsulaimani	<ul style="list-style-type: none">– Personal server code– Formatting the data in sensor application.– Difficulties Encountered and How They Were Overcome
Leen Hosiki	<ul style="list-style-type: none">– Sensor Client Code– Introduction– Output Description– Conclusion
Maysoon AlMalawi	<ul style="list-style-type: none">– Personal client code.– Role of sensor and personal applications.– Explanation of personal client code.

5.2. Work Coordination Among Group Members

First, we started by dividing the coding work according to the three main applications: medical, personal, and sensor applications. Danah worked on the medical server, Hadeel and Leen worked on the sensor application, and Anfal, Layla, and Maysoon worked on the personal server/client. For the personal server class, Anfal started with the methods, Layla was responsible for the personal/sensor connection, and Maysoon was responsible for the medical/personal connection. For the sensor client application, Leen worked on the methods generating random readings, while Hadeel worked on the rest of the methods including the main method.

The GUI code was done by Danah for the output, while the GUI for the welcoming and obtaining input from the user was done by Hadeel.

As for the report content, the work for the interaction diagram was done through a google meet and a shared file on draw.io between the members mentioned in 5.1. The RHMS Implementation section and the sections about the role of each class were done by the members who worked on their respective code. The run snapshots were the responsibility of Anfal and Hadeel for running on different and one machine respectively.

The rest of the report's content that wasn't concerned with a certain application was divided between us as we went on with the project.

5.3. Difficulties Encountered and How They Were Overcome

There were many difficulties throughout the course of this project. However, the main difficulty was dividing the work among a large number of people, and due to the server's and client's classes' dependency, the division of tasks and variation of members' style of coding necessitated the modification of a lot of parts. Also, working with the personal server being a client and a server at the same time and learning how to design GUI and connected to the source code was challenging.

5.4. Lessons Learned

This project is our first project in a while that has this much coding, not to mention the first project where we deal with server/client architecture and what comes with it. Because of that, the errors we dealt with in the code were mostly logical errors. This helped us understand the importance of good planning for the code and the logic used to solve the problem before starting to code. And we learned that a clean, readable code is preferable for easier debugging and better understanding of code than a faster code with less lines, especially when working on a group project as opposed to an individual one.

6. Conclusion

The RHMS application is a system for elderly patients to keep track of their health and interfere in case of an emergency. Implementation of this project required a client, the Sensor Client which measures and sends data; a client/server, the Personal Server which receives data, processes them, and sends them with an appropriate message; and a server, the Medical Server which receives messages and takes action.

Other notable features of this application are the used protocol and implemented interface. First, the TCP protocol was used for connecting the clients and servers. This protocol is considered to be both a reliable and secure protocol; it establishes agreements before the connection through a handshake—three-way handshake in this case. Second, the GUI, it was done with the Java Swing library. Elements like pop-up windows, buttons, and text fields were applied for an interface that is easy and simple to navigate.

7. References

- Kurose, J. F., & Ross, K. W. (2013). *Computer networking: A top-down approach*. Pearson.
- SujanRai, & Marcosarcticseal. (2021, October 26). *TCP 3-way handshake process*.
GeeksforGeeks. Retrieved January 30, 2023, from <https://www.geeksforgeeks.org/tcp-3-way-handshake-process/>
- Waseem, M. (2022, November 29). *Swing in java: Creating GUI using Java Swing*. Edureka.
Retrieved January 30, 2023, from <https://www.edureka.co/blog/java-swing/>