CLIENT SERVER PROJECT
Group Report

William Hogan
Taha Aflouk
Daniel Glynn

Architecture and Design

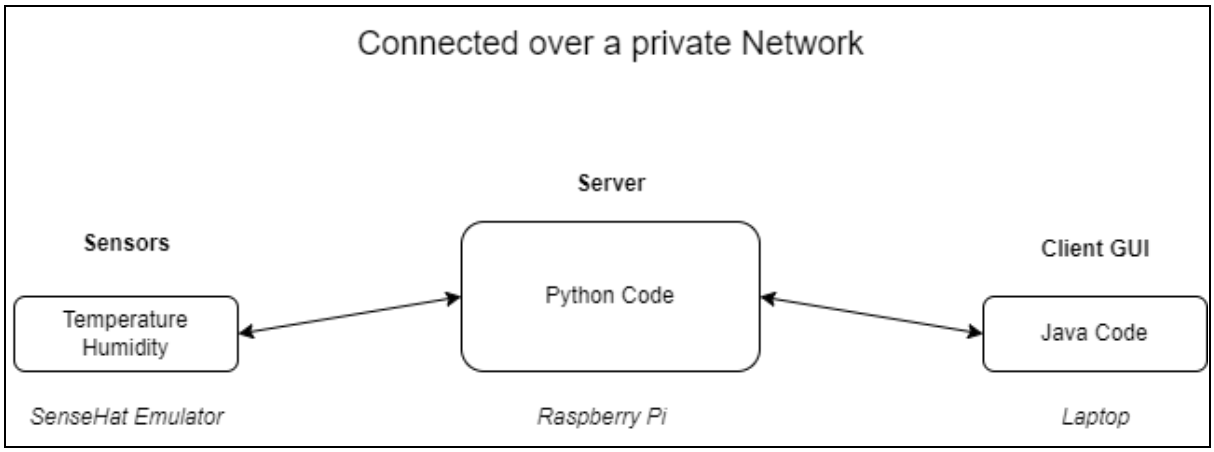

FIG 1: Flow chart of communication between Sensors, Server and Client.

The flowchart above provides a visual representation of the components that make up the Application developed by the group. All components are connected over a private network which provides the ability to request, receive and send information back and forward when desired.

Implementation

```
# @Authors: William, Daniel, TAflouk
# @Date:2022-11-29 18:30:00
# @LastModifiedBy: TAflouk
# @Last Modified time:2022-00-00 00:00:00

# -- IMPORTS --
from sense_emu import SenseHat
import random as r

# -- GLOBAL --
s = SenseHat()


# -- FUNC() --
def  get_weather_data(name):
    # This Func() is to get data from the sense hat
    # it has one parameter  str as an arg
    # returns a number
    name.lower()
    temp = round(s.get_temperature(),1)
    celcius = round(s.get_temperature(),1)
    fahrenheit = round(1.8 *celcius + 32, 1)
    humidity = round(s.get_humidity(),1)
    pressure = round(s.get_pressure(),1)
    if name == "temperature":
        return temp
    elif name == 'celcius':
```

```
        return celcius
    elif name == 'fahrenheit':
        return fahrenheit
    elif name == 'humidity':
        return rhumidity
    elif name == 'pressure':
        return pressure

# -- CLASSES --
class Sensor(object):
    sensor_counter = 0
    def __init__(self, sensor_name):
        self.sensor_name = sensor_name
        self.sensor_name.lower()
        if self.sensor_name == "temperature":
            self.data = get_weather_data("temperature")
        else:
            self.data = get_weather_data("humidity")
        Sensor.sensor_counter += 1

    def __repr__(self):
        return f"Type of Sensor:{self.sensor_name}\nData:{self.data}"

    def __str__(self):
        return f"Type of Sensor:{self.sensor_name}\nData:{self.data}"

    def get_temperature(self):
        return self.data

    def get_humidity(self):
        return self.data

    def get_name(self):
        return self.sensor_name

    def get_data(self):
        return self.data

if __name__ == "__main__":
    print("Testing.")
    s1 = Sensor("temperature")
    print(s1)
    #print(s1.get_sensor_name())
    print(s1.get_temperature())
```

FIG 2: Above is the Sensor class written in Python that communicates with the sensors to retrieve temperature and humidity readings.

```python
# @Authors: William, Daniel, TAflouk
# @Date:2022-11-25 13:10:00
# @LastModifiedBy: TAflouk
# @Last Modified time:2022-11-29 18:10:00

# --- IMPORTS ---
import time; import socket
from Sensor import Sensor
from time import sleep
import random as r

# --- FUNC() ---
def get_avrg(array):
    # This Func() is to get average of a list
    # it has one parameter  list of numbers int/float as an arg
    # returns a number
    total = 0
    for i in range(len(array)):
        total += array[i]
    return (total/len(array))

def convert_to_str(array):
    # This Func() is to convert a list to a sting
    # it has one parameter  list of numbers int/float as an arg
    # returns a string of numbers
    new_str = ""
    for i in range(len(array)):
        new_str += str(array[i])+ " "
    return new_str

# -- MAIN --
def main():
    serverObject = socket.socket() #creating our server
    server = socket.gethostname() #for testing purposes
    print(server) #name of computer...
    port = 2005
    # Use ipconfig (Windows) or ifconfig(Linux) to find the IP address
    # Change the IP address to that of your computer.
    serverObject.bind(("192.168.1.101", port))# setting up server
    print("waiting for connection....")
    serverObject.listen(5)
    client,addr = serverObject.accept()
    print("Got a connection from " + str(addr))
    data = client.recv(1) # get the data from the client
    print(type(data))
    menu= data.decode("utf-8") # convert that from Binary Code into English
letter
```

```
    #print(type(menu)) # print the type to test  what the user enter Note: for
testing
    #print(menu) #print that values to test
    stringOfNums = "" # initializ n empty string
    list_of_temp = [] # initializ n empty lists
    list_of_humi = []
    for i in range(10): # get 10 values from the Sensor class
        list_of_temp.append(Sensor("temperature").get_temperature()) # append
them to the lists
        list_of_humi.append(Sensor("temperature").get_humidity())
    if menu == "1": # check what is the user input
        val1  = min(list_of_humi) # store the values into a vars
        stringOfNums += str(val1)+" " # add the values to the string  as str
    # repeat the previous step
    elif menu == "2" :
        val2 = get_avrg(list_of_humi)
        stringOfNums += str(val2)+" "

    elif menu == "3" :
        val3 = max(list_of_humi)
        stringOfNums += str(val3) +" "

    elif menu == "4" :
        val4 = min(list_of_temp)
        stringOfNums += str(val4) +" "

    elif menu == "5" :
        val5 = (get_avrg(list_of_temp))
        stringOfNums += str(val5)+" "

    elif menu == "6" :
        val6 = max(list_of_temp)
        stringOfNums += str(val6)+" "

    elif menu == "7" :
        stringOfNums = convert_to_str(list_of_humi)

    elif(menu=="8"):
        stringOfNums = convert_to_str(list_of_temp)

    print(" number is: ",stringOfNums) #print the string to test
    client.send(stringOfNums.encode()) # send that string to the client
    sleep(.5) # sleep for a sec
    client.close() # close  client
    server.close() # Close Server
    serverObject.close() #  close the object


if __name__ == "__main__":
```

```
    main()
    #print("Testing")
    #print(Sensor("temperature"))
    #e = get_avrg([3,3,3])
    #f = convert_to_str([3,3,3])
    #print(f)
    #s = Sensor("temperature")
    #print(s)
```

FIG 3: Above is the Main class written in Python code that receives requests and sends information to the client.

```
package clientserver;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.InetAddress;
import java.net.Socket;
import javax.swing.*;

/*
 * Authors -        William, Daniel, TAflouk
 * Date -           28/11/2022
 * Last Modified -  30/11/2022 by William Hogan
 *
 * Create a connection to a server.
 * Connect over a socket to send requests and receive Temperature/Humidity
readings back from the Server.
 * Create a GUI with buttons to send requests
 * and display results of readings received from the server.
 * Change the Background color on the GUI based on the request sent.
 *      - minTemp, minHum = Blue
 *      - avgTemp, avgHum = Green
 *      - maxTemp, maxHum = Red
 *      fullTemp, fullHum = Yellow (this was optional)
 *
 */
@SuppressWarnings("serial")
public class ClientGUI  extends JFrame implements ActionListener{
```

```
    //Create Panel to add to the JFrame.
    private static JPanel panel = new JPanel();
    //Create Label and Button components to add to the Panel.
    private JLabel tempLabel = new JLabel("Temperature Readings");
    private JLabel humLabel = new JLabel("Humidity Readings");
    private JButton minTemp = new JButton("get min temp");
    private JButton minHum = new JButton("get min Hum");
    private JButton avgTemp = new JButton("get avg temp");
    private JButton avgHum = new JButton("get avg Hum");
    private JButton maxTemp = new JButton("get max temp");
    private JButton maxHum = new JButton("get max Hum");
    private JButton tempList = new JButton("full temp list");
    private JButton humList = new JButton("full Hum list");

    //Create textfields to display the readings received from the server.
      private JTextField minTempField = new JTextField();
      private JTextField minHumField = new JTextField();
      private JTextField avgTempField = new JTextField();
      private JTextField avgHumField = new JTextField();
      private JTextField maxTempField = new JTextField();
      private JTextField maxHumField = new JTextField();
      private JTextField fullTempField = new JTextField();
      private JTextField fullHumField = new JTextField();

    //Button to close the GUI and exit the system.
    private JButton exit = new JButton("exit");

    //Constructor to build the GUI.
    public ClientGUI() {

        // Call to super constructor inherited from JFrame class. Must be
first line in constructor.
        super();

        //set up JFrame with title.
        JFrame fr = new JFrame("Sensor readings");

        //Set properties for the JFrame.
        fr.setResizable(false);
        fr.setSize(460, 430);
        fr.setLocation(350, 100);
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fr.setVisible(true);

        //Set properties for the Panel.
        panel.setLayout(null);
        panel.setSize(460, 430);
```

```
        //Set the size and location of the components in the Panel.
        tempLabel.setBounds(100, 0, 160, 30);
        minTemp.setBounds(20, 30, 120,25);
        minTempField.setBounds(150, 30, 160, 25);
        avgTemp.setBounds(20, 60, 120,25);
        avgTempField.setBounds(150, 60, 160, 25);
        maxTemp.setBounds(20, 90, 120,25);
        maxTempField.setBounds(150, 90, 160, 25);
        tempList.setBounds(20, 120,120,25);
        fullTempField.setBounds(150, 120, 280, 25);

        humLabel.setBounds(110, 180, 160, 30);
        minHum.setBounds(20, 210, 120, 25);
        minHumField.setBounds(150, 210, 160, 25);
        avgHum.setBounds(20,240, 120, 25);
        avgHumField.setBounds(150, 240, 160, 25);
        maxHum.setBounds(20, 270, 120, 25);
        maxHumField.setBounds(150, 270, 160, 25);
        humList.setBounds(20, 300, 120,25);
        fullHumField.setBounds(150, 300, 280, 25);

        exit.setBounds(110, 360, 120, 25);

        //Add the components to the Panel.
        panel.add(tempLabel);
        panel.add(humLabel);
        panel.add(minTemp);
        panel.add(minTempField);
        panel.add(minHum);
        panel.add(minHumField);
        panel.add(avgTemp);
        panel.add(avgTempField);
        panel.add(avgHum);
        panel.add(avgHumField);
        panel.add(maxTemp);
        panel.add(maxTempField);
        panel.add(maxHum);
        panel.add(maxHumField);
        panel.add(tempList);
        panel.add(fullTempField);
        panel.add(humList);
        panel.add(fullHumField);

        panel.add(exit);

        //Add the Panel to the JFrame.
        fr.getContentPane().add(panel);
```

```
        //Add ActionListener to all buttons
        minTemp.addActionListener(this);
        minHum.addActionListener(this);
        avgTemp.addActionListener(this);
        avgHum.addActionListener(this);
        maxTemp.addActionListener(this);
        maxHum.addActionListener(this);
        tempList.addActionListener(this);
        humList.addActionListener(this);

        exit.addActionListener(this);
    }
    //Implementation of the abstract ActionPerformedEvent method in
    //the ActionListener Interface.
    @Override
    public void actionPerformed(ActionEvent e) {
        //Target Object to get the source of user action (mouse).
        //Tab and Spacebar also work here by default.
        Object target = e.getSource();
        //If user clicks exit button.
        if(target == exit) {
            System.exit(0);
        }
        //All buttons share the same implementation with the exception
        //of the panel color display being different depending on the button.

        //If user clicks minTemp button.
        if(target == minTemp) {
            //set panel color to cyan/blue.
            panel.setBackground(Color.cyan);
            //try block to receive data from the server.
            try {
            //Call getValueFromServer method, passing the panel and String 4
as arguments.
                //4 will be converted to a byte.
                String data = getValueFromServer(panel,"4");
                //display the String returned in the textfield.
                minTempField.setText(data);
            //Catch an input or output exception.
            } catch (IOException e1) {
                e1.printStackTrace();
            //Catch an interrupted connection exception.
            } catch (InterruptedException e1) {
                e1.printStackTrace();
            }
        }
        //if user clicks avgTemp button.
```

```
        if(target == avgTemp) {
            panel.setBackground(Color.green);
            try {
                String data = getValueFromServer(panel,"5");
                avgTempField.setText(data);
            } catch (IOException e1) {
                e1.printStackTrace();
            } catch (InterruptedException e1) {
                e1.printStackTrace();
            }
        }
        //if user clicks maxTemp button.
        if(target == maxTemp) {
            panel.setBackground(Color.red);
            try {
                String data = getValueFromServer(panel,"6");
                maxTempField.setText(data);
            } catch (IOException e1) {
                e1.printStackTrace();
            } catch (InterruptedException e1) {
                e1.printStackTrace();
            }
        }
        //if user clicks tempList button.
        if(target == tempList) {
            panel.setBackground(Color.yellow);
            try {
                String value = getValueFromServer(panel,"8");
                fullTempField.setText(value);
            } catch (IOException e1) {
                e1.printStackTrace();
            } catch (InterruptedException e1) {
                e1.printStackTrace();
            }
        }
        //if user clicks minHum button.
        if(target == minHum) {
            panel.setBackground(Color.cyan);
            try {
                String data = getValueFromServer(panel,"1");
                minHumField.setText(data);
            } catch (IOException e1) {
                System.out.println("IO Exception");
                e1.printStackTrace();
            } catch (InterruptedException e1) {
                System.out.println("Interrupted Exception");
                e1.printStackTrace();
            }
```

```
        }
        //if user clicks avgHum button.
        if(target == avgHum) {
            panel.setBackground(Color.green);
            try {
                String data = getValueFromServer(panel,"2");
                avgHumField.setText(data);
            } catch (IOException e1) {
                System.out.println("IO Exception");
                e1.printStackTrace();
            } catch (InterruptedException e1) {
                System.out.println("Interrupted Exception");
                e1.printStackTrace();
            }
        }
        //if user clicks maxHum button.
        if(target == maxHum) {
            panel.setBackground(Color.red);
            try {
                String data = getValueFromServer(panel,"3");
                maxHumField.setText(data);
            } catch (IOException e1) {
                System.out.println("IO Exception");
                e1.printStackTrace();
            } catch (InterruptedException e1) {
                System.out.println("Interrupted Exception");
                e1.printStackTrace();
            }
        }
        //if user clicks humList button.
        if(target == humList) {
            panel.setBackground(Color.yellow);
            try {
                String data = getValueFromServer(panel,"7");
                fullHumField.setText(data);
            } catch (IOException e1) {
                System.out.println("IO Exception");
                e1.printStackTrace();
            } catch (InterruptedException e1) {
                System.out.println("Interrupted Exception");
                e1.printStackTrace();
            }
        }

    }
    //Method to create connection to the server. Sends byte request across
socket and receives back information.
    //Takes the information received and returns it as a String.
```

```
    public static String getValueFromServer(JPanel panel,String menu) throws
IOException, InterruptedException  {
        //set up address of server
        InetAddress inet = InetAddress.getByName("192.168.1.102");
        //create socket using server ip address and port....
        Socket s = new Socket(inet, 2003);
        // send the menu value to the server
        DataOutputStream writeToServer = new
DataOutputStream(s.getOutputStream());
        writeToServer.writeBytes(menu);

        //wait patiently....
        Thread.sleep(200);
        //input from the socket (s)
        InputStream in = s.getInputStream();

        BufferedReader inputLine = new BufferedReader(
        new InputStreamReader(s.getInputStream()));
        //store the returned result in a string variable for use.
        String result = inputLine.readLine();
        //Close the socket.
        s.close();
        return result;
    }
    public static void main(String[] args) {

        new ClientGUI();

    }
}
```

FIG 4: Above is the Client code written in Java that sends requests to the Server and displays the information received in the GUI.

Verification

The expectation when running the Application is to firstly receive the correct readings from the Sensors. These readings will be stored in a list on the server. The server then listens for a connection on the assigned port number and when it receives a request for data, it will return the associated value/values over the socket after it has been converted to a string.

The server has functions that return:

- Minimum temperature
- Average temperature
- Maximum temperature
- Full temperature list
- Minimum humidity
- Average humidity
- Maximum humidity
- Full humidity list

The Java Client uses a GUI with buttons for sending requests and textfields for displaying the values returned by the Server. Each button has a string assigned which is firstly converted to a byte and then sent over the socket to the Server. The server decodes this byte and retrieves the corresponding value before encoding that value requested and sending back over the socket. The client then receives the value and converts it to a string before displaying it in the associated textfield for the user.

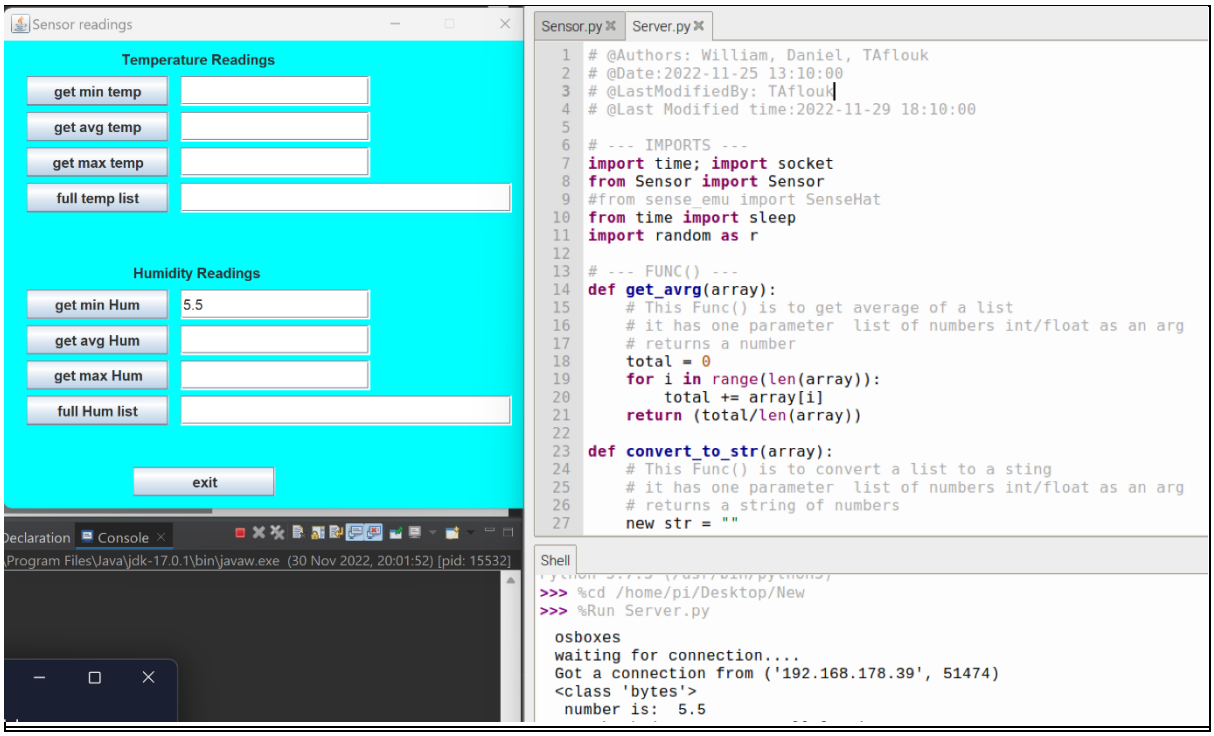

FIG 5: Screenshot of minimum humidity reading request displayed on client GUI.

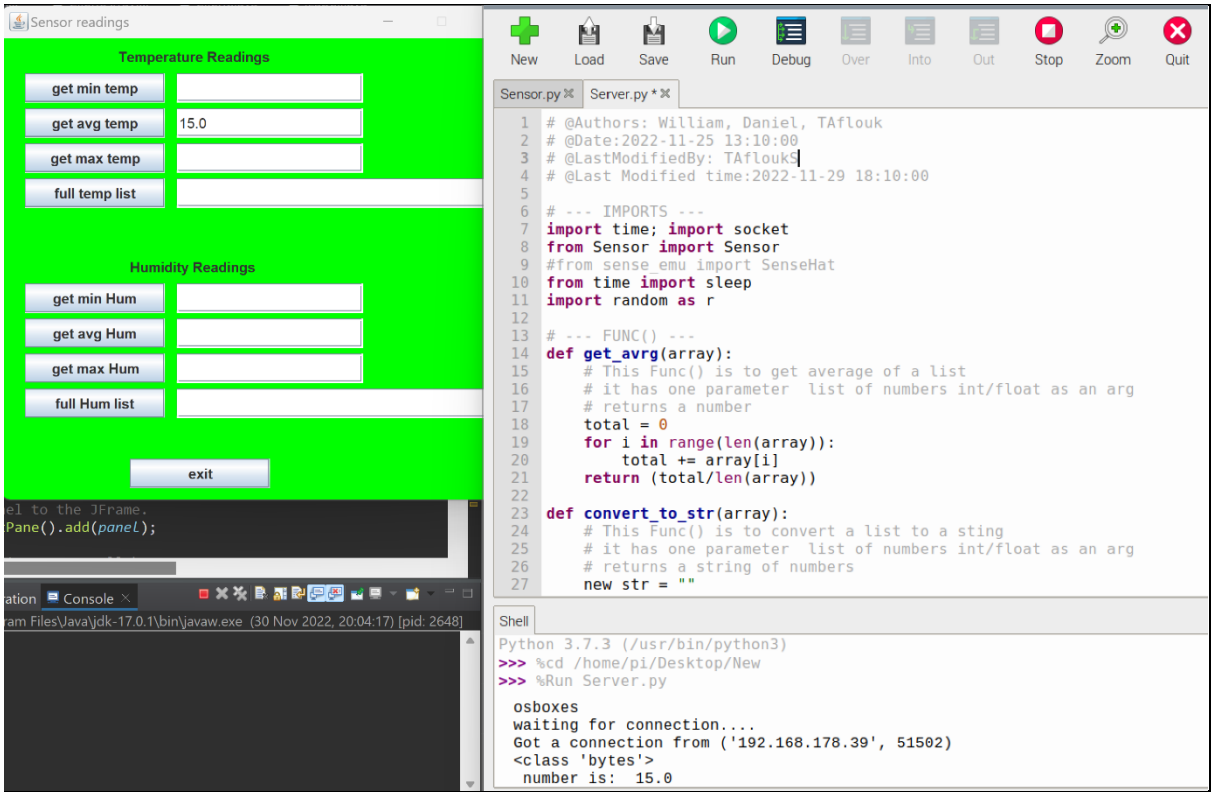

FIG 6: Screenshot of average temperature reading request displayed on client GUI.

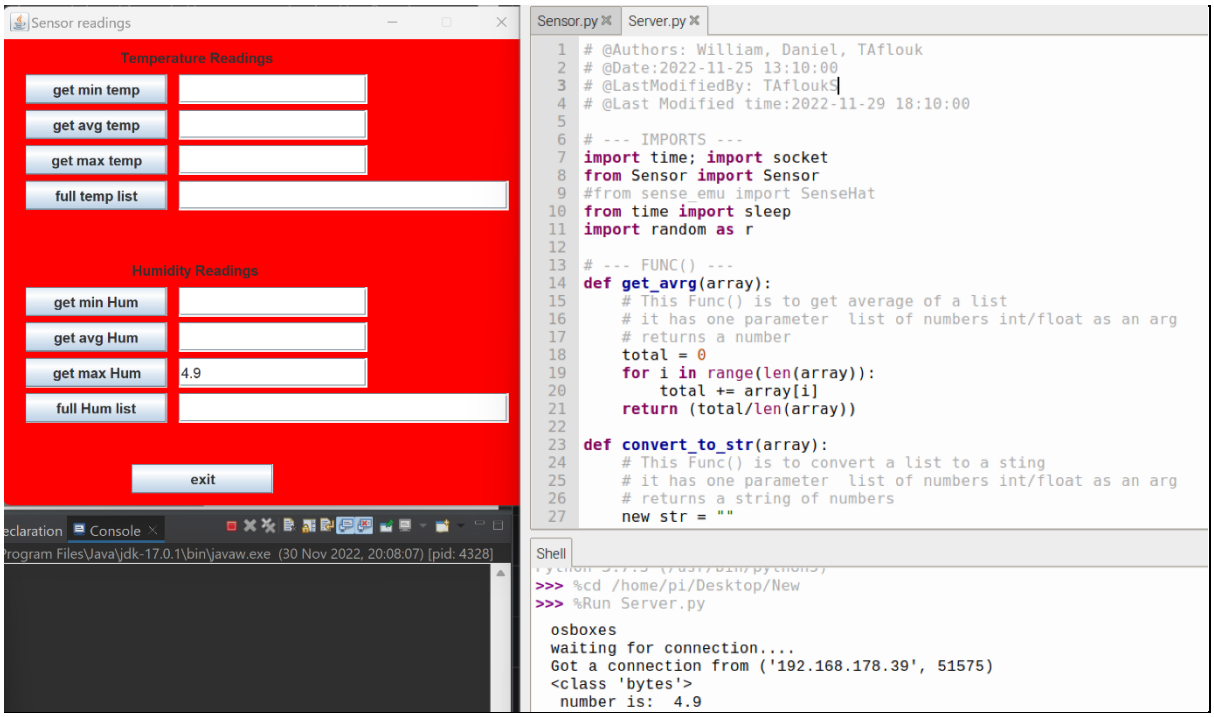

FIG 7: Screenshot of maximum humidity reading request displayed on client GUI.

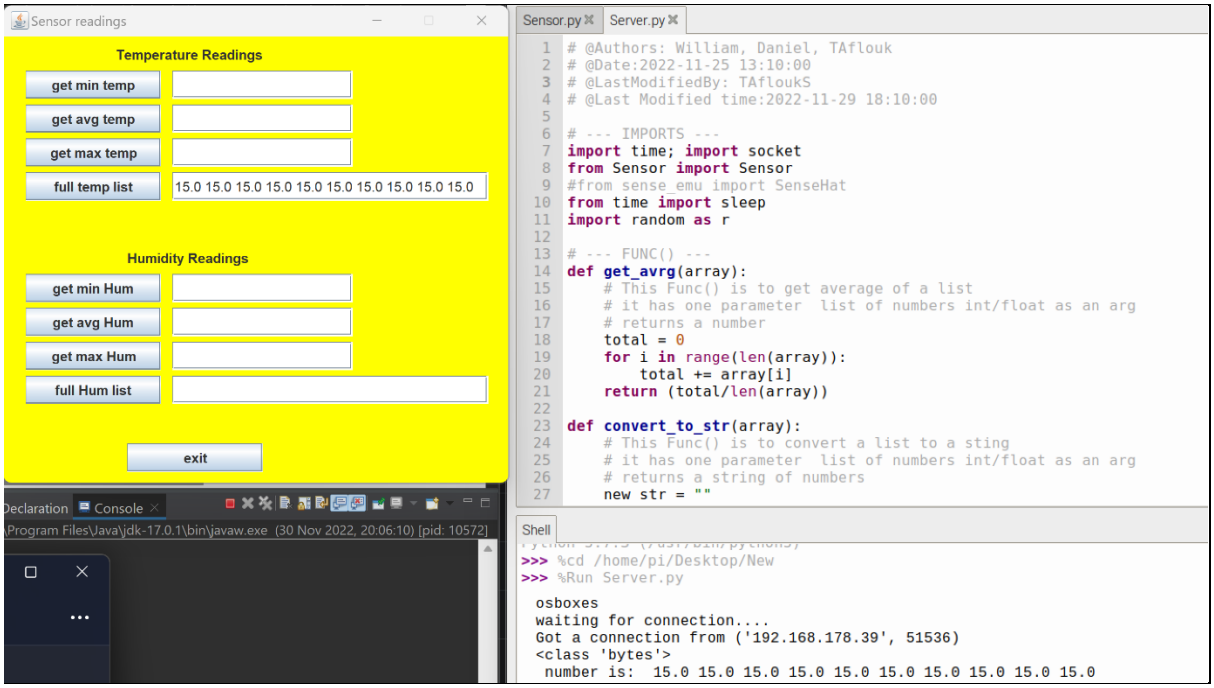

FIG 8: Screenshot of full temperature list displayed on client GUI.

Team Effort Contribution

Name	Architecture & Design	Implementation	Verification
Taha Aflouk			
Daniel Glynn			
William Hogan			