A picture containing text, clipart

Description automatically generated**STUDENT PLAGIARISM DISCLAIMER FORM**

***PLAGIARISM DISCLAIMER***

STUDENT NAME: \_\_Nathan Cleary\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

STUDENT NUMBER: A00273290\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

PROGRAMME: BSc in Software Development (Mobile Apps and Connected Devices)\_\_\_\_\_\_\_\_\_

YEAR: \_3rd\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

MODULE: \_Software Development for Connected Devices 3\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

LECTURER: \_\_Mary Pidgeon\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ASSIGNMENT TITLE:  \_Assignment 2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

DUE DATE: \_\_\_3/12/21\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

DATE SUBMITTED: \_\_\_\_\_\_1/12/21\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ADDITIONAL INFORMATION: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

I understand that plagiarism is a serious academic offence, and that AIT deals with it according to the AIT Policy on Plagiarism.

I have read and understand the AIT Policy on Plagiarism and I agree to the requirements set out therein in relation to plagiarism and referencing. I confirm that I have referenced and acknowledged properly all sources used in preparation of this assignment.  I understand that if I plagiarise, or if I assist others in doing so, that I will be subject to investigation as outlined in the AIT Policy on Plagiarism.

I understand and agree that plagiarism detection software may be used on my assignment. I declare that, except where appropriately referenced, this assignment is entirely my own work based on my personal study/or research. I further declare that I have not engaged the services of another to either assist in, or complete this assignment.

***Signed:*** Nathan Cleary

**Dated: 1/12/21**

**Assignment Cover Sheet**

**Software Development for Connected Devices 3**

Student Name: Nathan Cleary

Student Number: A00273290

Student Signature: Nathan Cleary

**Assignment 2**

Date Submitted: \_1/12/21\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Introduction**

For this assignment, our goal was to create an MQTT server that will host our home humidity and temperature monitoring application that will monitor the temperature and humidity in several rooms in a home. The temperature and humidity data will be collected using the sensehat on the Raspberry Pi. The raspberry Pi will then publish the data to the MQTT broker. The MQTT broker using mosquito service, will be hosted on the users’ PC along with the java program to subscribe to the topic on the broker and will then visually display that data to the user.

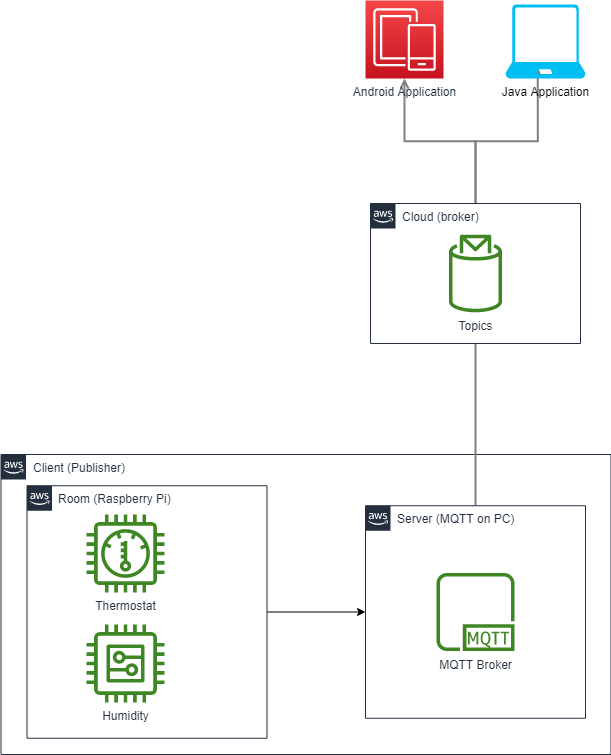
In the rest of this report, I outline the architecture and design of the system, implementation and the code, verification on the system working and future work on the system that could be done.

**Architecture and Design**

The architecture and design of the system composes of multiple parts. The main three being the publisher, the broker, and the subscribers. The data is collected by the sensors on the raspberry pi. The raspberry pi then publishes the data to the broker and then its stored within a topic.

A topic is a specific group or collection of messages to be sent and taken from and it organises them so you can have multiple connections going through the broker and they won’t get mixed up. For example, you could have one topic for the temperature and humidity for one room and other topic for another room. In this project I’ve only used one topic to represent the one room the readings are coming from.

The subscriber in my assignment is going to be the Java Application. The java application takes the data that has been published to a topic and displays them to the user in the form of a bar chart using Java Swing. The values of the topic are also printed to the console to be viewed there also.

****

**Implementation**

simpleCallback.java

public class SimpleCallback /\*implements MqttCallback\*/ {  
 static String[] *tempReadings* = new String[]{"temp/30/hum/40","temp/29/hum/45","temp/38/hum/30","temp/35/hum/35"};  
 static int *arrayPos* = 0;  
 static int *temperature*;  
 static int *humidity*;  
 static public ArrayList<Integer> *tempArray* = new ArrayList<>();  
 static public ArrayList<Integer> *humArray* = new ArrayList<>();  
  
 public static void main(String[] args){  
  
 /\* String topic = "test";  
 String broker = "tcp://192.168.1.100:1883";   
 String clientId = "Java Subscriber";

try {  
 MqttClient sampleClient = new MqttClient(broker, MqttClient.generateClientId());  
 MqttConnectOptions connOpts = new MqttConnectOptions();  
 connOpts.setCleanSession(true);  
 System.out.println("Connecting to broker: " + broker);  
 sampleClient.connect(connOpts);  
  
 sampleClient.setCallback(new SimpleCallback());  
 sampleClient.subscribe(topic, 1);  
 System.out.println("Connected");  
 while(true){  
 try {  
 Thread.sleep(10000);  
 System.out.println("did i get a message?" + topic);  
 sampleClient.  
 sampleClient.disconnect();  
 } catch(Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 System.out.println("Disconnected");  
 System.exit(0);  
 } catch(MqttException e){  
 e.printStackTrace();  
 }\*/  
 new GUI();  
 }  
 //Called when the client lost the connection to the broker  
 // we can override it with our own functionality but for now we do nothing  
 public void connectionLost(Throwable cause) {  
 }

// Use for when a message arrives via the broker from the publisher of  
 // the topic we subscribed to - room1/sensehat/temp.  
 // We override the method with what we would like to do with the value of the  
 // temp received  
 // For now we get the payload (the temp) and parse it from String to double  
  
 // using this method since mqtt doesnt run correctly  
 public static void messageArrived1(){  
 //take in one value from the hardcoded values  
 String messageRecieved = *tempReadings*[*arrayPos*];  
 System.*out*.println(messageRecieved);  
  
 //split the string into an array. the temp should be at index 1, and humidity at index 3  
 String[] values = messageRecieved.split("/");  
 *temperature* = Integer.*parseInt*(values[1]);  
 *humidity* = Integer.*parseInt*(values[3]);  
  
 //add the values into an array list to be taken and displayed by the gui  
 *tempArray*.add(*temperature*);  
 *humArray*.add(*humidity*);

// if there is still more elements in the array, call method again to take the next value  
 if(*arrayPos* < *tempReadings*.length - 1){  
 *arrayPos*++;  
 *messageArrived1*();  
 }  
 }  
  
 // use this method if mqtt does run correctly  
 //@Override  
 public void messageArrived(String topic, MqttMessage message) throws Exception {  
  
 // Print the topic, the String message and the double value  
  
 System.*out*.println("-------------------------------------------------");  
 System.*out*.println("| Topic:" + topic);  
 System.*out*.println("| Message: " + new String(message.getPayload()));  
 System.*out*.println("-------------------------");  
  
 // convert the payload recieved through mqtt into a string  
 String stringValue = new String(message.getPayload());  
  
 //split the string into an array. the temp should be at index 1, and humidity at index 3  
 String[] values = stringValue.split("/");  
 *temperature* = Integer.*parseInt*(values[1]);  
 *humidity* = Integer.*parseInt*(values[3]);  
  
 //add the values into an array list to be taken and displayed by the gui  
 *tempArray*.add(*temperature*);  
 *humArray*.add(*humidity*);  
  
 // Create a Room object with the temp value  
 //which has a getter and setter for the Temperature value and Id for the reading  
  
 //Rooms r1= new Rooms(value);  
  
 // Get the temp value and the Id for the  
 //which has a getter and setter for the Temperature value  
  
 //System.out.println("The temperature for reading number "+r1.getId()+ " is "+r1.getTemp());  
  
 }  
  
 // One of the three Interface MqttCallback's methods that must be included  
 // Called when an outgoing publish is complete  
 // As this client is only subscribing, this method is not used but  
 // it must be included as it is one of the interface MqttCallBack Methods  
 // @Override  
 public void deliveryComplete(IMqttDeliveryToken token) {  
 }  
}

GUI.java

public class GUI extends JFrame {  
 private JLabel temp= new JLabel();  
 private JLabel hum = new JLabel();  
 GridBagLayout gridBagLayout = new GridBagLayout();  
 GridBagConstraints c = new GridBagConstraints();  
  
 GUI(){  
 setSize(500,500);  
 setVisible(true);  
 getContentPane().setLayout(gridBagLayout);  
  
 // get values needed from the subscriber client  
 SimpleCallback.*messageArrived1*(); /\* If mqtt subscriber works remove this line. This puts hardcoded values into array \*/  
 ArrayList<Integer> tempArray = SimpleCallback.*tempArray*;  
 ArrayList<Integer> humArray = SimpleCallback.*humArray*;  
  
 // for each element in the array pass the temp, humidity and the column that the temp is in into the createColumn method  
 int colNum = 0;  
 for(int temper : tempArray){  
 int humid = humArray.get(tempArray.indexOf(temper));  
 createColumn(temper, humid, colNum);  
 // since 3 columns are created (temp, humidity, and a space for the next block) add 3 to the columns  
 colNum = colNum + 3;  
 }  
 // add the label to the bottom  
 JLabel label = new JLabel("Red = Temperature, Blue = Humidity");  
 c.gridy = 3;  
 c.gridx = 0;  
 c.ipady = 1;  
 c.gridwidth = tempArray.size() \* 3;  
 c.anchor = GridBagConstraints.*PAGE\_END*;  
 gridBagLayout.setConstraints(label, c);  
 add(label);  
  
 System.*out*.println(tempArray);  
 System.*out*.println(humArray);  
 }  
  
 private void createColumn(int temper, int humid, int colNum) {  
 // create variables  
 JPanel temp = new JPanel();  
 JPanel hum = new JPanel();  
 JLabel tempLabel = new JLabel();  
 JLabel humLabel = new JLabel();  
 JPanel space = new JPanel();  
  
  
 // set the temp background to red and the humidity background to blue  
 temp.setBackground(Color.*red*);  
 hum.setBackground(Color.*blue*);

// everything is anchored to the bottom  
 // add the temperature column, the height is the temperature value scaled up by 5. the x value is 1 and the y value is the Array (index \* 3)  
 // the width is 10. the constraits are added to the JLabel and then the JLabel is added to the JFrame  
 c.gridy = 1;  
 c.gridx = colNum;  
 c.ipady = temper \* 5;  
 c.ipadx = 10;  
 c.anchor = GridBagConstraints.*PAGE\_END*;  
 gridBagLayout.setConstraints(temp, c);  
 add(temp);  
  
 // set the label below the bar to show the temperature value  
 tempLabel.setText("" + temper);  
 c.anchor = GridBagConstraints.*PAGE\_END*;  
 c.ipady = 1;  
 c.gridy = 2;  
 c.gridx = colNum;  
 gridBagLayout.setConstraints(tempLabel, c);  
 add(tempLabel);  
  
 // add the humid column, the height is the humid value scaled up by 5. the x value is 1 and the y value is one to the right of the previous  
 // the width is 10. the constraints are added to the JLabel and then the JLabel is added to the JFrame  
 c.gridy = 1;  
 c.gridx = colNum + 1;  
 c.ipady = humid \* 5;  
 c.ipadx = 10;  
 gridBagLayout.setConstraints(hum, c);  
 add(hum);  
  
 // set the label below the bar to show the humidity value  
 humLabel.setText("" + humid);  
 c.anchor = GridBagConstraints.*PAGE\_END*;  
 c.ipady = 1;  
 c.gridy = 2;  
 c.gridx = colNum + 1;  
 gridBagLayout.setConstraints(humLabel, c);  
 add(humLabel);  
  
 // add a blank column to separate the the current set of readings and the next set of readings  
 c.gridy = 1;  
 c.gridx = colNum + 2;  
 c.ipady = humid \* 5;  
 c.ipadx = 10;  
 gridBagLayout.setConstraints(space, c);  
 add(space);  
 }  
}

**myMQTT.py**

from sense\_hat import SenseHat

import paho.mqtt.client as mqtt

import time

# set up the broker information into variables

broker = "192.168.0.108"

port = 1883

topic = "test"

# set up the sensehat

sense = SenseHat()

sense.clear()

# connect to the broker

client1 = mqtt.Client("RPI123")

client1.connect(broker, port, 60)

while True:

# get the temperature and humitidy readings from the sensehat

temp = sense.get\_temperature()

hum = sense.get\_humidity()

# round the values gotten from the sensehat

h = round(hum)

t = round(temp)

# put the values gotten into a string

msg = "/temp/" +str(t) +"/hum/" + str(h)

#publish the string into the topic on the broker

client1.publish(topic, msg)

#show the string as a message onto the screen on the sensehat and print to the console

sense.show\_message(msg, scroll\_speed=0.05)

print(msg)

time.sleep(5)

**Verification**

During the process of developing the system, I have encountered an issue that the subscriber clients would not receive the information published to the topic. The only method I have tried that works is having both the publisher and subscriber clients on the command terminal and while they are on the same device as the broker. Publishers outside of these seem to work as no errors are given but any subscribers outside of these do not function at all. I have tried all my devices, and this is the case for all. I have also set up the broker on the raspberry pi and have still gotten the same results.

Due to time constraints, I have adopted a different method to test if the subscribers still work by hardcoding received values to simulate the system working. The recievedMessage1 method is now called to take in these hardcoded values. I have also left in the code for the working subscriber client it is commented out to avoid errors. The visual representation of the values works perfectly and shows the correct values.

Chart, bar chart

Description automatically generated

**Future Work**

In the upscale of this system to add more rooms you will need to have more sensors meaning we need a raspberry Pi in each room. This of course raises the complexity and the way the topics work needs to change. You can add an identifier for the room within the message sent to the topic. Eg. “room/myRoom/temp/30/hum/40”. But the easiest way to add multiple room is to add multiple topics. Each room will have its own topic and making it easier to organise and control the traffic going through them. The client will then also display all reading by having a separate bar chart or graph for each room. A further way of developing this system is to implement more client subscribers to show the data. This can be done on other java programs on other machines, or on an android application.

**Diagram

Description automatically generated**