SOFE 4610-Fall 2022
Design and Analysis of IoT Software Systems
Project Final

Title: Greenhouse Humidity and Temperature Sensing System

Dr. Ramiro Liscano
12/06/2022

Name	Student ID
Minhal Syed	100618744
Hemshikha Sultoo	100670616
Shahroze Butt	100701891

Table of Contents

1. Project Description

This IoT project is intended to address issues related to humidity and temperature in indoor environments such as greenhouses. For this project, the system is being developed to operate wirelessly and the user will be able to communicate with the device through an app. The system will communicate over the MQTT server to convey messages to the user using the telemetry IoT communication style. Depending on the temperature and humidity changes, the system will send notifications to the user suggesting what might be the scenario that is causing the changes. For example, a sudden spike in humidity supported by low temperatures over a considerably long period of time might suggest that a door or window may be open.

2. Background

Our current project involves monitoring the temperature inside a greenhouse to ensure that it is always maintained at an ideal level of 27 °C/80 °F. However, it should only reach a maximum temperature of 32 °C/90 °F and a minimum of 24 °C/75 °F [1]. Since temperature and humidity have an inverse relationship, they can cooperate to ensure that plant health is constantly monitored to support healthy plant growth.

3. Major Components

Due to the numerous difficulties and mistakes we encountered while designing our system. Our device of choice was an ESP8266 Nodemcu. The reason behind this is because using the bundled Arduino uno and wifi shield, given did not significantly affect the system's compatibility or latency and we wanted to expedite the collection of data. Furthermore, the Nodemcu was attached to the breadboard, and a 10k ohm resistor was used to link the DHT11 sensor directly to the board. While the servo motor was directly connected to the ESP8266. It enables the system to understand the data, the sensor's collected data was transferred to the cloud. To allow us to provide the data sets, there was a connected Arduino IDE available. Additionally, the user would be able to view graphs of the collected data. In addition to the controls we gave the system, the servo motor had its own button that the user could use to manually switch it on and off.

While loops were used in the programming to create the cases. If the temperature had risen and the fan had sensed that it had reached a certain temperature, it would run through a series of different scenarios. It would show a warning to the user to allow them to close the greenhouse door if the temperature had dropped because it would have appeared that the door was open and letting heat escape. It would simply monitor the temperature if there was no temperature flux, and nothing would happen. Once the environment's temperature has been regulated, the fan gradually turns off.

Use Cases	Details
UC-1: Increase in temperature	The infrastructure is continuously monitored by the system to check for sudden increases in humidity or decreases in temperature. When the temperature or humidity hasn't returned to normal, the system collects the data and turns the fan on.
UC-2: Decrease in Temperature	When there is an abrupt rise in temperature and fall in humidity, when it keeps rising above a particular point, and when it hasn't returned to normal. The user would receive a warning to close the door to the green house.
UC-3: Stable environment	While data is being gathered and the environment hasn't changed all that much. The user could assess the app's stability by checking it.

4. Diagram of the System

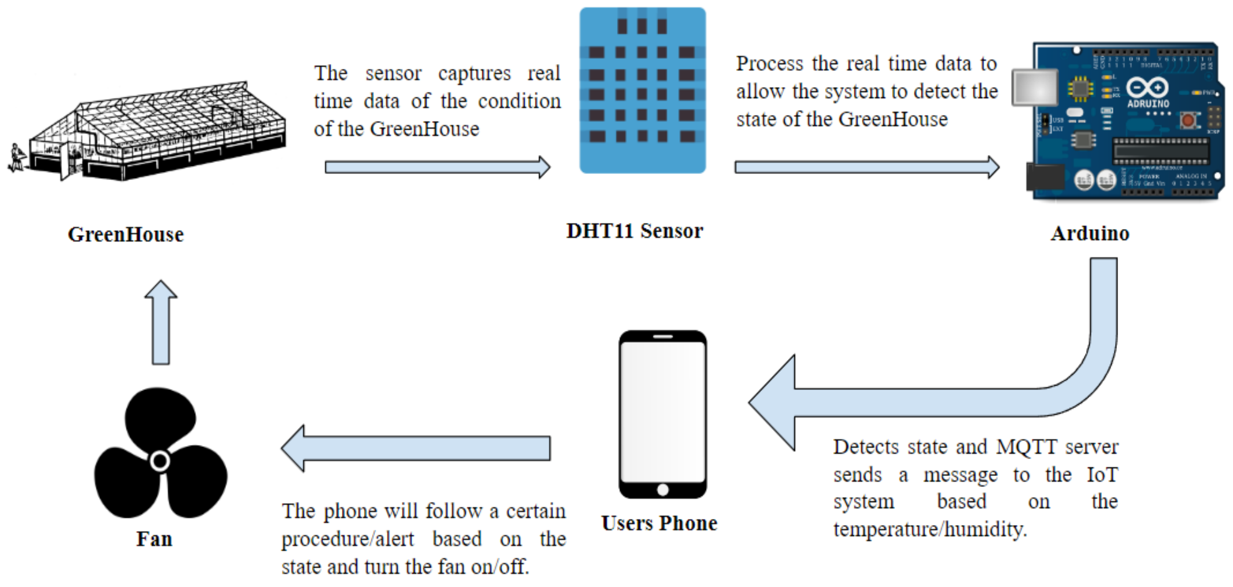

The sensor captures real time data of the condition of the GreenHouse
Process the real time data to allow the system to detect the state of the GreenHouse
GreenHouse
DHT11 Sensor
Arduino
Fan
Users Phone
Detects state and MQTT server sends a message to the IoT system based on the temperature/humidity.
The phone will follow a certain procedure/alert based on the state and turn the fan on/off.

Figure 1.0: *The interaction of the system when used in a live environment*

5. Flow Chart

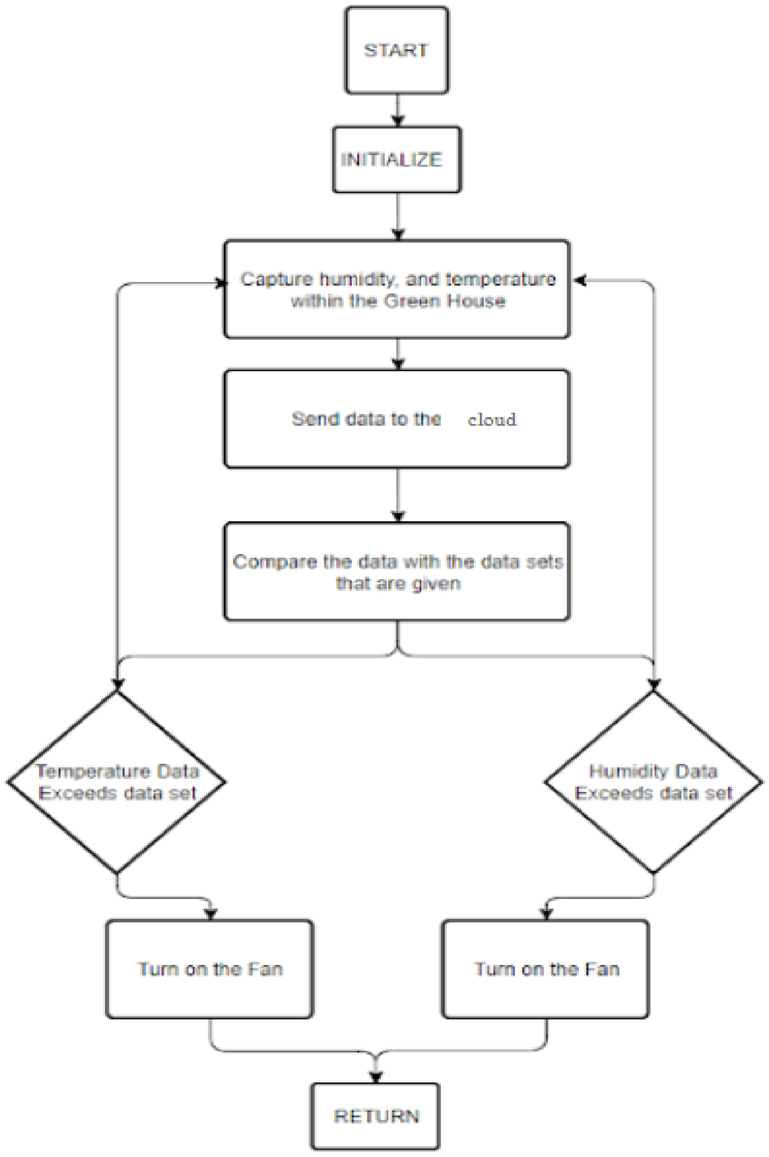

START
INITIALIZE
Capture humidity, and temperature within the Green House
Send data to the cloud
Compare the data with the data sets that are given
Temperature Data Exceeds data set
Humidity Data Exceeds data set
Turn on the Fan
Turn on the Fan
RETURN

Figure 1.1: *The flow diagram of the system*

The system starts initially by being powered on and initialized. The system captures the humidity, and temperature within the Greenhouse. It sends the data to the cloud and allows it to be graphed and the data is being processed. The data is then compared by the data sets that we provided. The data that is collected is saved on to a database that allows it to be graphed out by the system. Then the system goes through two different actions checking if the temperature and humidity exceed the data set then allowing the fan to turn on. However, if the data sets don’t exceed it goes back to capturing the data and once the humidity and temperature return the fan is set to off.

6. Sequence Diagram

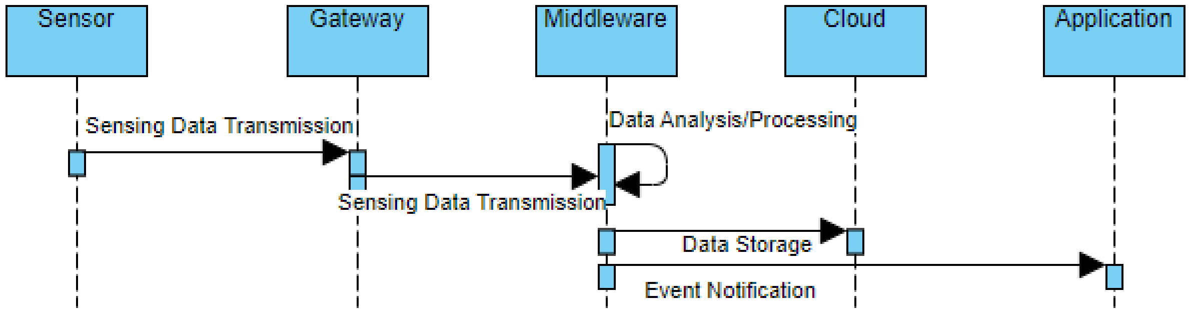


Figure 1.1: *The sequence diagram of event service*

The information was prepared, and it was delivered to the gateway along with the sequence diagram for our system. The gateway either calls for the API request or receives the sensor data, which is the reason. To enable data analysis and to process the data within the constraints imposed by the code, this data is transmitted from the gateway to the middleware. The data is subsequently kept in a database within the cloud. subsequently informing the application of the subsequent phases in the operation.

7. IoT Cloud Platform

Within our system we already had a working embedded system that we could attach with the arduino uno that is included in the box with the raspberry pi. We already have a functioning embedded system within our system that we could connect to the arduino uno that is included in the box with the raspberry pi. The droplet and the connection are then established. Data could then be saved on the database thanks to the droplet's connection to the node-red. The information might then be modified and given constraints based on what the system would need. However, rather than creating a system that required a longer wait time, we wanted it to be shown and submitted quickly.

Then, we considered removing the Raspberry Pi and attempting to connect the system directly from the Arduino. In our investigation, we discovered the ESP8266 Nodemcu, which enables the system to be constructed using Arduino and run on an IoT Cloud that provides data directly. Our setup consisted of a servo motor that was directly connected to the ESP8266, a DHT11 sensor connected to a 10k ohm resistor, and all of this running on a breadboard. We would merely provide the Arduino with the limitations, and once the system was running, it would transmit data over WiFi to the cloud where it would be saved and visualized for the user.

It included a temperature chart, a gauge, and a humidity value. The state of the greenhouse was indicated by a check to show that it was safe and a red flag to show that the temperature wasn't right. Additionally, a manual servo switch was offered to the user, but it was automatically turned on and off in accordance with the information and limitations we provided.

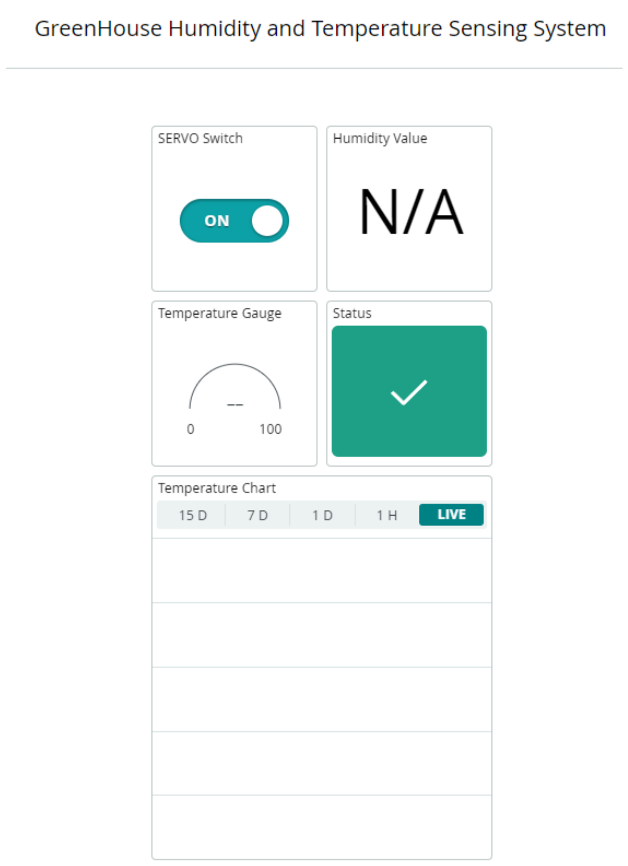


Figure 2.0: *System displaying positive status*

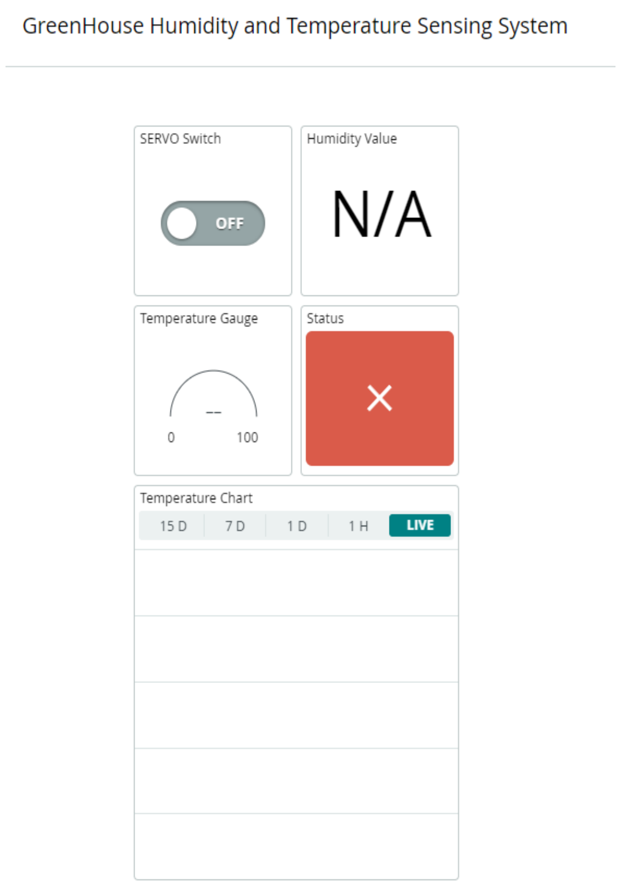


Figure 2.1: *Poor system status is displayed.*

8. Challenges

We had difficulties when designing our system, such as how to build a system without a Raspberry pi. The reason being is allowing a more integrated system that can produce a less latent system. However, when attempting to do so we wanted to use the kit provided that included the Arduino Uno with the usage of the WiFi Shield. We made an effort to establish a direct connection with the wireless connection by providing the SSID and the password directly. Even when we placed the system immediately next to the router, the connection would time out due to latency concerns. We wanted to attempt to attach it to WiFi and host an html page that is provided by Arduino as a URL. This information then displays the sensor's real-time data and serves as a more recent reference. The URL was then to be integrated and pushed to ThingSpeak. The DHT11 sensor's data can be collected and analyzed using ThingSpeak. It would then send commands to the Arduino for the servo to execute in accordance with the data gathered.

Another issue that we had faced was when integrating the ESP8266 we had connected the DHT11 sensor and it produced outputs, and when we solely attached the servo motor it had worked accordingly. However, when attempting to integrate both parts together it had shorted the ESP8266 causing the system to crash. When attempting to reboot it the system wasn't displaying any data thus telling us that we required a new ESP8266 board. However, we didn't have enough time to order another one or find an available board in such a short amount of time. The earliest they would deliver a new board would have been on December 14-17 causing us to go through a plethora of issues. Finally, we attempted to use the new Arduino IDE, but it produced compilation issues (beta version). As a result, we had to continuously track out the appropriate libraries to push along with the functional Arduino.

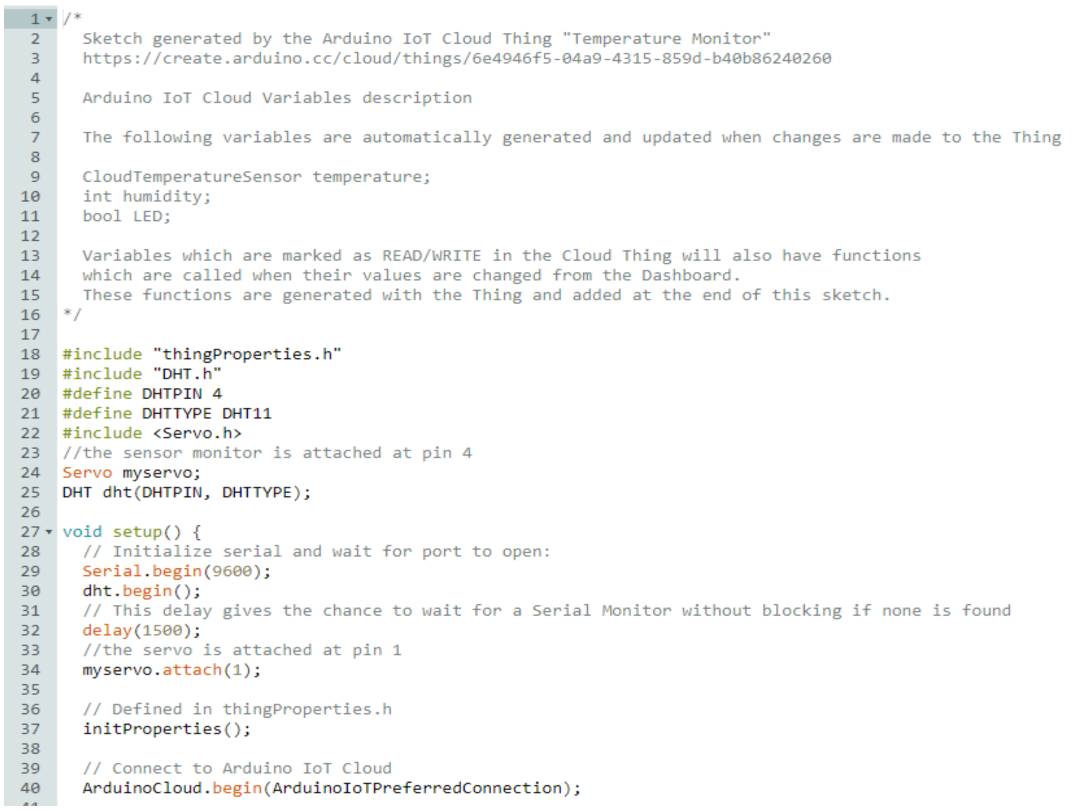
9. GitHub Link

https://github.com/hsultoo/IoTProject.git

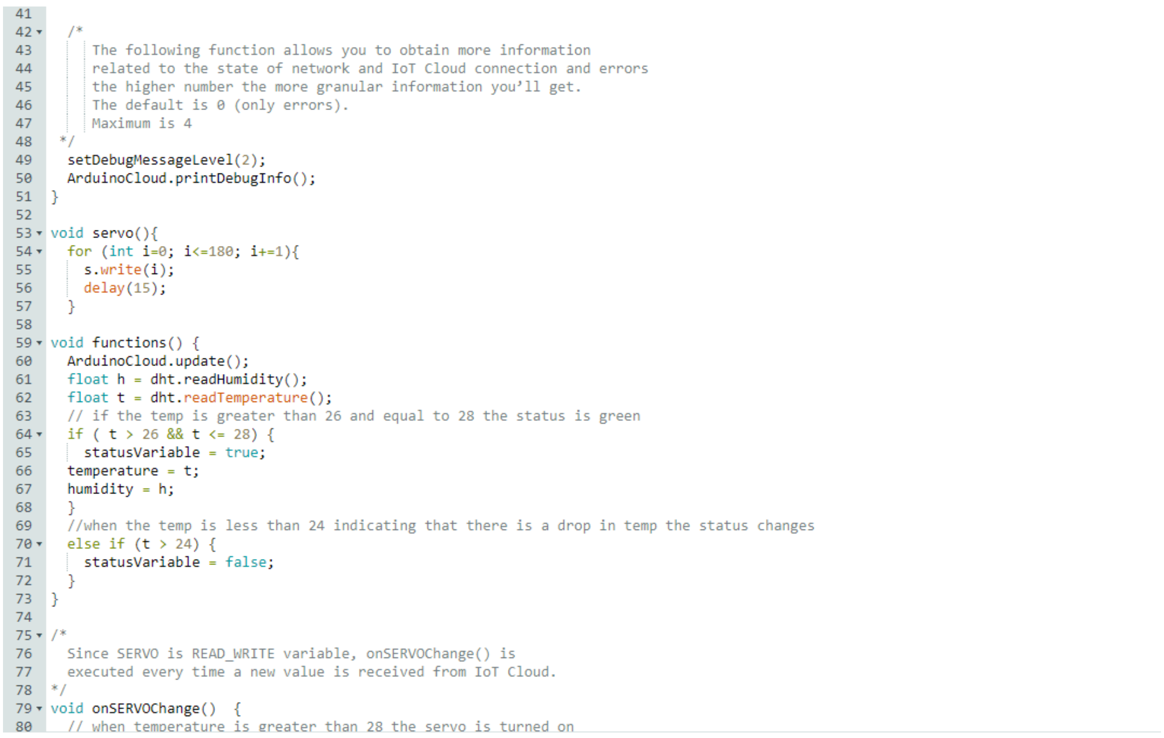
10. PowerPoint Link

https://docs.google.com/presentation/d/1yqzYkQUlRYPkqheB_98lsxAvbc4OfMTzrrUpg9LzQbM/edit?usp=sharing

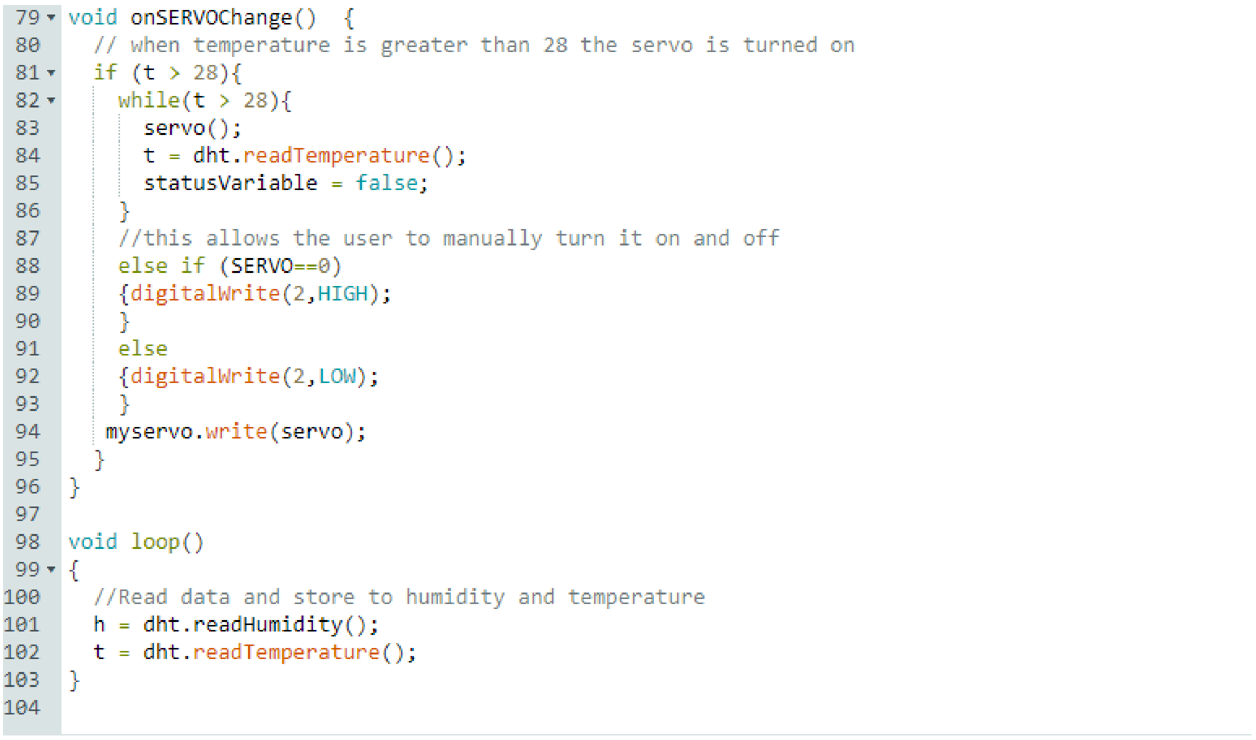
11. Code



```
/*
  Sketch generated by the Arduino IoT Cloud Thing "Temperature Monitor"
  https://create.arduino.cc/cloud/things/6e4946f5-04a9-4315-859d-b40b86240260

  Arduino IoT Cloud Variables description

  The following variables are automatically generated and updated when changes are made to the Thing

  CloudTemperatureSensor temperature;
  int humidity;
  bool LED;

  Variables which are marked as READ/WRITE in the Cloud Thing will also have functions
  which are called when their values are changed from the Dashboard.
  These functions are generated with the Thing and added at the end of this sketch.
*/

#include "thingProperties.h"
#include "DHT.h"
#define DHTPIN 4
#define DHTTYPE DHT11
#include <Servo.h>
//the sensor monitor is attached at pin 4
Servo myservo;
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  // Initialize serial and wait for port to open:
  Serial.begin(9600);
  dht.begin();
  // This delay gives the chance to wait for a Serial Monitor without blocking if none is found
  delay(1500);
  //the servo is attached at pin 1
  myservo.attach(1);

  // Defined in thingProperties.h
  initProperties();

  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
```



```
  /*
    The following function allows you to obtain more information
    related to the state of network and IoT Cloud connection and errors
    the higher number the more granular information you'll get.
    The default is 0 (only errors).
    Maximum is 4
 */
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();
}

void servo(){
  for (int i=0; i<=180; i+=1){
    s.write(i);
    delay(15);
  }

void functions() {
  ArduinoCloud.update();
  float h = dht.readHumidity();
  float t = dht.readTemperature();
  // if the temp is greater than 26 and equal to 28 the status is green
  if ( t > 26 && t <= 28) {
    statusVariable = true;
  temperature = t;
  humidity = h;
  }
  //when the temp is less than 24 indicating that there is a drop in temp the status changes
  else if (t > 24) {
    statusVariable = false;
  }
}

/*
  Since SERVO is READ_WRITE variable, onSERVOChange() is
  executed every time a new value is received from IoT Cloud.
*/
void onSERVOChange()  {
  // when temperature is greater than 28 the servo is turned on
```



```
void onSERVOChange()  {
  // when temperature is greater than 28 the servo is turned on
  if (t > 28){
    while(t > 28){
      servo();
      t = dht.readTemperature();
      statusVariable = false;
    }
    //this allows the user to manually turn it on and off
    else if (SERVO==0)
    {digitalWrite(2,HIGH);
    }
    else
    {digitalWrite(2,LOW);
    }
   myservo.write(servo);
  }
}

void loop()
{
  //Read data and store to humidity and temperature
  h = dht.readHumidity();
  t = dht.readTemperature();
}
```

12. Contribution matrix

Name	**Percentage**	**Work**
Hemshikha	33.3%	- **Coding/ Building device** - **Major Components** - **Device Demonstration Video** - **Format of Report** - **GitHub**
Minhal	33.3%	- **Coding/ Building device** - **Major Components** - **References** - **Slides** - **GitHub**
Shahroze	33.3%	- **Coding/ Building device** - **Flow Chart** - **Sequence Diagram** - **Device Demonstration Diagram** - **Slides** - **GitHub**

13. Reference

[1] N. Vercelletto, “How to get the right temperature and humidity level for healthy plants,” *HappySprout*, 31-Aug-2022. [Online]. Available: https://www.happysprout.com/outdoor-living/greenhouse-temprature-humidity/. [Accessed: 02-Nov-2022].