SMART BABY CARE MONITORING SYSTEM

HARDWARE IMPLIMENTATION REPORT

NAME:-J.ANJANI BABU

REG:-19BPS1081

TABLE OF CONTENTS

1. INTRODUCTION

1.1 OBJECTIVES AND GOALS

- The objective of our project is to build a smart baby-care/monitoring system, which would monitor the various physical parameters around the baby and inform the parents about it through wireless communication.
- We plan to integrate into the system and communicate to the parents, a temperature detector to detect if the temperature is getting higher or lower than certain levels, and a humidity detection mechanism to know if the baby is sweating, and help him/her to go to sleep by regulating the physical environment.
- This system would help the parents to take the best care for their child, even if they are unable to stay in front of the baby all the time.

1.2 APPLICATIONS

1. Used in cases where the baby is alone in the house thus he/she needs to be monitored. This may be due to many reasons:
    a. The parents are at work and there are no other family members.
    b. Even when the parents are at home, due to the inability to be with the baby at all times.
2. The measure of the temperature/humidity helps the parents to understand when the baby is in need of a cold/hot environment, etc. This is due to the fact that people of different age groups have very different senses and the difference between a baby and a parent is very high. Thus the parents will get to know the necessary limits as per the baby's requirement.

1.3 FEATURES

1. TEMPERATURE / HUMIDITY MEASURE:

The temperature and humidity measurements are provided to the user (parents in this case) on the *Arduino IoT Cloud Dashboard*, in the form of a gauge and a chart reading, for both temperature and humidity. In addition to this, a messaging system shows the same real-time temperature and humidity readings in the form of simple lines of text, using a widget to do the same on the *Arduino Iot Cloud Dashboard*. This way, the parents can keep a real-time watch on the baby's surroundings, even if they are not physically present in front of the baby. This feature makes the use of a DHT11 temperature and humidity sensor to sense the temperature and humidity of the surroundings.

2. FAN MECHANISM

The user is provided with a toggle switch on the *Arduino Iot Cloud Dashboard*, and based on the temperature and humidity readings from the previous feature, the user (parents in this case) can decide whether the surrounding is too hot for the baby or not, and click the toggle button in order to turn on a fan. Since this project is intended towards developing a miniaturised/prototype version of the actual system, a servo motor has been used to demonstrate the working (rotation on turning the switch on) of the rotating fan.

3. SOUND SENSOR

Lastly, the user is provided with another set of gauge and chart displays on the *Arduino Iot Cloud Dashboard*, in order to display the sound readings from the baby's surroundings in real-time. The main purpose of this feature is to detect the cry of the baby, and display the sound readings generated from it. When no sound is detected, the minimum reading is shown on the gauge, the chart and the same messaging system described earlier. However, a parent becomes aware that the baby is crying, when the sound reading being displayed becomes considerably higher than the minimum value, and stays so for some amount of time. A sound sensor is used to detect the sound of the cries of the baby in this feature.

2. DESIGN

2.1 BLOCK DIAGRAM

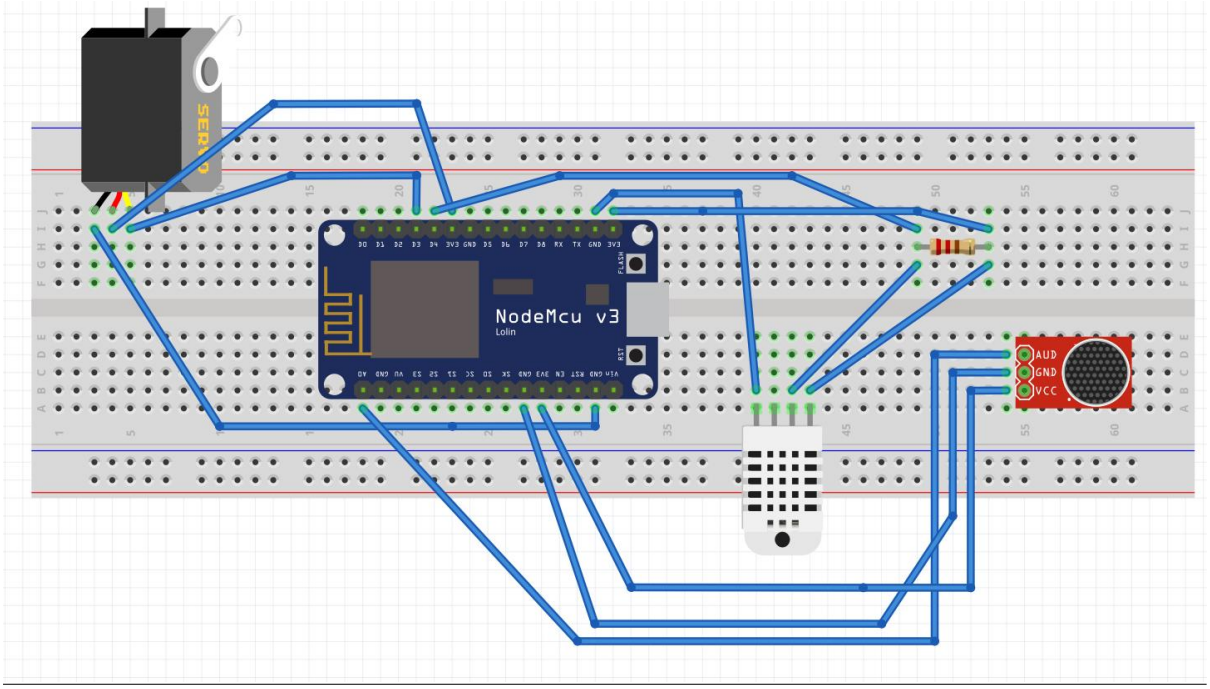

Fig.1 - The block diagram of the hardware implementation of our project.

Here, the block diagram of the hardware implementation of our project is depicted. The blue board in the middle is the NodeMCU (NodeMCU 1.0 (ESP-12E Module) board, which is the required microcontroller for our project, and it comes with an in-built ESP8266 wi-fi module for wireless connections. The white-coloured component at the bottom of the image is the DHT11 Temperature and Humidity Sensor, the red-cloured component at the right of the screen is the sound sensor, and the black coloured component at the top-left corner of the image is the servo motor. A 10 KOhm resistor is used at an appropriate position, as described in the connection details below.

2.2 HARDWARE ANALYSIS AND CONNECTIONS

All the connections are made using appropriate types (male-male and male-female) of jumper wires, with a breadboard as the connecting base.

As already visible, the DHT11 sensor is connected to the NodeMCU Board, with its output pin connected to the D4 digital pin of the NodeMCU via the 10 KOhm resistor, power pin to the 3V3 pin via the other end of the 10 KOhm resistor, and the ground pin to the GND pin of the NodeMCU. The rest of the configuration regarding the reading and displaying of the sensed temperature and humidity values is done in the software part i.e., the code for the given hardware, and the configuration done on the *Arduino Iot Cloud Dashboard*.

Then, the input pin of the servo motor is connected to the D3 digital pin of the NodeMCU board, power pin to the 3V3 pin and the ground pin to the GND pin of the NodeMCU board. The rest of the configuration regarding the turning on/off function of the fan (as described earlier) using a switch on the *Arduino Iot Cloud Dashboard* is done in the software part i.e., the code for the given hardware, and the configuration done on the *Arduino Iot Cloud Dashboard*.

Similarly, the sound sensor is connected to the Node MCU board, with its output pin connected to the A0 analog pin of the NodeMCU board, power pin connected to the 3V3 pin and the ground pin connected to the GND pin of the NodeMCU board. Here also, the rest of the configuration regarding the reading and displaying of the sensed sound level values is done in the software part i.e., the code for the given hardware, and the configuration done on the *Arduino Iot Cloud Dashboard*.

2.3 HARDWARE SNAPSHOTS

A few snapshots of the real-life hardware, made in accordance with the above-described block diagram, are given as follows:

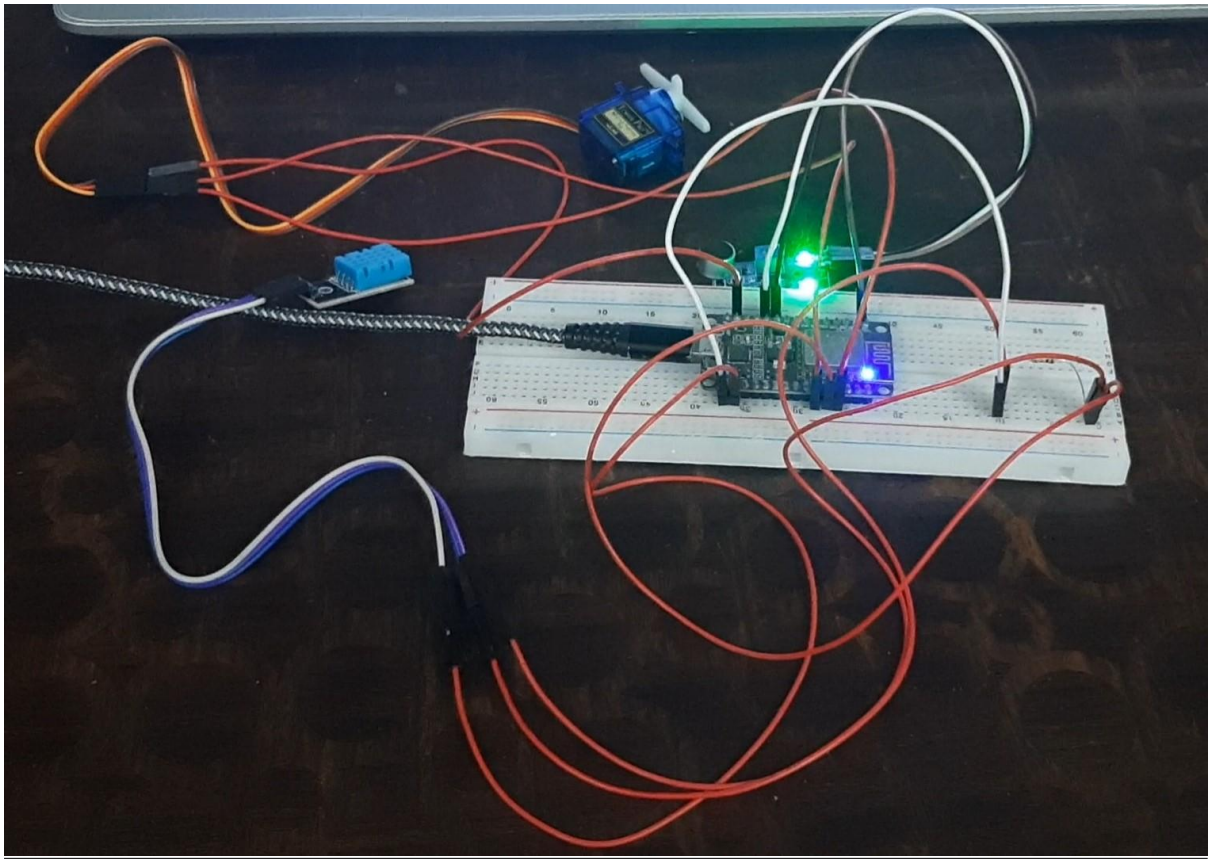

Fig.2 - Image showing the complete hardware implementation of our project, with all the connections dode in accordance with the described block diagram.

As visible in the above image, all the connections have been made with the help of appropriate jumper wires, with all the previously described hardware components (NodeMCU board, DHT11 sensor, servo motor, sound sensor) connected in accordance with the connections described for the given block diagram. This image was taken after starting the simulation, and that is why the inbuilt LEDs on the NodeMCU board and the sound sensor can be seen as turned on. Also, a slight difference from the block digram is that in the actual harware, the DHT11 sensor is of sky-blue colour, the sound sensor and the servo motor are deep blue in clour, and the NodeMCU board is grey in colour. All other aspects, especially the technical ones are the same as those described for the block diagram.

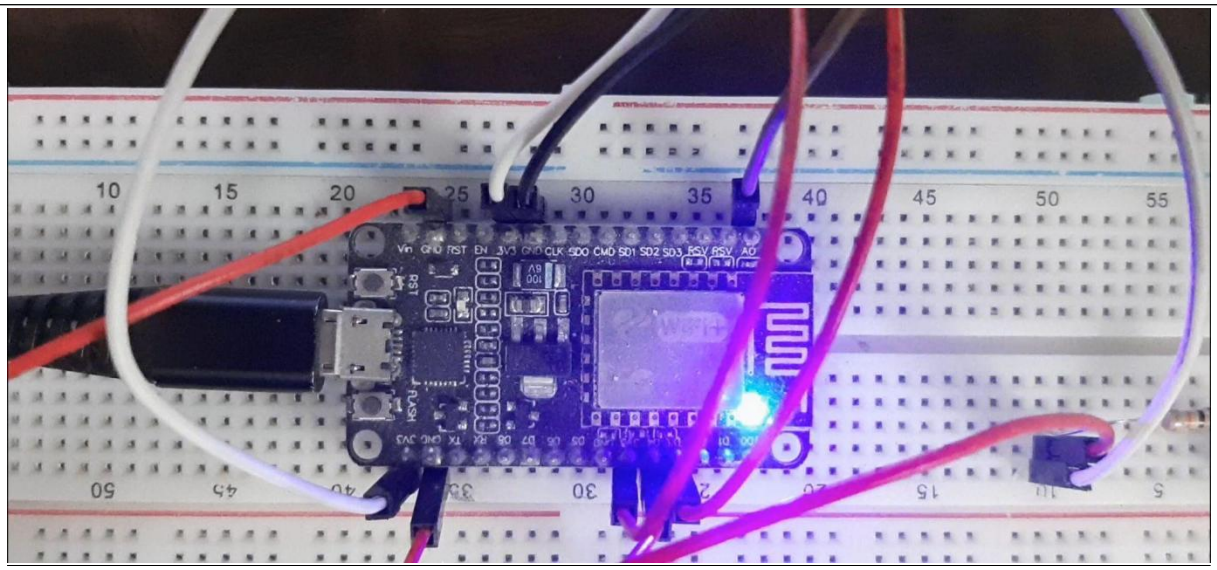

Fig.3 - A close-up view of the NodeMCU board, as seen in action.

The above image shows a close-up view of the NodeMCU board, as seen after starting the simulation (the inbuilt LED is turned on). Here, as it can be seen, all the exact pins, as described for the NodeMCU board in the block diagram, have been occupied, with the other end of every connection being made with previously mentioned pins of the appropriate hardware components.

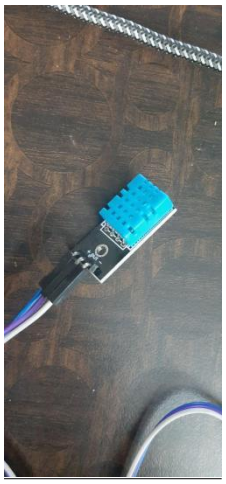

Fig.4 - A close-up view of the DHT11 sensor.

The above image shows the DHT11 sensor in action. Although there is no exclusive LED to denote its activity, running water vapours over the sensor body (using boiling water) and making it warm using the same source, actually shows a variation in the temperature and humidity readings on the respective gauge, chart and message display widgets on the *Arduino Iot Cloud Dashboard*.

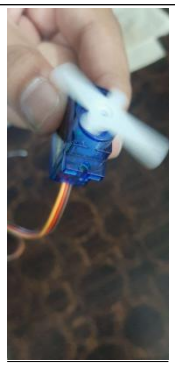

Fig.5 - A close-up view of the servo motor, as seen in action.

The above image shows the servo motor in action. The rotation of the servo motor blade (on clicking the toggle switch on the *Arduino Iot Cloud Dashboard*) can be verified with the fact that the blades are not aligned with the servo motor body, in the above image, which is only possible to be captured in an image if the motor blade is actually rotating. Since this is a miniature/prototype model of a real-life version, envisioned to be serving the real purpose of baby monitoring, a servo motor is used as a fan for this model. In the real-life model, a more efficient motor is actually thought to be used.

Fig.6 - A close-up view of the sound sensor, as seen in action.

The above image shows the sound sensor in action (the inbuilt green LEDs are turned on). Making loud noises (just like the cry of a baby), actually shows a variation in the sound level readings on the gauge, chart and message display widgets on the *Arduino Iot Cloud Dashboard*.

3. SOFTWARE - CODING AND ANALYSIS

3.1 Configuring the Arduino Iot Cloud Platform (Setup and Dashboard):

We create an account on the *Arduino Iot Cloud Platform*, and on the opening page i.e., the *Things* tab, we create a project, named *CPS_Project*, and set the main device as *NodeMCU 1.0 (ESP-12E Module)*, and name it as *Node_1*.

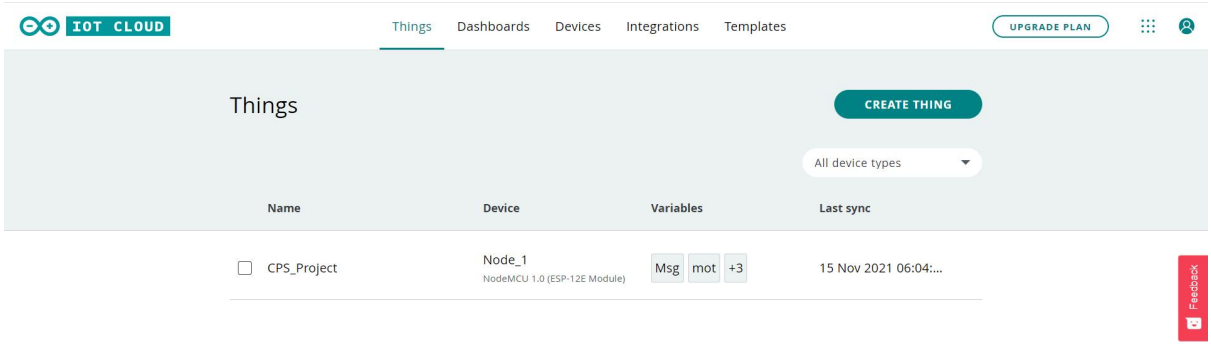

Fig.7 - The *Things* tab on the opening page of the *Arduino Iot Cloud Platform*, after doing the above-mentioned configurations.

Under the *Things tab,* we go to the *Setup* tab, and define 5 variables, namely

- **Humidity** (set as ***humidity*** for the code) of type *CloudRelativeHumidity*
- **mot** (set as ***mot*** for the code) of type *bool*
- **Msg** (set as ***msg*** for the code) of type *string*
- **sound** (set as ***sound*** for the code) of type *int*
- **Temperature** (set as ***temperature*** for the code) of type *CloudTemperatureSensor*

Since the *Arduino Iot Cloud Platform* provides us with many custom variable types (like *CloudRelativeHumidity*, *CloudTemperatureSensor*) as visible above, we make use of them for easier implementation of the respective components.

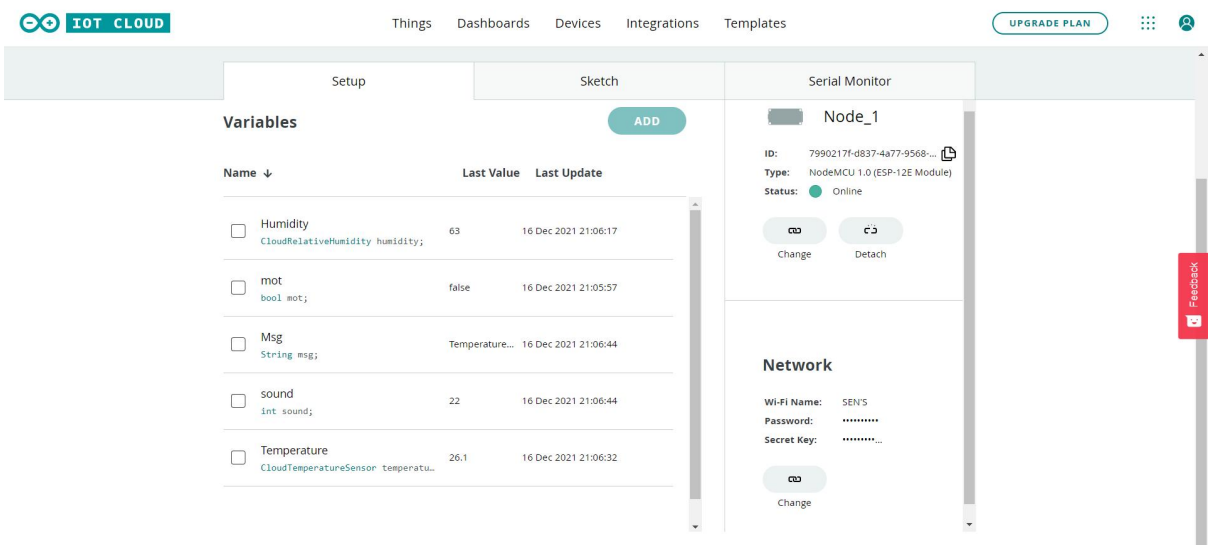

Fig.8 - The *Setup* tab under the *Things* tab, where the 5 different variables are declared and defined.

In the above image, there is a *Last Value* and a *Last Update* column, listed against the variable names, with some values displayed along them. Since the simulation has already been run previously, these values got logged at the time of the last run. Also, on the right of the screen, the *Status* of the *Node_1* device (displayed along with its *ID* and *Type*), shows that the device is *online*, which means that the device is connected to the cloud, and is ready to run the simulation.

Now, under the *Dashboard* tab, and build and configure the widgets required for the necessary outputs and interactions with the hardware. This is what was previously referred to as the *Arduino Iot Cloud Dashboard*. The following widgets are added to the dashboard, with the given configurations:

- A ***Messenger*** widget, which is named as *Message*, and linked to the *Msg* variable (previously defined on the *Setup* tab).
- For the temperature readings:
  - A ***Gauge*** widget, which is named as *Temperature*, and linked to the *Temperature* variable (previously defined on the *Setup* tab).
  - A ***Chart*** widget, which is named as *Temperature Variation*, and linked to the *Temperature* variable (previously defined on the *Setup* tab).
- For the humidity readings:
  - A ***Gauge*** widget, which is named as *Humidity*, and linked to the *Humidity* variable (previously defined on the *Setup* tab).

- A ***Chart*** widget, which is named as *Humidity Variation*, and linked to the *Humidity* variable (previously defined on the *Setup* tab).

- For the sound sensor readings:
  - A ***Gauge*** widget, which is named as *Sound Reading*, and linked to the *sound* variable (previously defined on the *Setup* tab).
  - A ***Chart*** widget, which is named as *Sound Variation*, and linked to the *sound* variable (previously defined on the *Setup* tab).
- A ***Switch*** widget for the on/off switch of the servo motor, which is designed to work as a fan, and can be turned on in the actual hardware with this switch from remote location, by just logging on to the *Arduino Iot Cloud Platform*. This widget is named Servo_Switch, and is linked to the *mot* variable (previously defined on the *Setup* tab).

***Note** -* Illustrations for the *Arduino Iot Cloud Dashboard* are given in the result and output section.

3.2 Writing the Code to Link the Hardware with the Cloud:

The entire code for configuring the hardware connections, and for linking the hardware with the *Arduino Iot Cloud Platform*, is given below, as written on the online IDE provided by the *Arduino Iot Cloud Platform*, under the *Sketch* tab:

```
#include "thingProperties.h"

#include "DHT.h"

#include <Servo.h>

#define DHTpin 2

#define DHTTYPE DHT11

DHT dht(DHTpin, DHTTYPE);

Servo myservo;

int value;

double angle;

const int soundpin=A0;
```

```
void setup() {
 Serial.begin(9600);
 delay(1500);
 myservo.attach(0);
 myservo.write(0);
 pinMode(soundpin,INPUT);

 initProperties();

 ArduinoCloud.begin(ArduinoIoTPreferredConnection);

 setDebugMessageLevel(2);
 ArduinoCloud.printDebugInfo();
}



void loop() {
 ArduinoCloud.update();
 dht_sound_sensor();
 onMotChange();
}



void dht_sound_sensor()
{
 float hm = dht.readHumidity();
 Serial.print("Humidity = ");
```

```
Serial.println(hm);

float temp = dht.readTemperature();

Serial.print("Temperature = ");

Serial.println(temp);

int soundsens=analogRead(soundpin);

Serial.print("Sound = ");

Serial.println(soundsens);

humidity = hm;

temperature = temp;

sound = soundsens;

msg = "Temperature = " + String (temperature) + "  Humidity = " + String(humidity) + " Sound = " + String(sound);

}



void onMotChange() {

 while(mot == 1)

{

  ArduinoCloud.update();

  myservo.write(360);

  delay(400);

  myservo.write(0);

  delay(400);

 }

}
```

Here, a detailed explanation of the code is given as follows:

- The necessary libraries are imported.
  - *thingProperties.h* for defining the properties of things on the cloud.
  - *DHT.h* for the properties and functions of the DHT11 sensor.
  - *<Servo.h>* for the properties and functions of the servo motor.
- The *DHTpin* (i.e. the pin for the DHT11 sensor) is set as 2, which is the D4 pin on the NodeMCU board (as already configured in the hardware). Also, the *DHTType* is defined as *DHT11*.
- A *DHT* object is defined as *dht* (for working with the DHT11 sensor), and a *Servo* object is defined as *myservo.*
- Three other variables, namely *value* (type = int), *angle* (type = double), and *soundpin* (type = const int, and initialised to A0) are declared.
- In the *setup* function, the initial port is set as *9600*, with a delay of *1500* milliseconds. The *attach* and *write* methods of the servo motor are set to 0 initially, and the *soundpin* variable (initially set to A0), is passed as the input pin for the sound sensor.
- Then, inside the *setup* function, the cloud properties are initialised, and the connection with the cloud is established. Also the *setDebugMessageLevel()* function is defined with the argument as 2, along with the *ArduinoCloud.printDebugInfo()* function. These two functions allow us to obtain more information related to the state of network and IoT Cloud connection and errors.
- Inside the *loop* function, whose main task is to run everything inside it repeatedly, three functions are called successively, namely the *ArduinoCloud.update()* function to update the cloud input/outputs at regular intervals, the *dht_sound_sensor()* function to perform the tasks related to reading the inputs and displaying the outputs of the *DHT11 sensor* and the *sound sensor*, on the *Messenger* widget, and their respective *gauge* and *chart* widgets.
- Inside the *dht_sound_sensor()* function, three variables, namely *hm* to read the humidity value, the *temp* to read the temperature value, and *soundsens* to read the sound value, are defined, and associated with the respective functions, to read their desired values. Then the values stored into each of these three variables are displayed on the *Serial Monitor*, along with an appropriate caption for each of them. Lastly, as defined earlier, the *temperature, humidity,* and the the *sound* variables from the *Setup* tab, which are already linked to the appropriate widget

on the *Arduino Iot Cloud Dashboard*, are assigned the values of the variables *temp*, *hm*, and *soundsens*, converted to string, and passed as input to the *msg* variable (for the *Messenger* widget), with an appropriate caption for each of the three values. This message is displayed on the *Messenger* widget. Also, since the *temperature*, *humidity*, and the *sound* variables are linked to the respective *gauge* and *chart* widgets, their values are reflected on these widgets, after each update of these variables for each individual iteration of the *dht_sound_sensor()* function.

- The *onMotChange()* function runs a while loop if it finds that the bool variable, named *mot*, is set to 1 i.e., the switch for the servo motor is turned on. If the switch is found to be in the *on* state, then the servo keeps on rotating (just as we would expect a fan to do), unless it is again turned off. To check for changes in the state of the switch, the *ArduinoCloud.update()* function is called after each iteration inside the *onMotChange()* function.

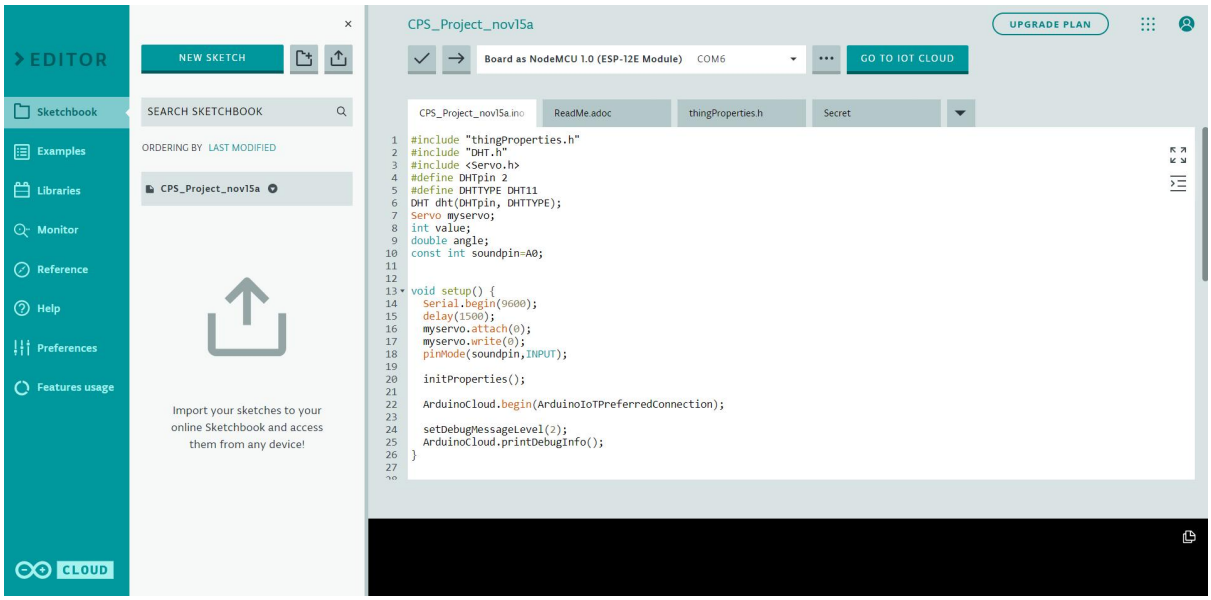

Fig.9 - An illustration of the *Sketch* tab i.e., the *Arduino IoT Cloud Platform* IDE, with the code written in it.

Now that the entire code is written, deployed, and pushed to the NodeMCU board, the entire simulation of the designed hardware-cloud model is ready to be run.

3.3 Results and Working of the Final Model:

Now the model is run, and the following illustrations depict the different outputs obtained from the model, on the *Arduino IoT Cloud Dashboard*:

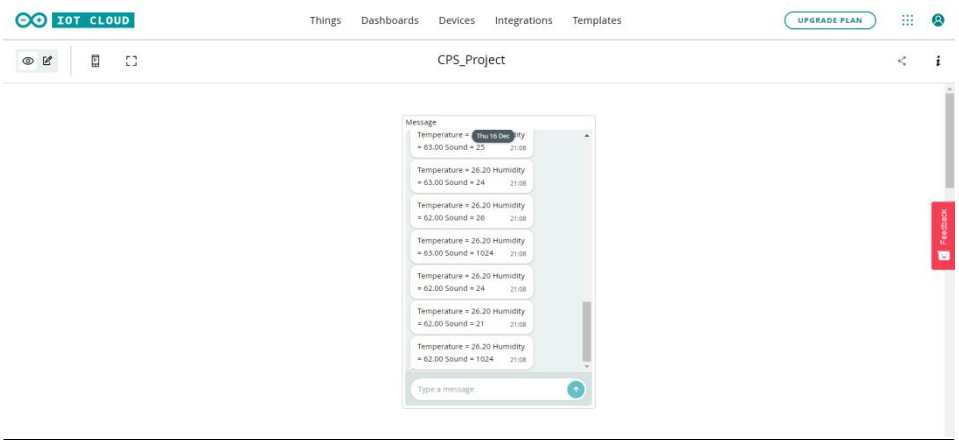

Fig.10 - The *Messenger* widget displaying the *temperature*, *humidity*, and the *sound* values on a real-time basis.

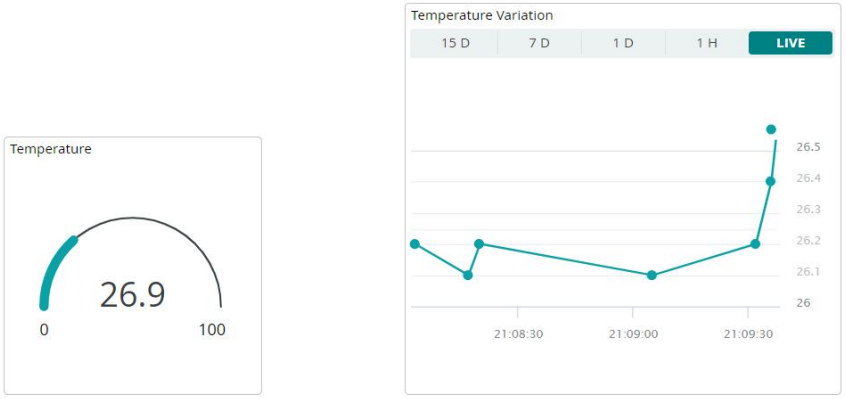

Fig.11 - The *Gauge* and the *Chart* widget for the *Temperature* variable displaying the *temperature* values on a real-time basis.

In the above image, the momentary temperature value is displayed as a reading on the *gauge* widget, and as a continuation variation on the *chart* widget.

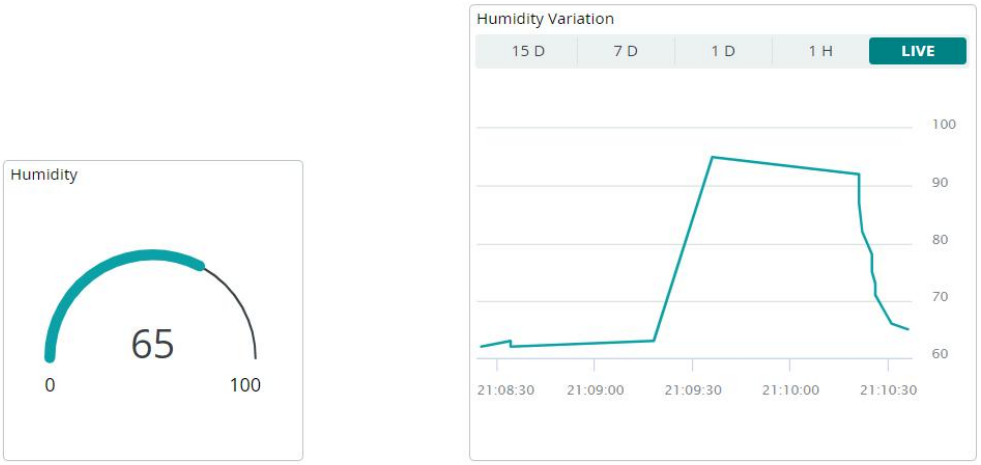

Fig.12 - The *Gauge* and the *Chart* widget for the *Humidity* variable displaying the *humidity* values on a real-time basis.

In the above image, the momentary humidity value is displayed as a reading on the *gauge* widget, and as a continuation variation on the *chart* widget.

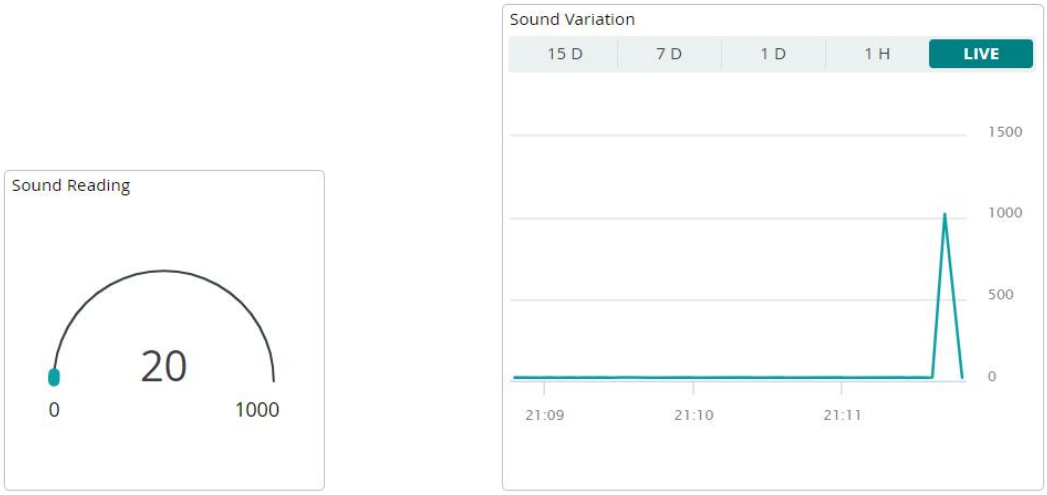

Fig.13 - The *Gauge* and the *Chart* widget for the *sound* variable displaying the *sound* values on a real-time basis.

In the above image, the momentary sound level value is displayed as a reading on the *gauge* widget, and as a continuation variation on the *chart* widget. At most times, the sound readings are low as no sound is detected at those times. high sound reading is displayed when the sensor detects loud noise.

Fig.14 - The *Switch* widget for the *mot* variable enabling the on/off feature of the *servo motor* on a real-time basis.

As soon as the switch is turned on, the servo motor, depicting a fan, starts rotating. It stops rotating once the switch is turned off.

4

4. PHOTOGRAPH OF THE PROJECT

PHOTOGRAPH OF THE PROJECT

An image of the hardware part of the project is displayed below. Since the software component has many aspects, they have already been displayed and described previously, in appropriate sections.

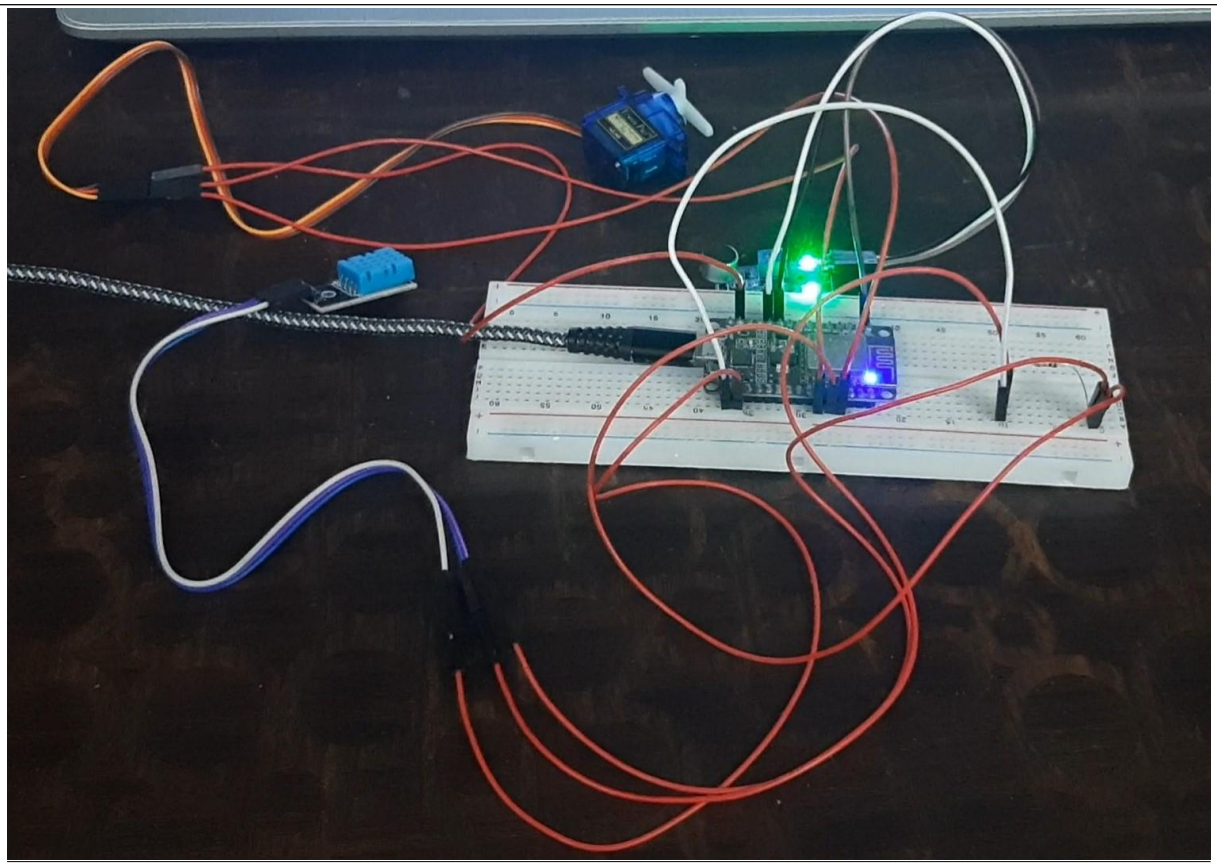

Fig.15 - The photograph of the project.