VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Semester 242 - CO3037

Internet of Things Application Development

An intelligent system

Solution for multiple applications

Lecturer: Mr. Le Trong Nhan

Students: Vu Nam Binh - 2152441
Ta Ngoc Nam - 2152788
Nguyen Quang Thien - 2152994

HO CHI MINH CITY, May 2025

Contents

1 Contribution

Contributor	Student ID	Contribution Percentage
Vu Nam Binh	2152441	100%
Ta Ngoc Nam	2152788	100%
Nguyen Quang Thien	2152994	100%

Table 1: Team contributions for the project.

2 Introduction

This is the project of the course Internet of Things Application Development from the Faculty of Computer Science and Engineering, in which we aim to design and implement a smart system using Yolo Uno board together with CoreIOT platform. Throughout this project, we have a chance to work with many types of sensors and devices such as DHT20, I2C LCD, Mini Fan, etc. Together with that, AI is also utilized to help predict data and send notifications to users to prevent possible risks to the system. A web application is also developed to display the data for users. We fetch the data from CoreIOT platform and display.

Figure 1: CoreIOT platform

3 Hardware devices

In this section, we will introduce all the sensors and devices that we used in this assignment to build up this smart system. Because almost all of these devices are bought from OhStem Education Company, the information about them are almost referenced directly from the documentation the company provided.

3.1 Temperature & Humidity Sensor

Short Description

The temperature & Humidity sensor (DHT20) utilizes I2C protocol for conservation with Yolo Uno board. This sensor has a high precision, low price and suitable for applications that need to measure the temperature and humidity of the environment.

Figure 2: Temperature & Humidity Sensor

Technical Specifications

- Input Voltage: 3.3V
- Range of humidity measurement: $0 \sim 100\%$ RH
- Range of temperature measurement: $-40 \sim +80°C$
- Humidity measurement precision: $\pm 3\%$ RH
- Temperature measurement precision: $\pm 0.5°C$
- Output Signal: I2C

3.2 Light Sensor

Short Description

Light sensor is a device which help recognize and determine the light intensity of the surrounding environment. This sensor is suitable for basic applications about light determination, day/night detection and other interesting ones.

Figure 3: Light Sensor

Technical Specifications

- Operation voltage: 3.3 V
- Current intensity: 0.5-3 mA
- Photoresistor: GL5528
- Resistance when there is light: 20 KΩ
- Resistance when there is no light: 1 MΩ
- Response time: 20-30 secs
- Maximum wave length: 540 nm

3.3 Passive Infrared Sensor (PIR Sensor)

Short Description

PIR sensor is used to detect the movement of objects that emit infrared radiation (such as the movement of people, animals, heat-generating objects, etc). This sensor is also known as PIR motion body temperature sensor.

We can adjust the sensitivity of the sensor to limit the distance to recognize movement to be farther or nearer, as well as the radiation intensity of the desired object.

Figure 4: PIR Sensor

Technical Specifications

- Operation voltage: 3.3 V
- Detection range: 360 degree of cone angle and maximum distance is 6m
- Operating temperature range: $32 - 122°F$ $(0 - 50°C)$
- Operating voltage: DC 3.8 V - 5 V
- Current intensity consumption: $\leq$ 50 uA
- IC: AS312
- Signal type: Digital
- Size: 24 mm x 48 mm x 16 mm

3.4 Tiny LED RGB

Short Description

Module 4 LED RGB contains of 4 LEDs RGB ws2812. Using integrated chip, user can control every single LED separately or all together. Additionally, user can adjust the brightness and generate any available colors in the RGB table. This module has a low price and it is suitable for applications that need animation, decoration.

Figure 5: Tiny LED RGB

Technical Specifications

- Operation voltage: 3.3 V
- Number of LED: 4 x RGB LED
- Maximum current intensity: 60 mA (1 LED), 240 mA (4 LEDs)
- Type of LED: WS2812
- Birghtness: $0 \sim 255$
- Control: Using 1 control pin
- Angle: 140°
- Size: 48 mm x 24 mm x 18 mm

3.5 Mini Fan

Short Description

The motor is a very common electronic device in our daily lives (such as fans, car engines, water pumps, ets). When the motor is supplied with electricity, it will rotate the motor shaft, thereby creating many different applications.

Figure 6: Mini Fan

Technical Specifications

- Operation voltage: 3.3 V
- Control signal: 2 pins
- Size: 2 4mm x 48 mm x 16 mm

3.6 LCD Screen 1602

Short Description

LCD screen 1602 comes with an I2C module using the HD44780 driver. This module is capable of displaying 2 lines with each line containing of maximum 16 characters. The LCD 1602 is durable and very popular. The screen is integrated with the I2C module, which help connection be simpler and much faster.

Technical Specifications

- Operation voltage: 3.3 V
- I2C address: 0x27
- Color: green
- Screw hole size: 3x M3
- Size: 80 mm x 42 mm x 19 mm
- Weight: 38 g

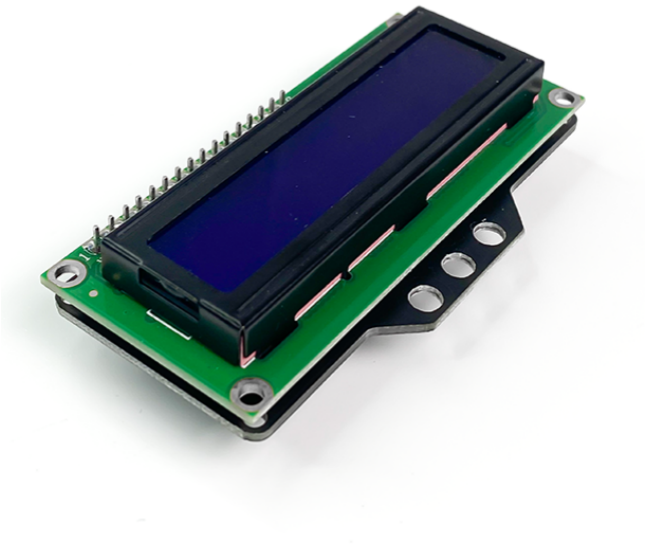

Figure 7: Enter Caption

3.7 4-digit 7-segment LED

Short Description

This module is designed to help user easily control and display information on 4 7-segment LEDs with only 3 communication pins via the 74HC595 shift register. In addition, the circuit also has the ability to expand the next LEDs via the serial output port. The circuit has an easy-to-use accompanying library suitable for multiple applications such as counters, clocks, etc.

Figure 8: 4-digit 7-segment LED

Technical Specifications

- Model: 74HC595 4-Digit Display 0.36 Inch Module
- Voltage: $3.3 \sim 5$ VDC
- Current intensity: $30 \sim 80$ mA
- IC driver: 74HC595
- 3 communicating pins: SCLK, RCLK, DIO
- Size: 42 mm x 24 mm x 12 mm

4 System Design & Development

4.1 Hardware Architecture & Connectivity

In this part, we will present the hardware architecture and show how we connect all sensors and devices

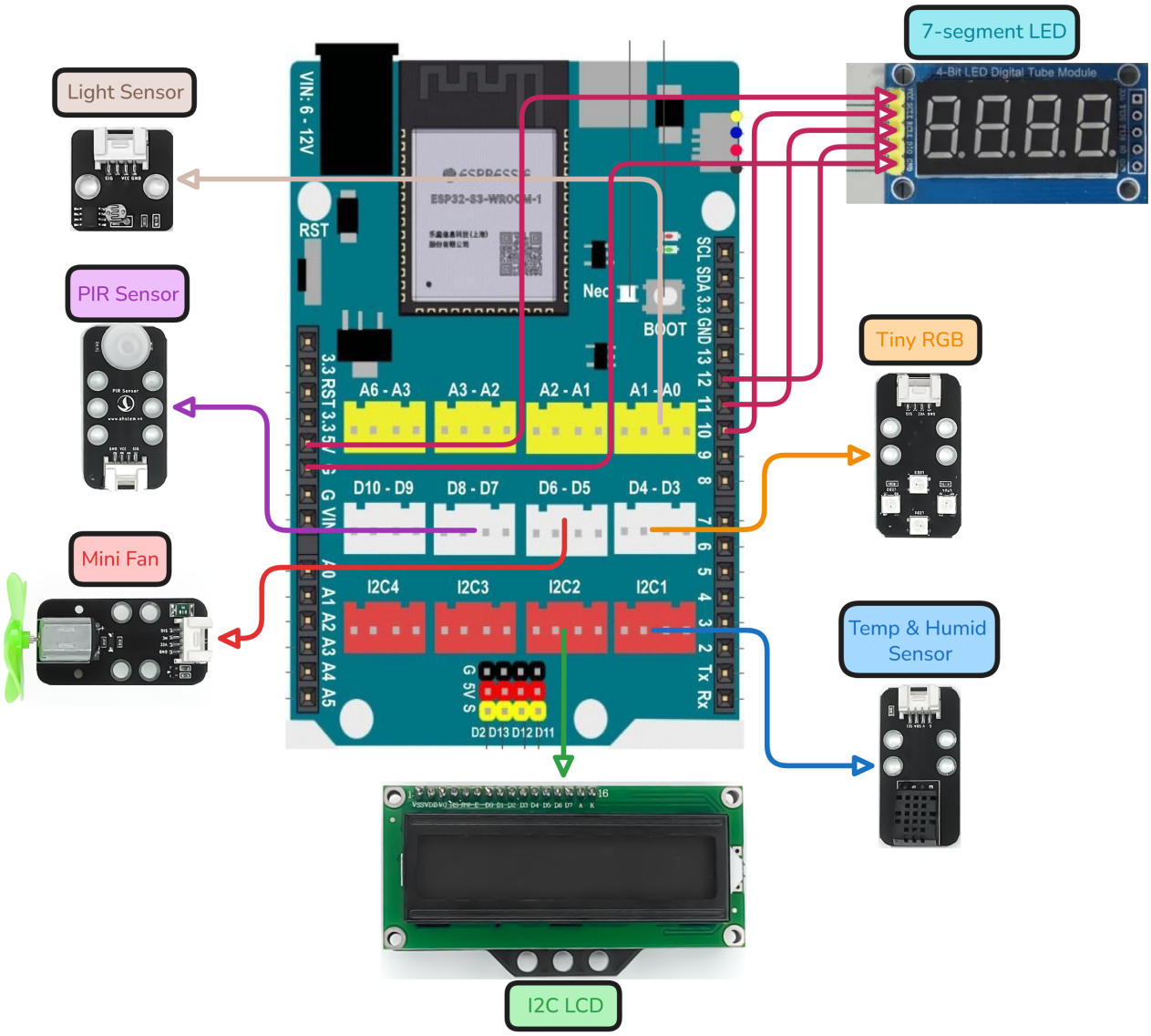

Figure 9: Hardware Architecture

We utilize the Yolo Uno board as the controlling center, which is integrated with various sensors and devices to develop a smart system that can help monitor the temperature, humidity and light. The data is then sent to CoreIOT platform for data visualization. Based on the collected data from three sensors, the Mini Fan and LED RGB will be controlled to operate. This comprehensive system serves as a practical IoT prototype for applications such as smart homes, environmental monitoring, or energy-efficient automation. Below is the general connectivity for the sensors or devices:

- **Temperature & Humidity Sensor:** This sensor utilizes I2C protocol for communication so it is connected to I2C1.
- **I2C LCD 1602:** This device utilizes I2C protocol for communication so it is connected to I2C2.

- **Tiny LED RGB:** This device is connected to D3 pin.
- **Mini Fan:** This device is connected to D5 pin.
- **PIR Sensor:** This sensor is connected to D7 pin.
- **Light Sensor:** This sensor is connected to A0 pin.
- **4-digit 7-segment LED:** This device is connected manually to the VCC, GND. Besides, its SCLK, RCLK, and DIO pins are connected to D10, D11, and D12 pins of the Yolo Uno board, respectively.

4.2 Software Architecture

In this assignment, we utilize the RTOS architecture to develope the system. We created multiple tasks with different using purposes and let them be exectued through RTOS. Below is the table showing all tasks and their purposes.

4.2.1 Task Design (RTOS Architecture)

Task Name	Period (ms)	Purpose
WiFi connection	30000	Check WiFi connection
CoreIOT connection	1000	Check CoreIOT platform connection
Send telemetry	5000	Send telemetry data to CoreIOT
Send attribute	2000	Send attribute data to CoreIOT
Thingsboard loop	10	Thingsboard loop
Light control	500	Control LED RGB
Fan control	500	Control Mini Fan
Motion detection	500	Detecting motion through PIR sensor
LCD update	1000	Update printed data on LCD 1602
Update 7-segment LED	1000	Update display time on 7-segment LED
Display 7-segment LED	0	Display system operation time on 7-segment LED

Table 2: Task design

All of these tasks are executed simultaneously using RTOS, so to make everything simple and easy to catch the idea, we will show the flowchart for every task separately below.

4.2.2 Task Wifi connection

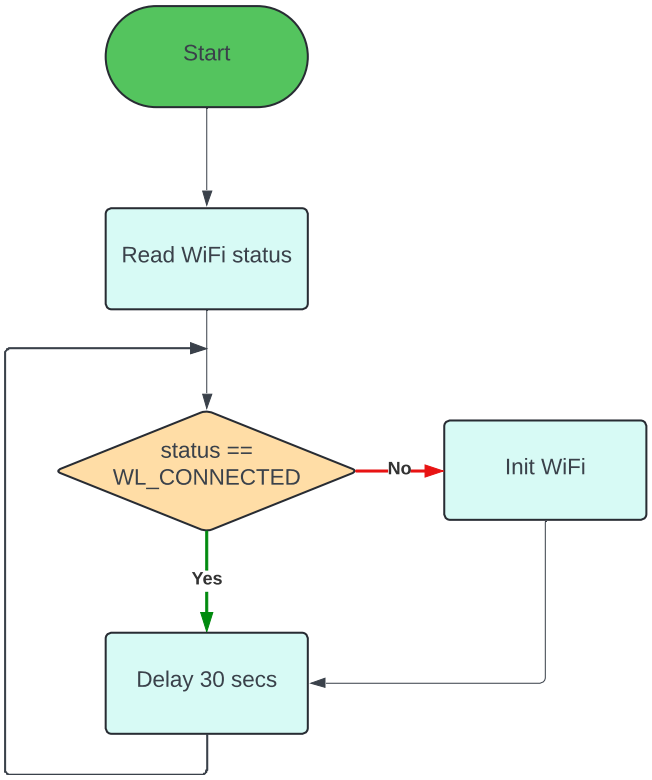

Figure 10: WiFi connection check task

This task is repeated every 30 seconds to check the status of the WiFi connection. In the case of losing connection, the Yolo Uno board will try to reconnect. This task is quite simple so we will not discuss more here.

4.2.3 Task CoreIOT connection

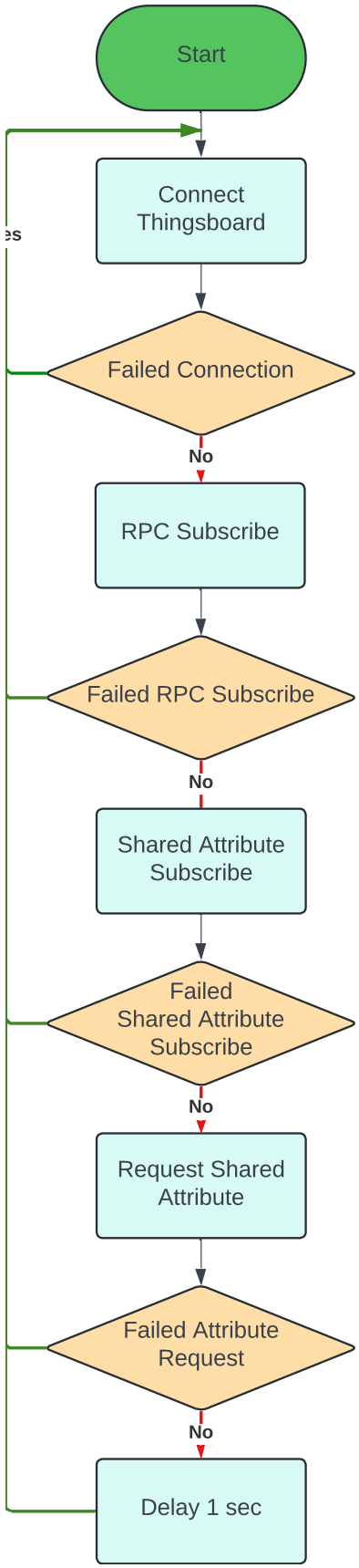

Figure 11: CoreIOT connection check task

This task is repeated every 1 second to check the connection to the CoreIOT platform. We need to provide it the TOKEN of the device we are using on CoreIOT to connect to this platform. If the connection can not be established, the error will be shown on the serial terminal. Moreover, this task also checks for the subscription for RPC, shared attributes and attributes request.

4.2.4 Task send telemetry data

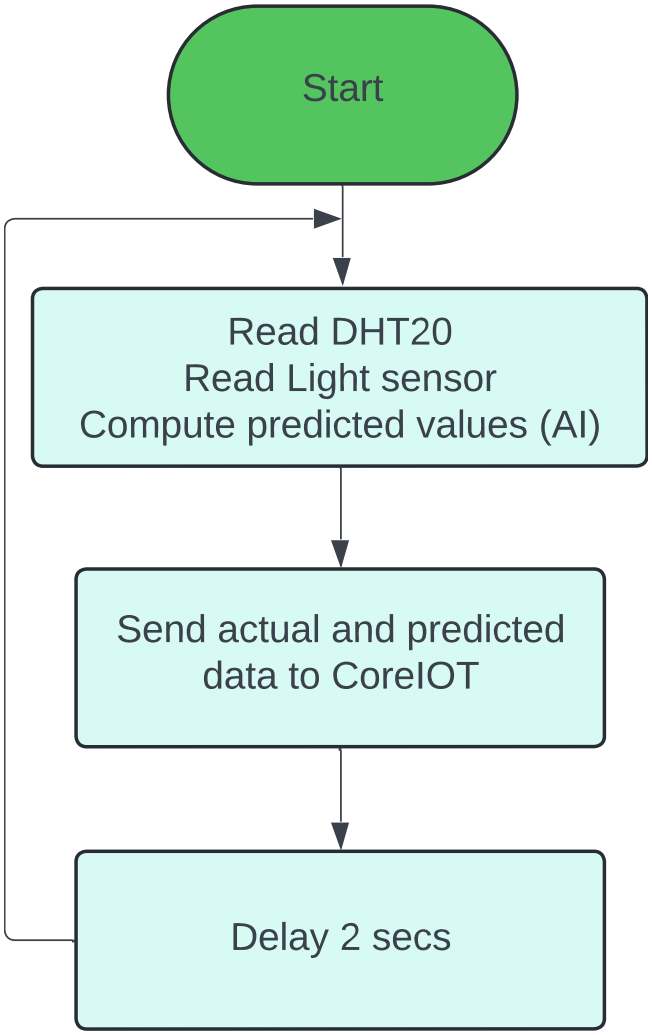

Figure 12: Send telemetry data task

Every 2 seconds, the temperature, humidity and light intensity values will be collected from the sensor. To get the temperature and humidity values, we use the functions **getTemperature()** and **getHumidity()** from library **dht20.h**. Meanwhile, to get the value of light intensity, we have to process ourselves because we do not have the library for it. The output signal of the Light sensor is a voltage level. This helps reduce the complexity in reading the sensor. Instead of using available library, we can read the data by simply using the very basic function **analogRead()** when working with Arduino. The data we can read varying in the range of 0 to 4095 (12 bits binary). After collecting the data, we process further to convert the result into percentage as shown in the code script below.

```
1     int light_tmp;
2     light_tmp = analogRead(LIGHT_SENSOR_PIN);
3     light = (light_tmp * 100) / 4096.0;
```

Listing 1: Light Sensor Read

Additionally, we also apply AI model here to predict the future values of the temperature and humidity. However, we will discuss about this separately in the next secion **Artificial Intelligence**.

4.2.5 Task light control

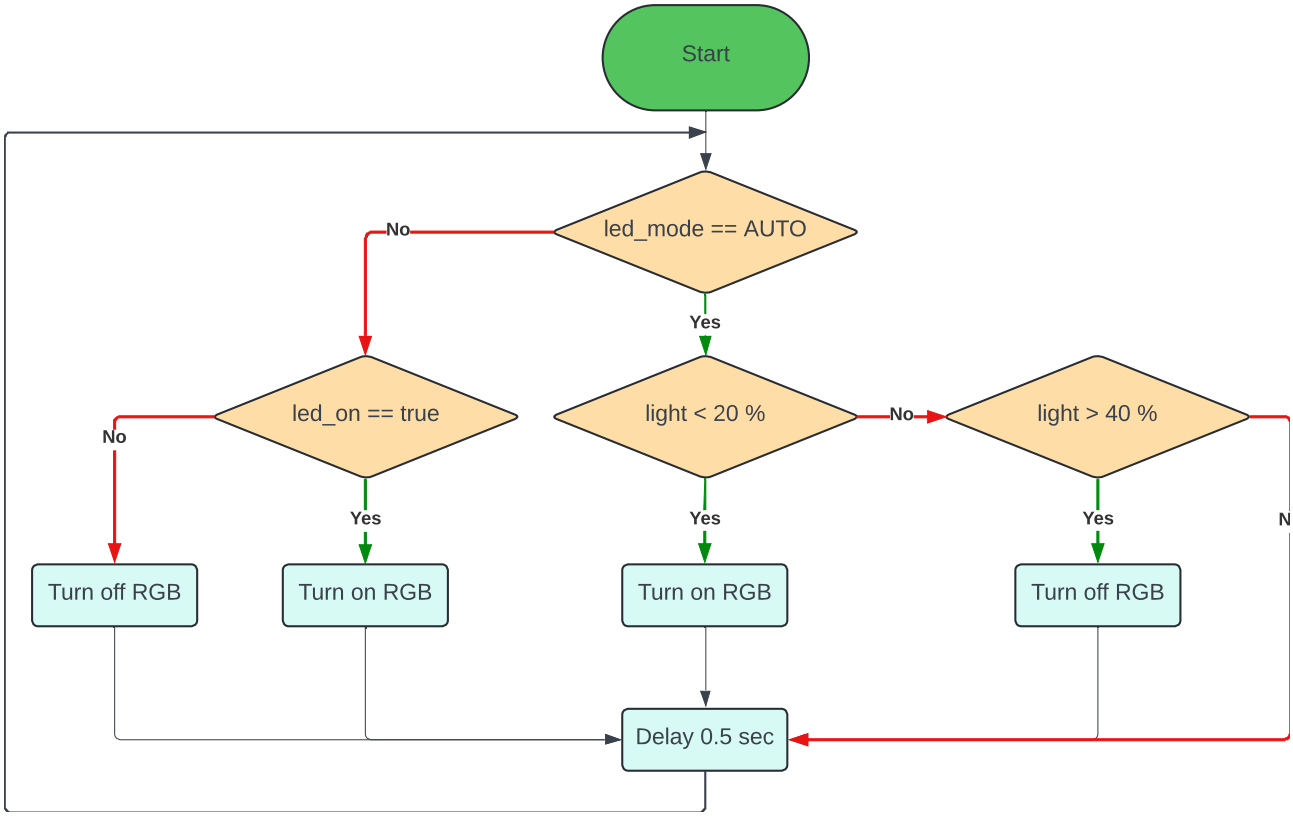

Figure 13: Light control task

This task is to control the Tiny RGB LED to turn on or off. Basically, we provide two operation modes for the RGB LED, which are **MANUAL** and **AUTOMATIC**.

- **AUTOMATIC:** This is the default mode for the control of RGB LED. The LED will be turned on and off when the light intensity value collected from the Light Sensor is less than 20% and greater than 40%, respectively. There is a range between 20% and 40% in which the status of the RGB LED is kept. This is to prevent the case of turning on and off the LED continuously (LED blinky) because this animation can easily make the user feel annoyed.
- **MANUAL:** This mode allows users to turn the LED on or off themselves. To use this mode, users need to switch on the switch **LED Mode** on the dashboard. After that, the user can decide to turn on or off the LED using the switch **LED Value** as shown in the figure 16. We provide this operation mode for the user to be easy in controlling the LED. In some situations, users need to turn off the LED totally, such as saving energy at late night or in the case that the light sensor is broken and the LED can not be controlled precisely in the **AUTOMATIC** mode, then they can control it themselves. We do not have the physical buttons or switches to control the operation of the RGB LED, so to switch the operation mode, together with choosing the status of the LED, users must control it through the virtual switches on the dashboard.

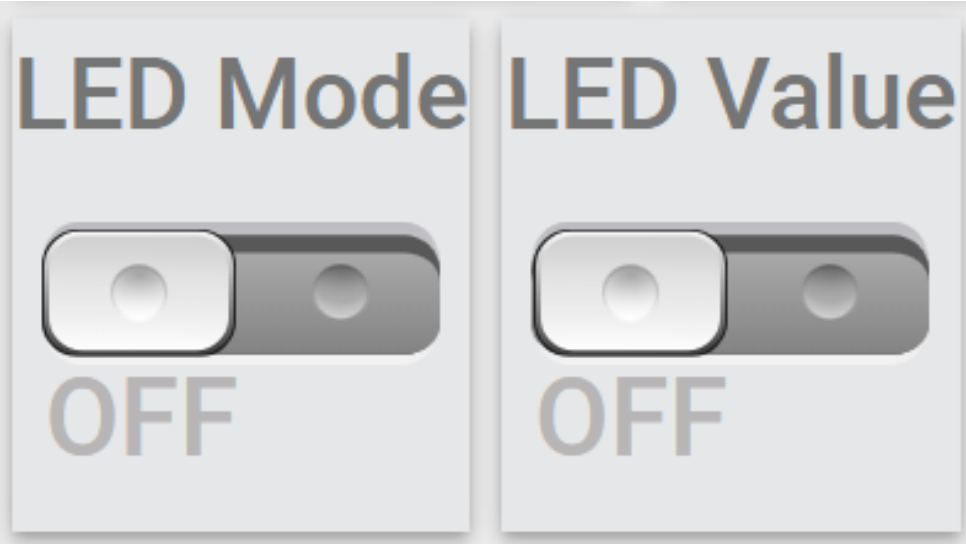

Figure 14: Switches LED Mode and LED Value on dashboard

Moreover, to control the Tiny RGB LED, we utilize the library **"Afafruit_NeoPixel.h"**. In fact, the RGB LED can display any available colors in the RGB table. However, we only provide the white color for the LED. It means that, whenever the LED is turned on, it will be in white color. The code script below is shown as an example code to control the operation of the LED.

```
#include "Adafruit_NeoPixel.h"
Adafruit_NeoPixel pixels(4, LIGHT_PIN, NEO_GRB + NEO_KHZ800);
void setup(){
    pixels.begin();
}
void task_light_control(){
    // Note: This is only the example to turn on the Tiny RGB LED with WHITE color, not the actual code to implement our system!
    for(int i=0; i<4; i++){
        pixels.setPixelColor(i, pixels.Color(255,255,255));
    }
    pixels.show();
}
```

Listing 2: Tiny RGB LED control example

4.2.6 Task fan control

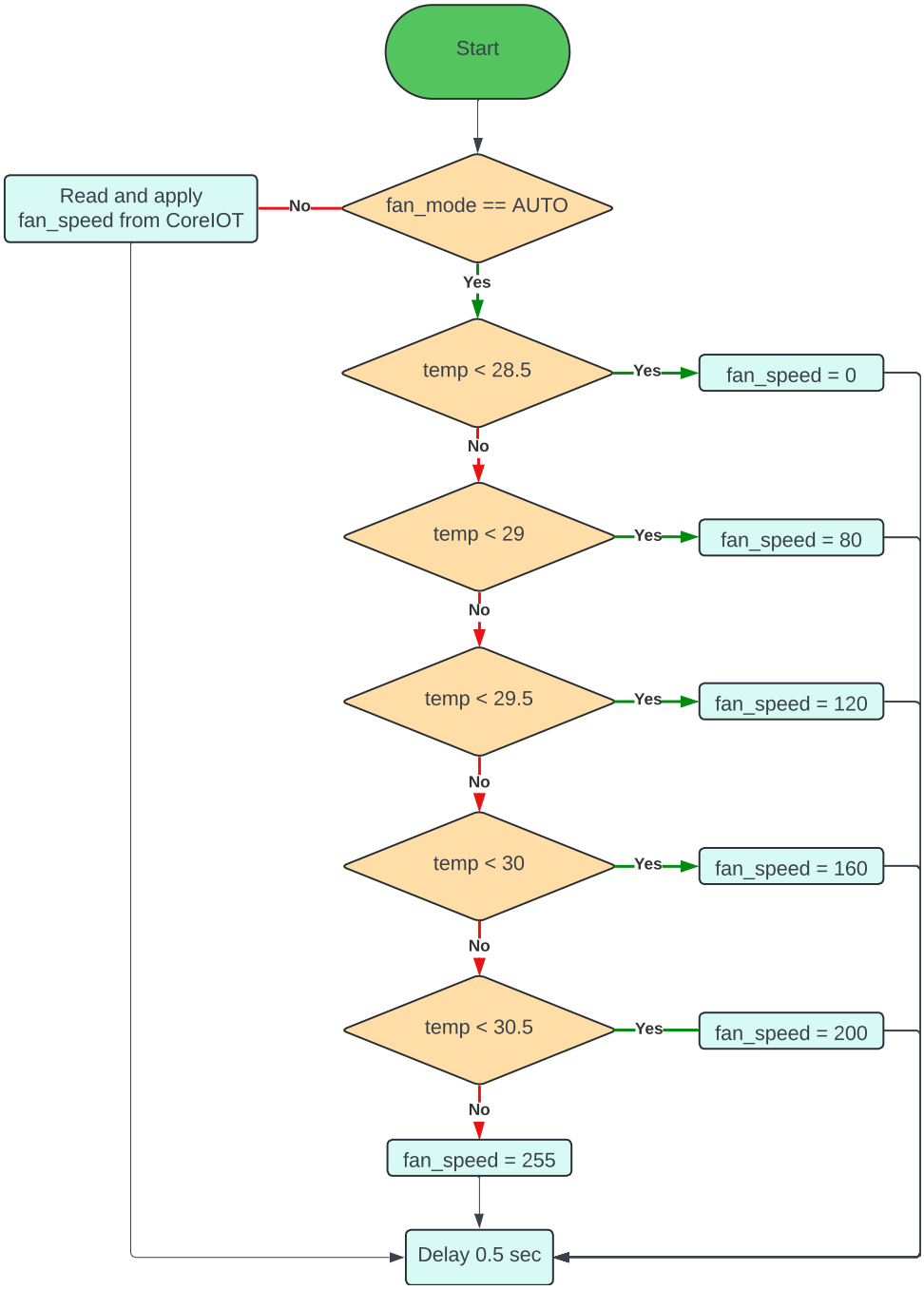

Figure 15: Fan control task

Similarly to task light control, we also provide two operation modes for the Mini Fan: **MANUAL** and **AUTOMATIC**.

- **AUTOMATIC:** This is the default mode for the control of Mini Fan. There will be 6 levels of fan speed corresponding to 6 temperature ranges of the temperature value collected from the sensor DHT20 as shown in the flowchart 15.
- **MANUAL:** This mode allows users to control the Mini Fan manually (setting the fan speed). To use this mode, users need to switch on the switch **Fan Mode** on the dashboard. After that, the user can decide to choose the fan speed using the knob **Fan Value** as shown in the figure 16.

We provide this operation mode for the user to be easy in controlling the fan. In some situations, users need to turn off the fan totally, such as saving energy at late night or in the case that the DHT20 sensor is broken and the fan can not be controlled precisely in the **AUTOMATIC** mode, then they can control it themselves. We do not have the physical buttons, switches or knobs to control the operation of the Mini Fan, so to switch the operation mode, together with choosing the speed for the fan, users must control it through the virtual switches and knobs on the dashboard.

Figure 16: Switch Fan Mode and knob Fan Value on dashboard

The Mini Fan can be controlled by using the function **analogRead(FAN_PIN, fan_speed)**. This helps simplify the code instead of using libraries and function call.

4.2.7 Task motion detect

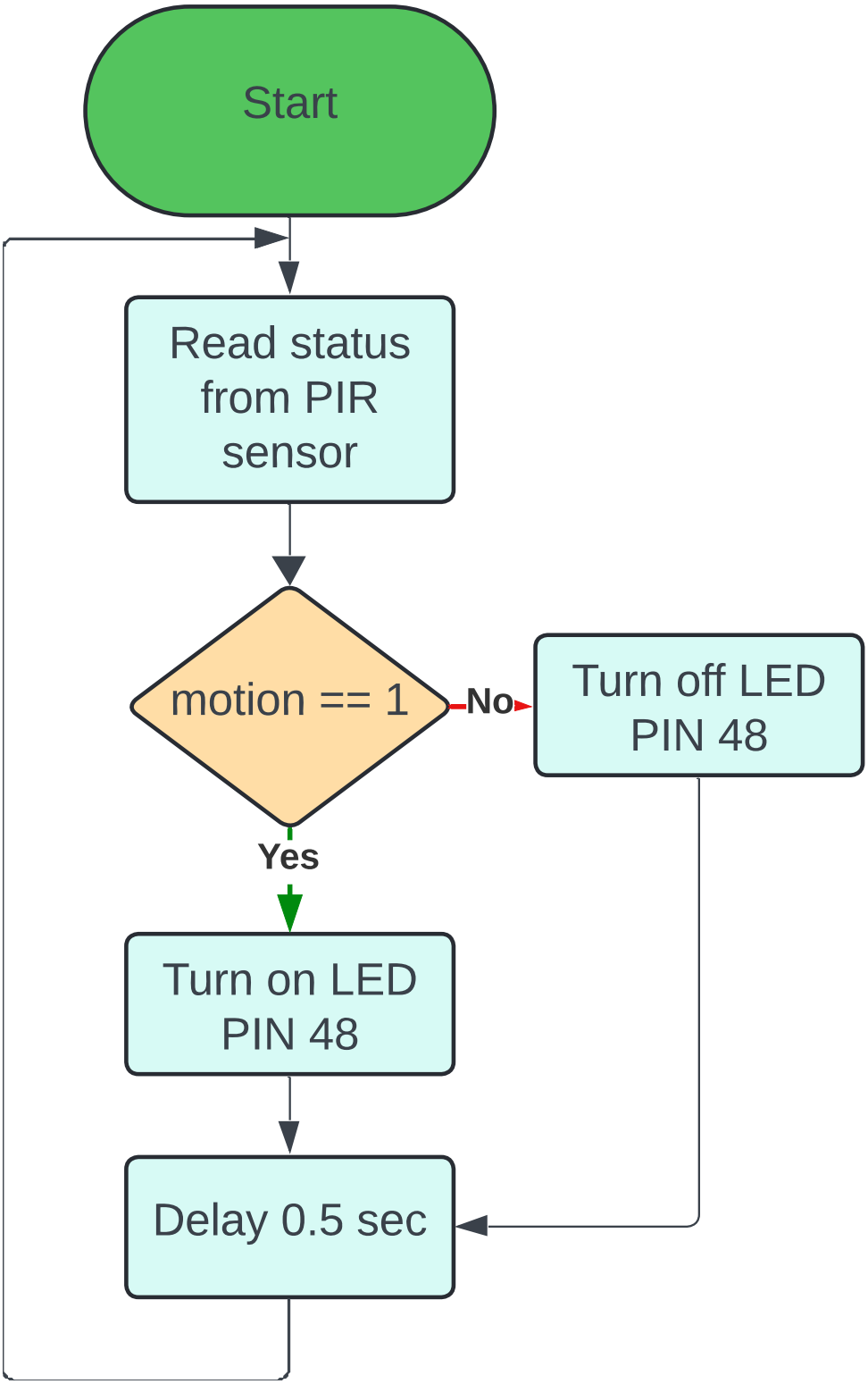

Figure 17: Motion detect task

The PIR sensor is used in this task and we simple need to use the **analogRead()** function to read the status of the sensor. If the returned value is 0, then there is no motion detected. Otherwise, the motion is detected and the LED GPIO_NUM_48 will be turned on to indicate the detected motion.

4.2.8 Task LCD

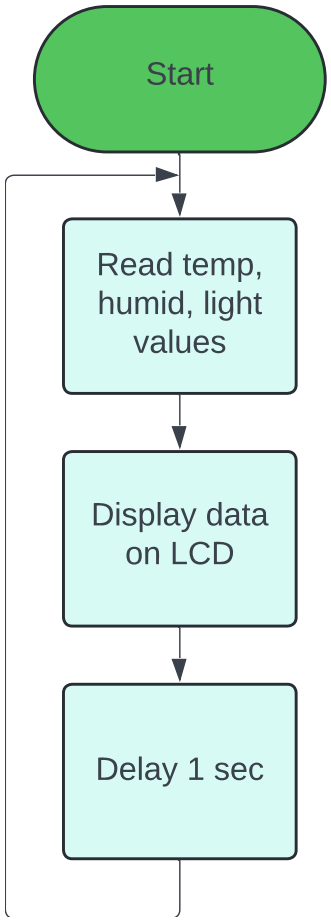

Figure 18: LCD control task

The LCD screen will display two lines. The first line will display the current temperature and humidity values collected from the DHT20 sensor in integer while the second one will display the text "HCMUT-IOT-CE" as shown in figure 19. The values are updated every 1 second.

Figure 19: LCD 1602 Screen

To control the lcd, we utilize the library **"LiquidCrystal_I2C.h"**. This LCD also uses the I2C protocol communication, so before using it, we need to find out its I2C address using the code shown below. Although the I2C address given on the OhStem Website is **0x27** but we found out that the LCD operates on the address **0x21**.

```
// CODE TO SEARCH FOR I2C ADDRESS, REFERENCE FROM: https://drive.google.com/file/d/1K-N7fz8qNcXyCX1r8TkXtuNTOyRsfO-n/view?usp=sharing
byte error, address;
int Devices;
Serial.println("Scanning...");
Devices = 0;
for(address = 1; address < 127; address++ )
{
Wire.beginTransmission(address);
error = Wire.endTransmission();
if (error == 0)
{
  Serial.print("I2C device found at address 0x");
  if (address<16)
  Serial.print("0");
  Serial.print(address,HEX);
  Serial.println("  !");
  Devices++;
}
else if (error==4)
{
  Serial.print("Unknown error at address 0x");
  if (address<16)
  Serial.print("0");
  Serial.println(address,HEX);
}
}
if (Devices == 0)
Serial.println("No I2C devices found\n");
else
Serial.println("done\n");
delay(5000);
```

Listing 3: Code to scan Address

The code below is an example of using the I2C LCD 1602 screen.

```
// Note: This is only the example to control the text to be displayed on the LCD 1602 screen, not the actual code to implement our system!
LiquidCrystal_I2C lcd(0x21,16,2);

void setup(){
    lcd.init();
    lcd.backlight();
}

// Display "Hello world!" on the first line and "HCMUT-IOT-CE" on the second line of the LCD 1602 screen
void task_lcd(void *pvParameters){
    lcd.clear();            // clear screen

    lcd.setCursor(0,0);      // set cursor to the first cell of the first row
    lcd.print("Hello world!");

    lcd.setCursor(0,1);     // set cursor to the first cell of the second row
    lcd.print("HCMUT-IOT-CE");
}
```

4.2.9 Task update seven segment LED

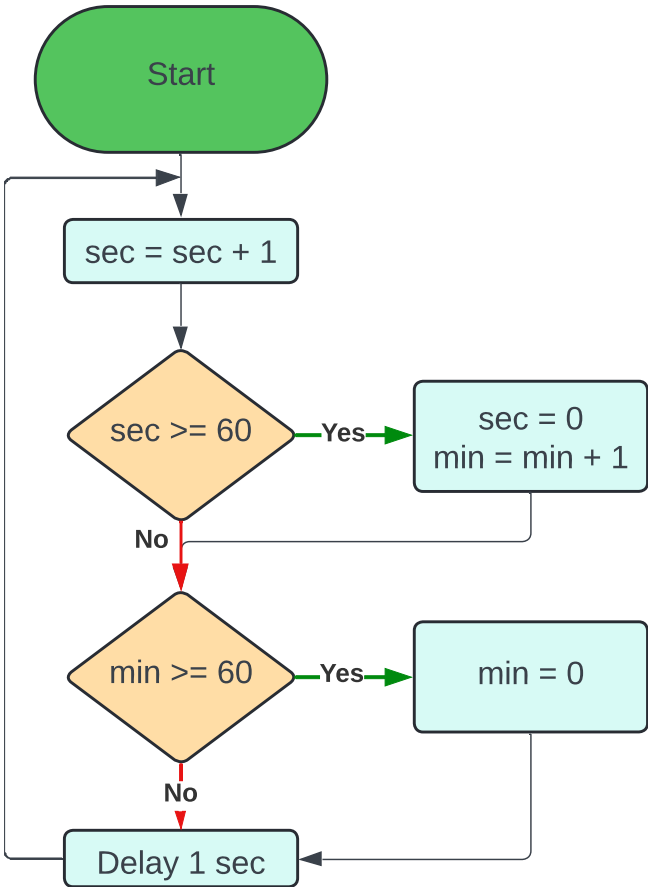

Figure 20: Update seven segment LED task

The module 4-digit 7-segment LED is used to display the running time of the system (maximum 1 hour because there are only 4 digits on the module). This 7-segment LED also does not require any library to use. Instead, there is an available code to control this module shown below.

```
// 4-digit 7-segment
unsigned char LED_0F[] =
{// 0   1     2   3    4    5     6   7    8    9    A    b    C    d     E   F
  -
  0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90,0x8C,0xBF,0xC6,0xA1,0x86,0xFF
,0xbf
};
unsigned char LED[4];

void LED_OUT(unsigned char X)
{
  unsigned char i;
  for(i=8;i>=1;i--)
  {
    if (X&0x80)
      digitalWrite(DIO_PIN,HIGH);
    else
      digitalWrite(DIO_PIN,LOW);
    X<<=1;
    digitalWrite(SCLK_PIN,LOW);
    digitalWrite(SCLK_PIN,HIGH);
  }
}
```

```
22
23    void LED4_Display(void)
24    {
25      unsigned char *led_table;
26      unsigned char i;
27      led_table = LED_0F + LED[0];
28      i = *led_table;
29
30      LED_OUT(i);
31      LED_OUT(0x01);
32      digitalWrite(RCLK_PIN,LOW);
33      digitalWrite(RCLK_PIN,HIGH);
34
35      led_table = LED_0F + LED[1];
36      i = *led_table;
37      LED_OUT(i);
38      LED_OUT(0x02);
39      digitalWrite(RCLK_PIN,LOW);
40      digitalWrite(RCLK_PIN,HIGH);
41
42      led_table = LED_0F + LED[2];
43      i = *led_table;
44      LED_OUT(i);
45      LED_OUT(0x04);
46      digitalWrite(RCLK_PIN,LOW);
47      digitalWrite(RCLK_PIN,HIGH);
48
49      led_table = LED_0F + LED[3];
50      i = *led_table;
51
52      LED_OUT(i);
53      LED_OUT(0x08);
54      digitalWrite(RCLK_PIN,LOW);
55      digitalWrite(RCLK_PIN,HIGH);
56    }
```

Listing 4: Light Sensor Read

This given code used an array **LED** containing 4 elements, in which LED[3] and LED[2] are used to display minutes while LED[1] and LED[0] are used to display seconds. The logic to process each of these 4 elements is also simple to understand. Whenever the LED[0] reaches 10, it means we have already counted for 1 more ten seconds, then LED[0] is returned to 0 while LED[1] is increased by 1. Whenever the LED[1] reaches 6, it means we already counted for 1 more minute, then LED[1] is returned to 0 and LED[2] is increased by 1. Whenever the LED[2] reaches 10, it means we have already counted for 10 minutes, then LED[2] is returned to 0 while LED[3] is increased by 1. Whenever LED[3] reaches 6, it means we have already counted for 1 hour, then LED[3] is returned to 0. The idea is shown clearly in figure 20 and the code below.

```
1     void task_update_seven_seg(void *pvParameters)
2     {
3       while(1)
4       {
5         LED[0] = LED[0] + 1;
6         if(LED[0] >= 10)
7         {
8           LED[0] = 0;
9           LED[1] = LED[1] + 1;
10        }
11        if(LED[1] >= 6)
12        {
13          LED[1] = 0;
14          LED[2] = LED[2] + 1;
15        }
```

```
16          if(LED[2] >= 10)
17          {
18            LED[2] = 0;
19            LED[3] = LED[3] + 1;
20          }
21          if(LED[3] >= 6)
22            LED[3] = 0;
23
24          vTaskDelay(1000);
25        }
26      }
```

Listing 5: Light Sensor Read

4.3 CoreIOT Dashboard

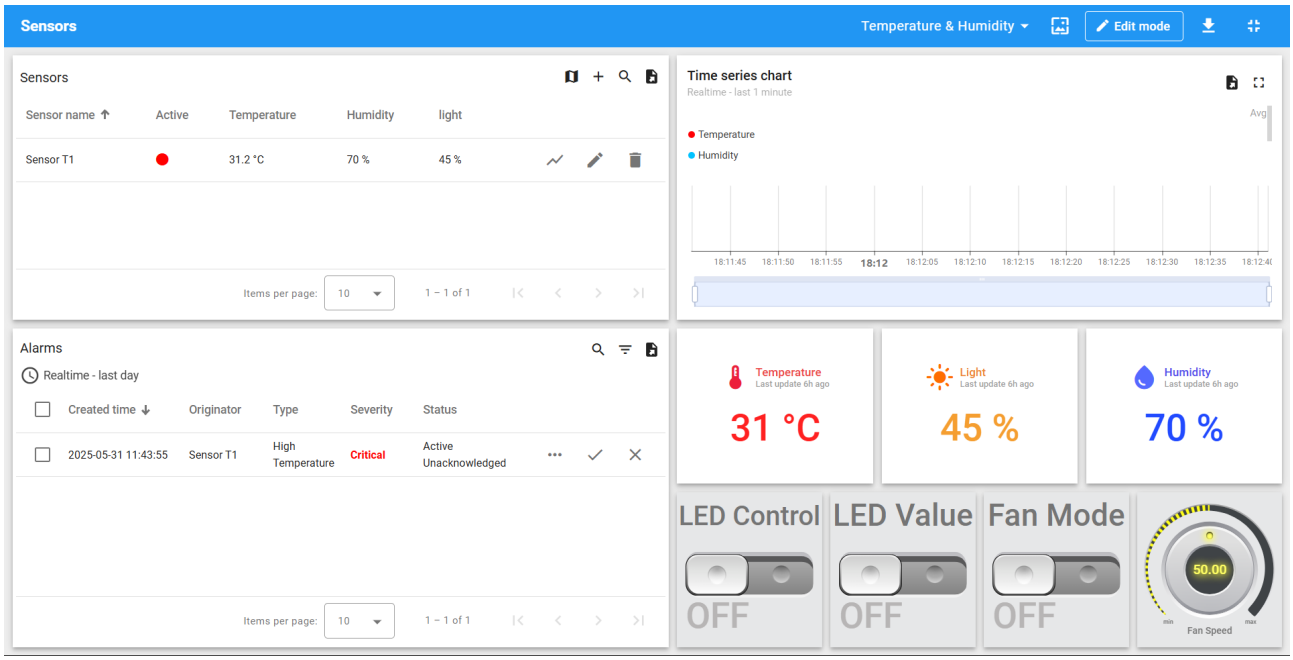

Figure 21: CoreIOT dashboard

These are elements contained in the dashboard:

- **Entity table:** This table contains all utilized sensors together with the values collected from the temperature & humidity sensor and the light sensor. In this assignment, we only have one device called **Sensor T1** as shown in the figure 21.
- **Alarms:** This table shows all abnormal values for temperature and humidity that we collect from the DHT20 sensor.
- **Time series chart:** This chart visualizes the collected value of temperature, humidity and light. Moreover, it also shows the predicted values for temperature and humidity.
- **Three value cards:** These cards are used to display the value of actual temperature, humidity and light.
- **LED Mode switch:** This switch is used to allow users to choose between **MANUAL** and **AUTOMATIC** working operation modes of the Tiny RGB LED. By default, the LED Mode is at state **OFF**, corresponding to **AUTOMATIC** mode. Users can switch it on to change to **MANUAL** mode.

- **LED Value switch:** This switch is used to allow users to turn on or off the Tiny RGB LED in **MANUAL** mode. In **AUTOMATIC** mode (switch **LED Control** is off), any operations on this switch have no effect on the operation of the system.
- **Fan Mode switch:** This switch is used to allow users to choose between **MANUAL** and **AUTOMATIC** working operation modes of the Mini Fan. By default, the Fan Mode is at state **OFF**, corresponding to **AUTOMATIC** mode. Users can switch it on to change to **MANUAL** mode.
- **Fan Value knob:** This knob is used to allow users to adjust the fan speed of the Mini Fan in **MANUAL** mode. In **AUTOMATIC** mode (switch **Fan Mode** is off), any operations on this knob have no effect on the operation of the system.

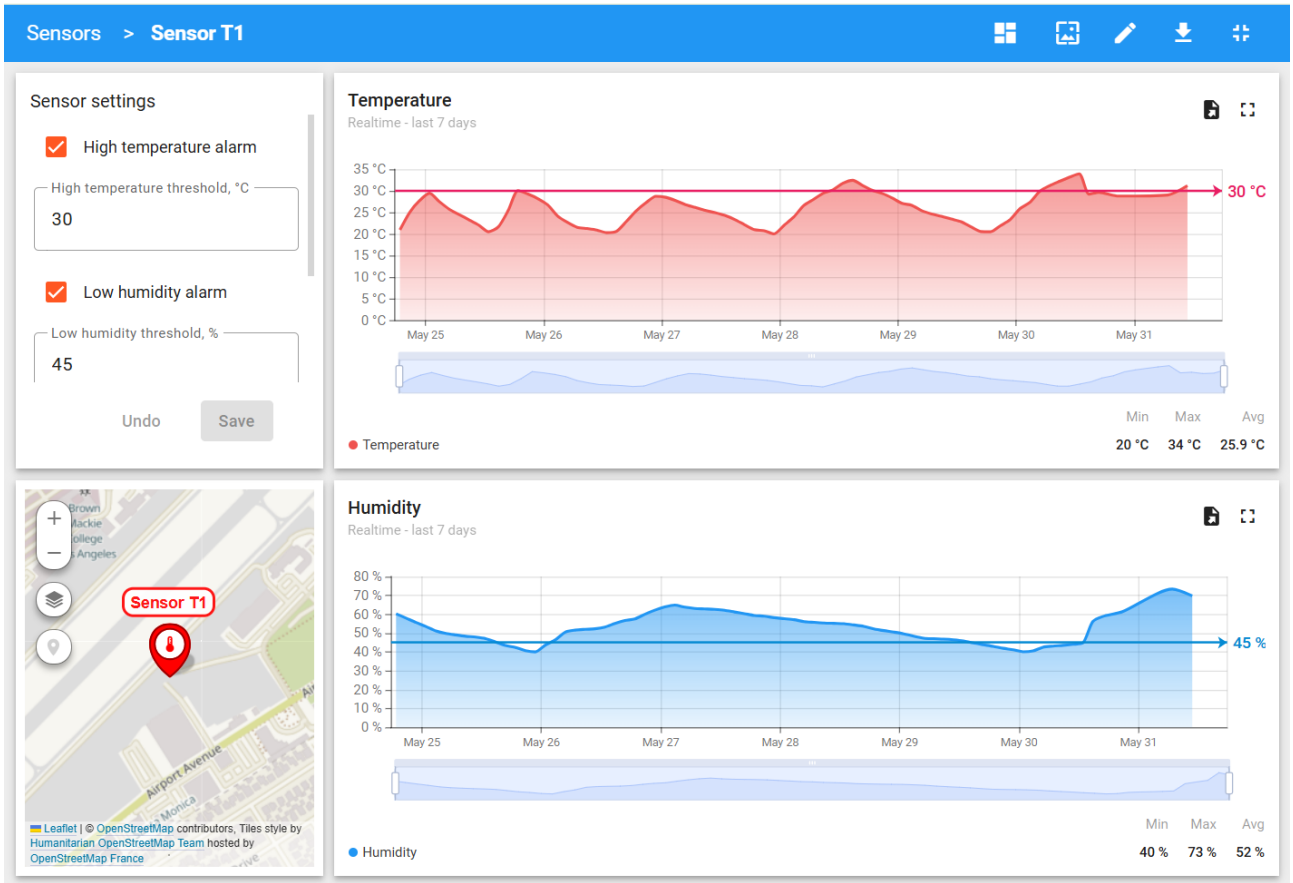

Figure 22: Sensor T1 section

4.4 Prediction Model

4.4.1 CNN Model Concept

A one-dimensional Convolutional Neural Network (CNN) is designed to capture local temporal correlations in multivariate time-series data. By applying convolution filters over sliding windows of three time steps (consisting of temperature and humidity readings), the model learns short-term patterns and then maps these features to a single-step prediction of both temperature and humidity.

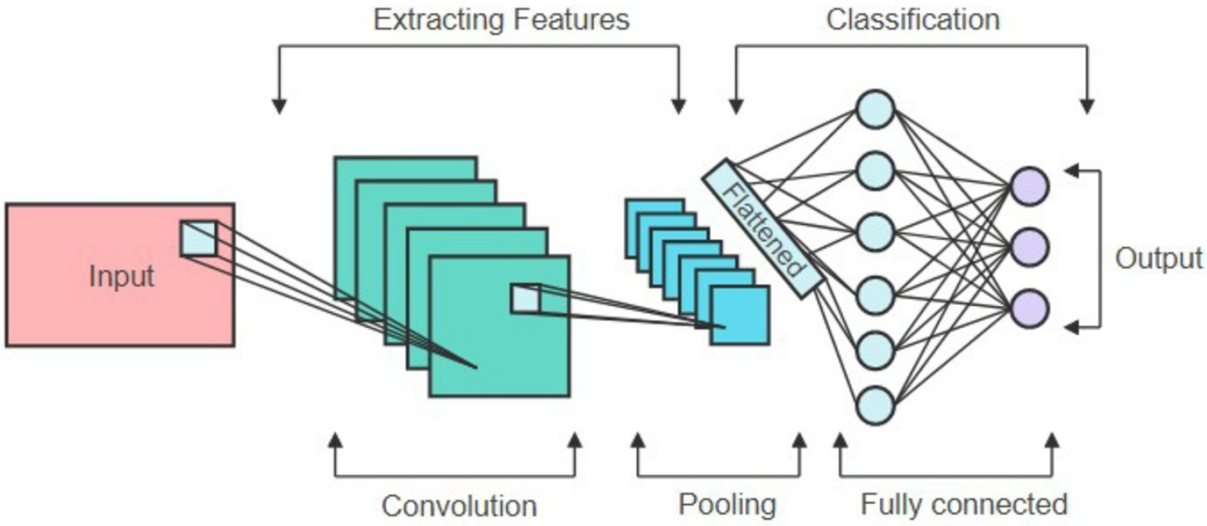

Figure 23: Sample of CNN model

4.4.2 Implement Model and Output

The CNN model takes the previous three readings of temperature and humidity as input and predicts the next **temperature** and **humidity** values simultaneously.

Model Architecture and Training

The CNN model was implemented with the following architecture:

- **Input Layer**: Shape $(n\_steps = 3, n\_features = 2)$ for 3 timesteps of 2 variables.
- **Conv1D Layer**: 64 filters, kernel size 2, ReLU activation.
- **MaxPooling1D Layer**: Pool size 2 to reduce temporal dimension.
- **Flatten Layer**: Flattens the feature maps into a vector.
- **Dense Layer**: 50 units, ReLU activation.
- **Output Layer**: Dense with $n\_features = 2$ units (linear activation) to predict.

Layer (type)	Output Shape	Param #
conv1d_46 (Conv1D)	(None, 2, 64)	320
max_pooling1d_46 (MaxPooling1D)	(None, 1, 64)	0
flatten_46 (Flatten)	(None, 64)	0
dense_92 (Dense)	(None, 50)	3,250
dense_93 (Dense)	(None, 2)	102

Total params: 3,672 (14.34 KB)
Trainable params: 3,672 (14.34 KB)
Non-trainable params: 0 (0.00 B)

Figure 24: CNN model structure

The model was compiled with:

- **Optimizer**: Adam
- **Loss Function**: Mean Squared Error (MSE)
- **Metrics**: Mean Absolute Error (MAE), Root Mean Squared Error (RMSE)

The training process included:

- **Number of Epochs**: 200
- **Verbose Mode**: 1 (displays progress bar and performance per epoch).

Model Export and Deployment

The trained CNN model can be saved in HDF5 or TensorFlow Lite formats for deployment on edge devices or cloud servers. This enables real-time predictions for temperature and humidity using sliding-window inference on incoming sensor streams.

Significance and Advantages

The CNN model offers:

- **Local Pattern Extraction**: Efficiently captures short-term trends in sensor data.
- **Parameter Efficiency**: Convolutional layers share weights, reducing model size.
- **Fast Inference**: Lightweight architecture suitable for real-time edge deployment.
- **Joint Prediction**: Simultaneously forecasts both temperature and humidity in 1 forward pass.

Experimental Result

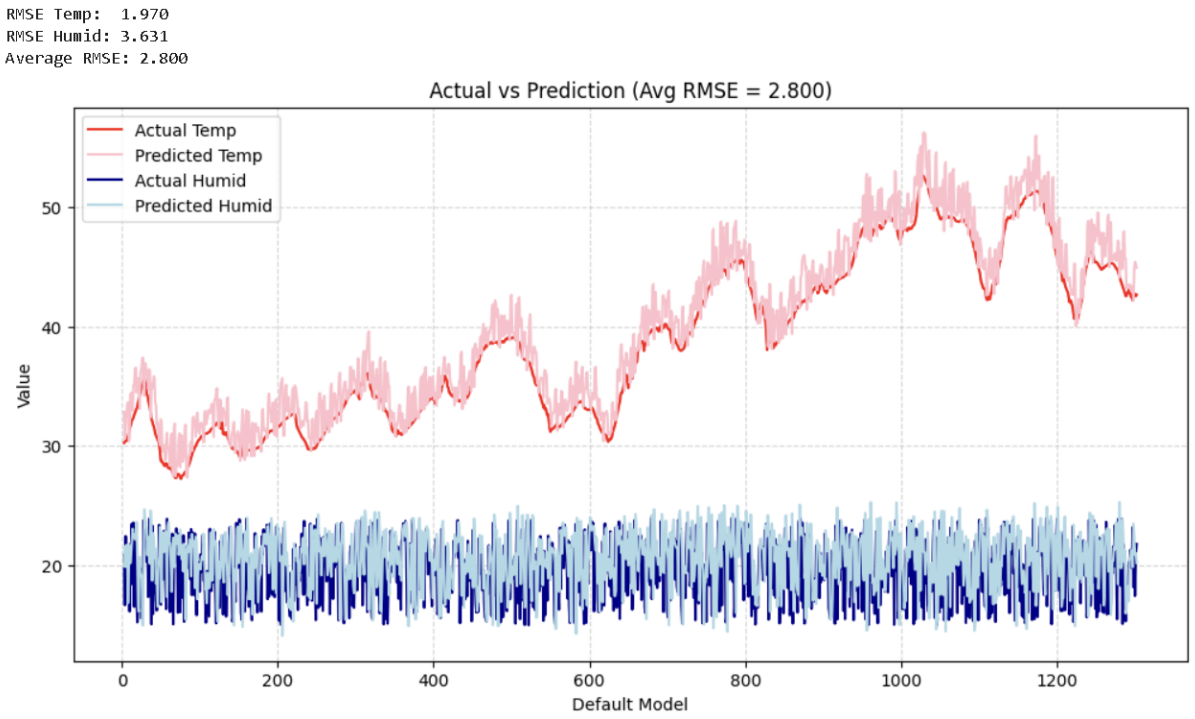

Figure 25: CNN model prediction

The mean **RMSE** on the test set is 1.970 for temperature and 3.631 for humidity.

4.5 Rule Chains

The system utilizes **Rule Chains** in ThingsBoard to process and send alerts based on the sensor data. The rule chain is designed to include several core stages: processing actual sensor data (*Actual Data*), handling prediction results (*Predicted Data*), formatting timestamps (*Datetime*), evaluating alert conditions (*Notification*), constructing notification content (*Alert*), and managing email delivery configuration (*Configuration*).

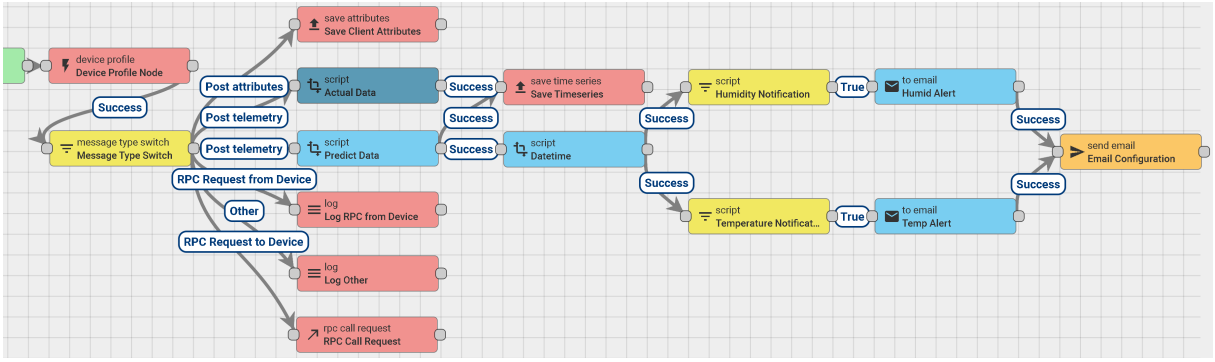

Figure 26: Set up Rule chains

Actual & Predicted Data

The **Actual Data** node (script) receives real-time sensor values (temperature, humidity, light) from the device. The **Predicted Data** node (script) handles forecasted values for temperature and humidity (`predicted_temp`, `predicted_humid`) produced by the device or a prediction model.

Datetime

The **Datetime** node (script) converts the timestamp into a human-readable local time string (for example, `hh:mm:ss, dd-mm-yyyy`). This formatted time is included in the alert emails to make notifications easier for users to understand.

Notification (Condition Checking)

The **Temperature Notification** and **Humidity Notification** nodes (script) act as filters that evaluate whether alert conditions are met. Specifically, if the predicted temperature or humidity exceeds a certain threshold, the flow proceeds to the alert creation stage.

Alert

The **Temp Alert** and **Humid Alert** nodes (email) build and customize the content of the email notifications for temperature and humidity alerts. The email templates are dynamically filled with device information, predicted values, and the forecasted time of occurrence.

Configuration (Email Delivery)

The **Email Configuration** node is responsible for the email delivery process. It contains the SMTP settings, sender and recipient information, and security protocols (TLS) needed to connect to the mail server and send alert emails to end users.

4.6 Web Implementation

4.6.1 Design Smart Home UI

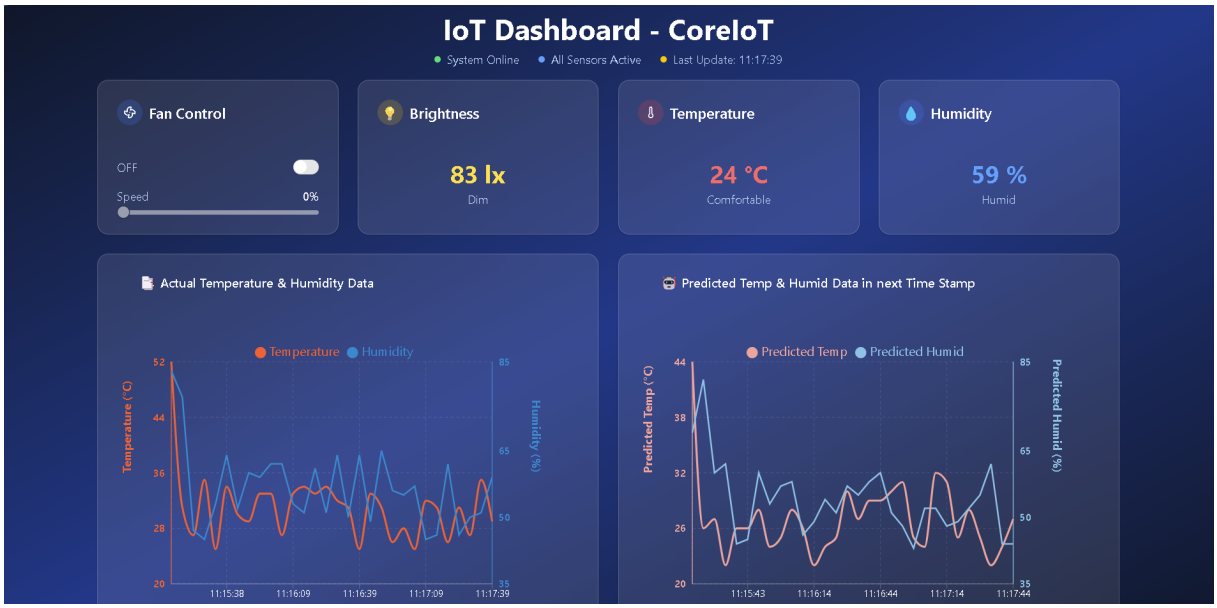

Figure 27: Web UI

The CoreIoT web dashboard is built upon the Vite platform in conjunction with React and TypeScript, yielding both rapid development performance and a smooth user experience. The adoption of Vite optimizes build and reload times during development, while React facilitates the construction of a dynamic user interface, and TypeScript enforces consistency and type safety across the entire project.

The dashboard's interface has been designed with a modern, intuitive layout, comprising four principal information panels: Fan Control, Brightness, Temperature, and Humidity. These interface components are developed using the shadcn-ui library, combined with radix-ui to ensure accessibility and a consistent user experience. Iconography within the dashboard is provided by lucid-react, thereby establishing clear visual consistency and functional recognition for each control element.

In terms of presentation, the entire application employs Tailwind CSS to manage styles via a utility-first approach, which enhances flexibility and maintainability of the interface. The predominant color scheme consists of navy tones complemented by gradients, thereby conveying a modern aesthetic appropriate to the IoT domain.

Sensor data are rendered dynamically through two distinct charts: one chart displays real-time temperature and humidity measurements, while the other presents predicted data for upcoming time intervals. Both charts are implemented using the Recharts library, which enables clear visualization of environmental metrics and supports effective monitoring and analysis.

System and sensor statuses are continuously updated in real time, as indicated by the "System Online," "All Sensors Active," and "Last Updated" indicators. This functionality demonstrates that the system integrates a data synchronization mechanism, whereby information from IoT devices is transmitted to the dashboard to ensure that all displayed data remain current and accurate.

In summary, the CoreIoT dashboard successfully realizes a web-based solution for environmental monitoring and control using modern web technologies. The application not only satisfies interface and user experience requirements but also ensures scalability and integration of real-world IoT data, thereby contributing to enhanced efficiency in system management and operation.

4.6.2 Set up API CoreIoT

To enable seamless data integration and monitoring, the system communicates directly with the CoreIoT platform through its public API. The following code snippets illustrate the main API calls for authentication and for fetching both historical and real-time telemetry data.

```
const loginRes = await fetch(
  'https://app.coreiot.io/api/auth/login', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ username: USER, password: PASS })
})
```

Listing 6: Login CoreIoT Account

Login to CoreIoT: Request sends the user credentials (username and password) to the CoreIoT authentication endpoint. If the credentials are valid, the server responds with an access token. Then the token is required for all subsequent requests to authenticate and authorize the client.

```
const historyRes = await fetch(
  `https://app.coreiot.io/api/plugins/telemetry/DEVICE/${DEVICE_ID}` +
  `/values/timeseries?keys=temperature,humidity,light,fanspeed,predicted_temp,
    predicted_humid` +
  `&startTs=${history_last_hour}&endTs=${now}`,
  { headers: { Authorization: `Bearer ${token}` } }
)
```

Listing 7: Get History Data

History Data: Retrieve historical telemetry data for a specific device within a specified time range. The request includes the device ID and a list of keys (such as temperature, humidity, light, fanspeed, predicted_temp, and predicted_humid). The start and end timestamps define the window for which historical data is fetched.

```
const res = await fetch(
  `https://app.coreiot.io/api/plugins/telemetry/DEVICE/${DEVICE_ID}` +
  `/values/timeseries?keys=temperature,humidity,light,fanspeed,predicted_temp,
    predicted_humid&limit=1`,
  { headers: { Authorization: `Bearer ${token}` } }
)
```

Listing 8: Get Telemetry Data

Telemetry Data: Fetch the most recent telemetry data (limit = 1) for the specified device. The returned data includes the latest values of temperature, humidity, light, fanspeed, predicted_temp, and predicted_humid.

In a nutshell, these API calls allow the application to securely log in, retrieve **historical sensor data** over a given time range, and access **real-time telemetry** from the **CoreIoT platform**. Make it feasible to perform efficient data monitoring, visualization, and alerting based on up-to-date and historical device information.

5 Demo & Snapshot

Please access this Google Drive link to watch the demo video for our IoT assignment system: Demo Video. This is the link to our group's GitHub repository where we used while doing the assignment: GitHub repository.

Besides, the figure below shows the example for the notification in our design:

Figure 28: Notification about the Predicted Temperature - Humidity values

As a result, whenever the system predicts an upcoming extreme temperature or humidity event, an alert email is immediately sent to the user. This enables proactive monitoring and risk prevention, as illustrated in Figure 28.

6 Perspective

In this assignment, we tried our best to make the system able to support many services. These are something that we can extend the system further more:

- Adding many more sensors: Some sensors should be added to support basic operations of an IoT system such as atmospheric pressure sensor, smoke sensor, air quality sensor, etc to support more services to users.
- Changing AI model: We should improve a better AI model to have better prediction and more precise warnings to users.

7 Conclusion

In this report, we showed you how we designed and implemented our IoT assignment system. We used more sensors and devices than required and they work well as shown in the demo video. AI application is also applied to detect possible risks and send notifications to users through mail using rule chain. The system is cost-effective with low cost sensors and devices and scalable, which offers a reliable solution for many IoT application such as smart office, smart irrigation.