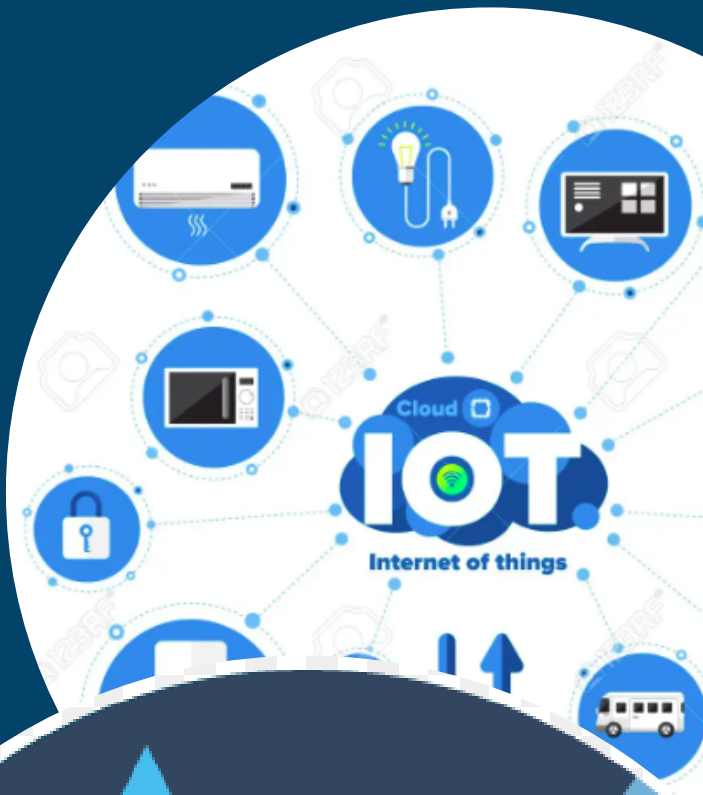
IoT-based Temperature and Humidity Monitoring System

PRESENTED BY
MAHAMMAD ARIF KOTWAL

JUNE 26



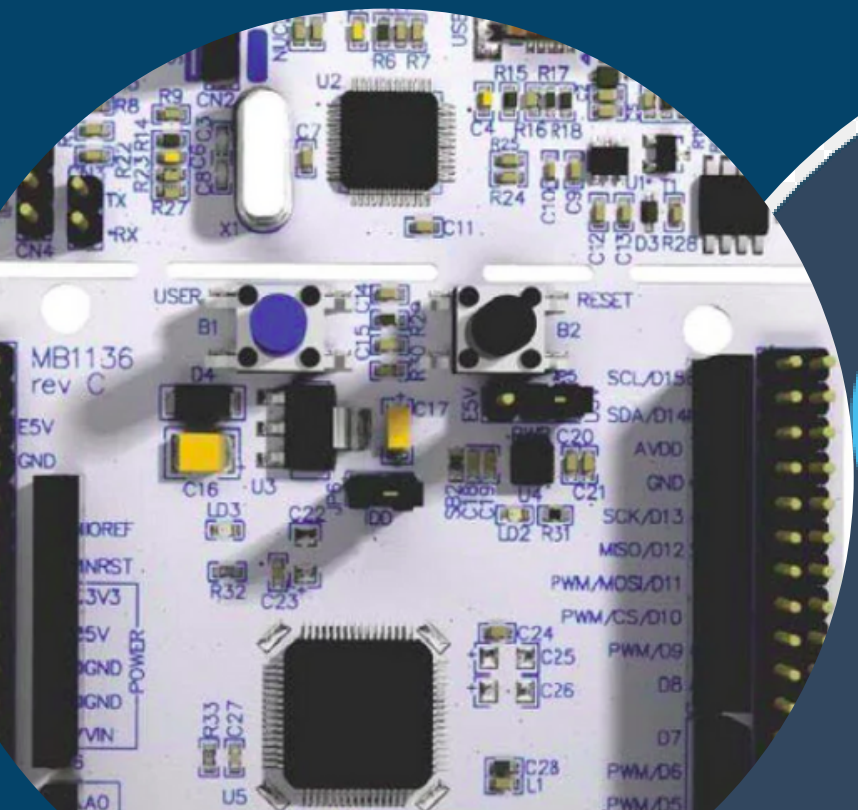


Table of Contents

1. Project Summary

The Internet of Things (IoT) is a rapidly growing field that has the potential to revolutionize the way we interact with the world around us. One of the most promising applications of IoT is in the area of environmental monitoring. By deploying IoT devices in various environments, we can collect real-time data on environmental conditions such as temperature, and humidity. This data can then be used to track changes in environmental conditions over time, identify potential problems, and take corrective action.

This project proposes the development of an IoT-based temperature and humidity monitoring system using STM32, AHT25, and W10 WiFi modules. The system will use an STM32 microcontroller to acquire sensor data from an AHT25 temperature and humidity sensor. The sensor data will then be published to MQTT using the W10 WiFi module, where it can be accessed by a web application or other IoT devices. The web application will allow users to visualize the sensor data and track changes in environmental conditions over time.
The system will be deployed in a real-world environment to test its performance. The deployment results will be used to assess the system's feasibility and identify any potential improvements.

The proposed system has the potential to be a valuable tool for tracking and monitoring environmental conditions. The system is relatively low-cost and easy to deploy, making it a feasible option for a wide range of applications. The system is also scalable, making it possible to deploy it in a variety of environments.

The results of this project will contribute to the body of knowledge on IoT-based environmental monitoring systems. The project will also provide a valuable case study for other researchers and developers who are interested in developing IoT-based ecological monitoring systems.

2. key features of the project

- Sensor Initialization with State Machine and Application state Machine maintained.
- UART Communication is based on Interrupt.
- Common function for UART receive data parsing.
- Timer based delay.
- STM32 Application is Master and configured with W10 (Wi-Fi) module accordingly with AT commands.
- FreeRTOS Task with priority implimantation.
- QUEUE are used for inter proces communication.
- Data is pushed to the MQTT server.
- Data is in JASON format.
- Temperature and Humidity data is pushed in every 1 Min.

3. Hardware Used and there Specification

1. STM32F411RE microcontroller

The STM32F411RE is a high-performance microcontroller based on the ARM Cortex-M4 processor. It has a number of features that make it well-suited for a variety of applications, including:

- 100 MHz CPU clock speed
- 128 KB of RAM
- 512 KB of Flash memory
- Floating point unit (FPU)
- 11 general-purpose timers
- 13 communication interfaces
- USB OTG
- RTC

2. AHT25 Humidity and Temperature Module

• Relative humidity and temperature output
• Superior sensor performance, typical accuracy RH: ±2%, T: ±0.3°C
• Fully calibrated and processed digital output, I²C protocol
• Wide voltage support 2.2 to 5.5V DC
• Excellent long-term stability
• Fast-response and anti-interference capability

3. W10 WiFi

The W10 WiFi module is a low-cost, easy-to-use WiFi module that can be used to connect IoT devices to the internet. The module has a built-in TCP/IP stack, so it can be easily connected to a variety of IoT platforms. The module also has a number of other features, such as:

- 100mW transmit power
- 11Mbps data rate
- 802.11 b/g/n compatibility
- Integrated antenna

4. Finite state machine implementation

This finite state machine has three states: STATE_INIT, STATE_READ_SENSOR, and STATE_PUBLISH_MQTT. The STATE_INIT state is the initial state, and it is where the sensor data is read. The STATE_READ_SENSOR state is where the sensor data is measured, and the PUBLISH state is where the sensor data is published to MQTT. The Callback in every 1 minute, and it updates the state of the finite state machine.

```
void StateMachine(void *argument)
{
while(1)
{
    switch (state) {
      case STATE_INIT:
        // Perform initialization tasks if necessary
           WE10_Init();
           MQTT_Init();

        // Transition to the next state
        state=STATE_READ_SENSOR;
        break;
      case STATE_READ_SENSOR:
        {
            read_sensor_values(&temperature, &humidity);
            state=STATE_PUBLISH_MQTT;
        }
        break;

      case STATE_PUBLISH_MQTT:
        {
          // Receive data from external source

          Send_Task(&argument);
          Recieve_Task(&argument);
          Callback(&p);
          // Transition to the next state
          state=STATE_READ_SENSOR;
        }
        break;
    }
  }
}
```

4.1. block diagram for finite state machine

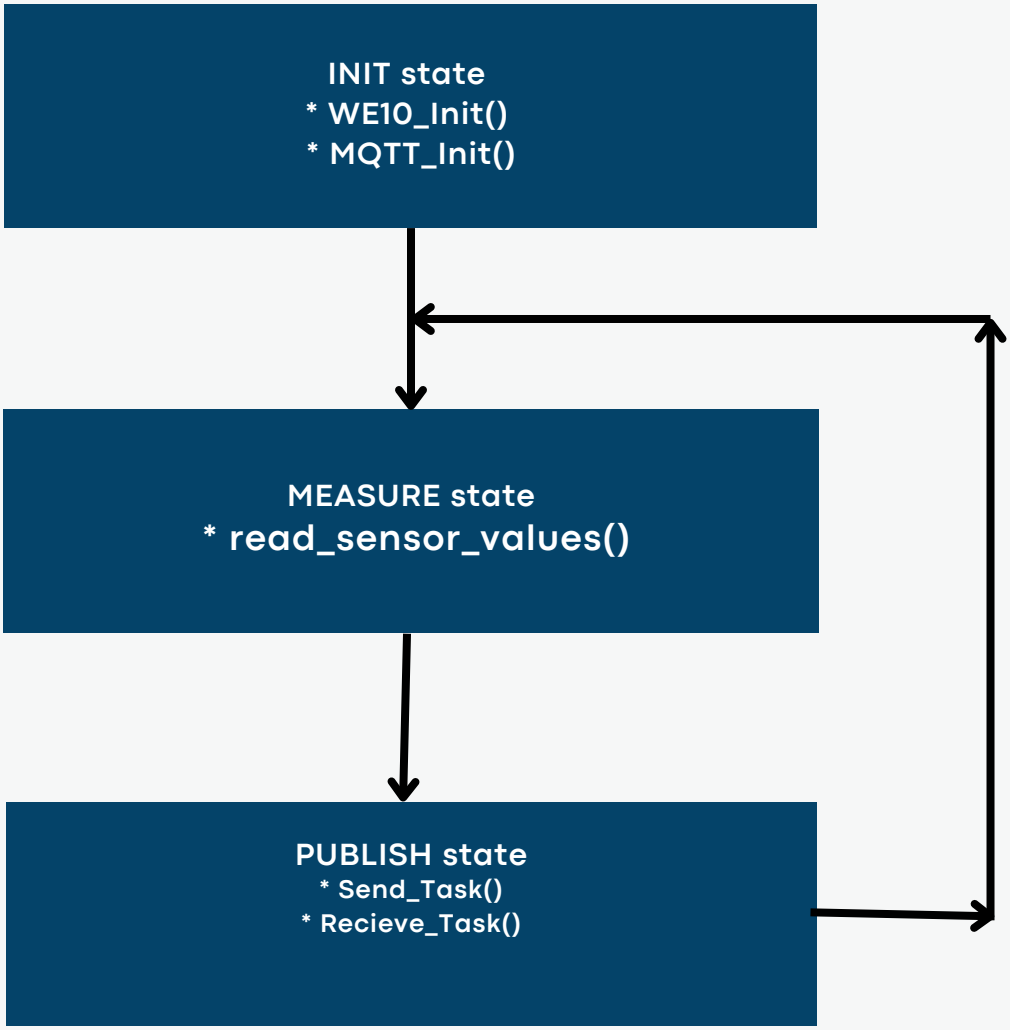


Figure.1

The block diagram shows the different states of the finite state machine and the actions that are performed in each state. The arrows represent the transitions between the states.

The Start block represents the beginning of the code. The State Machine block represents the main loop of the code. The STATE_INIT block represents the initialization state of the machine. The STATE_READ_SENSOR block represents the state where the sensor values are read and published to MQTT. The STATE_PUBLISH_MQTT block represents the state where data is sent and received from an external source. The End block represents the end of the code.

1. The StateMachine() function is called.
2. The switch() statement checks the current state of the machine.
3. If the current state is STATE_INIT, the WE10_Init() and MQTT_Init() functions are called to initialize the sensor and the MQTT connection.
4. The state is then changed to STATE_PUBLISH_MQTT.
5. If the current state is STATE_READ_SENSOR, the read_sensor_values() function is called to read the sensor values.
6. The state is then changed to STATE_PUBLISH_MQTT.
7. If the current state is STATE_PUBLISH_MQTT, the Send_Task() and Recieve_Task() functions are called to send and receive data from the external source.
8. The state is then changed to STATE_READ_SENSOR.
9. The while loop is repeated.

The StateMachine() function will continue to run until it is terminated. The flow of the code will depend on the current state of the machine.

Here is a more detailed explanation of the flow of the code:

- In the STATE_INIT state, the WE10_Init() and MQTT_Init() functions are called to initialize the sensor and the MQTT connection.
- In the STATE_READ_SENSOR state, the read_sensor_values() function is called to read the sensor values. The sensor values are then published to MQTT.
- In the STATE_PUBLISH_MQTT state, the Send_Task() and Recieve_Task() functions are called to send and receive data from the external source.
- The while loop is repeated until the StateMachine() function is terminated.

5. Why interrupt based uart communication

The advantages of interrupt-based UART communication over other methods:

- Improved efficiency: The CPU is not constantly polling the UART status, so it can be used for other tasks. This can improve the efficiency of the system and reduce power consumption.
- Reduced latency: Interrupt-based communication can be faster than polling, as the CPU is only interrupted when data is ready to be sent or received. This can improve the responsiveness of the system.
- Better multitasking: Interrupt-based communication allows the CPU to handle multiple UART events simultaneously. This can be useful for applications that require frequent and asynchronous serial communication.

Here are some of the disadvantages of interrupt-based UART communication:

- More complex code: Interrupt-based communication is more complex to implement than polling. This is because the programmer needs to write code to handle the interrupts.
- More overhead: Interrupt-based communication can have more overhead than polling. This is because the CPU needs to save and restore its state when it is interrupted.

Overall, interrupt-based UART communication is a more efficient and responsive way to communicate with serial devices. However, it is more complex to implement and can have more overhead.

Here are some examples of applications where interrupt-based UART communication would be a good choice:

- **Sensor networks:** Sensor networks often need to communicate with each other or with a central server. Interrupt-based communication can be used to improve the efficiency and responsiveness of these networks.
- **Wireless modules:** Wireless modules often use UART to communicate with the host microcontroller. Interrupt-based communication can be used to improve the performance of these modules.
- **Real-time systems:** Real-time systems often need to communicate with other devices in a timely manner. Interrupt-based communication can be used to ensure that these communications are not missed.

6. Connection diagram

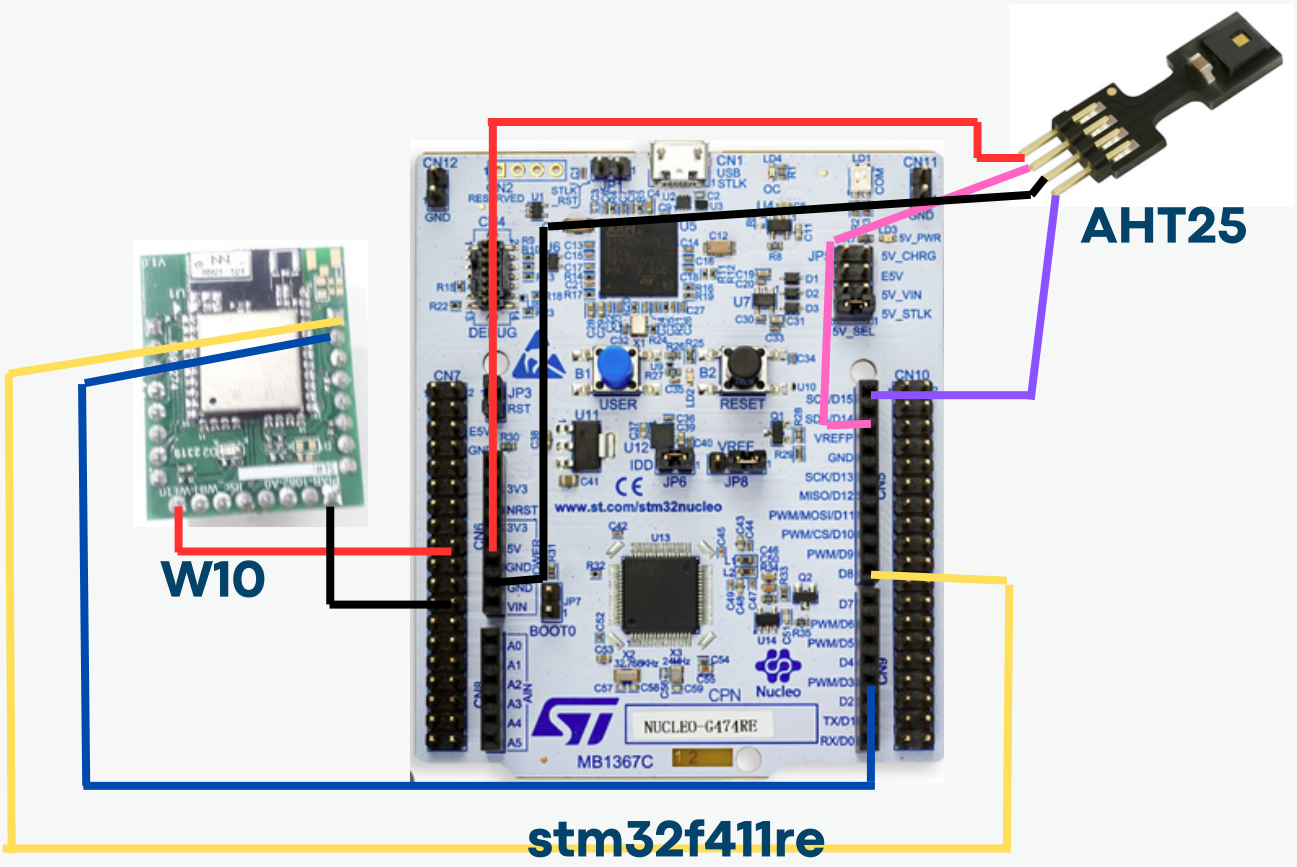


Figure.2

Pin configuration

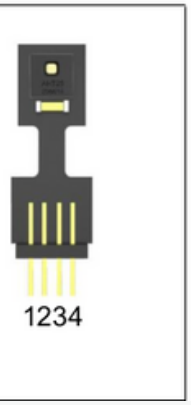
PIN	PIN NUMBER	COLOUR	COMPONENTS
vcc 5v	vcc	Red	w10 and AHT25
ground	GND	Black	w10 and AHT25
SCL	PB8	purple	AHT25
SDK	PB9	Pink	AHT25
Tx	PA9	yellow	W10
Rx	PA10	blue	W10

7. AHT25 Sensor

Product Description :

The AHT25 can be widely used in consumer electronics, medical, automotive, industrial, meteorological, and other fields, such as HVAC, dehumidifiers and refrigerators, and other home appliances, testing and testing equipment, and other related temperature and humidity testing and control products.

Pins	Name	Describe
1	VDD	Power supply(2.2v to 5.5v)
2	SDA	Serial Data Bidirectional port
3	GND	Ground
4	SCL	Serial clock Bidirectional port



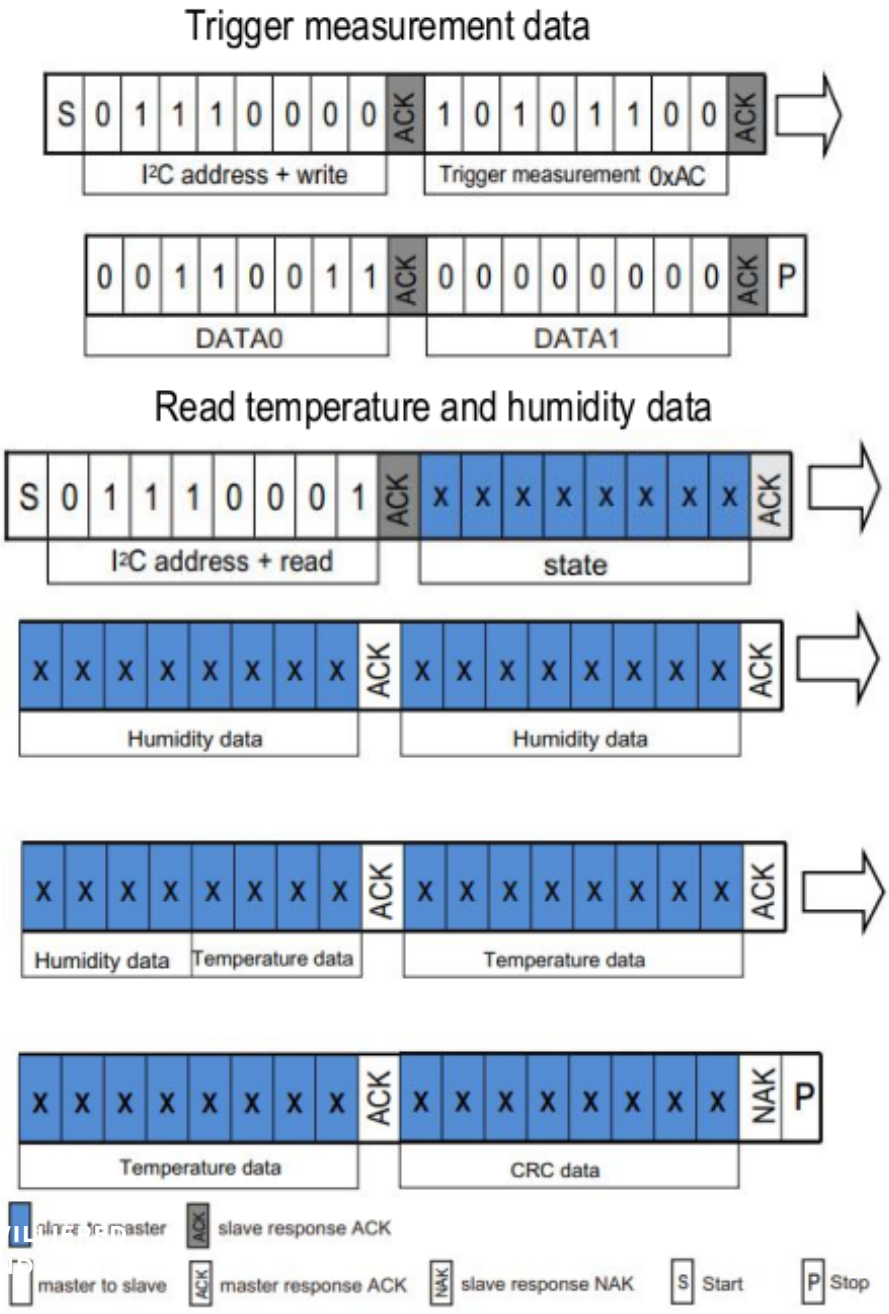
Note:

1. The power supply voltage of the host MCU must be consistent with the sensor when the product is used in the circuit.
2. If you need to further improve the reliability of the system, you can control the sensor power supply.
3. When the sensor is just powered on, give priority to the sensor VDD power supply, SCL and SDA high level can be set after 5ms.

To avoid signal conflicts, the microprocessor (MCU) must only drive SDA and SCL at low level. An external pull-up resistor (for example: 4.7kΩ) is required to pull the signal to a high level. Refer to Table 7 and Table 8 for detailed information about sensor input/output characteristics.

Slave address and configurations:

AHT25 read address	0x70
AHT25 write address	0x71
AHT25 start measure address	0xAC



Signal Conversion :

Relative Humidity Conversion :

The relative humidity RH can be calculated according to the relative humidity signal S RH output by SDA through the following formula

$$RH[\%] = (\frac{S_{RH}}{2^{20}}) * 100\%$$

Temperature Conversion :

The temperature T can be calculated by substituting the temperature output signal S T into the following formula:

(The result is expressed in temperature ℃):

$$T[^{\circ}C] = (\frac{S_T}{2^{20}}) * 200 - 50$$

In c Code conversion :

```
void read_sensor_values(float *temperature, float *humidity)
{
    uint8_t data[6];
    uint8_t cmd = AHT25_MEASURE_CMD;
    HAL_I2C_Master_Transmit(&hi2c1, AHT25_ADDR,&cmd, 1, HAL_MAX_DELAY);
    HAL_Delay(100);
    HAL_I2C_Master_Receive(&hi2c1,0x71, data, 6, HAL_MAX_DELAY);
    *humidity = ((float)((data[1] << 12) | (data[2] << 4) | (data[3] >> 4))) / 1048576.0 * 100.0;
    *temperature = ((float)(((data[3] & 0x0F) << 16) | (data[4] << 8) | data[5])) / 1048576.0 * 20
}
```

The code first declares two variables, data and cmd. The data variable will be used to store the raw sensor data, and the cmd variable will be used to send the measurement command to the sensor.
The next few lines of code send the measurement command to the sensor and then wait 100 milliseconds for the sensor to complete the measurement.
The next line of code reads the raw sensor data into the data variable.
The last two lines of code convert the raw sensor data into floating-point values and store them in the temperature and humidity variables.
The read_sensor_values() function is a simple example of how to read sensor data from an AHT25 sensor using the HAL I2C driver. The function takes two pointers to floating-point variables as input, and it stores the temperature and humidity values in these variables. The function returns void.

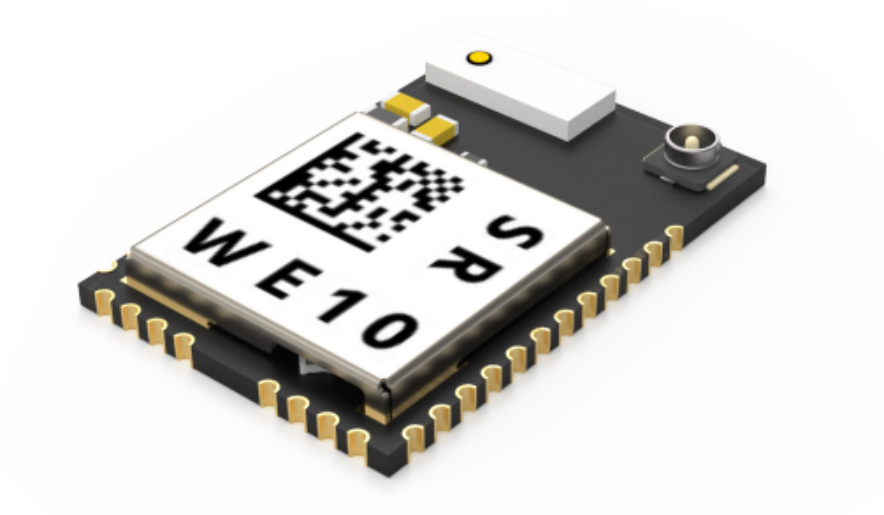
Signal Conversion :

```
*humidity = ((float)((data[1] << 12) | (data[2] << 4) | (data[3] >> 4))) / 1048576.0 * 100.0;
*temperature = ((float)(((data[3] & 0x0F) << 16) | (data[4] << 8) | data[5])) / 1048576.0 * 200.0 - 50.0;
```

The first line of code converts the raw humidity data from the sensor into a floating-point value. The raw humidity data is stored in the data[1], data[2], and data[3] bytes of the data array. The << and | operators are used to combine the three bytes into a single 32-bit integer. The / operator is then used to divide the integer by 1048576, which is the maximum value that can be stored in a 32-bit integer. The * operator is then used to multiply the result by 100, which converts the value into a percentage.

The second line of code converts the raw temperature data from the sensor into a floating-point value. The raw temperature data is stored in the data[3], data[4], and data[5] bytes of the data array. The & operator is used to mask out the lower 4 bits of the data[3] byte. The << and | operators are then used to combine the three bytes into a single 32-bit integer. The / operator is then used to divide the integer by 1048576, which is the maximum value that can be stored in a 32-bit integer. The * operator is then used to multiply the result by 200, which converts the value into degrees Celsius. The - 50.0 expression is then used to subtract 50 degrees Celsius from the result, which converts the value into degrees Fahrenheit.

8. W10 wifi module



The W10 WiFi module is a low-cost, easy-to-use WiFi module that can be used to connect IoT devices to the internet. The module has a built-in TCP/IP stack, so it can be easily connected to a variety of IoT platforms. The module also has a number of other features, such as:

- 100mW transmit power
- 11Mbps data rate
- 802.11 b/g/n compatibility
- Integrated antenna

9.code snippet to wifi module initialization and connection with interrupt-based uart commands

```
void WE10_Init ()
{
    char buffer[128];
    /********* CMD+RESET **********/
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD+RESET\r\n");
    HAL_UART_Transmit_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
    HAL_Delay(5000);
    HAL_UART_Receive_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
    /*********  CMD+WIFIMODE=1  **********/
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD+WIFIMODE=1\r\n");
    HAL_UART_Transmit_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
    HAL_Delay(2000);
    HAL_UART_Receive_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
    /********* CMD+CONTOAP=SSID,PASSWD **********/
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD+CONTOAP=MD ARIF,1234567890\r\n");
    HAL_UART_Transmit_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
    //memset(&buffer[0],0x00,strlen(buffer));
    HAL_Delay(5000);
    HAL_UART_Receive_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
    /********* CMD?WIFI**********/
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD?WIFI\r\n");
    HAL_UART_Transmit_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
     HAL_Delay(2000);
    HAL_UART_Receive_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
}
```

- The code first declares a buffer of 128 characters. The buffer will be used to store the commands that are sent to the WE10 module.
-
- The next few lines of code send the CMD+RESET command to the WE10 module. This command resets the module to its default state.
-
- The next line of code sends the CMD+WIFIMODE=1 command to the WE10 module. This command sets the module to operate in WiFi mode.
-
- The next line of code sends the CMD+CONTOAP=SSID, PASSWD command to the WE10 module. This command configures the module to connect to the WiFi network with the specified SSID and password.
-
- The next line of code sends the CMD.WIFI command to the WE10 module. This command queries the module for its WiFi status.
-
- The last line of code waits for 2000 milliseconds and then receives a response from the WE10 module. The response is stored in the buffer.
-
- The WE10_Init() function is a simple example of how to initialize a WE10 module and connect it to a WiFi network. The function takes no arguments and it returns void.

10.code snippet to MQTT initialization and connection with interrupt-based uart commands

```
void MQTT_Init()
{

    char buffer[128];
    HAL_Delay(2000);
    /*********CMD+MQTTNETCFG **********/
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD+MQTTNETCFG=dev.rightech.io,1883\r\n");
    HAL_UART_Transmit_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
     HAL_Delay(2000);
    HAL_UART_Receive_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
    /*********CMD+MQTTCONCFG **********/
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD+MQTTCONCFG=3,mqtt-arifm4348-ud8eo8,,,,,,,,,\r\n");
    HAL_UART_Transmit_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
     HAL_Delay(2000);
    HAL_UART_Receive_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
    /*********CMD+MQTTSTART **********/
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD+MQTTSTART=1\r\n");
    HAL_UART_Transmit_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
//  memset(&buffer[0],0x00,strlen(buffer));
    HAL_Delay(5000);
    HAL_UART_Receive_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
    /*********CMD+MQTTSUB *********/
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD+MQTTSUB=base/relay/led1\r\n");
    HAL_UART_Transmit_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
     HAL_Delay(2000);
    HAL_UART_Receive_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
}
```

The code you provided is a initialize a WE10 module and connect it to an MQTT broker. The code first declares a buffer of 128 characters. The buffer will be used to store the commands that are sent to the WE10 module.
The next few lines of code send the CMD+MQTTNETCFG command to the WE10 module. This command configures the module to connect to the MQTT broker at dev.rightech.io on port 1883. The CMD+MQTTCONCFG command configures the module to connect to the MQTT broker as a client with the username mqtt-arifm4348-ud8eo8 and no password. The CMD+MQTTSTART command starts the MQTT client and connects to the broker. The CMD+MQTTSUB command subscribes the client to the topic base/relay/led1.

The MQTT_Init() function is a simple example of how to initialize a WE10 module and connect it to an MQTT broker. The function takes no arguments and it returns void.
Here is a more detailed explanation of the code:

- The CMD+MQTTNETCFG command is used to configure the MQTT parameters of the WE10 module. The first parameter is the hostname or IP address of the MQTT broker. The second parameter is the port number of the MQTT broker.
- The CMD+MQTTCONCFG command is used to configure the MQTT client of the WE10 module. The first parameter is the username of the MQTT client. The second parameter is the password of the MQTT client.
- The CMD+MQTTSTART command is used to start the MQTT client of the WE10 module. This command connects the client to the MQTT broker.
- The CMD+MQTTSUB command is used to subscribe the MQTT client to a topic. The first parameter is the topic that the client wants to subscribe to.

```
/* Definitions for SendTask */
osThreadId_t SendTaskHandle;
const osThreadAttr_t SendTask_attributes = {
  .name = "SendTask",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityNormal,
};
/* Definitions for RecieveTask */
osThreadId_t RecieveTaskHandle;
const osThreadAttr_t RecieveTask_attributes = {
  .name = "RecieveTask",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityLow,
};
/* Definitions for myQueueTemp */
osMessageQueueId_t myQueueTempHandle;
const osMessageQueueAttr_t myQueueTemp_attributes = {
  .name = "myQueueTemp"
};
```

SendTask:

The SendTask is a task that will be created by the code.

The osThreadId_t SendTaskHandle variable is used to store the handle of the SendTask. The osThreadAttr_t SendTask_attributes structure defines the attributes of the SendTask.

The SendTask_attributes structure has three members:

- name: The name of the task.
- stack_size The size of the stack that will be allocated to the task.
- priority: The priority of the task.

In this case, the name of the task is "SendTask", the stack_size is 128 * 4 bytes, and the priority is osPriorityNormal.

The osThreadAttr_t structure is used to configure the attributes of a task. The name member is used to set the name of the task. The stack_size member is used to set the size of the stack that will be allocated to the task. The priority member is used to set the priority of the task.

RecieveTask:

The RecieveTask is a task that will be created by the code.

The osThreadId_t RecieveTaskHandle variable is used to store the handle of the RecieveTask. The osThreadAttr_t RecieveTask_attributes structure defines the attributes of the RecieveTask.

The RecieveTask_attributes structure has three members:

- name: The name of the task.
- stack_size The size of the stack that will be allocated to the task.
- priority: The priority of the task.

In this case, the name of the task is "RecieveTask", the stack_size is 128 * 4 bytes, and the priority is osPriorityLow.

The osThreadAttr_t structure is used to configure the attributes of a task. The name member is used to set the name of the task. The stack_size member is used to set the size of the stack that will be allocated to the task. The priority member is used to set the priority of the task.

myQueueTemp:

The myQueueTemp message queue will be created by the code.

The osMessageQueueId_t myQueueTempHandle variable is used to store the handle of the myQueueTemp message queue. The osMessageQueueAttr_t myQueueTemp_attributes structure defines the attributes of the myQueueTemp message queue.

The myQueueTemp_attributes structure has one member:

- name: The name of the message queue.

In this case, the name of the message queue is "myQueueTemp".

The osMessageQueueAttr_t structure is used to configure the attributes of a message queue. The name member is used to set the name of the message queue.

```
myTimerHandle = osTimerNew(Callback, osTimerPeriodic, NULL, &myTimer_attributes);
myQueueTempHandle = osMessageQueueNew (256, sizeof(uint16_t), &myQueueTemp_attributes);
SendTaskHandle = osThreadNew(Send_Task, NULL, &SendTask_attributes);
RecieveTaskHandle = osThreadNew(Recieve_Task, NULL, &RecieveTask_attributes);
osKernelStart();
```

The code you provided creates the myQueueTemp message queue, creates the SendTask and RecieveTask tasks, and starts the kernel.

The myQueueTempHandle = osMessageQueueNew (256, sizeof(uint16_t), &myQueueTemp_attributes); line creates the myQueueTemp message queue. The osMessageQueueNew() function is used to create a message queue. The first parameter is the maximum number of messages that can be stored in the queue, the second parameter is the size of each message, and the third parameter is a pointer to the attributes of the queue.

The SendTaskHandle = osThreadNew(Send_Task, NULL, &SendTask_attributes); line creates the SendTask task. The osThreadNew() function is used to create a task. The first parameter is the name of the task, the second parameter is a pointer to the function that will be executed by the task, and the third parameter is a pointer to the attributes of the task.

The RecieveTaskHandle = osThreadNew(Recieve_Task, NULL, &RecieveTask_attributes); line creates the RecieveTask task. The osThreadNew() function is used to create a task. The first parameter is the name of the task, the second parameter is a pointer to the function that will be executed by the task, and the third parameter is a pointer to the attributes of the task.

The osKernelStart(); line starts the kernel. The osKernelStart() function is used to start the kernel. The kernel is the core of the operating system. It is responsible for scheduling tasks, managing resources, and handling interrupts.

Send_Task Function :

```
void Send_Task(void *argument)
{
    data DatatoSend;
    while (1) {
        read_sensor_values(&temperature, &humidity);
        DatatoSend.temp = temperature;
        DatatoSend.humi = humidity;

        osMessageQueuePut(myQueueTempHandle, &DatatoSend, 0, 0);
        osDelay(1000);  // Delay for 1000 milliseconds (1 second) before the next iteration
    }
}
```

The code you provided defines the Send_Task function. The Send_Task is a task that will be executed by the code.

The Send_Task function first declares a variable of type data called DatatoSend.

The temp member of the data structure is used to store the temperature reading, and the humidity member of the data structure is used to store the humidity reading.

The Send_Task function then enters an infinite loop. In each iteration of the loop, the Send_Task function reads the temperature and humidity readings from the sensors, stores the readings in the DatatoSend structure, and then puts the DatatoSend structure on the myQueueTemp message queue. The osMessageQueuePut() function is used to put a message on a message queue. The first parameter is the handle of the message queue, the second parameter is a pointer to the message, the third parameter is the priority of the message, and the fourth parameter is the timeout value.

Send_Task Function :

```
void Recieve_Task(void *argument)
{
    data retvalue;
  for(;;)
  {
      osMessageQueueGet(myQueueTempHandle,&retvalue,NULL,osWaitForever);
          char buffer[128];
           sprintf (&buffer[0], "CMD+MQTTPUB=base/state/temperature,%.2f\r\n",retvalue.humi);
           HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer),1000);
          sprintf (&buffer[0], "CMD+MQTTPUB=base/state/temperature,%.2f\r\n",retvalue.temp);
          HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer),1000);
          HAL_TIM_Base_Start_IT(&htim2);
  }
}
```

The code you provided defines the Recieve_Task function. The Recieve_Task is a task that will be executed by the code.

The Recieve_Task function first declares a variable of type data called retvalue.

The temp member of the data structure is used to store the temperature reading, and the humi member of the data structure is used to store the humidity reading.

The Recieve_Task function then enters an infinite loop. In each iteration of the loop, the Recieve_Task function gets a message from the myQueueTemp message queue. The osMessageQueueGet() function is used to get a message from a message queue. The first parameter is the handle of the message queue, the second parameter is a pointer to the message, the third parameter is a pointer to the priority of the message, and the fourth parameter is the timeout value.

If the osMessageQueueGet() function succeeds, the Recieve_Task function then publishes the temperature and humidity readings to MQTT. The sprintf() function is used to format a string, and the HAL_UART_Transmit() function is used to transmit a string over UART.

The Recieve_Task function then starts the timer again.

12.Project code :

```
#include "main.h"
#include "cmsis_os.h"
#include"stdio.h"
#include"string.h"

#define AHT25_ADDR 0x70
#define AHT25_MEASURE_CMD 0xAC
float temperature, humidity;

I2C_HandleTypeDef hi2c1;
UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;
TIM_HandleTypeDef htim2;
/* Definitions for SendTask */
osThreadId_t SendTaskHandle;
const osThreadAttr_t SendTask_attributes = {
 .name = "SendTask",
 .stack_size = 128 * 4,
 .priority = (osPriority_t) osPriorityNormal,
};
/* Definitions for RecieveTask */
osThreadId_t RecieveTaskHandle;
const osThreadAttr_t RecieveTask_attributes = {
 .name = "RecieveTask",
 .stack_size = 128 * 4,
 .priority = (osPriority_t) osPriorityLow,
};
/* Definitions for myQueueTemp */
osMessageQueueId_t myQueueTempHandle;
const osMessageQueueAttr_t myQueueTemp_attributes = {
 .name = "myQueueTemp"
};
/* Definitions for myTimer */
osTimerId_t myTimerHandle;
const osTimerAttr_t myTimer_attributes = {
 .name = "myTimer"
};
/* USER CODE BEGIN PV */
typedef struct
{
float temp;
float humi;
}data;
/* USER CODE END PV */
int p=60000;

typedef enum {
 STATE_INIT,
 STATE_READ_SENSOR,
 STATE_PUBLISH_MQTT,
} state_t;

state_t state = STATE_INIT;

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM2_Init(void);
static void MX_I2C1_Init(void);
static void MX_USART1_UART_Init(void);
```

```
void Send_Task(void *argument);
void Recieve_Task(void *argument);
void Callback(void *argument);
void WE10_Init (void);
void MQTT_Init(void);
/* USER CODE BEGIN PFP */
void read_sensor_values(float *temperature, float *humidity)
{
uint8_t data[6];
 uint8_t cmd = AHT25_MEASURE_CMD;
 HAL_I2C_Master_Transmit(&hi2c1, AHT25_ADDR,&cmd, 1, HAL_MAX_DELAY);
 HAL_Delay(100);
 HAL_I2C_Master_Receive(&hi2c1,0x71, data, 6, HAL_MAX_DELAY);
 *humidity = ((float)((data[1] << 12) | (data[2] << 4) | (data[3] >> 4))) /
1048576.0 * 100.0;
 *temperature = ((float)(((data[3] & 0x0F) << 16) | (data[4] << 8) |
data[5])) / 1048576.0 * 200.0 - 50.0;
}

int main(void)
{
HAL_Init();
 SystemClock_Config();
 MX_GPIO_Init();
 MX_USART2_UART_Init();
 MX_I2C1_Init();
 MX_TIM2_Init();
 MX_USART1_UART_Init();
 WE10_Init();
 MQTT_Init();
 osKernelInitialize();


myTimerHandle = osTimerNew(Callback, osTimerPeriodic, NULL,
&myTimer_attributes);
 myQueueTempHandle = osMessageQueueNew (256, sizeof(uint16_t),
&myQueueTemp_attributes);
 SendTaskHandle = osThreadNew(Send_Task, NULL, &SendTask_attributes);
 RecieveTaskHandle = osThreadNew(Recieve_Task, NULL, &RecieveTask_attributes);
 osKernelStart();

while (1)
  {
  }

}
void StateMachine(void *argument)
{
while(1)
{
switch (state) {
 case STATE_INIT:
 // Perform initialization tasks if necessary
 WE10_Init();
 MQTT_Init();
 // Transition to the next state
 state=STATE_READ_SENSOR;
 break;
```

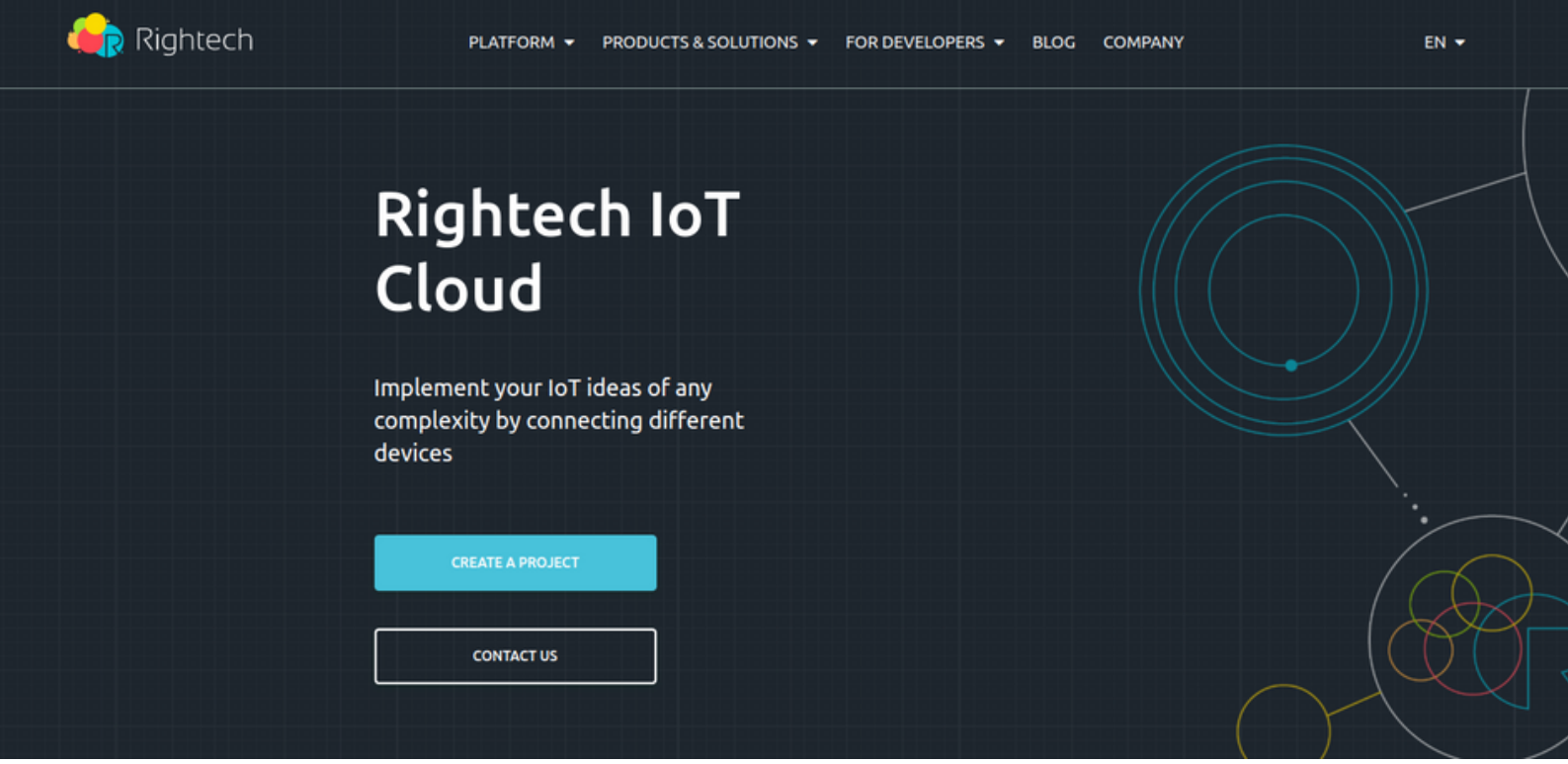
```
 case STATE_READ_SENSOR:
   {
 read_sensor_values(&temperature, &humidity);
 state=STATE_PUBLISH_MQTT;
   }
 break;

 case STATE_PUBLISH_MQTT:
   {
 // Receive data from external source

 Send_Task(&argument);
 Recieve_Task(&argument);
 Callback(&p);
 // Transition to the next state
 state=STATE_READ_SENSOR;
   }
 break;
   }
  }
}
static void MX_TIM2_Init(void)
{
/* USER CODE BEGIN TIM2_Init 0 */

 /* USER CODE END TIM2_Init 0 */

 TIM_SlaveConfigTypeDef sSlaveConfig = {0};
 TIM_MasterConfigTypeDef sMasterConfig = {0};
 /* USER CODE BEGIN TIM2_Init 1 */

 /* USER CODE END TIM2_Init 1 */
 htim2.Instance = TIM2;
 htim2.Init.Prescaler = 15999;
 htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
 htim2.Init.Period = 60000;
 htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
 htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
 if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
   {
 Error_Handler();
   }
 sSlaveConfig.SlaveMode = TIM_SLAVEMODE_EXTERNAL1;
 sSlaveConfig.InputTrigger = TIM_TS_ITR0;
 if (HAL_TIM_SlaveConfigSynchro(&htim2, &sSlaveConfig) != HAL_OK)
   {
 Error_Handler();
   }
 sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
 sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
 if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) !=
{
 Error_Handler();                                                    HAL_OK )
   }
 /* USER CODE BEGIN TIM2_Init 2 */
 /* USER CODE END TIM2_Init 2 */

}
void SystemClock_Config(void)
```

```
{
RCC_OscInitTypeDef RCC_OscInitStruct = {0};
 RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

 __HAL_RCC_PWR_CLK_ENABLE();

 __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
 RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
 RCC_OscInitStruct.HSIState = RCC_HSI_ON;
 RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
 RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
 if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
   {
 Error_Handler();
   }
 RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
 RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
 RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
 RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
 RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
   {
 Error_Handler();
   }
}
static void MX_I2C1_Init(void)
{

hi2c1.Instance = I2C1;

 hi2c1.Init.ClockSpeed = 100000;
 hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
 hi2c1.Init.OwnAddress1 = 0;
 hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
 hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
 hi2c1.Init.OwnAddress2 = 0;
 hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
 hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
 if (HAL_I2C_Init(&hi2c1) != HAL_OK)
   {
 Error_Handler();
   }
}
static void MX_USART1_UART_Init(void)
{
huart1.Instance = USART1;

 huart1.Init.BaudRate = 115200;
 huart1.Init.WordLength = UART_WORDLENGTH_8B;
 huart1.Init.StopBits = UART_STOPBITS_1;
 huart1.Init.Parity = UART_PARITY_NONE;
 huart1.Init.Mode = UART_MODE_TX_RX;
 huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
 huart1.Init.OverSampling = UART_OVERSAMPLING_16;
 if (HAL_UART_Init(&huart1) != HAL_OK)
   {
 Error_Handler();
   }
}
static void MX_USART2_UART_Init(void)
{
```

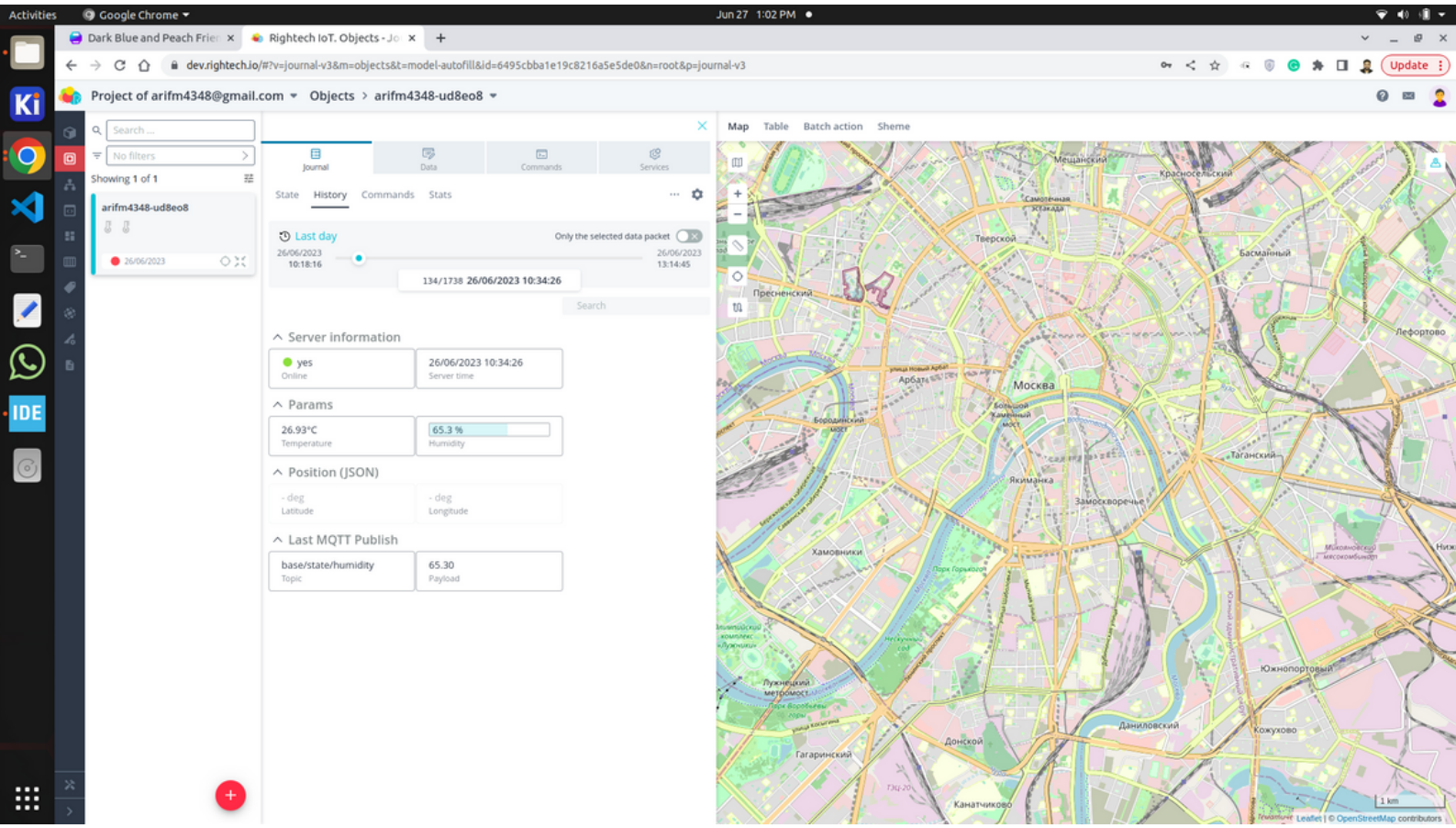
```
 huart2.Instance = USART2;
 huart2.Init.BaudRate = 115200;
 huart2.Init.WordLength = UART_WORDLENGTH_8B;
 huart2.Init.StopBits = UART_STOPBITS_1;
 huart2.Init.Parity = UART_PARITY_NONE;
 huart2.Init.Mode = UART_MODE_TX_RX;
 huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
 huart2.Init.OverSampling = UART_OVERSAMPLING_16;
 if (HAL_UART_Init(&huart2) != HAL_OK)
   {
 Error_Handler();
   }
}

static void MX_GPIO_Init(void)
{
__HAL_RCC_GPIOA_CLK_ENABLE();
 __HAL_RCC_GPIOB_CLK_ENABLE();
}

void Send_Task(void *argument)
{
data DatatoSend;
 while (1) {
 read_sensor_values(&temperature, &humidity);
 DatatoSend.temp = temperature;
 DatatoSend.humi = humidity;

 osMessageQueuePut(myQueueTempHandle, &DatatoSend, 0, 0);
 osDelay(1000); // Delay for 1000 milliseconds (1 second) before the
next iteration
   }
}


void Recieve_Task(void *argument)
{
data retvalue;
 for(;;)
   {
          osMessageQueueGet(myQueueTempHandle,&retvalue,NULL,osWaitForever);
 char buffer[128];
 sprintf (&buffer[0], "CMD+MQTTPUB=base/state/temperature,%.2f\r\
n",retvalue.humi);
 HAL_UART_Transmit(&huart2, (uint8_t*)buffer,
strlen(buffer),1000);
 sprintf (&buffer[0], "CMD+MQTTPUB=base/state/temperature,%.2f\r\
n",retvalue.temp);
                              HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer),1000);
 HAL_TIM_Base_Start_IT(&htim2);
   }
}
void WE10_Init ()
{
char buffer[128];
/********* CMD+RESET **********/
//memset(&buffer[0],0x00,strlen(buffer));
sprintf (&buffer[0], "CMD+RESET\r\n");
HAL_UART_Transmit_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
 HAL_Delay(5000);
```

```
HAL_UART_Receive_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
/********* CMD+WIFIMODE=1 **********/
//memset(&buffer[0],0x00,strlen(buffer));
sprintf (&buffer[0], "CMD+WIFIMODE=1\r\n");
HAL_UART_Transmit_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
HAL_Delay(2000);
HAL_UART_Receive_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
/********* CMD+CONTOAP=SSID,PASSWD **********/
//memset(&buffer[0],0x00,strlen(buffer));
sprintf (&buffer[0], "CMD+CONTOAP=MD ARIF,1234567890\r\n");
HAL_UART_Transmit_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
//memset(&buffer[0],0x00,strlen(buffer));
HAL_Delay(5000);
HAL_UART_Receive_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
/********* CMD?WIFI**********/
//memset(&buffer[0],0x00,strlen(buffer));
sprintf (&buffer[0], "CMD?WIFI\r\n");
HAL_UART_Transmit_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
 HAL_Delay(2000);
HAL_UART_Receive_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
}

void MQTT_Init()
{
char buffer[128];
HAL_Delay(2000);
/*********CMD+MQTTNETCFG **********/
//memset(&buffer[0],0x00,strlen(buffer));
sprintf (&buffer[0], "CMD+MQTTNETCFG=dev.rightech.io,1883\r\n");
HAL_UART_Transmit_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
 HAL_Delay(2000);
HAL_UART_Receive_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
/*********CMD+MQTTCONCFG **********/
//memset(&buffer[0],0x00,strlen(buffer));
                sprintf (&buffer[0], "CMD+MQTTCONCFG=3,mqtt-arifm4348-ud8eo8,,,,,,,,\r\
n");
HAL_UART_Transmit_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
 HAL_Delay(2000);
HAL_UART_Receive_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
/*********CMD+MQTTSTART **********/
//memset(&buffer[0],0x00,strlen(buffer));
sprintf (&buffer[0], "CMD+MQTTSTART=1\r\n");
HAL_UART_Transmit_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
// memset(&buffer[0],0x00,strlen(buffer));
HAL_Delay(5000);
HAL_UART_Receive_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
/*********CMD+MQTTSUB *********/
//memset(&buffer[0],0x00,strlen(buffer));
```

```
sprintf (&buffer[0], "CMD+MQTTSUB=base/relay/led1\r\n");
            HAL_UART_Transmit_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
            HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
 HAL_Delay(2000);
HAL_UART_Receive_IT(&huart1, (uint8_t*)buffer, strlen(buffer));
            HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
}
void Callback(void *argument) {

}
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
if (htim->Instance == TIM5) {
 HAL_IncTick();
  }
}

void Error_Handler(void)
{
__disable_irq();
 while (1)
  {
  }
}

#ifdef USE_FULL_ASSERT

void assert_failed(uint8_t *file, uint32_t line)
{
}
#endif /* USE_FULL_ASSERT */
```



Rightech IoT Cloud is a tool for developers. RIC is independent of specific equipment and protocols, which makes it easier for developers to combine different devices under one solution.Platform tools allow developers to create IoT solutions without extra code and reuse 90% of that solution to launch similar cases.



14.JSON (JavaScript Object Notation):

We can download the stored data from Rightech IoT in different formats and those are

- GPX, or GPS Exchange Format, is an XML schema designed as a common GPS data format for software applications.
- CSV (comma-separated values) file is a text file that has a specific format that allows data to be saved in a table-structured format.
- JSON (JavaScript Object Notation, pronounced /ˈdʒeɪsən/; also /ˈdʒeɪ ˌsɒn/) is an open standard file format and data interchange format that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays (or other serializable values).

I have downloaded it in JSON format which is given below.


```
{
    "_org": "64632961858b98a72bb922ef",
    "_mid": "64785fff858b98a72bb94534",
    "_oid": "6495cbba1e19c8216a5e5de0",
    "humidity": 63.93,
    "_ts": 1687758253540151,
    "id": "6495cbba1e19c8216a5e5de0",
    "_lic": "5d3b5ff00a0a7f30b695afe3",
    "online": true,
    "time": 1687758253540,
    "_bot": false,
    "payload": "63.93",
    "_gid": "64632961858b98a72bb922ef",
    "topic": "base/state/humidity",
    "temperature": 27.77,
    "_id": "649923cac6849e1137383547"
}
```

The JSON object you provided contains information about a humidity and temperature reading. The object has the following properties:

- _org: The organization ID.
- _mid: The message ID.
- _oid: The object ID.
- humidity: The humidity reading.
- _ts: The timestamp of the reading.
- id: The ID of the device that sent the reading.
- _lic: The license ID.
- online: Whether the device is online.
- time: The time of the reading.
- _bot: Whether the device is a bot.
- payload: The payload of the message.
- _gid: The group ID.
- topic: The topic of the message.
- temperature: The temperature reading.
- _id: The ID of the JSON object.

The topic property indicates that the message was published to the base/state/humidity topic. The payload property contains the value of the humidity reading, which is 63.93. The temperature property contains the value of the temperature reading, which is 27.77.

15.JSON code :


```
{
   "_org":    "64632961858b98a72bb922ef",
"_mid": "64785fff858b98a72bb94534",
"_oid": "6495cbba1e19c8216a5e5de0",
"humidity": 64.2,
"_ts": 1687757972005852,
"id": "6495cbba1e19c8216a5e5de0",
"_lic": "5d3b5ff00a0a7f30b695afe3",
"online": true,
"time": 1687757972005,
"_bot": false,
"payload": "64.20",
"_gid": "64632961858b98a72bb922ef",
"topic": "base/state/humidity",
"temperature": 27.64,
"_id": "649923cac6849e1137383547"
}, {
   "_org":    "64632961858b98a72bb922ef",
"_mid": "64785fff858b98a72bb94534",
"_oid": "6495cbba1e19c8216a5e5de0",
"humidity": 64.26,
"_ts": 1687757978377336,
"id": "6495cbba1e19c8216a5e5de0",
"_lic": "5d3b5ff00a0a7f30b695afe3",
"online": true,
"time": 1687757978377,
"_bot": false,
"payload": "64.26",
"_gid": "64632961858b98a72bb922ef",
"topic": "base/state/humidity",
"temperature": 27.64,
"_id": "649923cac6849e1137383547"
}, {
   "_org":    "64632961858b98a72bb922ef",
"_mid": "64785fff858b98a72bb94534",
"_oid": "6495cbba1e19c8216a5e5de0",
"humidity": 64.24,
"_ts": 1687757984860343,
"id": "6495cbba1e19c8216a5e5de0",
"_lic": "5d3b5ff00a0a7f30b695afe3",
"online": true,
"time": 1687757984860,
"_bot": false,
"payload": "64.24",
"_gid": "64632961858b98a72bb922ef",
"topic": "base/state/humidity",
"temperature": 27.64,
"_id": "649923cac6849e1137383547"
}, {
   "_org":    "64632961858b98a72bb922ef",
"_mid": "64785fff858b98a72bb94534",
"_oid": "6495cbba1e19c8216a5e5de0",
```

"humidity": 64.22,
"_ts": 1687757990401143,
"id": "6495cbba1e19c8216a5e5de0",
"_lic": "5d3b5ff00a0a7f30b695afe3",
"online": true,
"time": 1687757990401,
"_bot": false,
"payload": "64.22",
"_gid": "64632961858b98a72bb922ef",
"topic": "base/state/humidity",
"temperature": 27.64,
"_id": "649923cac6849e1137383547"
}, {
"_org": "64632961858b98a72bb922ef",
"_mid": "64785fff858b98a72bb94534",
"_oid": "6495cbba1e19c8216a5e5de0",
"humidity": 64.42,
"_ts": 1687757996686076,
"id": "6495cbba1e19c8216a5e5de0",
"_lic": "5d3b5ff00a0a7f30b695afe3",
"online": true,
"time": 1687757996686,
"_bot": false,
"payload": "64.42",
"_gid": "64632961858b98a72bb922ef",
"topic": "base/state/humidity",
"temperature": 27.64,
"_id": "649923cac6849e1137383547"
}, {
"_org": "64632961858b98a72bb922ef",
"_mid": "64785fff858b98a72bb94534",
"_oid": "6495cbba1e19c8216a5e5de0",
"humidity": 64.35,
"_ts": 1687758005919560,
"id": "6495cbba1e19c8216a5e5de0",
"_lic": "5d3b5ff00a0a7f30b695afe3",
"online": true,
"time": 1687758005919,
"_bot": false,
"payload": "64.35",
"_gid": "64632961858b98a72bb922ef",
"topic": "base/state/humidity",
"temperature": 27.64,
"_id": "649923cac6849e1137383547"
}, {
"_org": "64632961858b98a72bb922ef",
"_mid": "64785fff858b98a72bb94534",
"_oid": "6495cbba1e19c8216a5e5de0",
"humidity": 64.35,
"_ts": 1687758008317052,
"id": "6495cbba1e19c8216a5e5de0",
"_lic": "5d3b5ff00a0a7f30b695afe3",

"online": true,
"time": 1687758008317,
"_bot": false,
"payload": "27.73",
"_gid": "64632961858b98a72bb922ef",
"topic": "base/state/temperature",
"temperature": 27.73,
"_id": "649923cac6849e1137383547"
}, {
"_org": "64632961858b98a72bb922ef",
"_mid": "64785fff858b98a72bb94534",
"_oid": "6495cbba1e19c8216a5e5de0",
"humidity": 64.32,
"_ts": 1687758009699033,
"id": "6495cbba1e19c8216a5e5de0",
"_lic": "5d3b5ff00a0a7f30b695afe3",
"online": true,
"time": 1687758009699,
"_bot": false,
"payload": "64.32",
"_gid": "64632961858b98a72bb922ef",
"topic": "base/state/humidity",
"temperature": 27.73,
"_id": "649923cac6849e1137383547"
}, {
"_org": "64632961858b98a72bb922ef",
"_mid": "64785fff858b98a72bb94534",
"_oid": "6495cbba1e19c8216a5e5de0",
"humidity": 64.2,
"_ts": 1687758015169781,
"id": "6495cbba1e19c8216a5e5de0",
"_lic": "5d3b5ff00a0a7f30b695afe3",
"online": true,
"time": 1687758015169,
"_bot": false,
"payload": "64.20",
"_gid": "64632961858b98a72bb922ef",
"topic": "base/state/humidity",
"temperature": 27.73,
"_id": "649923cac6849e1137383547"
}, {
"_org": "64632961858b98a72bb922ef",
"_mid": "64785fff858b98a72bb94534",
"_oid": "6495cbba1e19c8216a5e5de0",
"humidity": 64.1,
"_ts": 1687758020818809,
"id": "6495cbba1e19c8216a5e5de0",
"_lic": "5d3b5ff00a0a7f30b695afe3",
"online": true,
"time": 1687758020818,
"_bot": false,
"payload": "64.10",

"_gid": "64632961858b98a72bb922ef",
"topic": "base/state/humidity",
"temperature": 27.73,
"_id": "649923cac6849e1137383547"
}, {
"_org": "64632961858b98a72bb922ef",
"_mid": "64785fff858b98a72bb94534",
"_oid": "6495cbba1e19c8216a5e5de0",
"humidity": 64.15,
"_ts": 1687758027169890,
"id": "6495cbba1e19c8216a5e5de0",
"_lic": "5d3b5ff00a0a7f30b695afe3",
"online": true,
"time": 1687758027169,
"_bot": false,
"payload": "64.15",
"_gid": "64632961858b98a72bb922ef",
"topic": "base/state/humidity",
"temperature": 27.73,
"_id": "649923cac6849e1137383547"
}, {
"_org": "64632961858b98a72bb922ef",
"_mid": "64785fff858b98a72bb94534",
"_oid": "6495cbba1e19c8216a5e5de0",
"humidity": 64.03,
"_ts": 1687758033550415,
"id": "6495cbba1e19c8216a5e5de0",
"_lic": "5d3b5ff00a0a7f30b695afe3",
"online": true,
"time": 1687758033550,
"_bot": false,
"payload": "64.03",
"_gid": "64632961858b98a72bb922ef",
"topic": "base/state/humidity",
"temperature": 27.73,
"_id": "649923cac6849e1137383547"
}, {
"_org": "64632961858b98a72bb922ef",
"_mid": "64785fff858b98a72bb94534",
"_oid": "6495cbba1e19c8216a5e5de0",
"humidity": 63.99,
"_ts": 1687758038809783,
"id": "6495cbba1e19c8216a5e5de0",
"_lic": "5d3b5ff00a0a7f30b695afe3",
"online": true,
"time": 1687758038809,
"_bot": false,
"payload": "63.99",
"_gid": "64632961858b98a72bb922ef",
"topic": "base/state/humidity",
"temperature": 27.73,
"_id": "649923cac6849e1137383547"

PROJECT REPORT

Questions? Contact us.

Mahammad Arif Kotwal
arifm4348@gmail.com
8123150453