

UNIVERSITY OF MORATUWA



Smart Motor Controller

EN3250 Internet of Things

GROUP: 04

GROUP MEMBERS:

G.Ganeshaaraj	170180N
T.Luheerathan	170351P
C.Shobanapriyan	170581U
S.Sivaluxan	170596U

Contents

1. Overview
2. System Architecture
3. Communication protocols used.
 - I. Wi-Fi
 - II. MQTT Protocol
 - III. HTTP Protocol
4. Methodology and System Components
 - I. Real-Time Data Acquisition
 - II. Data Logging and Storing
 - III. Controlling with Hardware
5. Conclusion

Smart Motor Controller

1) Overview

Agriculture is the key component of the economy for the countries like Sri Lanka. Identifying the problems in the traditional agriculture and trying to solve them with the help of technology will lead to yield a lot of crops. Here we have identified an agriculture problem and we have developed a system to overcome the some identified problems in agriculture.

In traditional system, what the farmers do is, they observe their farming land and based on their previous experiences, they decide to water the crops. Farmers need to be in the land to switch on and off the motor. This procedure has following disadvantages

- Possibility for inefficient management in a vast field area
- Farmers should always physically present near the field /no remote access
- Inexperienced farmers will struggle to make decisions on watering the crops.

Our systems capabilities are,

- System provides updates on the soil moisture, soil temperature, humidity, atmosphere temperature for every 30 minutes of period.
- User can monitor the current parameters or the trend of these parameters thorough the dashboard.
- User can turn on the motor at anytime from anywhere based on these parameters. The controls given in the dashboard are made it possible.
- We cannot expect the user to monitor the dashboard all the time and turn on the motor. Users, who do not often monitor the dashboard, can get alert messages when these parameters are exceeded above the user set threshold.
- Threshold levels can be set based on current parameters, trend of the parameters and the crop, which is being cultivated. Based on the alert message user can decide on whether to turn on the motor or not.
- An inexperienced farmer can take watering decision based on above parameters.

2) System Architecture

In the system OpenWeather API and Getambee API are used for acquiring soil moisture, humidity, temperature, and other weather parameters in a real time basis. Data from above APIs are stored in a firebase real-time database for every 30 minutes interval and can be used for further analysis. Using a highly secured and user authenticable pin enabled dashboard, system offers the opportunity to monitor the real time data, monitor the trend of each respective data and control motor. The PIN can be configured by user, and it is also stored in firebase. Vonage API is used to send alert messages to the user. Mosquito MQTT server is used to communicate (sending control information) with the NodeMCU. A HTTP server running on the NodeMCU and a phone is used to configure a Wi-Fi connection for NodeMCU. Following picture depicts the architecture. Upcoming sections explain the design in detail.

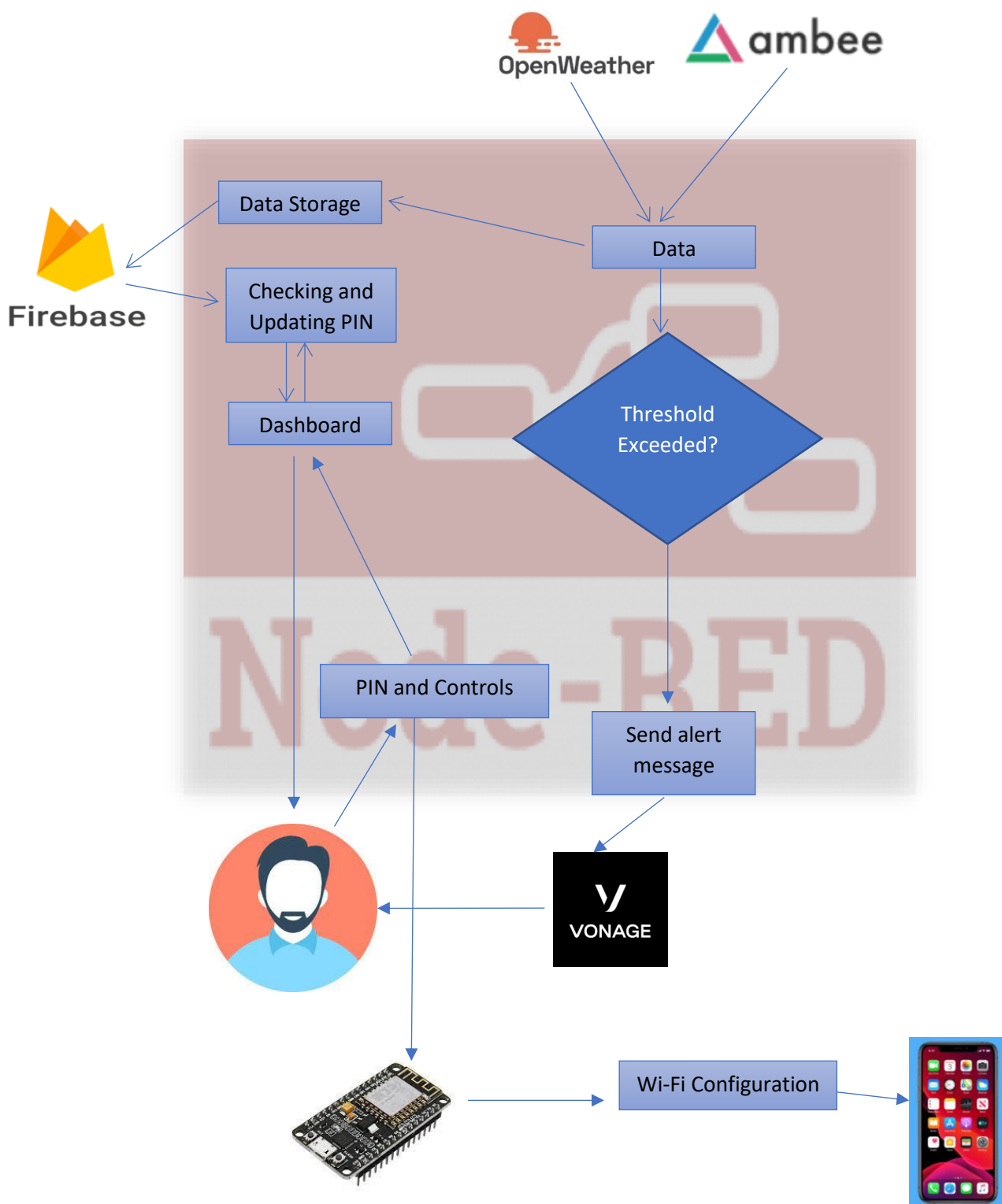


Figure 1 - System Architecture

3) Communication Protocol Used

I) Wi-Fi

Wi-Fi is widely used wireless technology which connects users within a several meters. Even though Wi-Fi has not been particularly designed for IoT, there are recent innovation which supports IoT. NodeMCU supports 802.11b, 802.11b and 802.11n Wi-Fi standards. In our projects, Wi-Fi is used to connect the NodeMCU with Node-Red deployed in the IBM cloud and transfer the user interaction like when to be switched on the motor and how long to be switched on the motor. The Wi-Fi connection is turned off while the motor is working and the NodeMCU is set in deepsleep state for power saving. Wi-Fi is used to configure the Wi-Fi credentials on NodeMCU. Here NodeMCU works as access point and phone works as a client. Both make a local network. HTTP server running on the NodeMCU is sent to the phone. SSID and password are sent to the NodeMCU through the http page from phone. Then NodeMCU is connected to specified SSID as a client.

II) MQTT Protocol

Message Queue Telemetry Transport (MQTT) is a simple messaging protocol, designed for constrained devices with low bandwidth. So, it is the perfect solution for Internet of Things applications. MQTT allows to send the commands to control outputs, read and publish data from sensor nodes and much more. Therefore, it makes it easy to establish a communication between multiple devices. In our project, we used MQTT to send the controls and deepsleep time form Node-red.

III) HTTP Protocol

The Hypertext Transfer Protocol, shortly HTTP, is an application layer protocol which is widely used in internet to transfer HTML, pictures, audio, video files to the client side from a host server. This protocol is rarely used in Internet of Things due to the high overhead. Even though this protocol is rarely used, we can identify few situations where HTTP serve much more than other protocol. In our project, we are using HTTP to configure Wi-Fi credentials on NodeMCU dynamically. A phone/device connected with NodeMCU is enough to do this.

4) Methodology and System Components

I) Real-Time Data Acquisition

This project needed below stated data in real time.

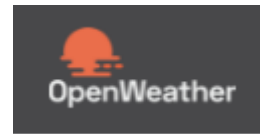
- Atmospheric temperature
- Atmospheric Humidity
- Soil Temperature
- Soil Moisture

These real time data were acquired with the help open-source APIs. They are.

- Atmospheric temperature –OpenWeatherMap API
- Atmospheric Humidity- OpenWeatherMap API
- Soil Temperature- Ambee API
- Soil Moisture- Ambee API

OpenWeatherMap API

OpenWeatherMap is a company owned by OpenWeather Ltd, which provides historical, real time and forecasts global weather data for any given geographical location via online APIs.



The reasons for choosing this API are stated below,

Figure 2 – Open weather logo

1. Availability of real time temperature and humidity data
2. Easy integration with node-red
3. They are offering various pricing schemes based on the calls requested per minute. In our case we chose the free of charge package which offers 60 calls per minute. This particular volume of calls was adequate for our application.

Ambee API

Ambee is also another open-source API, which provides real time data on weather, soil, and pollen and air quality for a given geographical location.



The reasons for choosing this API are stated below,

Figure 3 – Ambee logo

1. Availability of real time soil temperature and soil moisture data
2. They are offering various pricing schemes based on the calls requested per minute. In our case we chose the free of charge package which offers 150 calls per day. This particular volume of calls was adequate for our demonstration but not sufficient for real time implementation.
3. This API is comparatively lot harder to integrate with node-red, but we accomplished this task with the help of curl request.

Node-RED

Node-RED is a programming tool which combines hardware components and API/Online services. This tool provides a user-friendly browser-based edition. For our project we hosted node-RED in IBM cloud. We used Node-RED to accomplish the following functions and these functions are elaborated in following sections.

1. Built an interactive web dashboard which offers.
 - a. User friendly access to real time data.
 - b. Graphical features to interpret the data more efficiently.
 - c. Security features which enable safe access and privacy
 - d. Access to change the location of interest.
 - e. Automatic message generation and transmission, from node-RED to user's smart phone during certain conditions.
- a. User friendly access to real time data

Our web dashboard contains multiple tabs to enhance user friendliness, to effectively view data and control the motor. The tabs and their respective function are described below.

- [1] Home: - Authentication is required by user to protect the privacy
- [2] MyFarm-Weather: – Displays gauges and graphs to visually represent the temperature and humidity of the given location.

- [3] MyFarm-Soil: – Displays gauges and graphs to visually represent the soil temperature and soil moisture of the given location.
- [4] Account-settings: – Give the user the option to change password, location, and thresholds.
- [5] Control panel: – Enables user to control motor.

Some of the examples are given below,

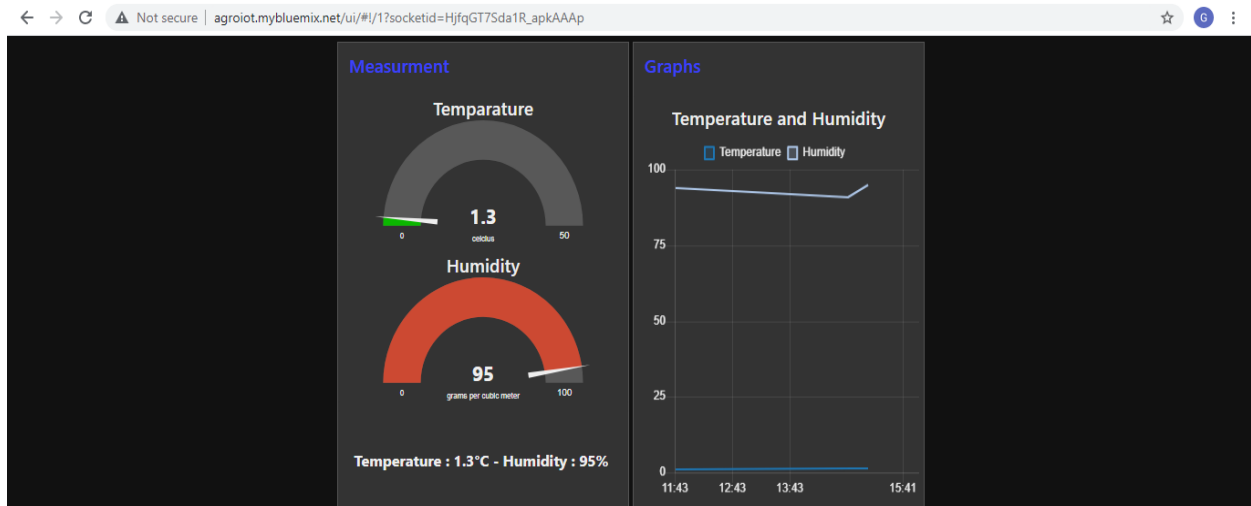


Figure 4 – User interface for MyFarm-Weather Tab

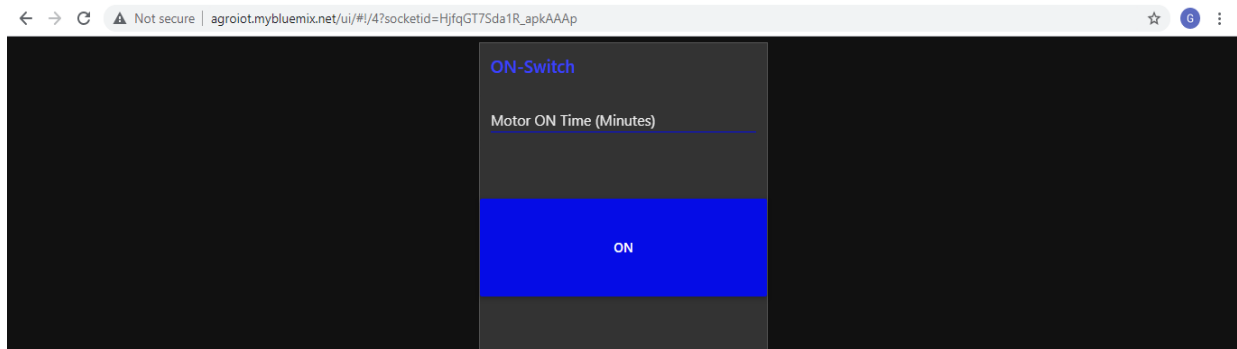


Figure 5 – User interface for Control-Panel Tab

- b. Graphical features to interpret the data more efficiently.

We used gauges and graphs in the MyFarm-Weather and MyFarm-Soil tabs. Description of gauge and graphs is given below.

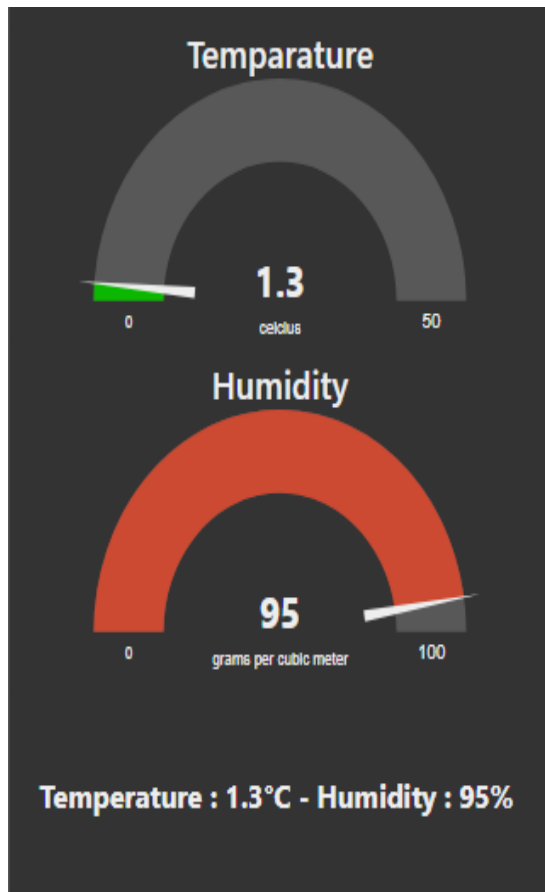


Figure 6- Gauge to display temperature and humidity.

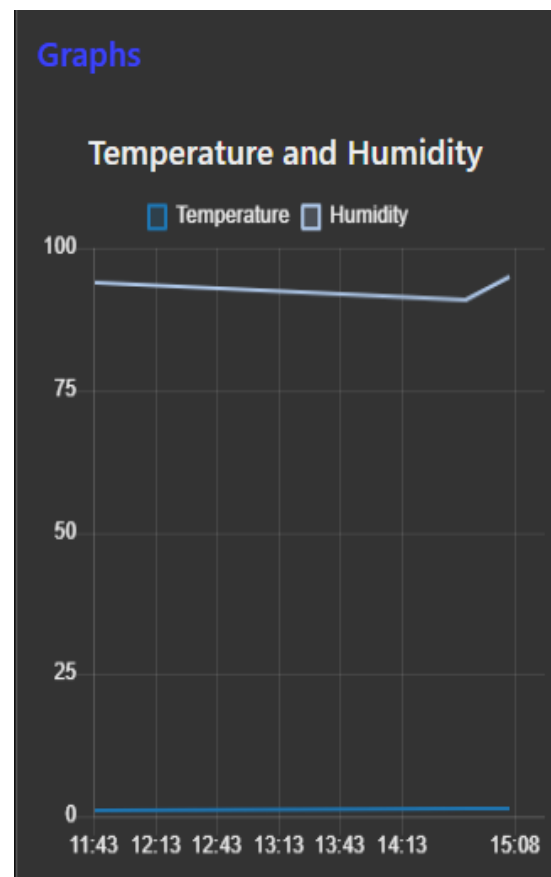


Figure 6- Graph to display temperature and humidity.

c. Security features which enable safe access and privacy

In addition to the soil and weather parameters, a pin is also stored in the real-time database. It enables the user authentication in the dashboard. Whenever user tries to load the dashboard, he should authenticate his identity. A default PIN is provided when user buys the product. The user should login to the system and configure a PIN and the configured PIN is stored in the real-time database. When user tries to access dashboard, the PIN provided by him is checked with the PIN in the firebase and allow him to access. It prevents unauthorised persons accessing the system.

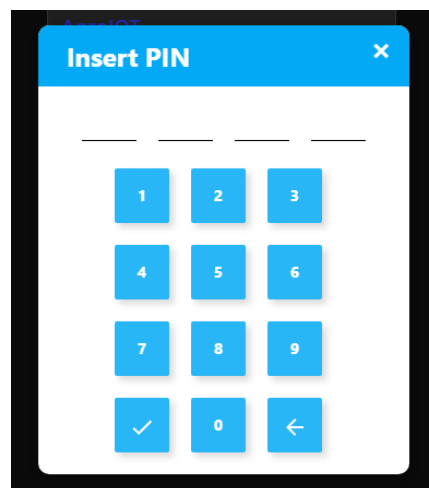
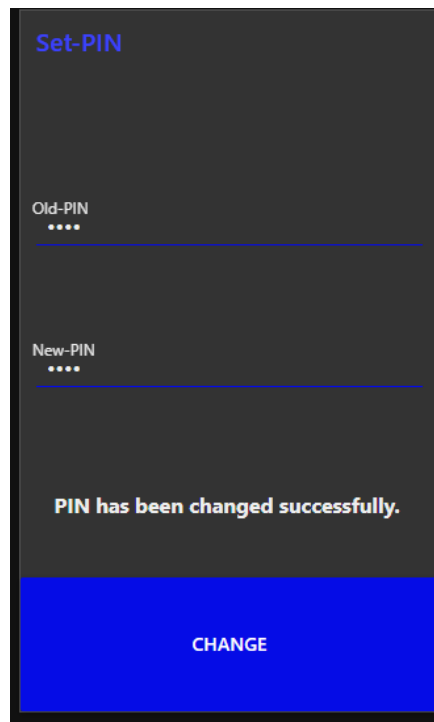


Figure 7 –Dialog box to type the password.

A mobile application screen titled "Set-PIN" in blue text. It features two input fields: "Old-PIN" and "New-PIN", each with four dots representing masked characters. Below these fields, a message states "PIN has been changed successfully." in white text. At the bottom, there is a large blue button with the word "CHANGE" in white capital letters.

Set-PIN

Old-PIN
....

New-PIN
....

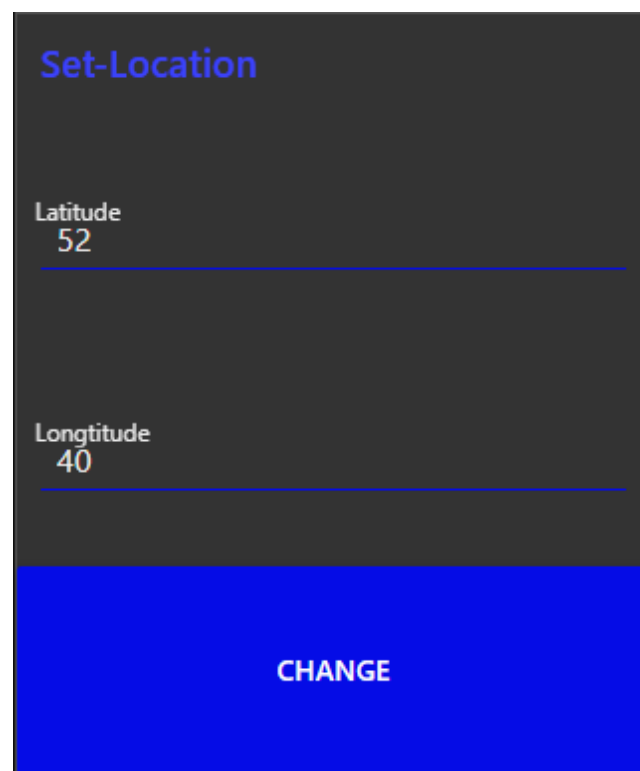
PIN has been changed successfully.

CHANGE

Figure 8– PIN configuration window

- d. Access to change the location of interest.

User is given the option to change the location of interest by changing the value of latitude and longitude. These changes can be made in the Account-settings tab.

A mobile application screen titled "Set-Location" in blue text. It features two input fields: "Latitude" with the value "52" and "Longitude" with the value "40". Below these fields, there is a large blue button with the word "CHANGE" in white capital letters.

Set-Location

Latitude
52

Longitude
40

CHANGE

Figure 9 – Interface to change location settings.

- e. Automatic message generation and transmission, from node-RED to user's smart phone during certain conditions.

By considering the water and the current consumption for pumping the water, we gave manual access for user to control the motor manually. The user can observe the current weather parameters (atmosphere temperature, atmosphere humidity) and soil parameters (soil temperature, soil moisture) through the dashboard and according to that user can turn on the motor manually. But there is a possibility for user may not have time to continuously observe the dashboard. As a solution we integrated a user alerting feature to our application. Through that user can set a threshold value for those 4 parameters. Then if those parameters met certain condition an alert message will send to the user mobile phone. Then user can look the dashboard at the time and turn on the motor manually. We used Vonage API to achieve this task.

Vonage API

Vonage is one of the most popular communication APIs providers in the world. Using this API, customers make interconnection between the IoT devices and their mobile devices. We have listed down some of the popular APIs provided by them,

1. Communication API
2. Authentication API
3. Management API
4. Programmable Number

The reason for choosing this API is stated below.

1. This API is readily available in the node-red called node-red-contrib-nexmo
2. Able to customize the mobile number.

The below stated steps are followed to configure the user alerting feature in node-red.

1. Created a UI-tab to get the threshold value from the user.
2. Write function using those parameters.
3. Create the Vonage account and get the API key.
4. Install node-red-contrib-nexmo pallet.
5. Configure the send SMS node and set the mobile number.

II) Data logging and storing

In our case the real time data was gathered with the help of the Open-weather API and Ambee API. The real time data which was acquired with the help of above stated APIs needed to be store for further analysis to get useful insights to improve the farming.

We tried Firebase as our initial step to store data. Afterwards we shifted to Google Cloud based Google sheets to store the data because we had difficulties in using node-red palletes. Parallely, we were exploring on other possibilities which could aid us to use firebase. Finally, we were able to incorporate both firebase and Google sheets in the project.

So In this project we use Google Cloud as cloud service provider and using that we are storing not only the data but also the date and time in a real time basis. We used Google cloud API and Google Sheet API to acquire the service.

Google cloud API and Google Sheet API

The Google Drive API allows user to create apps that leverage Google Drive Cloud storage. This REST API that allows us to leverage the Google Cloud service is incorporated within the app.

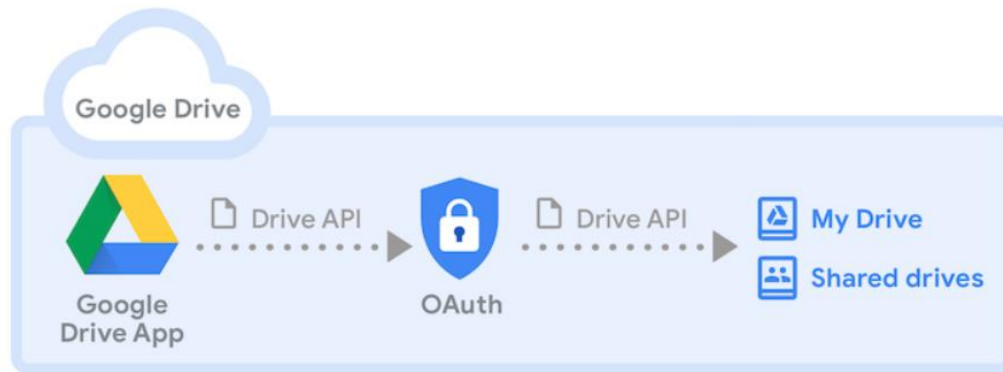


Figure 10 – Google Drive integration

The google sheet API is a RESTful interface that lets you read and modify a spread sheet data. The common use of this API includes the following tasks.

1. Create spread sheets
2. Read and write spread sheet cell values
3. Update spread sheets
4. Manage connected sheets

The reason for choosing this API is stated below

1. This API is readily available in the node-red called node-red-contrib-google sheets.
2. Faced problem in configuring the node-red-contrib-firebase palette to store real time data.
3. Satisfied the requirement of our use case.

Following are the steps we followed to configure our own google cloud.

1. Go to google cloud console.
2. Create the project.
3. Go to API & Services dashboard and clicked on ENABLE APIS AND SERVICES tab.
4. Enable the Google Drive API and Google Sheet API.
5. Clicked on the credentials tab and create CREDENTIALS as service account.
6. Get in to google drive and create new sheets to store the data.
7. Clicked on the share button and give access to the client email which is available in the created credentials.

Following are the steps to enable data logging and storing feature in node-red.

1. Install node-red-contrib-google-sheets pallet.
2. Add the credential JSON of created google service account.
3. Give the spread sheet ID and set the sheet name and shell name.
4. Write function to extract the data from the APIs and input to the Google Sheet node.

Firebase API

As said earlier firebase real time database can be used for data logging. We initially tried with multiple palettes which supports firebase in node-red. All of them ended up in failure. Then we used the 'HTTP request' palette to get the stored data and store the data in real-time.

Initially a real-time database was created on the firebase and the authentication for reading and writing also set out. Then, we created a 'HTTP request' palette to get and store the real-time data. A GET request with URL along with authentication key is passed to read data. A POST request is passed to store data. In POST request, a payload which should be stored (in our case, real-time data from APIs) is passed along with URL. The basic structure created for getting and storing data can be found here.

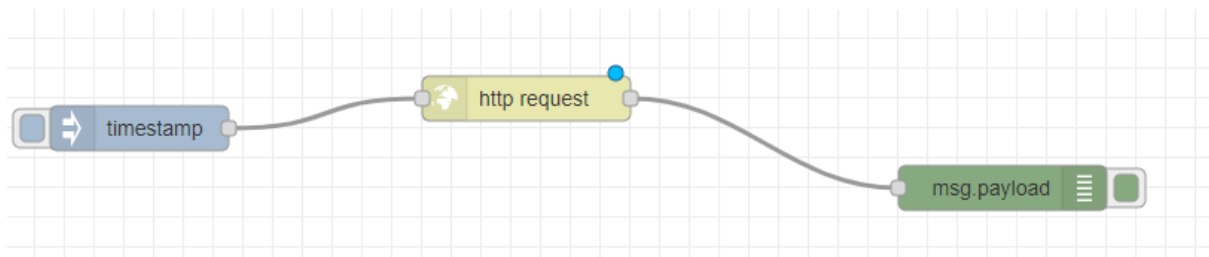


Figure 11 – GET request for reading data

The screenshot shows the 'Edit http request node' dialog box. At the top, there are 'Delete', 'Cancel', and 'Done' buttons. Below is a 'Properties' tab with a settings icon. The configuration includes:

- Method:** A dropdown menu set to 'GET'.
- URL:** A text field containing 'https://agroiote-db-316900-default-rtdb.firebaseio.c'.
- Payload:** A dropdown menu set to 'Ignore'.
- Options:** Four unchecked checkboxes: 'Enable secure (SSL/TLS) connection', 'Use authentication', 'Enable connection keep-alive', and 'Use proxy'.
- Return:** A dropdown menu set to 'a UTF-8 string'.
- Name:** A text field containing 'Name'.

At the bottom, there is an 'Enabled' checkbox which is currently checked.

Figure 12 – HTTP request palette configuration

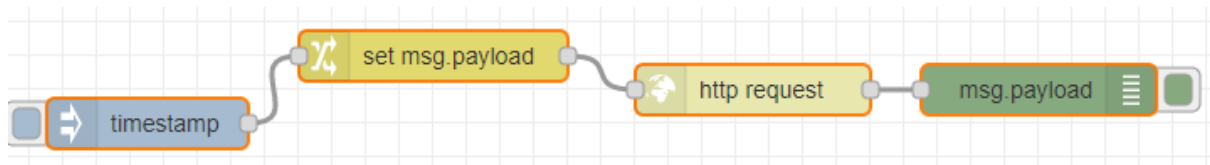


Figure 13 – POST request for writing data

Edit http request node

Delete Cancel Done

Properties

Method POST

URL <https://agroiot-d7202-default-rtdb.firebaseio.com/c>

☐ Enable secure (SSL/TLS) connection

☐ Use authentication

☐ Enable connection keep-alive

☐ Use proxy

Return a UTF-8 string

Name Name

☐ Enabled

Figure 14– HTTP request palette configuration

<https://agroiot-db-316900-default-rtdb.firebaseio.com/>

agroiot-db-316900-default-rtdb

- date
- humidity
- soil_moist
- soil_temp
- temp
- userconfig: 1000

Database location: United States (us-central1)

Figure 15- Firebase Data Fields

III) Controlling With Hardware

NodeMCU ESP8266

NodeMCU is an open-source Lua based firmware and development board which is specially used in IoT based applications. It has ESP8266 Wi-Fi SoC firmware and ESP-12 module-based hardware. NodeMCU ESP8266 has following features.

- Micro controller: Tensilica 32-bit RISC CPU Xtensa LX106
- Operating Voltage: 3.3V
- Input Voltage: 7-12V
- Digital I/O pins: 16
- Analog Input pins: 1
- UARTs: 1
- SPIs: 1
- L2Cs: 1
- Flash Memory: 4MB
- SRAM: 64KB
- CLK speed: 80 MHz

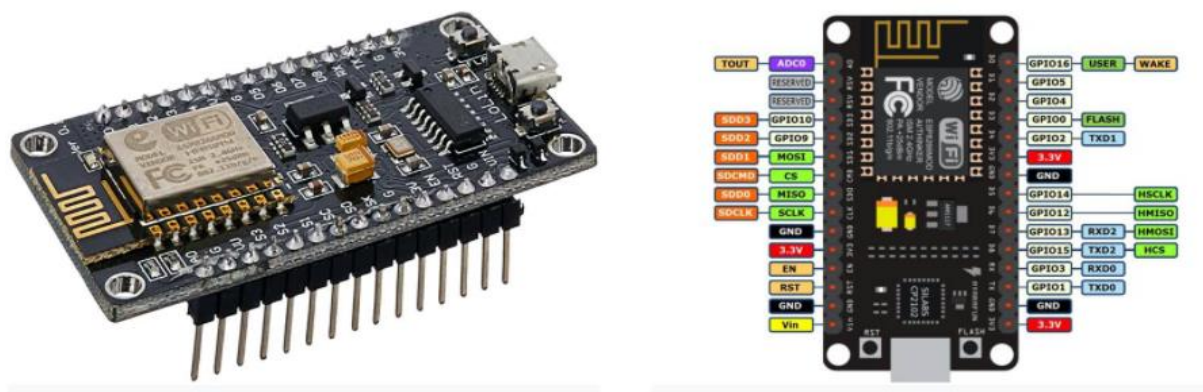


Figure 16- NodeMCU ESP8266 and Pinout

In this project, Arduino IDE was used to program NodeMCU ESP8266. We implemented following functionalities in NodeMCU ESP8266 using Arduino.

- Make Wi-Fi Connectivity for the device using Wi-FiManager library
- Make connectivity between Node-Red and NodeMCU using PubSubClient library and MQTT broker.
- Implement deep sleep mode.

Scan and connect to Wi-Fi on NodeMCU using Wi-Fi Manager

Most of the IoT devices must be connected to the internet to begin operation. We used ESP8266 Wi-Fi Manager to make preferred Wi-Fi connectivity. Wi-Fi manager function can be added to the existing program to provide an option for the users to scan and connect to any Wi-Fi network and once the connection is established the device can perform its normal function until the network connection has to be changed again.

We programmed NodeMCU to follow the following steps.

- When your ESP8266 starts up, it sets it up in Station mode and tries to connect to a previously saved Access Point.
- If this is unsuccessful (or no previous network saved) it moves the ESP into Access Point mode and spins up a DNS and WebServer (default ip 192.168.4.1).
- Connect to the newly created Access Point using any Wi-Fi enabled device through browser.
- Choose one of the access points scanned, enter password, click save.
- ESP will be connected. If not, reconnect to AP and reconfigure.
- Also change the connected Wi-Fi and restart the process using Trigger pin D1 that is connected to GND. (by giving nonzero voltage, The Wi-Fi credentials can be reset)

```
*WM: [2] Connecting as wifi client...
*WM: [3] STA static IP:
*WM: [2] setSTAConfig static ip not set, skipping
*WM: [1] Connecting to SAVED AP: LUHEE-PHONE0
*WM: [3] Using Password: 00000000
*WM: [3] WiFi station enable
*WM: [3] enableSTA PERSISTENT ON
*WM: [1] connectTimeout not set, ESP waitForResult...
*WM: [2] Connection result: WL_NO_SSID_AVAIL
*WM: [3] lastconxresult: WL_NO_SSID_AVAIL
*WM: [1] AutoConnect: FAILED
*WM: [2] Starting Config Portal
*WM: [3] WIFI station disconnect
*WM: [3] WiFi station enable
*WM: [2] Disabling STA
*WM: [2] Enabling AP
*WM: [1] StartAP with SSID: AgroIoT
*WM: [2] AP has anonymous access!
*WM: [1] SoftAP Configuration
*WM: [1] -----
*WM: [1] ssid:           AgroIoT
*WM: [1] password:
*WM: [1] ssid_len:        7
*WM: [1] channel:          1
*WM: [1] authmode:
*WM: [1] ssid_hidden:
*WM: [1] max_connection:   4
*WM: [1] country:         CN
*WM: [1] beacon_interval: 100(ms)
*WM: [1] -----
*WM: [1] AP IP address: 192.168.4.1
*WM: [3] setupConfigPortal
*WM: [1] Starting Web Portal
*WM: [3] dns server started with ip: 192.168.4.1
*WM: [2] HTTP server started
*WM: [2] WiFi Scan completed in 2182 ms
*WM: [2] Config Portal Running, blocking, waiting for clients...
.....
```

Figure 17 – Trying to connect saved AP, Establishing HTTP server when failed.

```

*WM: [1] Connecting to NEW AP: LUHEE-PHONE0
*WM: [2] Using Password: 00000000
*WM: [3] WiFi station enable
*WM: [3] enableSTA PERSISTENT ON
*WM: [1] connectTimeout not set, ESP waitForConnectResult...
*WM: [2] Connection result: WL_CONNECTED
*WM: [3] lastconxresult: WL_CONNECTED
*WM: [1] Connect to new AP [SUCCESS]
*WM: [1] Got IP Address:
*WM: [1] 192.168.7.75
*WM: [2] disconnect configportal
*WM: [2] restoring usermode STA
*WM: [2] wifi status: WL_CONNECTED
*WM: [2] wifi mode: STA
*WM: [1] config portal exiting
connected... :)
*WM: [3] unloading
Attempting MQTT connection...connected

```

Figure 18 – Connected to the user given SSID and MQTT establishment

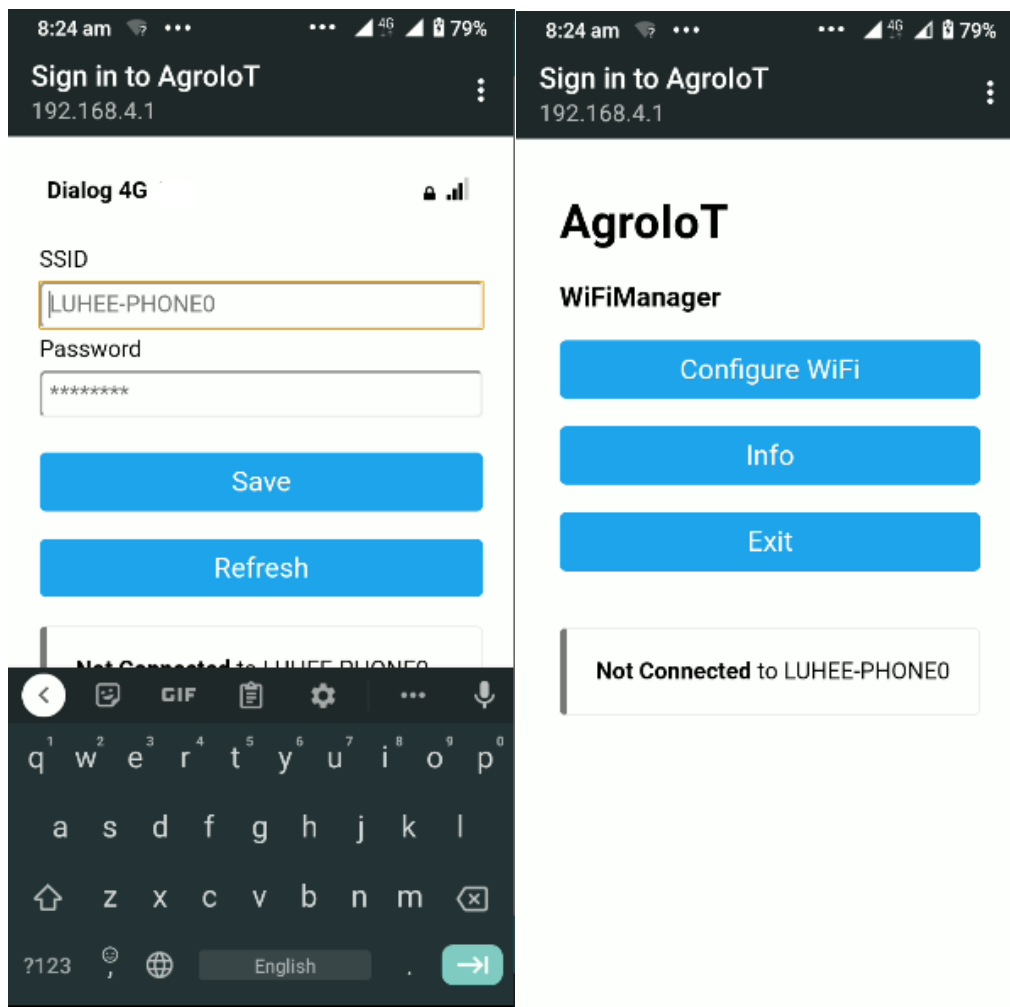


Figure 19 – HTTP portal to configure Wi-Fi-Connection.

Connectivity between Node-Red and NodeMCU using PubSubClient and MQTT.

In our project we used MQTT protocol to make connectivity between Node-Red and NodeMCU. PubSubClient library was used to provide client for doing publish and subscribe messaging with a server that supports MQTT.

Mosquitto MQTT – Eclipse mosquitto open-source message broker that implements MQTT protocol. Mosquitto provides lightweight method of carrying out messaging using publish/subscribe model. It is designed for low bandwidth and low power, high latency, or unreliable networks.

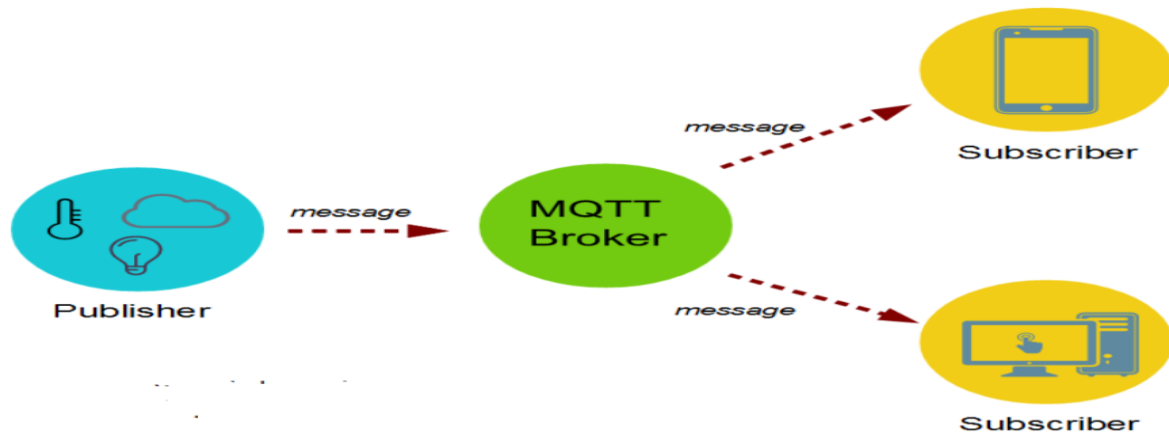


Figure 20- MQTT Architecture

In our project, we subscribe the topic “Agrolot/IN” with the messages that comes from Node-RED. Message in that topic includes the status of the motor (ON/OFF) and the duration of the operation. Switching on the motor will be executed by NodeMCU.

Implement deep sleep mode for power saving.

ESP8266 module operates in following modes.

1. Active mode – In this mode, chip is powered on, and chip can receive and transmit data. Most power consuming mode.
2. Modem sleep mode - In this mode, the CPU is operational, and the Wi-Fi radios are disabled. It makes the Wi-Fi Modem circuit to turn off while connected with the Wi-Fi AP with no data transmission to optimize power consumption.
3. Light sleep mode - In this mode, the CPU and all peripherals are paused. Any wake-up will wake up the chip. Without data transmission, the Wi-Fi Modem circuit can be turned off and CPU suspended to save power consumption.
4. Deep sleep mode - In this mode only the RTC is functional, and all other components of the chip are powered off. Less power consuming mode compared to other three modes.

In our project we used deep sleep mode. We executed the instructions of the deep sleep mode after receiving message “ON” with the topic “AgroIoT/IN” from Node-Red. NodeMCU will sleep for the certain amount of time which was received from Node-Red with the “ON” message and with the topic “AgroIoT/IN”. Once the sleep is over NodeMCU connects to the same Wi-Fi which was configured earlier and starts functioning again. Deep sleep mode is very less power consuming mode. And the current requirement is only 20 micro amperes.

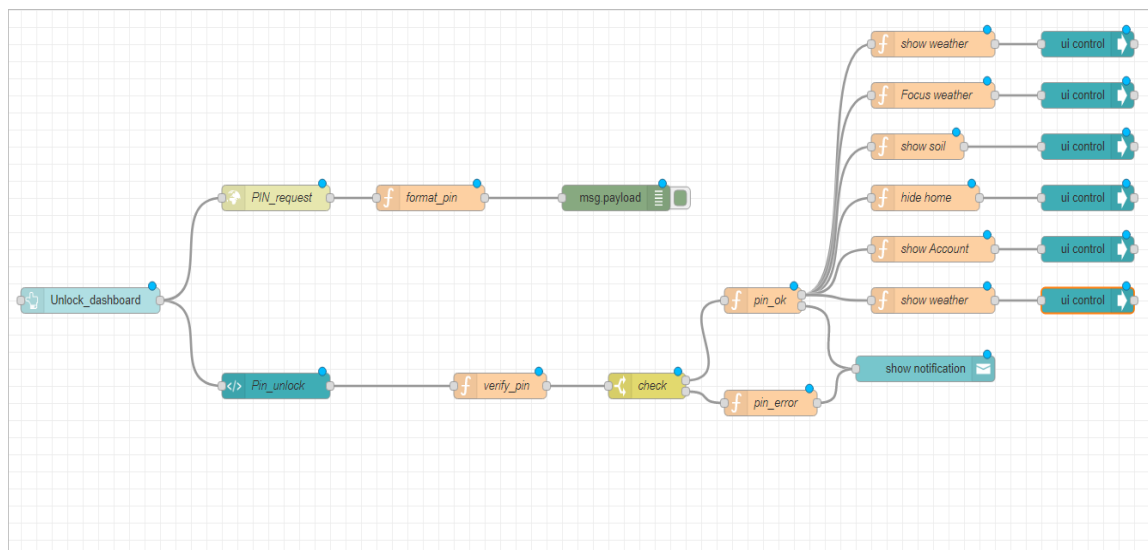
5) Conclusion

Agriculture is the backbone of our country. So, supporting agriculture with the technology innovation leads a good economy. The system prototyped here can reduce the overload of a farmer to a certain extent and motivate inexperienced farmers to involve themselves in agriculture. Agriculture based sensors yield an accurate result in comparison to the data we collected from open source APIs. Due to prevailing Covid-19 situation, we were unable to carry out the project with these sensors. Anyhow, as engineering students we are happy to support agriculture.

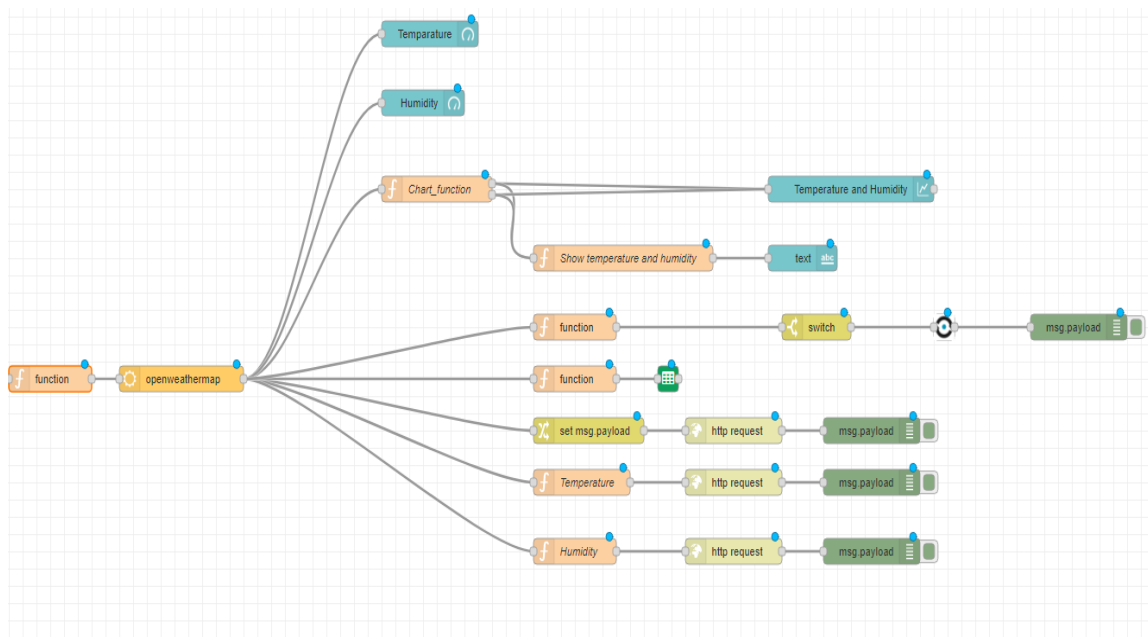
ANNEX

I. Node-red Flows

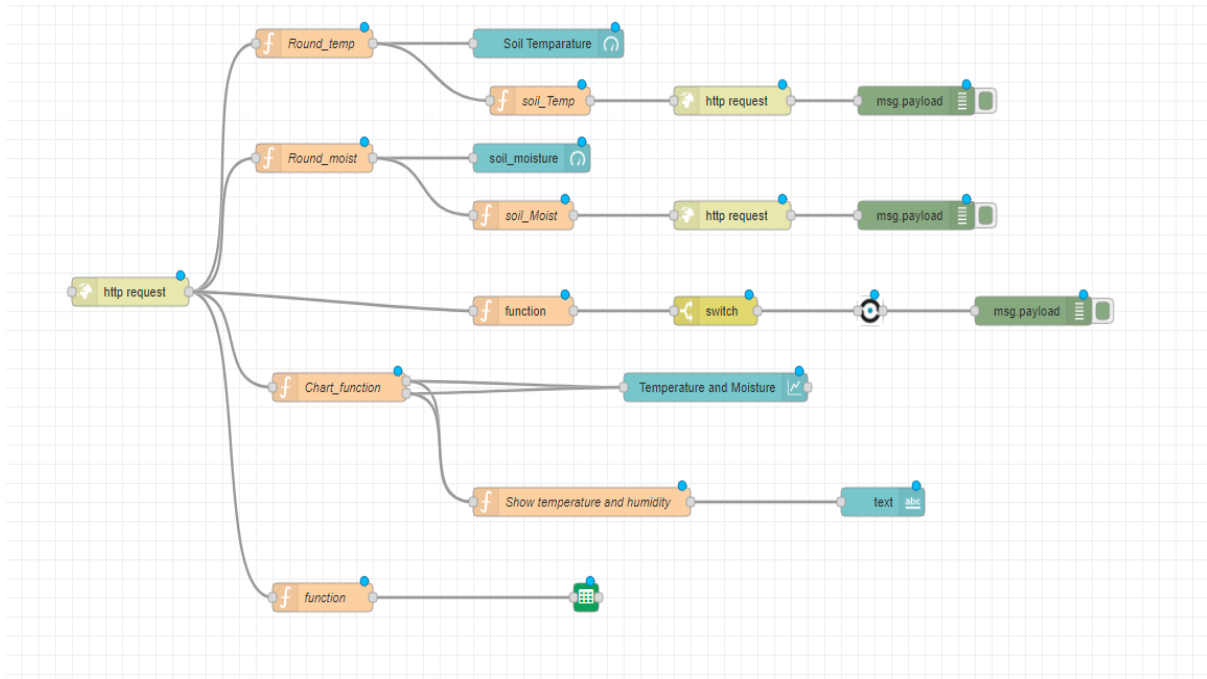
a. Home tab



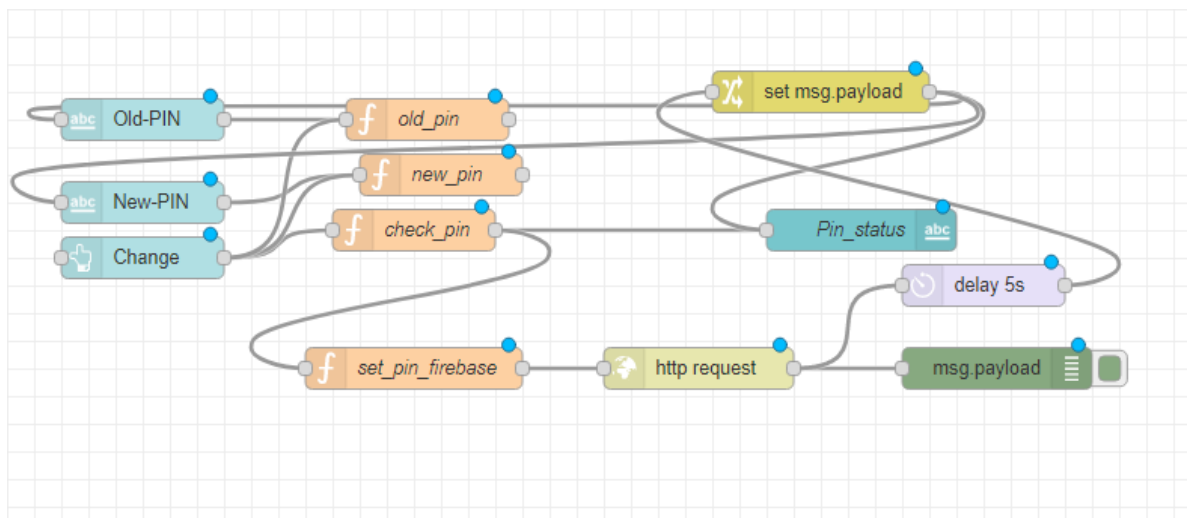
b. MyFarm-Weather tab



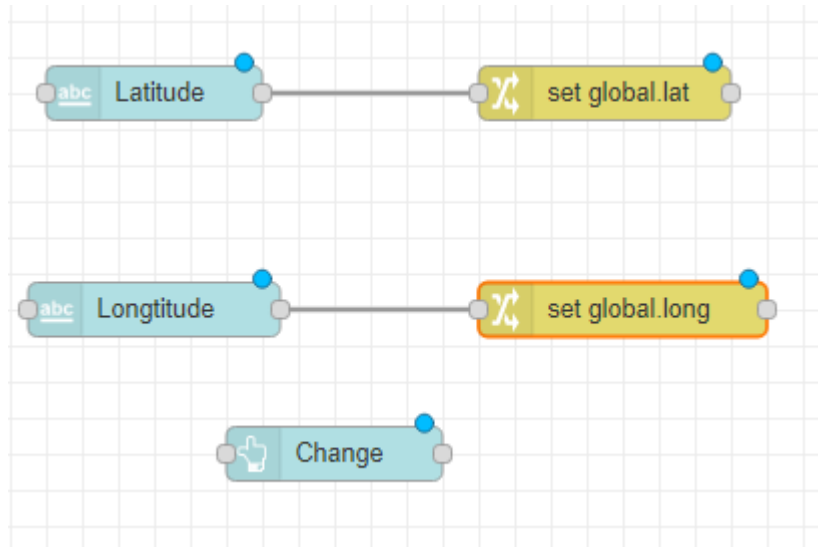
c. MyFarm-Soil tab



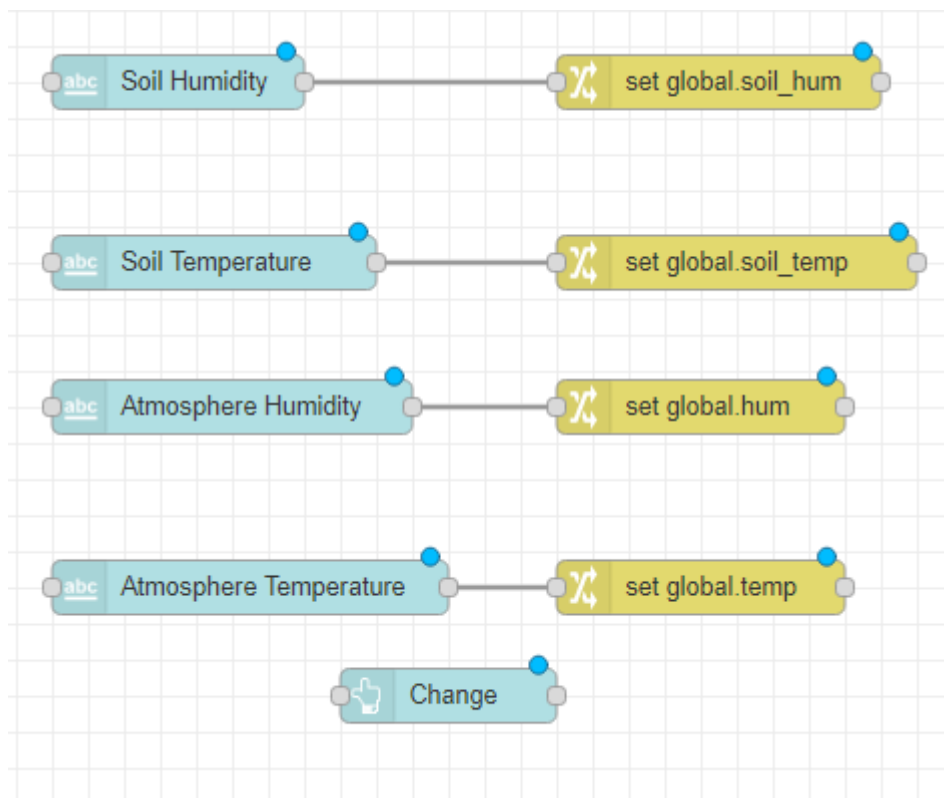
d. Account-setting tab – Change pin group



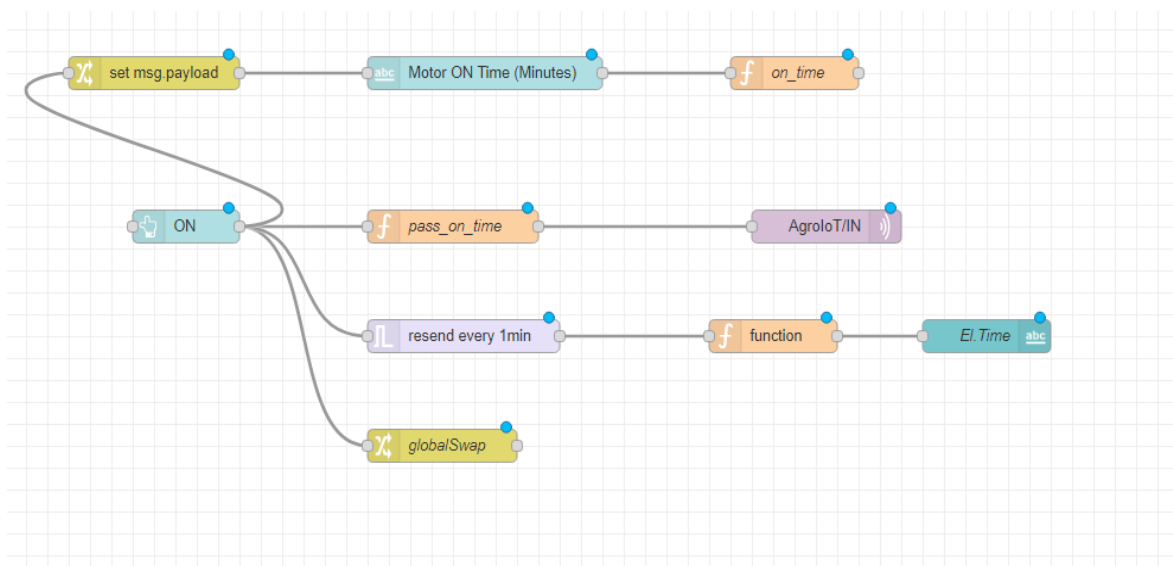
e. Account-setting tab – Change location group



f. Account-setting tab – Change threshold group



g. Control panel tab



II. Esp-8266 Code – MQTT, Deepsleep, Wi-Fi connection and motor turning on

```

#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <string.h>
#include <DNSServer.h>
#include <ESP8266WebServer.h>
#include <WiFiManager.h>           //https://github.com/tzapu/WiFiManager

const char* mqtt_server = "test.mosquitto.org";
#define trigger D1 //to change to another wifi
WiFiClient espClient;
PubSubClient client(espClient);

void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic); //to print the topic received from node red
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]); //to print the payload received from node red
  }
  Serial.println();

  // Switch on the LED if received ON to indicate switch on Motor

```

```

if ((char)payload[0] == 'O' && (char)payload[1] == 'N' ) {
    char *buff=(char *)calloc(length-3,sizeof(char));
    for (int j = 3; j < length; j++)
    {
        *(buff+j-3)=*(payload+j);
    }
    int deep_t =(String(buff)).toInt();    // calculte the received time in min
utes
    Serial.print("deep sleep time is:");
    Serial.print(deep_t);
    Serial.println(" minutes");
    int value_micro_second=deep_t*60000000;    // convert time (minutes to seco
nds)
    digitalWrite(BUILTIN_LED, LOW);    // Turn the LED on
    delay(2000);

    ESP.deepSleep(value_micro_second); //deep sleep for time value_micro_secon
d
} else {
    digitalWrite(BUILTIN_LED, HIGH);    // Turn the LED off
}
}

void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Create a random client ID
        String clientId = "ESP8266Client-";
        clientId += String(random(0xffff), HEX);
        // Attempt to connect
        if (client.connect(clientId.c_str())) {
            Serial.println("connected");

            // subscribe the topic AgroIoT/IN
            client.subscribe("AgroIoT/IN");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}
}

```

```

void setup() {
  // put your setup code here, to run once:
  pinMode(BUILTIN_LED, OUTPUT);    // Initialize the BUILTIN_LED pin as an output
  Serial.begin(115200);
  pinMode(trigger, INPUT);
  //to connect wifi
  WiFiManager wifiManager;

  //wifiManager.resetSettings();

  wifiManager.autoConnect("AgroIoT");

  //if the connection is successful, you will get here.
  Serial.println("connected... :");
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}

void loop() {
  // put your main code here, to run repeatedly:
  // to change the wifi
  if(digitalRead(trigger) == HIGH){
    WiFiManager wifiManager;

    wifiManager.resetSettings();

    wifiManager.autoConnect("AgroIoT");

    //if the connection is successful, you will get here.
    Serial.println("connected... :");
  }
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
}

```

Link to access dashboard - <http://agroiot.mybluemix.net/ui/>

PIN - 1000