

Internet of Things (IoT)

IoT Coursework Project

Hamza Noweder

Lecturer: Sean Sturley

School of Engineering and Computing
University of the West of Scotland

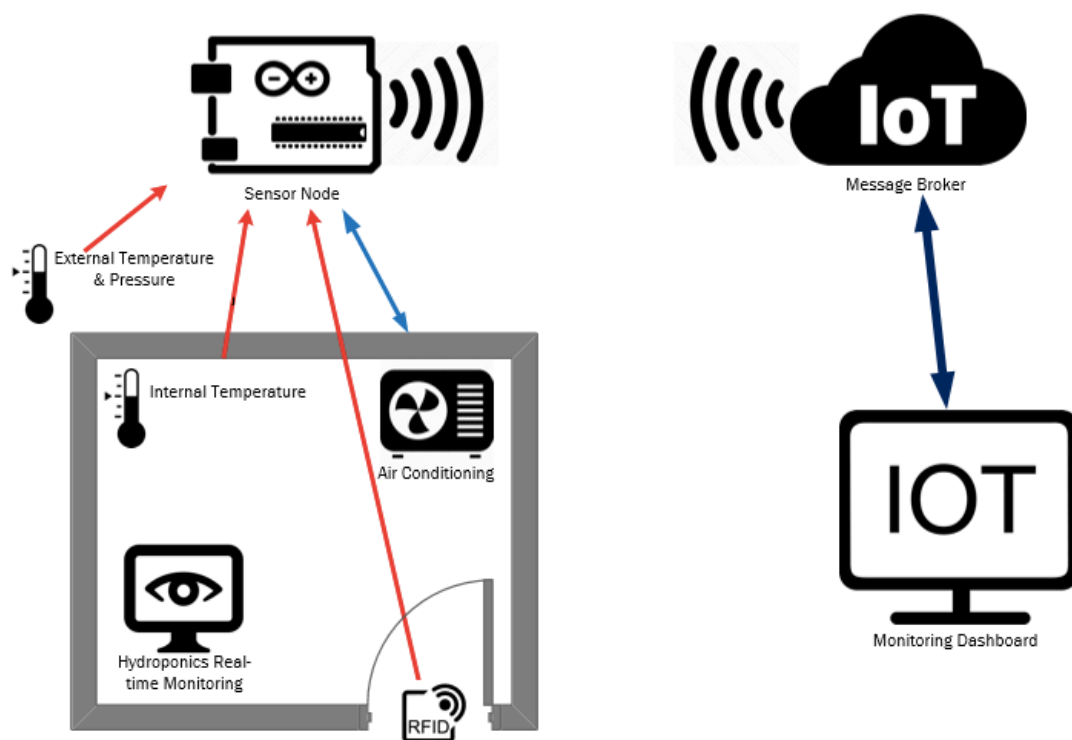
April 12, 2018

Table of Contents

Chapter 1:	Introduction	3
Chapter 2:	Arduino Setup	4
2.1	Arduino HW Components	4
2.2	Arduino HW Schematic Diagram	5
2.3	Arduino SW Setup	6
Chapter 3:	Virtual Machine Setup	11
3.1	Operating System	11
3.2	Wi-Fi Connectivity	11
3.3	Installing Nginx HTTP Web Server	12
3.4	Installing Node.js and npm	12
3.5	Installing Node-RED	13
3.6	Installing Node-RED Dashboard	14
3.7	Installing Mosquitto (MQTT Broker)	14
Chapter 4:	Solution Customization	15
4.1	Node-RED Flow Customized Design	16
4.2	Node-RED Dashboard (User Interface)	16
4.3	Monitoring Temperature and Humidity	17
4.4	Monitoring RFID	18
4.5	A/C Auto Balancing Mode	19
4.6	A/C Function Node	20
4.7	A/C Remote Switch	21
4.8	A/C Current Status	22
Appendix	23

Chapter 1: Introduction

This project requires to design and build a POC that takes two separate temperature measurements, records the use of RFID tags used to gain access to this equipment room, and monitor and manage an air conditioning, as shown the below high level diagram.



The detailed requirements of this project are listed below:

1. Record the temperature in the growing area.
2. Record the temperature in the equipment room.
3. Record entry to the equipment room.
4. If the temperature difference is more than 2°Centigrade then either heating or cooling of the equipment room is performed.
5. Report and record the results to a suitable IoT management system.
6. The IoT management system (Dashboard) shows the status of the air conditioning system and can remotely override the POC hardware

Chapter 2: Arduino Setup

The setup from Arduino side requires both SW and HW configuration.

2.1 Arduino HW Components

The required HW components to setup the Arduino side are listed below.

Item	Part Type	Description
DHT1	DHT11 Humidity and Temperature Sensor	Assigned for measuring equipment room's temperature and humidity
DHT2	DHT11 Humidity and Temperature Sensor	Assigned for measuring the outside temperature and humidity
LED	Red (633nm) LED	LED used as an indicator for the A/C status
RFID	RFID-RC522-v2	Displaying room entry and allowing authorized ones to access the room
Breadboard & Wires	Breadboard & Wires	For wiring purposes
R1	220Ω Resistor	tolerance $\pm 5\%$; resistance 220Ω; package 1206 [SMD]
WeMos D1 R1	WeMos D1 R1	Arduino microprocessor compatible with ESP8266 Wi-Fi built-in chipset

2.2 Arduino HW Schematic Diagram

Below schematic diagram shows how to do the wiring between previously mentioned components.

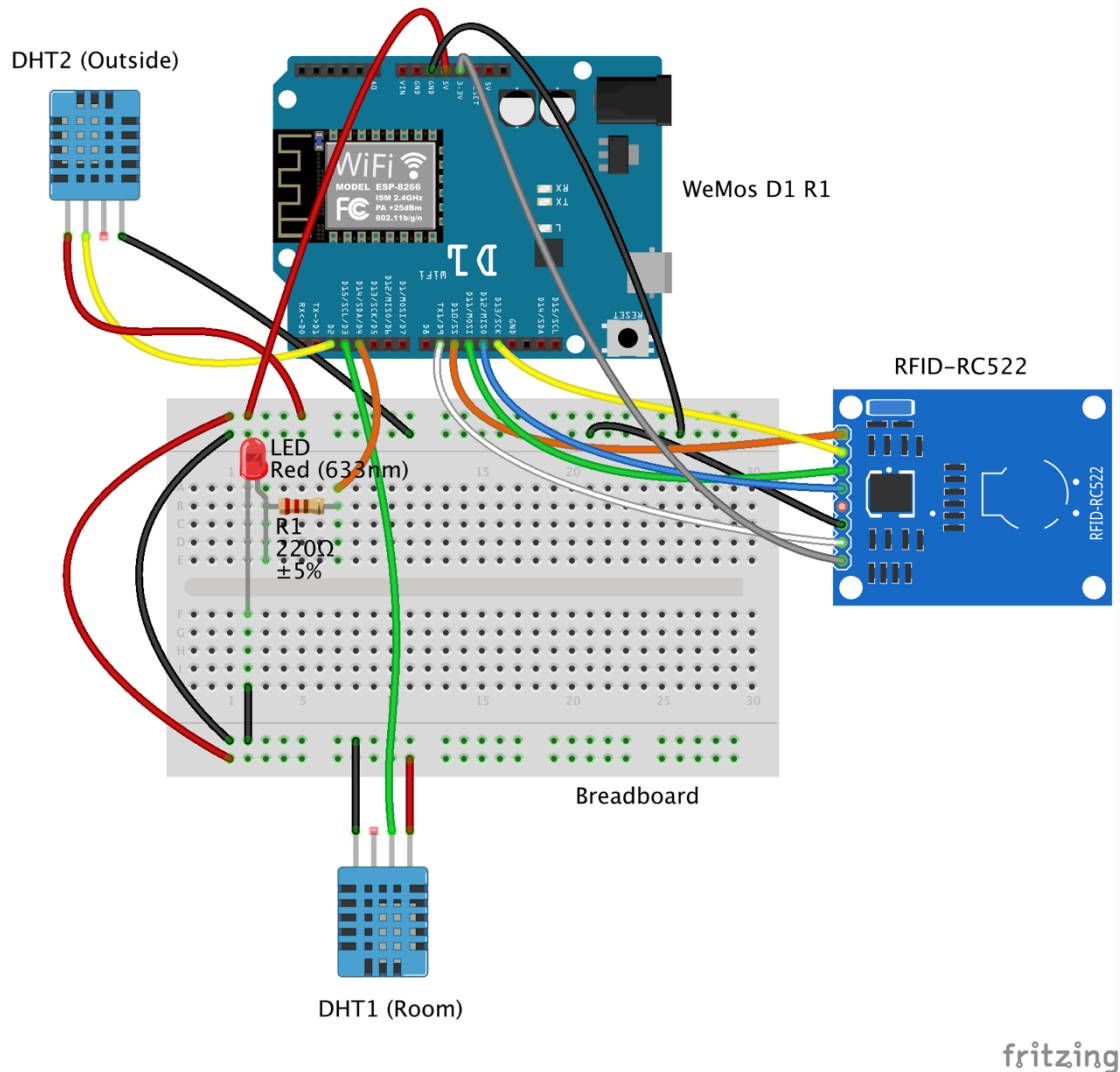


Figure 1: Schematic Diagram

Figure below shows the real scenario of Arduino setup as well.

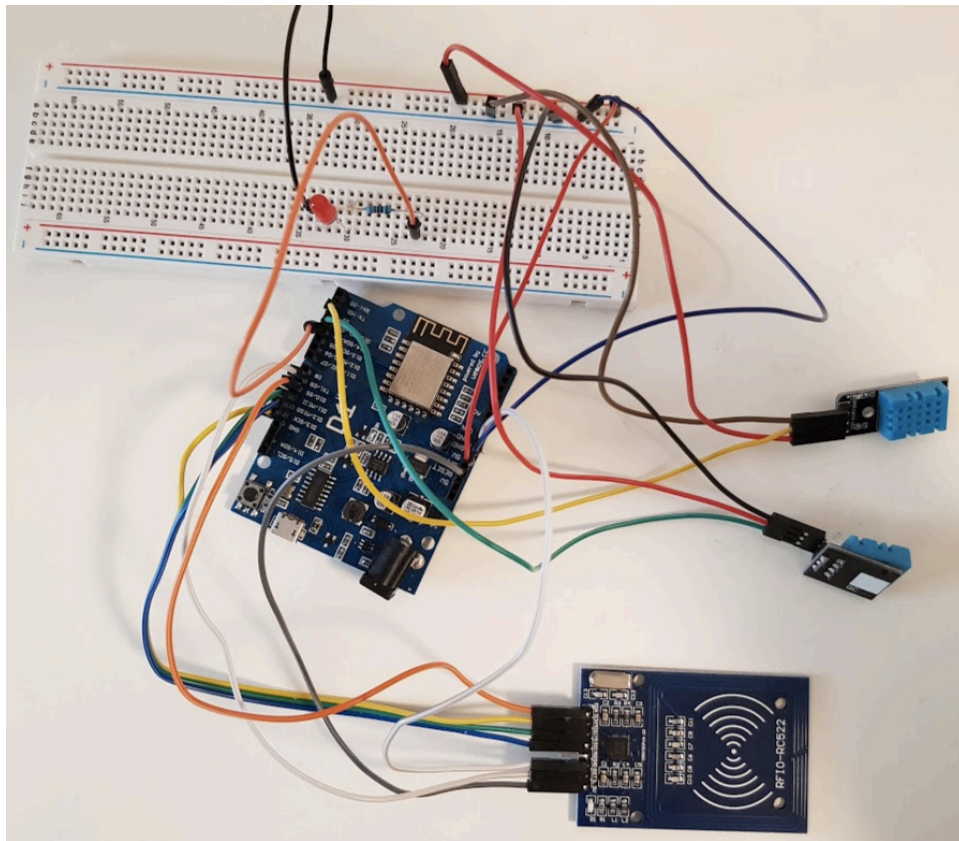


Figure 2: Real Photo of Arduino Setup

The approach of using Wemos D1 instead of Arduino UNO is because of the Wi-Fi built-in ESP8266 chipset, which is not supported in the Arduino UNO (separate component needed). This can reduce the cost and complexity of the solution.

The Wemos D1 Should be powered by using a USB cable connected to PC.

2.3 Arduino SW Setup

To configure the Arduino SW side, below steps should be followed in sequence.

2.3.1 USB Driver

When you attach the USB cable to your Windows PC, it will detect a new hardware and tries to install the USB drivers. In case the USB driver wasn't found in your PC, you will need to download the USB drivers. The USB interface chip is the CH340G. You will need to install the drivers for this chip by searching for the "CH340G drivers" and you can find several sites for all OS versions.

2.3.2 IDE installation

The open-source Arduino software (IDE) should be downloaded from below link based on the required OS:

<https://www.arduino.cc/en/Main/Software>

2.3.3 Setting up the WeMos D1 Board in IDE

Firstly, go to *Preferences* and paste the following URL:

http://arduino.esp8266.com/stable/package_esp8266com_index.json in the “Additional Boards Manager URLs” field as illustrated in below figure.

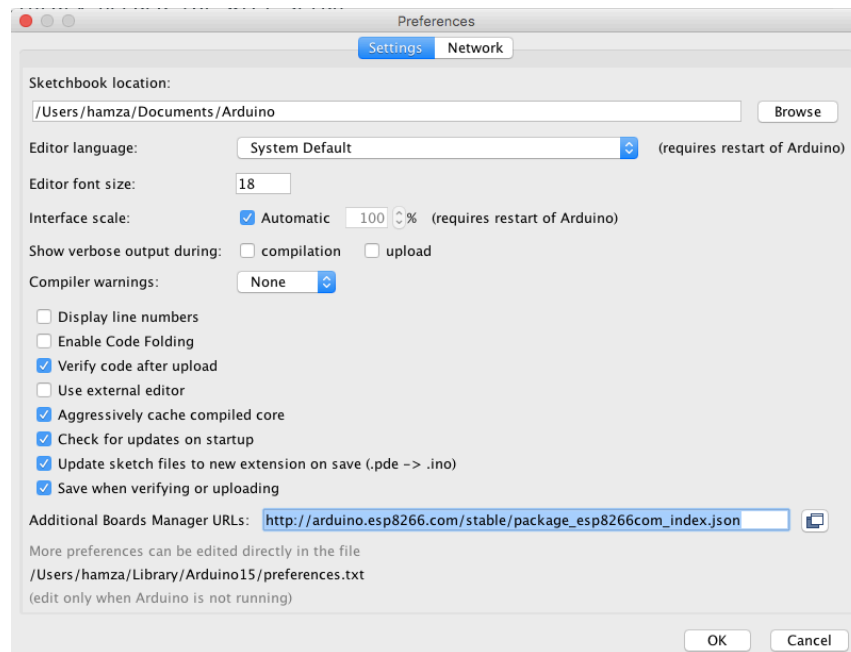


Figure 3: Additional Boards Manager URL

Next, go to *Tools > Board: > Boards Manager*

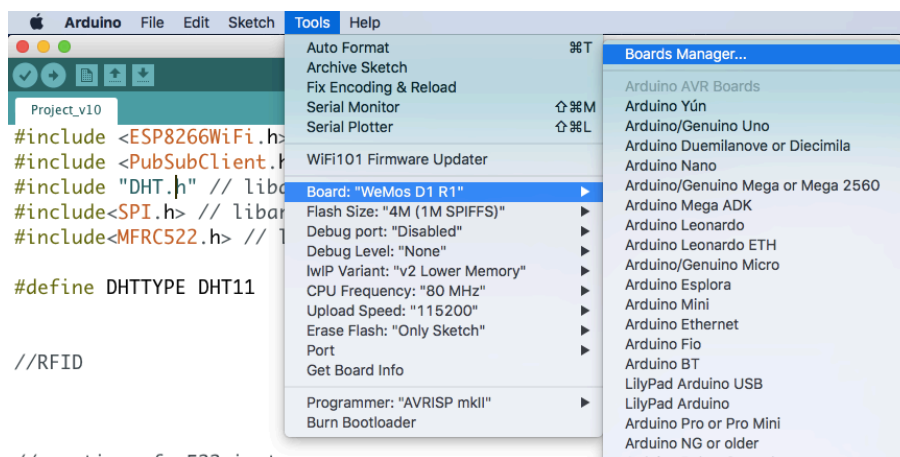


Figure 4: Boards Manager

Search for “ESP8266” and install it as in below figure.

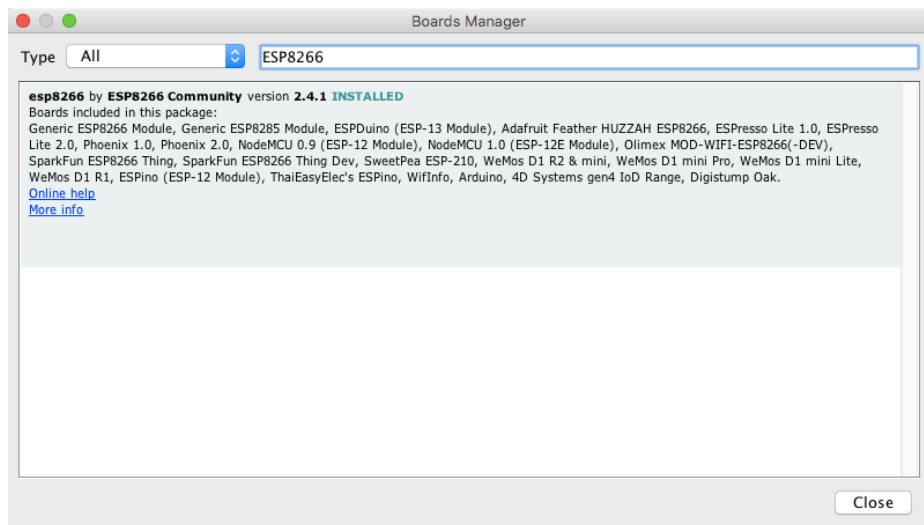


Figure 5: Searching ESP8266

Then, go to *Tools > Board:* and select “WeMos D1 R1”

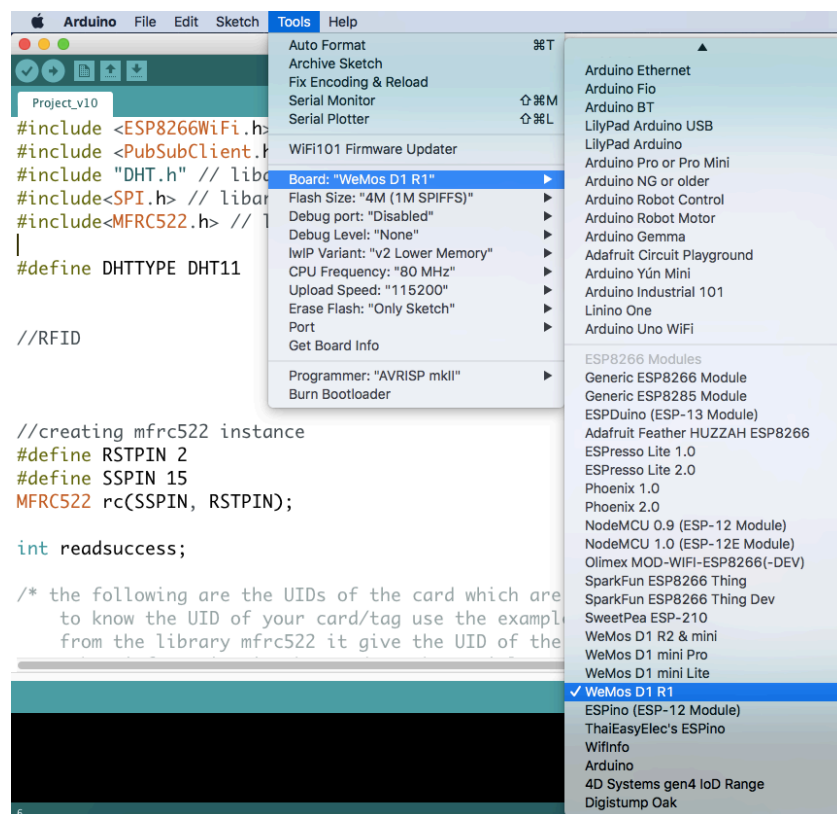


Figure 6: Selecting the right board WeMos D1 R1

2.3.4 Configuring Port Speed

Go to *Tools > Upload Speed:* and select “115200”

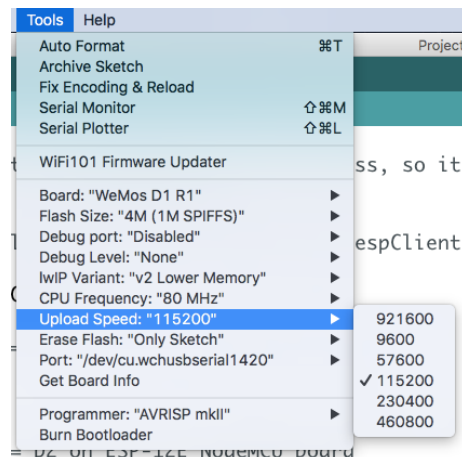


Figure 7: Upload Speed

2.3.5 Activate the Port

Go to *Tools > Port:* and select and active the used port.

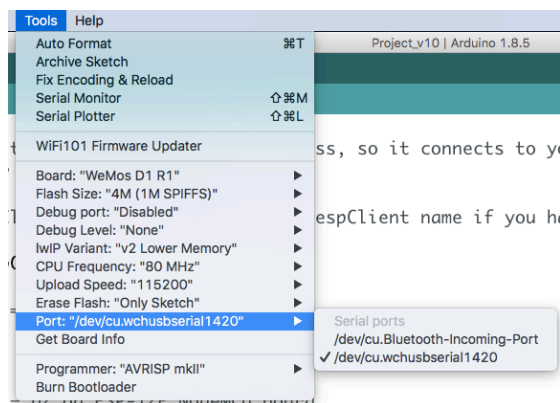


Figure 8: Selecting Port

2.3.6 Installing Missing Libraries

New libraries can be installed manually by uploading the relative .ZIP library files from *Sketch > Include Library > Add .ZIP Library*

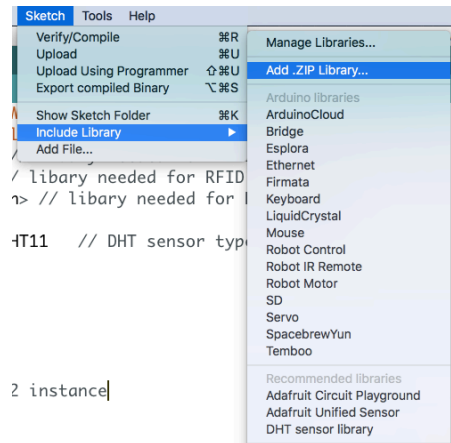


Figure 9: Uploading Libraries

Below libraries should be installed which are available in my GitHub link below:

- DHT.zip → used for DHT humidity and temperature sensors
- DHT_Adafruit_Sensor.zip → used for DHT humidity and temperature sensors
- MFRC522.zip → used for RFID sensor
- pubsubclient-2.6.zip → used to publish/subscribe to MQTT server as a client

GitHub URL: <https://github.com/noweder/Arduino-IoT-Project>

2.3.7 Uploading the Arduino Code

The Arduino final code can be downloaded from the above GitHub link or can be found in the Appendix section.

After opening the code in IDE, it can be uploaded to the WeMos D1 by pressing the “Upload” button and wait till the uploading is completed.

2.3.8 Connecting WeMos D1 to a Wi-Fi AP

A mobile hotspot is used in this scenario as Wi-Fi AP with SSID / Password as following:
AndroidAP / hamooz1234

These details are reflected in the Arduino code as in below code lines:

```
const char* ssid = "AndroidAP";
const char* password = "hamooz1234";
```

If the Wi-Fi AP is powered on, the WeMos D1 will get connected to it and will show below message in the Serial Monitor (Tools > Serial Monitor) indicating the assigned IP address:

WiFi connected - ESP IP address: 192.168.43.179

Chapter 3: Virtual Machine Setup

The Virtual machine will be used to run the IoT message broker and IoT monitoring dashboard. Following steps should be followed in sequence.

3.1 Operating System

The created virtual machine will be running Ubuntu Server 16.04.4 LTS.

It can be downloaded from following link: <https://www.ubuntu.com/download/server>

3.2 Wi-Fi Connectivity

The VM Ubuntu server must connect to same Wi-Fi AP which the WeMos D1 is connected to. To achieve this, below Wi-Fi USB adapter will be plugged in the PC and will be used by the VM Ubuntu server as an external wireless USB adapter to connect to same Wi-Fi AP (AndroidAP). This approach can allow both WeMos D1 and VM Ubuntu server to connect to same network with two separate connectivities.



In order to activate the wireless adapter in the VM, it should be enabled in VM settings from *Devices > USB*. Afterwards, you can connect to the Wi-Fi AP through the WiFi settings of the VM by enabling Wi-Fi and connecting to “AndroidAP”. Run `ipconfig` command in Terminal window to check the assigned IP address which will be within the same network of the WeMos D1 and hence, you should be able to ping the IP address of the WeMos D1.

Note: It is recommended to disable the Wired connection to make sure no other gateway apart from the shared Wi-Fi AP gateway.

In this design, the assigned IP address for the VM server is: 192.168.43.211

```

hamza@hamza-VirtualBox: ~
RX bytes:38377 (38.3 KB) TX bytes:84339 (84.3 KB)

hamza@hamza-VirtualBox:~$ ifconfig
enp0s3 Link encap:Ethernet HWaddr 08:00:27:ba:b4:04
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:6640 errors:0 dropped:0 overruns:0 frame:0
TX packets:3079 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:7040165 (7.0 MB) TX bytes:280296 (280.2 KB)

lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:2928 errors:0 dropped:0 overruns:0 frame:0
TX packets:2928 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:1039893 (1.0 MB) TX bytes:1039893 (1.0 MB)

wlcx83a35e0f0b Link encap:Ethernet HWaddr c8:3a:35:e0:f0:b
inet addr:192.168.43.211 Bcast:192.168.43.255 Mask:255.255.255.0
inet6 addr: fe80::b547:abcb:b4ed:49f4/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:425 errors:0 dropped:0 overruns:0 frame:0
TX packets:501 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:39592 (39.5 KB) TX bytes:85589 (85.5 KB)

hamza@hamza-VirtualBox:~$

```

Figure 10: running ifconfig

```

hamza@hamza-VirtualBox:~$ ping 192.168.43.179
PING 192.168.43.179 (192.168.43.179) 56(84) bytes of data.
64 bytes from 192.168.43.179: icmp_seq=1 ttl=255 time=143 ms
64 bytes from 192.168.43.179: icmp_seq=2 ttl=255 time=168 ms
64 bytes from 192.168.43.179: icmp_seq=3 ttl=255 time=145 ms
64 bytes from 192.168.43.179: icmp_seq=5 ttl=255 time=856 ms
64 bytes from 192.168.43.179: icmp_seq=6 ttl=255 time=330 ms
64 bytes from 192.168.43.179: icmp_seq=7 ttl=255 time=215 ms
64 bytes from 192.168.43.179: icmp_seq=8 ttl=255 time=222 ms

--- 192.168.43.179 ping statistics ---
9 packets transmitted, 7 received, 22% packet loss, time 8016ms
rtt min/avg/max/mdev = 143.743/297.656/856.120/235.620 ms
hamza@hamza-VirtualBox:~$

```

Figure 11: Ping the WeMos D1

3.3 Installing Nginx HTTP Web Server

Please run below command in sequence to complete Nginx HTTP web server configuration:

```

sudo apt-get update
sudo apt-get install nginx
sudo ufw allow 'Nginx HTTP'

```

3.4 Installing Node.js and npm

To install Node.js, please run below command:

```

sudo apt-get install nodejs-legacy

```

To install npm, run below command:

```

sudo apt-get install npm

```

Next, we will use npm to install Node-RED.

3.5 Installing Node-RED

Node-RED is a switchboard for the Internet of Things, a visual tool that helps you connect your favourite apps, websites, and hardware together to do new and useful things. Most often compared to IFTTT, but our approach is to use Node-RED as it has much more powerful and flexible interface, and a large open source community creating nodes to interact with a wide variety of apps and services.

Run below command:

```
sudo npm install -g --unsafe-perm node-red node-red-admin
```

Allow the default port 1880 used by Node-RED in the Firewall:

```
sudo ufw allow 1880
```

Apply below commands to start Node-RED automatically on startup:

```
sudo nano /etc/systemd/system/node-red.service
```

Copy and paste in the following, then save and close the file.

```
[Unit]
Description=Node-RED
After=syslog.target network.target
[Service]
ExecStart=/usr/local/bin/node-red-pi --max-old-space-size=128
-v
Restart=on-failure
KillSignal=SIGINT
# log output to syslog as 'node-red'
SyslogIdentifier=node-red
StandardOutput=syslog
# non-root user to run as
WorkingDirectory=/home/hamza/
User=hamza
Group=hamza
[Install]
WantedBy=multi-user.target
```

Enable the service file to be executed on startup by running below command:

```
sudo systemctl enable node-red
```

On your browser, navigate to <http://127.0.0.1:1880> and the Node-RED web page will be displayed.

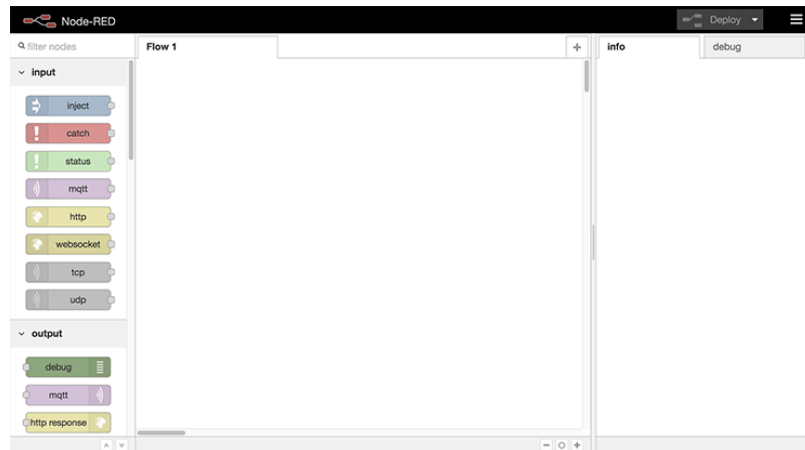


Figure 12: Node-RED Web Page

3.6 Installing Node-RED Dashboard

To install the Node-RED dashboard, run below commands in sequence:

- `cd ~/.node-red`
- `npm i node-red-dashboard`

After the installation finishes, new nodes will be added, such as the “Dashboard Switch” node.

To access the dashboard (UI), type <http://127.0.0.1:1880/ui> in your browser.

3.7 Installing Mosquitto (MQTT Broker)

MQTT is a simple messaging protocol, designed for constrained devices and with low-bandwidth. So, it's the perfect solution for Internet of Things applications.

The MQTT broker is primarily responsible for **receiving** all messages, **filtering** the messages, **decide** who is interested in it and then **publishing** the message to all subscribed clients.

To install the Mosquitto broker and make it auto start enter the following command:

- `sudo apt install mosquitto`
- `sudo systemctl enable mosquitto.service`

As the MQTT broker listens on port 1883, we need to allow this port on the Firewall in order to allow MQTT clients such as the WeMos D1 to communicate with the broker through this port. Run below command to allow the port on Firewall:

```
sudo ufw allow 1883
```

Chapter 4: Solution Customization

After completing the required setup of both the WeMos D1 and VM Ubuntu server, this chapter will highlight the implementation of the customized design meeting the project design requirements.

All MQTT nodes (in & out) are using below common settings.

- Server: 192.168.43.211:1883 (MQTT Broker IP 192.168.43.211 and used port 1883)
- QoS: 2
- Retain: False

Below is an example of one of the nodes:

Figure 13: Node Settings

The same MQTT broker IP is also configured in the Arduino code line below:

```
const char* mqtt_server = "192.168.43.211";
```

4.1 Node-RED Flow Customized Design

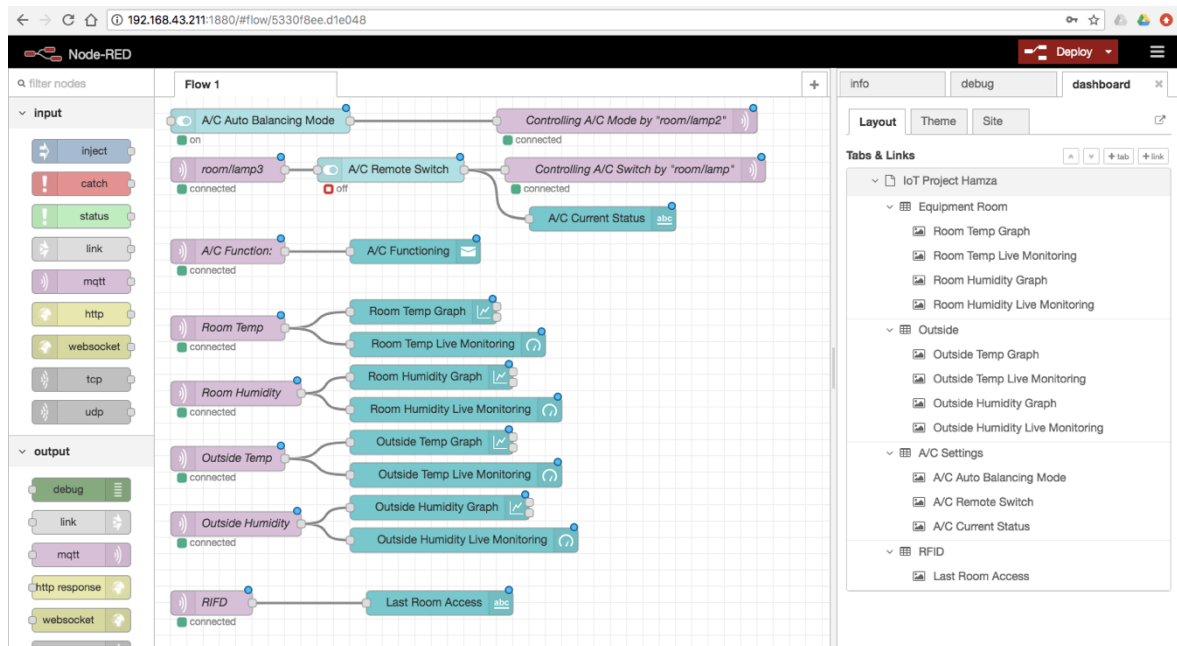


Figure 14: Node-RED Flow Customized Design

4.2 Node-RED Dashboard (User Interface)

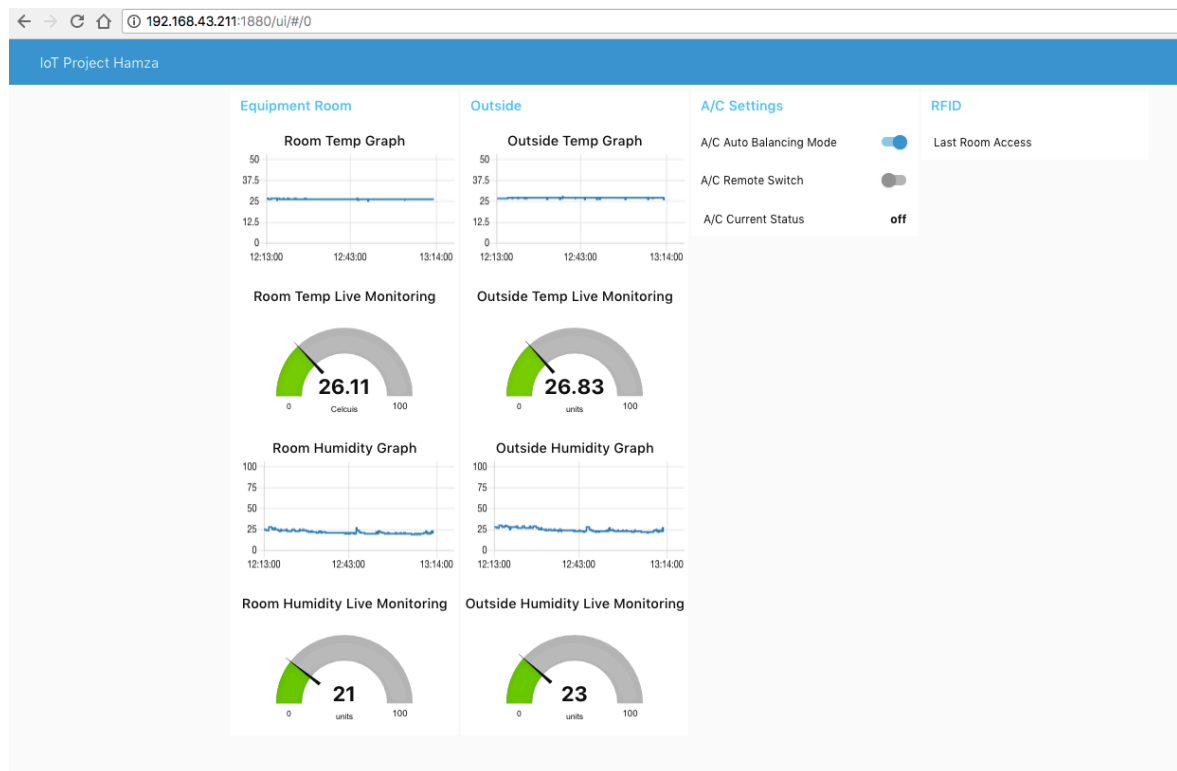
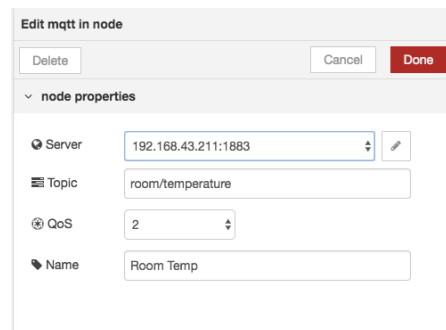


Figure 15: Node-RED Dashboard

4.3 Monitoring Temperature and Humidity

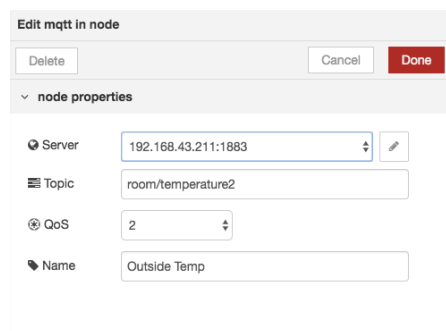
Room Temp MQTT node is subscribed to topic (room/temperature)



The screenshot shows the 'Edit mqtt in node' dialog box. At the top, there are three buttons: 'Delete', 'Cancel', and 'Done'. Below these is a section titled 'node properties' with a dropdown arrow. Under 'node properties', there are four fields: 'Server' with the value '192.168.43.211:1883', 'Topic' with the value 'room/temperature', 'QoS' with the value '2', and 'Name' with the value 'Room Temp'.

Figure 16: Room Temp MQTT node

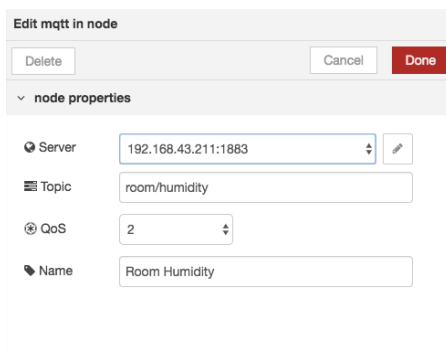
Outside Temp MQTT node is subscribed to topic (room/temperature2)



The screenshot shows the 'Edit mqtt in node' dialog box. At the top, there are three buttons: 'Delete', 'Cancel', and 'Done'. Below these is a section titled 'node properties' with a dropdown arrow. Under 'node properties', there are four fields: 'Server' with the value '192.168.43.211:1883', 'Topic' with the value 'room/temperature2', 'QoS' with the value '2', and 'Name' with the value 'Outside Temp'.

Figure 17: Outside Temp MQTT node

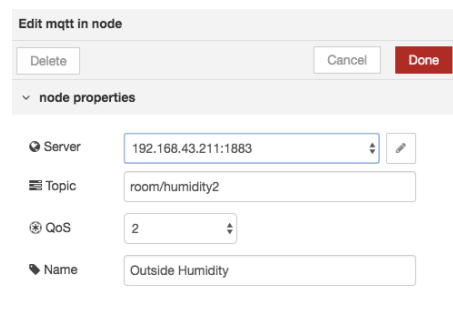
Room Humidity MQTT node is subscribed to topic (room/humidity)



The screenshot shows the 'Edit mqtt in node' dialog box. At the top, there are three buttons: 'Delete', 'Cancel', and 'Done'. Below these is a section titled 'node properties' with a dropdown arrow. Under 'node properties', there are four fields: 'Server' with the value '192.168.43.211:1883', 'Topic' with the value 'room/humidity', 'QoS' with the value '2', and 'Name' with the value 'Room Humidity'.

Figure 18: Room Humidity MQTT node

Outside Humidity MQTT node is subscribed to topic (room/humidity2)



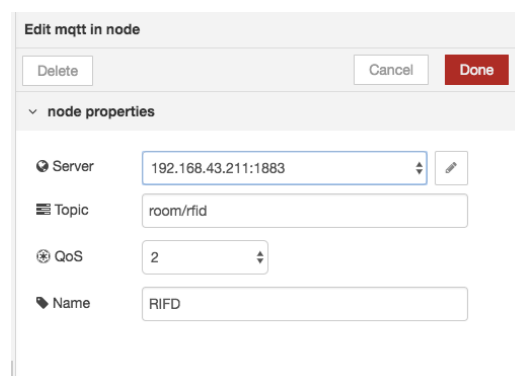
The screenshot shows a dialog box titled "Edit mqtt in node" with buttons for "Delete", "Cancel", and "Done". Under the "node properties" section, the following fields are visible:

- Server: 192.168.43.211:1883
- Topic: room/humidity2
- QoS: 2
- Name: Outside Humidity

Figure 19: Outside Humidity MQTT node

4.4 Monitoring RFID

RFID MQTT node is subscribed to topic (room/rfid)



The screenshot shows a dialog box titled "Edit mqtt in node" with buttons for "Delete", "Cancel", and "Done". Under the "node properties" section, the following fields are visible:

- Server: 192.168.43.211:1883
- Topic: room/rfid
- QoS: 2
- Name: RFID

Figure 20: RFID MQTT node

When scanning a card or a tag on the RFID reader, the name, type and UID will be displayed in the dashboard as below.

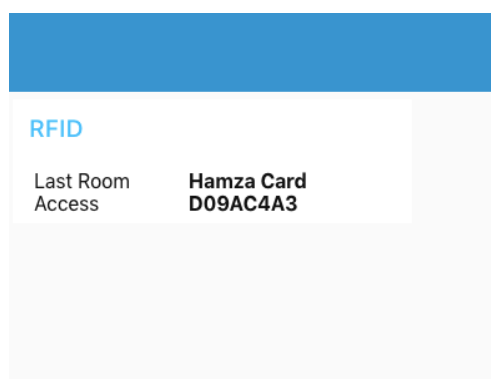


Figure 21: RFID

The current design allows only authorized cards that are already pre-defined in the Arduino code to access the room. If the UID is not pre-defined in the code, the Arduino Serial Monitor will show "unauthorized card" indicating the access is not allowed for this UID.

4.5 A/C Auto Balancing Mode

In this project, the key aim is to maintain a workable balance between the growing room and the equipment room by activating the A/C when there is more than 2 degrees difference between the equipment room and outside.

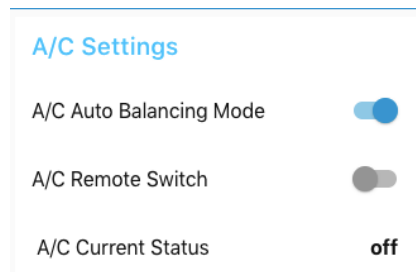


Figure 22: A/C Settings

A dedicated ON/OFF switch is used to ENABLE/DISABLE the auto temperature balancing mode. By applying this approach, the prototype will be more realistic and more intelligent, as the A/C auto balancing mode switch has a higher priority of controlling the A/C than the A/C manual remote switch.

When the auto balancing mode is ON, this means that if temperature difference between the room and outside environment is more than 2 degrees, the A/C must be activated to achieve the temperature balance again without interruption. Hence, while the auto balancing mode is ON, user cannot manually switch the A/C unless the auto balancing mode is switched off, which is the ideal behaviour.

When the auto balancing mode is OFF, this means that A/C will not be automatically activated if the temperature difference between the room and outside environment is more than 2 degrees. This allows the user to activate/deactivate the A/C manually when desired.

The connected MQTT node publishes the Auto Balancing mode status (on or off) in topic (room/lamp2), which the WeMos D1 is also subscribed to as in the Arduino code.

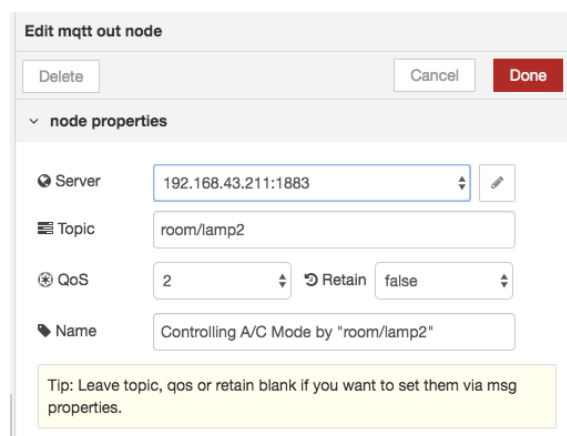
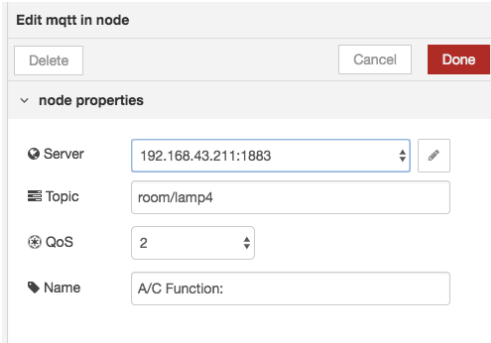


Figure 23: Auto Balancing MQTT out node

4.6 A/C Function Node

This node is used to display a notification at the top-left of the UI whether A/C is heating or cooling while the Auto balancing mode is enabled.

Its relative MQTT node is subscribed to topic (room/lamp4) at which the WeMos D1 publishes the status of A/C (heating or cooling)



The screenshot shows a dialog box titled "Edit mqtt in node". At the top, there are three buttons: "Delete", "Cancel", and "Done". Below these is a section titled "node properties" with a dropdown arrow. Under "node properties", there are four fields: "Server" with the value "192.168.43.211:1883", "Topic" with the value "room/lamp4", "QoS" with the value "2", and "Name" with the value "A/C Function:". Each field has a small edit icon to its right.

Figure 24: A/C Function MQTT node

Below is an example when cooling is active where the room temperature is more than 2 degrees higher than the outside temperature.

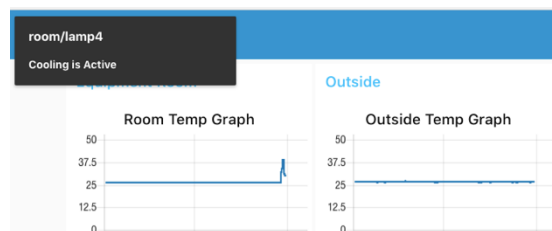


Figure 25: A/C Cooling Notification

It should be also noted that in this design, the LED will be ON Static indicating that A/C is cooling, while the LED will be blinking if the A/C is heating.

4.7 A/C Remote Switch

This switch is used to manually switch ON/OFF the A/C, however, as discussed earlier, the Auto Balancing Mode should firstly be disabled in order to use the manual switch.

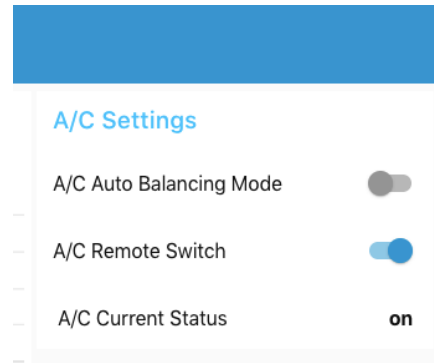


Figure 26: A/C Remote Switch

The output of this switch is published on topic (room/lamp) using MQTT out node below, where this topic is used to control the A/C if Auto Balancing Mode is OFF (payload of topic "room/lamp2" is off)

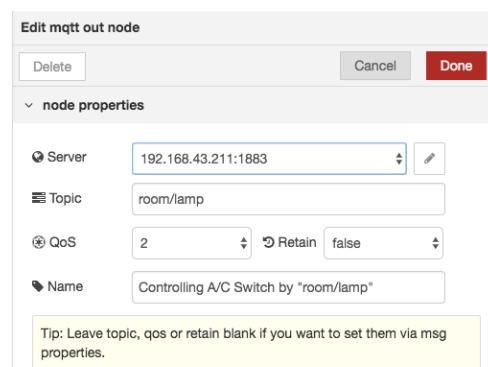


Figure 27: A/C Remote Switch MQTT out node

While the input of this switch is controlled by an MQTT in node which is subscribed to topic (room/lamp3), this topic is used by the WeMos D1 to publish and update A/C switch status.

4.8 A/C Current Status

This node shows the current status of the A/C (on or off) on the UI based on the message payload of the A/C Remote Switch.

Edit text node

Delete Cancel Done

▼ node properties

Group A/C Settings [IoT Project Hamza]

Size auto

Label A/C Current Status

Value format {{msg.payload}}

Layout

label value label value label value

label value label value

Name A/C Current Status

Figure 28: A/C Status Node

Appendix

Arduino Code:

```
#include <ESP8266WiFi.h> // library needed for WiFi setup
#include <PubSubClient.h> // library needed for publish/subscribe to MQTT server as a client
#include "DHT.h" // library needed for DHT11 temperature sensors
#include<SPI.h> // library needed for RFID function
#include<MFRC522.h> // library needed for RFID sensor

#define DHTTYPE DHT11 // DHT sensor type : DHT 11

//RFID

//creating mfrc522 instance
#define RSTPIN 2
#define SSPIN 15
MFRC522 rc(SSPIN, RSTPIN);

int readsucces;

/* the following are the UUIDs of the card which are authorised
to know the UUID of your card/tag use the example code 'DumpInfo'
from the library mfrc522 it give the UUID of the card as well as
other information in the card on the serial monitor of the arduino*/

byte defcard[][4]={{{0xD0,0x9A,0xC4,0xA3},{0x2A,0xE8,0x7D,0x63}}}; //for multiple cards, new
ID cards should be added here to be authorized
int N=2; //change this to the number of cards/tags you will use (number of above IDs)
byte readcard[4]; //stores the UUID of current tag which is read

//function to get the UUID of the card
int getid(){
if(!rc.PICC_IsNewCardPresent()){
return 0;
}
if(!rc.PICC_ReadCardSerial()){
return 0;
}
Serial.println("THE UUID OF THE SCANNED CARD IS:");
for(int i=0;i<4;i++){
readcard[i]=rc.uid.uidByte[i]; //storing the UUID of the tag in readcard
Serial.print(readcard[i],HEX);
}
Serial.println("");
Serial.println("Now Comparing with Authorised cards");
rc.PICC_HaltA();
return 1;
}

int uidnum=99;
//*****

// Change the credentials below, so your ESP8266 connects to your router
const char* ssid = "AndroidAP";
const char* password = "hamooz1234";

// Change the variable to your VM Ubuntu Server IP address, so it connects to your MQTT
broker
const char* mqtt_server = "192.168.43.211";

// Initializes the espClient. You should change the espClient name if you have multiple
ESPs running in your home automation system
WiFiClient espClient;
PubSubClient client(espClient);

// DHT Sensor - GPIO 5 = D1 on ESP-12E NodeMCU board
const int DHTPin = 5;
const int DHTPin2 = 16;
```

```

// Lamp - LED - GPIO 4 = D2 on ESP-12E NodeMCU board
const int lamp = 4;
int ledblink=0; //for blinking loop
int mod=5; //for AC mode heating / cooling / off
float difference; // tempreture difference
int am=1; // AC auto=1/manual=0 mode

// Initialize DHT sensor.
DHT dht(DHTPin, DHTTYPE);
DHT dht2(DHTPin2, DHTTYPE);

// Timers auxiliar variables
long now = millis();
long lastMeasure = 0;

// Don't change the function below. This functions connects your ESP8266 to your router
void setup_wifi() {
  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("WiFi connected - ESP IP address: ");
  Serial.println(WiFi.localIP());
}

// This functions is executed when some device publishes a message to a topic that your
ESP8266 is subscribed to
// Change the function below to add logic to your program, so when a device publishes a
message to a topic that
// your ESP8266 is subscribed you can actually do something
void callback(String topic, byte* message, unsigned int length) {
  Serial.print("Message arrived on topic: ");
  Serial.print(topic);
  Serial.print(". Message: ");
  String messageTemp;
  for (int i = 0; i < length; i++) {
    Serial.print((char)message[i]);
    messageTemp += (char)message[i];
  }
  Serial.println();

  // Feel free to add more if statements to control more GPIOs with MQTT

  // If a message is received on the topic room/lamp2, you check if the message is either on
or off. Turns the lamp GPIO according to the message
  if(topic=="room/lamp2"){
    Serial.print("Changing mode to ");
    if(messageTemp == "on"){
      am=1;
      Serial.print("auto");
    }
    else if(messageTemp == "off"){
      am=0;
      mod=3;
      Serial.print("manual");
    }
  }
  //If auto mode is off
  if(am==0){
    // If a message is received on the topic "room/lamp", you check if the message is either on
or off. Turns the lamp GPIO according to the message
    if(topic=="room/lamp"){
      Serial.print("Changing Room lamp to ");
      if(messageTemp == "on"){
        digitalWrite(lamp, HIGH);
        Serial.print("On");
      }
      else if(messageTemp == "off"){
        digitalWrite(lamp, LOW);
        Serial.print("Off");
      }
    }
  }
}

```



```

}
}
Serial.println();
}

// This functions reconnects your ESP8266 to your MQTT broker
// Change the function below if you want to subscribe to more topics with your ESP8266
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect("ESP8266Client")) {
      Serial.println("connected");
      // Subscribe or resubscribe to a topic
      // You can subscribe to more topics (to control more LEDs in this example)
      *****
      client.subscribe("room/lamp"); // Subscribing to Topic "room/lamp" used to switch the A/C
      (LED) ON/OFF by the IoT dashboard "A/C Remote Switch"
      client.subscribe("room/lamp2"); // Subscribing to Topic "room/lamp2" used to control the
      A/C mode (OFF: Manual Switch || ON: Auto Balancing Mode) by IoT dashboard "A/C Auto
      Balancing Mode" Switch
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

// The setup function sets your ESP GPIOs to Outputs, starts the serial communication at a
// baud rate of 115200
// Sets your mqtt broker and sets the callback function
// The callback function is what receives messages and actually controls the LEDs
void setup() {
  pinMode(lamp, OUTPUT);
  dht.begin();
  Serial.begin(115200); //set baud rate for serial communication between arduino and c to
  115200
  setup_wifi(); // call the function to connect the wifi
  client.setServer(mqtt_server, 1883); //connect to mqtt server using ip and port
  client.setCallback(callback); //run the procedure for resiving topics from mqtt server

  SPI.begin();
  rc.PCD_Init(); //initialize the receiver
  rc.PCD_DumpVersionToSerial(); //show details of card reader module
}

// For this project, you don't need to change anything in the loop function. Basically it
// ensures that you ESP is connected to your broker
void loop() {

  //keep cheking if wifi and mqtt is still connected
  if (!client.connected()) {
    reconnect();
  }
  if(!client.loop())
    client.connect("ESP8266Client");

  now = millis();
  // Publishes new temperature and humidity every 10 seconds
  if (now - lastMeasure > 10000) {
    lastMeasure = now;
    // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
    float h = dht.readHumidity();
    float h2 = dht2.readHumidity();

    // Read temperature as Celsius (the default)
    float t = dht.readTemperature();
    float t2 = dht2.readTemperature();
    // Read temperature as Fahrenheit (isFahrenheit = true)
    float f = dht.readTemperature(true);
    float f2 = dht2.readTemperature(true);

```

```

// Check if any reads failed and exit early (to try again).
if (isnan(h) || isnan(t) || isnan(f)) {
  Serial.println("Failed to read from DHT sensor!");
  return;
}

if (isnan(h2) || isnan(t2) || isnan(f2)) {
  Serial.println("Failed to read from DHT2 sensor!");
  return;
}

// Computes temperature values in Celsius
float hic = dht.computeHeatIndex(t, h, false);
float hic2 = dht2.computeHeatIndex(t2, h2, false);

static char temperatureTemp[7];
static char temperatureTemp2[7];

dtostrf(hic, 6, 2, temperatureTemp);
dtostrf(hic2, 6, 2, temperatureTemp2);

static char humidityTemp[7];
static char humidityTemp2[7];
dtostrf(h, 6, 2, humidityTemp);
dtostrf(h2, 6, 2, humidityTemp2);

// Publishes Temperature and Humidity values
client.publish("room/temperature", temperatureTemp);
client.publish("room/temperature2", temperatureTemp2);

client.publish("room/humidity", humidityTemp);
client.publish("room/humidity2", humidityTemp2);

Serial.print("Humidity: ");
Serial.print(h);
Serial.print(" %\t Temperature: ");
Serial.print(t);
Serial.print(" *C ");
Serial.print(f);
Serial.print(" *F\t Heat index: ");
Serial.print(hic);
Serial.println(" *C ");

Serial.print("Humidity2: ");
Serial.print(h2);
Serial.print(" %\t Temperature2: ");
Serial.print(t2);
Serial.print(" *C ");
Serial.print(f2);
Serial.print(" *F\t Heat index: ");
Serial.print(hic2);
Serial.println(" *C ");
// Serial.print(hif);
// Serial.println(" *F");

if ((t > t2 || t2 > t) && am == 1) { // Comparing the difference in temperature and powering
  A/C when difference is more than 2 degrees (IF A/C MODE IS AUTO "am=1"), also publish the
  LED again
  difference = t - t2;
  if (difference > 2) { // if Room Temp is more than 2 degree higher than Outside Temp -->
  A/C will start COOLING to achieve balanced temp
  Serial.println("A/C Cooling ON...");
  digitalWrite(lamp, HIGH); //
  mod=0; //mode for cooling
  client.publish("room/lamp3", "on"); // update A/C Switch as ON
  client.publish("room/lamp4", "Cooling is Active");

  } else if (difference < -2) { // if Room Temp is 2 degrees lower than Outside Temp --> A/C
  will start Heating the room to achieve balanced temp
  Serial.println("A/C Heating ON...");
  //digitalWrite(lamp, HIGH); //
  mod=1; //mode for heating
  client.publish("room/lamp3", "on"); // update A/C Switch as ON
  client.publish("room/lamp4", "Heating is Active");

  } else {
  Serial.println("Balanced Temperature... A/C OFF");

```

```

digitalWrite(lamp, LOW);
mod=2;//mode for off
client.publish("room/lamp3", "off"); //turn off light
}

}
}

if(mod==1){ //if mode is heating blink led to show it is heating not cooling
if (ledblink == 1) {
digitalWrite(lamp, LOW);
ledblink=0;
// turn LED OFF
} else {
digitalWrite(lamp, HIGH); // turn LED ON
ledblink=1;
}
}
delay (300);

readsuccess = getid();

if(readsuccess){
int match=0;
//this is the part where compare the current tag with pre defined tags
for(int i=0;i<N;i++){
Serial.print("Testing Against Authorised card no: ");
Serial.println(i+1);
if(!memcmp(readcard,defcard[i],4)){
match++;
uidnum=i; // shows the sequence number of the readed card based on defined IDs list after
comparing it with this list at beggining.
}
}
if(match){
Serial.println("CARD AUTHORISED"); // if readed card matchs one of the defined cards, then
it is an authorized card/tag
switch(uidnum){
case(0):Serial.println("D09AC4A3");client.publish("room/rfid","Hamza Card D09AC4A3");break;
// in case uidnum=0, then will print and publich the UID at index 0 (1st UID) on the pre-
defined list "defcard"
case(1):Serial.println("2AE87D63");client.publish("room/rfid","Hamza Tag 2AE87D63");break;
// in case uidnum=1, then will print and publich the UID at index 1 (2nd UID) on the pre-
defined list "defcard"
// more cases "for example: case(2)" should be added here as well if new UIDs are added in
the UID pre-defined list "defcard"
}

}
else {
Serial.println("CARD NOT Authorised"); // if readed card DO NOT match any of the defined
cards, then print NOT authorized card
client.publish("room/rfid","Not authorized"); // publish card not authorized
}
}
}
}

```