

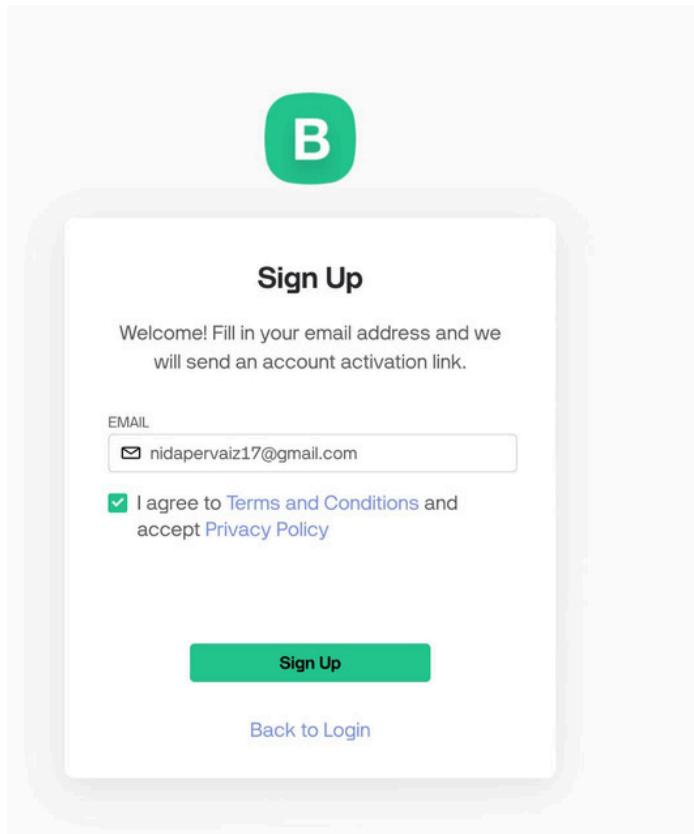
Implementing Leading IoT Cloud Platforms with ESP8266

Pakistan Institute of Engineering and Applied Sciences

Blynk

Performed by Nida Pervaiz

Step 1: First i **signed up** on <https://blynk.cloud>, the official website for Blynk IoT (Blynk 2.0). Following are the sign up steps. Entered my email and verified it through the link that i got in email from Blynk.



Step 2: Created **new template** and filled in details

- Name: SmartStreetLight
- Hardware: ESP8266
- Connection Type: WiFi

A screenshot of the Blynk Console interface. The top navigation bar shows the URL 'sgp1.blynk.cloud/dashboard?760752/global/devices/1'. The left sidebar has a 'Devices' tab selected, along with other options like 'Dashboards', 'Automations', 'Demand Response', 'Fleet Management', and 'In-App Messaging'. The main content area features a large button with the text 'Start by creating your first template'. Below this, a descriptive text states: 'Template is a digital model of a physical object. It is used in Blynk platform as a template to be assigned to devices.' At the bottom of the main area is a green 'New Template' button.

SmartStreetLight

Firmware configuration

```
#define BLYNK_TEMPLATE_ID  
"TMPLGCEFsHe9"  
#define BLYNK_TEMPLATE_NAME  
"SmartStreetLight"
```

Step 3: Inside the template, went to Datastreams tab. Clicked “+ New Datastream”
 → chose Virtual Pin.
 Name: Pin1
 Pin: 0
 Pin Mode: Output
 And created it by clicking create.

Digital Pin Datastream

NAME: Pin1

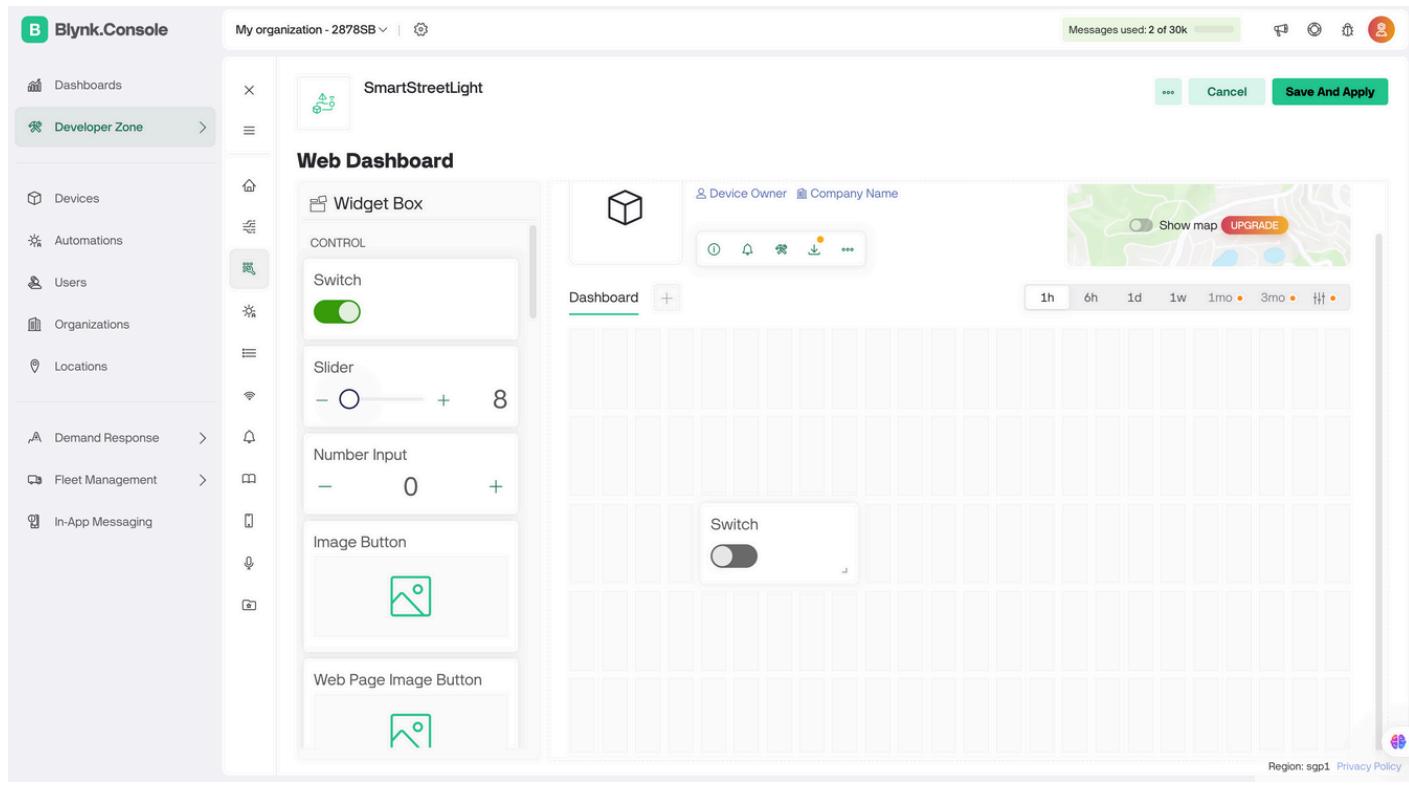
PIN: 0

PIN MODE: Output

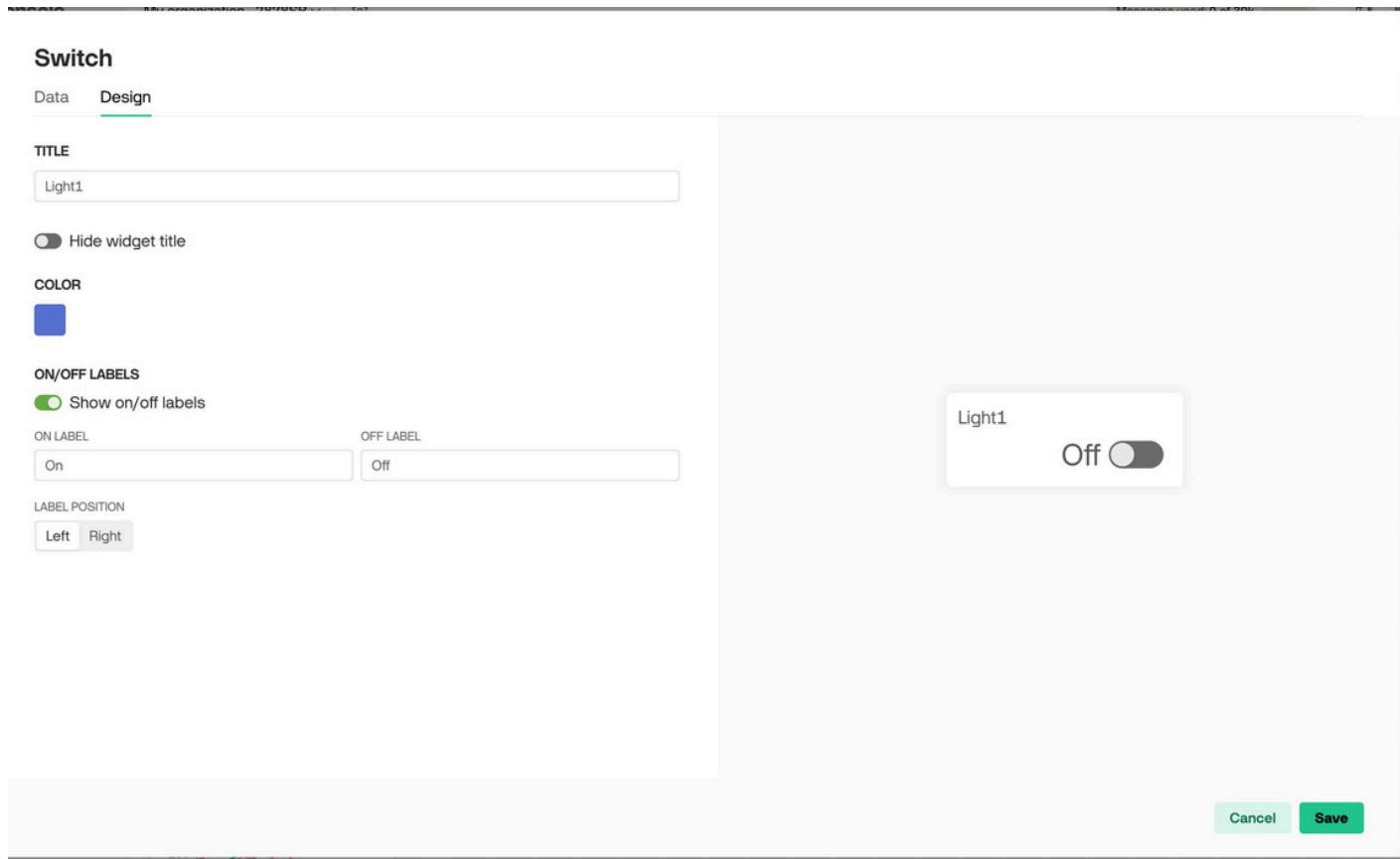
Create

This sets up a channel (pin) to send or receive data like turning an LED on/off..

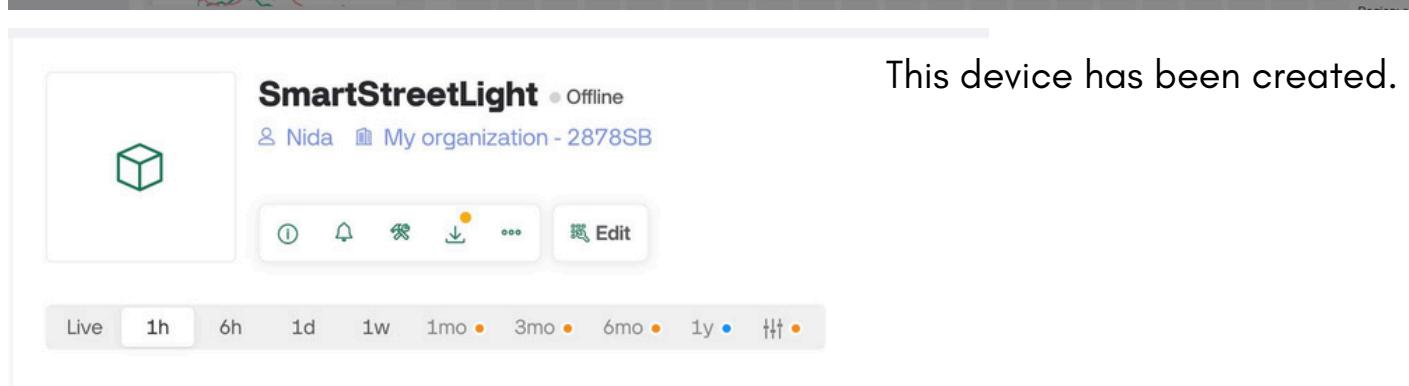
Step 4: Create a **New Device** from Template and added a switch to it by drag and drop and then i edited its design etc.



The screenshot shows the Blynk Console interface for creating a new device. On the left, a sidebar lists various sections: Dashboards, Developer Zone (selected), Devices, Automations, Users, Organizations, Locations, Demand Response, Fleet Management, and In-App Messaging. The main area is titled "SmartStreetLight" and "Web Dashboard". It features a "Widget Box" section with a "CONTROL" tab containing a "Switch" (green on), "Slider" (value 8), "Number Input" (value 0), "Image Button", and "Web Page Image Button". The main workspace is a grid where a "Switch" widget has been placed. Top right buttons include "Cancel" and "Save And Apply". Bottom right shows "Region: sgp1 Privacy Policy".



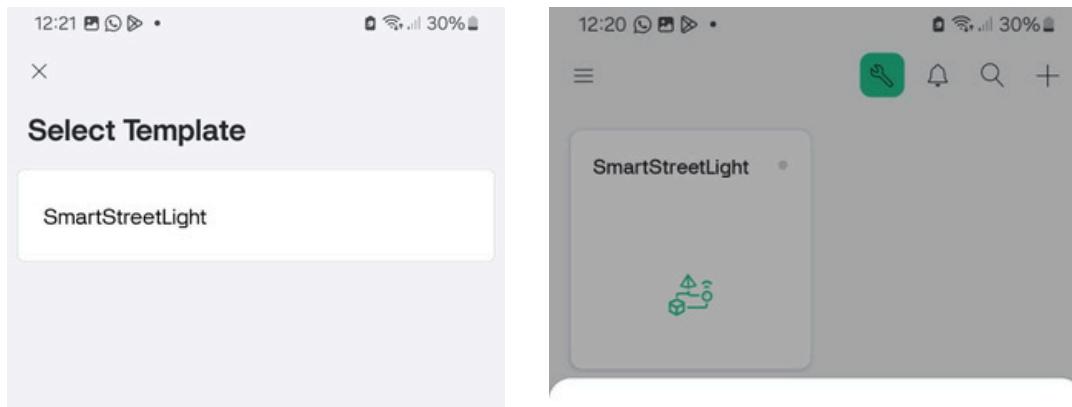
The screenshot shows the "Switch" design configuration interface. The "Design" tab is selected. Under "TITLE", the value "Light1" is entered. A checkbox "Hide widget title" is checked. Under "COLOR", a blue square is selected. Under "ON/OFF LABELS", a checkbox "Show on/off labels" is checked, with "ON LABEL" set to "On" and "OFF LABEL" set to "Off". Under "LABEL POSITION", "Left" is selected. On the right, a preview window shows the switch labeled "Light1" with the "Off" state selected. Bottom right buttons are "Cancel" and "Save".



The screenshot shows the summary card for the "SmartStreetLight" device. It displays the device name, status as "Offline", owner "Nida", organization "My organization - 2878SB", and a "Edit" button. Below the card is a timeline with time intervals: Live, 1h, 6h, 1d, 1w, 1mo (orange dot), 3mo (orange dot), 6mo (orange dot), 1y (blue dot), and 11y (blue dot).

Step 5:

Since we already created a template named 'SmartStreetLight' on the Blynk Cloud, we'll now set up the mobile app to control our device using that template."



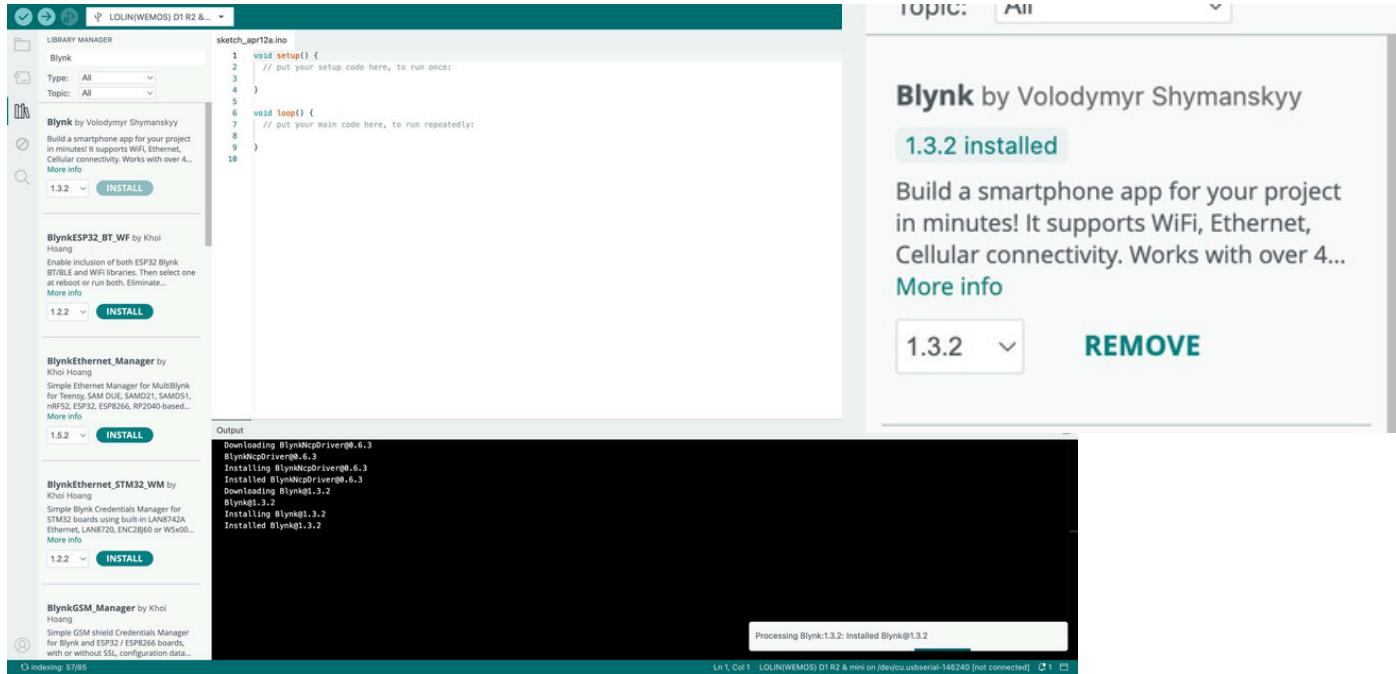
Upon selecting this template it already has a button that we added on blynk cloud in our device SmartStreetLight. I then setup the settings of it like Data stream, off/on values etc.

The left screenshot shows the initial configuration of the button. It has a circular shape with the word "OFF" inside. Below it is a "Data" section with a "DATASTREAM" field containing "Choose datastream..." and a "+" button. Underneath is a "Settings" section with a "MODE" switch set to "Switch". A note below says, "When finger is released - the button will stay in the pressed state". At the bottom are "Settings" and "Design" tabs.

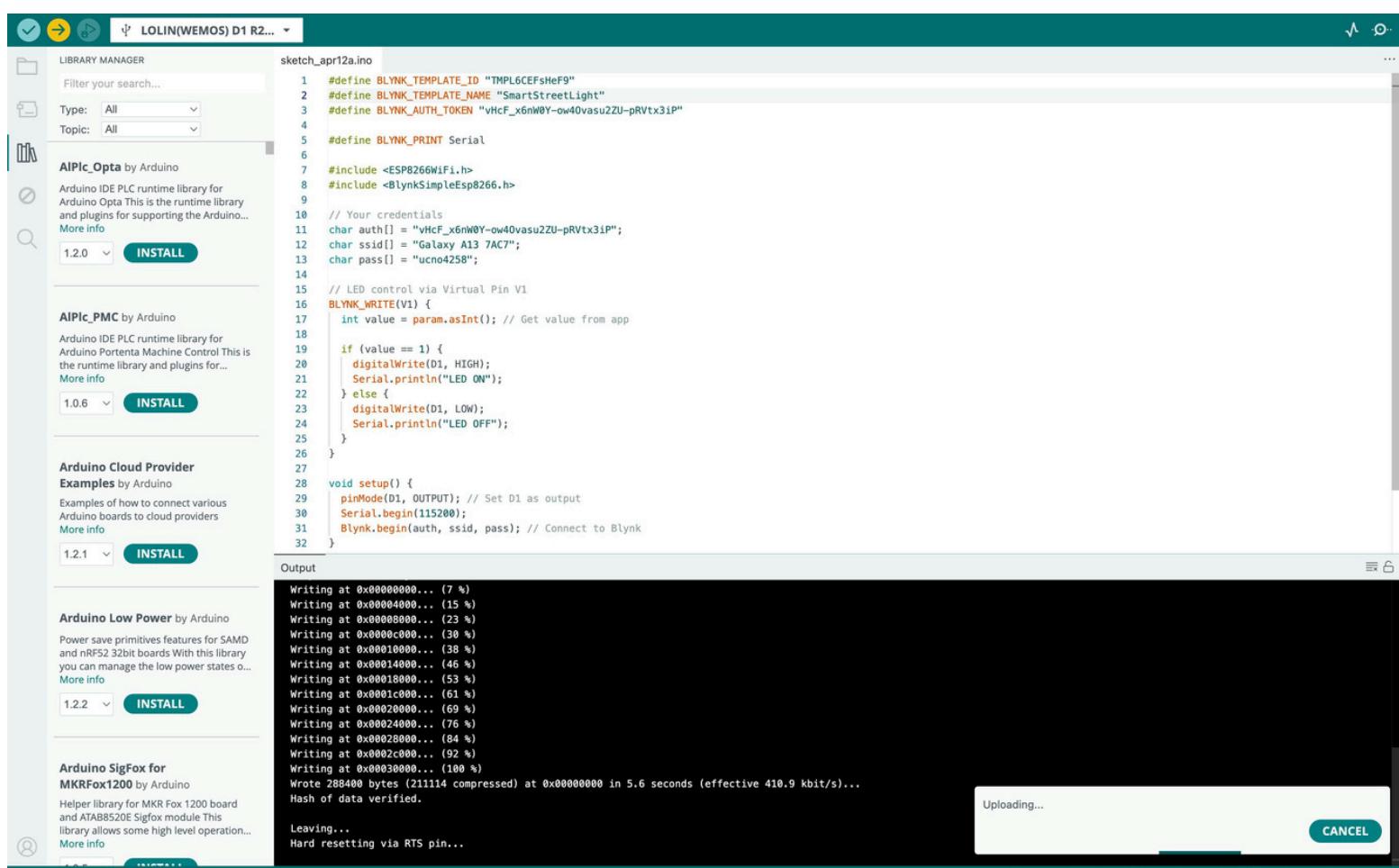
The right screenshot shows the configuration after setting up the data stream. The "Data" section now lists "Integer V1 (V1)" as the selected data stream, with a note "Integer, 0/1, id=2" and a " > " button. Below it is an "OFF/ON VALUES" section with a toggle switch labeled "Use datastream's Min/Max" and "0-1". Underneath is another "Settings" section with a "MODE" switch set to "Switch", the same note about staying in the pressed state, and the same "Settings" and "Design" tabs at the bottom.

Step 6: Setting up Arduino IDE to program the ESP8266 module for Blynk-based IoT control.

First of all installed the Blynk library.



Wrote the code to control and LED. I have attached the screen shot and have also added the code in text form here. In the code i added BLYNK-AUTHENTICATION TOKEN, BLYNK_TEMPLATE_NAME, BLYNK_TEMPLATE_ID, and connected my laptop and esp8266 with my mobile internet and then compiled and uploaded the code.



```
#define BLYNK_TEMPLATE_ID "TMPL6CEF9"
#define BLYNK_TEMPLATE_NAME "SmartStreetLight"
#define BLYNK_AUTH_TOKEN "vHcF_x6nW0Y-ow4Ovasu2ZU-pRVtx3iP"

#define BLYNK_PRINT Serial

#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>

// Your credentials
char auth[] = "vHcF_x6nW0Y-ow4Ovasu2ZU-pRVtx3iP";
char ssid[] = "Galaxy A13 7AC7";
char pass[] = "ucno4258";

// LED control via Virtual Pin V1
BLYNK_WRITE(V1) {
    int value = param.asInt(); // Get value from app

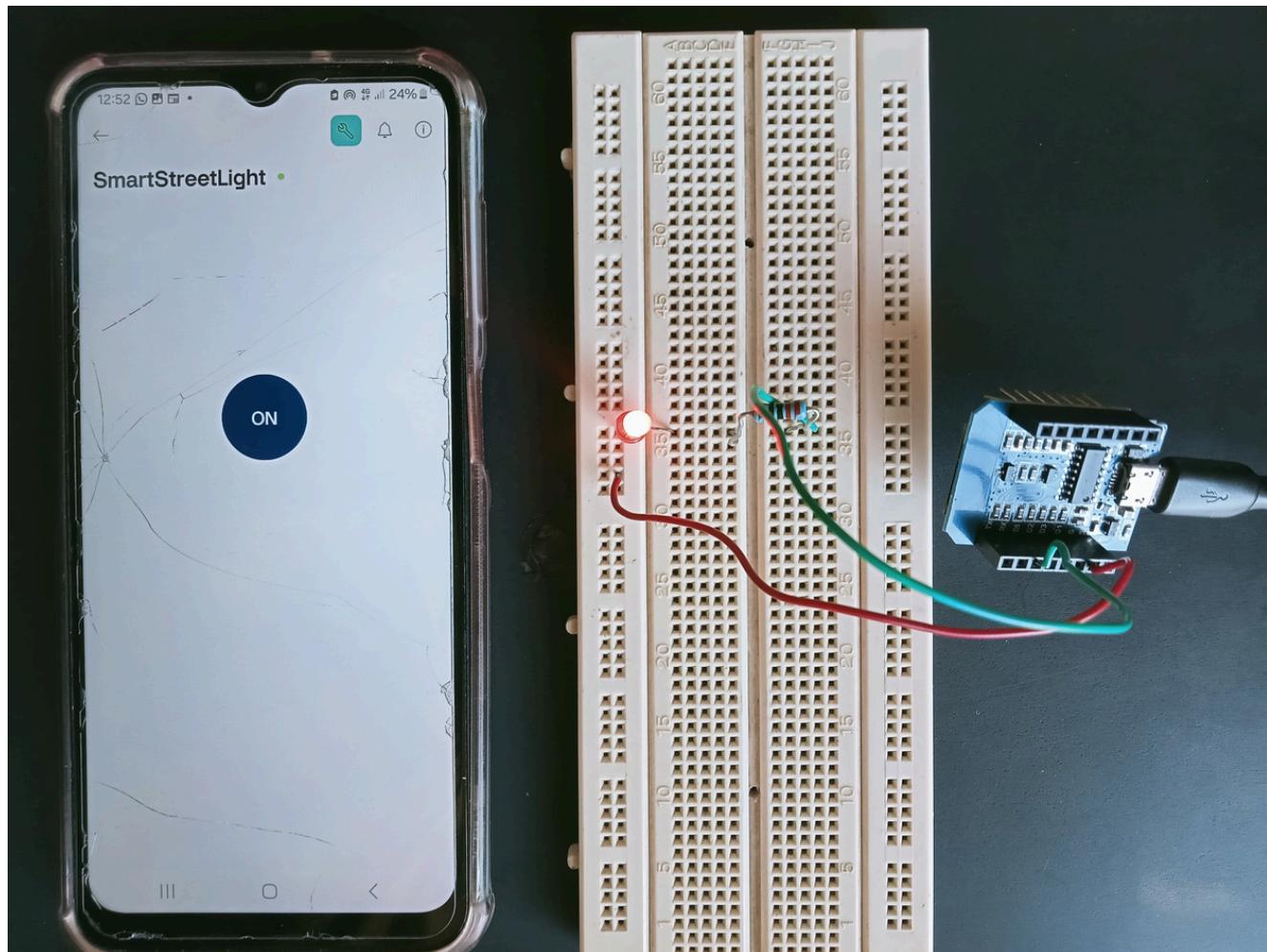
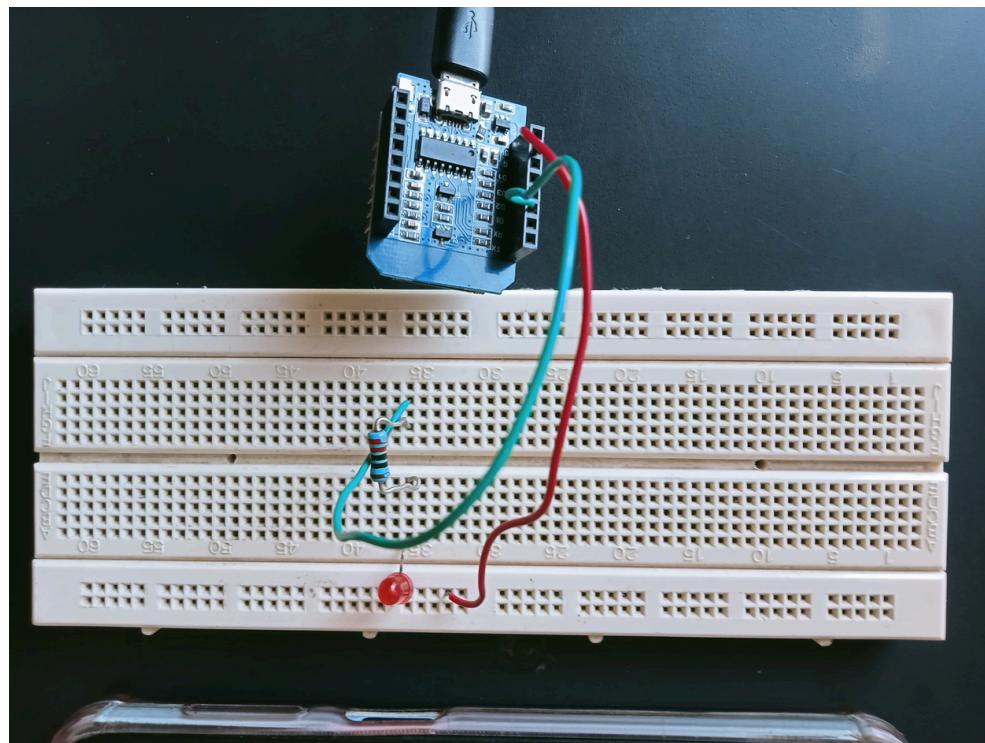
    if (value == 1) {
        digitalWrite(D1, HIGH);
        Serial.println("LED ON");
    } else {
        digitalWrite(D1, LOW);
        Serial.println("LED OFF");
    }
}

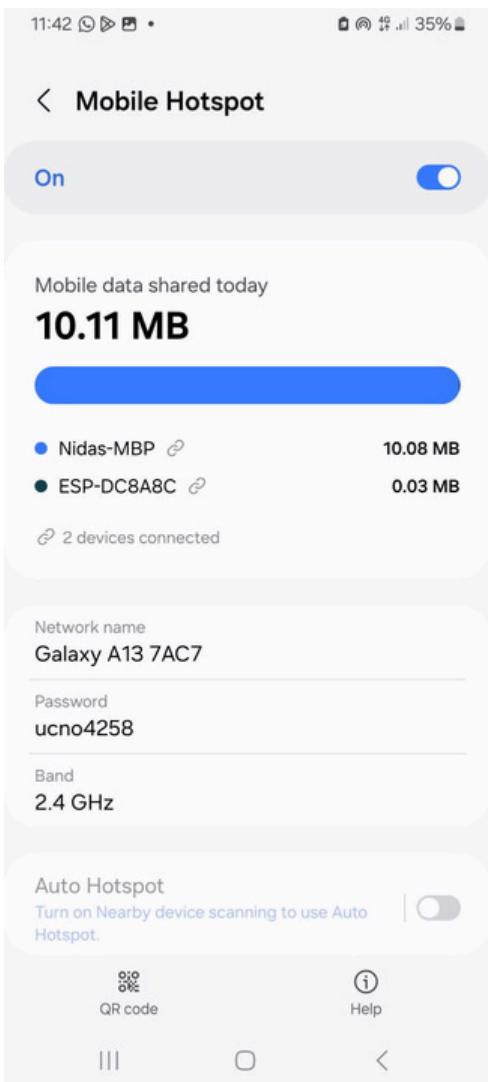
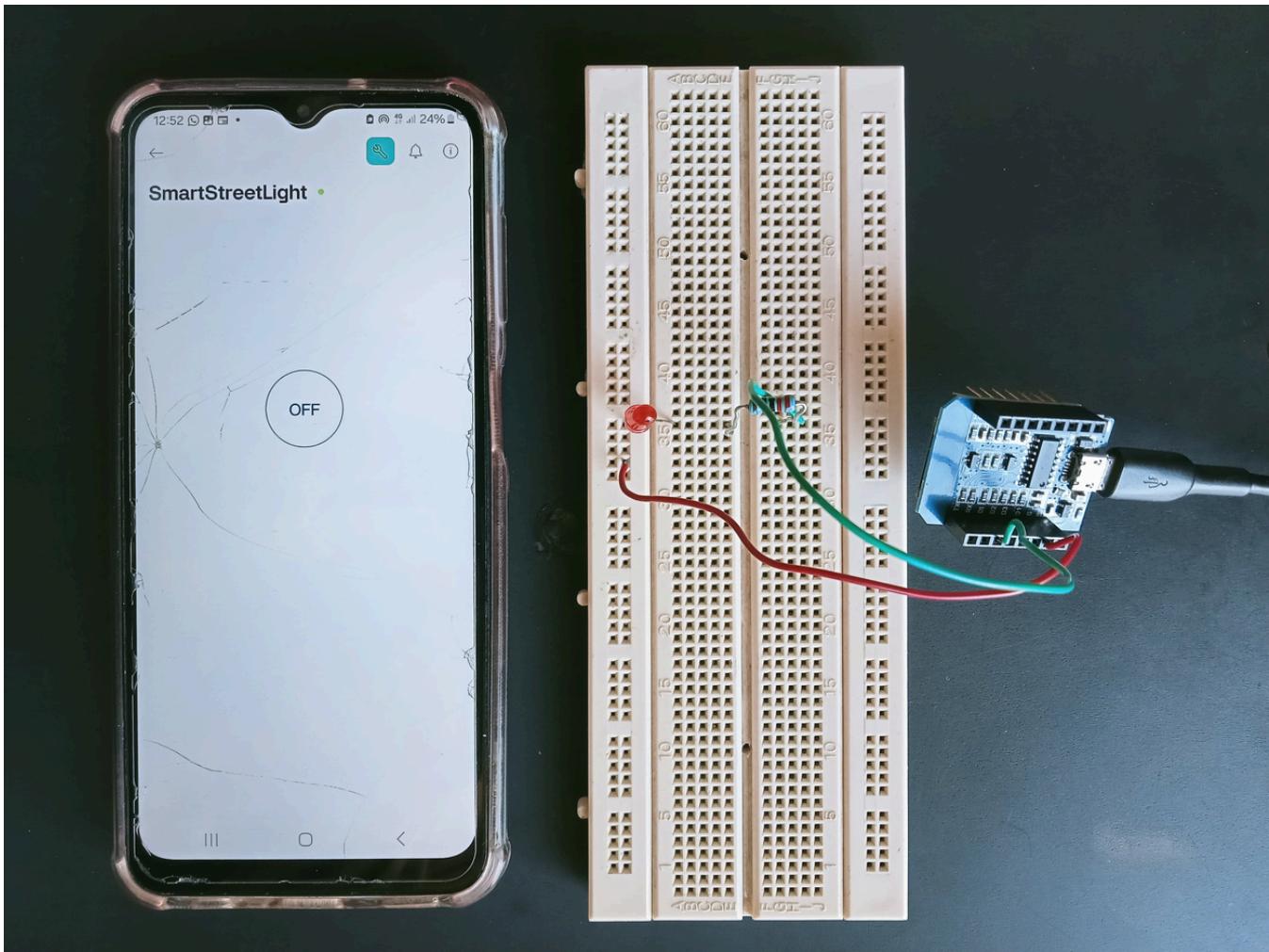
void setup() {
    pinMode(D1, OUTPUT); // Set D1 as output
    Serial.begin(115200);
    Blynk.begin(auth, ssid, pass); // Connect to Blynk
}

void loop() {
    Blynk.run();
}
```

Step 7:

Following is the circuit that i connected with the esp8266. It shows that when i turned the button ON, the LED got ON and when i turned the button OFF, the LED turned OFF.





Challenges:

No issues faced. Every step was smooth and easy

THINGSPEAK

 Performed by Wajeeha Jahangir

Step 1: Installed the ThingSpeak and Adafruit BME280 (for BME 280 sensor) libraries.

Output

Downloading ThingSpeak@2.1.0

ThingSpeak@2.1.0

Installing ThingSpeak@2.1.0

Installed ThingSpeak@2.1.0

The screenshot shows the Arduino Library Manager interface. At the top, there are three circular icons: a checkmark, a right arrow, and a refresh symbol. To the right of these is a USB port icon followed by the text "LOLIN(WEMOS) D1 R2 &...". Below this is a sidebar with icons for file, sketch, tools, and search. The main area is titled "LIBRARY MANAGER". A search bar contains the text "ThingSpeak". Underneath it, there are two dropdown menus: "Type: All" and "Topic: All". The first result listed is "ThingSpeak by MathWorks <support@thingspeak.com>". It shows the version "2.1.0 installed" in green. Below the version number is a description: "ThingSpeak Communication Library for Arduino, ESP8266 & EPS32 ThingSpeak (<https://www.thingspeak.com>) is an analyt...". There is a "More info" link and a "REMOVE" button. To the right of the "REMOVE" button is a dropdown menu showing "2.1.0" and a "2.1.0" link. The second result listed is "ThingSpeak_asukiaaa by Asuki Kono". It describes the library as an API manager for ThingSpeak that writes field values for ThinkgSpeak. There is a "More info" link and an "INSTALL" button. To the right of the "INSTALL" button is a dropdown menu showing "1.0.1" and a "1.0.1" link.

Output

```
Already installed Adafruit Unified Sensor@1.1.15
Downloading Adafruit BME280 Library@2.2.4
Adafruit BME280 Library@2.2.4
Installing Adafruit BME280 Library@2.2.4
Installed Adafruit BME280 Library@2.2.4
Downloading Adafruit BusIO@1.17.0
Adafruit BusIO@1.17.0
Installing Adafruit BusIO@1.17.0
Installed Adafruit BusIO@1.17.0
```

LIBRARY MANAGER

Adafruit_BME280 library

Type: All

Topic: All

Adafruit BME280 Library by Adafruit

2.2.4 installed

Arduino library for BME280 sensors.
Arduino library for BME280 humidity and pressure sensors.

[More info](#)

2.2.4

[REMOVE](#)

ThingSpeak™ Channels Apps Support ▾

Commercial Use How to Buy

To use ThingSpeak, you must sign in with your existing MathWorks account or create a new one.

Non-commercial users may use ThingSpeak for free. Free accounts offer limits on certain functionality. Commercial users are eligible for a time-limited free evaluation. To get full access to the MATLAB analysis features on ThingSpeak, log in to ThingSpeak using the email address associated with your university or organization.

To send data faster to ThingSpeak or to send more data from more devices, consider the [paid license options](#) for commercial, academic, home and student usage.

MathWorks®

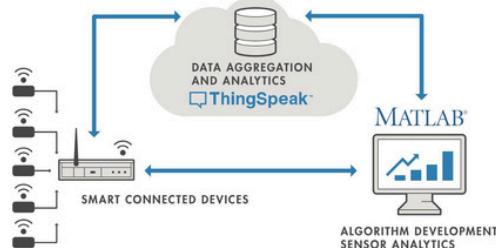
Email

bscs2253@pieas.edu.pk

No account? [Create one!](#)

By signing in, you agree to our [privacy policy](#).

Next



MathWorks®

[← bscs2253@pieas.edu.pk](#)

Password

.....|

[Forgot Password?](#)

[Sign In](#)

Signed in successfully.

Sign-up successful

Congratulations, you have successfully linked your MathWorks account to ThingSpeak. Use the following email ID and its associated MathWorks account password on all subsequent logins to ThingSpeak.

Email ID: bscs2253@pieas.edu.pk

Welcome to ThingSpeak!

OK

Step 3: Then created a new channel, entered its name and description etc. As the channel was supposed to get data from the BME 280 sensor so named it so. But we didn't have any sensor available so i hardcoded the temperature value (a single field for now).

ThingSpeak™ [Channels](#) [Apps](#) [Devices](#) [Support](#)

New Channel

Name: BME280Readings

Description: Readings from BME280 (ESP8266) [+G](#)

Field 1: Temperature

Field 2:

Field 3:

Field 4:

Field 5:

Field 6:

Field 7:

Field 8:

Metadata:

Tags:
(Tags are comma separated)

Help

Channels store data and can hold any type of data. You can add fields to a channel, which are used to store specific data types.

Channel

- Percentage
- Channel
- Description
- Field#: Can be up to 8 fields
- Metadata
- Tags: Enter tags to specify the channel
- Show Channel
- Location of the channel
- Location city
- Latitude
- Longitude
- Video URL
- the full path



Edited the chart's x and y-axis titles.

Step 4: Wrote code in Arduino to publish the temperature. Following is the code.

```
/*
Based on Random Nerd Tutorials example
*/
#include <ESP8266WiFi.h>
#include "ThingSpeak.h"

// WiFi credentials
const char* ssid = "WajeehaPhone";
const char* password = "987684";

// ThingSpeak settings
unsigned long myChannelNumber = 1;
const char* myWriteAPIKey = "68LKDJ8K6FW5DNNN
";
WiFiClient client;

// Timer variables
unsigned long lastTime = 0;
unsigned long timerDelay = 30000; // 30 seconds

// Hardcoded temperature value
float temperatureC = 28.5;

void setup() {
    Serial.begin(115200);
    WiFi.mode(WIFI_STA);
    ThingSpeak.begin(client);
}

void loop() {
    if ((millis() - lastTime) > timerDelay) {
```

```
// Connect or reconnect to WiFi
if (WiFi.status() != WL_CONNECTED) {
Serial.print("Connecting to WiFi");
while (WiFi.status() != WL_CONNECTED) {
WiFi.begin(ssid, password);
delay(5000);
Serial.print(".");
}
Serial.println("\nWiFi connected");
}

// Display and send hardcoded temperature
Serial.print("Sending hardcoded temperature (°C): ");
Serial.println(temperatureC);
int x = ThingSpeak.writeField(myChannelNumber, 1, temperatureC, myWriteAPIKey);
if (x == 200) {
Serial.println("Channel update successful.");
} else {
Serial.println("Problem updating channel. HTTP error code " + String(x));
}

lastTime = millis();
}
```

```
18 // Timer variables
19 unsigned long lastTime = 0;
20 unsigned long timerDelay = 30000; // 30 seconds
21
22 // Hardcoded temperature value
23 float temperatureC = 28.5;
24
25
26 void setup() {
27   Serial.begin(115200);
28   WiFi.mode(WIFI_STA);
29   ThingSpeak.begin(client);
30 }
31
32 void loop() {
33   if ((millis() - lastTime) > timerDelay) {
34
35     // Connect or reconnect to WiFi
36     if (WiFi.status() != WL_CONNECTED) {
37       Serial.print("Connecting to WiFi");
38       while (WiFi.status() != WL_CONNECTED) {
39         WiFi.begin(ssid, password);
40         delay(5000);
41
42       }
43     }
44   }
45 }
```

Output Serial Monitor

```
Writing at 0x00000000... (7 %)
Writing at 0x00004000... (15 %)
Writing at 0x00008000... (23 %)
Writing at 0x0000c000... (30 %)
Writing at 0x00010000... (38 %)
Writing at 0x00014000... (46 %)
Writing at 0x00018000... (53 %)
Writing at 0x0001c000... (61 %)
Writing at 0x00020000... (69 %)
Writing at 0x00024000... (76 %)
Writing at 0x00028000... (84 %)
Writing at 0x0002c000... (92 %)
Writing at 0x00030000... (100 %)
Wrote 283776 bytes (20157 compressed) at 0x00000000 in 5.5 seconds (effective 412.5 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

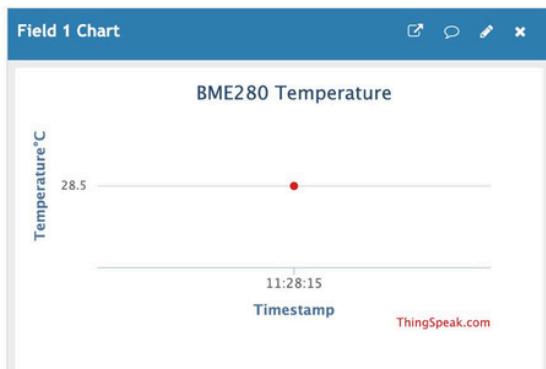
Step 5:

We can now see the published values on the ThingSpeak platform. to our created channel. One value and then i waited for some time so that more values get published

Channel Stats

Created: 27.minutes.ago

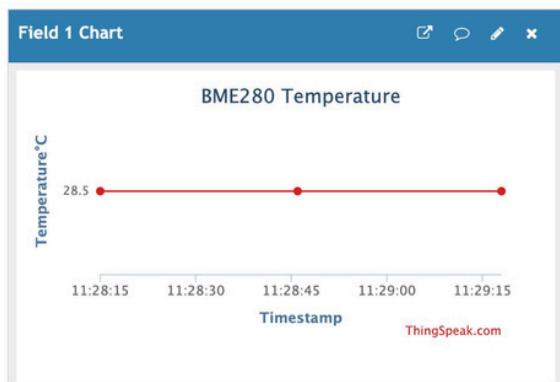
Entries: 1



Channel Stats

Created: 28.minutes.ago

Entries: 3



Step 6:

Modifying it for multiple fields (temperature, pressure, humidity). This time i used the random function to generate random values of these three variables and published it to ThingSpeak channel.

[Private View](#)[Public View](#)[Channel Settings](#)[Sharing](#)[API Keys](#)[Data Import / Export](#)

Channel Settings

Percentage Complete 50%

Channel ID 2916321

Name BME280Readings

Description Readings from BME280 (ESP8266)

Field 1 Temperature Field 2 Humidity Field 3 Pressure

Help

Channels store all the data that can hold any type of data, plus in a channel, you can use Thing

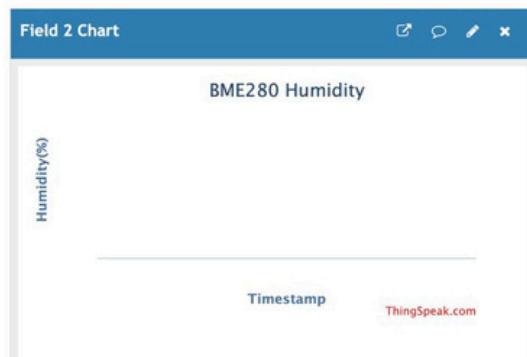
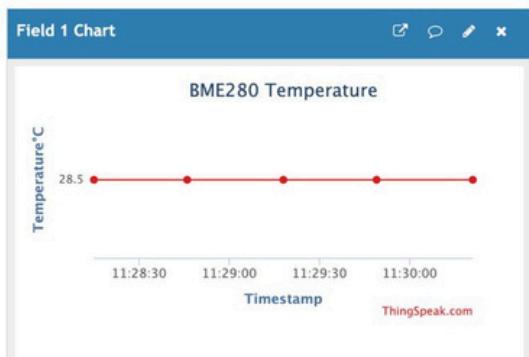
Channel Settings

- Percentage complete: Cal name, description, locati
- Channel Name: Enter a ur
- Description: Enter a descr
- Field#: Check the box to e up to 8 fields.
- Metadata: Enter informati
- Tags: Enter keywords that

Changed the names of x and y-axis according to my variables.

Last entry: 4 minutes ago

Entries: 5



Step 7: Next in the arduino IDE i wrote the following code for 3 fields.

```
#include <ESP8266WiFi.h>
#include "ThingSpeak.h"

// WiFi credentials
const char* ssid = "WajeehaPhone";
const char* password = "987684";

// ThingSpeak settings
unsigned long myChannelNumber = 1;
const char* myWriteAPIKey = "68LKDJ8K6FW5DNNN";

WiFiClient client;

// Timer
unsigned long lastTime = 0;
unsigned long timerDelay = 10000; // 10 seconds

void setup() {
    Serial.begin(115200);
    WiFi.mode(WIFI_STA);
    ThingSpeak.begin(client);
    randomSeed(analogRead(A0)); // Seed random number generator
}

void loop() {
    if ((millis() - lastTime) > timerDelay) {
```

```
// Connect to WiFi if not connected
if (WiFi.status() != WL_CONNECTED) {
    Serial.print("Connecting to WiFi");
    while (WiFi.status() != WL_CONNECTED) {
        WiFi.begin(ssid, password);
        delay(5000);
        Serial.print(".");
    }
    Serial.println("\nConnected to WiFi");
}

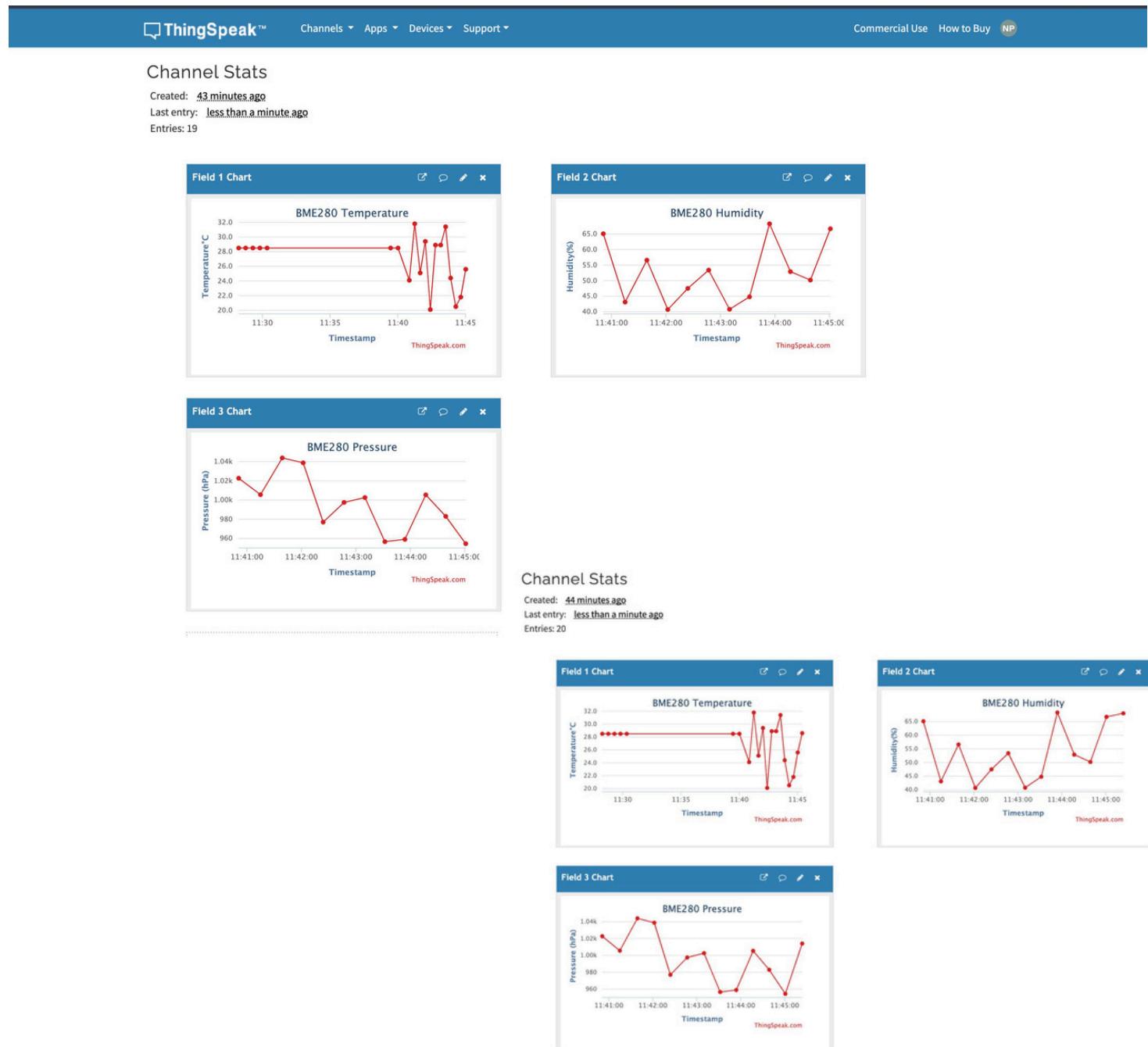
// Generate random values
float temperature = random(200, 350) / 10.0; // 20.0 to 35.0 °C
float humidity = random(400, 700) / 10.0; // 40.0 to 70.0 %
float pressure = random(9500, 10500) / 10.0; // 950.0 to 1050.0 hPa
// Log to Serial
Serial.println("Sending to ThingSpeak:");
Serial.print("Temperature: "); Serial.println(temperature);
Serial.print("Humidity: "); Serial.println(humidity);
Serial.print("Pressure: "); Serial.println(pressure);
// Send values to corresponding fields
ThingSpeak.setField(1, temperature);
ThingSpeak.setField(2, humidity);
ThingSpeak.setField(3, pressure);
int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
if (x == 200) {
    Serial.println("Channel update successful.\n");
} else {
    Serial.println("Problem updating channel. HTTP error code " + String(x) + "\n");
}
lastTime = millis();
}
}
```

Output

```
Writing at 0x00000000... (7 %)
Writing at 0x00000400... (15 %)
Writing at 0x00000800... (23 %)
Writing at 0x0000c000... (30 %)
Writing at 0x00010000... (38 %)
Writing at 0x00014000... (46 %)
Writing at 0x00018000... (53 %)
Writing at 0x0001c000... (61 %)
Writing at 0x00020000... (69 %)
Writing at 0x00024000... (76 %)
Writing at 0x00028000... (84 %)
Writing at 0x0002c000... (92 %)
Writing at 0x00030000... (100 %)
Wrote 285232 bytes (209161 compressed) at 0x00000000 in 5.5 seconds (effective 414.3 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

After some time, multiple values have been published to channel, which are shown here.



Serial Monitor:

```
Output Serial Monitor X

Message (Enter to send message to 'LOLIN(WEMOS) D1 R2 & mini' on '/dev/cu.usbserial-14110')

Connected to WiFi
Sending to ThingSpeak:
Temperature: 24.10
Humidity: 65.10
Pressure: 1022.70
Channel update successful.

Sending to ThingSpeak:
Temperature: 33.50
Humidity: 47.00
Pressure: 950.40
Problem updating channel. HTTP error code -401

Sending to ThingSpeak:
Temperature: 31.80

dexing: 70/86
```

```
7 // ThingSpeak settings
8 unsigned long myChannelNumber = 1;
9 const char* myWriteAPIKey = "68LKDJ8K6FW5DNNN";
10
11 WiFiClient client;
12
13 // Timer
14 unsigned long lastTime = 0;
15 unsigned long timerDelay = 10000; // 10 seconds
16
17 void setup() {
18   Serial.begin(115200);
19   WiFi.mode(WIFI_STA);
20   ThingSpeak.begin(client);
21   randomSeed(analogRead(A0)); // Seed random number generator
22 }
23
24 void loop() {
25   if ((millis() - lastTime) > timerDelay) {
26
27     // Connect to WiFi if not connected
28     if (WiFi.status() != WL_CONNECTED) {
29       Serial.print("Connecting to WiFi");
30       while (WiFi.status() != WL_CONNECTED) {
31         WiFi.begin(ssid, password);
32         delay(5000);
33         Serial.print(".");
34       }
35     }
36   }
37 }
```

```
Output Serial Monitor X

Message (Enter to send message to 'LOLIN(WEMOS) D1 R2 & mini' on '/dev/cu.usbserial-14110')

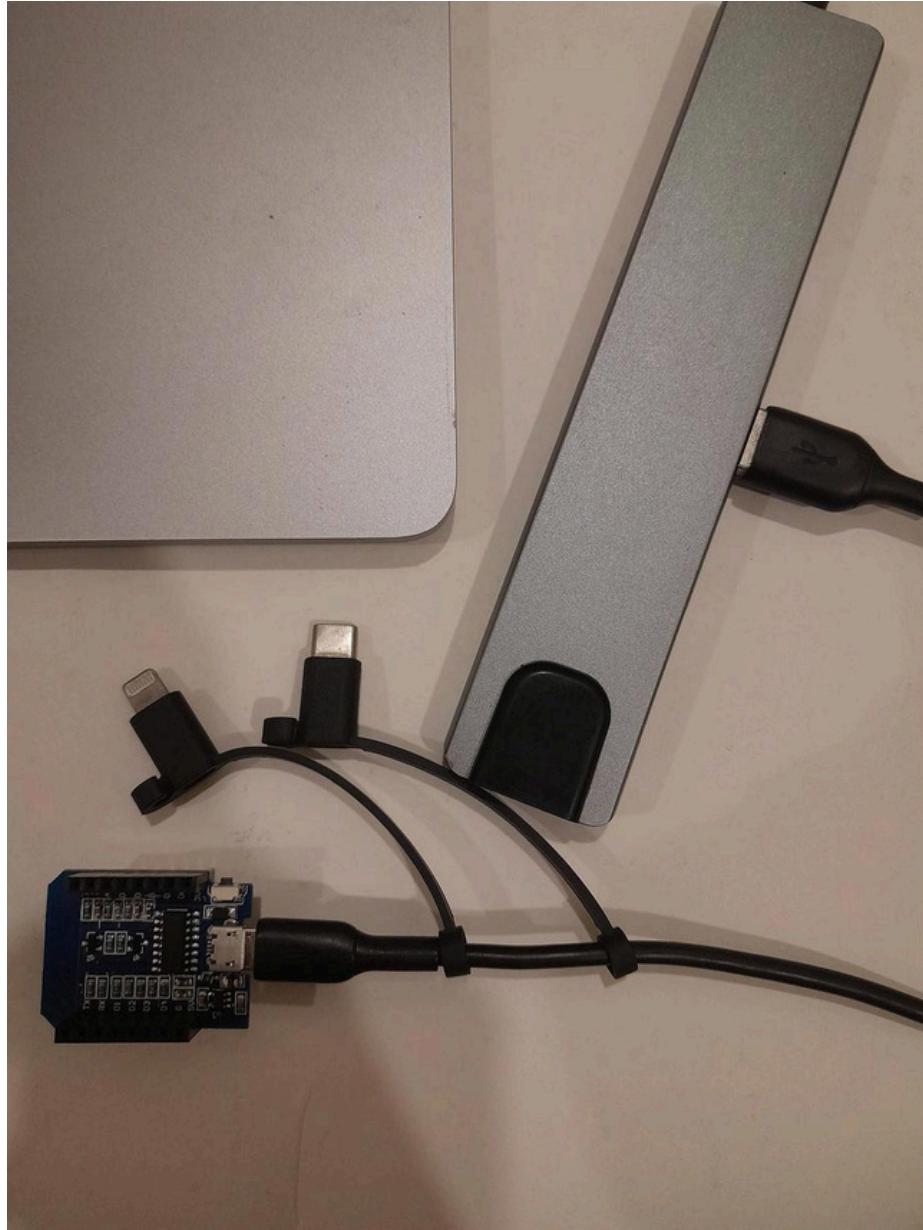
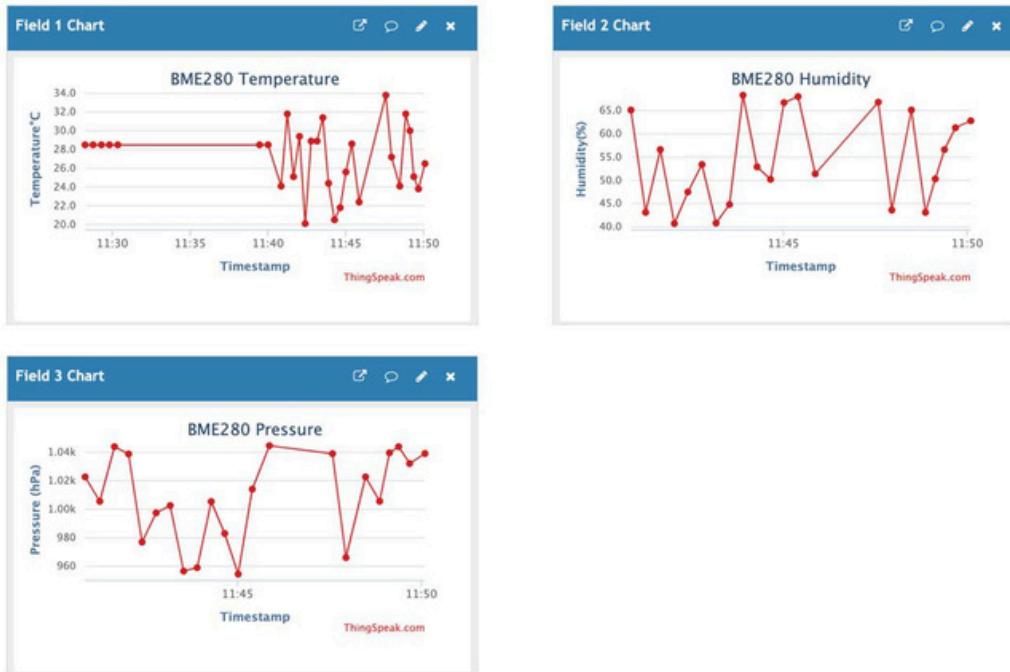
Sending to ThingSpeak:
Temperature: 31.80
Humidity: 43.10
Pressure: 1005.60
Channel update successful.

Sending to ThingSpeak:
Temperature: 30.00
Humidity: 50.30
Pressure: 1039.60
Channel update successful.

Sending to ThingSpeak:
Temperature: 25.10
Humidity: 56.60
Pressure: 1043.90
```

Channel Stats

Created: [about an hour ago](#)
Last entry: [less than a minute ago](#)
Entries: 28



Challenges:

All the lab steps were performed without any difficulty.

AWS IoT Core

Overview:

- Creating a Thing in the AWS, generating a certificate and attaching a policy to it.
- Installing ESP8266 sketch data upload tool in Arduino IDE.
- Modifications in the Arduino sketch according to the thing.
- Uploading AWS certificates & code to the NodeMCU ESP8266.
- Testing/Subscription of thing on Amazon Web Services (AWS).

Step 1:

Creating a Thing in the AWS, generating a certificate and attaching a policy to it.

Open browser and enter AWS console to visit AWS management console page. First you need to signup.

Search for IoT core in find services search bar.

Click on the IoT core that will redirect to the Amazon IoT console page. On the left navigation page click on manage tab.

Go for register a thing. Click on create a single thing. Set a name for the thing and click next. Click on create certificate. Once a certificate is created successfully download certificate, private key and root CA certificate. It will redirect to another tab where you will have to choose RSA 2048 bit key to download the certificate.

Come back to the AWS IoT page and click on activate and done. Certificate is created successfully.

Now go for secure tab and click on policy section and then click on create. Set a policy name and for action and resource section enter * to allow device to perform all aws actions on all aws resources. For effect choose allow and then click on create.

Once policy is created successfully click on the certificate tab and click on 3 dots of certificate file and go for attach policy. A popup will show you all the policies choose the right policy and then click on attach.

Thus we have created a thing and attached a policy and certificate to it successfully.

S Services Search [Alt+S] Europe (Stockholm) Urwah Rasheed

Manage

- All devices
- Software packages
- Remote actions
- Message routing
- Retained messages
- Security
- Fleet Hub

Device software

- Billing groups
- Settings

Manage

Manage your devices, IoT data, remote actions, security, and applications

Create and maintain AWS IoT things, interact with devices, and monitor the security of your devices.

Create a thing resource

AWS IoT thing resources represent your physical devices in the cloud and maintain a record of their properties and data. Thing resources also let your devices access AWS IoT services. Create a thing resource to register a single device with AWS IoT.

Create thing

AWS Services Search [Alt+S] Europe (Stockholm) Urwah Rasheed

AWS IoT > Manage > Things > Create things > Create single thing

Step 1
Specify thing properties

Step 2 - optional
Configure device certificate

Step 3 - optional
Attach policies to certificate

Specify thing properties Info

A thing resource is a digital representation of a physical device or logical entity in AWS IoT. Your device or entity needs a thing resource in the registry to use AWS IoT features such as Device Shadows, events, jobs, and device management features.

Thing properties Info

Thing name Enter a unique name containing only letters, numbers, hyphens, colons, or underscores. A thing name can't contain any spaces.

Additional configurations

You can use these configurations to add detail that can help you to organize, manage, and search your things.

AWS Services Search [Alt+S] Europe (Stockholm) Urwah Rasheed

AWS IoT

- Monitor
- Connect
 - Connect one device
 - Connect many devices
 - Domain configurations Updated
- Test
 - MQTT test client
 - Query connectivity status New

Tags - optional

Policy statements Policy examples

Policy document Info Builder | JSON

An AWS IoT policy contains one or more policy statements. Each policy statement contains actions, resources, and an effect that grants or denies the actions by the resources.

Policy effect	Policy action	Policy resource
Allow	*	*

Add new statement Remove Cancel Create

AWS IoT > Security > Certificates > Create certificate

Create certificate Info

Certificates authenticate devices and clients so that they can connect to AWS IoT. Your device won't be able to connect to IoT without authentication and an appropriate policy.

Certificate

Certificate branch selection

Auto-generate new certificate (recommended)
Generate a new certificate, public key, and private key using AWS IoT's certificate authority and register it with AWS IoT.

Create certificate with certificate signing request (CSR)
Upload your own certificate signing request (CSR) file to create and register a certificate that's based on a private key you own.

Download certificates and keys

Download certificates and keys

Download and install the certificate and key files to your device so that it can connect securely to AWS IoT. You can download the certificate now, or later, but the key files can only be downloaded now.

Device certificate

ede1f34676f...te.pem.crt

[Download](#)

Key files

The key files are unique to this certificate and can't be downloaded after you leave this page.

Download them now and save them in a secure place.

⚠ This is the only time you can download the key files for this certificate.

Public key file

ede1f34676f7457d09fa69e...b4882c6-public.pem.key

[Download](#)

Private key file

ede1f34676f7457d09fa69e...4882c6-private.pem.key

[Download](#)

[Continue](#)

The screenshot shows the AWS IoT Certificates Details page. A green success message at the top states: "Successfully attached the policy ESP8266_Iotc to certificate ede1f34676f7457d09fa69ef8a70d7751e9c42ebc83c9fad9d14e0b9fb4882c6." Below this, the certificate details are listed:

Details	Value
Certificate ID	ede1f34676f7457d09fa69ef8a70d7751e9c42ebc83c9fad9d14e0b9fb4882c6
Status	Active
Created	April 13, 2025, 01:38:39 (UTC+05:00)
Valid	April 13, 2025, 01:36:39 (UTC+05:00)
Expires	January 01, 2050, 04:59:59 (UTC+05:00)
Subject	CN=AWS IoT Certificate
Issuer	OU=Amazon Web Services O=Amazon.com Inc. L=Seattle ST=Washington C=US

Step 3:

Installing Necessary Arduino Libraries

So first go to the library manager and search for “JSON” & install the library as shown in the figure below.

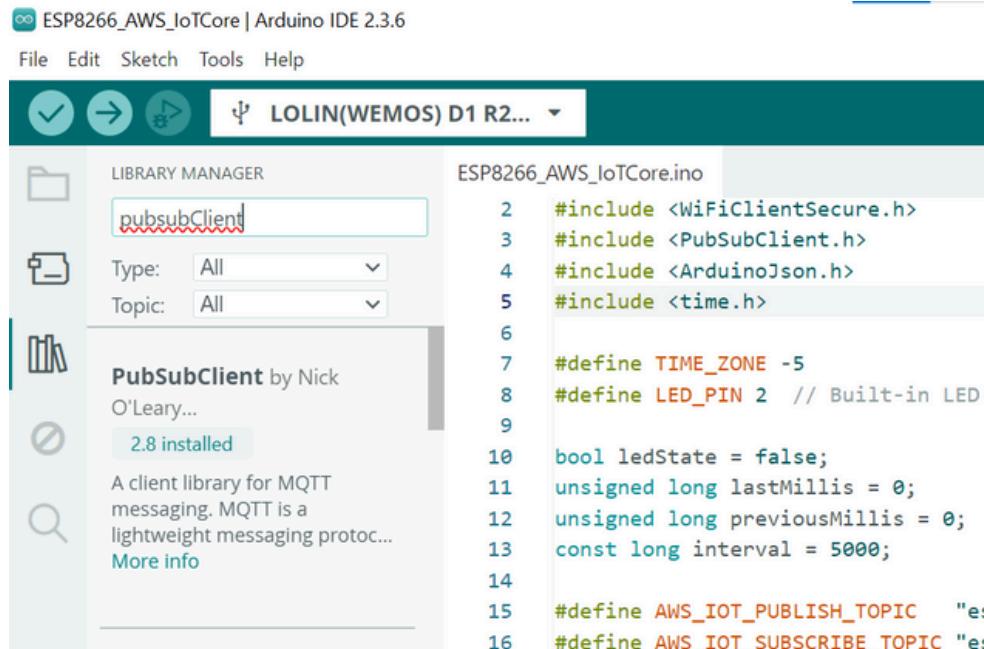
The screenshot shows the Arduino IDE Library Manager. The search bar at the top contains "json". The results list shows the "ArduinoJson" library by Benoit Blanchon, version 7.4.1, which is installed. The main code editor window shows a portion of the "ESP8266_AWS_IoTCore.ino" sketch, specifically the WiFi and AWS IoT configuration code.

```
#define TIME_ZONE -5
#define LED_PIN 2 // Built-in LED on GPIO2 (D4 on most Es
bool ledState = false;
unsigned long lastMillis = 0;
unsigned long previousMillis = 0;
const long interval = 5000;

#define AWS_IOT_PUBLISH_TOPIC "esp8266/pub"
#define AWS_IOT_SUBSCRIBE_TOPIC "esp8266/sub"

// WiFi and AWS IoT configuration
const char* WIFI_SSID = "eman's S24 Ultra";
const char* WIFI_PASSWORD = "12345678";
const char* THINGNAME = "ESP8266";
const char* MQTT_HOST = "aen6j86nfbv4-ats.iot.eu-north-1.
```

Again go to the library manager and search for “PubSubClient” & install the library from Nick O’Leary.



Source Code/Program for connecting AWS IoT Core with ESP8266

```
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <time.h>

#define TIME_ZONE -5
#define LED_PIN 2 // Built-in LED on GPIO2 (D4 on most ESP8266 boards)

bool ledState = false;
unsigned long lastMillis = 0;
unsigned long previousMillis = 0;
const long interval = 5000;

#define AWS_IOT_PUBLISH_TOPIC "esp8266/pub"
#define AWS_IOT_SUBSCRIBE_TOPIC "esp8266/sub"

// WiFi and AWS IoT configuration
const char* WIFI_SSID = "eman's S24 Ultra";
const char* WIFI_PASSWORD = "12345678";
const char* THINGNAME = "ESP8266t";
const char* MQTT_HOST = "aen6j86nfblv4-ats.iot.eu-north-1.amazonaws.com";
// AWS IoT certificates
static const char cacert[] PROGMEM = R"EOF(
```

-----BEGIN CERTIFICATE-----

MIIIDQTCCAimgAwIBAgIBMyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF
ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6
b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAwMFowOTEL
MAkGA1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJv
b3QgQ0EgMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALJ4gHHKeNXj
ca9HgFB0fW7Y14h29Jlo91ghYPl0hAEvrAlhtOgQ3pOsqTQNroBvo3bSMgHFzZM
9O6lI8c+6zf1tRn4SWiw3te5djgdYZ6k/oI2peVKVuRF4fn9tBb6dNqcmzU5L/qw
IFAGbHrQgLKm+a/sRxmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUcAwhmahRWa6
VOujw5H5SNz/0egwLX0tdHA1I4gk957EW67c4cX8jJGKLhD+rcdqsq08p8kDiL
93FcXmn/6pUCyziKrlA4b9v7LWIbxccEVOF34GfID5yHI9Y/QCB/IIDEgEw+OyQm
jgSubJrlqg0CAwEAAsNCMEAwDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMC
AYYwHQYDVR0OBByEFIQYzIU07LwMIJQuCFmcx7IQTgoIMA0GCSqGSIb3DQEBCwUA
A4IBAQCY8jdaQZChGsV2USggNiMOruYou6r4IK5lpDB/G/wkjUu0yKGX9rbxenDI
U5PMCCjjmCXPI6T53iHTfIUJrU6adTrCC2qJeHZERxhlblBjjt/msv0tadQ1wUs
N+gDS63pYaACbvXy8MWy7Vu33PqUXHeeE6V/Uq2V8viTO96LXFvKWlJbYK8U90vv
o/ufQJVtMVT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1bldZwgJcJmApzyMFo6IQ6XU
5MsI+yMRO+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXeGADbkpy
rqXRfboQnoZsG4q5WTP468SQvvG5

-----END CERTIFICATE-----

)EOF";

static const char client_cert[] PROGMEM = R"KEY(

-----BEGIN CERTIFICATE-----

MIIDWjCCAkKgAwIBAgIVAMlhSbr7KpTZFgHt8EEuoDfDghswMA0GCSqGSIb3DQEBr
CwUAME0xSzBJBgNVBAsMQkFtYXpvbiBXZWlgU2VydmljZXmgTz1BbWF6b24uY29t
IEluYy4gTD1ZWFOdGxIIFNUPVdhc2hpmd0b24gQz1VUzAeFw0yNTA0MTMxNzI3
MDdaFw00OTEyMzEyMzU5NTlaMB4xHDAaBgNVBAMME0FXUyBJb1QgQ2VydGlmaWNh
dGUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQczmdl6GwE81pqwFqH1
R9FLywulbq4WQWX/9ZYnVVA60wRgvwVEIDePFu/hIJDKG/57/ob7HB5t+pGYGmX
iMdQRQbdb1T8wf7B8bvYMcm0+PhxaGP1viBf4mGfQnNzLcNDJMrAB5VfpHj6hAZC
+i2uytMzQIF8Om6qpVW6DwtXv5ZaLLc78rtqnmlL7jgEeH6fSyHs46ftsatnhKQ
LQiLvejw36aV5L60Zvb9aT6Jgc2Y15eLCZc3dNeBlyHBrSPofVEf3UZLjWArEJS
F5G/OWta0Gh7K9CAIYM0UWPa4ZQxIRvXBgu8K/iHUOJquIoBUYj/2UMDoxAptB1
89UxAgMBAAGjYDBeMB8GA1UdlwQYMBaAFHuMMxtITnRFQGSklaa4QrwFQGQjMB0G
A1UdDgQWBbTv1kRe5RqSfoZC9/UKhQ5VHfxF1DAMBgNVHRMBAf8EAjAAMA4GA1Ud
DwEB/wQEAwIhgDANBgkqhkiG9w0BAQsFAAOCAQEAOA3TyymVPqpPO1KFTJbiCm3f
w+dupstEmyC01PPDZsn1zyXRslukubdIKeTRTGIHPA+y/0IAFov7//Fy2yvIVGU
dQEm7nqRtKD8aG8T7fdwP8u0mS5mmuiP+iHWxEWj4fhfV9i+FdPNaZV+UfX0IWxk
6xBC9T/WY9lJxi5GHt2BOgu3Lza4Bzjp4lVRzn3Z8tMmLFxHbpqJWyyym8oPc+v
UPugdFLaqzChP0HCdTmucE/QAb1mcxFoogQmwXskyNwMDExau2DyiC5QFUECnfTw
h/1knVJTYCjVZ9V6aMGSGZg5yanCz1D6N+bGU9RIBDDZ9dqNEiNOST7wVnB5iA==

-----END CERTIFICATE-----

)KEY";

```

static const char privkey[] PROGMEM = R"KEY(
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAs5nZehsBPNaasBah9UfRS8sLi26uFkFI//WWJ1VQOtMEYL8F
RJQ3jxbv4SCSQyhv+e/6G+xwebfqRmBpl4jHUEUG3W9U/MH+wfG72DHjtPj4cWhj
9b4gX+Jhn0Jzcy3DQyTKwAeVX6R4+oQGQvotrsrTM0CBfDpuqqVVug8LV7+WWiy3
O/K7ap5opS+44BHh+n0sh7OOOn7bGrZ4SkC0li73o8N+mleS+tGb2/Wk+qyYHNmNe
XiwmXN3TXgdchwa0j6H1RH91GS41gKxCUheRvzlrbWtBoeyvQgJWDNFFj2uGUMZU
b1wYLvCv4h1DiarpaAVGI/9lDA6MQKbQdfPVMQIDAQABAoIBAEz+NGQLNaGellxA
p5ed/RRv6/gPL6QszwxGfONJyIYfNi4/VYPLwnGeFKG/M9SEflR4UxErcCBN/qnC
5SFoKoG3xbSh6J4ekBcWiT+qQFlLqj370XZk1j4kr1L1ysZipOWg3SYrWhbRTGeg
t4p486KOlrJDAf9+kTiofjKwBl3L9U/1nnVVS1YU7siUXm8pa3BkUxCgUAOUMiPL
VbTiL3aQ4KzIE6S1A9XoeFyZ2GG8ZVbCk3v82WWpqfZfJqteICpgx03HLwLo5/3T
Vzq0Z456tK4FfbG5Indi1fcq+ox0uBTD9VIVFlhyyVSpMELZcrDtZgGi1AJdV
yjiqJp0CgYEA1+9jpt8fUTtAD9YH7UvwPlb1LaKYH94bRrjYZBp1c6vwKQRj6A7T
B2MJzFFX3hvjlcuqBRgr6Yz7iD+HnhWNqErAz74V6sb1MbpLFrUv/+XBmfHQbVXP
P5T/6WclfiWORql/U/u+sL3vCl3jkxxWGwEe1EKo0guHnnoPYfFCA+8CgYEA1Oyj
63pXfmIEhcpJzQXJN/JGJ2OM1YM7+QkkZUL5RDC0neTN5MW8MhDEP2RdN033mCoY
AqlyHXZZdnuH6Gh7MYhFm8+q90pTlc9xY6dPZA/C33a/vEwF8RTLT/b3S316YCB1
Ka/AD1n7Y3ZWEklZuJ8Kw9aT2OPeXKySzpAsGN8CgYACdu0APpOw7agxhhPZFYDM
cA1g1/Y8huBw0jNGETc99rxq+23YfdLUzsD54APMuSzCefyaykfWBpHoYDFAEKbp
QNEqjdTaNSLnxmSNAcxmc0zeYPAsD0qqQx0YoOBr8CnLjJaEct8eTUyplFV0nGlv
NLgRXCFpJFCL8oCycXp7aQKBgHzXyzzUXNyTaAdyFleJ3vNYsF2D9x1xdwaLn7Vn
0XN0A1LYxwguECxivE5W8Hju1A96Dt3zClfRzp6zy3ovWJwwRAZESsfxDB6r/9
Z4VJ4H7Zx44GHS2/fX4DVfgDOG+IWGn6zGez4LvutIAUUT4q30sIT/4S3aXEPTg
f5qNAoGAeNsnNb9lbPx9GrsjMPtwiGpX+AwG2CBvURA5zdfYolWWCgimqOwkXrDz
Ce5A1b+48LP0PI0ndn7bS+58YRZeUEdget7tUX9cxwiKFxMSrB4xQThHOWf+1dMQ
zP7IV/gDI48g2UfyL3WnTKkB/aRLvDIOcmWAuTHZdQJT9AcARPE=
-----END RSA PRIVATE KEY-----
)KEY";
WiFiClientSecure net;
BearSSL::X509List cert(cacert);
BearSSL::X509List client_crt(client_cert);
BearSSL::PrivateKey key(privkey);
PubSubClient client(net);

time_t now;
time_t nowish = 1510592825;

void NTPConnect(void) {
  Serial.print("Setting time using SNTP");
  configTime(TIME_ZONE * 3600, 0 * 3600, "pool.ntp.org", "time.nist.gov");
  now = time(nullptr);
}

```

```

while (now < nowish) {
    delay(500);
    Serial.print(".");
    now = time(nullptr);
}
Serial.println("done!");
struct tm timeinfo;
gmtime_r(&now, &timeinfo);
Serial.print("Current time: ");
Serial.print(asctime(&timeinfo));
}

void messageReceived(char *topic, byte *payload, unsigned int length) {
    Serial.print("Received [");
    Serial.print(topic);
    Serial.print("]: ");
    for (int i = 0; i < length; i++) {
        Serial.print((char)payload[i]);
    }
    Serial.println();
}

// Optional: Add code here to handle incoming messages if needed
}

void connectAWS() {
    delay(3000);
    WiFi.mode(WIFI_STA);
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

    Serial.println(String("Attempting to connect to SSID: ") + String(WIFI_SSID));

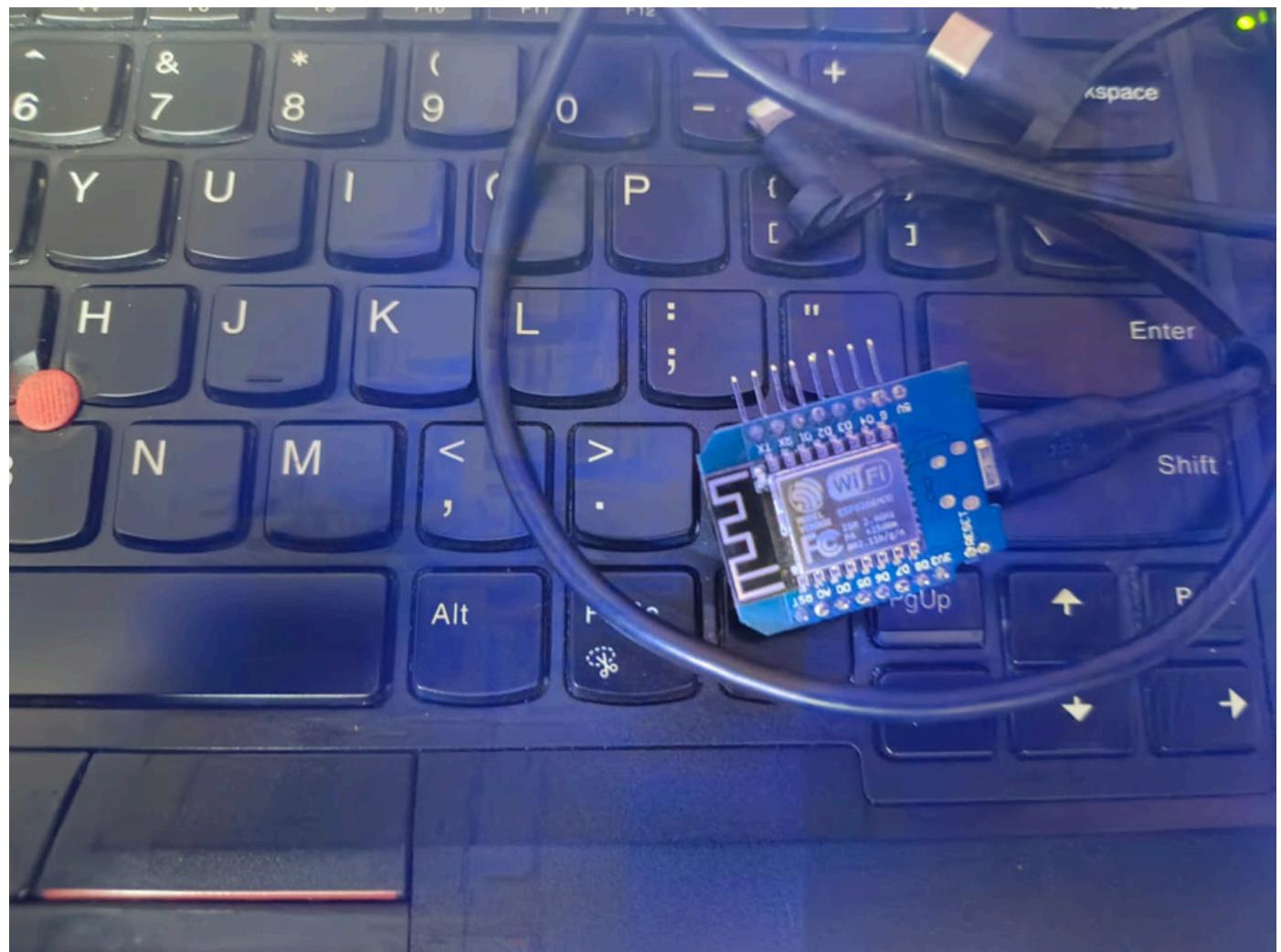
    while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        delay(1000);
    }

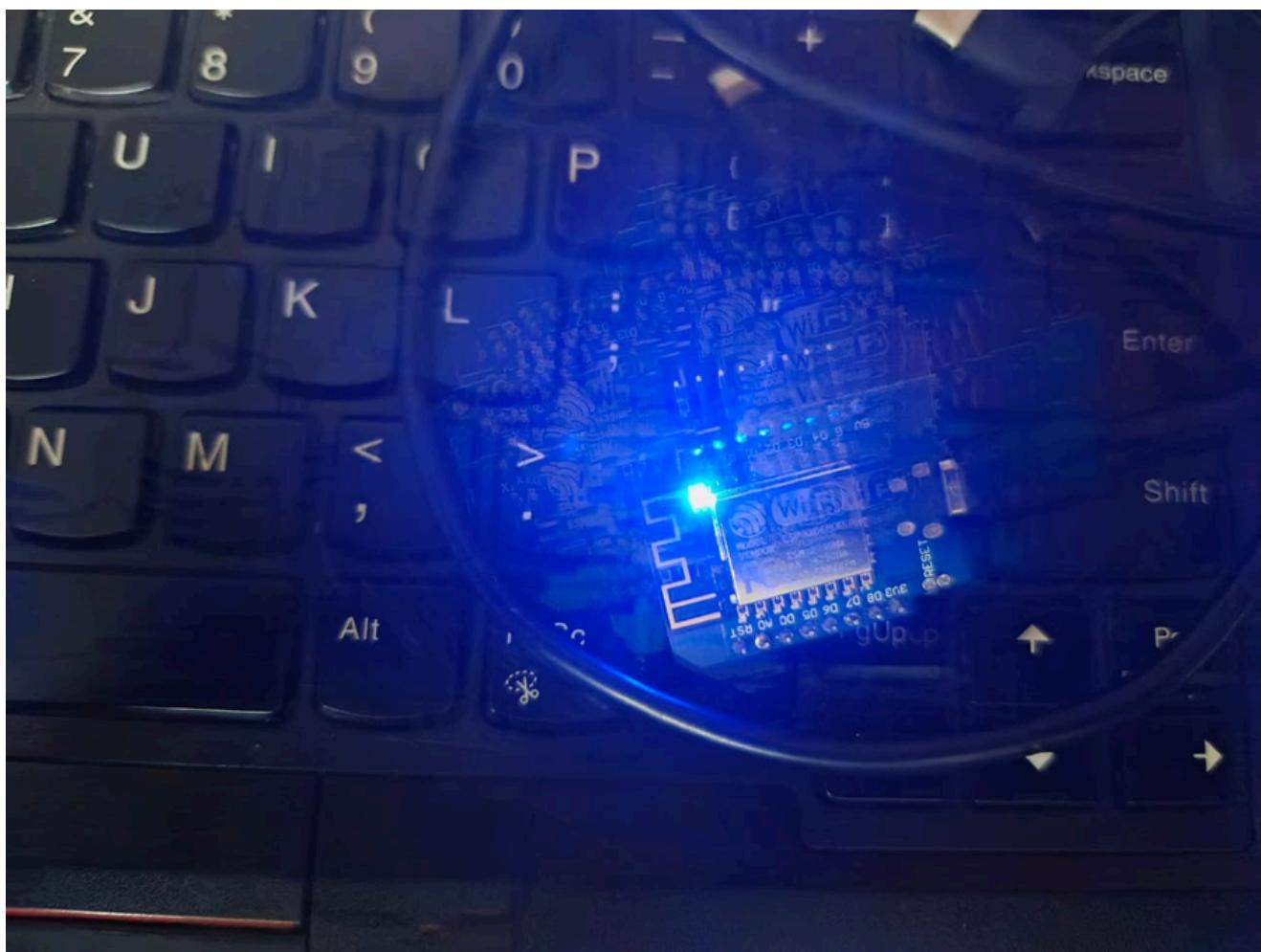
    NTPConnect();
    net.setTrustAnchors(&cert);
    net.setClientRSACert(&client_crt, &key);
    client.setServer(MQTT_HOST, 8883);
    client.setCallback(messageReceived);
    Serial.println("Connecting to AWS IOT");
    while (!client.connect(THINGNAME)) {
        Serial.print(".");
        delay(1000);
    }
}

```

```
if (!client.connected()) {  
Serial.println("AWS IoT Timeout!");  
return;  
}  
  
client.subscribe(AWS_IOT_SUBSCRIBE_TOPIC);  
Serial.println("AWS IoT Connected!");  
}  
  
void publishMessage() {  
StaticJsonDocument<200> doc;  
doc["time"] = millis();  
doc["led_state"] = ledState ? "ON" : "OFF";  
doc["message"] = "LED status update";
```

Circuit/Connected ESP8266- didn't have sensors avaialble.





Serail Monitor:

```
Attempting to connect to SSID: eman's S24 Ultra
.....Setting time using SNTP.....done!
Current time: Sun Apr 13 18:39:23 2025
Connecting to AWS IOT
AWS IoT Connected!
LED is now ON
```

```
LED is now ON
```

```
Published LED status to AWS IoT
```

```
LED is now OFF
```

```
LED is now ON
```

```
LED is now OFF
```

```
LED is now ON
```

```
LED is now ON
```

```
Published LED status to AWS IoT
```

```
Received [esp8266/sub]: {
```

```
    "message": "Hello from AWS IoT console"
```

```
}
```

```
LED is now OFF
```

AWS IoT Platform: Values getting published

Subscribe to a topic Publish to a topic

Topic filter [Info](#)
The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.

esp8266/pub

► Additional configuration

Subscribe

Subscriptions esp8266/pub

Message payload

```
{  
  "message": "Hello from AWS IoT console"  
}
```

[Alt+S] Pause Clear Export Edit

Europe (Stockholm) ▾ Urwah Rasheed

▼ esp8266/pub April 13, 2025, 23:17:29 (UTC+0500)

```
{  
  "time": 850186,  
  "led_state": "OFF",  
  "message": "LED status update"  
}
```

► Properties

▼ esp8266/pub April 13, 2025, 23:17:24 (UTC+0500)

```
{  
  "time": 845185,  
  "led_state": "ON",  
  "message": "LED status update"  
}
```

Subscribe to a topic **Publish to a topic**

Topic name
The topic name identifies the message. The message payload will be published to this topic with a Quality of Service (QoS) of 0.

Q esp8266/sub

Message payload

```
{  
  "message": "Hello from AWS IoT console"  
}
```

► Additional configuration

Publish

Critical Analysis and Comparison of Platforms:

By Areeb Ur Rehman

In this project, we have discussed three Internet of Things (IoT) platforms—Blynk, ThingSpeak, and AWS IoT Core. Each of them has unique strengths and weaknesses.

1. Blynk (Blynk 2.0):

Blynk is a platform designed to make IoT development easier, especially for people who are new to the field. It allows users to create mobile applications that can control microcontrollers like the ESP8266 or ESP32 over the internet.

Pros:

- **Very Beginner-Friendly:** Blynk is known for its simple drag-and-drop interface, especially on its mobile app. You don't need to write much code to create a working app to control your IoT device.
- **Fast Setup:** You can get a project running in minutes. Creating templates, adding widgets like switches or buttons, and linking them to your device is fast and smooth.
- **Mobile Control:** The platform shines when it comes to controlling IoT devices through your phone. The app interface is clean and highly responsive.
- **Virtual Pins:** These let you link the app interface directly to your code logic—like controlling an LED with a virtual switch.

Cons:

- **Restricted Free Tier:** Free accounts come with less functionality, fewer devices, and fewer templates. For more sophisticated features, a premium plan is required.
- **Unsuitable for Big Projects:** Blynk works well for prototypes but struggles to scale for intricate multi-device systems.
- **Basic Security:** Advanced security features like encryption keys and rules are not supported, despite the fact that it makes use of authentication tokens.
- **Strongly Connected to the Blynk Ecosystem:** Projects mostly depend on cloud services provided by Blynk. Very little can be done offline.

2. ThingSpeak:

ThingSpeak is an open-source IoT platform that focuses on collecting, storing, and visualizing data from devices. It's especially popular in educational and research settings.

Pros:

- **Great for Data Logging:** ThingSpeak is excellent if your project involves collecting sensor data and plotting it over time.
- **Built-in MATLAB Integration:** A powerful feature that allows you to analyze data directly in the cloud using MATLAB code.
- **Simple Dashboard Tools:** To keep an eye on data in real time, you can configure widgets, gauges, and charts.
- **Easy-to-Use API:** Using HTTP to upload data is straightforward and well-documented. There are numerous Arduino examples available.

Cons:

- **Old-School Interface:** The user interface can feel awkward and is out of date. Not very user-friendly for novices.
- **Requires MathWorks Account:** This is an additional step that may be perplexing, particularly if you are only doing a class project or doing an experiment.
- **Basic Control Features:** Unlike Blynk, ThingSpeak isn't really made for controlling devices—it's more about monitoring and analytics.
- **Limited Customization of UI Elements:** You can't build complex or interactive dashboards easily.

3. AWS IoT Core:

AWS IoT Core is Amazon's cloud-based platform for managing and communicating with IoT devices. It's a professional-grade service designed for real-world, large-scale deployments.

Pros:

- **Highly Scalable:** AWS IoT Core is built for serious business and industrial applications. It can handle thousands—even millions—of devices.
- **Advanced Security:** Assures safe connection between devices and the cloud by utilizing policies, private keys, and certificates.

- **Strong Integration Options:** AWS IoT Core can be connected to other services such as SNS (for notifications), DynamoDB (for data storage), and AWS Lambda (for automation).
- **Supports Standard Protocols:** Supports Standard Protocols: It is adaptable to a variety of devices due to its strong support for MQTT and HTTPS.

Cons:

- **Not for Beginners:** The platform is very technical and assumes a strong understanding of networking, security, and cloud architecture.
- **Complex Setup Process:** Even for something as simple as sending a message, there are numerous processes involved in creating a "Thing," adding a certificate, and converting keys to DER format.
- **Requires External Tools:** You may need to use tools like OpenSSL, the AWS CLI, and Arduino IDE plugins just to get everything working.
- **Costs Add Up:** While AWS has a free tier, when your device count or data usage increases, costs can increase.

END