Pakistan Institute of Engineering and Applied Sciences

Report

Assignment-01: Implementing Leading IoT Cloud Platforms with ESP8266/ESP32

Group Members:

- M. Hammad Tahir
- Ehmaan Shafqat
- Khadija Arif
- Ali Raza

Course Name: Practical IoT

Instructor: Dr. Naveed Akhtar

Submission Date: 13/04/2025

Objectives:

The goal of this assignment is to provide students with hands-on experience in integrating the ESP8266/ESP32 microcontroller with three prominent IoT cloud platforms: AWS IoT Core, Blynk, and ThingSpeak. Students will work in groups of four to explore the unique features and capabilities of each platform, develop applications, and document their findings.

Required Components:

- WeMos D1 Mini (ESP8266) / ESP 32
- Breadboard
- Jumper Wires
- USB cable for programming

Platform Integration

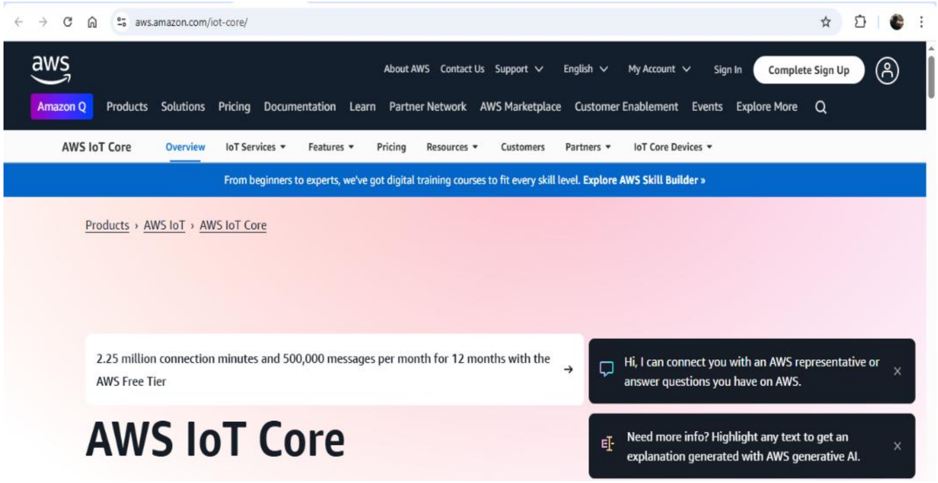
1. AWS IoT Core

Overview of AWS IoT Core

AWS IoT Core is a fully managed cloud service for IoT devices and teams. This innovative product enables organizations to connect, control, and grow their device fleets—while abstracting away difficult, time-consuming operational tasks such as provisioning servers or configuring encryption protocols.

AWS Console Setup (Thing, Certificate, Policy)

Visit following link: aws.amazon.com/iot-core/.



From beginners to experts, we've got digital training courses to fit every skill level. **Explore AWS Skill Builder »**

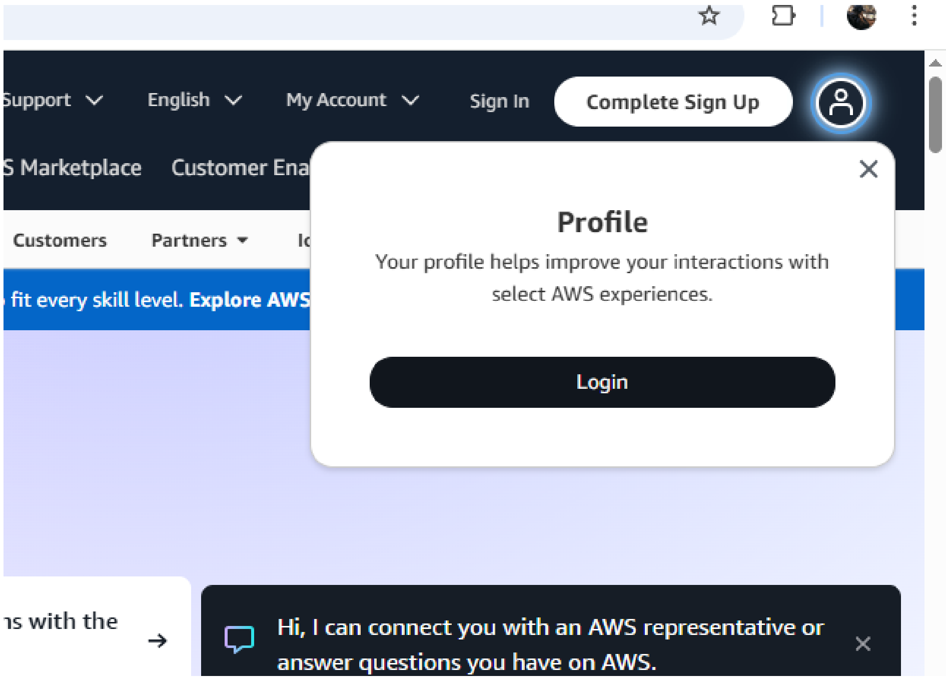
2.25 million connection minutes and 500,000 messages per month for 12 months with the AWS Free Tier

AWS IoT Core

Hi, I can connect you with an AWS representative or answer questions you have on AWS.

Need more info? Highlight any text to get an explanation generated with AWS generative AI.

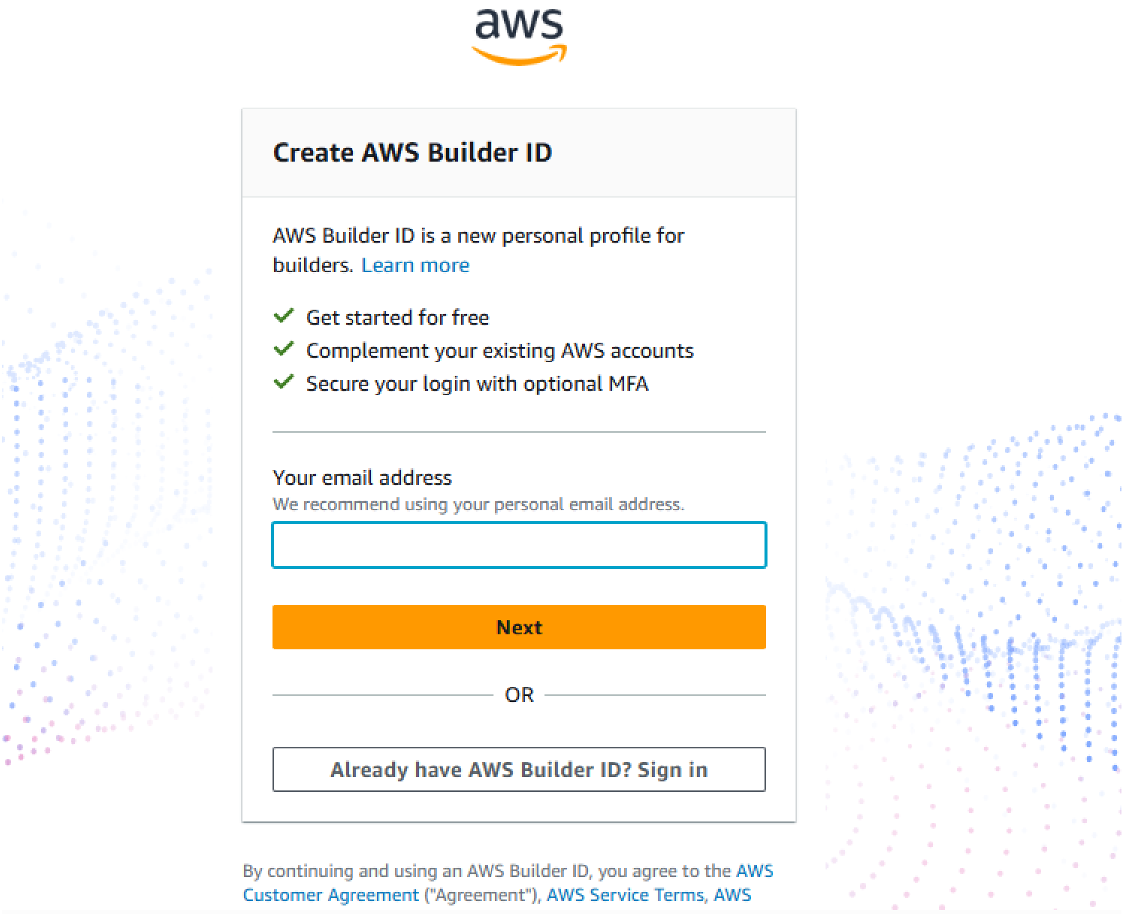
Create an account on clicking “**Complete Sign Up**”



Profile

Your profile helps improve your interactions with select AWS experiences.

Hi, I can connect you with an AWS representative or answer questions you have on AWS.



Create AWS Builder ID

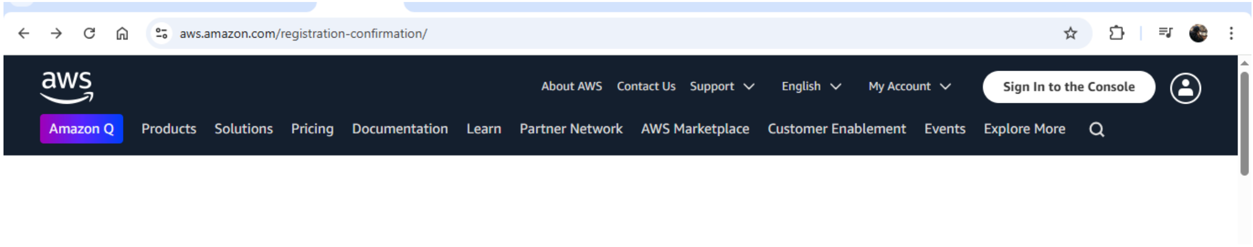
AWS Builder ID is a new personal profile for builders. Learn more

✓ Get started for free
✓ Complement your existing AWS accounts
✓ Secure your login with optional MFA

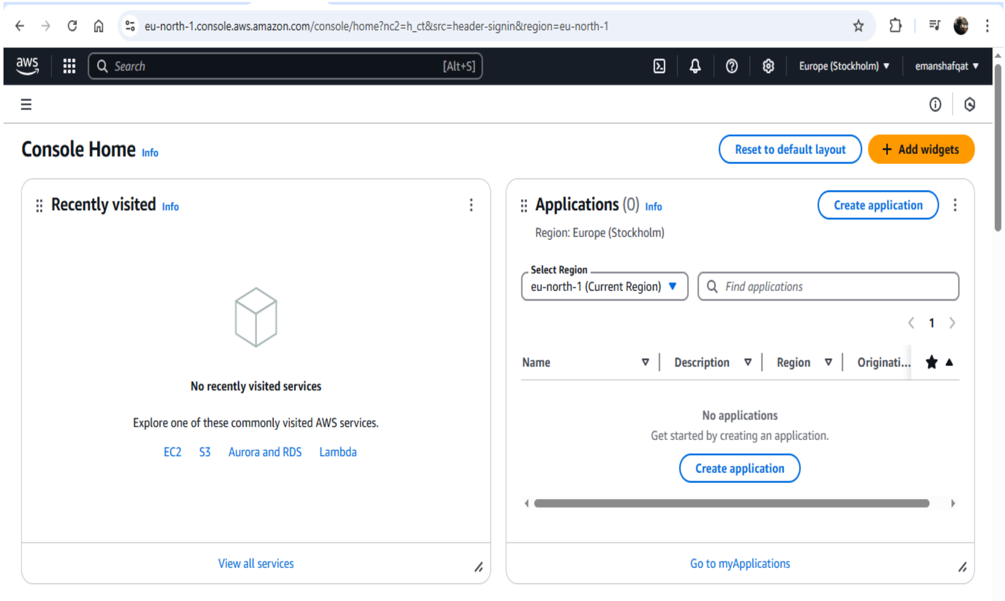
Your email address
We recommend using your personal email address.

By continuing and using an AWS Builder ID, you agree to the AWS Customer Agreement ("Agreement"), AWS Service Terms, AWS

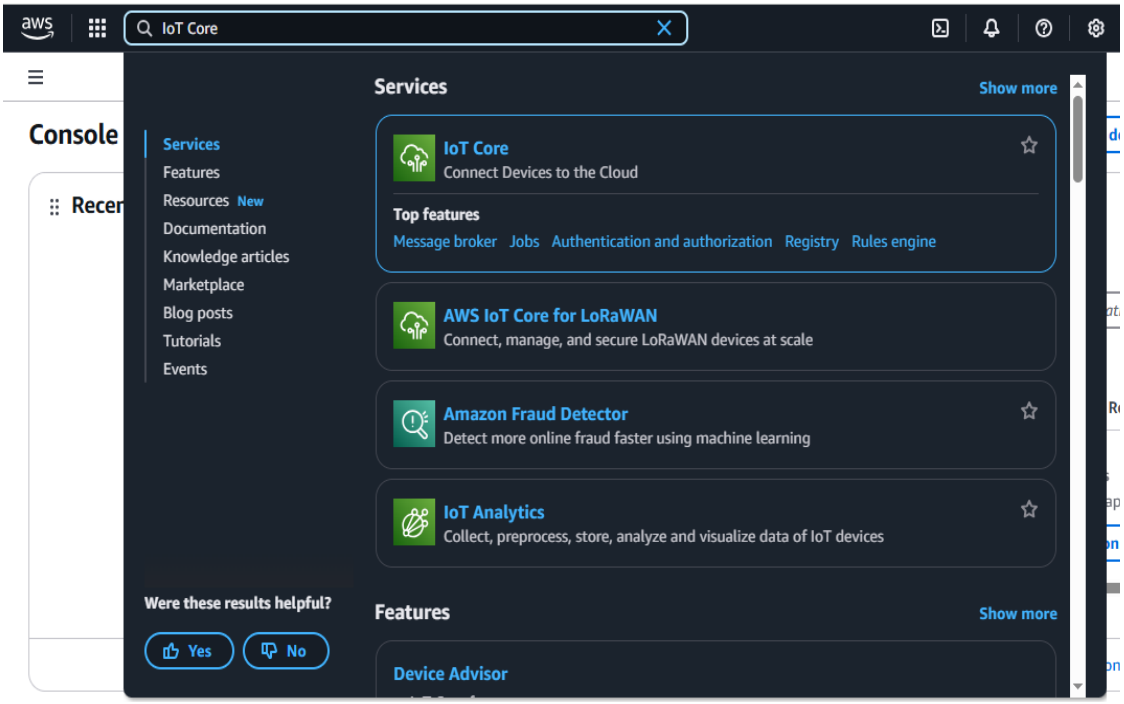
After creating account, sign in to the console.

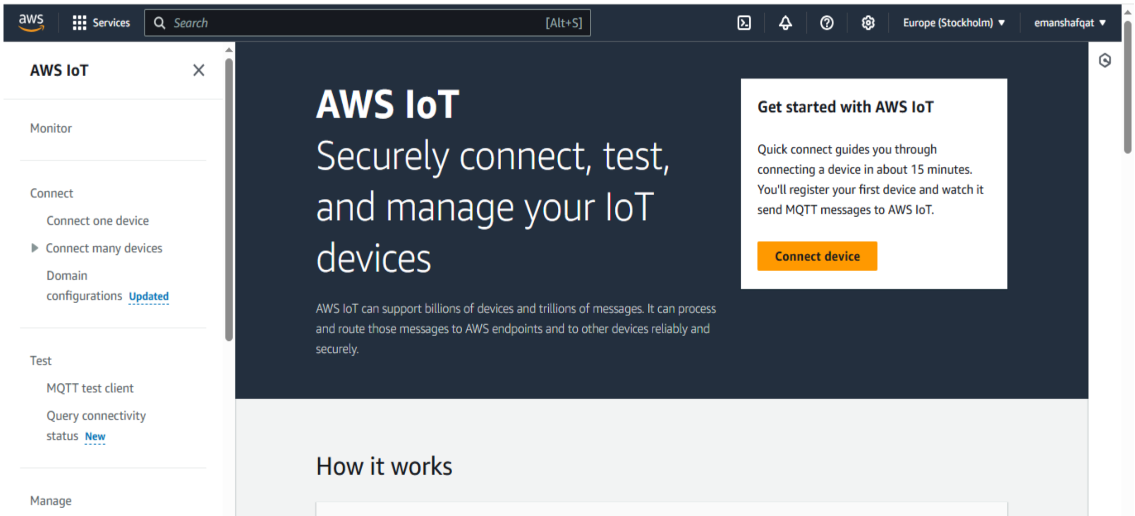


After successfully signing in, the AWS Management Console window will open. In the services search tab at the top write ‘IoT core’ & hit enter.



You can click on IoT Core, so an AWS IoT Dashboard will appear now.





AWS IoT

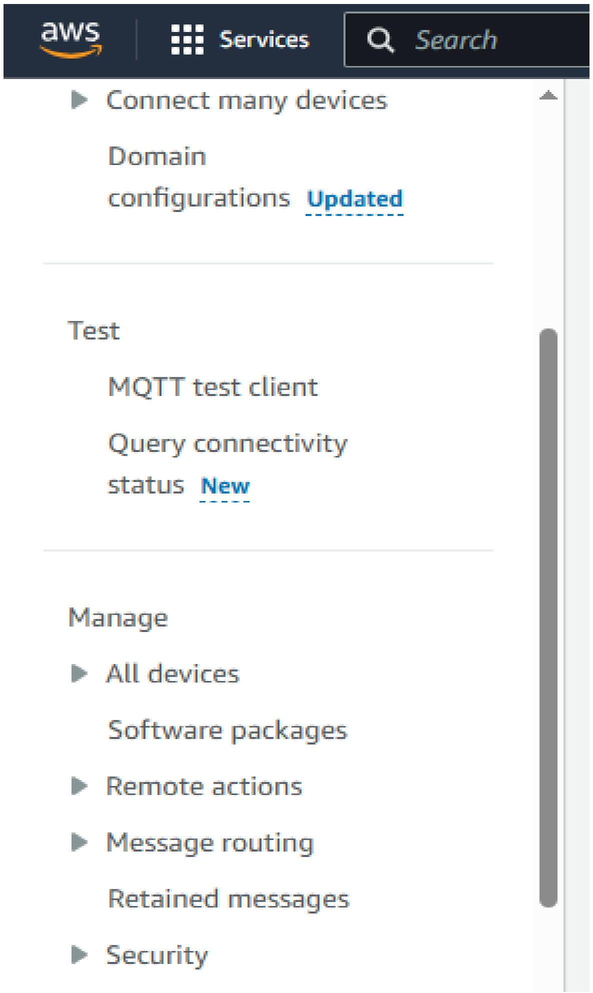
Securely connect, test, and manage your IoT devices

AWS IoT can support billions of devices and trillions of messages. It can process and route those messages to AWS endpoints and to other devices reliably and securely.

Get started with AWS IoT

Quick connect guides you through connecting a device in about 15 minutes. You'll register your first device and watch it send MQTT messages to AWS IoT.

On the left side of the dashboard, there are so many options. But for this assignment we need to work with two options here. One is the **manage** option and the other one is the **secure** option.

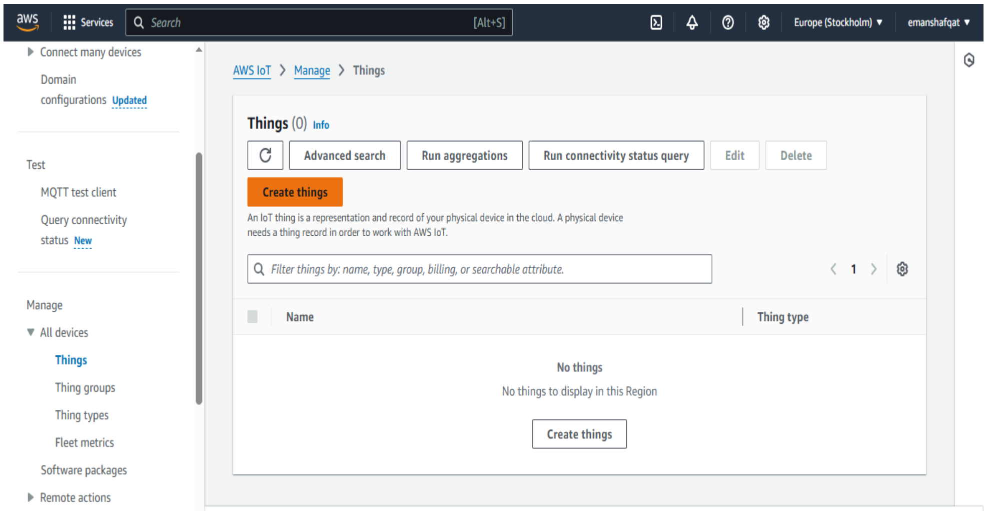


Creating a Thing

Now we need to create a thing associated with our project. For this, follow the following steps:

- Specifying thing properties
- Configuring device certificate
- Attaching policies to certificate

Under the manage option click on Thing. Now we need to create a Thing here. So, click on Create Things here.



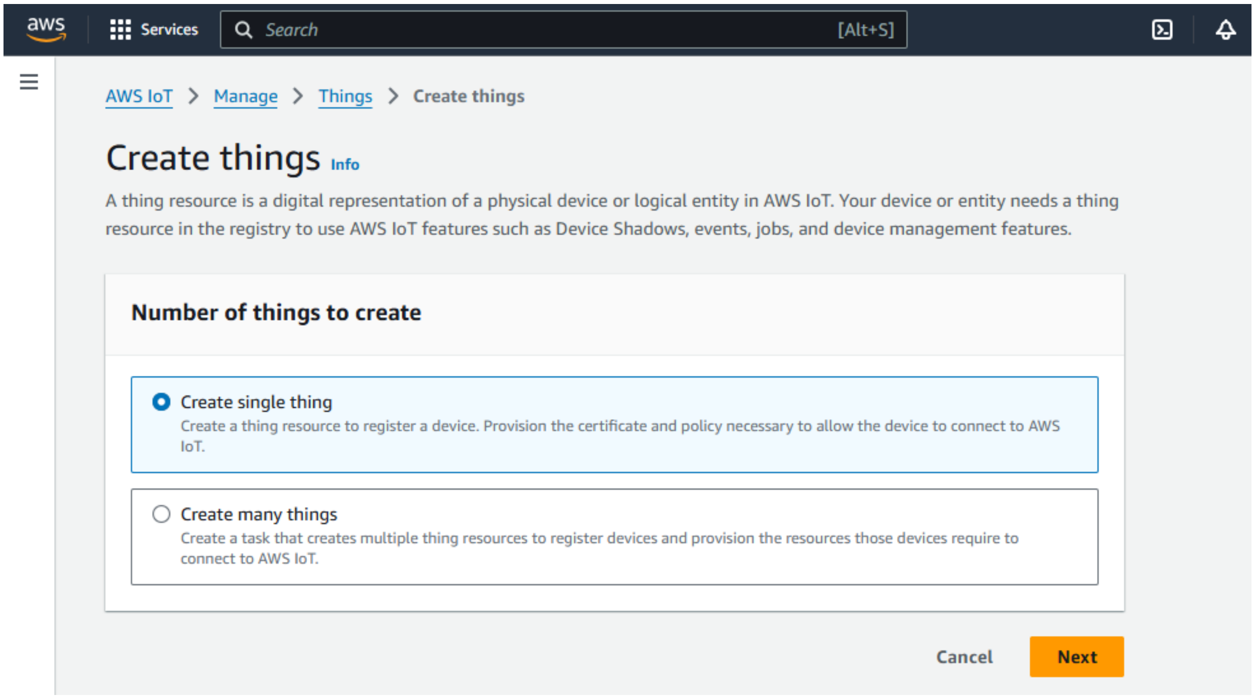
Things (0)

An IoT thing is a representation and record of your physical device in the cloud. A physical device needs a thing record in order to work with AWS IoT.

No things

No things to display in this Region

You can select whether create a single thing or create many things. But for our applications, select create a single thing. Then click on Next.



Create things

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.

Number of things to create

● Create single thing
Create a thing resource to register a device. Provision the certificate and policy necessary to allow the device to connect to AWS IoT.

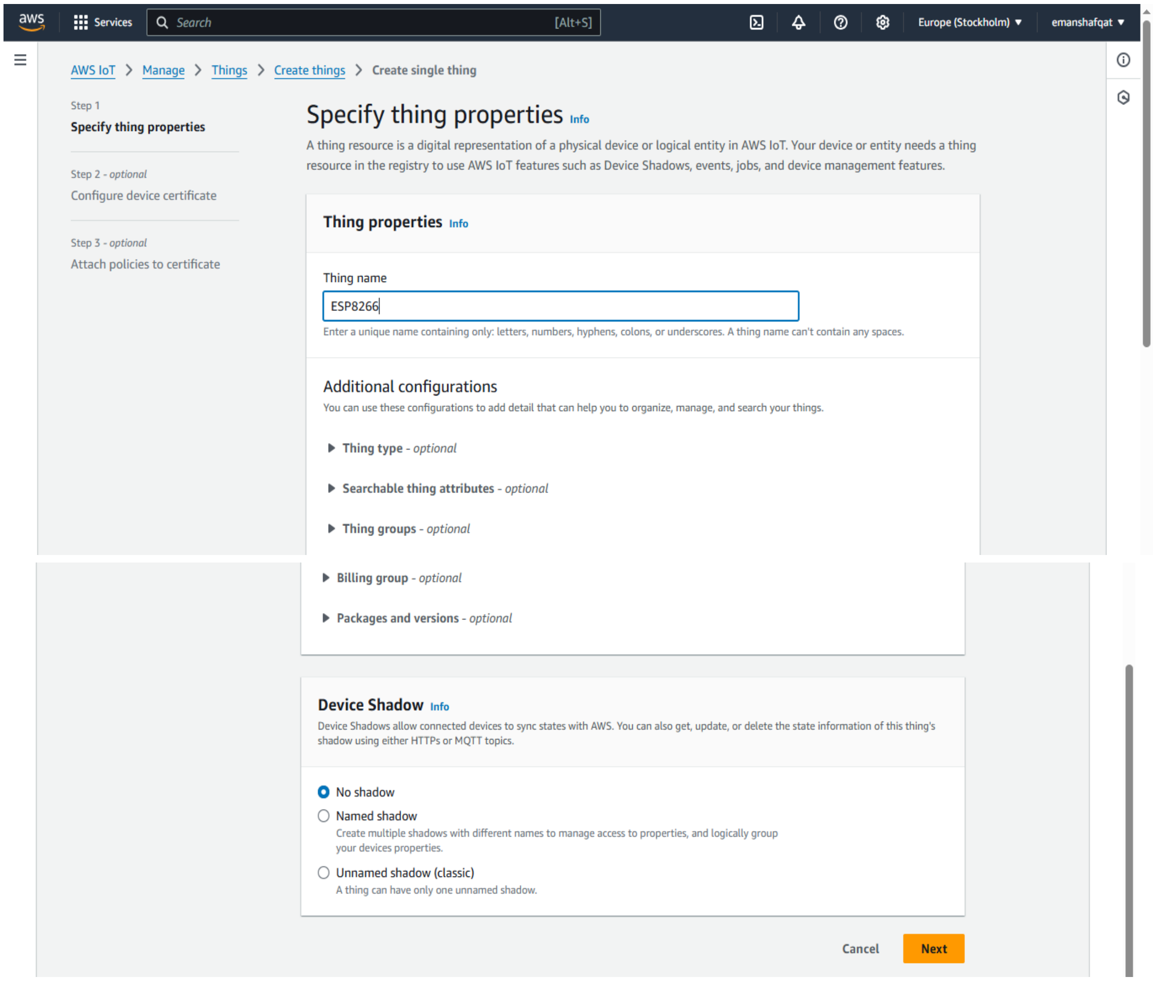
○ Create many things
Create a task that creates multiple thing resources to register devices and provision the resources those devices require to connect to AWS IoT.

Specify Thing Properties

Here we need to specify the Thing properties. First, give a thing name. You can name it anything. For example, I will name it *ESP8266*.

Under additional configurations, there is no need to make any changes.

Under the device shadow option, select the first option as No shadow. Then click on Next.



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: ESP8266

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.

- ▶ **Thing type** - *optional*
- ▶ **Searchable thing attributes** - *optional*
- ▶ **Thing groups** - *optional*
- ▶ **Billing group** - *optional*
- ▶ **Packages and versions** - *optional*

Device Shadow Info

Device Shadows allow connected devices to sync states with AWS. You can also get, update, or delete the state information of this thing's shadow using either HTTPs or MQTT topics.

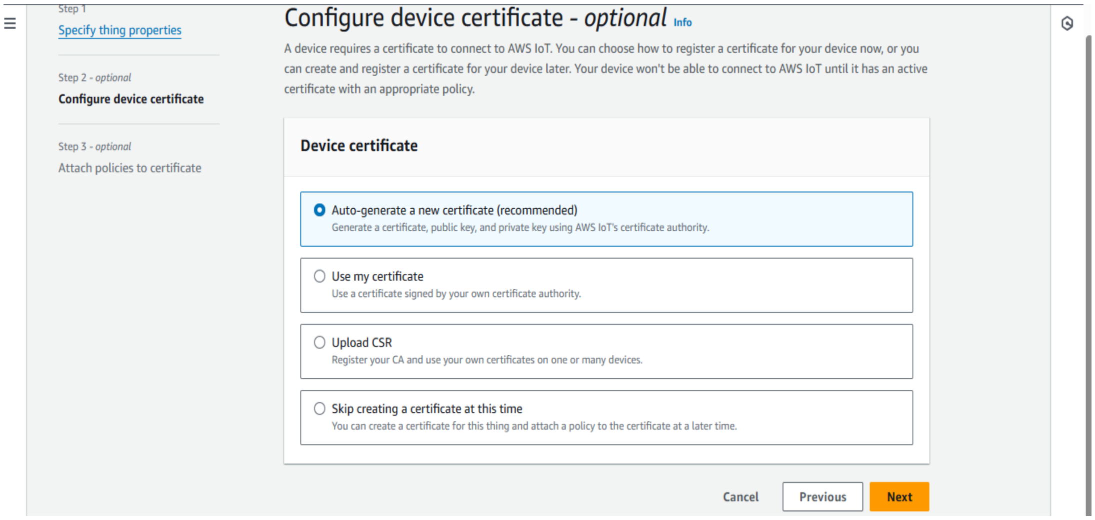
● No shadow

○ Named shadow
Create multiple shadows with different names to manage access to properties, and logically group your devices properties.

○ Unnamed shadow (classic)
A thing can have only one unnamed shadow.

Generate Device Certificate

Select the Auto Generate New Certificate. Then click on Next.


Step 1
Specify thing properties

Step 2 - *optional*
Configure device certificate

Step 3 - *optional*
Attach policies to certificate

Configure device certificate - *optional* Info

A device requires a certificate to connect to AWS IoT. You can choose how to register a certificate for your device now, or you can create and register a certificate for your device later. Your device won't be able to connect to AWS IoT until it has an active certificate with an appropriate policy.

Device certificate

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

○ Use my certificate
Use a certificate signed by your own certificate authority.

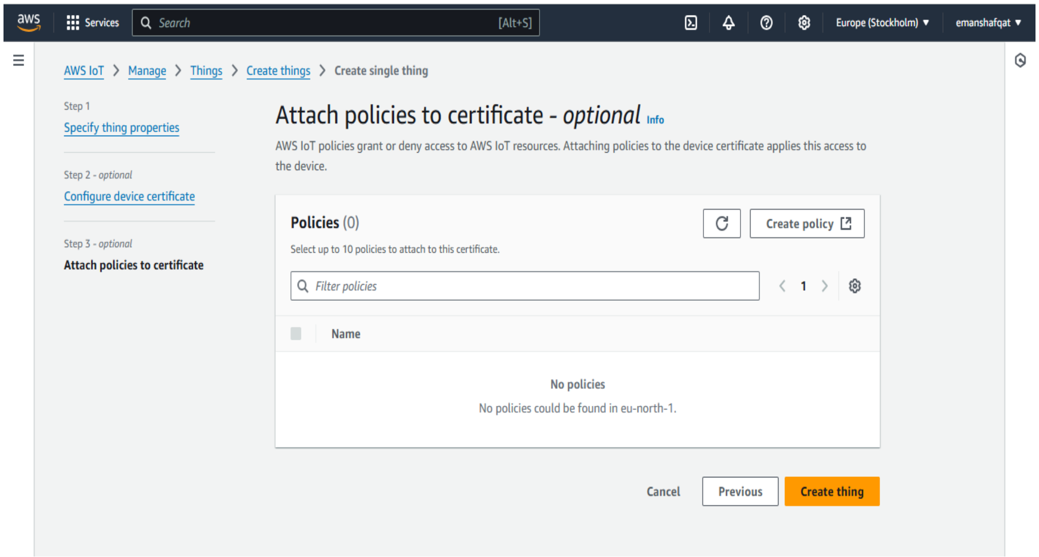
○ Upload CSR
Register your CA and use your own certificates on one or many devices.

○ Skip creating a certificate at this time
You can create a certificate for this thing and attach a policy to the certificate at a later time.

Create & Attach Policy

Now we need to attach a policy to the Things we created. But no policies are here right now. So we need to create a policy first.

So click on create policy. Here give any name to the policy. For example, I will give it a name as ***“ESP8266_Policy“***.



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

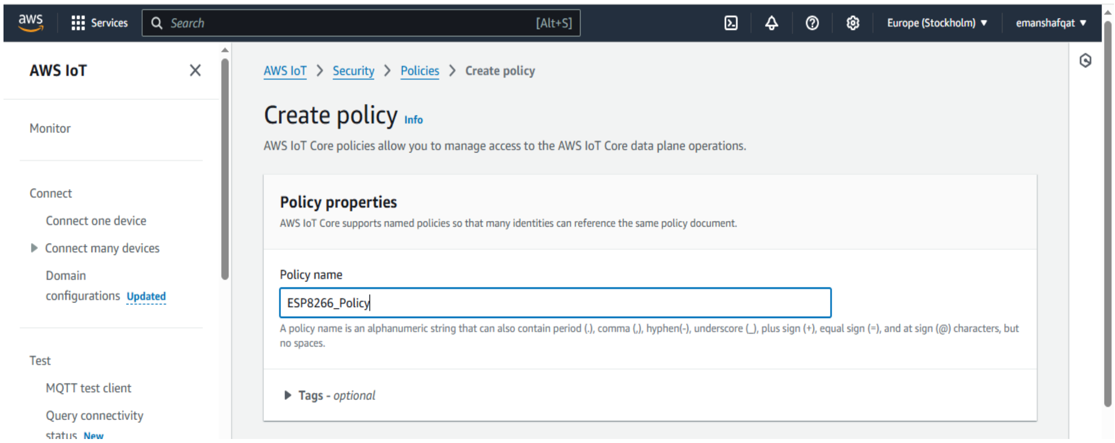
Attach policies to certificate - *optional* Info

AWS IoT policies grant or deny access to AWS IoT resources. Attaching policies to the device certificate applies this access to the device.

Policies (0)

Select up to 10 policies to attach to this certificate.

Name
No policies No policies could be found in eu-north-1.



AWS IoT > Security > Policies > Create policy

Create policy Info

AWS IoT Core policies allow you to manage access to the AWS IoT Core data plane operations.

Policy properties

AWS IoT Core supports named policies so that many identities can reference the same policy document.

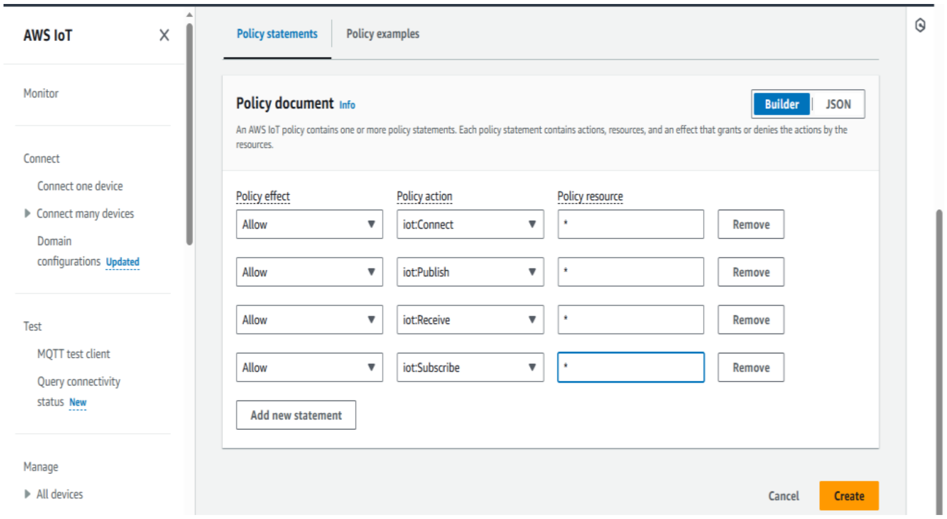
Policy name: ESP8266_Policy

A policy name is an alphanumeric string that can also contain period (.), comma (,), hyphen(-), underscore (_), plus sign (+), equal sign (=), and at sign (@) characters, but no spaces.

▸ Tags - *optional*

Now the add statement part is very important. Under the action, type IoT. So multiple options will pop up. From here we will only need to publish, Subscribe, Connect and Receive.

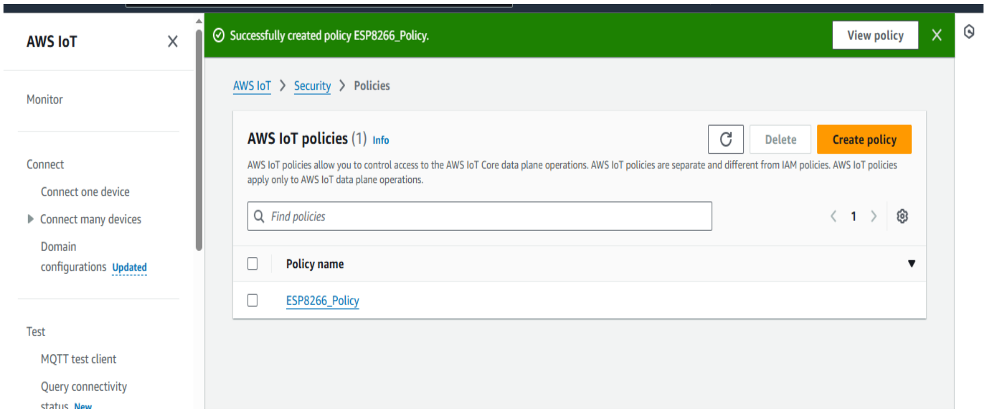
Click on create to create the policy. So the policy has been created successfully.



Policy document Info

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	iot:Connect	*	Remove
Allow	iot:Publish	*	Remove
Allow	iot:Receive	*	Remove
Allow	iot:Subscribe	*	Remove



Successfully created policy ESP8266_Policy.

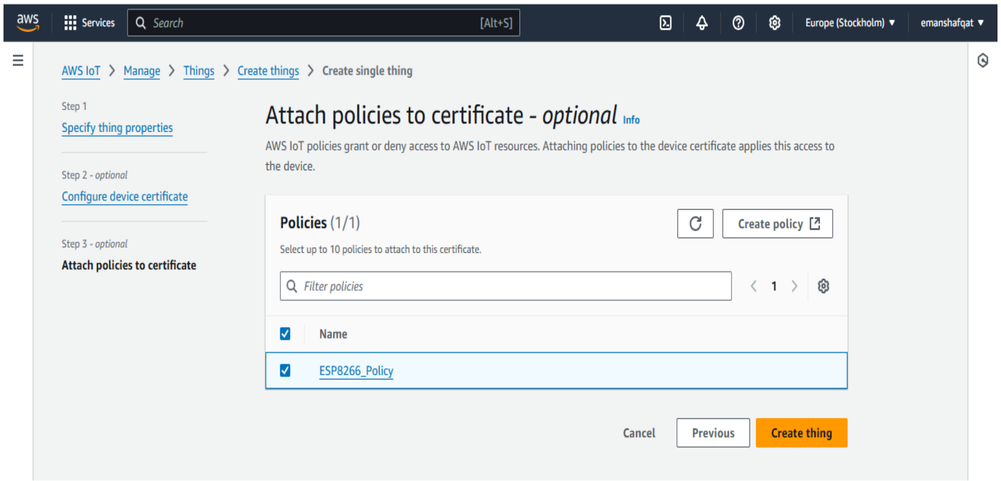
AWS IoT > Security > Policies

AWS IoT policies (1) Info

AWS IoT policies allow you to control access to the AWS IoT Core data plane operations. AWS IoT policies are separate and different from IAM policies. AWS IoT policies apply only to AWS IoT data plane operations.

☐	Policy name
☐	ESP8266_Policy

Now go back to Create Thing option. So a policy option will appear. We need to attach the policies to the certificate. So select the appeared policy and click on create a thing.



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

Attach policies to certificate - *optional* Info

AWS IoT policies grant or deny access to AWS IoT resources. Attaching policies to the device certificate applies this access to the device.

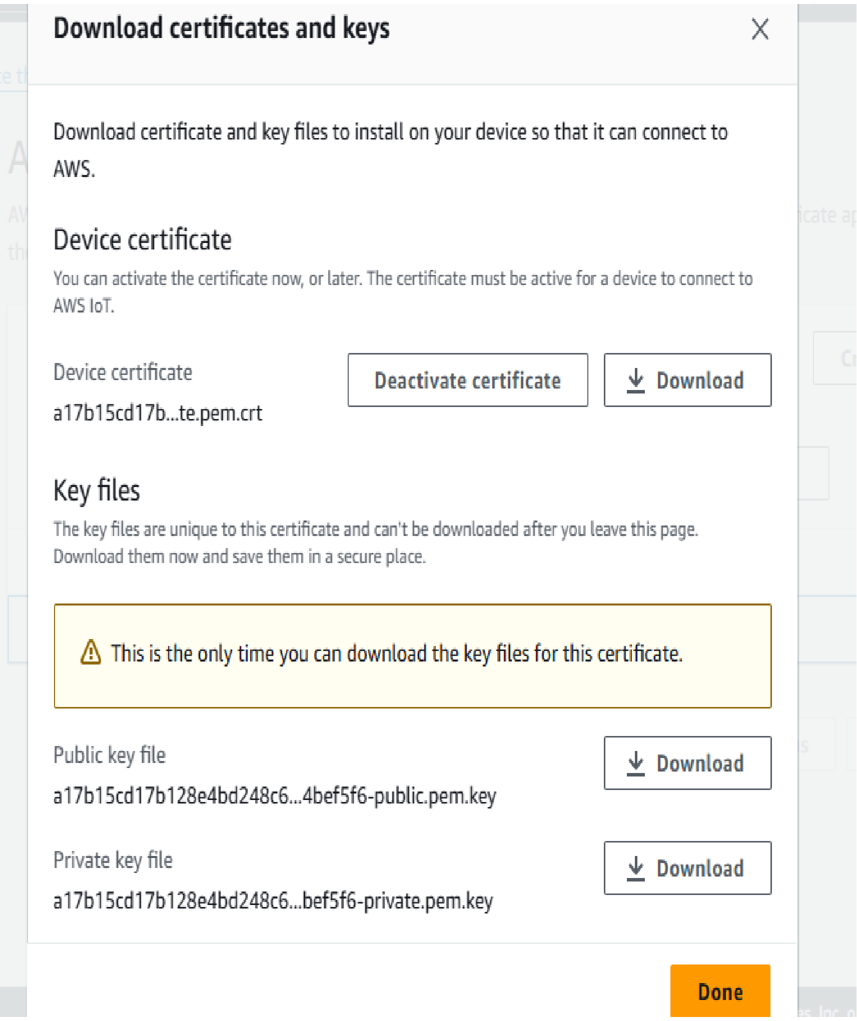
Policies (1/1)

Select up to 10 policies to attach to this certificate.

☑	Name
☑	ESP8266_Policy

Downloading Certificates and Keys

Now we need to download the required certificates from this list.



Download certificates and keys

Download certificate and key files to install on your device so that it can connect to AWS.

Device certificate

You can activate the certificate now, or later. The certificate must be active for a device to connect to AWS IoT.

Device certificate
a17b15cd17b...te.pem.crt

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.

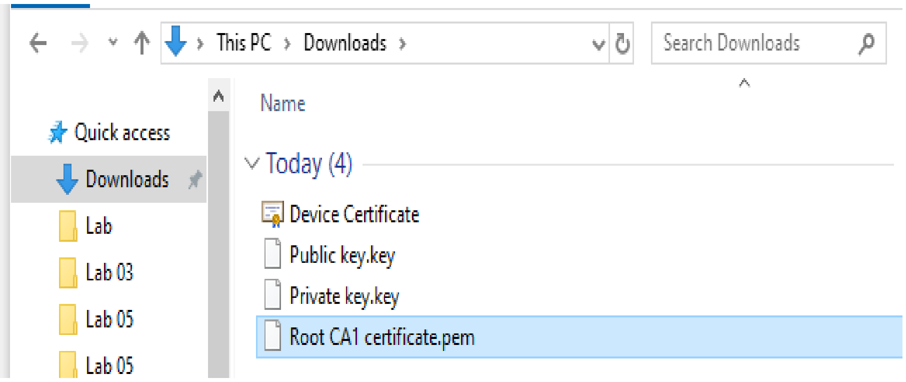
Public key file
a17b15cd17b128e4bd248c6...4bef5f6-public.pem.key

Private key file
a17b15cd17b128e4bd248c6...bef5f6-private.pem.key

First, download the **device certificate** and then rename it as a device certificate for identification.

Also, download the **public key** and rename it as a public key. Then download the **private key** and rename it as a private key.

In the Root CA Certificates, there are two certificates here. But we just need a **Root CA1 certificate**, so download it as well.

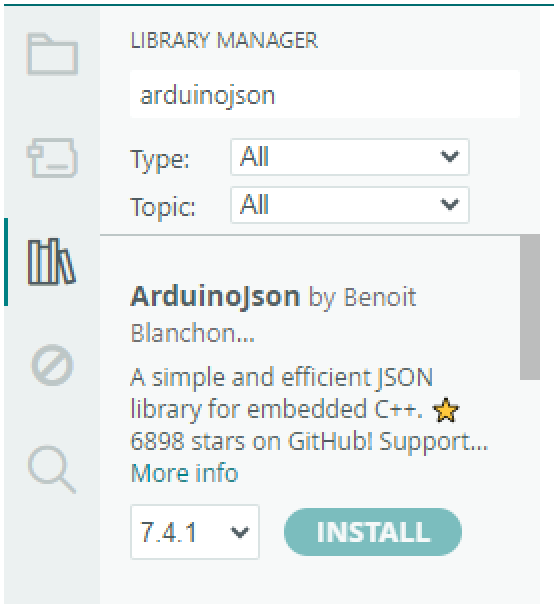


So we have downloaded all the certificates that we need for our project.

Installing Necessary Arduino Libraries

1. ArduinoJSON Library

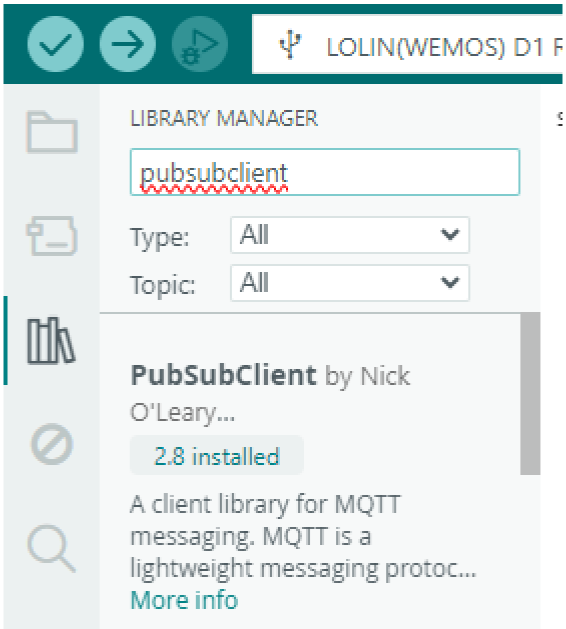
So first go to the library manager and search for "JSON" & install the library as shown in the figure below.



A simple and efficient JSON library for embedded C++. ⭐ 6898 stars on GitHub! Support...

2. PubSubClient Library

Again go to the library manager and search for "PubSubClient" & install the library from Nick O'Leary.



pubsubclient

Type: All

Topic: All

PubSubClient by Nick O'Leary...

2.8 installed

A client library for MQTT messaging. MQTT is a lightweight messaging protoc...

Code Snippet & Explanation

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

#define TIME_ZONE -5

#define LED_PIN 2 // GPIO2 for the built-in LED on ESP8266

unsigned long lastMillis = 0;
#define AWS_IOT_STATE_TOPIC "esp8266/state"
#define AWS_IOT_CONTROL_TOPIC "esp8266/control"

// Wi-Fi and AWS IoT credentials
const char WIFI_SSID[] = "eman's S24 Ultra"; // Replace with your Wi-Fi SSID
const char WIFI_PASSWORD[] = "12345678";     // Replace with your Wi-Fi password

#define THINGNAME "ESP8266"

const char MQTT_HOST[] = "a1cpvpt3ls1ke-ats.iot.eu-north-1.amazonaws.com";

#define MQTT_PORT 8883
#define PUBLISH_INTERVAL 5000
```

```
static const char cacert[] PROGMEM = R"EOF(
-----BEGIN CERTIFICATE-----
MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF
ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDEx
BBbWF6
b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAwMFo
wOTEL
MAkGA1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIF
Jv
b3QgQ0EgMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALJ4gHHKeNX
j
ca9HgFB0fW7Y14h29Jlo91ghYPl0hAEvrAIthtOgQ3pOsqTQNroBvo3bSMgHFzZM
9O6II8c+6zf1tRn4SWiw3te5djgdYZ6k/oI2peVKVuRF4fn9tBb6dNqcmzU5L/qw
IFAGbHrQgLKm+a/sRxmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUcAwhmahRWa6
VOujw5H5SNz/0egwLX0tdHA114gk957EWW67c4cX8jJGKLhD+rcdqsq08p8kDi1L
93FcXmn/6pUCyziKrlA4b9v7LWIbxcceVOF34GfID5yHI9Y/QCB/IIDEgEw+OyQm
jgSubJrIqg0CAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBA
MC
AYYwHQYDVR0OBBYEFIQYzIU07LwMlJQuCFmcx7IQTgoIMA0GCSqGSIb3DQEBCwU
A
A4IBAQCY8jdaQZChGsV2USggNiMOruYou6r4lK5IpDB/G/wkjUu0yKGX9rbxenDI
U5PMCCjjmCXPI6T53iHTfIUJrU6adTrCC2qJeHZERxhlbI1Bjjt/msv0tadQ1wUs
N+gDS63pYaACbvXy8MWy7Vu33PqUXHeeE6V/Uq2V8viTO96LXFvKWlJbYK8U90vv
o/ufQJVtMVT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1bldZwgJcJmApzyMZFo6IQ6XU
5MsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXeGADbkpy
rqXRfboQnoZsG4q5WTP468SQvvG5
-----END CERTIFICATE-----
)EOF";

static const char client_cert[] PROGMEM = R"KEY(
-----BEGIN CERTIFICATE-----
MIIDWTCCAkGgAwIBAgIUSR7Qavws/No1cr3y0aCgf0uS75UwDQYJKoZIhvcNAQEL
BQAwTTFLMEkGA1UECwxCQW1hem9uIFdlYiBTZXJ2aWNlcyBPPUFtYXpvbi5jb20g
SW5jLiBMPVNlYXR0bGUgU1Q9V2FzaGluZ3RvbiBDPVVTMB4XDTI1MDQxMzExMjgw
NFoXDTQ5MTIzMTIzNTk1OVowHjEcMBoGA1UEAwwTQVdTIElvVCBDZXJ0aWZpY2F0
ZTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAKlcHYAyq5VOmnam8gK
H
VcArlvVQlEBLmLduvNo4Ecy29FTr1ETjcQqoqjMobJcIdtS2ccTdMLFjPykhJ7bb
FjPtVrbCsFX1kb2lkrsQCS3Dn6x7b3E+qKmSMWuF1FFogoh1nqigDLKX0n24CSyP
jRZppRqz38sSe94oNpK6GsStOqEEWnI2LigkcdmQq9ere0Ko2B6d3hnQ/pdkFyP1
Evt0CFfE4QyDBhIH7XMjq2uoI5T/RMRgB9uycNR6EPzf9hJjyFdHVd9wLTSfldd y
bgrngHSFR+n3JFPubMEMAH/n0+sYuF1BupRgsGjhj8k7fidT97e8E53doXY4M+ld
tBUCAwEAAaNgMF4wHwYDVR0jBBgwFoAUcCQ5GNQfG3LfUxfJ1OpVcq8rcjswHQYD
VR0OBBYEFEsIggeF87npHJ70erYiI3mfjStoMAwGA1UdEwEB/wQCMAAwDgYDVR0P
AQH/BAQDAgeAMA0GCSqGSIb3DQEBCwUAA4IBAQA/cCQFV2cPrwZaQrP5vwom+yEm
zLrqBni12uElQ7gdu7jhlrwY2vK0G7mEtsaVqk1RIgUB++yfUvuloZnAJG0Hu6MQ
Xnx2n7blR7krYaqObiQddJOkOVxLz6/GVqwHItR9yztELIxB9FX21JbNkEuXB0QK
gY4p1pK673RiD6303oQd/xWRC664pZHGer6f47YXqBFNcUf9JrtROj58U9EPE6ku
UuL1cxcBaQN9J/HvfG1hkYW3rpc/HcTRtwAdfC1hE5YzX5RqQzeX0EnLv+h9DgWB
2N/yN4oJ61H4Jm/lYCwh+/M9zJQQoPQ0TPjb5MEqSt9KkgdpIYqeoPbSxOc2
-----END CERTIFICATE-----

)KEY";

static const char privkey[] PROGMEM = R"KEY(
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAqVwdgDKrlU6adqbyAodVwCuW9VCUQEuYt2682jgRzLb0VOvU
RONxCqiqMyhslwh21LZxxN0wsWM/KSEnttsWM+1WtsKwVfWRvaWSuxAJLcOfrHtv
cT6oqZIxa4XUUWiCiHWeqKAMspfSfbgJLI+NFmmlGrPfyxJ73ig2kroaxK06oQRa
cjYuKCRx2ZCr16t7QqjYHp3eGdD+l2QXI/US+3QIV8ThDIMGEgftcyOra6gjlP9E
xGAH27Jw1HoQ/N/2EmPIV0dV33AtNJ+V13JuCueAdIVH6fckU+5swQwAf+fT6xi4
XUG6lGCwaOGPyTt+J1P3t7wTnd2hdjgz6V20FQIDAQABAoIBAQCXkqecerzx7d10
X8ms1a67RMfgWfO073OIiAVhUFkt3HgkIMRjuyMKpRVen8lACwU/LuH3jPbta6O1
jcTbdI0UgE/hVbE33YZIMw11ec6dNL2NQxPkSG3vBbZRQv9YtK5OruYGiOeUVyi9
953OUdO1wWtQ0Ld0KeDOXTMqcoFOjz21dAAKySE1SZIxK990aUvHZU/2QV1Rn5gD
QWcJtSM1EjMGblJaI1YbOHQH/eV/BjeJF2XlOWOiFvapgaUrO3TJ8eN61tYAjYca
BLmVhlYuniu1O1qnoPXN2cGlefjDmYli3aqVLxxw4O9p3EjEtxYHxVBQcQN6CsMO
KJv/SzopAoGBAN4BA3CLfPRb+8jnNbsjdOfaNSUHc3JMTJQ9eIA41S+3Imf5fQJI
JBn7xFgBd3P9HPwquh9d6FSqYQe60jADY194nuuhntGPbeFzj6rxgjqwtk0aUFtc
67Ok82Mbk7/s5BMIu1hfiEoa3/c1mgpSMcrPKkx5mdn+cizeVWwDIYJfAoGBAMNL
WXbmYMO3Y4XepP7ldbaFu8duaqS0MKJzKKhNVeZY7BjhWhWi9Pmxe8JYSeS+xkYr
w+gXuZe3LVIobtJ0QnF6aTDoarynNx4YCk50AXNgY2sDGAW0P79s+zte+YevEaDm
Gt6zKUdpp0Umw1m1yKFS27nUiAHmYMZfbVjJTSYLAoGBALvpnbC40SbBaUkvtAJY
S88om41hVi3sOXHM6l+WO5Vs7TmXK2IwECbSPerEKwlJuSwVOn/sbAimQwVY9Crh
405feh03K4h0YGzR9UybqkDlbRmsaVI8P+JKgB1mrVRqPs+Bt5phGqqnPuYwP9f2
pODnJAWhdEHaJ2ilNSb9K3L/AoGBAKTsT/wiJ5uAlpJp6Gq9+n+ORnrv2y2GYf/+
QlqHHMCvXYrRXFMjT8BByyLfjl+XYDfII+QvjPL7pVFZheVCew378zNzQxhYGirW
4HVKcCMwRiUL8tB2XNNaxHtqBE46YSfAyvZA1N5S8ghq6NQsuObL43T2b3kzRYyg
7+Xn8bfrAoGAHDiHiP5CdiOZl1rokvPc6FbA7OgWWiH/AYjhOqQxgxF3Re0efOah
OZo5t+jt/ToqCXUqwFVozbbo4YpWOBkvtnlwLaaUBpxdaZRBbKpISDJKGMBz+2Lc
UX+cmCUKMMpjZc3eK76e4L+XwaYKnq/HoOho9zpvInDDNhtqoP6eyKI=
-----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() {
    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("]: ");
    String message;
    for (int i = 0; i < length; i++) {
        message += (char)payload[i];
    }
    Serial.println(message);

    if (message == "ON") {
        digitalWrite(LED_PIN, HIGH);
        publishLEDState(true);
    } else if (message == "OFF") {
        digitalWrite(LED_PIN, LOW);
        publishLEDState(false);
    }
}

void publishLEDState(bool state) {
    if (!client.connected()) {
        Serial.println("Cannot publish - not connected");
        return;
    }
    StaticJsonDocument<200> doc;
    doc["device"] = THINGNAME;
    doc["led_state"] = state ? "ON" : "OFF";
    doc["timestamp"] = time(nullptr);

    char jsonBuffer[512];
    serializeJson(doc, jsonBuffer);

    Serial.print("Publishing to ");
    Serial.print(AWS_IOT_STATE_TOPIC);
    Serial.print(": ");
    Serial.println(jsonBuffer);

    if (client.publish(AWS_IOT_STATE_TOPIC, jsonBuffer)) {
        Serial.println("Publish succeeded");
    } else {
        Serial.println("Publish failed!");
        Serial.println(client.state()); // Print MQTT connection state
    }
}

// Add the missing connectAWS function
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_CONTROL_TOPIC);
  Serial.println("AWS IoT Connected!");
}

void setup() {
  Serial.begin(115200);
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, HIGH);
  connectAWS();
  publishLEDState(true);
}

void loop() {
  static unsigned long lastPublish = 0;

  if (!client.connected()) {
    Serial.println("Disconnected - Attempting reconnect");
    connectAWS();
  }

  client.loop();  // Maintain MQTT connection

  // Regular state publishing
  if (millis() - lastPublish > PUBLISH_INTERVAL) {
    lastPublish = millis();
    bool currentState = (digitalRead(LED_PIN) == HIGH); // LOW = ON for ESP8266
    publishLEDState(currentState);

    // Debug connection status
    Serial.print("MQTT Connected: ");
    Serial.println(client.connected() ? "YES" : "NO");
    Serial.print("WiFi Connected: ");
    Serial.println(WiFi.status() == WL_CONNECTED ? "YES" : "NO");
  }
}
```

Explanation of Code:

This code implements an ESP8266 IoT device that connects to AWS IoT Core to control an LED remotely and report its status. Here's a concise breakdown:

Key Components:

1. WiFi & AWS Connection:
   - Connects to WiFi using provided credentials
   - Establishes secure MQTT connection to AWS IoT using certificates
   - Syncs time via NTP (required for AWS authentication)
2. LED Control:
   - Built-in LED on GPIO2 (inverted logic: HIGH=OFF, LOW=ON)
   - Listens to MQTT topic esp8266/control for "ON"/"OFF" commands
3. State Reporting:
   - Publishes JSON messages to esp8266/state every 5 seconds containing:

{"device":"ESP8266","led_state":"ON/OFF","timestamp":123456789}

   - Also publishes immediately when LED state changes
4. Error Handling:
   - Automatic reconnection if connection drops
   - Detailed serial output for debugging

How It Works:

1. On startup: Connects to AWS and publishes initial state
2. In loop:
   - Maintains MQTT connection
   - Listens for incoming commands
   - Periodically publishes current state
3. When "ON"/"OFF" received:
   - Changes LED state

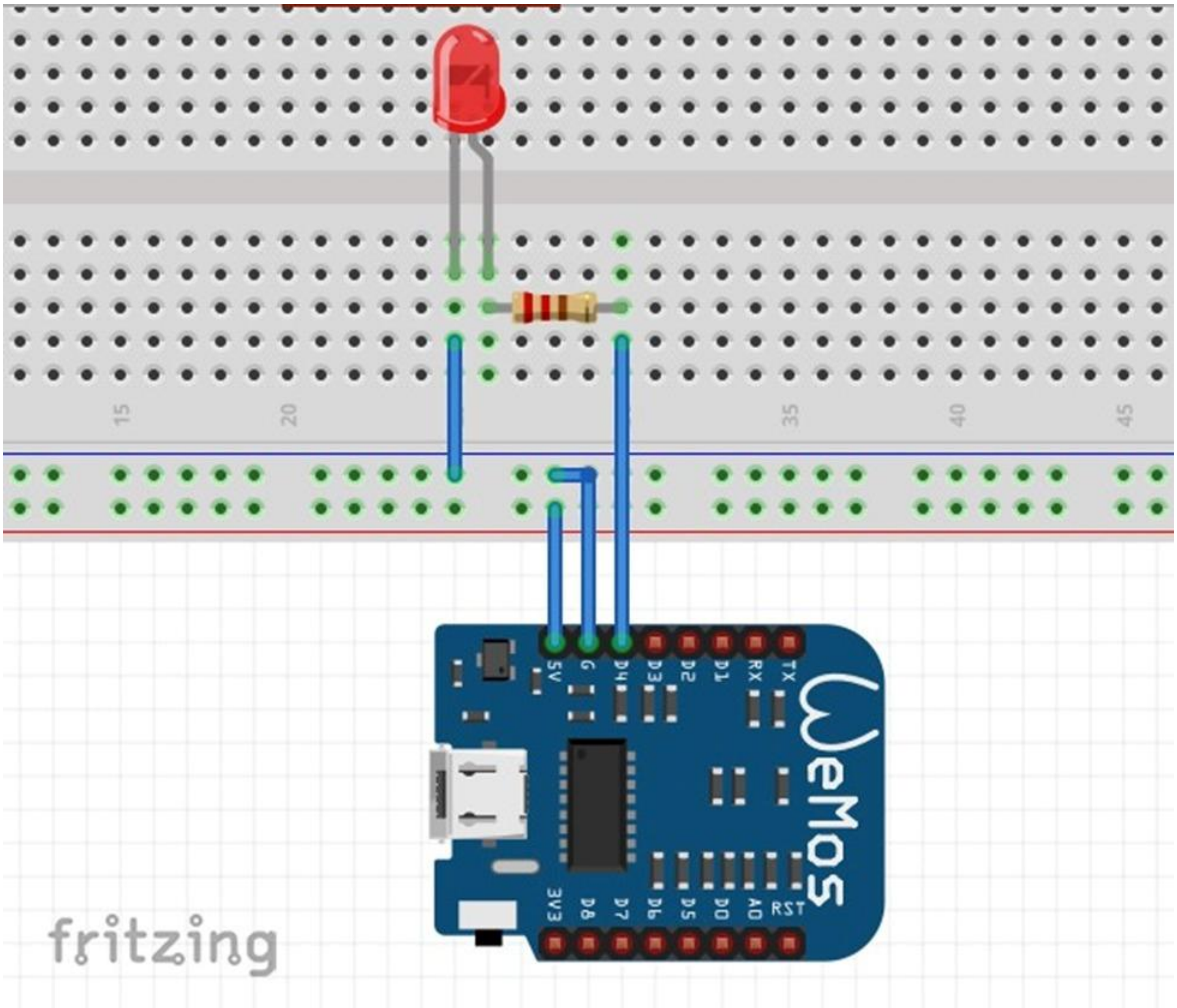
  - Publishes update

Usage:

Send "ON" or "OFF" (exactly, no quotes) to esp8266/control topic in AWS IoT Console to control the LED. The device's state will be visible in the esp8266/state topic.

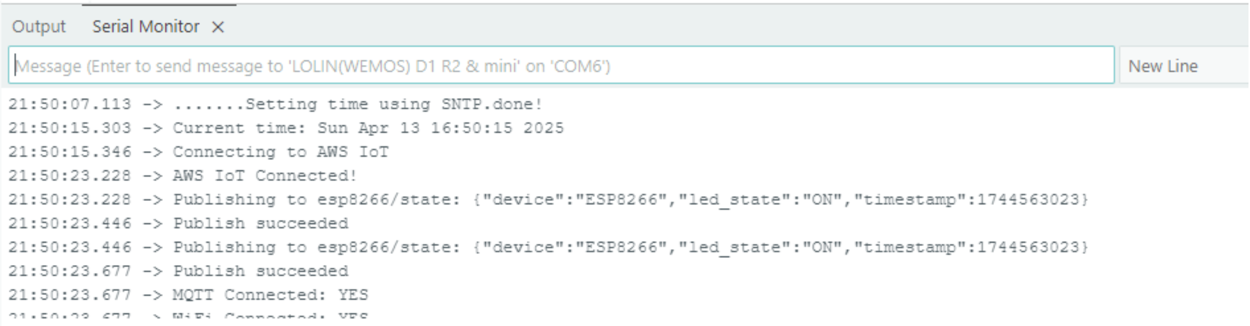
The code includes robust error handling and status monitoring via Serial Monitor (115200 baud).

Circuit diagram:



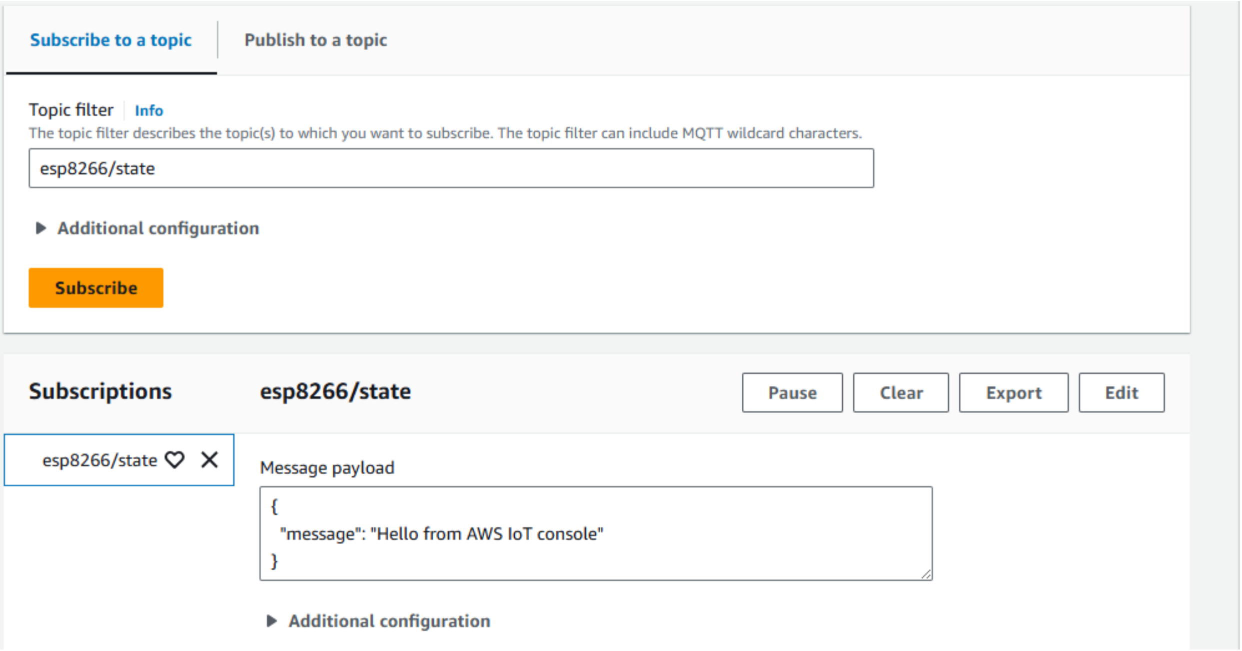
Results (Screenshots of data in AWS or MQTT client)

Output:


Output
Serial Monitor ×
Message (Enter to send message to 'LOLIN(WEMOS) D1 R2 & mini' on 'COM6')
New Line

```
21:50:07.113 -> .......Setting time using SNTP.done!
21:50:15.303 -> Current time: Sun Apr 13 16:50:15 2025
21:50:15.346 -> Connecting to AWS IoT
21:50:23.228 -> AWS IoT Connected!
21:50:23.228 -> Publishing to esp8266/state: {"device":"ESP8266","led_state":"ON","timestamp":1744563023}
21:50:23.446 -> Publish succeeded
21:50:23.446 -> Publishing to esp8266/state: {"device":"ESP8266","led_state":"ON","timestamp":1744563023}
21:50:23.677 -> Publish succeeded
21:50:23.677 -> MQTT Connected: YES
21:50:23.677 -> WiFi Connected: YES
```

Subscribing Sensor Data to AWS Dashboard



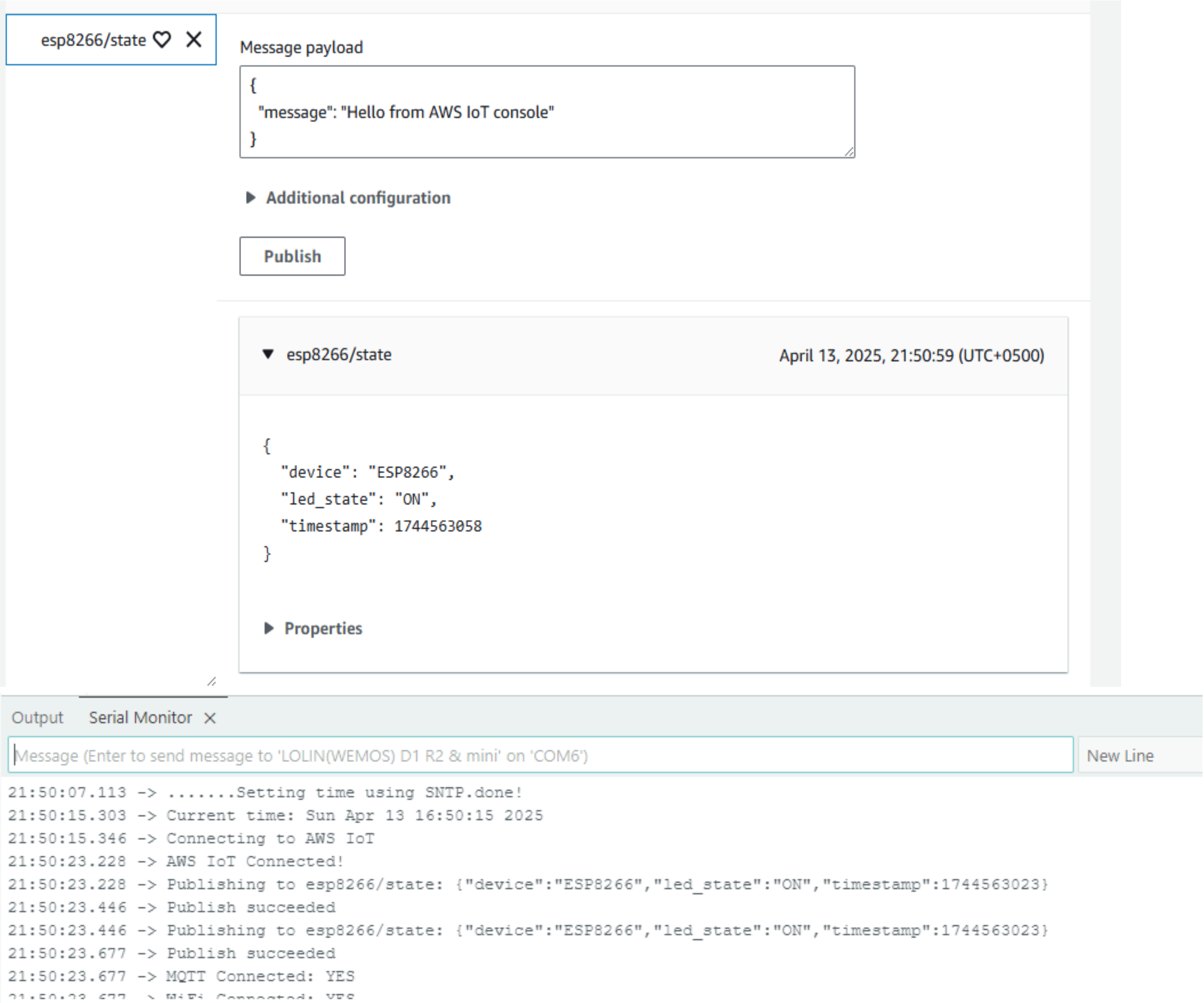
Topic filter

The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.

esp8266/state

Message payload

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



Message payload

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

▶ Additional configuration
Publish

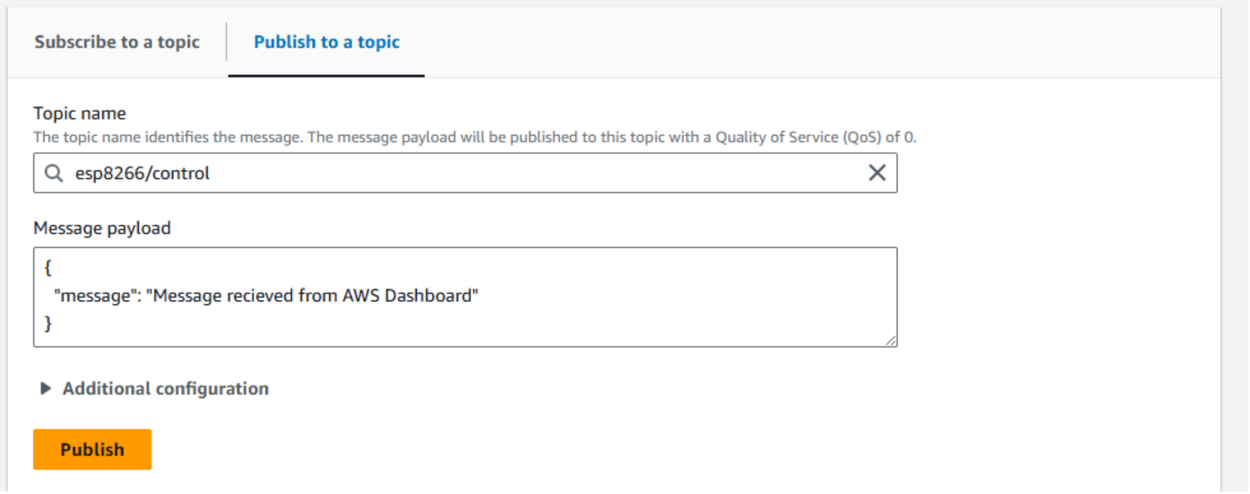
▼ esp8266/state April 13, 2025, 21:50:59 (UTC+0500)

```
{
  "device": "ESP8266",
  "led_state": "ON",
  "timestamp": 1744563058
}
```

▶ Properties
Output
Serial Monitor ✕
Message (Enter to send message to 'LOLIN(WEMOS) D1 R2 & mini' on 'COM6')
New Line

```
21:50:07.113 -> .......Setting time using SNTP.done!
21:50:15.303 -> Current time: Sun Apr 13 16:50:15 2025
21:50:15.346 -> Connecting to AWS IoT
21:50:23.228 -> AWS IoT Connected!
21:50:23.228 -> Publishing to esp8266/state: {"device":"ESP8266","led_state":"ON","timestamp":1744563023}
21:50:23.446 -> Publish succeeded
21:50:23.446 -> Publishing to esp8266/state: {"device":"ESP8266","led_state":"ON","timestamp":1744563023}
21:50:23.677 -> Publish succeeded
21:50:23.677 -> MQTT Connected: YES
21:50:23.677 -> WiFi Connected: YES
```

Publishing Data to Serial Monitor



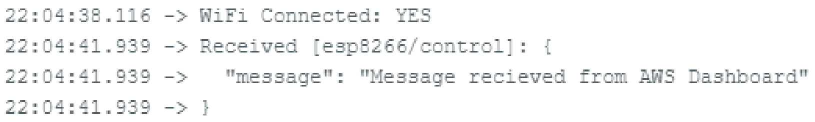
Topic name

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

esp8266/control

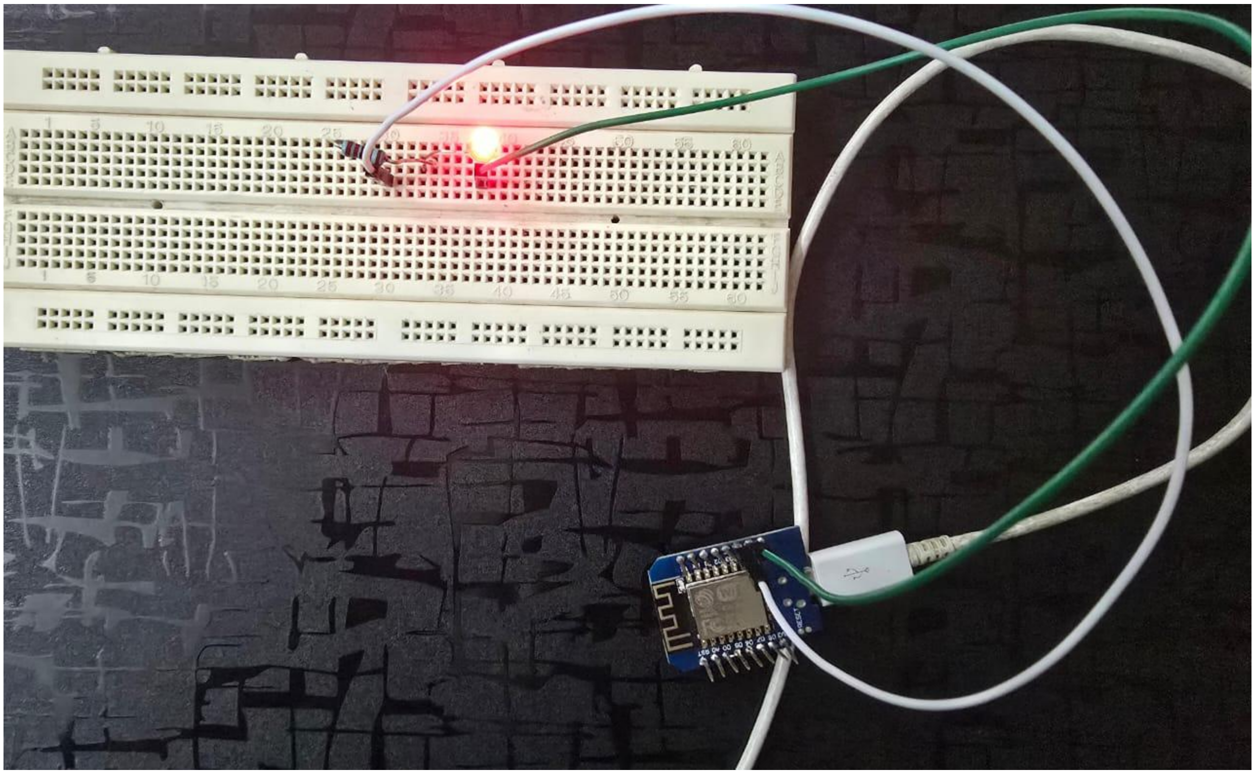
Message payload

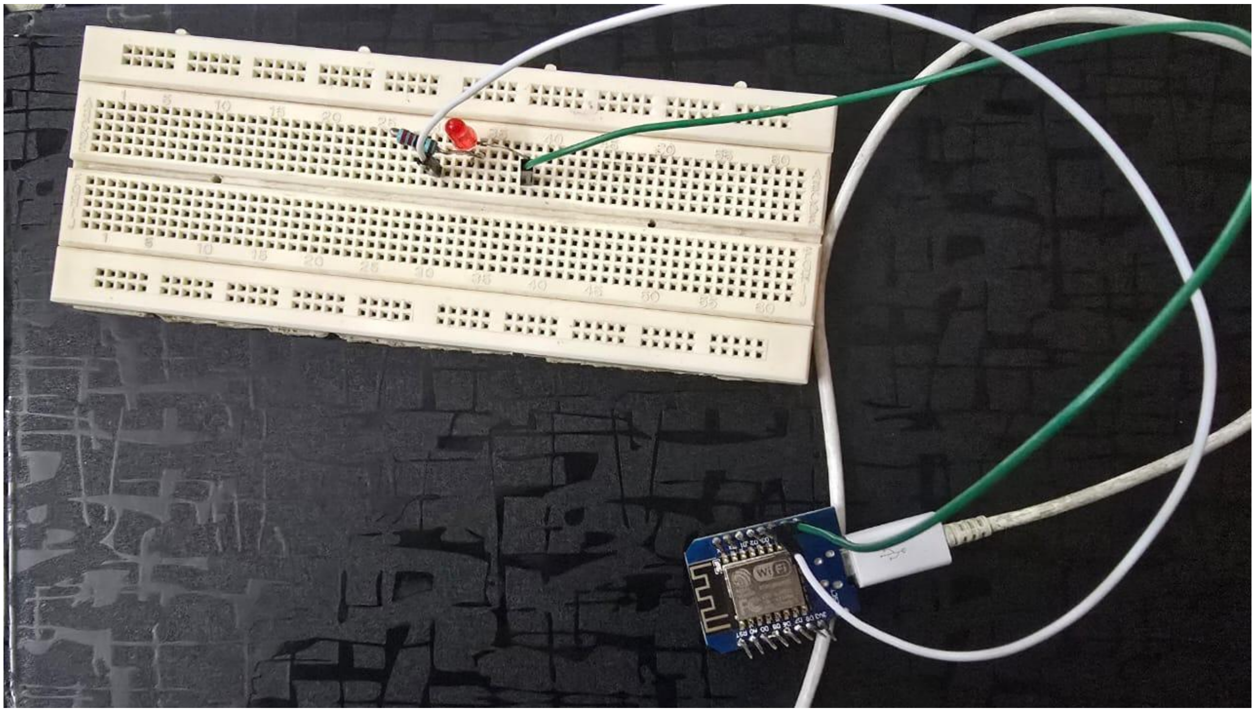
```
{
  "message": "Message recieved from AWS Dashboard"
}
```



```
22:04:38.116 -> WiFi Connected: YES
22:04:41.939 -> Received [esp8266/control]: {
22:04:41.939 ->   "message": "Message recieved from AWS Dashboard"
22:04:41.939 -> }
```

Circuit:





2. Blynk

Overview of Blynk Platform

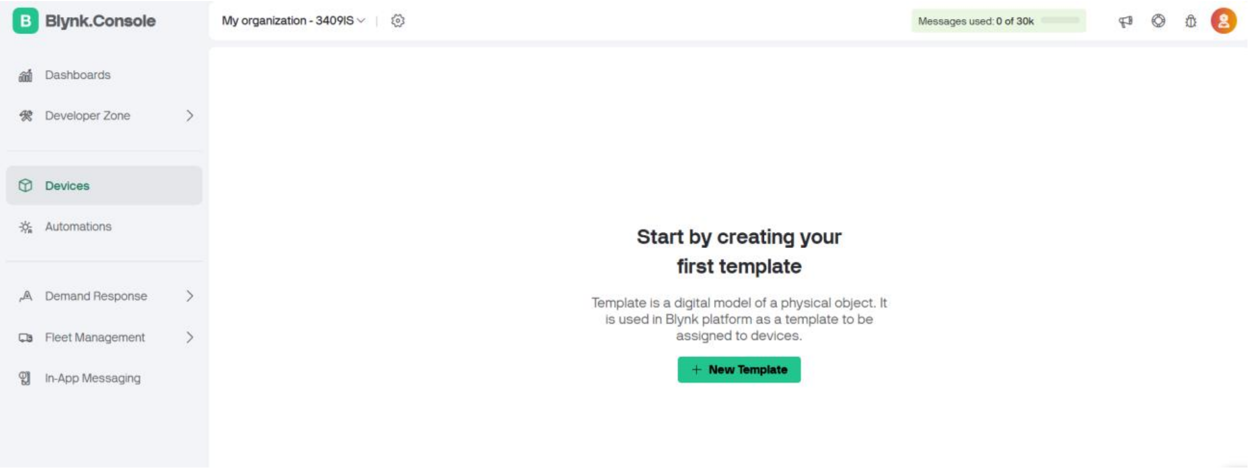
If you want to make any IoT device then **Blynk** is a popular **Internet of Things (IoT)** platform that allows you to control hardware remotely through smartphones or tablets. It provides a user-friendly drag-and-drop interface to simplify the creation of **customized interfaces** for designing and controlling various IoT devices and projects. Blynk supports many different types of development boards and different types of connection types such as Ethernet, WiFi, GSM, and Satellite. It facilitates **real-time data visualization, remote monitoring, and interactive automation**.

Account & Project Setup

To use the Blynk app in our project we perform the following steps to configure the Blynk setup as follow:

Setup Blynk 2.0 App

Step 1: Go to the **Blynk** website and click the "**Sign Up**" button. Enter your email address and create a password.

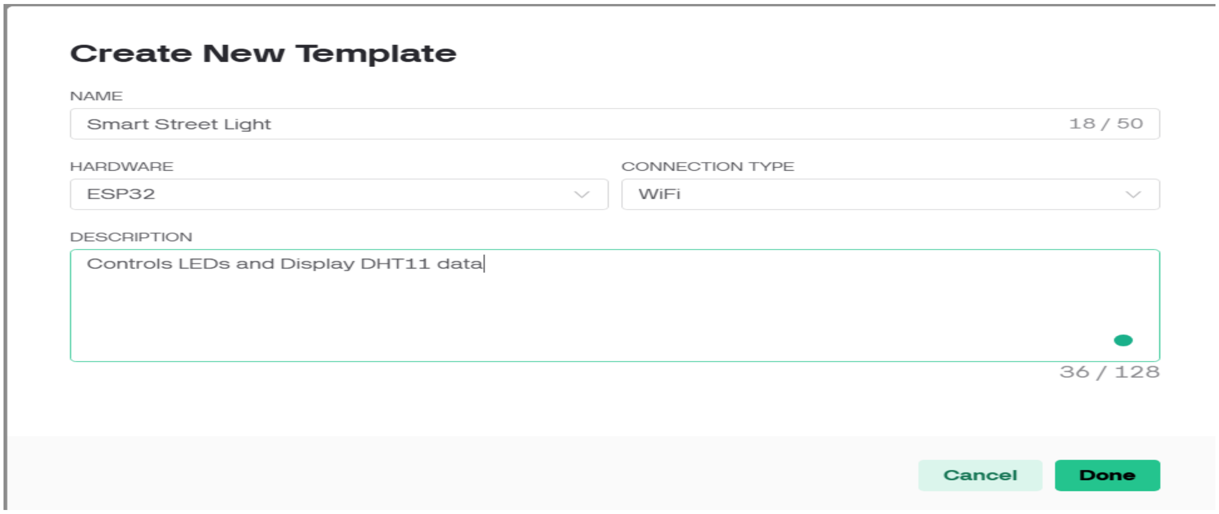


Start by creating your first template

Template is a digital model of a physical object. It is used in Blynk platform as a template to be assigned to devices.

Step 2: Once you have signed up on the Blynk and logged in, tap the "**+ New Template**" button to create a new template.

Step 3: Name the template, select the Development Board, set **WiFi** as the connection type and then save this template.



Create New Template

NAME

Smart Street Light 18 / 50

HARDWARE

ESP32

CONNECTION TYPE

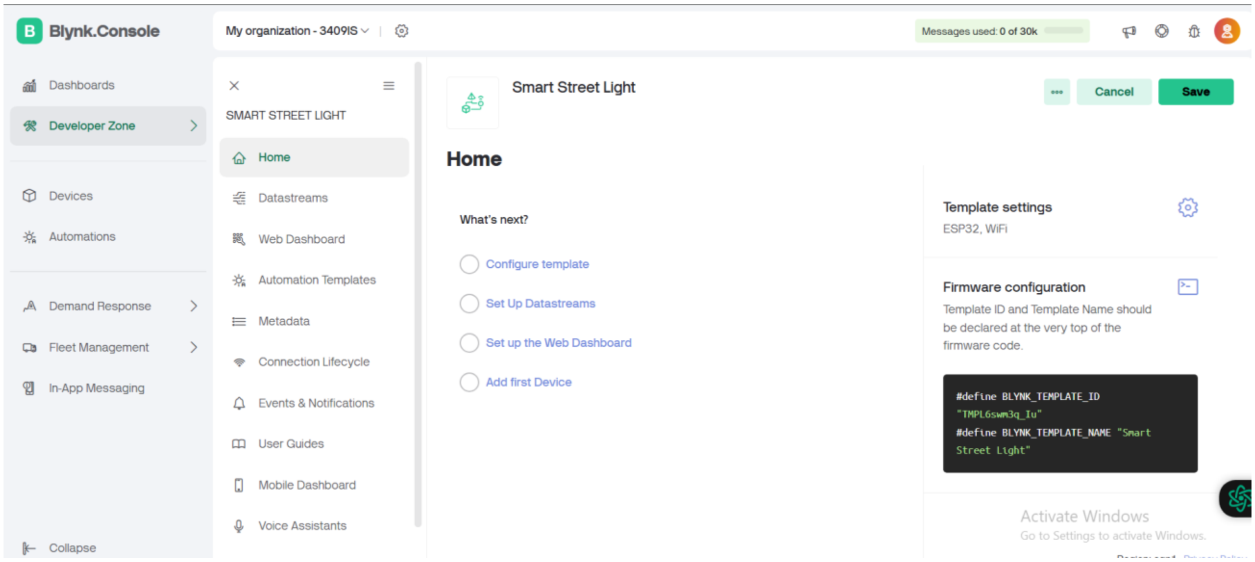
WiFi

DESCRIPTION

Controls LEDs and Display DHT11 data

36 / 128

Step 4: Once a new template is created you will see the below screen on your blynk account.



Smart Street Light

Home

What's next?

○ Configure template

○ Set Up Datastreams

○ Set up the Web Dashboard

○ Add first Device

Template settings

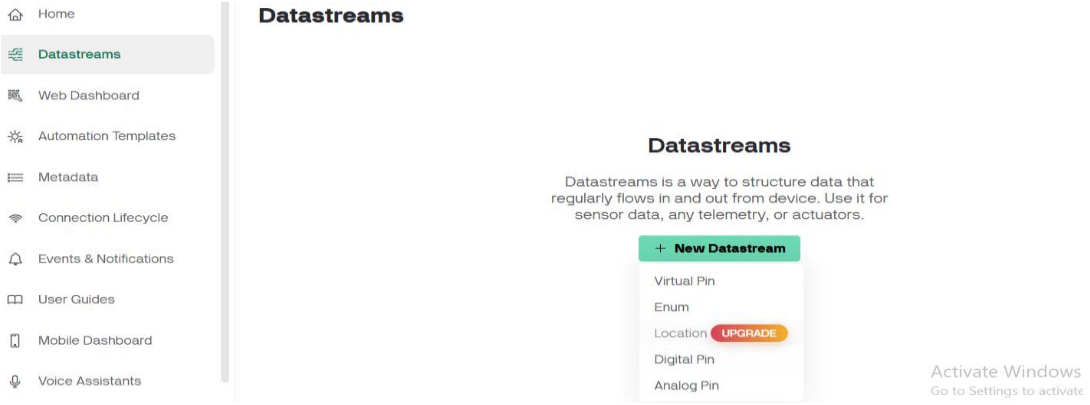
ESP32, WiFi

Firmware configuration

Template ID and Template Name should be declared at the very top of the firmware code.

```
#define BLYNK_TEMPLATE_ID
"TMPL6swm3q_Iu"
#define BLYNK_TEMPLATE_NAME "Smart
Street Light"
```

Step 5: Go to **Datastreams** and click on “**+ New Datastream**”. Then Choose “**Virtual Pin**”.

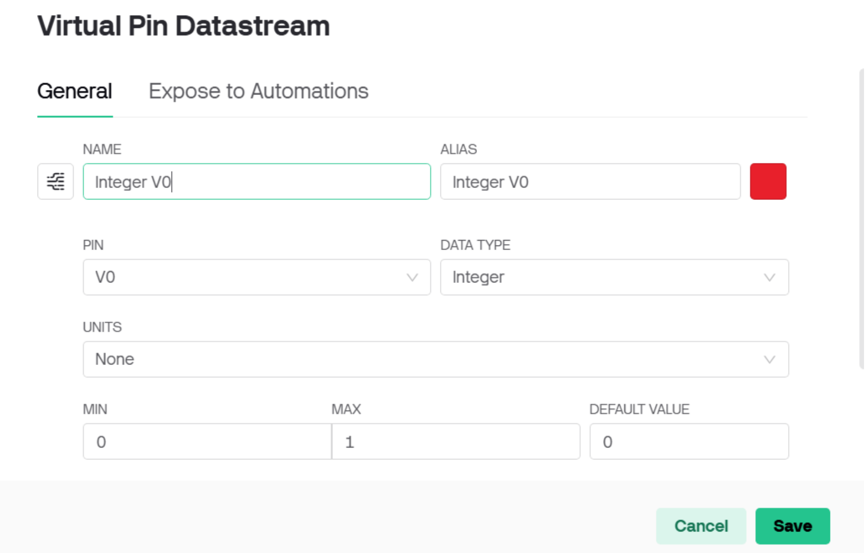


Datastreams

Datastreams

Datastreams is a way to structure data that regularly flows in and out from device. Use it for sensor data, any telemetry, or actuators.

Step 6: Name Datastream and select pin. Assign it’s PIN MODE as **Output** and Create.



Virtual Pin Datastream

General | Expose to Automations

NAME: Integer V0

ALIAS: Integer V0

PIN: V0

DATA TYPE: Integer

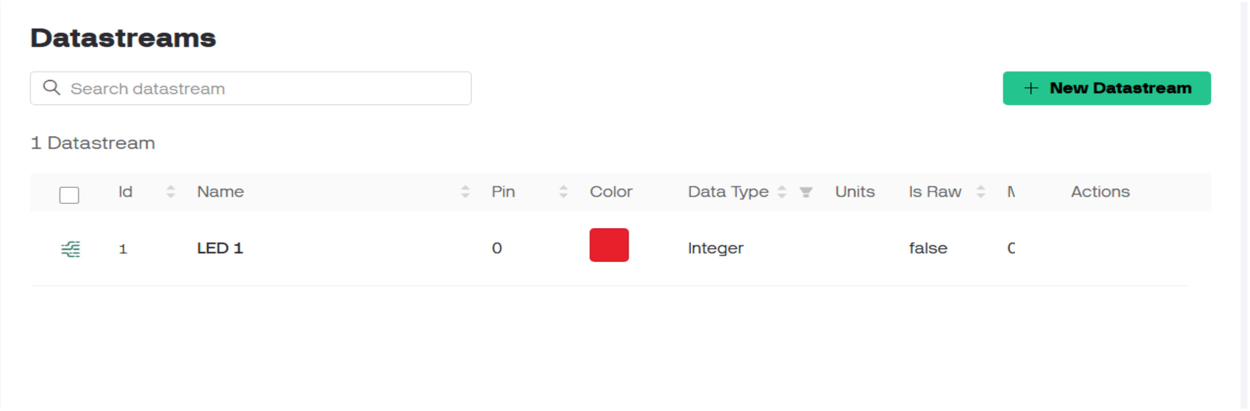
UNITS: None

MIN: 0

MAX: 1

DEFAULT VALUE: 0

Step 7: Once a datastream is created on the blynk website, it will look something like this below.

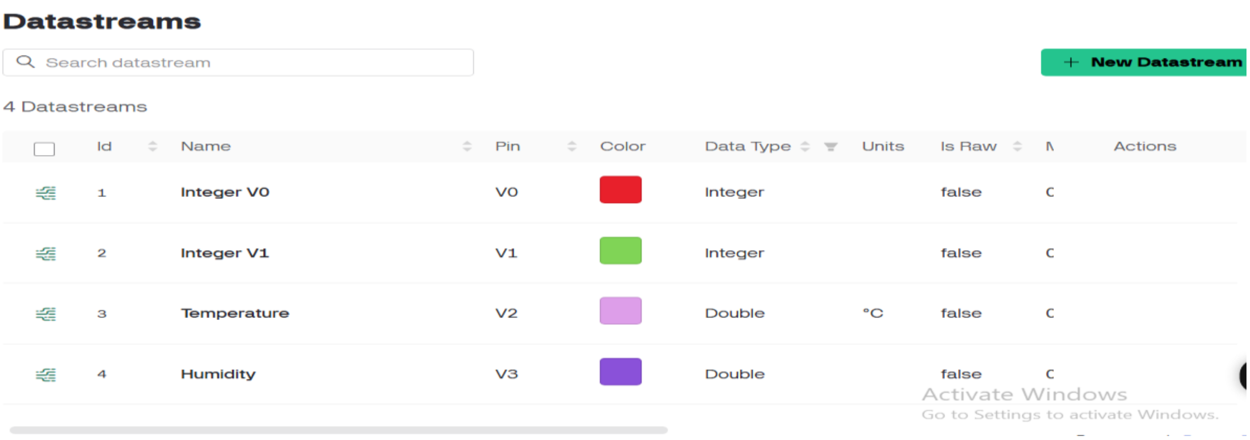


Datastreams

1 Datastream

□	Id	Name	Pin	Color	Data Type	Units	Is Raw	N	Actions
	1	LED 1	0	🟥	Integer		false	C	

Step 8: I have made one more data stream with the same values.

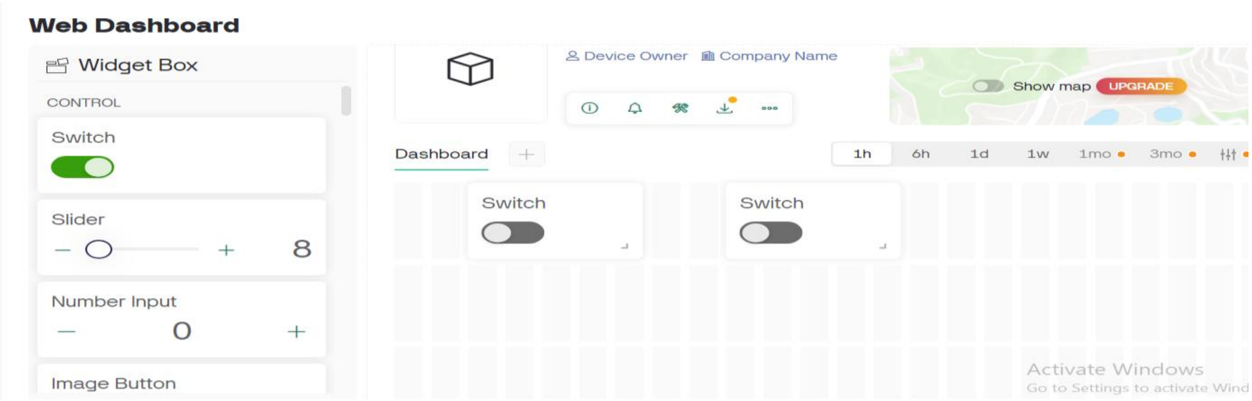


Datastreams

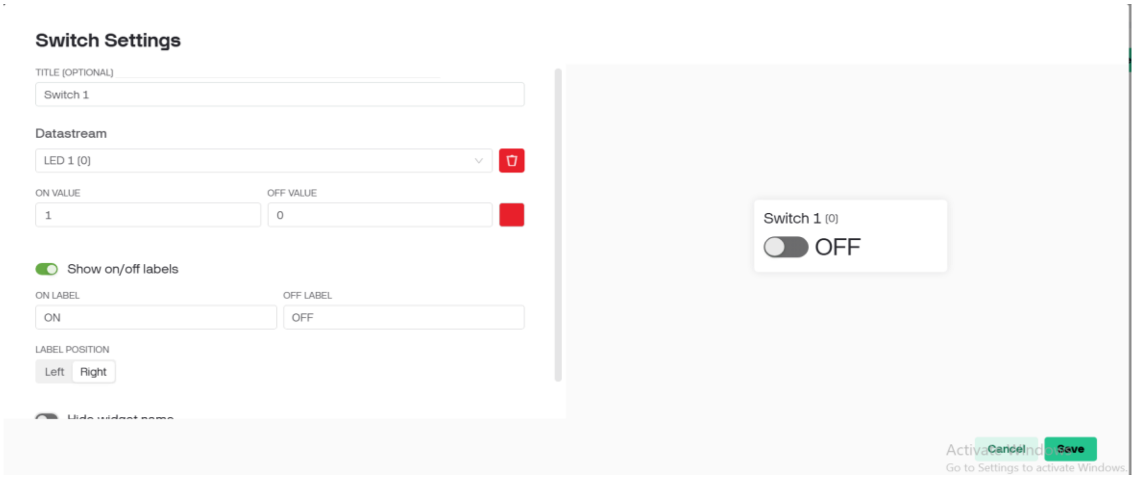
4 Datastreams

□	Id	Name	Pin	Color	Data Type	Units	Is Raw	N	Actions
	1	Integer V0	V0	🟥	Integer		false	C	
	2	Integer V1	V1	🟩	Integer		false	C	
	3	Temperature	V2	🟪	Double	°C	false	C	
	4	Humidity	V3	🟪	Double		false	C	

Step 9: Go to "**Web Dashboard**" and add Switch widgets.



Step 10: Click on the **Gear button**, give values according to your choice, and save.


Switch Settings

TITLE (OPTIONAL): Switch 1

Datastream: LED 1 (0)

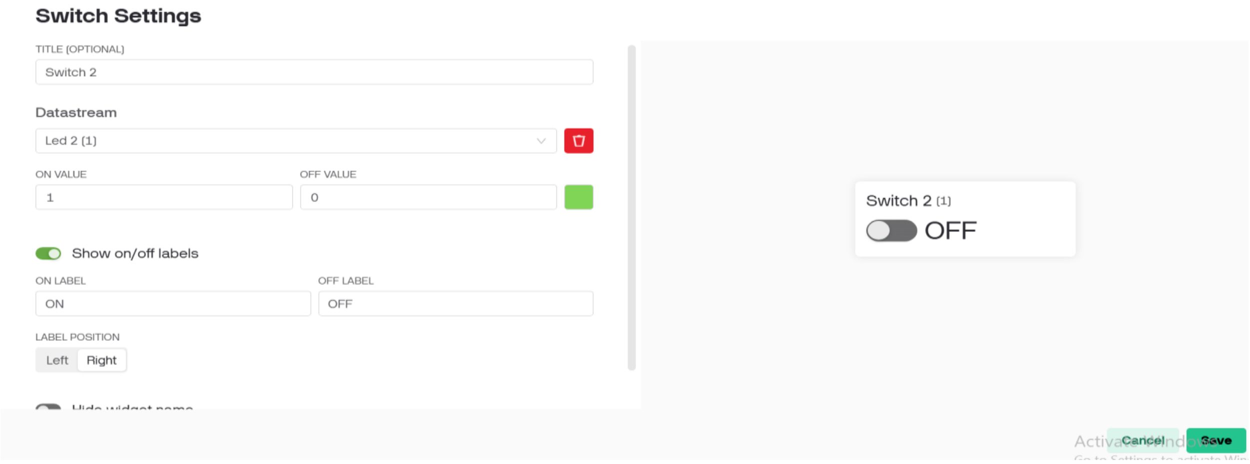
ON VALUE: 1
OFF VALUE: 0

Show on/off labels

ON LABEL: ON
OFF LABEL: OFF

LABEL POSITION: Left Right

Step 11: I have made one more switch and assigned all values. Also, I added 2 Gauges to display DHT11 sensor data.


Switch Settings

TITLE (OPTIONAL): Switch 2

Datastream: Led 2 (1)

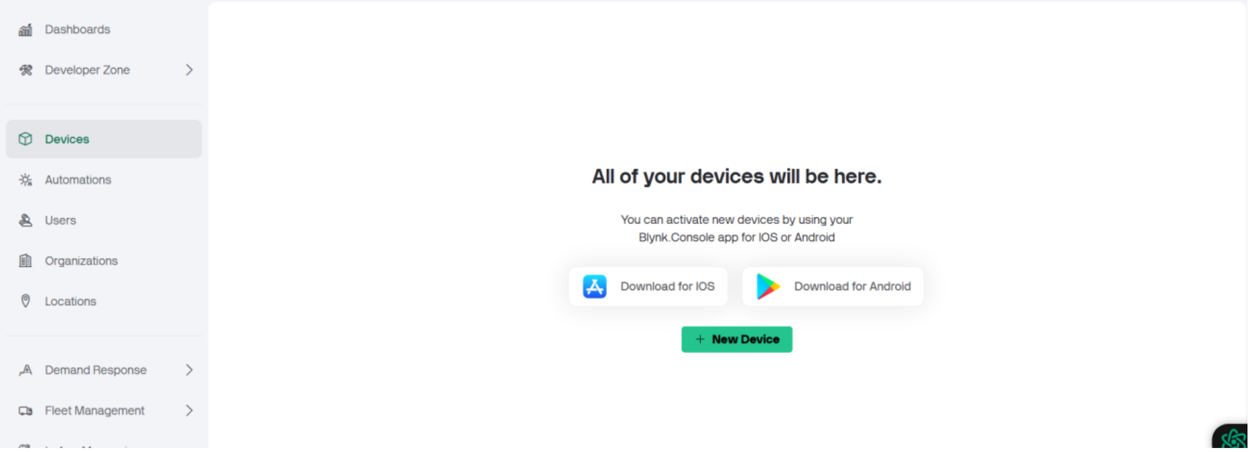
ON VALUE: 1
OFF VALUE: 0

Show on/off labels

ON LABEL: ON
OFF LABEL: OFF

LABEL POSITION: Left Right

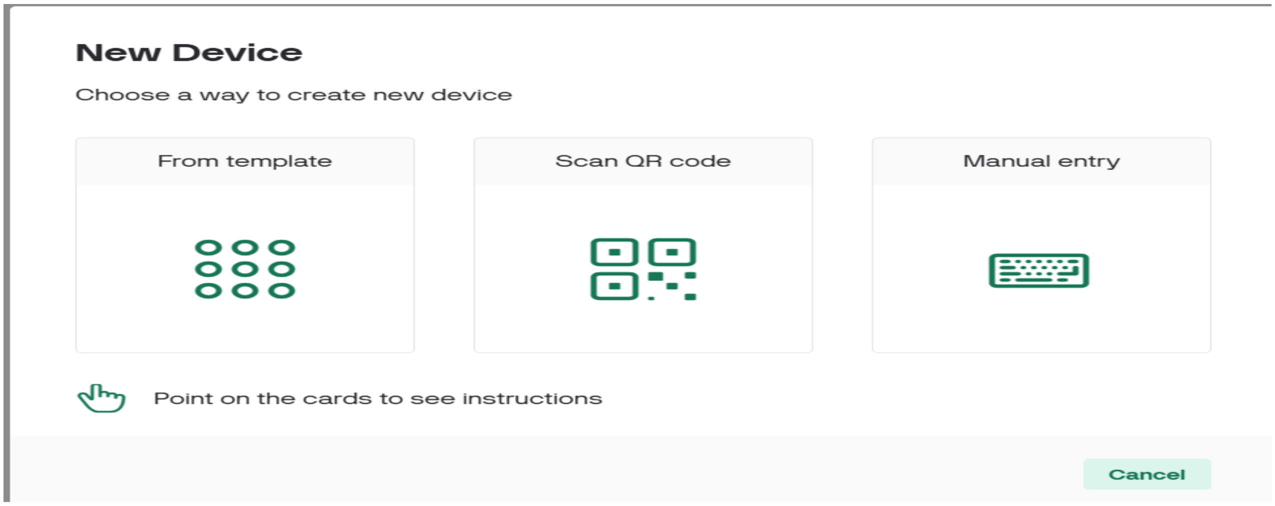
Step 12: Go to the lens icon and click on "**+ New Device**".



All of your devices will be here.

You can activate new devices by using your Blynk.Console app for IOS or Android

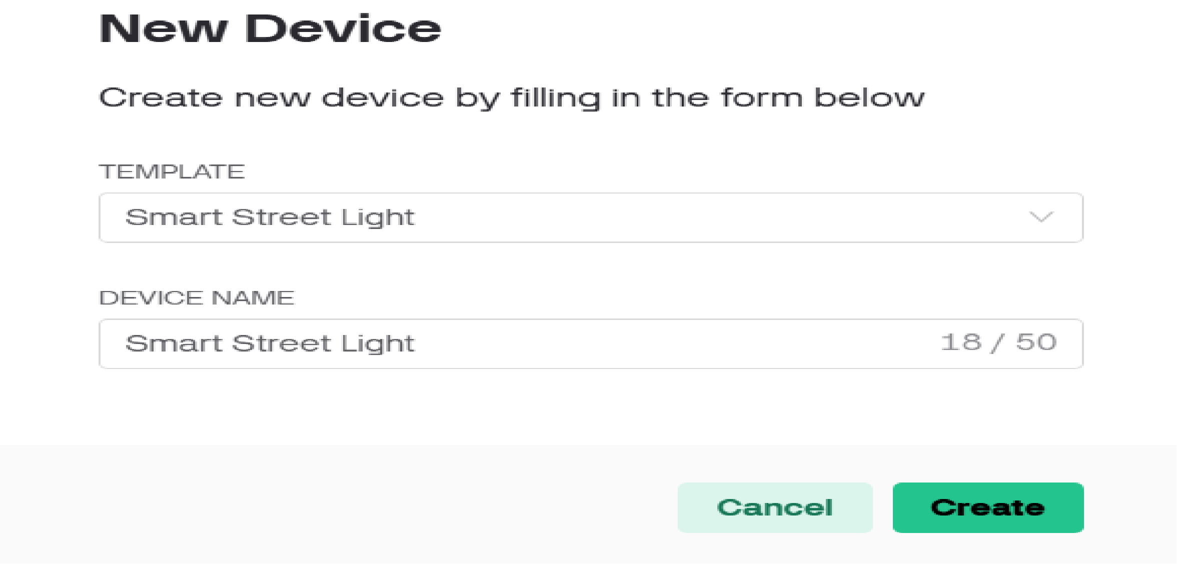
Step 13: Click "**From template**", select the template, and create.



New Device

Choose a way to create new device

Point on the cards to see instructions



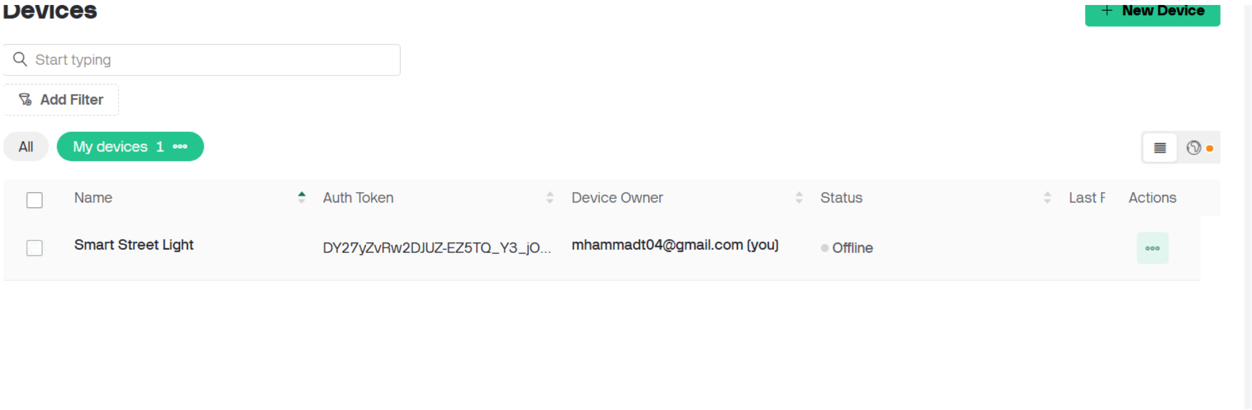
New Device

Create new device by filling in the form below

TEMPLATE: Smart Street Light

DEVICE NAME: Smart Street Light 18 / 50

Step 13: Now you can see the device. Click on “**Device Info**” and copy the **Auth Token**.



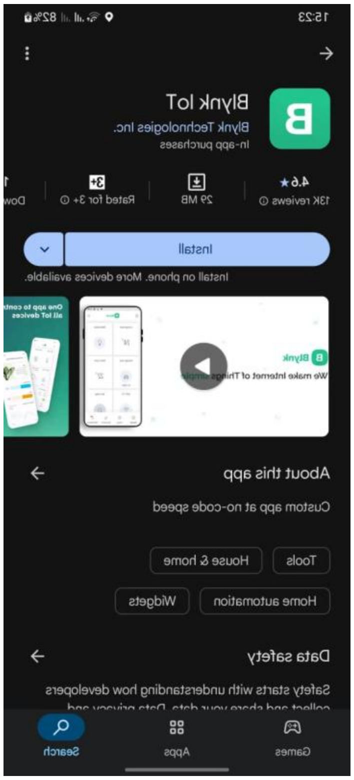
☐	Name	Auth Token	Device Owner	Status	Last F	Actions
☐	Smart Street Light	DY27yZvRw2DJUZ-EZ5TQ_Y3_jO...	mhammadt04@gmail.com (you)	Offline		•••

Blynk 2.0 Application Setup

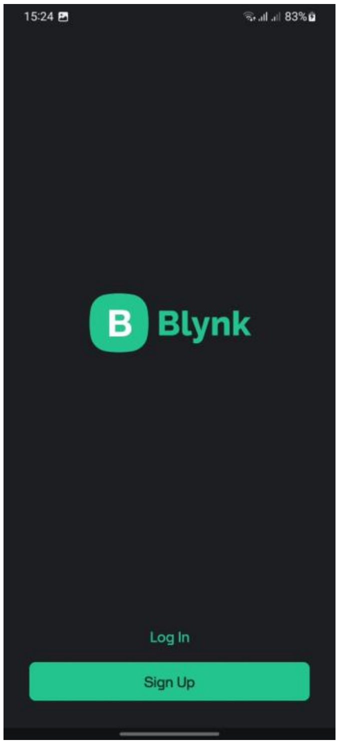
Now after Web setup, we have to make a **Mobile App Interface** so that we can able to control it from any **Android** or **IOS**

Step 1: Download the **Blynk app**.

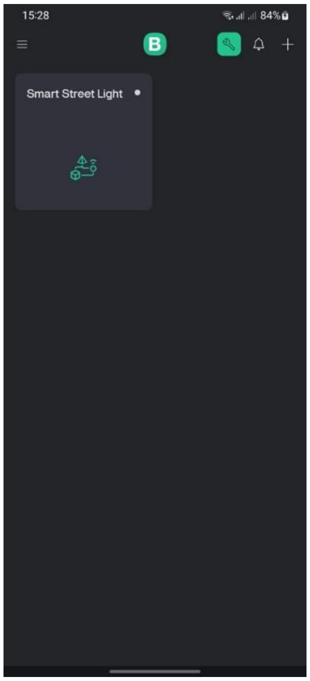
The Blynk app is available for both IOS and Android devices. You can download the app from the **App Store** or **Google Play**.



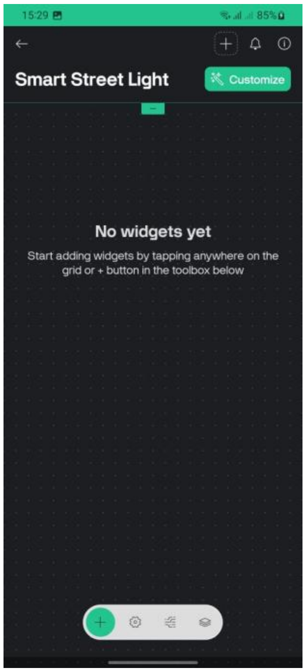
Step 2: Now open the **Blynk IOT app** and **Log in** with the credentials.



Step 4: Select the device we already created.



Step 5: Now you are on the dashboard of the device then click on the tool button to enter in **Developer Mode**.

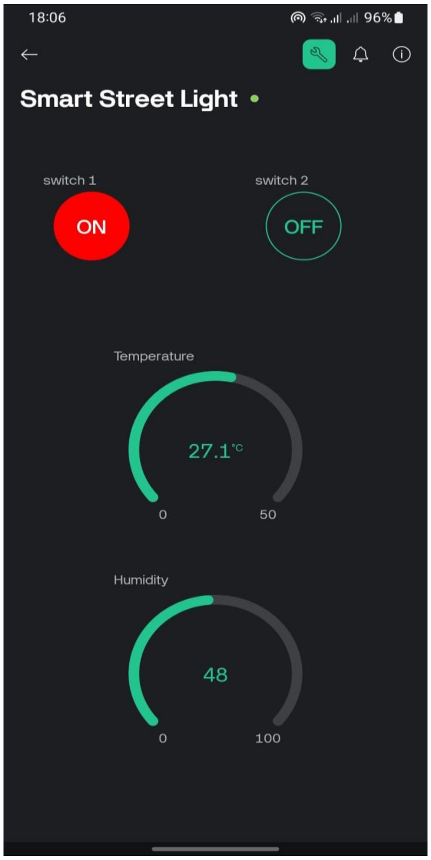


No widgets yet

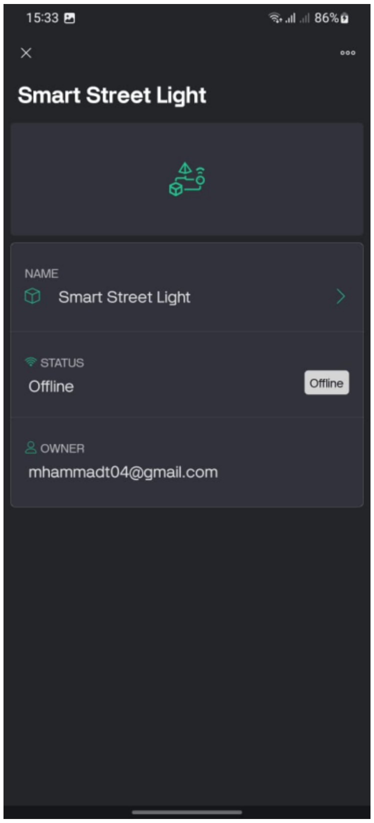
Start adding widgets by tapping anywhere on the grid or + button in the toolbox below

Step 6: Add Buttons and Gauges then click on the button and gauges icons to assign values like DataStream and Mode. You can also change its design and looks.

Step 7: You Have successfully created a device dashboard.



Step 8: You can check the device status also.



15:33

Smart Street Light

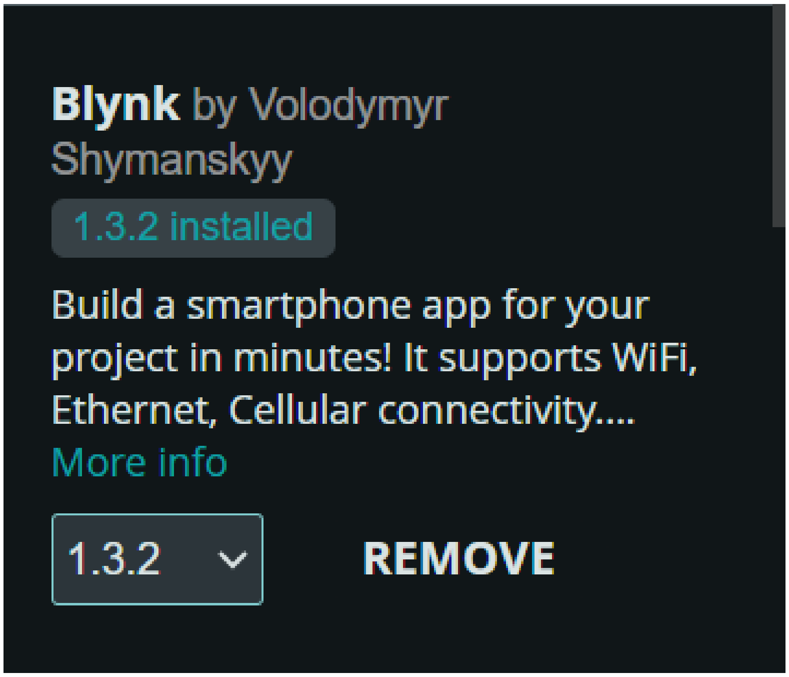
NAME
Smart Street Light

STATUS
Offline

OWNER
mhammadt04@gmail.com

Setup Arduino IDE

Step 1: Open Arduino IDE, Then go to the “LIBRARY MANAGER” and install the Blynk library.



Blynk by Volodymyr Shymanskyy

1.3.2 installed

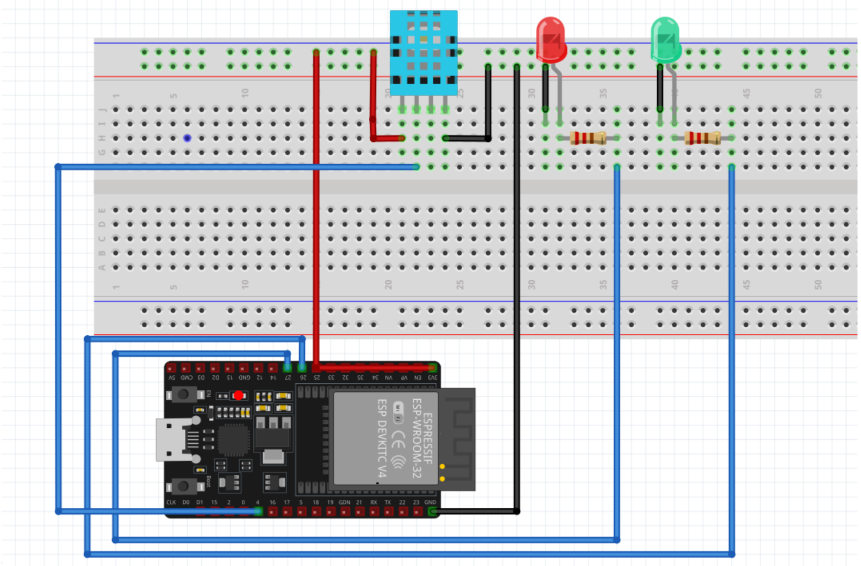
Build a smartphone app for your project in minutes! It supports WiFi, Ethernet, Cellular connectivity....

ESP32 Setup

Now we will set esp32, our DHT11 sensor, and 2 LEDs to perform this task. For this, we need to take the following steps:

Circuit Diagram:

We will connect 2 LEDs along with a safety resistance of 220 ohms whose negative terminal is connected to the ground and the positive terminal is connected with GPIO pins 26, and 27 through resistances they will act as actuators that we will control through the Blynk app. For DHT11 we will use it as the sensor to display its data on Blynk. For this, we provide the power supply to it and connect its data pin to D4. As shown in following diagram:

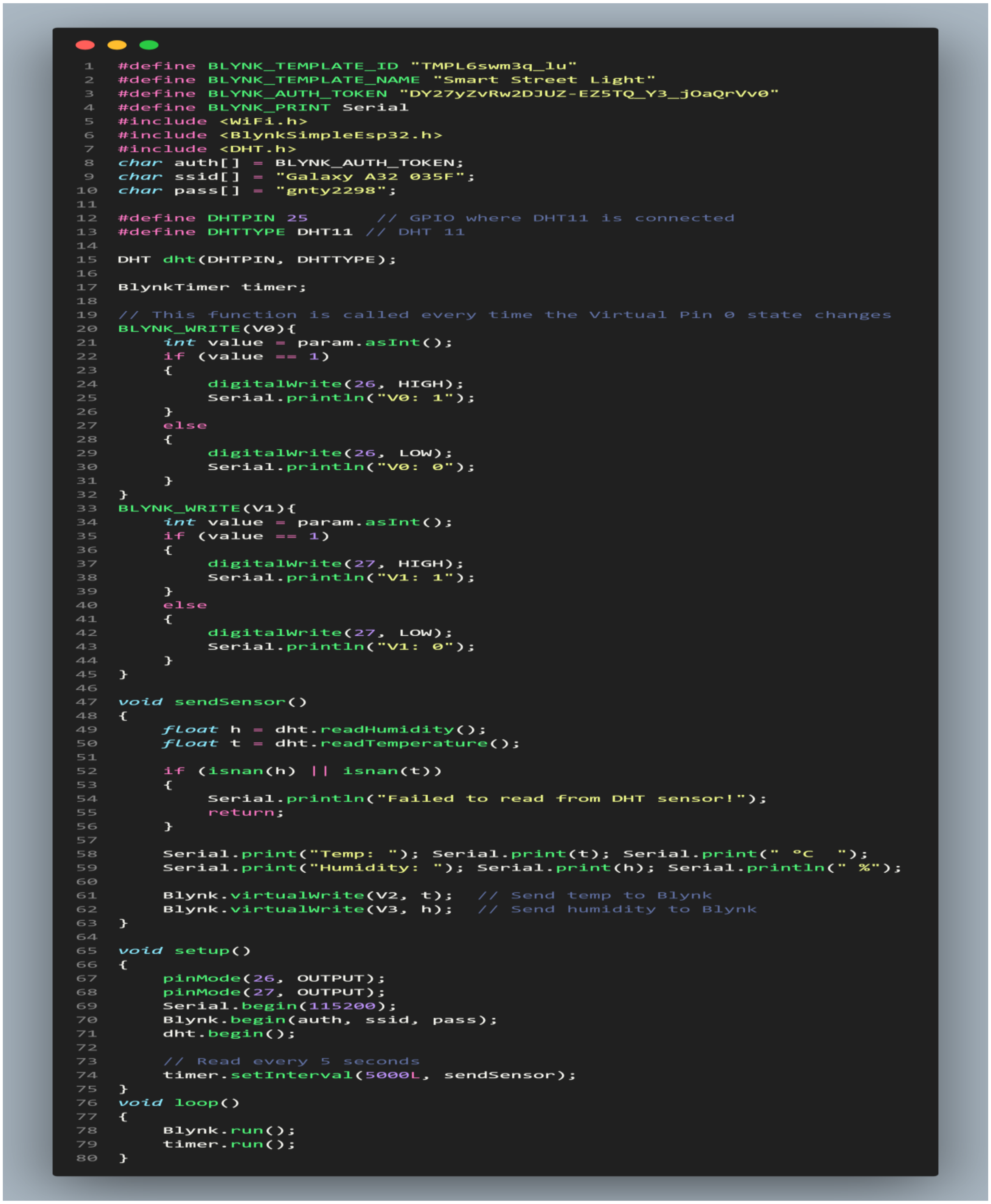


Code Snippet & Explanation

Code:

```
#define BLYNK_TEMPLATE_ID "TMPL6swm3q_lu"
#define BLYNK_TEMPLATE_NAME "Smart Street Light"
#define BLYNK_AUTH_TOKEN "DY27yZvRw2DJUZ-EZ5TQ_Y3_jOaQrVv0"
#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <BlynkSimpleEsp32.h>
#include <DHT.h>
char auth[] = BLYNK_AUTH_TOKEN;
char ssid[] = "Galaxy A32 035F";
char pass[] = "gnty2298";

#define DHTPIN 25      // GPIO where DHT11 is connected
#define DHTTYPE DHT11 // DHT 11

DHT dht(DHTPIN, DHTTYPE);
```

```
BlynkTimer timer;

// This function is called every time the Virtual Pin 0 state changes
BLYNK_WRITE(V0){
    int value = param.asInt();
    if (value == 1)
    {
        digitalWrite(26, HIGH);
        Serial.println("V0: 1");
    }
    else
    {
        digitalWrite(26, LOW);
        Serial.println("V0: 0");
    }
}
BLYNK_WRITE(V1){
    int value = param.asInt();
    if (value == 1)
    {
        digitalWrite(27, HIGH);
        Serial.println("V1: 1");
    }
    else
    {
        digitalWrite(27, LOW);
        Serial.println("V1: 0");
    }
}

void sendSensor()
{
    float h = dht.readHumidity();
    float t = dht.readTemperature();

    if (isnan(h) || isnan(t))
    {
        Serial.println("Failed to read from DHT sensor!");
        return;
    }

    Serial.print("Temp: "); Serial.print(t); Serial.print(" °C  ");
    Serial.print("Humidity: "); Serial.print(h); Serial.println(" %");

    Blynk.virtualWrite(V2, t);  // Send temp to Blynk
    Blynk.virtualWrite(V3, h);  // Send humidity to Blynk
}

void setup()
{
    pinMode(26, OUTPUT);
    pinMode(27, OUTPUT);
    Serial.begin(115200);
    Blynk.begin(auth, ssid, pass);
    dht.begin();

    // Read every 5 seconds
    timer.setInterval(5000L, sendSensor);
}
void loop()
{
    Blynk.run();
    timer.run();
}
```



```
#define BLYNK_TEMPLATE_ID "TMPL6swm3q_lu"
#define BLYNK_TEMPLATE_NAME "Smart Street Light"
#define BLYNK_AUTH_TOKEN "DY27yZvRw2DJUZ-EZ5TQ_Y3_jOaQrVv0"
#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <BlynkSimpleEsp32.h>
#include <DHT.h>
char auth[] = BLYNK_AUTH_TOKEN;
char ssid[] = "Galaxy A32 035F";
char pass[] = "gnty2298";

#define DHTPIN 25       // GPIO where DHT11 is connected
#define DHTTYPE DHT11 // DHT 11

DHT dht(DHTPIN, DHTTYPE);

BlynkTimer timer;

// This function is called every time the Virtual Pin 0 state changes
BLYNK_WRITE(V0){
    int value = param.asInt();
    if (value == 1)
    {
        digitalWrite(26, HIGH);
        Serial.println("V0: 1");
    }
    else
    {
        digitalWrite(26, LOW);
        Serial.println("V0: 0");
    }
}
BLYNK_WRITE(V1){
    int value = param.asInt();
    if (value == 1)
    {
        digitalWrite(27, HIGH);
        Serial.println("V1: 1");
    }
    else
    {
        digitalWrite(27, LOW);
        Serial.println("V1: 0");
    }
}

void sendSensor()
{
    float h = dht.readHumidity();
    float t = dht.readTemperature();

    if (isnan(h) || isnan(t))
    {
        Serial.println("Failed to read from DHT sensor!");
        return;
    }

    Serial.print("Temp: "); Serial.print(t); Serial.print(" °C  ");
    Serial.print("Humidity: "); Serial.print(h); Serial.println(" %");

    Blynk.virtualWrite(V2, t);  // Send temp to Blynk
    Blynk.virtualWrite(V3, h);  // Send humidity to Blynk
}

void setup()
{
    pinMode(26, OUTPUT);
    pinMode(27, OUTPUT);
    Serial.begin(115200);
    Blynk.begin(auth, ssid, pass);
    dht.begin();

    // Read every 5 seconds
    timer.setInterval(5000L, sendSensor);
}
void loop()
{
    Blynk.run();
    timer.run();
}
```

Code Explanation

```
#define BLYNK_TEMPLATE_ID “xxxxxxxxxxxx"

#define BLYNK_TEMPLATE_NAME "xxxx"

#define BLYNK_AUTH_TOKEN "Your Auth Token"

#define BLYNK_PRINT Serial
```

Here we have defined the **Blynk template ID**, template name, and authentication token.

Don’t forget to replace "xxxxxxxxxxxx" with your actual template ID, “xxxx” with your template name, and "Your Auth Token" with your Blynk authentication token. You can find it in the “Device info” of the web device dashboard or in the Blynk app.

```
char ssid[] = "Your SSID";

char pass[] = "Your Password";
```

Remember to replace "**Your SSID**" with your WiFi network name and "**Your Password**" with your WiFi password.

```
#define DHTPIN 25      // GPIO where DHT11 is connected

#define DHTTYPE DHT11 // DHT 11


DHT dht(DHTPIN, DHTTYPE);


BlynkTimer timer;
```

Here we define GPIO pin 25 as DHTPIN and create a DHT object to handle sensor data. And we made a timer variable from the Blynk library so that we can use it to publish data after a specific interval.

```
BLYNK_WRITE(V0) {

    // Handles changes in Virtual Pin V0 state

}

BLYNK_WRITE(V1) {

    // Handles changes in Virtual Pin V1 state

}
```

Inside these functions, the code checks the received value. If the value is 1, it sets the corresponding digital pin (D0 or D1) to HIGH. If the value is 0, it sets the digital pin to LOW.

```
void sendSensor(){
  float h = dht.readHumidity();
  float t = dht.readTemperature();

  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  Serial.print("Temp: "); Serial.print(t); Serial.print(" °C  ");
  Serial.print("Humidity: "); Serial.print(h); Serial.println(" %");

  Blynk.virtualWrite(V2, t);  // Send temp to Blynk
  Blynk.virtualWrite(V3, h);  // Send humidity to Blynk
}
```

This function reads the Temperature and Humidity data from the sensor and Publish them on respective Virtual PINs.

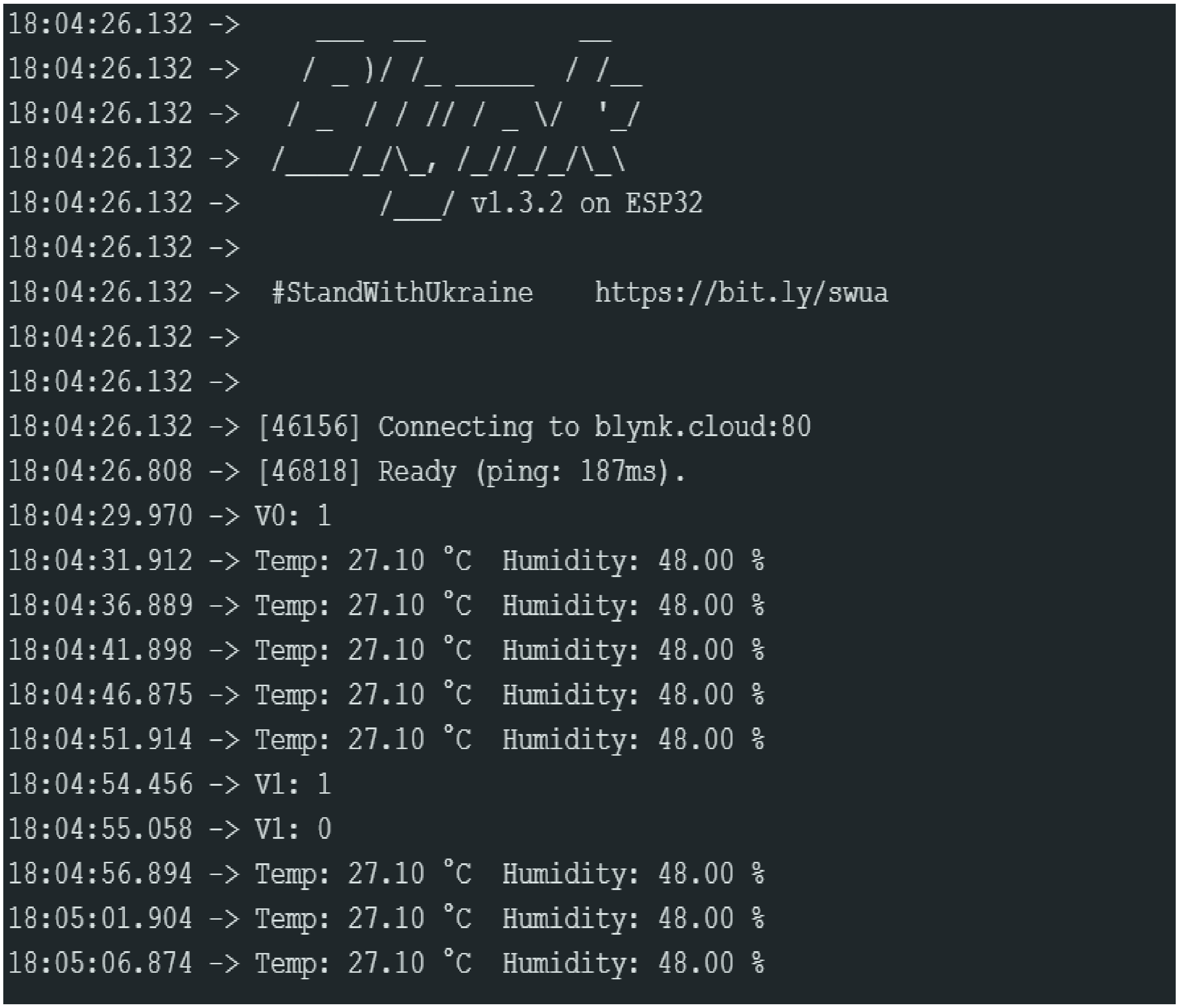
```
void setup() {
    pinMode(D0, OUTPUT);
    pinMode(D1, OUTPUT);
    Serial.begin(115200);
    Blynk.begin(auth, ssid, pass);
    dht.begin();

    // Read every 5 seconds
    timer.setInterval(5000L, sendSensor);
}
```

In the setup function, the **digital pins D0** and **D1** are configured as **output pins**. Serial communication starts and the board connects to the Blynk server using the provided **WiFi credentials** and **authentication token**. Also, it set a timer along it, it binds a function that we created in previously to handle and send the sensor data.

```
void loop() {

    Blynk.run();

    timer.run();

}
```

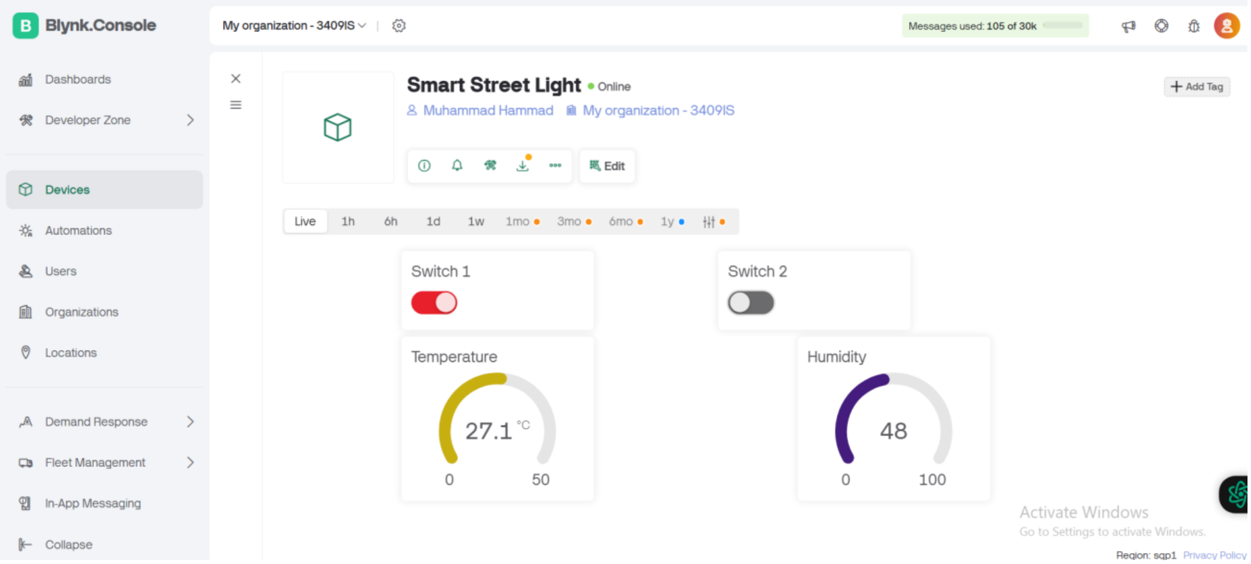
The loop function continuously calls **Blynk.run()**, **timer.run()** allowing the Blynk library to process any incoming commands from the Blynk app and post sensor data to dashboard.

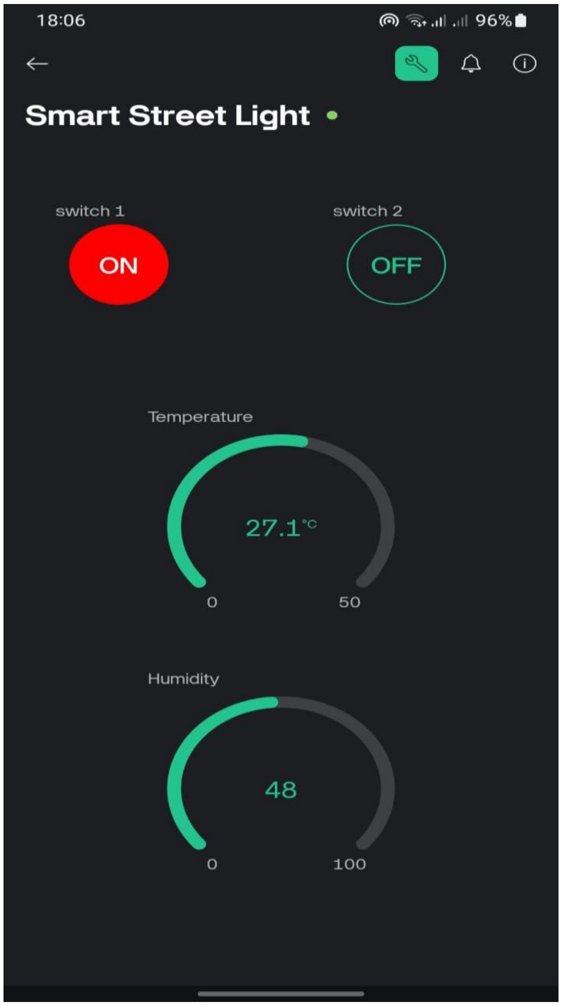
Results (Screenshots serial Monitor):

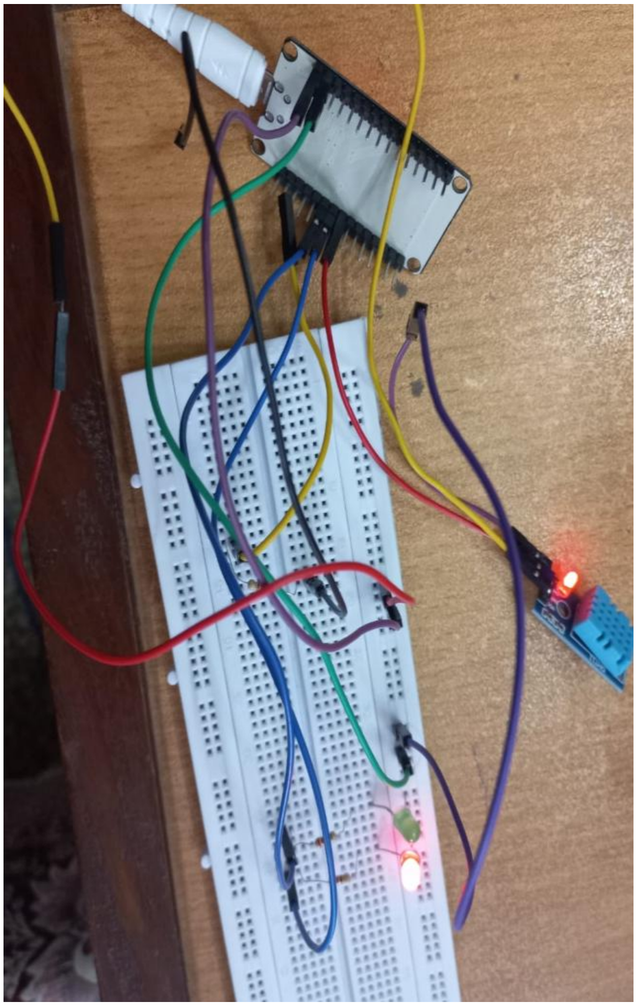


```
18:04:26.132 ->     ___  __          __
18:04:26.132 ->    / _ )/ /_ _____  / /__
18:04:26.132 ->   / _  / / // / _ \/  '_/
18:04:26.132 ->  /____/_/\_, /_//_/_/\_\
18:04:26.132 ->         /___/ v1.3.2 on ESP32
18:04:26.132 ->
18:04:26.132 ->  #StandWithUkraine    https://bit.ly/swua
18:04:26.132 ->
18:04:26.132 ->
18:04:26.132 -> [46156] Connecting to blynk.cloud:80
18:04:26.808 -> [46818] Ready (ping: 187ms).
18:04:29.970 -> V0: 1
18:04:31.912 -> Temp: 27.10 °C  Humidity: 48.00 %
18:04:36.889 -> Temp: 27.10 °C  Humidity: 48.00 %
18:04:41.898 -> Temp: 27.10 °C  Humidity: 48.00 %
18:04:46.875 -> Temp: 27.10 °C  Humidity: 48.00 %
18:04:51.914 -> Temp: 27.10 °C  Humidity: 48.00 %
18:04:54.456 -> V1: 1
18:04:55.058 -> V1: 0
18:04:56.894 -> Temp: 27.10 °C  Humidity: 48.00 %
18:05:01.904 -> Temp: 27.10 °C  Humidity: 48.00 %
18:05:06.874 -> Temp: 27.10 °C  Humidity: 48.00 %
```

Dashboards:


Blynk.Console
My organization - 3409IS
Messages used: 105 of 30k
Dashboards
Developer Zone
Devices
Automations
Users
Organizations
Locations
Demand Response
Fleet Management
In-App Messaging
Collapse
Smart Street Light
Online
Muhammad Hammad
My organization - 3409IS
Edit
Add Tag
Live
1h
6h
1d
1w
1mo
3mo
6mo
1y
Switch 1
Switch 2
Temperature
27.1 °C
0
50
Humidity
48
0
100
Activate Windows
Go to Settings to activate Windows.
Region: sgp1
Privacy Policy


18:06
96%
Smart Street Light
switch 1
ON
switch 2
OFF
Temperature
27.1°C
0
50
Humidity
48
0
100



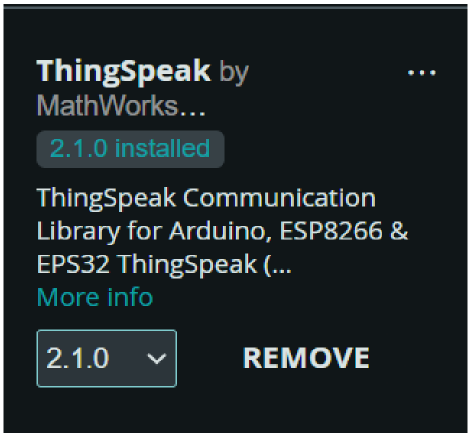
Challenges & Solutions

- A stable Internet Connection is required for the proper functioning of the app
- Correct Selection of DataStream Pins As mentioned in the tutorial they were using digital pins but in the Arduino code, they are using virtual Pins that confirm to resolve this we use virtual pins in the Blynk app also.

3. ThingSpeak

Overview of ThingSpeak

ThingSpeak allows you to publish your sensor readings to their website and plot them in charts with timestamps. Then, you can access your readings from anywhere in the world. So we need the thingspeak library in Ardino IDE.



ThingSpeak by MathWorks...

2.1.0 installed

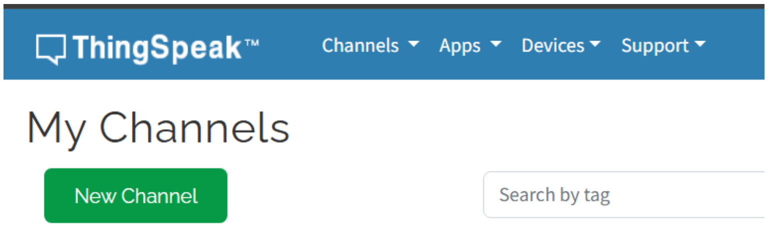
ThingSpeak Communication Library for Arduino, ESP8266 & EPS32 ThingSpeak (...

Configuration Steps

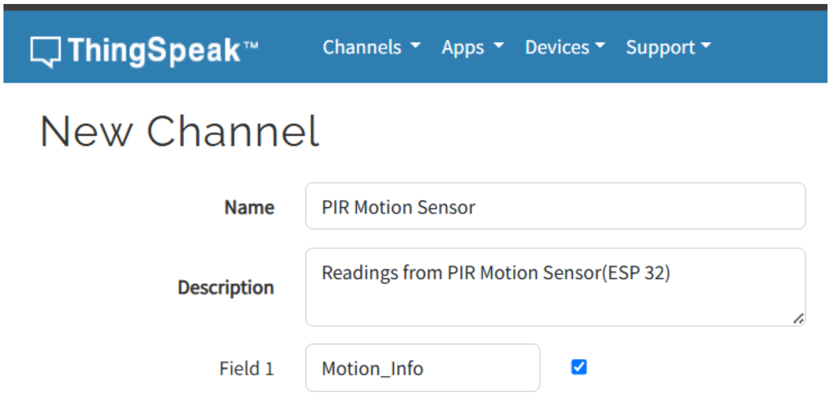
To configure things speak to use we need to follow following steps:

Channel Creation

Now we create our account at thingspeak website which is the platform provided by MathWorks. Then open the “**Channels**” tab and select “**My Channels**”.



By clicking the **New Channel** button, we gave the name and description of the channel. For motion data, we only need one field so we enable and named it accordingly.



New Channel

Name: PIR Motion Sensor

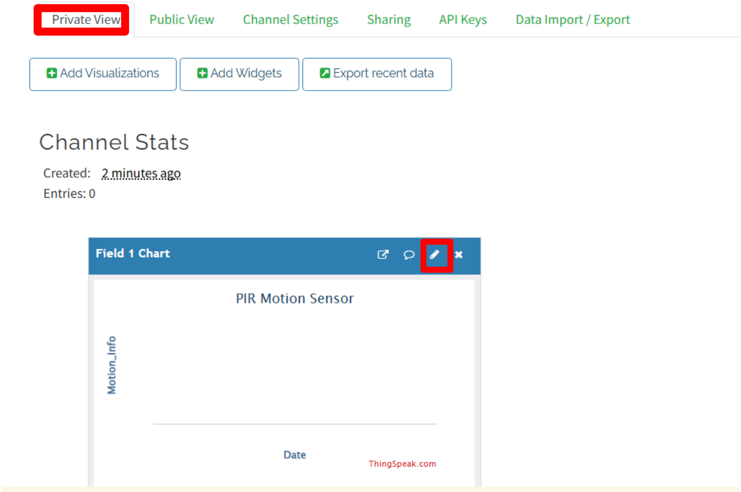
Description: Readings from PIR Motion Sensor(ESP 32)

Field 1: Motion_Info ☑

Then we click the **save** button.

Customizing Chart

Now we customize the chart, go to your **Private View** tab and click on the edit icon.

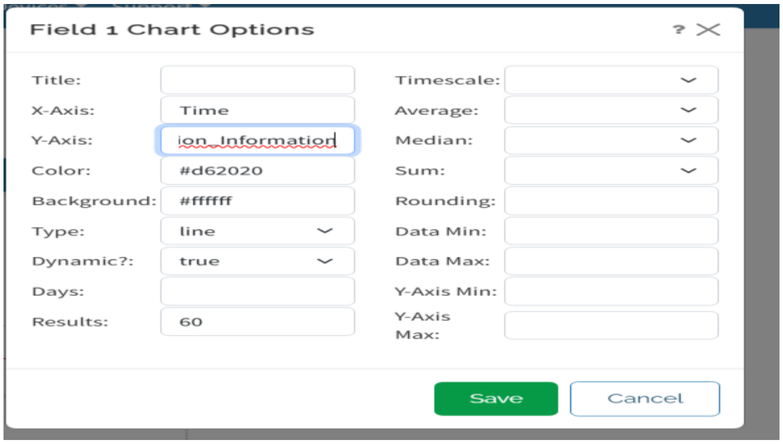


Channel Stats

Created: 2 minutes ago

Entries: 0

we gave a title to our chart, customize the background color, x and y axis.



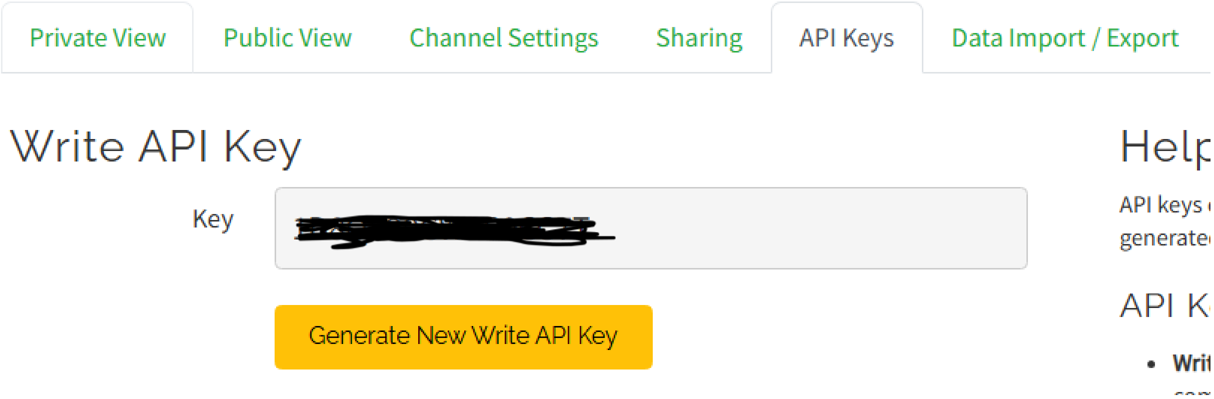
Field 1 Chart Options

Title:		Timescale:	
X-Axis:	Time	Average:	
Y-Axis:	ion_Information	Median:	
Color:	#d62020	Sum:	
Background:	#ffffff	Rounding:	
Type:	line	Data Min:	
Dynamic?:	true	Data Max:	
Days:		Y-Axis Min:	
Results:	60	Y-Axis Max:	

Then press the "**Save**" button.

API Key Integration

To send values from the ESP8266 to ThingSpeak, we need the Write API Key. We open the "**API Keys**" tab and copy the Write API Key to a safe place because we'll need it in a moment.



Write API Key

Key [redacted]

Help

API keys
generate

API K

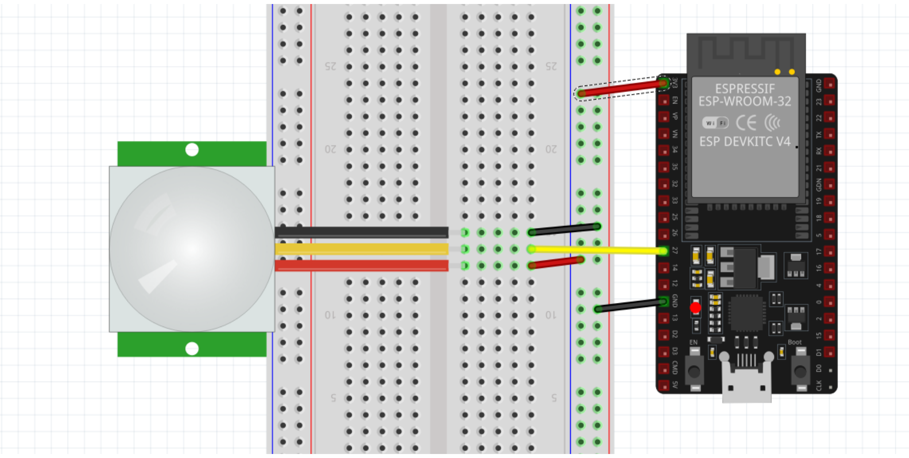
- Writ

ESP32 Setup

Now we will set esp32 and our PIR sensor to send data. For this, we need to take the following steps:

Circuit Diagram:

We'll be using the Mini AM312 PIR Motion Sensor that operates at 3.3V. Its middle pin is a data pin that will be connected to GPIO 27 and the 2 pins are ground and VCC that are connected respectively as shown n the circuit diagram.

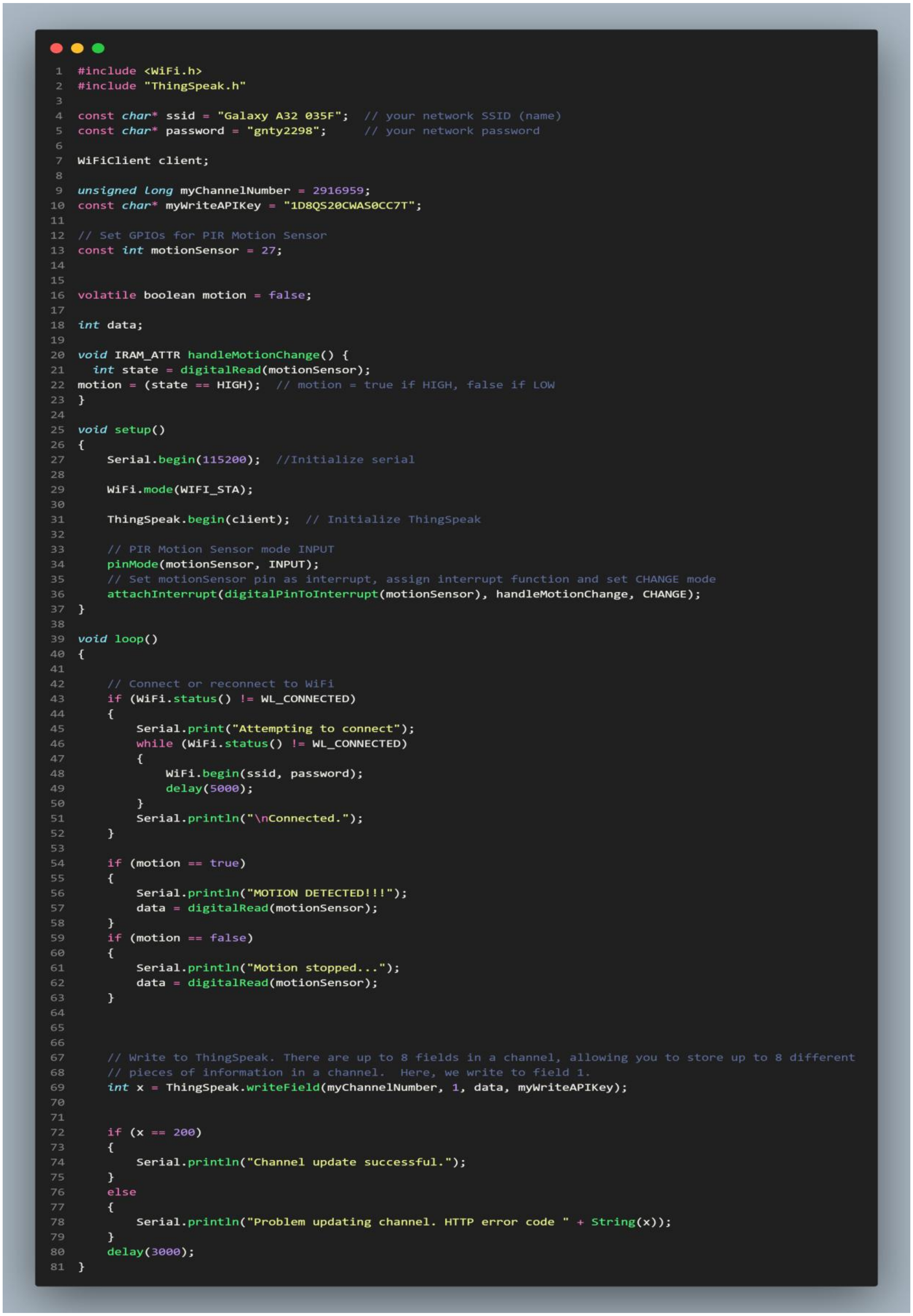


Code Snippet & Explanation

Code:

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

const char* ssid = "Galaxy A32 035F";  // your network SSID (name)
const char* password = "gnty2298";     // your network password

WiFiClient client;

unsigned long myChannelNumber = 2916959;
const char* myWriteAPIKey = "1D8QS20CWAS0CC7T";

// Set GPIOs for PIR Motion Sensor
const int motionSensor = 27;


volatile boolean motion = false;

int data;

void IRAM_ATTR handleMotionChange() {
  int state = digitalRead(motionSensor);
motion = (state == HIGH);  // motion = true if HIGH, false if LOW
}

void setup()
{
    Serial.begin(115200);  //Initialize serial

    WiFi.mode(WIFI_STA);

    ThingSpeak.begin(client);  // Initialize ThingSpeak

    // PIR Motion Sensor mode INPUT
    pinMode(motionSensor, INPUT);
    // Set motionSensor pin as interrupt, assign interrupt function and set CHANGE
mode
```

```
    attachInterrupt(digitalPinToInterrupt(motionSensor), handleMotionChange,
CHANGE);
}

void loop()
{

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

    if (motion == true)
    {
        Serial.println("MOTION DETECTED!!!");
        data = digitalRead(motionSensor);
    }
    if (motion == false)
    {
        Serial.println("Motion stopped...");
        data = digitalRead(motionSensor);
    }



    // Write to ThingSpeak. There are up to 8 fields in a channel, allowing you to
store up to 8 different
    // pieces of information in a channel.  Here, we write to field 1.
    int x = ThingSpeak.writeField(myChannelNumber, 1, data, myWriteAPIKey);


    if (x == 200)
    {
        Serial.println("Channel update successful.");
    }
    else
    {
        Serial.println("Problem updating channel. HTTP error code " + String(x));
    }
    delay(3000);
}
```



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

const char* ssid = "Galaxy A32 035F";  // your network SSID (name)
const char* password = "gnty2298";     // your network password

WiFiClient client;

unsigned long myChannelNumber = 2916959;
const char* myWriteAPIKey = "1D8QS20CWAS0CC7T";

// Set GPIOs for PIR Motion Sensor
const int motionSensor = 27;


volatile boolean motion = false;

int data;

void IRAM_ATTR handleMotionChange() {
  int state = digitalRead(motionSensor);
motion = (state == HIGH);  // motion = true if HIGH, false if LOW
}

void setup()
{
    Serial.begin(115200);  //Initialize serial

    WiFi.mode(WIFI_STA);

    ThingSpeak.begin(client);  // Initialize ThingSpeak

    // PIR Motion Sensor mode INPUT
    pinMode(motionSensor, INPUT);
    // Set motionSensor pin as interrupt, assign interrupt function and set CHANGE mode
    attachInterrupt(digitalPinToInterrupt(motionSensor), handleMotionChange, CHANGE);
}

void loop()
{

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

    if (motion == true)
    {
        Serial.println("MOTION DETECTED!!!");
        data = digitalRead(motionSensor);
    }
    if (motion == false)
    {
        Serial.println("Motion stopped...");
        data = digitalRead(motionSensor);
    }



    // Write to ThingSpeak. There are up to 8 fields in a channel, allowing you to store up to 8 different
    // pieces of information in a channel.  Here, we write to field 1.
    int x = ThingSpeak.writeField(myChannelNumber, 1, data, myWriteAPIKey);


    if (x == 200)
    {
        Serial.println("Channel update successful.");
    }
    else
    {
        Serial.println("Problem updating channel. HTTP error code " + String(x));
    }
    delay(3000);
}
```

Code explanation:

To make the code work, insert our network credentials in the following variables:

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

we insert the number of the channel that we're publishing to. If you only have one channel created in ThingSpeak, the channel number is 1 or we can use unique channel ID that thingspeak gave us.

```
unsigned long myChannelNumber = 1;
```

Finally, we insert the Write API key from the previous steps:

```
const char * myWriteAPIKey = "XXXXXXXXXXXXXXXX";
```

How the Code Works

First, we include the necessary libraries.

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

Insert our network credentials in the following variables:

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Create a Wi-Fi client to connect to ThingSpeak.

```
WiFiClient client;
```

Insert your channel number as well as your write API key:

```
unsigned long myChannelNumber = 2916959;
const char * myWriteAPIKey = "XXXXXXXXXXXXXXXX";
```

The motionSensor is the GPIO PIN of ESP 32.

```
const int motionSensor = 27;
```

motion is the Boolean variable that tells us that if the sensor detects any motion and volatile tell the compiler not to optimize it because it can be modified during the interrupt routine.

```
volatile boolean motion = false;
```

In the setup(), initialize the Serial Monitor:

```
Serial.begin(115200);  //Initialize serial
```

Set the ESP8266 as a Wi-Fi station:

```
WiFi.mode(WIFI_STA);
```

Initialize ThingSpeak:

```
ThingSpeak.begin(client);  // Initialize ThingSpeak
```

In the loop(), connect or reconnect to Wi-Fi in case the connection was lost:

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

Then we check if the motion is detected or not print the respective message and store the corresponding value that is 0 or 1 in the data variable.

```
if (motion == true)
    {
        Serial.println("MOTION DETECTED!!!");
        data = digitalRead(motionSensor);
    }
    if (motion == false)
    {
        Serial.println("Motion stopped...");
        data = digitalRead(motionSensor);
    }
```

Finally, write to ThingSpeak. We use the writeField() method that accepts as arguments:

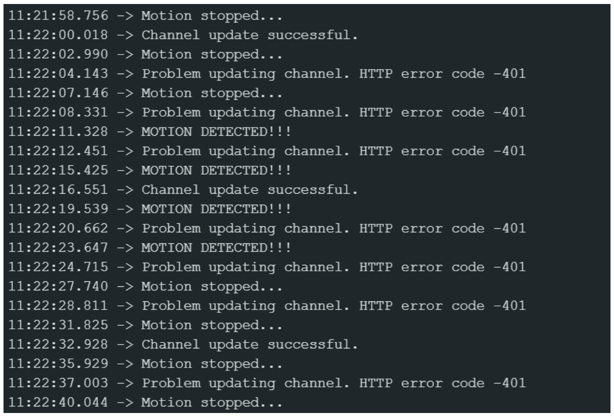
- the channel number;
- the field number

- the value you want to publish (data);
- your write API key.

This function returns the code 200 if it has succeeded in publishing the readings.

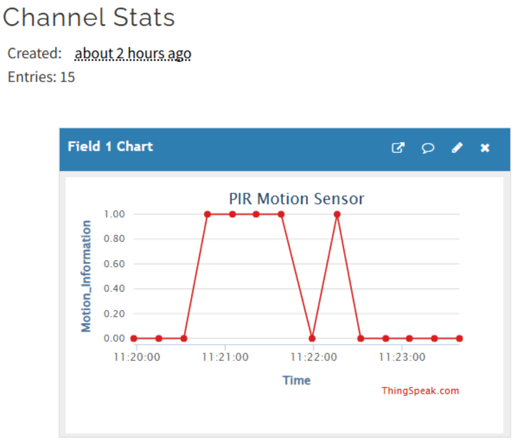
```
int x = ThingSpeak.writeField(myChannelNumber, 1, temperatureC,
myWriteAPIKey);
if(x == 200){
  Serial.println("Channel update successful.");
}
else{
  Serial.println("Problem updating channel. HTTP error code " +
String(x));
}
```

Results (Screenshots serial Monitor):



```
11:21:58.756 -> Motion stopped...
11:22:00.018 -> Channel update successful.
11:22:02.990 -> Motion stopped...
11:22:04.143 -> Problem updating channel. HTTP error code -401
11:22:07.146 -> Motion stopped...
11:22:08.331 -> Problem updating channel. HTTP error code -401
11:22:11.328 -> MOTION DETECTED!!!
11:22:12.451 -> Problem updating channel. HTTP error code -401
11:22:15.425 -> MOTION DETECTED!!!
11:22:16.551 -> Channel update successful.
11:22:19.539 -> MOTION DETECTED!!!
11:22:20.662 -> Problem updating channel. HTTP error code -401
11:22:23.647 -> MOTION DETECTED!!!
11:22:24.715 -> Problem updating channel. HTTP error code -401
11:22:27.740 -> Motion stopped...
11:22:28.811 -> Problem updating channel. HTTP error code -401
11:22:31.825 -> Motion stopped...
11:22:32.928 -> Channel update successful.
11:22:35.929 -> Motion stopped...
11:22:37.003 -> Problem updating channel. HTTP error code -401
11:22:40.044 -> Motion stopped...
```

Graph:



Channel Stats

Created: about 2 hours ago

Entries: 15

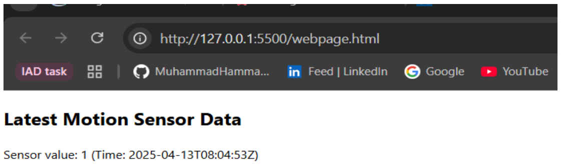
Challenges & Solutions

- Stable internet connection
- Some data did not publish because of error code -401 because the controller (like ESP32) sends data **too frequently**, ThingSpeak will **reject some requests**, which can lead to the error you're seeing to solve this problem we need to send data after some time.
- Sensitivity and delay knob settings as the sensitivity knob sets the range and delay knob sets the time to retain the value detected(we Turn counterclockwise (left) to Shorten delay time.)

Web application:

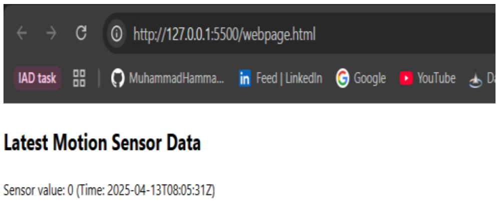
```
<!DOCTYPE html>
<html>
<head>
    <title>ThingSpeak Data Viewer</title>
</head>
<body>
    <h2>Latest Motion Sensor Data</h2>
    <p id="sensorData">Loading...</p>
    <p id="test"></p>

    <script>
    async function getLatestData() {
      const url =
"https://api.thingspeak.com/channels/2916959/fields/1.json?api_key=81OWXYUKN6ZPL0W7&
results=1";
      const response = await fetch(url);

      const data = await response.json();

      document.getElementById("sensorData").innerText = `Sensor value:
${data.feeds[0].field1} (Time: ${data.feeds[0].created_at})`;
    }

    setInterval(getLatestData, 5000);  // Fetch every 5 seconds
    getLatestData();  // Run immediately
    </script>
</body>
</html>
```

Output:



Latest Motion Sensor Data

Sensor value: 1 (Time: 2025-04-13T08:04:53Z)



Latest Motion Sensor Data

Sensor value: 0 (Time: 2025-04-13T08:05:31Z)

Comparison of IoT Platforms

Features	IoT Platforms		
	AWS IoT Core	**Blynk**	**Thingspeak**
Ease of Integration	It is much harder to implement (High complexity)	Easy and simple to use	Easy to implement
Features	Highly secure, supports MQTT, HTTP, integration with other AWS services	Mobile dashboard, device control, OTA updates	Data logging, MATLAB analytics, real-time visualization
Scalability	Highly scalable for millions of devices	Moderate scalability; best for small to medium projects	Limited scalability; suitable for small-scale
Use Cases	Industrial IoT, Smart cities, Connected vehicles	Home automation, DIY projects,	Academic projects, basic prototyping