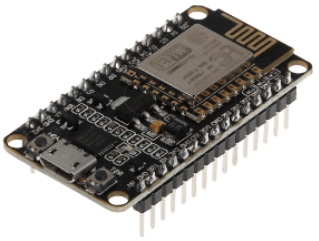
I2E301: Internet of Things (IoT) Programming with Arduino & ESP



*Prior knowledge of **electronics** and **programming** is required for this class. I2E201, Advanced Electronics & Programming with Arduino is a mandatory prerequisite for all grades, unless the student has extensive Arduino experience from somewhere else.*

Requirement: Laptop for programming
Prerequisite: I2E201 Course
Sessions: 8 Classes (90m each)

Topics covered:

- Learn everything about ESP8266 NodeMCU and how to make it work with Arduino IDE
- Use ESP8266 Microcontroller to connect to multiple WiFis, create your own Access Point
- Learn how IoT devices (like Next, Ring) go through the initial setup routine
- Create your first IoT Device that can be controlled remotely
- Create an IoT device that forecasts weather and displays on an LCD Screen
- Working with OLED Displays
- Use IFTTT to automate and integrate with your IoT device

Curriculum & Projects:

- Introduction to ESP8266 NodeMCU
- Understanding ESP8266 Arduino Core
- Creating your own Access Point
- Scan and list all available networks nearby
- Create your own IoT device running as web server
- Connecting LCD to ESP using I2C interface
- Learn to use OLED Displays and Graphics Library
- Fetch data from the internet (APIs) on the IoT device
- Arduino, ESP, IFTTT and Maker WebHooks

Recommended Additional Material (if I2E201 not attended, $120)

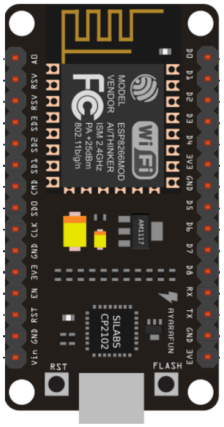
- Arduino Kit: https://amzn.to/2HfEIck
- https://amzn.to/2qIvn6p, https://amzn.to/2J9ZnyH

I2E301: Internet of Things (IoT) Programming with Arduino & ESP

Course Overview

- Introduction to ESP8266 NodeMCU
  - ESP8266 Core for the Arduino IDE Setup
  - Project 1: First Program: Blink
- Understanding ESP8266 Arduino Core
  - Project 2: Connecting ESP8266 to WiFi as a Station
  - Project 3: Connect ESP8266 to Multiple WiFi
  - Project 4: Using setDebugOutput: WiFi Diagnostic
- Creating your own Access Point
  - Project 5: Using ESP8266 as Soft Access Point
  - Project 6: Create a WiFi access point & provide a web server
- Scan all available networks nearby
  - Project 7: Scan WiFi Networks and print them all
  - Project 8: Scan WiFi Networks in Async Mode
- Remotely controllable iOT device
  - Project 9: Create your first iOT Device that can be controlled remotely
  - Project 10: Use 16x2 I2C LCD with ESP8266
- Using IFTTT with Arduino and ESP
  - Project 11: Create a Maker Webhook and test from browser
  - Project 12: Send an Email whenever a motion is detected in the room
- Fetch data from the internet on the iOT device
  - Project 13: Create an iOT device that displays weather on LCD
  - Project 14: Use 128x32 OLED SSD1306 Display with ESP
- References

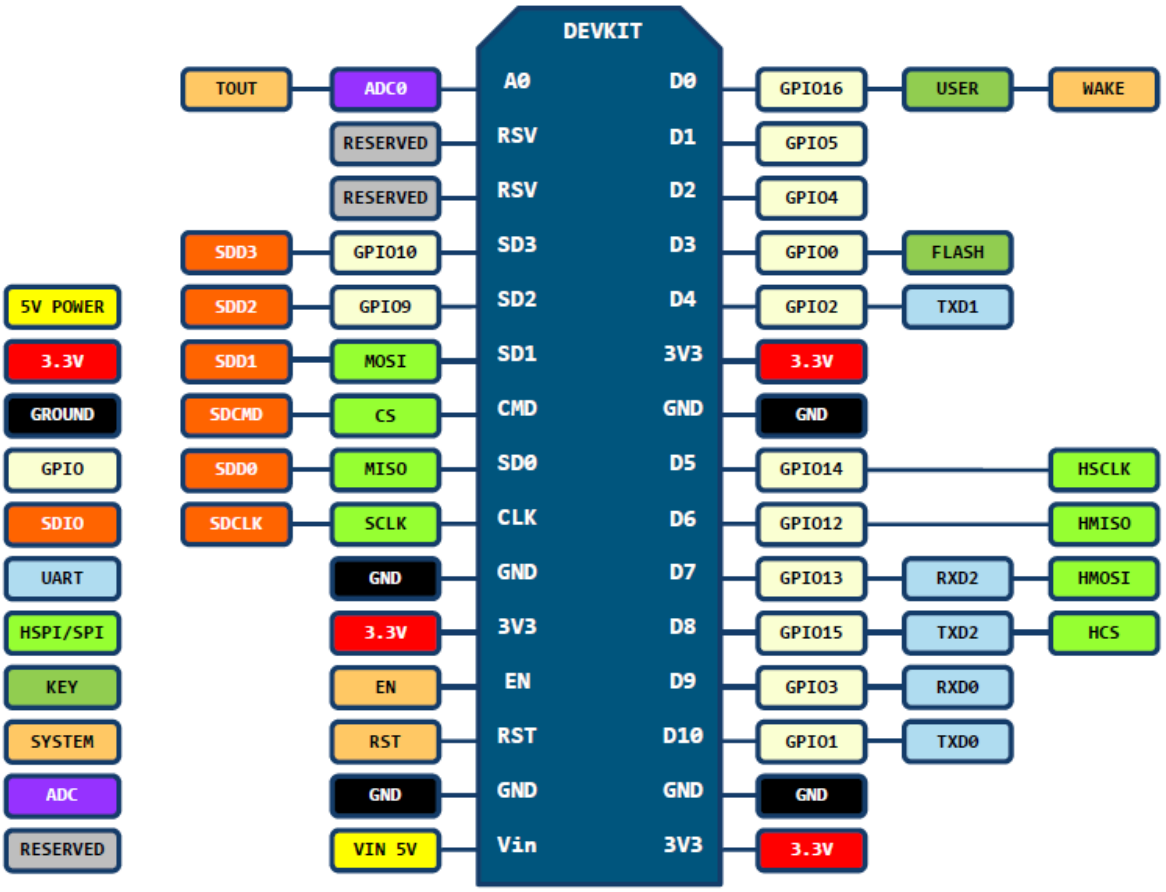
ESP8266 NodeMCU



NodeMCU is an open source **IoT** (Internet of Things) platform. It includes firmware which runs on the **ESP8266 Wi-Fi SoC** from **Espressif Systems**, and hardware which is based on the **ESP-12** module.

The term "NodeMCU" by default refers to the firmware rather than the development kits.

PIN Definition



D0(GPIO16) can only be used as gpio read/write, no interrupt supported, no pwm/i2c/ow supported.

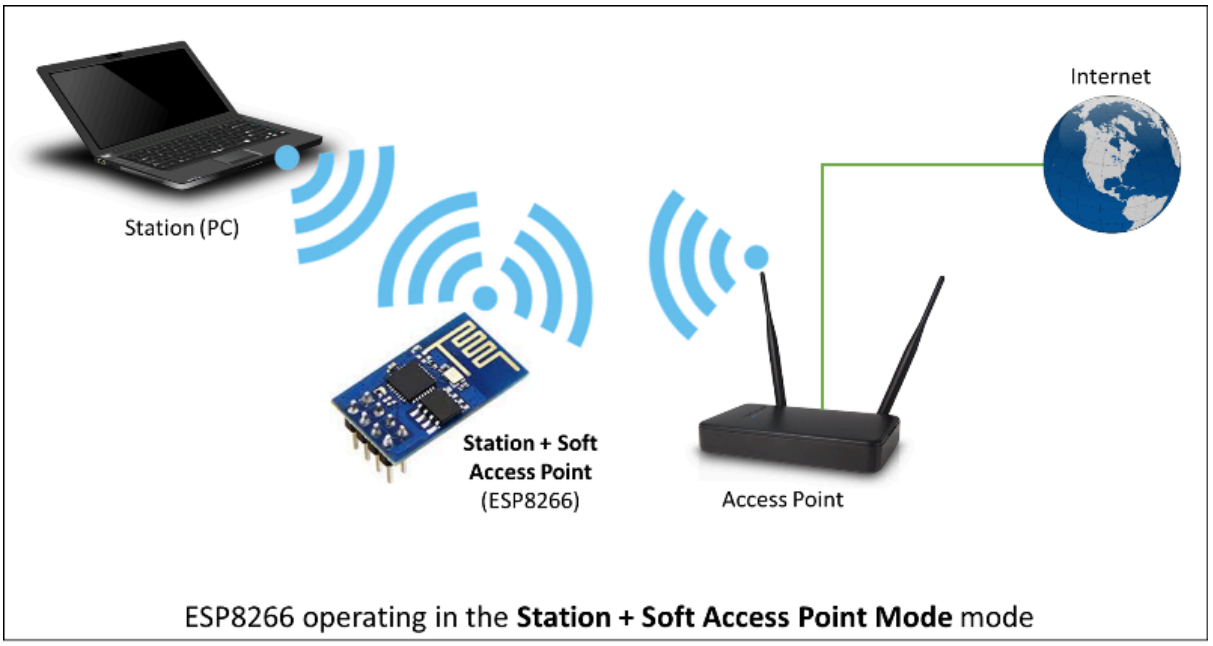
Learn More

- https://nodemcu.readthedocs.io/en/latest/en/modules/gpio/
- https://en.wikipedia.org/wiki/NodeMCU
- https://arduino-esp8266.readthedocs.io/en/2.4.2/

GPIO (General Purpose Input / Output) Pins

IO index	ESP8266 pin	IO index	ESP8266 pin
D0 [*]	GPIO16	D7	GPIO13
D1	GPIO5	D8	GPIO15
D2	GPIO4	9 / RX	GPIO3
D3	GPIO0	10 /TX	GPIO1
D4	GPIO2	11 / SD2	GPIO9
D5	GPIO14	12 / SD3	GPIO10
D6	GPIO12		

Who's Who / Naming



ESP8266 operating in the **Station + Soft Access Point Mode** mode

Devices that connect to Wi-Fi network are called **stations** (STA). Connection to Wi-Fi is provided by an **access point** (AP), that acts as a hub for one or more stations. The access point on the other end is connected to a wired network (a.k.a. **Internet**). An access point is usually integrated with a **router** to provide access from Wi-Fi network to the internet. Each access point is recognized by a **SSID** (Service Set IDentifier), that essentially is the name of network you select when connecting a device (station) to the Wi-Fi network.

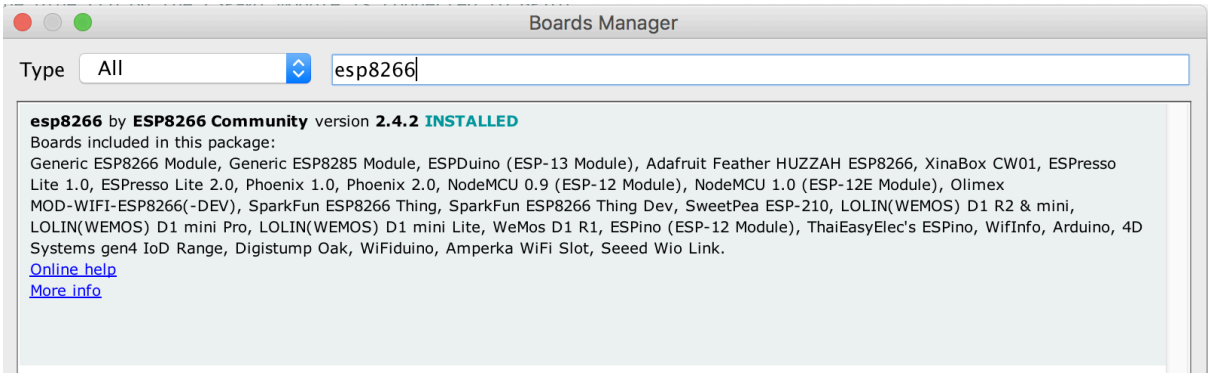
ESP8266 module can operate as a station, so we can connect it to the Wi-Fi network. It can also operate as a **soft access point** (soft-AP), to establish its

own Wi-Fi network. Therefore we can connect other stations to such ESP module. ESP8266 is also able to operate both in station and soft access point mode. This provides possibility of building e.g. mesh networks.

Setup

1. Add http://arduino.esp8266.com/stable/package_esp8266com_index.json to the Additional Boards Manager in Arduino -> Preferences -> Settings

2. Go to Tools -> Boards -> Board Manager and search for ESP8266

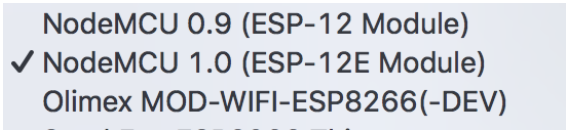


3. Install esp8266 by ESP8266 Community

4. Restart Arduino Program

5. Connect ESP8266 to the USB Cable

6. Go to Tools -> Boards and select NodeMCU 1.0 (ESP 12E Module)



7. Go to Tools -> Port and select appropriate USB Port

8. You are all set to execute your first program. Continue to Project 1.

Project 1: First program: Blink

The purpose of this first program is to validate the overall setup.

Circuit

Connect a blue LED to the pin D7 of the ESP8266. Make sure to connect -ve to ground

Sketch

```
void setup() {
      pinMode(D7, OUTPUT);      // Initialize the D7 pin as an
output
}

// the loop function runs over and over again forever
void loop() {
      digitalWrite(D7, LOW);    // Turn the LED on (Note that LOW is
                                // the voltage level but actually the
                                // LED is on; this is because it is
                                // active low on the ESP-01)
      delay(1000);              // Wait for a second
      digitalWrite(D7, HIGH);   // Turn the LED off by making HIGH
      delay(1000);              // Wait for a second
}
```

Make sure to change Tools -> Upload Speed to **115200** baud rate for faster uploads. Compile the sketch and upload it to the ESP8266

Result

You should see the Blue LED blinking at every second. If the blink sketch is working, you can now proceed to more advanced programs.

Troubleshooting

If there is an error in uploading the sketch, make sure you have selected the right board and the right port from the Tools Menu

Understanding **ESP8266 Arduino Core**

ESP8266 is all about Wi-Fi. To hook up ESP module to Wi-Fi (like hooking up a mobile phone to a hot spot), you need just couple of lines of code:

Project 2: Connect ESP8266 to Wi-Fi

```
#include <ESP8266WiFi.h>
void setup()
{
  Serial.begin(115200);
  Serial.println();

  WiFi.begin("network-name", "pass-to-network");

  Serial.print("Connecting");
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println();

  Serial.print("Connected, IP address: ");
  Serial.println(WiFi.localIP());
}
void loop() { }
```

In the line WiFi.begin("network-name", "pass-to-network") replace **network-name** and **pass-to-network** with name and password to the Wi-Fi network you like to connect. Then upload this sketch to ESP module and open serial monitor to check the connection status and ESP8266 IP Address.

In the first line of sketch `#include <ESP8266WiFi.h>` we are including ESP8266WiFi library. This library provides ESP8266 specific Wi-Fi routines we are calling to connect to network.

Actual connection to Wi-Fi is initialized by calling:

```
WiFi.begin("network-name", "pass-to-network");
```

Connection process can take couple of seconds and we are checking for this to complete in the following loop:

```
while (WiFi.status() != WL_CONNECTED)
{
  delay(500);
  Serial.print(".");
}
```

The while() loop will keep looping while WiFi.status() is other than WL_CONNECTED. The loop will exit only if the status changes to WL_CONNECTED.

The last line will then **print out IP address** assigned to ESP module by DHCP:

```
Serial.println(WiFi.localIP());
```

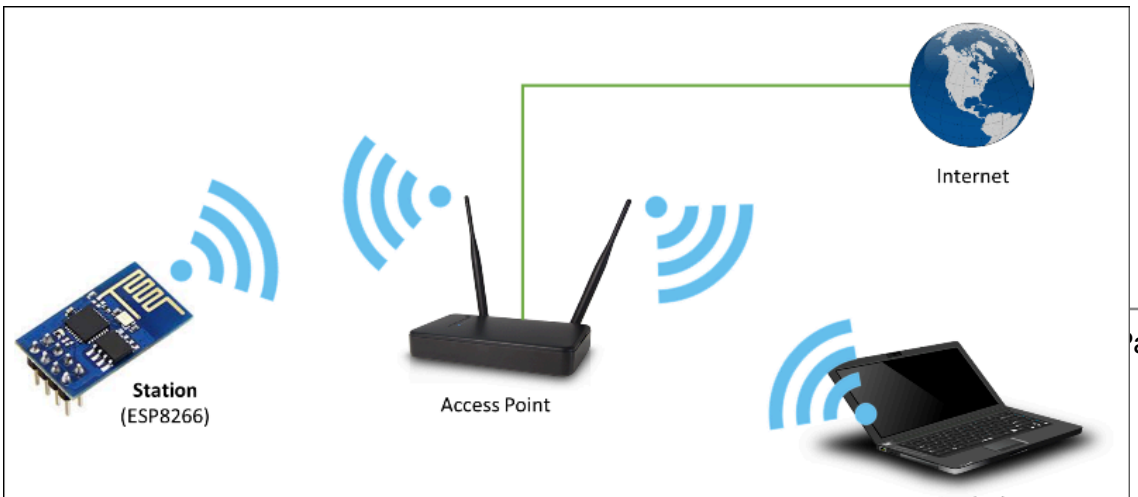
If you don't see the last line but just more and more dots **.....,** then likely name or password to the Wi-Fi network in sketch is entered incorrectly.

Verify name and password by connecting from scratch to this Wi-Fi a PC or a mobile phone.

Note: if connection is established, and then lost for some reason, ESP will automatically reconnect to last used access point once it is back on-line. This will be done automatically by Wi-Fi library, without any user intervention.

Station Mode

Station (STA) mode is used to get ESP module connected to a Wi-Fi network established by an access point.



Station class has several features to facilitate management of Wi-Fi connection. In case the connection is lost, ESP8266 will automatically reconnect to the last used access point, once it is again available. The same happens on module reboot. This is possible since ESP is saving credentials to last used access point in flash (non-volatile) memory. Using the saved data ESP will also reconnect if sketch has been changed but code does not alter the Wi-Fi mode or credentials.

Project 3: Connect ESP8266 to Multiple WiFi

In this project, we will make ESP connect to next available access point if current connection is lost.

Sketch

```
#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>

ESP8266WiFiMulti wifiMulti;
boolean connectioWasAlive = true;

void setup()
{
  Serial.begin(115200);
  Serial.println();

  wifiMulti.addAP("primary-network-name", "pass-to-primary-network");
  wifiMulti.addAP("secondary-network-name", "pass-to-secondary-network");
  wifiMulti.addAP("tertiary-network-name", "pass-to-tertiary-network");
}

void monitorWiFi()
{
  if (wifiMulti.run() != WL_CONNECTED)
  {
    if (connectioWasAlive == true)
    {
      connectioWasAlive = false;
      Serial.print("Looking for WiFi ");
    }
    Serial.print(".");
    delay(500);
  }
  else if (connectioWasAlive == false)
  {
    connectioWasAlive = true;
    Serial.printf(" connected to %s\n", WiFi.SSID().c_str());
  }
}

void loop()
{
  monitorWiFi();
```

```
}
```

Prepare Access Points

To try this sketch in action you need two (or more) access points. In lines below replace **primary-network-name** and **pass-to-primary-network** with name and password to your primary network. Do the same for secondary network.

```
wifiMulti.addAP("primary-network-name", "pass-to-primary-network");
wifiMulti.addAP("secondary-network-name", "pass-to-secondary-network");
```

You may add more networks if you have more access points.

```
wifiMulti.addAP("tertiary-network-name", "pass-to-tertiary-network");
...
```

Try it Out

Now upload updated sketch to ESP module and open serial monitor. Module will first scan for available networks. Then it will select and connect to the network with stronger signal. In case connection is lost, module will connect to next one available.

This process may look something like:

```
Looking for WiFi ..... connected to sensor-net-1
Looking for WiFi ....... connected to sensor-net-2
Looking for WiFi .... connected to sensor-net-1
```

In above example ESP connected first to **sensor-net-1**. Switch **sensor-net-1** off, ESP will discover that connection is lost and start searching for another configured network. If sensor-net-2 is available, ESP will connect to it.

Function **monitorWiFi**() is in place to show when connection is lost by displaying Looking for WiFi. Dots **....** are displayed during process of searching for another configured access point. Then a message like connected to **sensor-net-2** is shown when connection is established.

Project 4: Using setDebugOutput: WiFi Diagnostic

Please note that you may simplify above sketch by removing function **monitorWiFi**() and putting inside loop() only **wifiMulti.run()**. ESP will still reconnect between configured access points, if required. Now you won't be able to see it on serial monitor unless you add **Serial.setDebugOutput(true)** to enable Wi-Fi Diagnostic.

Sketch

```
#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>

ESP8266WiFiMulti wifiMulti;

void setup()
{
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();

  wifiMulti.addAP("primary-network-name", "pass-to-primary-network");
  wifiMulti.addAP("secondary-network-name", "pass-to-secondary-network");
  wifiMulti.addAP("tertiary-network-name", "pass-to-tertiary-network");
}

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

That's it! This is really all the code you need to make ESP automatically reconnecting between available networks.

Try it Out

After uploading sketch and opening the serial monitor, the messages will look as below.

Initial connection to sensor-net-1 on power up:

```
f r0, scandone
f r0, scandone
state: 0 -> 2 (b0)
state: 2 -> 3 (0)
state: 3 -> 5 (10)

add 0
aid 1
cnt
```

```
chg_B1:-40

connected with sensor-net-1, channel 1
dhcp client start...
ip:192.168.1.10,mask:255.255.255.0,gw:192.168.1.9
```

Lost connection to sensor-net-1 and establishing connection to sensor-net-2:

```
bcn_timout,ap_probe_send_start
ap_probe_send over, rest wifi status to disassoc
state: 5 -> 0 (1)
rm 0
f r-40, scandone
f r-40, scandone
f r-40, scandone
state: 0 -> 2 (b0)
state: 2 -> 3 (0)
state: 3 -> 5 (10)
add 0

aid 1
cnt

connected with sensor-net-2, channel 11
dhcp client start...
ip:192.168.1.102,mask:255.255.255.0,gw:192.168.1.234
```

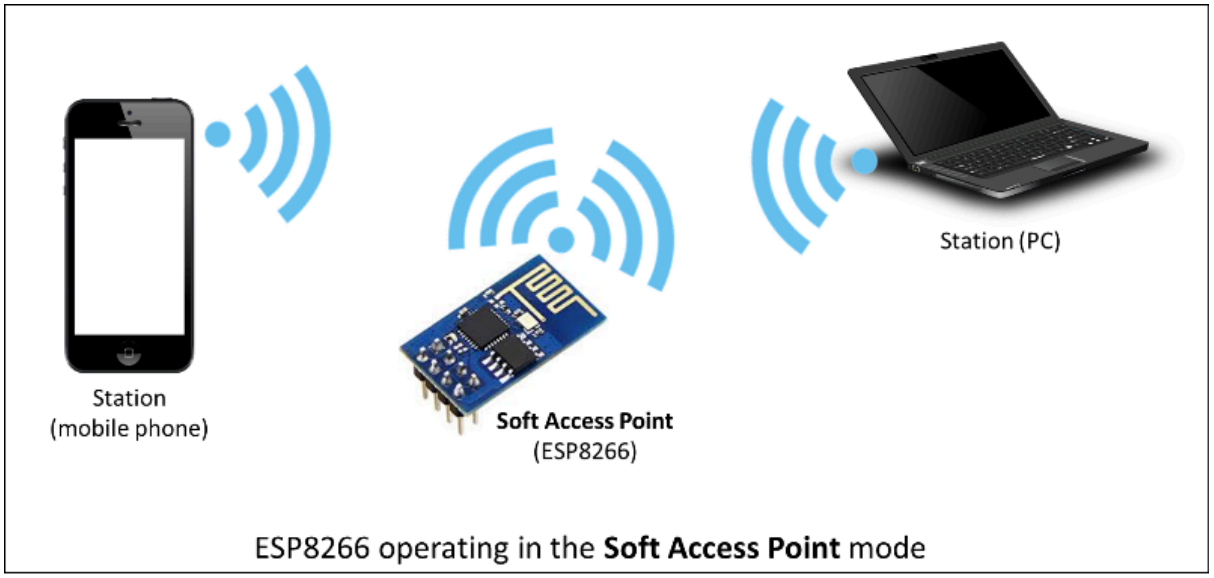
Conclusion

The minimalist sketch with **ESP8266WiFiMulti** class is a great example of what ESP8266 can do behind the scenes with just couple lines of code.

As shown in above example, reconnecting between access points takes time and is not seamless. Therefore, in practical applications, you will likely need to monitor connection status to decide e.g. if you can send the data to external system or should wait until connection is back.

Soft Access Point

An **access point** (AP) is a device that provides access to Wi-Fi network to other devices (stations) and connects them further to a wired network. ESP8266 can provide similar functionality except it does not have interface to a wired network. Such mode of operation is called **soft access point** (soft-AP). The maximum number of stations connected to the soft-AP is five.



ESP8266 operating in the **Soft Access Point** mode

The soft-AP mode is often used as an intermediate step before connecting ESP to a Wi-Fi in a station mode. This is when SSID and password to such network is not known upfront. ESP first boots in soft-AP mode, so we can connect to it using a laptop or a mobile phone. Then we are able to provide credentials to the target network. Once done ESP is switched to the station mode and can connect to the target Wi-Fi. **This is how most of the iOT devices (Nest, Ring, etc) setup work.**

Another handy application of soft-AP mode is to set up **mesh networks**. ESP can operate in both soft-AP and Station mode so it can act as a node of a mesh network.

Project 5: Using ESP8266 as Soft Access Point

Setting up soft-AP with ESP8266 can be done with just couple lines of code.

Sketch

```
#include <ESP8266WiFi.h>

void setup()
{
  Serial.begin(115200);
  Serial.println();

  Serial.print("Setting soft-AP ... ");
  boolean result = WiFi.softAP("ESPsoftAP_01", "pass-to-soft-AP");
  if(result == true)
  {
    Serial.println("Ready");
  }
  else
  {
    Serial.println("Failed!");
  }
}

void loop()
{
  Serial.printf("Stations connected = %d\n", WiFi.softAPgetStationNum());
  delay(3000);
}
```

Try it Out

In line boolean result = WiFi.softAP("ESPsoftAP_01", "pass-to-soft-AP") change **pass-to-soft-AP** to some meaningful password and upload sketch. Open serial monitor and you should see:

```
Setting soft-AP ... Ready
Stations connected = 0
Stations connected = 0
…
```

Then take your mobile phone or a PC, open the list of available access points, find ESPsoftAP_01and connect to it. This should be reflected on serial monitor as a new station connected:

```
Stations connected = 1
Stations connected = 1
...
```

If you have another Wi-Fi station available then connect it as well. Check serial monitor again where you should now see two stations reported.

Can we Make it Simpler?

Can we make this sketch even simpler? Yes, we can! We can do it by using alternate **if - else** statement as below:

```
WiFi.softAP("ESPsoftAP_01", "pass-to-soft-AP") ? "Ready" : "Failed!"
```

Such statement will return either **Ready** or **Failed**! depending on result of WiFi.softAP(...). This way we can considerably shorten our sketch without any changes to functionality:

```
#include <ESP8266WiFi.h>

void setup()
{
  Serial.begin(115200);
  Serial.println();

  Serial.print("Setting soft-AP ... ");
  Serial.println(WiFi.softAP("ESPsoftAP_01", "pass-to-soft-AP") ? "Ready" :
"Failed!");
}

void loop()
{
  Serial.printf("Stations connected = %d\n", WiFi.softAPgetStationNum());
  delay(3000);
}
```

This is very neat piece of code. If **? : conditional** operator is new to you, I recommend to start using it and make your code shorter & more elegant.

Conclusion

ESP8266WiFi library makes it easy to turn ESP8266 into soft access point.

Once you try above sketch check out next Project as a next step. It demonstrates how to access ESP6266 operating in soft-AP mode from a web browser.

Project 6: Create a WiFi access point & provide a web server

Setting up soft-AP with ESP8266 can be done with just couple lines of code.

Sketch

```
/* Create a WiFi access point and provide a web server on it. */

#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

/* Set these to your desired credentials. */
const char *ssid = "ESPsoftAP_01";
const char *password = "12345678";

ESP8266WebServer server(80);

/* Just a little test message.  Go to http://192.168.4.1 in a web browser
   connected to this access point to see it.
*/
void handleRoot() {
  server.send(200, "text/html", "<h1>Welcome to the I2E301 Class!</h1>");
}

void setup() {
  delay(1000);
  Serial.begin(115200);
  Serial.println();
  Serial.print("Configuring access point...");
  /* You can remove the password parameter if you want the AP to be open.*/
  WiFi.softAP(ssid, password);

  IPAddress myIP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(myIP);
  server.on("/", handleRoot);
  server.begin();
  Serial.println("HTTP server started");
}

void loop() {
  server.handleClient();
}
```

Try it Out

Connect your device to the **ESPsoftAP_01** Wi-Fi Access Point and point your web browser to http://192.168.4.1 . Using this server, you can serve any information back to the browser.

Scan

To connect a mobile phone to a hot spot, you typically open Wi-Fi settings app, list available networks and pick the hot spot you need. Then enter a password (or not) and you are in. You can do the same with ESP. Functionality of scanning for, and listing of available networks in range is implemented by the Scan Class. This example shows the bare minimum code we need to check for the list of available networks.

Disconnect

To start with, enable module in station mode and then disconnect.

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

Running **WiFi.disconnect()** is to shut down a connection to an access point that module may have automatically made using previously saved credentials.

Scan for Networks

After some delay to let the module disconnect, go to scanning for available networks:

```
int n = WiFi.scanNetworks();
```

Now just check if returned n if greater than 0 and list found networks:

```
for (int i = 0; i < n; i++)
{
  Serial.println(WiFi.SSID(i));
}
```

This is that simple.

Project 7: Scan WiFi Networks and print them all

Sketch

```
#include "ESP8266WiFi.h"

void setup()
{
  Serial.begin(115200);
  Serial.println();

  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);
}

void loop()
{
  Serial.print("Scan start ... ");
  int n = WiFi.scanNetworks();
  Serial.print(n);
  Serial.println(" network(s) found");
  for (int i = 0; i < n; i++)
  {
    Serial.println(WiFi.SSID(i));
  }
  Serial.println();

  delay(5000);
}
```

Try it Out

Upload this sketch to ESP module and open a serial monitor. If there are access points around (sure there are) you will see a similar list repeatedly printed out:

```
Scan start ... 5 network(s) found
Tech_D005107
HP-Print-A2-Photosmart 7520
ESP_0B09E3
Hack-4-fun-net
UPC Wi-Free
```

When looking for the text **scan start ...** displayed, you will notice that it takes noticeable time for the following text **n network(s) found** to show up. This is because execution of **WiFi.scanNetworks()** takes time and our program is waiting for it to complete before moving to the next line of code. What if at the same time we would like ESP to run time critical process (e.g. animation) that should not be disturbed?

It turns out that this is fairly easy to do by scanning networks in **async mode.** That's will be our next Project

Project 8: Scan WiFi Networks in Async Mode

In this project, we will also print out other parameters of available networks besides SSID.

Async Scan

What we like to do, is to trigger process of scanning for networks and then return to executing code inside the **loop**(). Once scanning is complete, at a convenient time, we will check the list of networks. The "time critical process" will be simulated by a blinking LED at 250ms period.

We like the blinking pattern not be disturbed at any time.

No delay() function

To implement such functionality we refrain from using any **delay**() inside the **loop**(). Instead we define period when to trigger particular action. Then inside **loop**() we check **millis**() (internal clock that counts milliseconds) and fire the action if the period expires.

Setup

First we define scanning period and internal variable **lastScanMillis** that holds time when the last scan was made.

```
#define SCAN_PERIOD 5000
long lastScanMillis;
```

Then inside the **loop**() we check if **SCAN_PERIOD** expired, so it is time to fire next scan:

```
if (currentMillis - lastScanMillis > SCAN_PERIOD)
{
  WiFi.scanNetworks(true);
  Serial.print("\nScan start ... ");
  lastScanMillis = currentMillis;
}
```

Please note that **WiFi.scanNetworks(true)** has an extra parameter true that was not present in previous example above. This is an instruction to scan in **asynchronous mode**, i.e. trigger scanning process, do not wait for result (*processing will be done in background*) and move to the next line of

code. We need to use asynchronous mode otherwise the 250ms LED blinking pattern would be disturbed as scanning takes longer than 250ms.

Check When Done

Finally we periodically check for scan completion to print out the result once ready. To do so, we use the function **WiFi.scanComplete()**, that upon completion returns the number of found networks. If scanning is still in progress it returns **-1**. If scanning has not been triggered yet, it would return **-2**.

```
int n = WiFi.scanComplete();
if(n >= 0)
{
  Serial.printf("%d network(s) found\n", n);
  for (int i = 0; i < n; i++)
  {
    Serial.printf("%d: %s, Ch:%d (%ddBm) %s\n", i+1, WiFi.SSID(i).c_str(),
WiFi.channel(i), WiFi.RSSI(i), WiFi.encryptionType(i) == ENC_TYPE_NONE ?
"open" : "");
  }
  WiFi.scanDelete();
}
```

Please note the function **WiFi.scanDelete()** is deleting scanning results from memory, so it is not printed out over and over again on each **loop()** run.

Complete sketch is below. The code inside **setup**() is the same as described in the previous example except for an additional **pinMode**() to configure the output pin for LED.

Sketch

```
#include "ESP8266WiFi.h"

#define BLINK_PERIOD 250
long lastBlinkMillis;
boolean ledState;

#define SCAN_PERIOD 5000
long lastScanMillis;


void setup()
 {
  Serial.begin(115200);
  Serial.println();

  pinMode(LED_BUILTIN, OUTPUT);

  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);
```

```
}

void loop()
{
  long currentMillis = millis();

  // blink LED
  if (currentMillis - lastBlinkMillis > BLINK_PERIOD)
  {
    digitalWrite(LED_BUILTIN, ledState);
    ledState = !ledState;
    lastBlinkMillis = currentMillis;
  }

  // trigger Wi-Fi network scan
  if (currentMillis - lastScanMillis > SCAN_PERIOD)
  {
    WiFi.scanNetworks(true);
    Serial.print("\nScan start ... ");
    lastScanMillis = currentMillis;
  }

  // print out Wi-Fi network scan result uppon completion
  int n = WiFi.scanComplete();
  if(n >= 0)
  {
    Serial.printf("%d network(s) found\n", n);
    for (int i = 0; i < n; i++)
    {
      Serial.printf("%d: %s, Ch:%d (%ddBm) %s\n", i+1, WiFi.SSID(i).c_str(),
WiFi.channel(i), WiFi.RSSI(i), WiFi.encryptionType(i) == ENC_TYPE_NONE ?
"open" : "");
    }
    WiFi.scanDelete();
  }
}
```

Try it Out

Upload above sketch to ESP module and open a serial monitor. You should see similar list printed out every 5 seconds:

```
Scan start ... 5 network(s) found
1: Tech_D005107, Ch:6 (-72dBm)
2: HP-Print-A2-Photosmart 7520, Ch:6 (-79dBm)
3: ESP_0B09E3, Ch:9 (-89dBm) open
4: Hack-4-fun-net, Ch:9 (-91dBm)
5: UPC Wi-Free, Ch:11 (-79dBm)
```

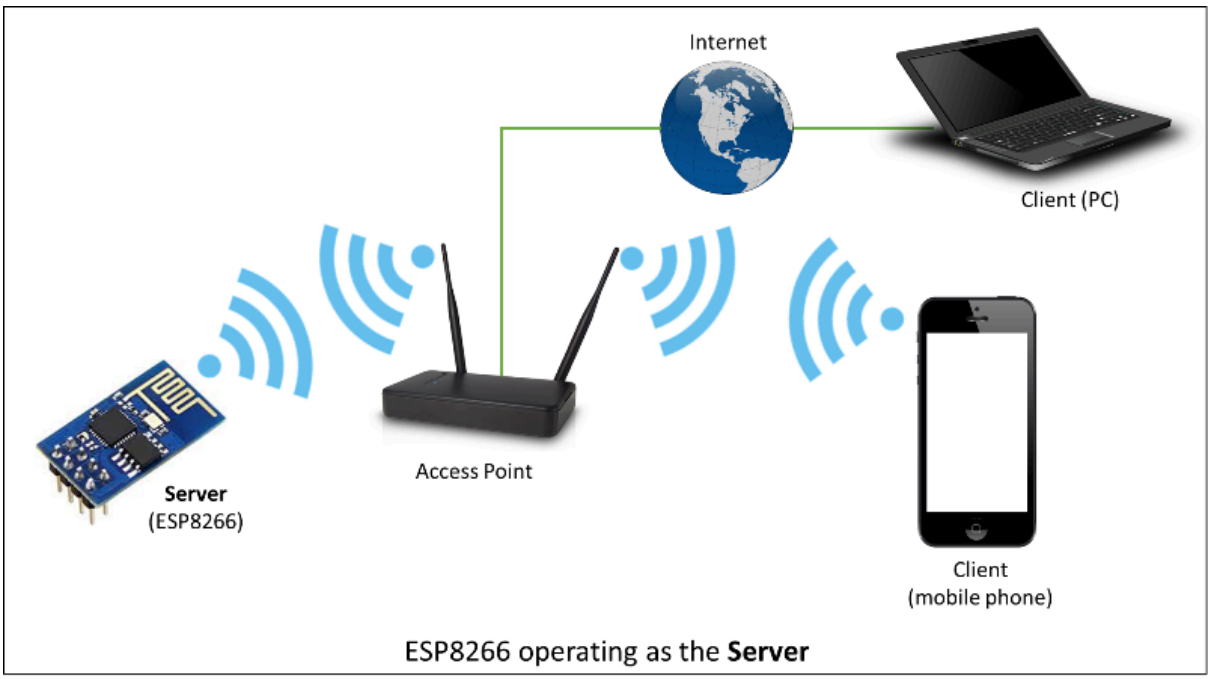
Check the LED. It should be blinking undisturbed four times per second.

Conclusion

The scan class API provides comprehensive set of methods to do scanning in both synchronous as well as in asynchronous mode. Therefore we can easy implement code that is doing scanning in background without disturbing other processes running on ESP8266 module.

Project 9: Create your first iOT Device using ESP8266

In this project, we will learn how to create a web server using ESP8266 NodeMCU and control your circuit remotely using a browser.

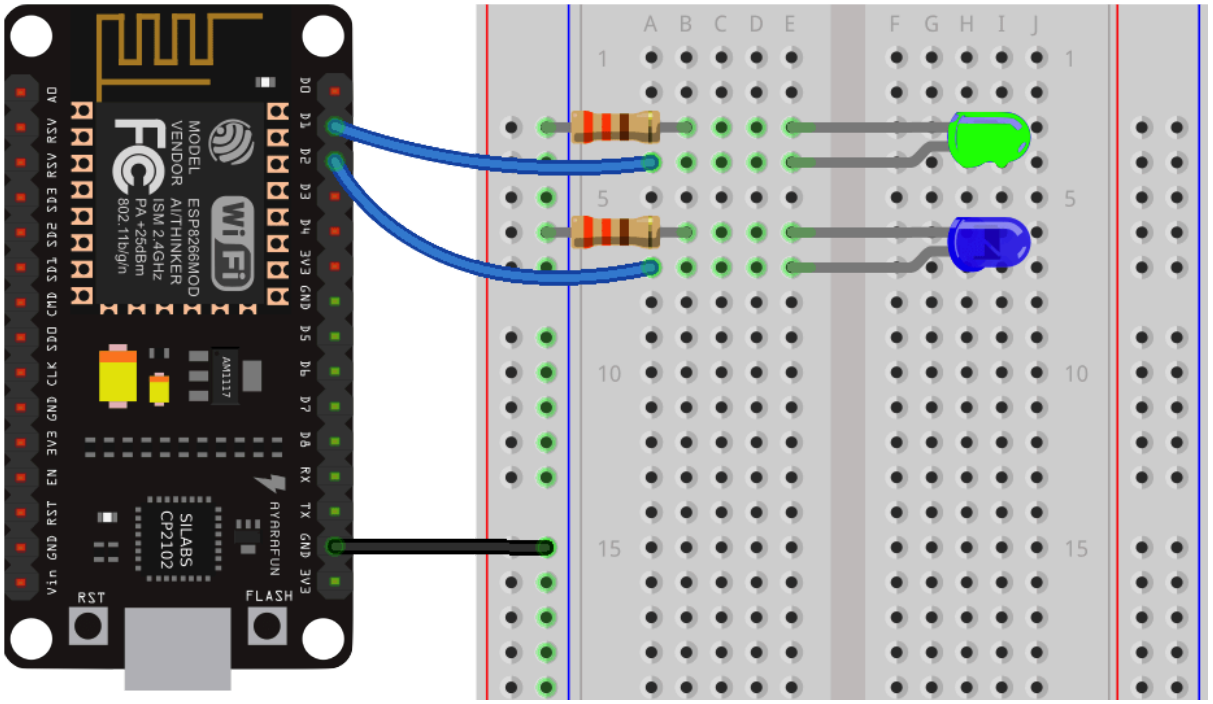


ESP8266 operating as the **Server**

To keep it simple, we will connect two different color LEDs to two different GPIO pins and control them over Wi-Fi.

Circuit

Connect Green LED to GPIO5 and Blue LED to GPIO4



Sketch

Note: Get the code from Tinkercad shared project

```
// Load Wi-Fi library
#include <ESP8266WiFi.h>

// Replace with your network credentials
const char* ssid     = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

// Set web server port number to 80
WiFiServer server(80);

// Variable to store the HTTP request
String header;

// Auxiliary variables to store the current output state
String gpio5State = "off";
String gpio4State = "off";

// Assign output variables to GPIO pins
const int gpio5 = 5;
const int gpio4 = 4;

void setup() {
  Serial.begin(115200);
  // Initialize the output variables as outputs
  pinMode(gpio5, OUTPUT);
```

```
  pinMode(gpio4, OUTPUT);
  // Set outputs to LOW
  digitalWrite(gpio5, LOW);
  digitalWrite(gpio4, LOW);

  // Connect to Wi-Fi network with SSID and password
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  // Print local IP address and start web server
  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  server.begin();
}
```

Setting up and starting the server on ESP8266

```
// Connect to Wi-Fi network with SSID and password
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
```

```
void loop(){
  WiFiClient client = server.available();   // Listen for incoming clients

  if (client) {                     // If a new client connects,
    Serial.println("New Client.");// print a message out in the serial port
    String currentLine = "";        // make a String to hold incoming data
    while (client.connected()) {  // loop while the client's connected
      if (client.available()) {   // if there's bytes to read from the
        char c = client.read();   // client, read a byte, then
        Serial.write(c);          // print it out the serial monitor
        header += c;
        if (c == '\n') {          // if the byte is a newline character


// if the current line is blank, you got 2 newline chars in a row
// that's the end of the client HTTP request, so send a response:
if (currentLine.length() == 0) {

// HTTP headers always start with a response code  (e.g. HTTP/1.1 200 OK)
// and a content-type so the client knows what's coming, then a blank line:
```

```
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();

// turns the GPIOs on and off
            if (header.indexOf("GET /5/on") >= 0) {
                Serial.println("GPIO 5 on");
                gpio5State = "on";
                digitalWrite(gpio5, HIGH);
            } else if (header.indexOf("GET /5/off") >= 0) {
                Serial.println("GPIO 5 off");
                gpio5State = "off";
                digitalWrite(gpio5, LOW);
            } else if (header.indexOf("GET /4/on") >= 0) {
                Serial.println("GPIO 4 on");
                gpio4State = "on";
                digitalWrite(gpio4, HIGH);
            } else if (header.indexOf("GET /4/off") >= 0) {
                Serial.println("GPIO 4 off");
                gpio4State = "off";
                digitalWrite(gpio4, LOW);
            }

// Display the HTML web page
client.println("<!DOCTYPE html><html>");
client.println("<head><meta name=\"viewport\" content=\"width=device-width,
initial-scale=1\">");
client.println("<link rel=\"icon\" href=\"data:,\">");

// CSS to style the on/off buttons
// Feel free to change the background-color & font-size
client.println("<style>html { font-family: Helvetica; display: inline-block;
margin: 0px auto; text-align: center;}");
client.println(".button { background-color: #195B6A; border: none; color:
white; padding: 16px 40px;");
client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor:
pointer;}");
client.println(".button2 {background-color: #77878A;}</style></head>");

// Web Page Heading
client.println("<body><h1>My First iOT Device: Server</h1>");

// Display current state, and ON/OFF buttons for GPIO 5
client.println("<p>GPIO 5: GREEN LED - State " + gpio5State + "</p>");

// If the gpio5State is off, it displays the ON button
if (gpio5State=="off") {
   client.println("<p><a href=\"/5/on\"><button class=\"button\">
ON</button></a></p>");
} else {
```

```
   client.println("<p><a href=\"/5/off\"><button class=\"button button2\">
OFF</button></a></p>");
}
// Display current state, and ON/OFF buttons for GPIO 4
client.println("<p>GPIO 4: BLUE LED - State " + gpio4State + "</p>");

// If the gpio4State is off, it displays the ON button
if (gpio4State=="off") {
  client.println("<p><a href=\"/4/on\"><button class=\"button\">
ON</button></a></p>");
} else {
  client.println("<p><a href=\"/4/off\"><button class=\"button button2\">
OFF</button></a></p>");
}
client.println("</body></html>");

// The HTTP response ends with another blank line
client.println();
Break;                    // Break out of the while loop
} else {                  // if you got a newline, then clear currentLine
  currentLine = "";
  }
} else if (c != '\r') { // if you got anything else but a carriage
                        // return character,
   currentLine += c;    // add it to the end of the currentLine
   }
  }
 }
 header = "";             // Clear the header variable
 client.stop();           // Close the connection
 Serial.println("Client disconnected.");
 } //if (client)
} //loop()
```

Make sure to make following changes before uploading the sketch

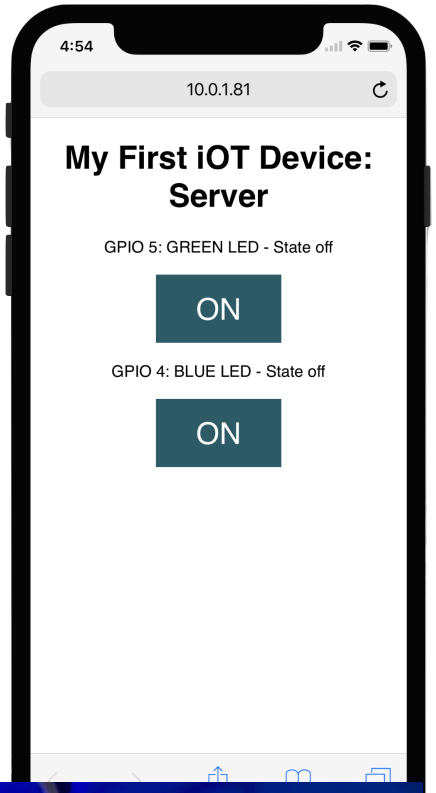
- Change **ssid** and **password** in the sketch
- Change Tools -> Upload Speed to **115200** baud rate for faster uploads.
- Make sure Serial Monitor is open set to 115200 baud
- Compile the sketch and upload it to the ESP8266

Run the Sketch

1. Check Serial monitor and note down your ESP8266 IP Address


```
Connecting to CloudMatrix
...
WiFi connected.
IP address:
10.0.1.84
```

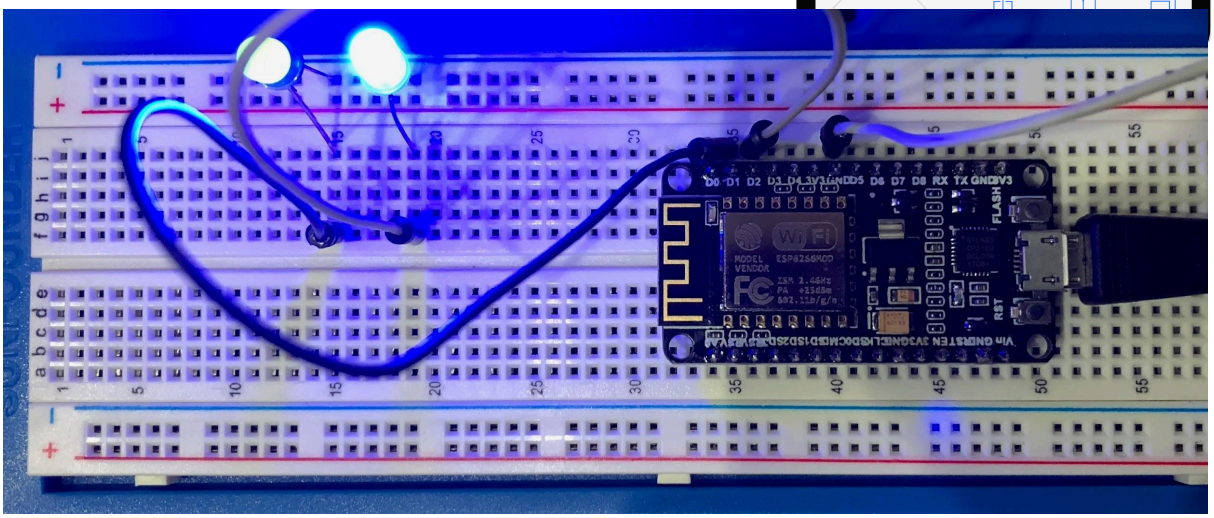
2. Go to a browser on phone, tablet or computer and type http://<ESP8266_IP_ADDDRESS>, for example: http://10.0.1.84, you should see a page like this:



3. Using the ON / OFF buttons in the UI, you should be able to control the LEDs connected to GPIO5 and GPIO4 of ESP8266

4. Check the Serial Monitor for logs and troubleshooting

5. Now you can control your first iOT device with any device that as a browser in your local network.



You can use this project as the basis for any other iOT device. As an example, you can use Ultrasonic sensor along with ESP8266 NodeMCU to detect garage door movement and report (on the web-page) if the garage door is open of close.

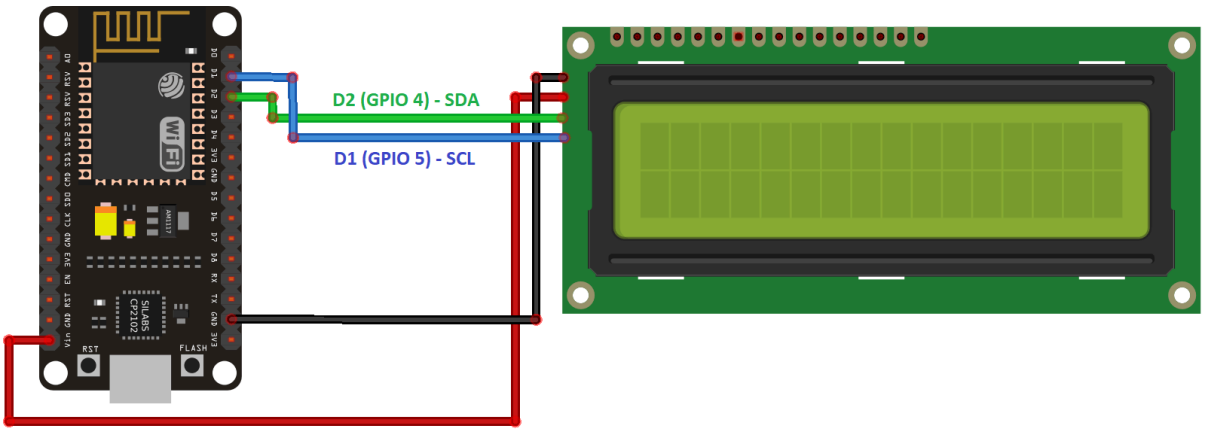
In the next course (I2E401), we will learn how you can control your own iOT devices from Amazon Alexa or your own mobile app.

Project 10: Use 16x2 I2C LCD with ESP8266

For this project we'll be using a 16×2 I2C LCD display, but LCDs with other sizes should also work.

The advantage of using an I2C LCD is that the wiring is really simple. You just need to wire the SDA and SCL pins. Additionally, it comes with a built-in **potentiometer** you can use to adjust the contrast between the background and the characters on the LCD. On a "regular" LCD you need to add a potentiometer to the circuit to adjust the contrast. *(Remember Project 7 from I2E201 Class)*

Circuit



PIN Connections

I2C LCD	ESP8266
GND	GND
VCC	VIN
SDA	D2 (GPIO 4)
SCL	D1 (GPIO 5)

Install the LiquidCrystal_I2C Library

For this project you need to install the LiquidCrystal_I2C library in your Arduino IDE that allows you to work with LCD I2C interface.

1. Download the LiquidCrystal_I2C library from below: https://github.com/marcoschwartz/LiquidCrystal_I2C/archive/master.zip
2. You should have a .zip folder in your Downloads folder.
3. Unzip the .zip folder and you should get LiquidCrystal_I2C-master folder
4. Rename your folder from LiquidCrystal_I2C-master to LiquidCrystal_I2C
5. Move the LiquidCrystal_I2C folder to your Arduino IDE installation libraries folder
6. Reopen your Arduino IDE

Get the LCD I2C Address

I2C is a two wire protocol that allows multiple devices to be used, with only two pins used on the microcontroller. This is accomplished by setting an address on each device on the bus. Not all I2C LCD's use the same address. With the LCD properly wired to the ESP, upload the I2C Scanner sketch

```
#include <Wire.h>
void setup() {
  Wire.begin();
  Serial.begin(115200);
  Serial.println("\nI2C Scanner");
}
void loop() {
  byte error, address;
  int nDevices;
  Serial.println("Scanning...");
  nDevices = 0;
  for(address = 1; address < 127; address++ ) {
    Wire.beginTransmission(address);
    error = Wire.endTransmission();
    if (error == 0) {
      Serial.print("I2C device found at address 0x");
      if (address<16) {
        Serial.print("0");
      }
      Serial.println(address,HEX);
      nDevices++;
    }
    else if (error==4) {
      Serial.print("Unknow error at address 0x");
      if (address<16) {
        Serial.print("0");
      }
      Serial.println(address,HEX);
    }
  }
  if (nDevices == 0) {
    Serial.println("No I2C devices found\n");
  }
  else {
```

```
    Serial.println("done\n");
  }
  delay(5000);
}
```

After uploading the code, open the Serial Monitor at a baud rate of 115200. The I2C address should be displayed in the Serial Monitor.

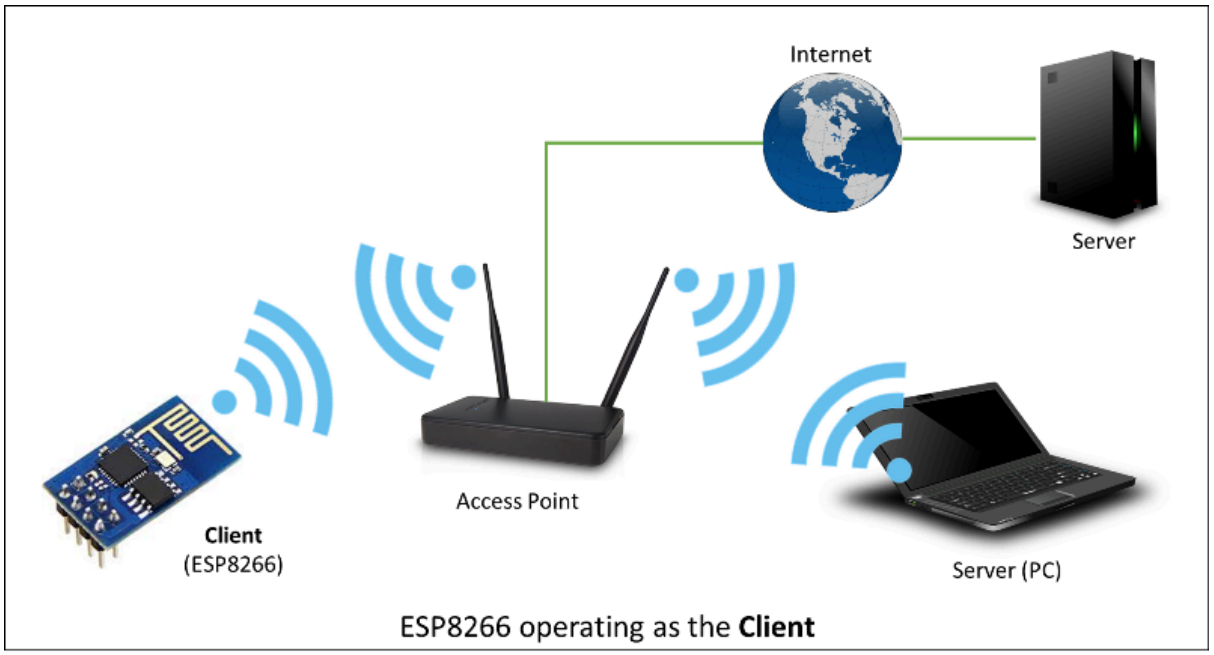
Displaying Text on the LCD

Sketch

```
#include <LiquidCrystal_I2C.h>

// set the LCD number of columns and rows
int lcdColumns = 16;
int lcdRows = 2;

// set LCD address, number of columns and rows
// if you don't know your display address, run an I2C scanner sketch
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);

void setup(){
  // initialize LCD
  lcd.init();
  // turn on LCD backlight
  lcd.backlight();
}

void loop(){
  // set cursor to first column, first row
  lcd.setCursor(0, 0);
  // print message
  lcd.print("Hello, World!");
  delay(1000);
  // clears the display to print new message
  lcd.clear();
  // set cursor to first column, second row
  lcd.setCursor(0,1);
  lcd.print("Hello, World!");
  delay(1000);
  lcd.clear();
}
```

Client

The Client class creates clients that can access services provided by servers in order to send, receive and process data.



ESP8266 operating as the **Client**

This is typical operation performed by a client to access server's API to retrieve specific information. For instance we may want to contact weather APIs to periodically check for weather forecast, etc.

Project 11: Create an IFTTT Web Service

In this project, we will learn how to use NodeMCU to send request to the popular IFTTT (If This Then This) Web Service. When the service receives the request (trigger) it performs an action, which will be to send an Email.

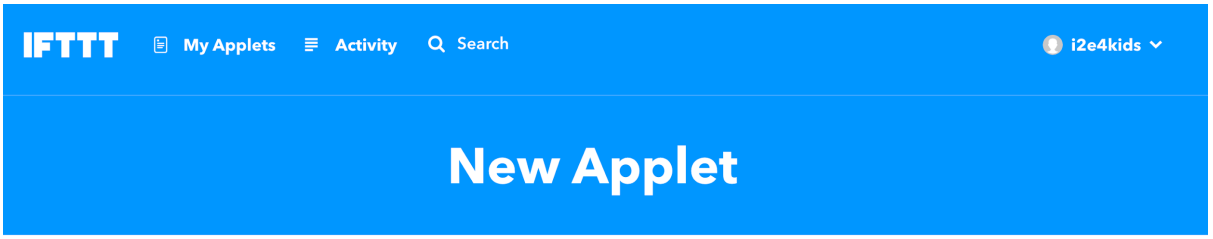
Register on IFTTT

Step 1: Go to IFTTT.com and click on Sign-up

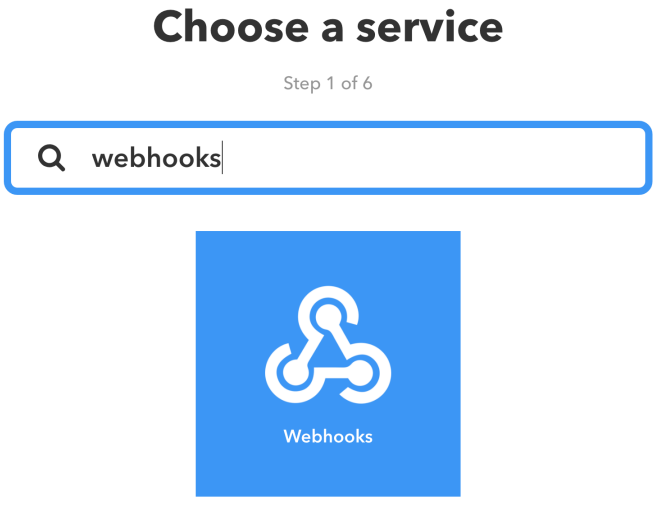
Step 2: You can use your Google Account to sign-in or you can sign-up directly on IFTTT.com

Step 3: Once logged in, Click on My Applets -> New Applet

Step 4: Click on + this to setup trigger

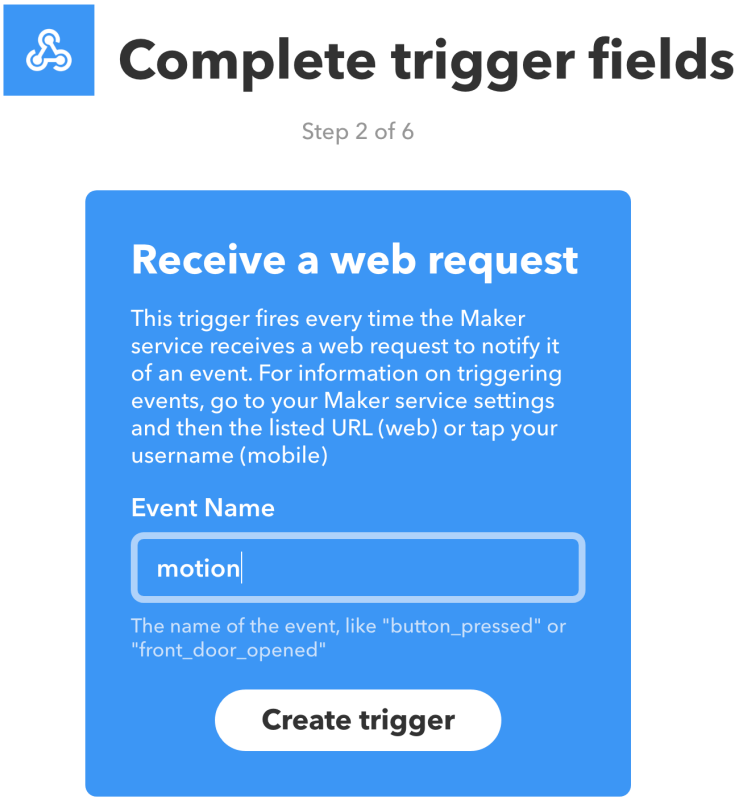


Step 5: Search for WebHooks and select it to create a web service

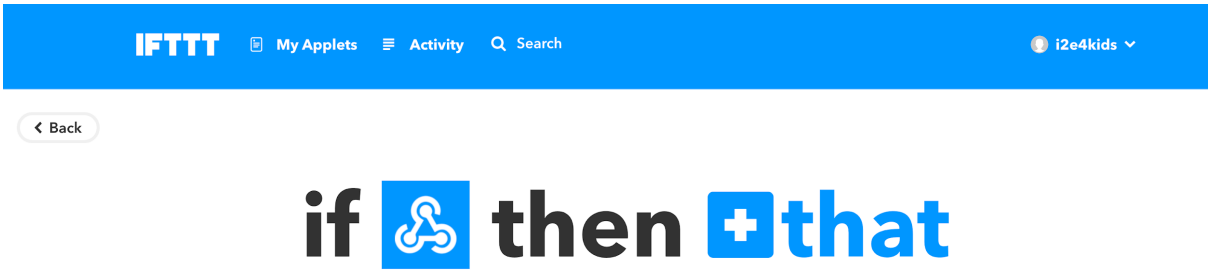


Step 6: Select Receive a web service request

Step 7: Enter the Event Name: **motion** and click Create trigger



Step 8: Click on + **that** and select Email



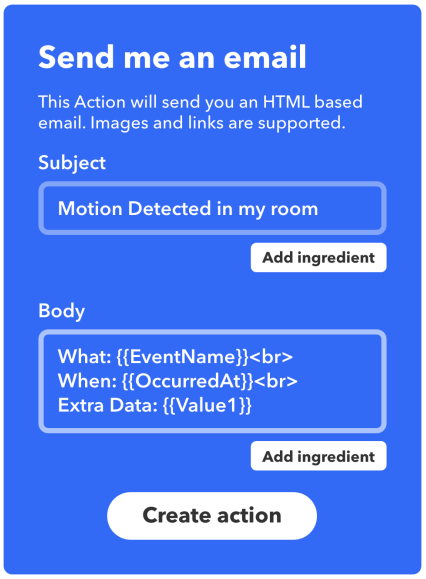
Step 9: Click on Send me an email

Step 10: Change Subject to “Motion Detected in my room”

Step 11: You may want to delete Value2 and Value3 from the Body

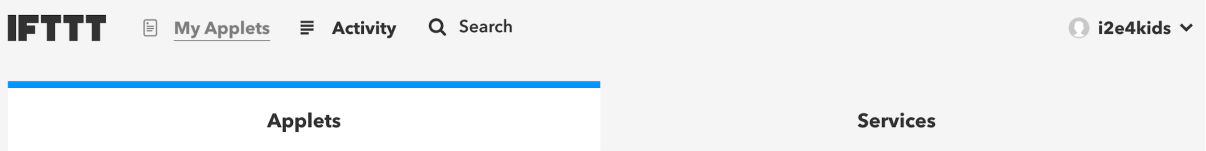
Note: We can always change the subject and body later also

Step 12: Click Create action and Finish

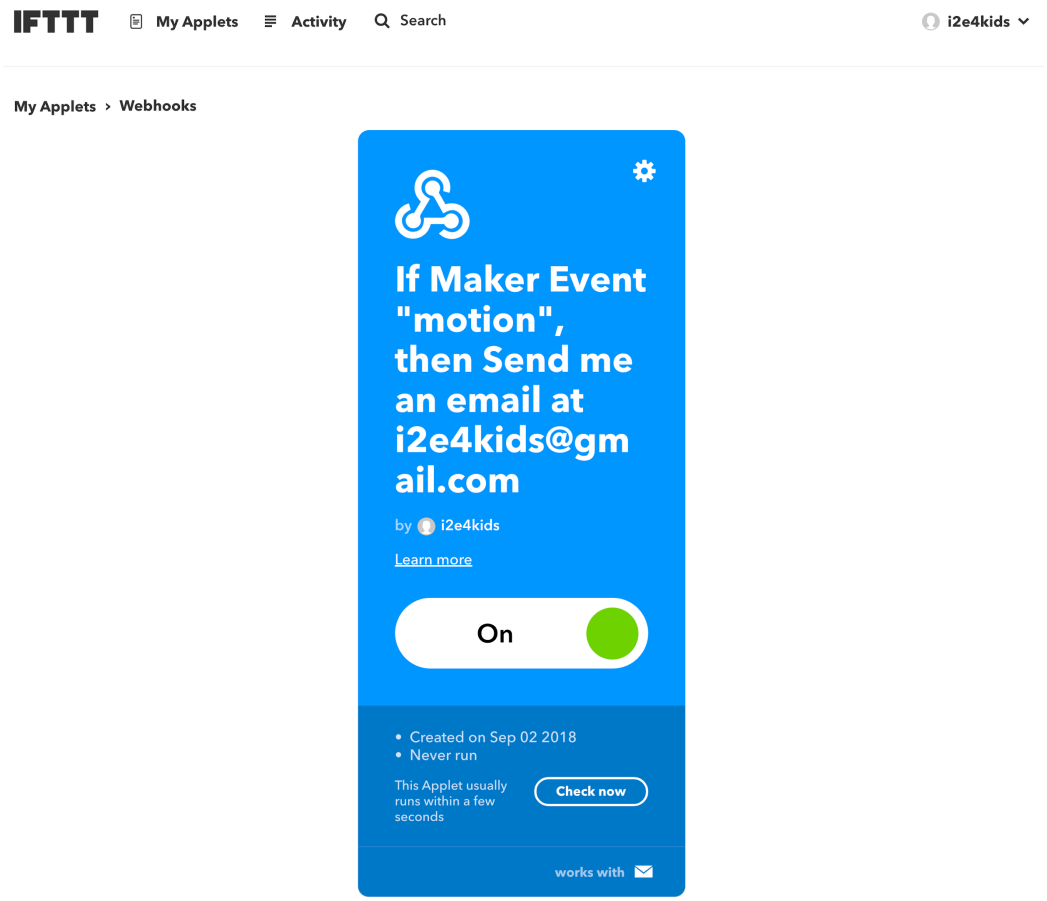


Trying the Web Service Trigger

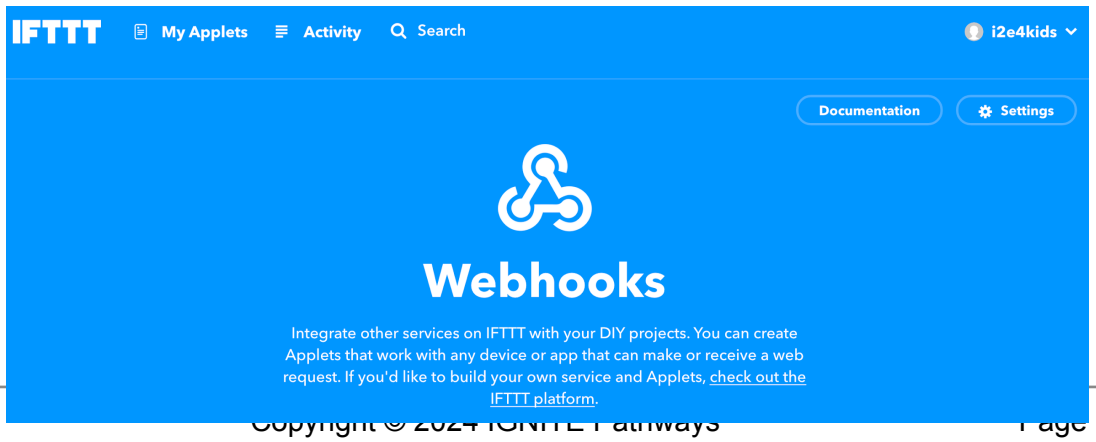
1. Go to My Applets from the top Menu



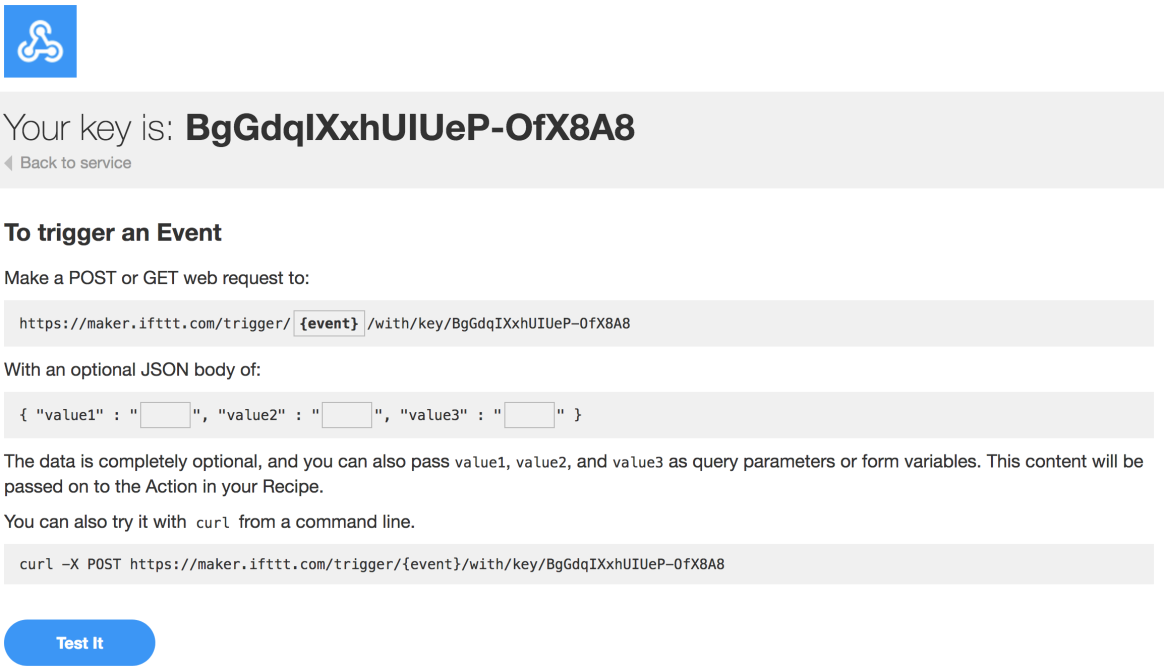
2. Select your Applet



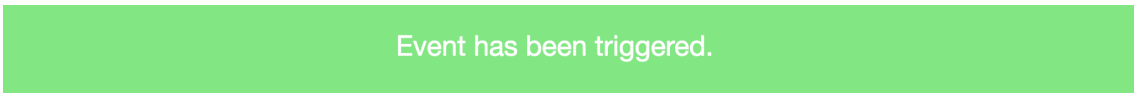
3. Click on MyApplets > **WebHooks** and you'll see following page with links to Documentation and Settings on top-right



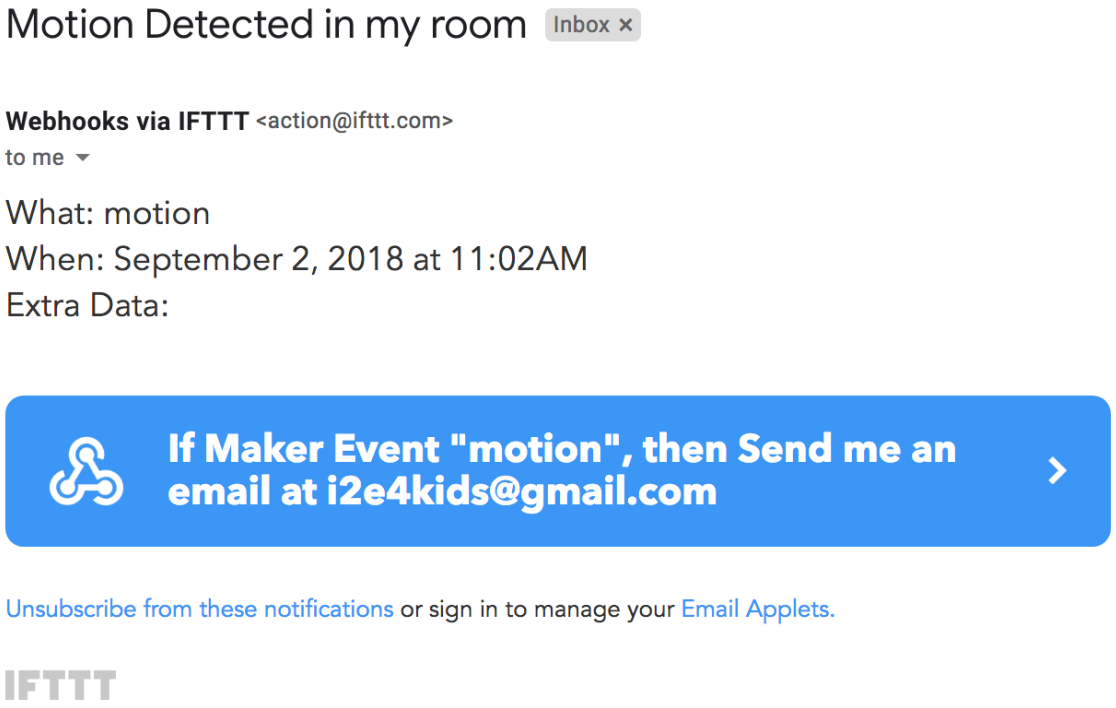
4. Click on Documentation to try your newly created web-service



5. Replace {event} with your event name “motion” and click **Test it**



6. Check your email and you should see an email from IFTTT



7. In the next project, we will tie this email trigger to the actual motion detector connected to ESP8266 NodeMCU

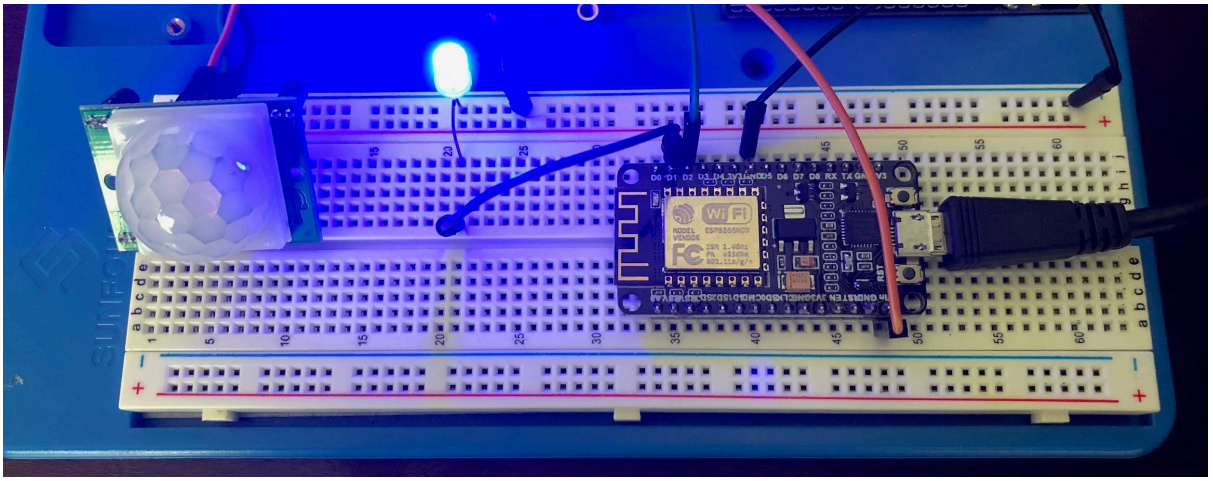
Project 12: Create an iOT device which sends an Email whenever motion is detected in my room

In this project, we will call the IFTTT web service from the ESP whenever a motion is detected.

Circuit

Connections

1. Connect GPIO5 to LED Positive
2. Connect GPIO4 to Motion Signal (middle pin)
3. Connect PIR Motion Sensor GND & VCC to GND & 5V of ESP
4. Connect LED Negative to GND of ESP



Sketch

```
#include <ESP8266WiFi.h>

const int ledPin= 5;     //GPIO5
const int motionPin= 4;  //MOTION ON GPIO4
// Replace with your SSID and password details
char ssid[] = "REPLACE_WITH_SSID";
char pass[] = "REPLACE_WITH_PASSWORD";

// posting interval of 10 minutes - to reduce spam!
const unsigned long postInterval = 10 * 60 * 1000;
// last time you connected to the server, in milliseconds
long lastConnectionTime = -postInterval;
WiFiClient client;

void setup(){
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);
  pinMode(motionPin, INPUT);
```

```
  // Set outputs to LOW
  digitalWrite(ledPin, LOW);
  WiFi.begin(ssid,pass);
  Serial.println();
  Serial.print("Connecting..");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\tWiFi Connected");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}
void loop(){
  int value= digitalRead(motionPin);
  if (value == HIGH) {
    Serial.print("Motion Detected: ");
    Serial.println(millis());
    digitalWrite(ledPin, HIGH);
    if (millis() - lastConnectionTime > postInterval) {
      // note the time that the connection was made:
      lastConnectionTime = millis();
      call_IFTTT_WebService();
    }
    delay(5000); //Keep the LED on for 5 seconds
    digitalWrite(ledPin, LOW);
    if (client.available()) {
      Serial.print("IFTTT Response: ");
      while (client.available()) {
        Serial.write(client.read());
      }
      Serial.println("----------");
    }
  }
  else {
    digitalWrite(ledPin, LOW);
  }
}
```

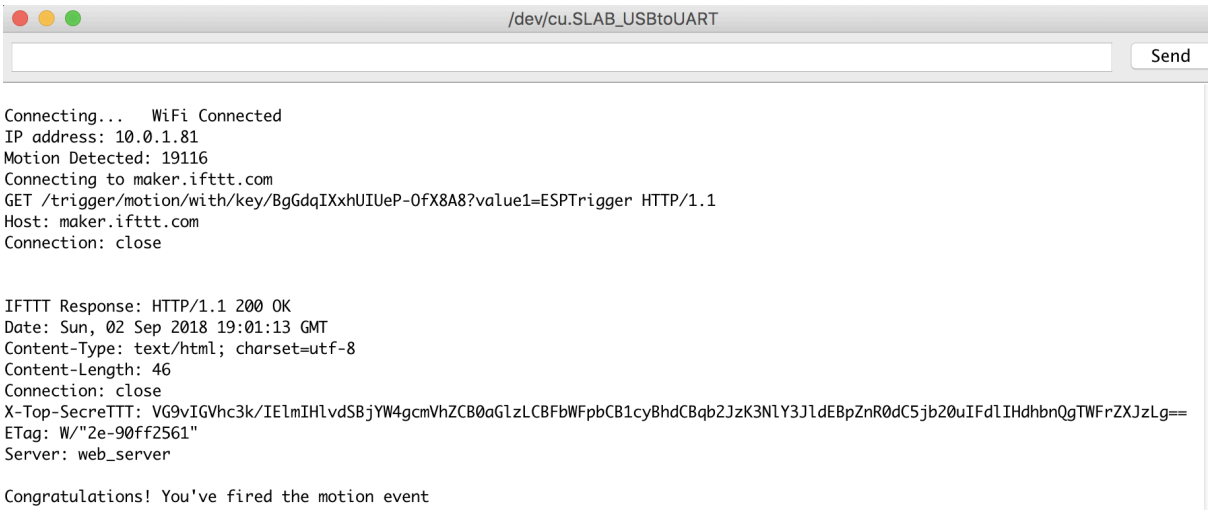
In Second Tab: IFTTT_WebService

```
//IFTTT Make WebHooks Service Trigger
const char* host = "maker.ifttt.com";
//Replace with your key
const char* key = "REPLACE_WITH_YOUR_WEBHOOKS_KEY";
const int httpPort = 80;

void call_IFTTT_WebService() {
  Serial.print("Connecting to ");
  Serial.println(host);
  if (!client.connect(host,httpPort)) {
    Serial.println("connection failed");
    return;
  }

  String url = "/trigger/motion/with/key/";
  url += key;
```

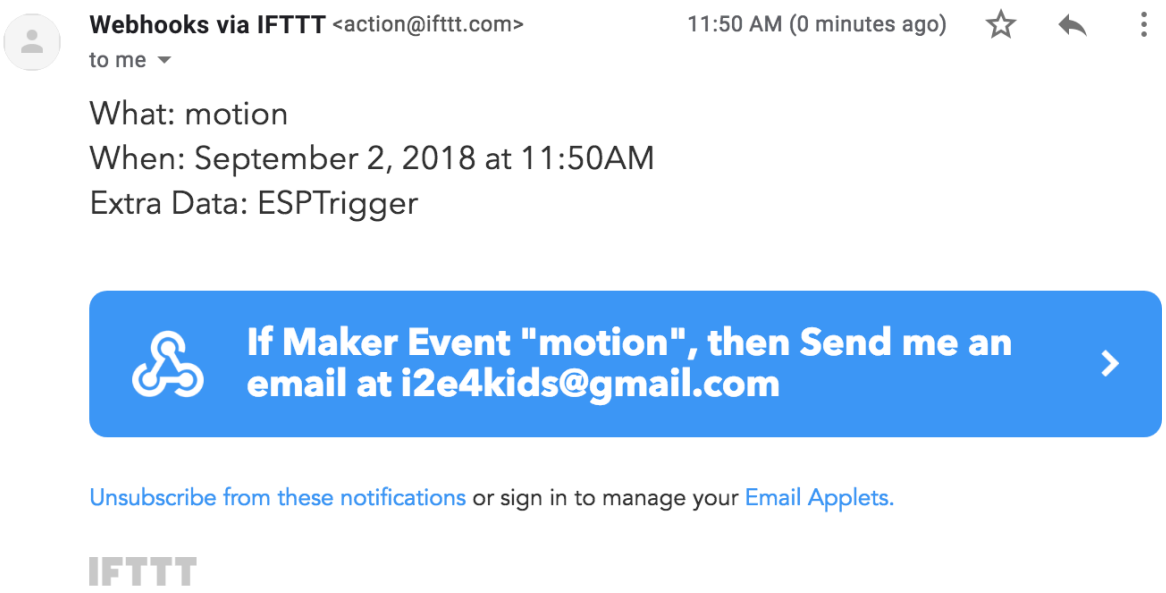
```
  url += "?value1=ESPTrigger"; //Can be relaced with any other data
  String req = String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n";

  Serial.println(req);
  client.print(req);
}
```

Try it out

Upload and run the sketch. Make sure to have the Serial Monitor Open



Check your email



Project 13: ESP8266 Weather Forecaster

In this project, we're going to learn how to connect ESP8266 NodeMCU to a server as a client, consume APIs to build a weather forecaster.

OpenWeatherMap APIs

OpenWeatherMap provides free APIs that we will use for this example. To use the forecast API, we need an API key, known as the APPID. We will use **e838b3766d0cc6be948657fc651cdc27** as our APPID. You can also sign-up on openweathermap.org to get your own APPID

`http://api.openweathermap.org/data/2.5/forecast?q={city},{country}&APPID={APP_KEY}&mode=json&units=Imperial&cnt=2`

Example:
`http://api.openweathermap.org/data/2.5/forecast?q=dublin,us&APPID=e838b3766d0cc6be948657fc651cdc27&mode=json&units=imperial&cnt=2`

Response


```
{
   "cod":"200",
   "message":0.0117,
   "cnt":2,
   "list":[
      {
         "dt":1535598000,
         "main":{
            "temp":72.12,
            "temp_min":72.12,
            "temp_max":73.24,
            "pressure":994.95,
            "sea_level":1030.85,
            "grnd_level":994.95,
            "humidity":95,
            "temp_kf":-0.63
         },
         "weather":[
            {
               "id":500,
               "main":"Rain",
               "description":"light rain",
               "icon":"10n"
            }
         ],
         "clouds":{
            "all":68
         },
         "wind":{
            "speed":6.73,
            "deg":316.502
```

```
        },
        "rain":{
           "3h":0.029999999999999
        },
        "sys":{
           "pod":"n"
        },
        "dt_txt":"2018-08-30 03:00:00"
     },
     {
        "dt":1535608800,
        "main":{
           "temp":70.39,
           "temp_min":70.39,
           "temp_max":71.24,
           "pressure":995.49,
           "sea_level":1031.63,
           "grnd_level":995.49,
           "humidity":97,
           "temp_kf":-0.47
        },
        "weather":[
           {
              "id":500,
              "main":"Rain",
              "description":"light rain",
              "icon":"10n"
           }
        ],
        "clouds":{
           "all":76
        },
        "wind":{
           "speed":6.15,
           "deg":348.005
        },
        "rain":{
           "3h":0.095000000000001
        },
        "sys":{
           "pod":"n"
        },
        "dt_txt":"2018-08-30 06:00:00"
     }
  ],
  "city":{
     "id":5152333,
     "name":"Dublin",
     "coord":{
        "lat":40.0992,
        "lon":-83.1141
     },
     "country":"US",
     "population":41751
  }
}
```

Weather parameters in API response

- `code` Internal parameter
- `message` Internal parameter
- `city`
  - `city.id` City ID
  - `city.name` City name
  - `city.coord`
    - `city.coord.lat` City geolocation, latitude
    - `city.coord.lon` City geolocation, longitude
  - `city.country` Country code (GB, JP etc.)
- `cnt` Number of lines returned by this API call
- `list`
  - `list.dt` Time of data forecasted, unix, UTC
  - `list.main`
    - `list.main.temp` Temperature. Unit Default: Kelvin, Metric: Celsius, Imperial: Fahrenheit.
    - `list.main.temp_min` Minimum temperature at the moment of calculation. This is deviation from 'temp' that is possible for large cities and megalopolises geographically expanded (use these parameter optionally). Unit Default: Kelvin, Metric: Celsius, Imperial: Fahrenheit.
    - `list.main.temp_max` Maximum temperature at the moment of calculation. This is deviation from 'temp' that is possible for large cities and megalopolises geographically expanded (use these parameter optionally). Unit Default: Kelvin, Metric: Celsius, Imperial: Fahrenheit.
    - `list.main.pressure` Atmospheric pressure on the sea level by default, hPa
    - `list.main.sea_level` Atmospheric pressure on the sea level, hPa
    - `list.main.grnd_level` Atmospheric pressure on the ground level, hPa
    - `list.main.humidity` Humidity, %
    - `list.main.temp_kf` Internal parameter
  - `list.weather` (more info Weather condition codes)
    - `list.weather.id` Weather condition id
    - `list.weather.main` Group of weather parameters (Rain, Snow, Extreme etc.)
    - `list.weather.description` Weather condition within the group
    - `list.weather.icon` Weather icon id
  - `list.clouds`
    - `list.clouds.all` Cloudiness, %
  - `list.wind`
    - `list.wind.speed` Wind speed. Unit Default: meter/sec, Metric: meter/sec, Imperial: miles/hour.
    - `list.wind.deg` Wind direction, degrees (meteorological)
  - `list.rain`
    - `list.rain.3h` Rain volume for last 3 hours, mm
  - `list.snow`
    - `list.snow.3h` Snow volume for last 3 hours
  - `list.dt_txt` Data/time of calculation, UTC

For more information on the API details, refer to:
https://openweathermap.org/forecast5

Installing the ArduinoJson library

For this project you need to install the ArduinoJson library in your Arduino IDE that allows you to **Decode** and **Encode** JSON with Arduino or ESP8266.

7. Download the ArduinoJson from below:
   https://github.com/bblanchon/ArduinoJson/archive/master.zip
8. You should have a .zip folder in your Downloads folder.
9. Unzip the .zip folder and you should get ArduinoJson-master folder
10. Rename your folder from ArduinoJson-master to ArduinoJson
11. Move the ArduinoJson folder to your Arduino IDE installation libraries folder
12. Reopen your Arduino IDE

Sketch

```
#include <ESP8266WiFi.h>
#include <ArduinoJson.h>

// Replace with your SSID and password details
char ssid[] = "REPLACE_WITH_SSID";
char pass[] = "REPLACE_WITH_PASSWORD";

// last time you connected to the server, in milliseconds
unsigned long lastConnectionTime = 0;
// posting interval of 10 minutes
const unsigned long postInterval = 10 * 60 * 1000;

bool DEBUG = false;
void setup() {
  Serial.begin(115200);

  WiFi.begin(ssid,pass);
  Serial.println();
  Serial.print("Connecting..");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi Connected");
  printWiFiStatus();
}

void loop() {
  //check if 10mins has passed then conect again and pull
```

```
  if (millis() - lastConnectionTime > postInterval) {
    // note the time that the connection was made:
    lastConnectionTime = millis();
    makehttpRequest();
  }
}

// print Wifi status
void printWiFiStatus() {
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print your WiFi shield's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);
}

//to parse json data received from OWM
void parseJson(const char * jsonString) {
  DynamicJsonBuffer jsonBuffer(2000);
  // FIND FIELDS IN JSON TREE
  JsonObject& root = jsonBuffer.parseObject(jsonString);
  if (!root.success()) {
    Serial.println("parseObject() failed");
    return;
  }

  JsonArray& list = root["list"];
  JsonObject& nowT = list[0];
  JsonObject& later = list[1];

  String city = root["city"]["name"];
  float tempNow = nowT["main"]["temp"];
  float tempMin = nowT["main"]["temp_min"];
  float tempMax = nowT["main"]["temp_max"];
  float humidityNow = nowT["main"]["humidity"];
  String weatherNow = nowT["weather"][0]["description"];
  showCurrentWeather(city, weatherNow, tempNow, tempMax, tempMin,
humidityNow);

  float tempLater = later["main"]["temp"];
  float humidityLater = later["main"]["humidity"];
  String weatherLater = later["weather"][0]["description"];
  //Optionally display forecast to LCD display
}
```

Second Tab / File

```
#include <WiFiClient.h>

// Open Weather Map API server name
#define APISERVER "api.openweathermap.org"

// Replace the next line to match your city and 2 letter country code
#define CITY_COUNTRY "dublin,us"
```

```
// Replace the next line with your API Key
#define API_KEY "e838b3766d0cc6be948657fc651cdc27"

// 20 minute delay between updates after an error
#define DELAY_ERROR     (20*60*1000)

// HTTP request
const char APISERVER_REQ[] =
    "GET  /data/2.5/forecast?q=" CITY_COUNTRY "&APPID=" API_KEY
"&mode=json&units=imperial&cnt=2 HTTP/1.1\r\n"
    "User-Agent: ESP8266/0.1\r\n"
    "Accept: */*\r\n"
    "Host: " APISERVER "\r\n"
    "Connection: close\r\n"
    "\r\n";

static char respBuf[4096];
WiFiClient client;

// to request data from API Service
void makehttpRequest() {
  // close any connection before send a new request to allow client make
connection to server
  if (DEBUG) { Serial.println("Making Http Request"); }
  client.stop();

  // if there's a successful connection:
  if (!client.connect(APISERVER, 80)) {
    Serial.println(F("connection failed"));
    delay(DELAY_ERROR);
    return;
  }
  if (DEBUG) { Serial.println("connected"); }
  // send the HTTP PUT request:
  client.print(APISERVER_REQ);
  client.flush();

  unsigned long timeout = millis();
  while (client.available() == 0) {
    if (millis() - timeout > 5000) {
      Serial.println(">>> Client Timeout !");
      client.stop();
      return;
    }
  }

  int respLen = 0;
  bool skip_headers = true;
  while (client.connected() || client.available()) {
    if (skip_headers) {
      String aLine = client.readStringUntil('\n');
      if (DEBUG) {
        Serial.println(aLine);
      }
      // Blank line denotes end of headers
      if (aLine.length() <= 1) {
        skip_headers = false;
      }
```

```
    }
    else {
      int bytesIn;
      bytesIn = client.read((uint8_t *)&respBuf[respLen], sizeof(respBuf) -
respLen);
      if (bytesIn > 0) {
        respLen += bytesIn;
        if (respLen > sizeof(respBuf)) respLen = sizeof(respBuf);
      }
      else if (bytesIn < 0) {
        Serial.print(F("read error "));
        Serial.println(bytesIn);
      }
    }
    delay(1);
  }
  client.stop();

  if (respLen >= sizeof(respBuf)) {
    Serial.print(F("respBuf overflow "));
    Serial.println(respLen);
    delay(DELAY_ERROR);
    return;
  }

  // Terminate the C string
  respBuf[respLen++] = '\0';
  if (DEBUG) {
    Serial.print(F("respLen "));
    Serial.println(respLen);
    Serial.println(respBuf);
  }
  parseJson(respBuf);
}
```

Third Tab: Display Weather on the LCD

```
#include <LiquidCrystal_I2C.h>
// set the LCD number of columns and rows
int lcdColumns = 16;
int lcdRows = 2;

// set LCD address, number of columns and rows
// if you don't know your display address, run an I2C scanner sketch
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);

void initializeLCD() {
  // initialize LCD
  lcd.init();
  // turn on LCD backlight
  lcd.backlight();
}

void showCurrentWeather(String city, String weatherNow, float tempNow, float
tempMax, float tempMin, float humidityNow) {
    Serial.print(city); Serial.printf(" %g%sF", tempNow, "\u00B0");
    Serial.println(", " + weatherNow);
```

```
    Serial.printf("%.1f%sF - %.1f%sF", tempMax, "\u00B0", tempMin, "\u00B0");
    Serial.printf(", Humidity %g%c", humidityNow, 37);
    Serial.println();

    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Hello, World!");
    lcd.setCursor(0, 1);
    lcd.print("Hello, World!");
}
```

Project 14: Use 128x32 I2C OLED SSD1306 with ESP

For this project we'll be using a 128x32 I2C OLED display, but OLEDs with other sizes should also work.

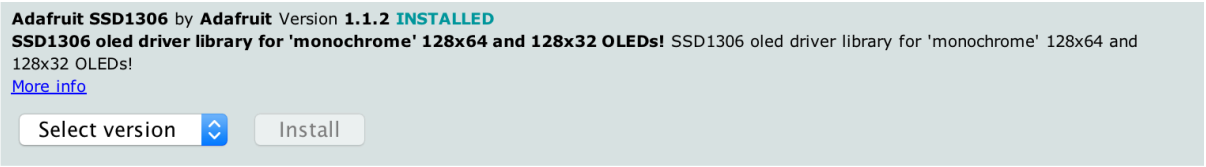
PIN Connections

I2C LCD	ESP8266
GND	GND
VCC	VIN
SDA	D2 (GPIO 4)
SCL	D1 (GPIO 5)

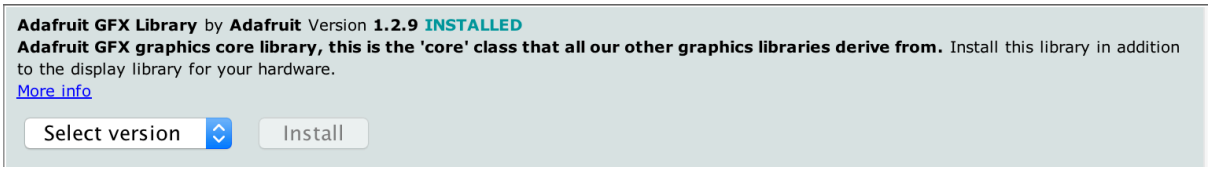
Install the Adafruit_SSD1306 & Adafruit_GFX Library

For this project you need to install the LiquidCrystal_I2C library in your Arduino IDE that allows you to work with LCD I2C interface.

1. Go to Sketch -> Include Library -> Manage Libraries and search for SSD1306
2. Select Adafruit SSD1306 and click Install



3. Search for Adafruit GFX Library, select and click Install



4. Reopen your Arduino IDE

Get the OLED I2C Address

I2C is a two wire protocol that allows multiple devices to be used, with only two pins used on the microcontroller. This is accomplished by setting an address on each device on the bus. Most all I2C OLED use the same address, but it's good to scan to be sure. With the OLED properly wired to the ESP, upload the I2C Scanner sketch (It's the same sketch we use for LCD)

After uploading the code, open the Serial Monitor at a baud rate of 115200. The I2C address should be displayed in the Serial Monitor.

Displaying Text on the OLED Display

We will update the previous project to display weather on the OLED display. Add a new tab in the previous project and use following sketch. Make sure to update the parseJSON method and call **`showWeatherOnOLED`** method to display weather on OLED Display instead of LCD

Sketch

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define OLED_RESET 3 //Change from 4 to 3 for ESP
Adafruit_SSD1306 display(OLED_RESET);
#define LOGO16_GLCD_HEIGHT 16
#define LOGO16_GLCD_WIDTH  16

void initializeOLED() {
  // by default, we'll generate the high voltage from the 3.3v line
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);  // initialize with I2C addr
0x3C (for the 128x32) or what you found from scanner sketch
  // Clear the buffer.
  display.clearDisplay();
}

void displayIPAddress(String ssid, IPAddress
ip) {
  // Clear the buffer.
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0,0);
  display.print("SSID: ");
  display.println(ssid); display.setCursor(0,16);
  display.print("IP Address: "); display.println(ip);
  display.display();
}

void showWeatherOnOLED(String city, String weatherNow, float tempNow, float
tempMax, float tempMin, float humidityNow) {
    Serial.print(city); Serial.printf(" %g%sF", tempNow, "\u00B0");
Serial.println(", " + weatherNow);
    Serial.printf("%.1f%sF - %.1f%sF",
```

```
tempMax, "\u00B0", tempMin, "\u00B0");
    Serial.printf(", Humidity %g%c", humidityNow, 37);
    Serial.println();
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(0, 0);
    display.print(city); display.printf(": %.1fF", tempNow);
    display.printf(", H %g%c", humidityNow, 37);
    display.setCursor(0, 16);
    display.print(weatherNow);
    display.display();
}
```

References

Arduino Website

URL: https://www.arduino.cc/
This is a great resource learn more about Arduino, different tyoes of Arduino boards, download Arduino libraries, and collaborate with others

Sparkfun

URL: https://learn.sparkfun.com/

Adafruit

URL: https://learn.adafruit.com/
Learn Arduino and other microcontrollers. Download libraries

Register Guide

URL: http://www.resistorguide.com/
Learn all about registers

Wikipedia

URL: https://en.wikipedia.org/wiki

Electronic Circuits

URL: https://www.build-electronic-circuits.com
Learn More about Electronics