# WiFi ESP8266 (LiLon)

GitHub link : <https://github.com/esp8266/Arduino>

Instructables : <https://www.instructables.com/id/NodeMCU-ESP8266-Details-and-Pinout/> (good)

With Arduino : <https://lastminuteengineers.com/esp8266-nodemcu-arduino-tutorial/> (really really good)

***The ESP8266 Integrates 802.11b/g/n HT40 Wi-Fi transceiver, so it can not only connect to a WiFi network and interact with the Internet, but it can also set up a network of its own, allowing other devices to connect directly to it. This makes the ESP8266 NodeMCU even more versatile.***

***Warning:***

***The ESP8266 requires a 3.3V power supply and 3.3V logic levels for communication. The GPIO pins are not 5V-tolerant! If you want to interface the board with 5V (or higher) components, you’ll need to do some level shifting.***

***200420 paid A$12.05 ea***

From <https://esp8266-shop.com/product/nodemcu-esp8266-esp-12e/>

The ESP8266 is a System on a Chip (SoC), manufactured by the Chinese company Espressif. It consists of a Tensilica L106 32-bit micro controller unit (MCU) and a Wi-Fi transceiver. It has 11 GPIO pins, and an analog input as well. This means that you can program it like any normal Arduino or other microcontroller. And on top of that, you get Wi-Fi communication, so you can use it to connect to your Wi-Fi network, connect to the Internet, host a web server with real web pages, let your smartphone connect to it, etc

It contains a built-in 32-bit low-power CPU, ROM and RAM. It is a complete and self-contained Wi-Fi network solution that can carry software applications as a stand-alone device or connected with a microcontroller (MCU). The module has built-in AT Command firmware to be used with any MCU via COM port. The ESP8266 can be flashed and programed using the Arduino IDE. Due to its large open source developer community, a large number of libraries for this popular microcontroller is available .

Processor:

The ESP8266EX integrates a Tensilica L106 32-bit RISC processor, which achieves extra-low power consumption and reaches a maximum clock speed of 160 MHz. The Real-Time Operating System (RTOS) and Wi-Fi stack allow 80% of the processing power to be available for user application programming and development.

Memory:

32 KB instruction RAM

80 KB user data RAM

16 KB ETS system data RAM

External QSPI flash: up to 16 MB is supported (512 KB to 4 MB typically included)

IEEE 802.11 b/g/n Wi-Fi

Integrated TR switch, balun, LNA, power amplifier and matching networkWEP or WPA/WPA2 authentication, or open networks

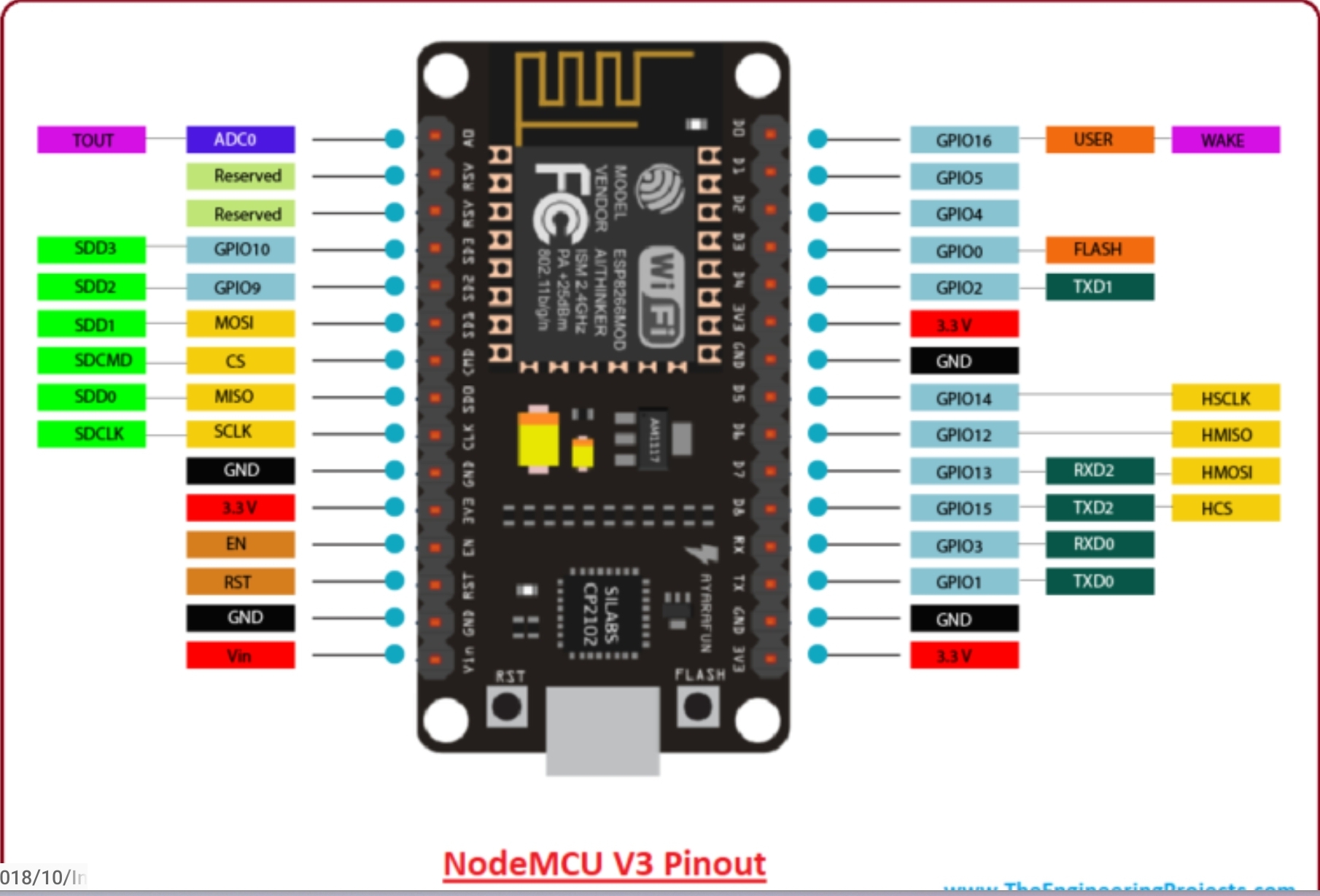
16 GPIO pins

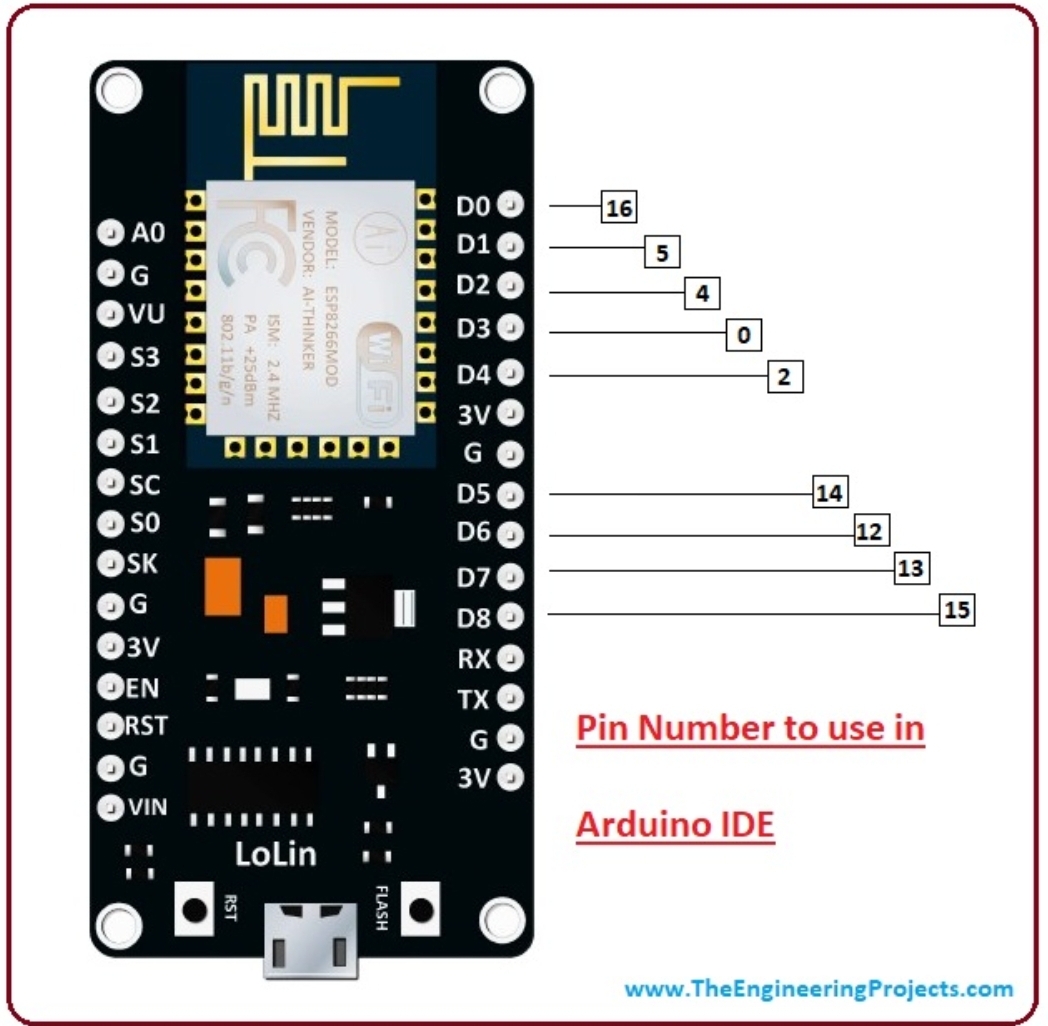
SPI I²C (software implementation)

I²S interfaces with DMA (sharing pins with GPIO)

UART on dedicated pins, plus a transmit-only UART can be enabled on GPIO210-bit ADC (succes

sive approximation ADC)





The board also has a LED indicator which is user programmable and is connected to the D0 pin of the board

### How to Power NodeMCU V3?

You can see from the pinout image above, there are five ground pins and three 3V3 pins on the board. The board can be powered up using the following three ways.

USB Power. It proves to an ideal choice for loading programs unless the project you aim to design requires separate interface i.e. disconnected from the computer.

Provide 3.3V. This is another great option to power up the module. If you have your own off-board regulator, you can generate an instant power source for your development kit.

Power Vin. This is a voltage regulator that comes with the ability to support up to 800 mA. It can handle somewhere between 7 to 12 V. You cannot power the devices operating at 3.3 V, as this regulator unable to generate as low as 3.3V.

Power to the ESP8266 NodeMCU is supplied via the on-board MicroB USB connector. Alternatively, if you have a regulated 5V voltage source, the VIN pin can be used to directly supply the ESP8266 and its peripherals.

### Of interest

The term NodeMCU usually refers to the firmware, while the board is called Devkit.

NodeMCU Devkit 1.0 consists of an ESP-12E on a board, which facilitates its use.

It also has a voltage regulator, and a USB interface.

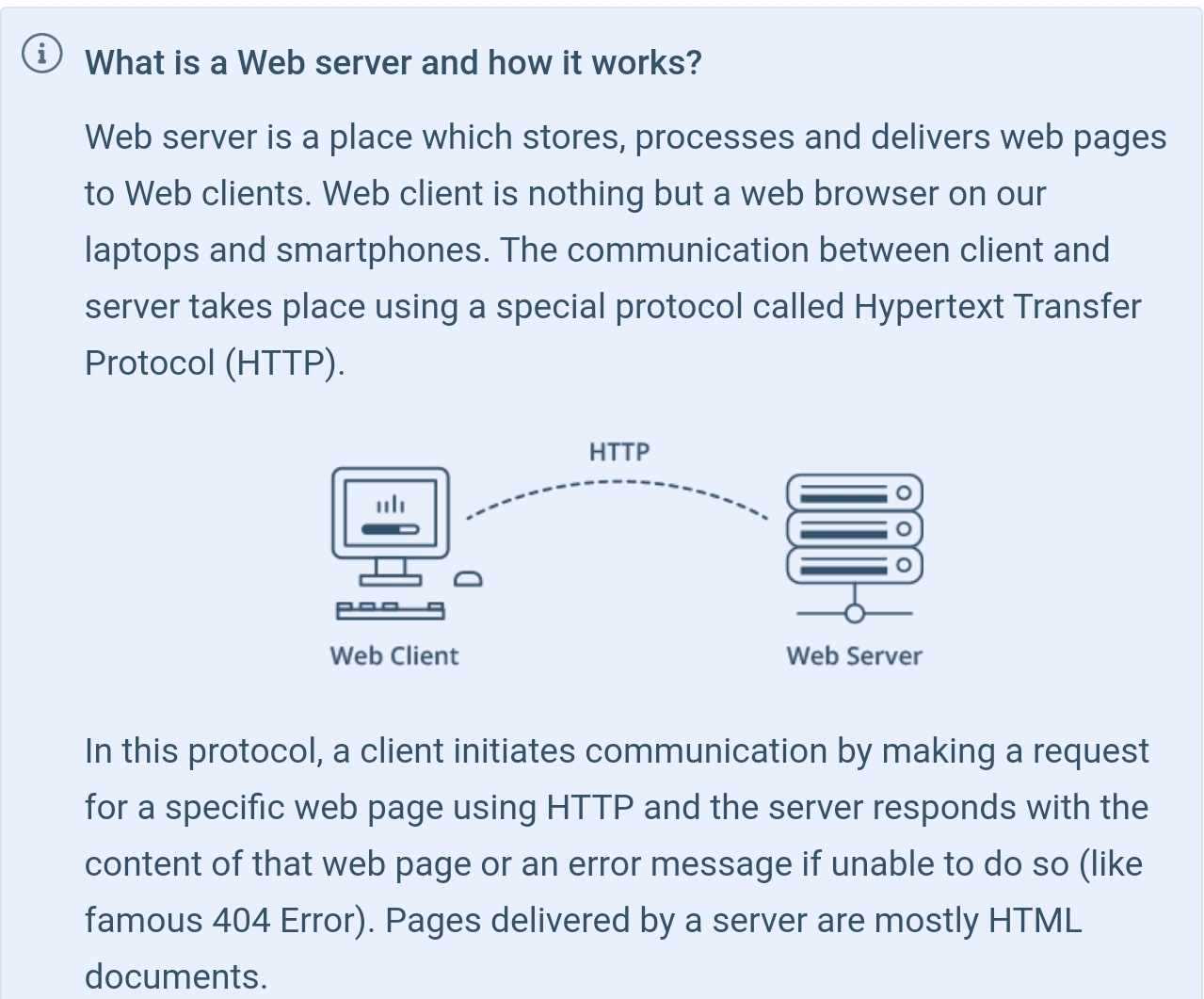
### GPIO Pins

ESP8266 NodeMCU has 17 GPIO pins which can be assigned to various functions such as I2C, I2S, UART, PWM, IR Remote Control, LED Light and Button programmatically.

Each digital enabled GPIO can be configured to internal pull-up or pull-down, or set to high impedance. When configured as an input, it can also be set to edge-trigger or level-trigger to generate CPU interrupts.

## ESP8266 NodeMCU Web Server In Arduino IDE

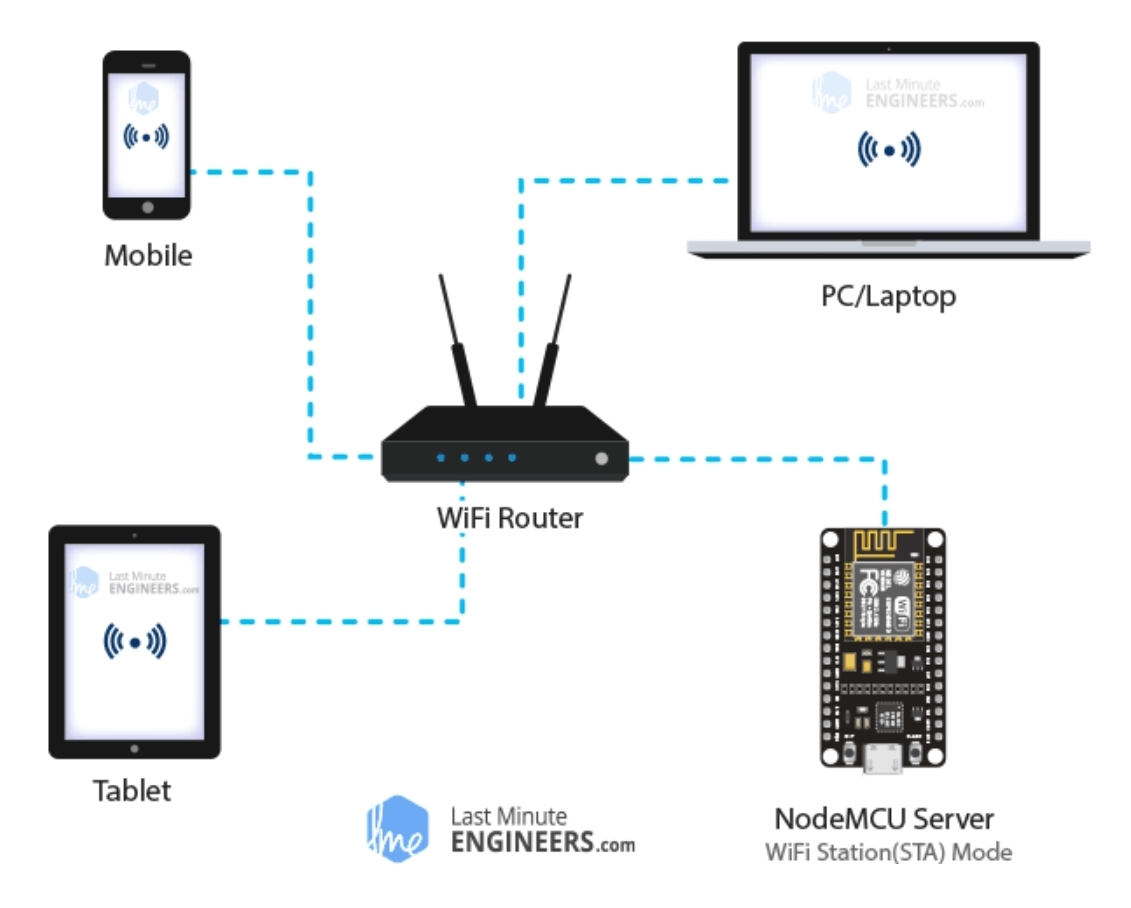
<https://lastminuteengineers.com/creating-esp8266-web-server-arduino-ide/>



ESP8266 Operating Modes

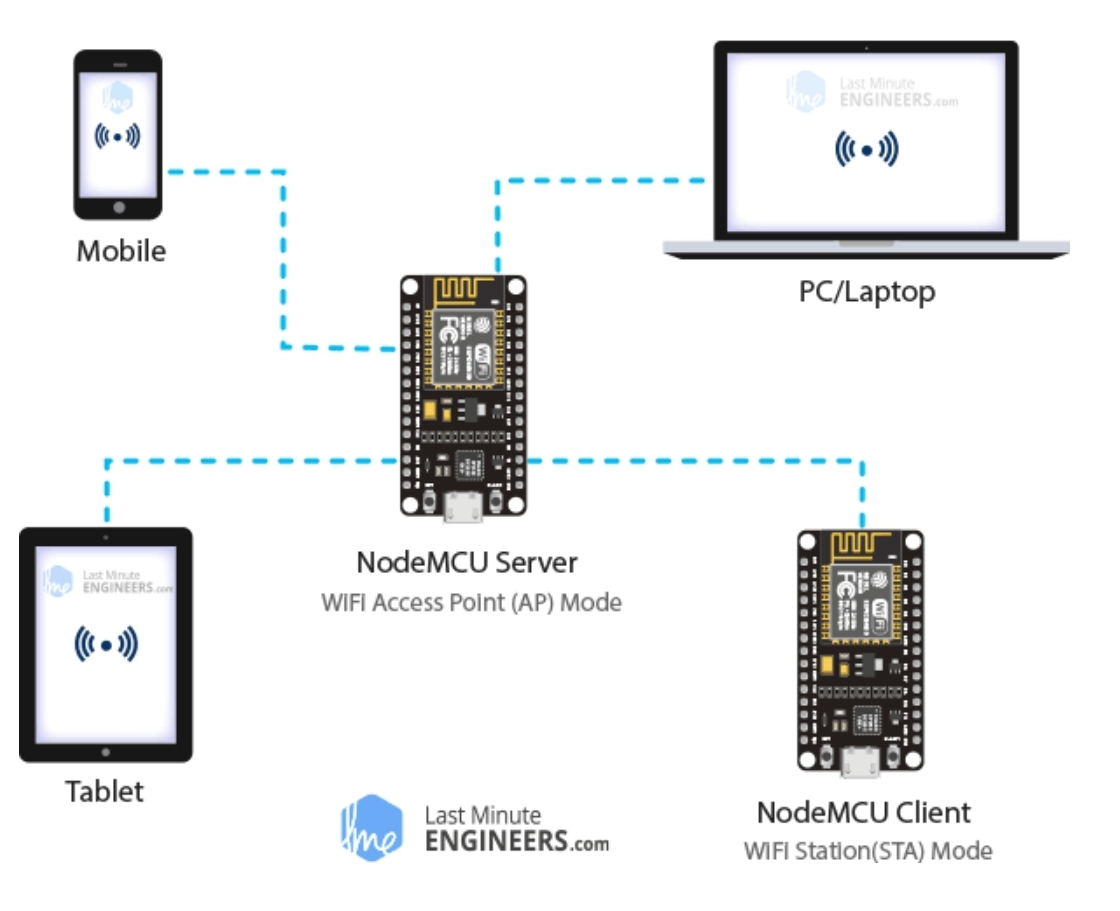
One of the greatest features ESP8266 provides is that it cannot only connect to an existing WiFi network and act as a Web Server, but it can also set up a network of its own, allowing other devices to connect directly to it and access web pages. This is possible because ESP8266 can operate in three different modes: Station mode, Soft Access Point mode, and both at the same time. This provides possibility of building mesh networks.

STA or Station mode



### Soft Access Point (AP) Mode

The ESP8266 that creates its own WiFi network and acts as a hub (Just like WiFi router) for one or more stations is called Access Point (AP). Unlike WiFi router, it does not have interface to a wired network. So, such mode of operation is called Soft Access Point (soft-AP). Also the maximum number of stations that can connect to it is limited to five.



In AP mode ESP8266 creates a new WiFi network and sets SSID (Name of the network) and IP address to it. With this IP address, it can deliver web pages to all connected devices under its own network.

### ESP8266 as HTTP Server using WiFi Access Point (AP) mode

This example demonstrates how to turn the ESP8266 into an access point (AP), and serve up web pages to any connected client. To start with, plug your ESP8266 NodeMCU into your computer and Try the sketch out; and then we will dissect it in some detail.

#include <ESP8266WiFi.h>

#include <ESP8266WebServer.h>

/\* Put your SSID & Password \*/

const char\* ssid = "NodeMCU"; // Enter SSID here

const char\* password = "12345678"; //Enter Password here

/\* Put IP Address details \*/

IPAddress local\_ip(192,168,1,1);

IPAddress gateway(192,168,1,1);

IPAddress subnet(255,255,255,0);

ESP8266WebServer server(80);

uint8\_t LED1pin = D7;

bool LED1status = LOW;

uint8\_t LED2pin = D6;

bool LED2status = LOW;

void setup() {

Serial.begin(115200);

pinMode(LED1pin, OUTPUT);

pinMode(LED2pin, OUTPUT);

WiFi.softAP(ssid, password);

WiFi.softAPConfig(local\_ip, gateway, subnet);

delay(100);

server.on("/", handle\_OnConnect);

server.on("/led1on", handle\_led1on);

server.on("/led1off", handle\_led1off);

server.on("/led2on", handle\_led2on);

server.on("/led2off", handle\_led2off);

server.onNotFound(handle\_NotFound);

server.begin();

Serial.println("HTTP server started");

}

void loop() {

server.handleClient();

if(LED1status)

{digitalWrite(LED1pin, HIGH);}

else

{digitalWrite(LED1pin, LOW);}

if(LED2status)

{digitalWrite(LED2pin, HIGH);}

else

{digitalWrite(LED2pin, LOW);}

}

void handle\_OnConnect() {

LED1status = LOW;

LED2status = LOW;

Serial.println("GPIO7 Status: OFF | GPIO6 Status: OFF");

server.send(200, "text/html", SendHTML(LED1status,LED2status));

}

void handle\_led1on() {

LED1status = HIGH;

Serial.println("GPIO7 Status: ON");

server.send(200, "text/html", SendHTML(true,LED2status));

}

void handle\_led1off() {

LED1status = LOW;

Serial.println("GPIO7 Status: OFF");

server.send(200, "text/html", SendHTML(false,LED2status));

}

void handle\_led2on() {

LED2status = HIGH;

Serial.println("GPIO6 Status: ON");

server.send(200, "text/html", SendHTML(LED1status,true));

}

void handle\_led2off() {

LED2status = LOW;

Serial.println("GPIO6 Status: OFF");

server.send(200, "text/html", SendHTML(LED1status,false));

}

void handle\_NotFound(){

server.send(404, "text/plain", "Not found");

}

String SendHTML(uint8\_t led1stat,uint8\_t led2stat){

String ptr = "<!DOCTYPE html> <html>\n";

ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";

ptr +="<title>LED Control</title>\n";

ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";

ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;} h3 {color: #444444;margin-bottom: 50px;}\n";

ptr +=".button {display: block;width: 80px;background-color: #1abc9c;border: none;color: white;padding: 13px 30px;text-decoration: none;font-size: 25px;margin: 0px auto 35px;cursor: pointer;border-radius: 4px;}\n";

ptr +=".button-on {background-color: #1abc9c;}\n";

ptr +=".button-on:active {background-color: #16a085;}\n";

ptr +=".button-off {background-color: #34495e;}\n";

ptr +=".button-off:active {background-color: #2c3e50;}\n";

ptr +="p {font-size: 14px;color: #888;margin-bottom: 10px;}\n";

ptr +="</style>\n";

ptr +="</head>\n";

ptr +="<body>\n";

ptr +="<h1>ESP8266 Web Server</h1>\n";

ptr +="<h3>Using Access Point(AP) Mode</h3>\n";

if(led1stat)

{ptr +="<p>LED1 Status: ON</p><a class=\"button button-off\" href=\"/led1off\">OFF</a>\n";}

else

{ptr +="<p>LED1 Status: OFF</p><a class=\"button button-on\" href=\"/led1on\">ON</a>\n";}

if(led2stat)

{ptr +="<p>LED2 Status: ON</p><a class=\"button button-off\" href=\"/led2off\">OFF</a>\n";}

else

{ptr +="<p>LED2 Status: OFF</p><a class=\"button button-on\" href=\"/led2on\">ON</a>\n";}

ptr +="</body>\n";

ptr +="</html>\n";

return ptr;

}

### Accessing the Web Server in AP mode

After uploading the sketch, open the Serial Monitor at a baud rate of 115200. And press the RESET button on ESP8266. If everything is OK, it will show HTTP server started message.

Next, find any device that you can connect to a WiFi network – phone, laptop, etc. And look for a network called NodeMCU. Join the network with password 123456789.

After connecting to your NodeMCU AP network, load up a browser and point it to 192.168.1.1 The NodeMCU should serve up a web page showing current status of LEDs and two buttons to control them. If take a look at the serial monitor at the same time, you can see status of NodeMCU’s GPIO pins.

Now, click the button to turn LED1 ON while keeping an eye on the URL. Once you click the button, the ESP8266 receives a request for /led1on URL. It then turns the LED1 ON and serves a web page with status of LED updated. It also prints the status of GPIO pin on the serial monitor.

You can test LED2 button and check that it works in a similar way.

Now, let’s take a closer look at the code to see how it works, so that you are able to modify it to fulfill your needs.

Detailed Code Explanation

The sketch starts by including ESP8266WiFi.h library.

This library provides ESP8266 specific WiFi methods we are calling to connect to network. Following that we also include the ESP8266WebServer.h library, which has some methods available that will help us setting up a server and handle incoming HTTP requests without needing to worry about low level implementation details.

#include <ESP8266WiFi.h>

#include <ESP8266WebServer.h>

As we are setting the ESP8266 NodeMCU in Access Point (AP) mode, it will create a WiFi network. Hence, we need to set its SSID, Password, IP address, IP subnet mask and IP gateway.

/\* Put your SSID & Password \*/

const char\* ssid = "NodeMCU"; // Enter SSID here

const char\* password = "12345678"; //Enter Password here

/\* Put IP Address details \*/

IPAddress local\_ip(192,168,1,1);

IPAddress gateway(192,168,1,1);

IPAddress subnet(255,255,255,0);

Next, we declare an object of ESP8266WebServer library, so we can access its functions. The constructor of this object takes port (where the server will be listening to) as a parameter. Since 80 is the default port for HTTP, we will use this value. Now you can access the server without needing to specify the port in the URL.

// declare an object of ESP8266WebServer library

ESP8266WebServer server(80);

Next, we declare the NodeMCU’s GPIO pins to which LEDs are connected and their initial state.

uint8\_t LED1pin = D7;

bool LED1status = LOW;

uint8\_t LED2pin = D6;

bool LED2status = LOW;

Inside Setup() Function

We configure our HTTP server before actually running it. First of all, we open a serial connection for debugging purpose and set GPIO ports to OUTPUT.

Serial.begin(115200);

pinMode(LED1pin, OUTPUT);

pinMode(LED2pin, OUTPUT);

Then, we set up a soft access point to establish a Wi-Fi network by proving SSID, Password, IP address, IP subnet mask and IP gateway.

WiFi.softAP(ssid, password);

WiFi.softAPConfig(local\_ip, gateway, subnet);

delay(100);

In order to handle incoming HTTP requests, we need to specify which code to execute when a particular URL is hit. To do so, we use on method. This method takes two parameters. First one is a URL path and second one is the name of function which we want to execute when that URL is hit.

For example, the first line of below code snippet indicates that when a server receives an HTTP request on the root (/) path, it will trigger the handle\_OnConnect() function. Note that the URL specified is a relative path.

Likewise, we need to specify 4 more URLs to handle two states of 2 LEDs.

server.on("/", handle\_OnConnect);

server.on("/led1on", handle\_led1on);

server.on("/led1off", handle\_led1off);

server.on("/led2on", handle\_led2on);

server.on("/led2off", handle\_led2off);

We haven’t specified what the server should do if the client requests any URL other than specified with server.on() . It should respond with an HTTP status 404 (Not Found) and a message for the user. We put this in a function as well, and use server.onNotFound() to tell it that it should execute it when it receives a request for a URI that wasn’t specified with server.on

server.onNotFound(handle\_NotFound);

Now, to start our server, we call the begin method on the server object.

server.begin(); Serial.println("HTTP server started");

Inside Loop() Function

To handle the actual incoming HTTP requests, we need to call the handleClient() method on the server object. We also change the state of LED as per the request.

void loop() {

server.handleClient();

if(LED1status) {

digitalWrite(LED1pin, HIGH);

} else {

digitalWrite(LED1pin, LOW);

}

if(LED2status) {

digitalWrite(LED2pin, HIGH);

} else {

digitalWrite(LED2pin, LOW);

}

}

Next, we need to create a function we attached to root (/) URL with server.on. Remember? At the start of this function, we set the status of both the LEDs to LOW (Initial state of LEDs) and print it on serial monitor. In order to respond to the HTTP request, we use the send method. Although the method can be called with a different set of arguments, its simplest form consists of the HTTP response code, the content type and the content.

In our case, we are sending the code 200 (one of the HTTP status codes), which corresponds to the OK response. Then, we are specifying the content type as “text/html“, and finally we are calling SendHTML() custom function which creates a dynamic HTML page containing status of LEDs.

void handle\_OnConnect() {

LED1status = LOW;

LED2status = LOW;

Serial.println("GPIO7 Status: OFF | GPIO6 Status: OFF");

server.send(200, "text/html", SendHTML(LED1status,LED2status));

}

Likewise, we need to create four functions to handle LED ON/OFF requests and 404 Error page.

void handle\_led1on() {

LED1status = HIGH;

Serial.println("GPIO7 Status: ON");

server.send(200, "text/html", SendHTML(true,LED2status));

}

void handle\_led1off() {

LED1status = LOW;

Serial.println("GPIO7 Status: OFF");

server.send(200, "text/html", SendHTML(false,LED2status));

}

void handle\_led2on() {

LED2status = HIGH;

Serial.println("GPIO6 Status: ON");

server.send(200, "text/html", SendHTML(LED1status,true));

}

void handle\_led2off() {

LED2status = LOW;

Serial.println("GPIO6 Status: OFF");

server.send(200, "text/html", SendHTML(LED1status,false));

}

void handle\_NotFound(){

server.send(404, "text/plain", "Not found");

}

Displaying the HTML Web Page

SendHTML() function is responsible for generating a web page whenever the ESP8266 web server gets a request from a web client. It merely concatenates HTML code into a big string and returns to the server.send() function. The function takes status of LEDs as a parameter to dynamically generate the HTML content.

The first text you should always send is the <!DOCTYPE> declaration that indicates that we’re sending HTML code.

String SendHTML(uint8\_t led1stat,uint8\_t led2stat){

String ptr = "<!DOCTYPE html> <html>\n";

Next, the <meta> viewport element makes the web page responsive in any web browser. While title tag sets the title of the page.

ptr +="<head>

<meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";

ptr +="<title>LED Control</title>\n";

Styling the Web Page

Next, we have some CSS to style the buttons and the web page appearance. We choose the Helvetica font, define the content to be displayed as an inline-block and aligned at the center.

ptr +="<style>

html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";

Following code then sets color, font and margin around the body, H1, H3 and p tags.

ptr +="body{margin-top: 50px;}

h1 {color: #444444;margin: 50px auto 30px;}

h3 {color: #444444;margin-bottom: 50px;}\n";

ptr +="p {font-size: 14px;color: #888;margin-bottom: 10px;}\n";}

Some styling is applied to the buttons as well with properties like color, size, margin, etc. The ON and OFF button has different background color while :active selector for buttons ensure button click effect

ptr +=".button {display: block;width: 80px;

background-color: #1abc9c;border: none;

color: white;

padding: 13px 30px;

text-decoration: none;font-size: 25px;

margin: 0px auto 35px;

cursor: pointer;

border-radius: 4px;

}\n";

ptr +=".button-on {background-color: #1abc9c;}\n";

ptr +=".button-on:active {background-color: #16a085;}\n";

ptr +=".button-off {background-color: #34495e;}\n";

ptr +=".button-off:active {background-color: #2c3e50;}\n";

Setting the Web Page Heading

Next, heading of the web page is set; you can change this text to anything that suits your application.

ptr +="<h1>ESP8266 Web Server</h1>\n";

ptr +="<h3>Using Access Point(AP) Mode</h3>\n";

Displaying the Buttons and Corresponding State

To dynamically generate the buttons and LED status, we use if statement. So, depending upon the status of the GPIO pins, ON/OFF button is displayed.

if (led1stat) {

ptr +="<p>LED1 Status: ON</p>

<a class=\"button button-off\"

href=\"/led1off\">OFF</a>\n";

} else {

ptr +="<p>LED1 Status: OFF</p>

<a class=\"button button-on\" href=\"/led1on\">ON</a>\n";

}

if (led2stat) {

ptr +="<p>LED2 Status: ON</p>

<a class=\"button button-off\"

href=\"/led2off\">OFF</a>\n";

} else {

ptr +="<p>LED2 Status: OFF</p>

<a class=\"button button-on\"

href=\

etc