ESP8266 Web Server

Step By Step

by Alexander Chukhryaev

1st edition

INDEX

Disclaimer

This eBook has been written for information purposes only. Every effort has been made to make this eBook as complete and accurate as possible.

The purpose of this eBook is to educate. It's a first edition of this book.

The author (Alexander Chukhryaev) does not warrant that the information contained in this eBook is fully complete and shall not be responsible for any errors or omissions.

The author (Alexander Chukhryaev) shall have neither liability nor responsibility to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by this eBook.

This eBook contains code examples which you can use on your own projects, excepted where otherwise noted.

You cannot redistribute this eBook.

This eBook is only available for free download at:
http://acoptex.com/wp/download

Please send an email to the author (Alexander Chukhryaev - info@acoptex.com), if you find this eBook anywhere else.

Introduction

This eBook will help you to build a web server with the ESP8266 WiFi module.

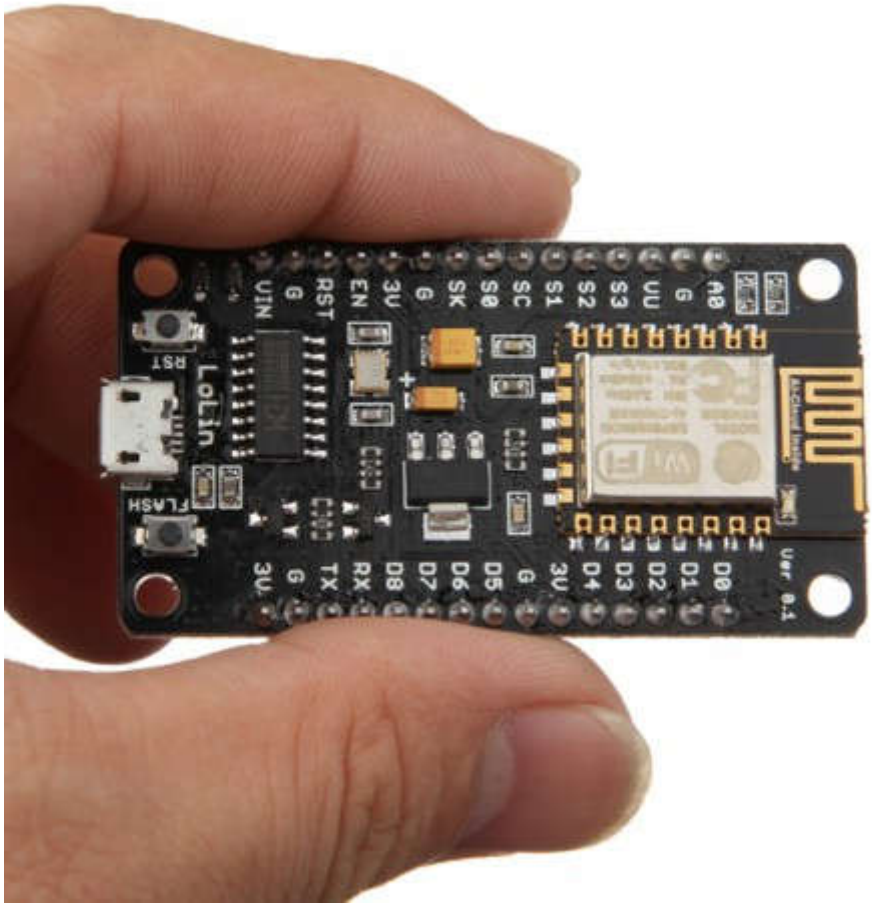


Image credit : Acoptex

Whether you are building a robot or working with Arduino, knowing how to use ESP8266 WiFi module will come in handy.

Have fun with your projects. Thank you for reading,
P.S. Make sure you visit our website to see the latest projects!
http://acoptex.com

Connect with Acoptex.Com

If you have any questions, please don't hesitate to contact us.

Here are some ways to stay in touch.

Visit our website
(http://acoptex.com)

Subscribe on YouTube
(https://www.youtube.com/channel/UCPQwh8eONWeNUeZ24CqDR1g/)

Like on facebook
(https://www.facebook.com/acoptexco/)

Follow on Twitter
(https://twitter.com/Acoptexcom/)

Check us on GitHub
(https://github.com/AcoptexCom/)

Follow us on Instagram
(https://www.instagram.com/acoptexcom/)

Understanding The ESP8266 WiFi Module

The ESP8266 WiFi Module is a self contained SOC (System On a Chip) with integrated TCP/IP protocol stack that can give any microcontroller access to your WiFi network.

The ESP8266 WiFi Module is capable of either hosting an application or offloading all Wi-Fi networking functions from another application processor.

Each ESP8266 WiFi Module comes pre-programmed with an AT command set firmware, meaning, you can simply hook this up to your Arduino board and get about as much WiFi-ability as a WiFi Shield offers.

The ESP8266 WiFi Module is an extremely cost effective board. This module has a powerful enough on-board processing and storage capability that allows it to be integrated with the sensors and other application specific devices through its GPIOs (General Purpose Input Output) with minimal development up-front and minimal loading during runtime. Its high degree of on-chip integration allows for minimal external circuitry, including the front-end module, is designed to occupy minimal PCB area.

The ESP8266 WiFi Module supports APSD for VoIP applications and Bluetooth co-existance interfaces, it contains a self-calibrated RF allowing it to work under all operating conditions, and requires no external RF parts.

In short, the ESP8266 WiFi Module is a TTL "Serial to Wireless Internet" device, a small microprocessor with built in wi-fi. It is faster than most Arduino boards and has more memory than most Arduino boards and has less pins than an Arduino board.

The ESP8266 comes in a wide variety of versions (as shown on the picture below). The ESP-12E NodeMCU or ESP-12E is the most practical version and this module we will use in this ebook. You can find this ESP8266 WiFi module here.

Image credit: Acoptex

ESP8266 specifications:

- 802.11 b/g/n protocol
- Wi-Fi Direct (P2P), soft-AP
- Integrated TCP/IP protocol stack
- Built-in low-power 32-bit CPU
- SDIO 2.0, SPI, UART

So, what can you do with this module? You can:

- create a web server
- send HTTP requests
- control outputs
- read inputs and interrupts
- send emails
- post tweets
- build IoT gadgets
- and so on.

This project was tested with ESP-01, ESP-07, ESP-12 and ESP-12E. So you can follow this ebook with any of those boards.

ESP8266 ESP-12E NodeMCU WiFi Module

NodeMCU is, at the moment, the most popular alternative firmware that runs on the ESP8266. Based on the eLua project, it runs a Lua interpreter onboard the ESP8266, which is able to execute commands written in the Lua scripting language. The commands are sent to the ESP8266 via the Serial UART interface.

NodeMCU is a great starting point for Makers as it provides an interactive environment that allows running commands not only for controlling the ESP8266's wireless interface, but also its GPIO and hardware functionality such as PWM. In addition, we have access to the full scope of the Lua programming language for writing our applications. In the case of the default firmware (AT+Commands Interpreter), the application code would have to be developed using a programming language suited to the host microcontroller or SoC that would be sending the commands over Serial (e.g., C/C++ for the Atmel/ARM microcontrollers on Arduino boards).

Not only does the NodeMCU firmware allows us to execute commands interactively, but we can save our applications as a script in the ESP-12E's flash memory, and instruct it to run the application code every time it restarts!

ESP8266 ESP-12E WiFi module has three operational modes:

1. Access Point (AP) — In AP, the Wi-Fi module acts as a Wi-Fi network, or access point (hence the name). It allows other devices to connect to it. And establishes a two-way communication between the ESP8266 and the device that is connected to it via Wi-Fi.

2. Station (STA) — In STA mode, the ESP-12E can connect to an AP (access point) such as the Wi-Fi network from your house. This allows any device connected to that network to communicate with the module.

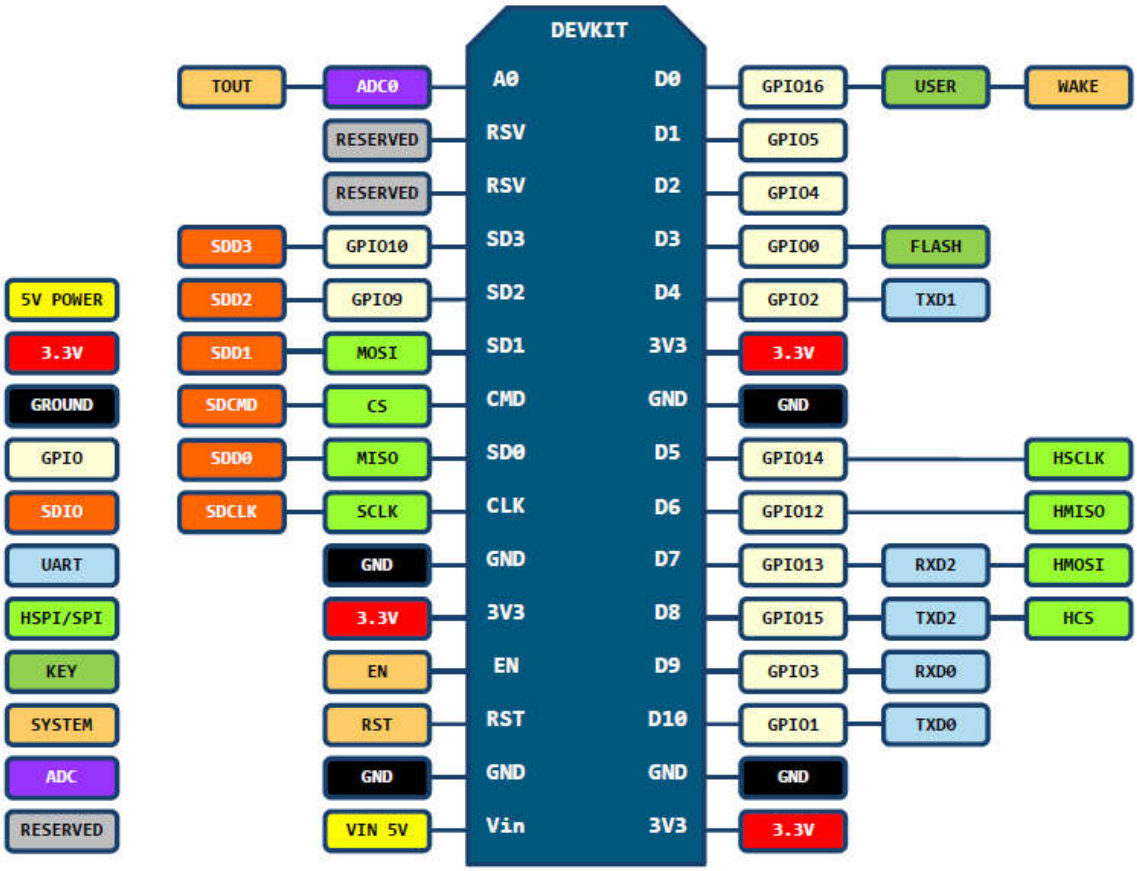
3. Both — In this mode ESP-012E act as both an AP as well as in STA mode. Refer to the following site for more ESP8266 AT commands.

Disadvantages of the NodeMcu module

The main disadvantage is the ability to execute only LUA scripts located in the RAM. This type of memory is small, the volume is only 20 KB, so writing large scripts causes a number of difficulties. First of all, the whole algorithm will have to be divided into linear blocks. These blocks must be written to separate files on the system. All these modules are executed using the dofile operator.

At writing it is necessary to observe a rule - at data interchange between modules it is necessary to use global variables, and at calculation inside of modules - local. It is also important to call the collectgarbage (garbage collector) function at the end of each written script.

ESP8266 ESP-12E NodeMCU WiFi Module Pinout



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

TX - transmit pin. GPIO pin
RX - receive pin. GPIO pin
3V3 (or 3V or 3.3V) - power supply pin (3-3.6V).
GND (or G) - ground pin.
RST - reset pin. Keep it on high (3.3V) for normal operation. Put it on 0V to reset the chip.
EN - Chip enable. Keep it on high (3.3V) for normal operation.
Vin - External power supply 5VDC.
D0-D8 - GPIO (General Purpose Input Output) pins
D5-D8 - SPI interface
D1-D2– I²C/TWI Interface
SC (or CMD) - (Chip Select) - the pin that the master can use to enable and disable specific devices. GPIO pin

SO (or SDO) - Master In Slave Out (MISO) - SPI communication. The Slave line for sending data to the master. GPIO pin
SK (or CLK) - **SCK (Serial Clock)** - SPI communication.The clock pulses which synchronize data transmission generated by the master. GPIO pin
S1 (or SD1) - Master Out/Slave In (MOSI). SPI communication. The Master line for sending data to the peripherals. GPIO pin
S2 (or SD2) - GPIO pin
S3 (or SD3) - GPIO pin
VU (or VUSB) - external power 5VDC.
A0 - ADC output.
RSV - reserved

Arduino IDE

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software. This software can be used with any Arduino board, ESP8266 WiFi module, ESP32 WiFi module. The Arduino IDE is a multiplatform software, which means that it runs on Windows, Mac OS X or Linux (it was created in JAVA). First you need to download, install and prepare your Arduino IDE to work with the ESP8266 WiFi module. Then you can program your ESP8266 WiFi module using the simple C programming language.

Preparations:

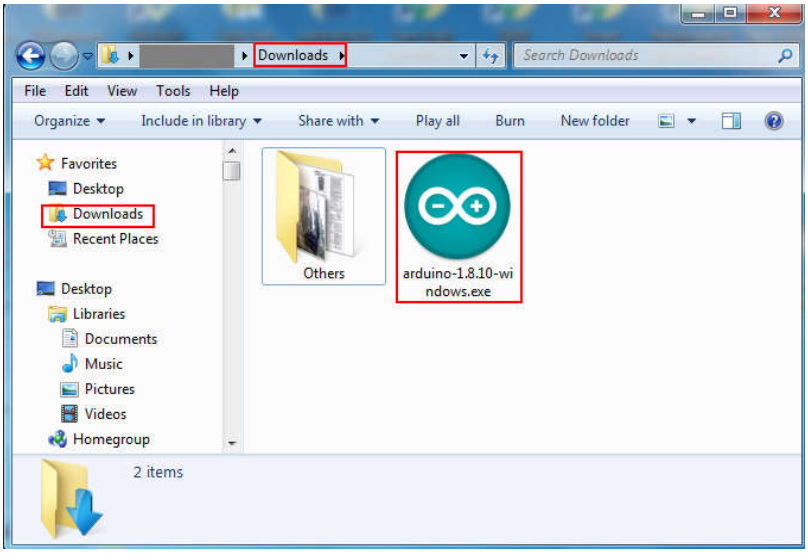
You need to have the JAVA installed in your computer (PC). If you do not have it, go to this website: https://www.java.com/en/download/, download and install the latest version.

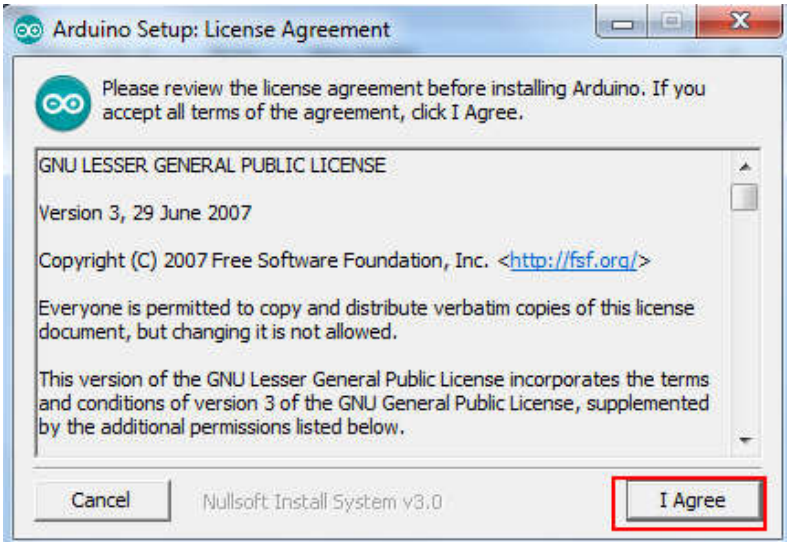
Downloading Arduino IDE:

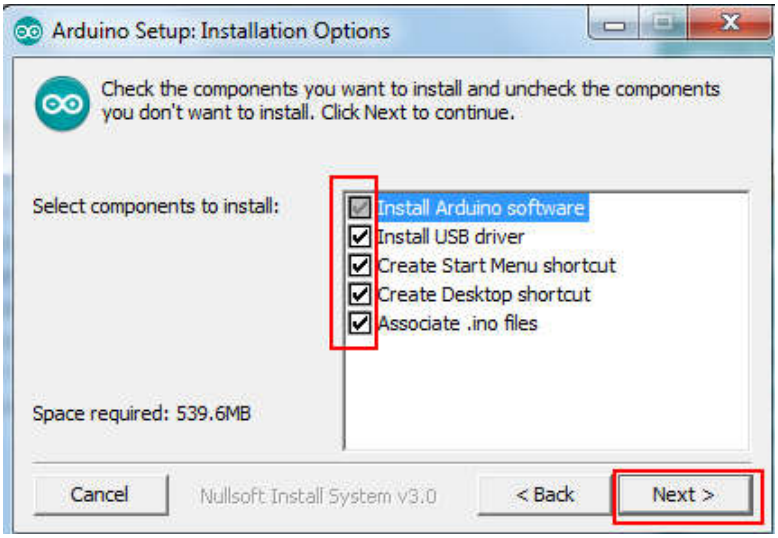
Go to arduino.cc webpage, select your operating system (OS) and download the Arduino IDE

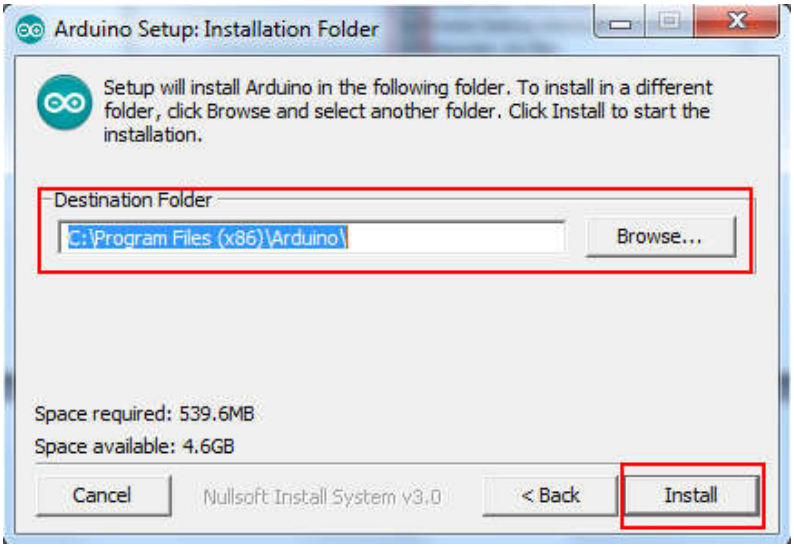
Installing Arduino IDE

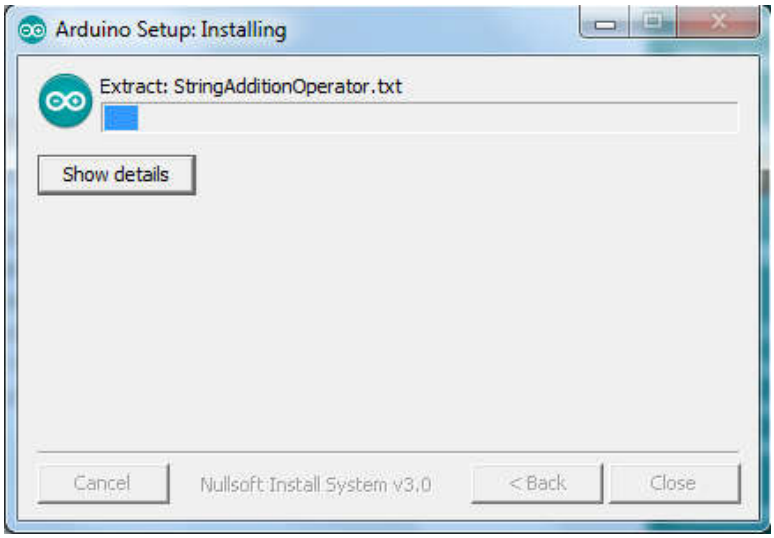
Go to your PC **Download** folder and double-click on file named "**arduino-(...).exe**". Follow the installation wizard that shows on your PC screen.

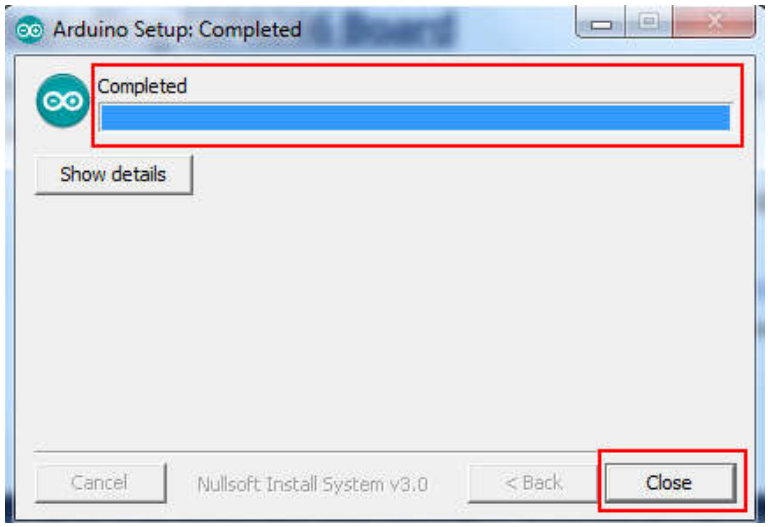










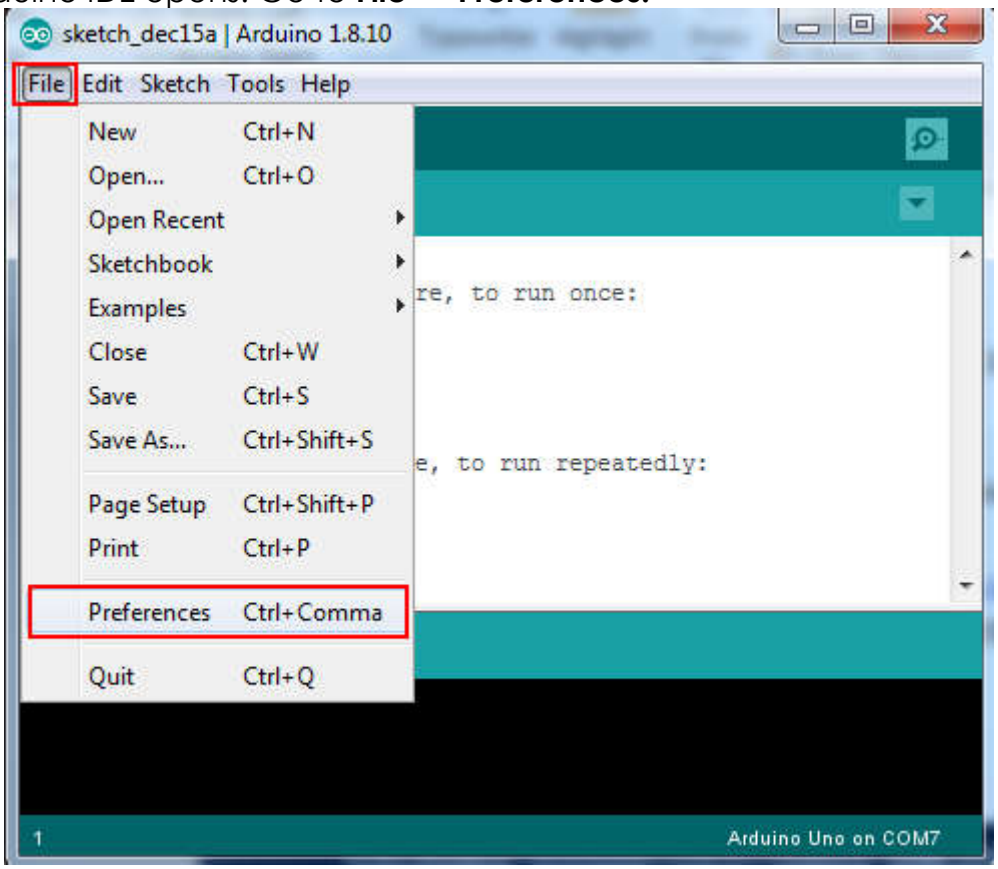


Congrats! You have installed Arduino IDE to your PC now. The Arduino environment has to be set up to make it compatible with the ESP8266 ESP-12E module. It is required to have Arduino IDE version 1.6.4 or higher in order to install the ESP8266's platform packages.

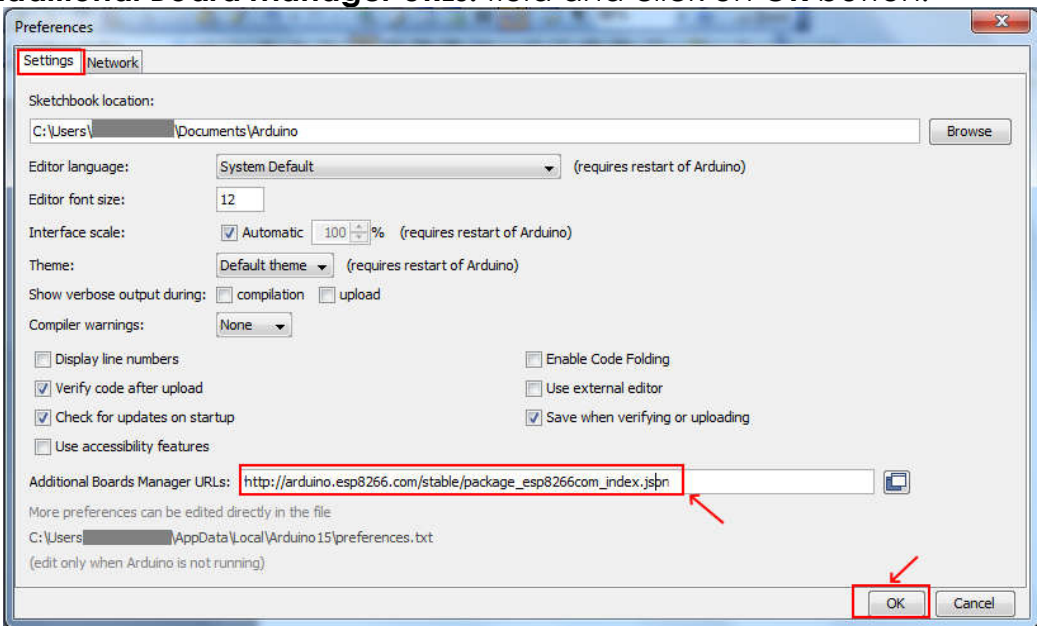
Installing ESP8266's platform packages

Double-click on Arduino shortcut, located on your PC desktop.

The Arduino IDE opens. Go to **File -> Preferences.**

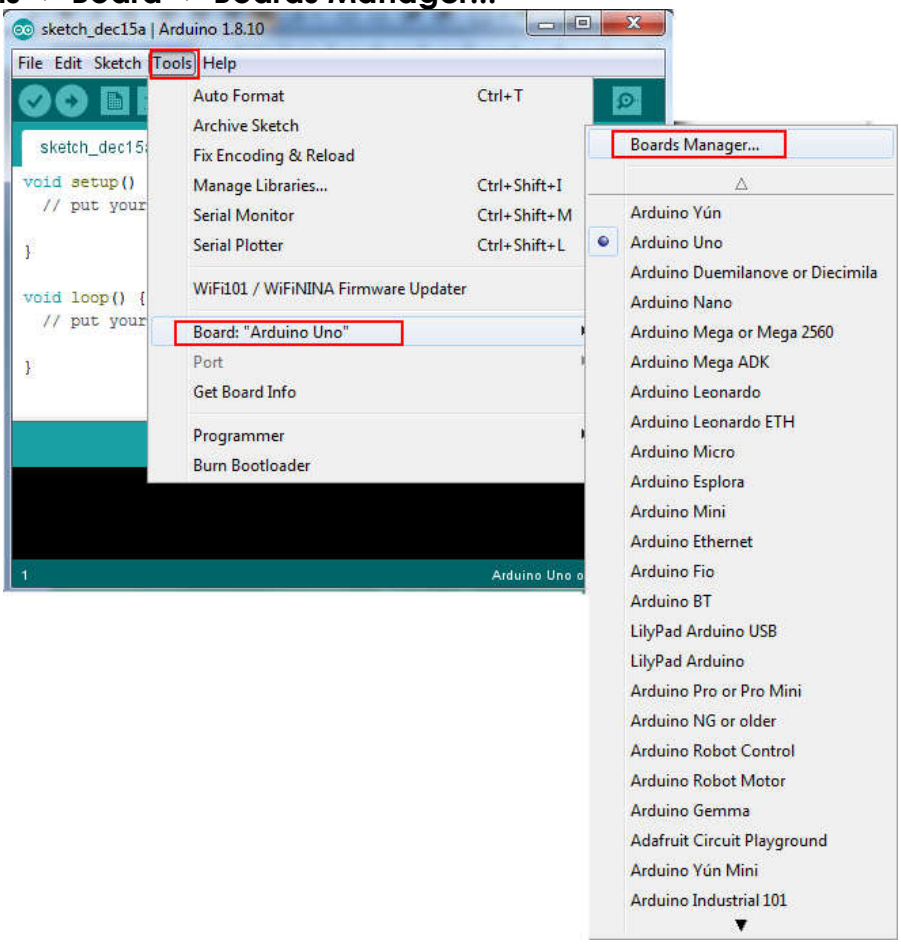


Type http://arduino.esp8266.com/stable/package_esp8266com_index.json into **Additional Board Manager URLs:** field and click on **OK** button.

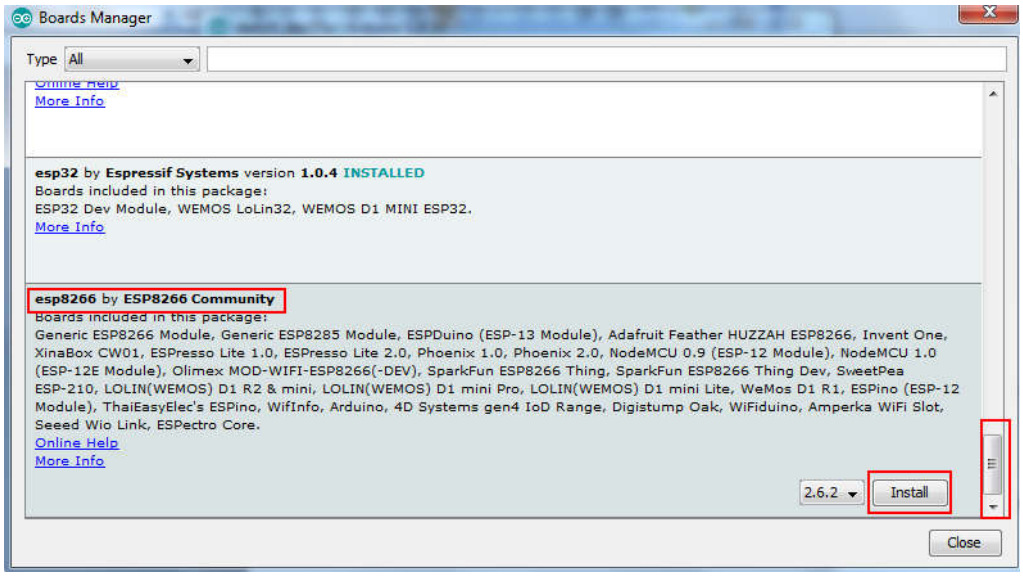


If you already have a URL in there, and want to keep it, you can separate multiple URLs by placing a comma between them. (Arduino IDE 1.6.5 added an expanded text box, separate links in here by line.)

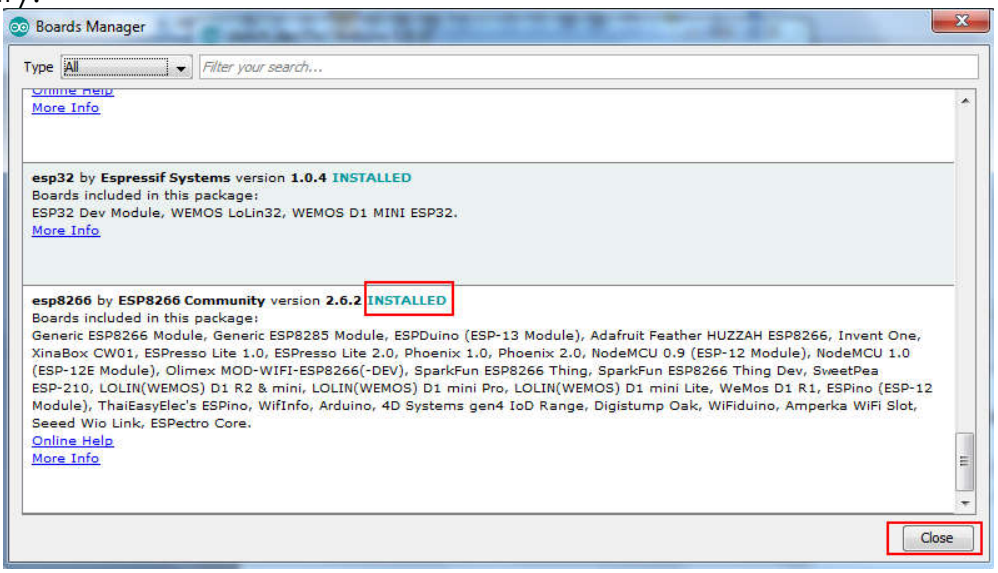
Go to **Tools -> Board -> Boards Manager...**



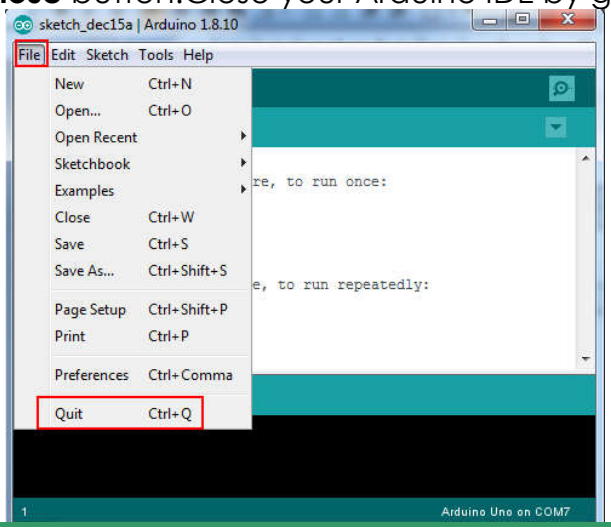
Scroll down, select **esp8266 by ESP8266 Community** and click on **Install** button.



The board definitions and tools for the ESP8266 include a whole new set of gcc, g++, and other reasonably large, compiled binaries, so it may take a few minutes to download and install (the archived file is ~110MB). Once the installation has completed, an Arduino-blue "INSTALLED" will appear next to the entry.



Congrats! You have downloaded, installed Arduino IDE and prepared it for ESP8266. Click on **Close** button.Close your Arduino IDE by goining to **File->Quit**



Do Wiring

Let's build a standalone ESP8266 ESP-12E Web Server that controls two outputs (in our case two LEDs). You can then replace LEDs with any other electronic appliances.

This ESP8266 ESP-12E Web Server can be accessed with any device (smartphone, tablet, PC) through any web browser on your local network.

You need the following parts to build your circuit:

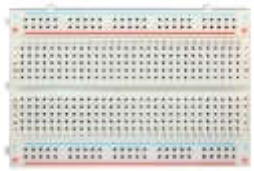
ESP8266 ESP-12E WiFi module 1 pc

LED 2 pcs

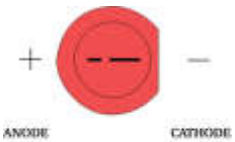
Resistor 2 pcs (220 or 330 Ohms)

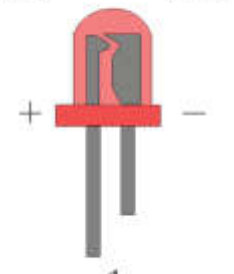
Jumper cables (wires) female to male 3 pcs

Breadboard medium size 1 pc



Micro USB cable 1 pc





Please note: LED longer leg is positive (anode), the shorter leg is negative (cathode).

1. Connect green LED anode pin to GPIO 4 pin, red LED anode pin to GPIO 5 pin of your ESP8266 ESP-12E WiFi module.
2. Connect green LED and red LED cathode pins through resistors to ground (G) pin of your ESP8266 ESP-12E WiFi module.

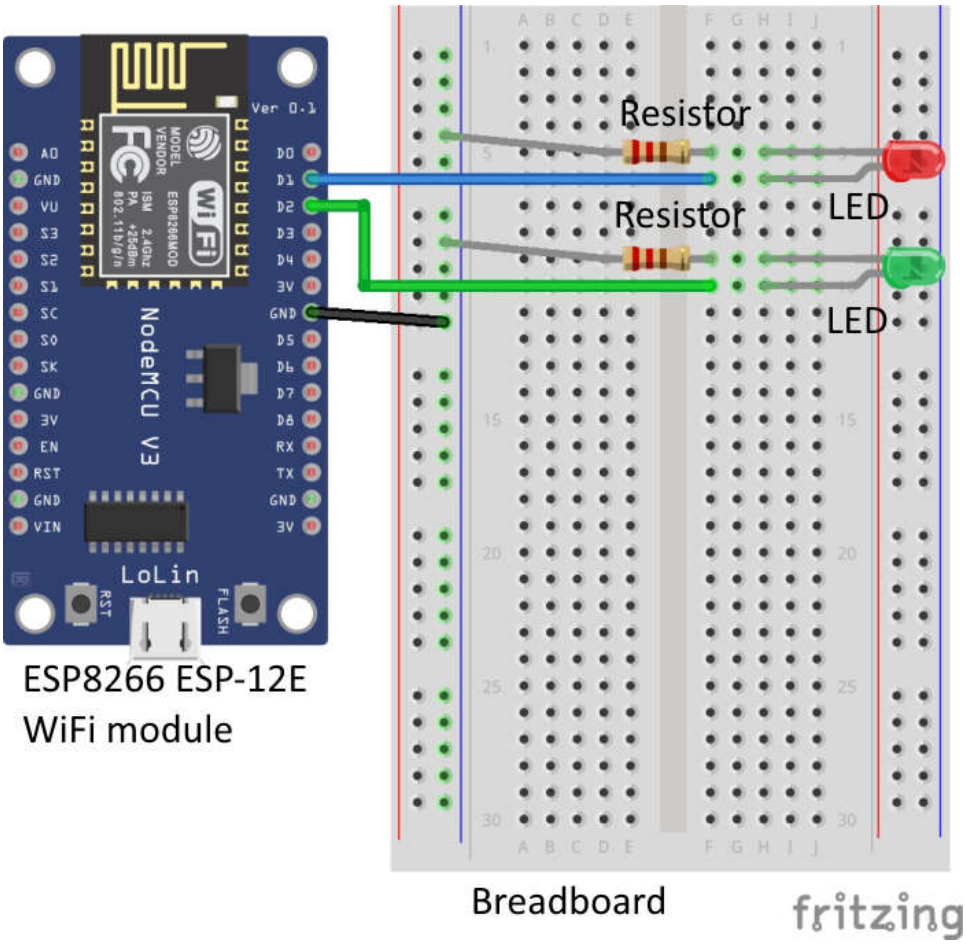


Image credit: Fritzing

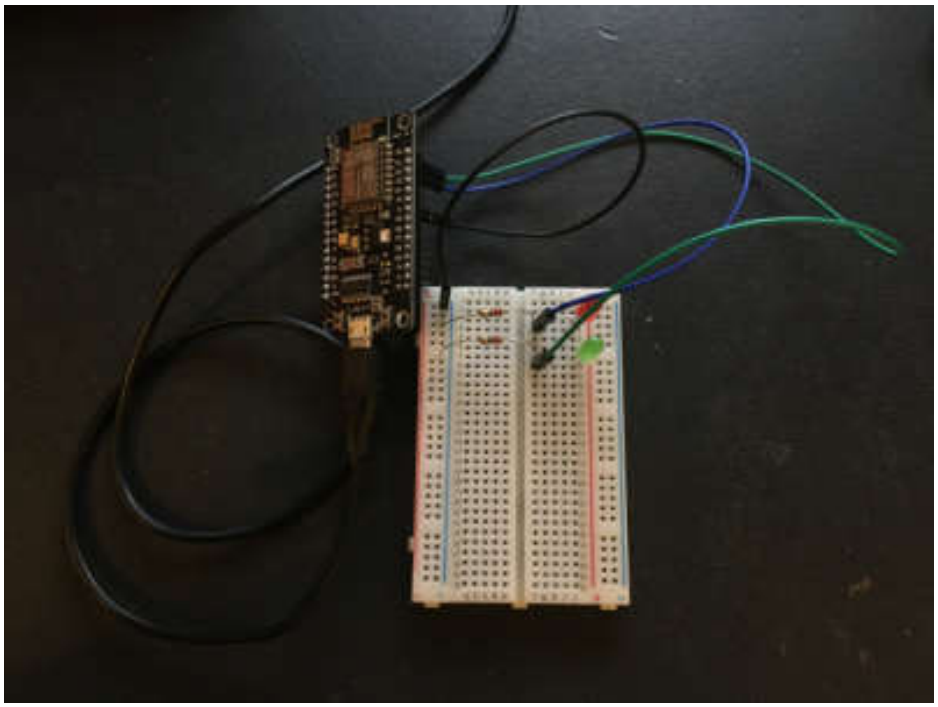


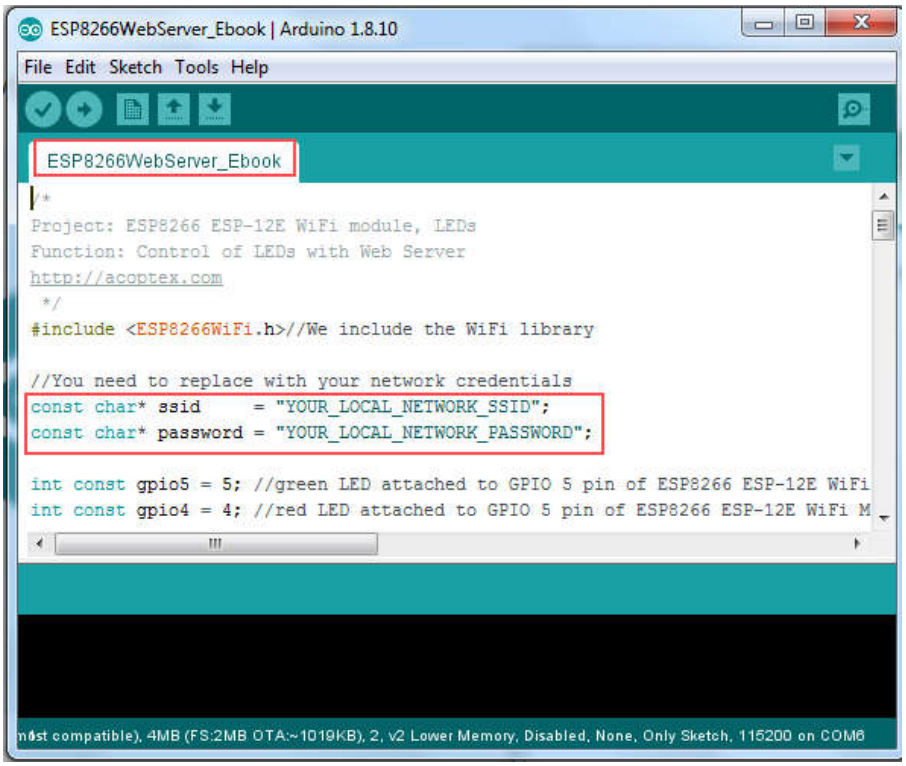
Image credit: Acoptex

Uploading The Sketch

If you are using an ESP-12E NodeMCU Kit, uploading the sketch is very simple, since it has built-in programmer.

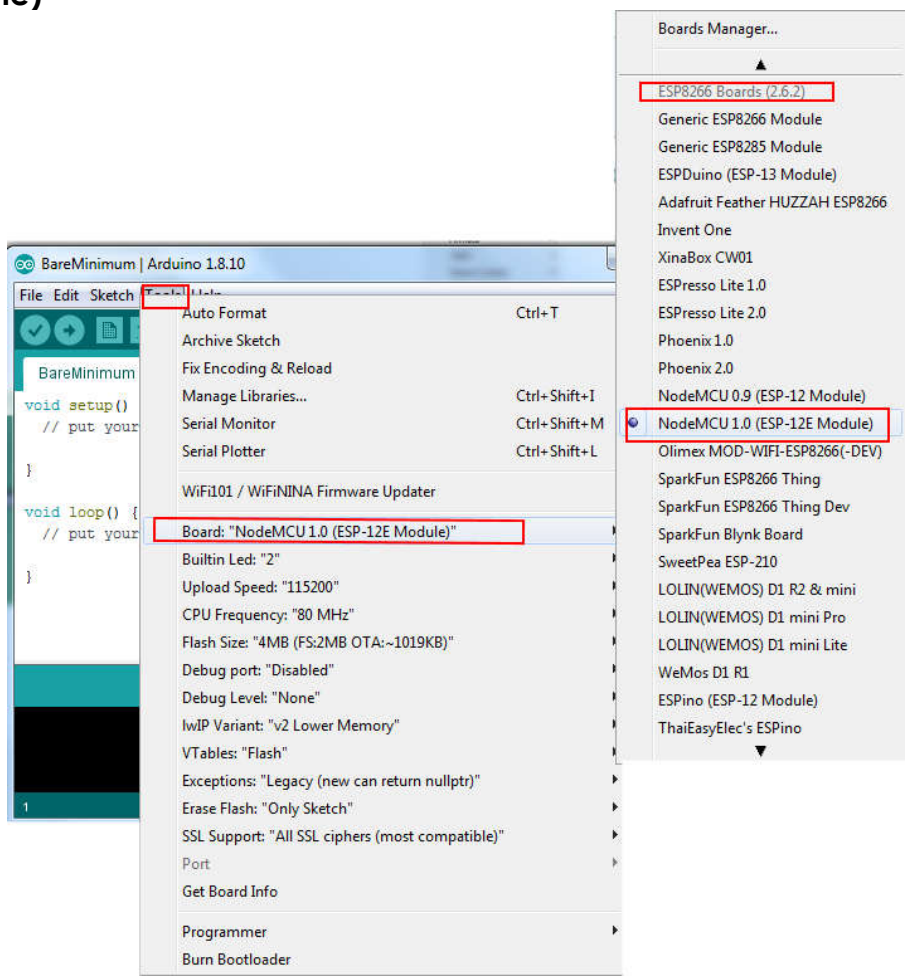
Before use ESP8266 ESP-12E WiFi module (LoLin NODEMCU V3, you need to download the manufacture's driver (CH340) for this chip and install it in your PC - http://www.wch.cn/download/CH341SER_EXE.html . See the description of driver installation package below: CH340/CH341 USB to serial WINDOWS driver installation package that supports 32/64 bit Windows 10 / 8.1 / 8/7 / VISTA / XP, SERVER 2016/2012/2008/2003, 2000 / ME / 98, through Microsoft digital signature authentication, support USB to 3-wire and 9-wire serial port, with the product release To the end user. Applicable scope: CH340G, CH340C, CH340B, CH340E, CH340T, CH340R, CH341A, CH341T, CH341H chips.
If you have CP2102 chip then you need to download the manufacture's driver for this chip and install it in your PC. You can download them here: https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

1. Plug your ESP8266 ESP-12E WiFi module into your PC USB port.
2. Re-open your Arduino IDE.
3.Go to GitHub and download the sketch:
https://github.com/AcoptexCom/Ebooks/blob/master/ESP8266WebServer_Ebook/ESP8266WebServer_Ebook.ino
4. Open the sketch in Arduino IDE editor.

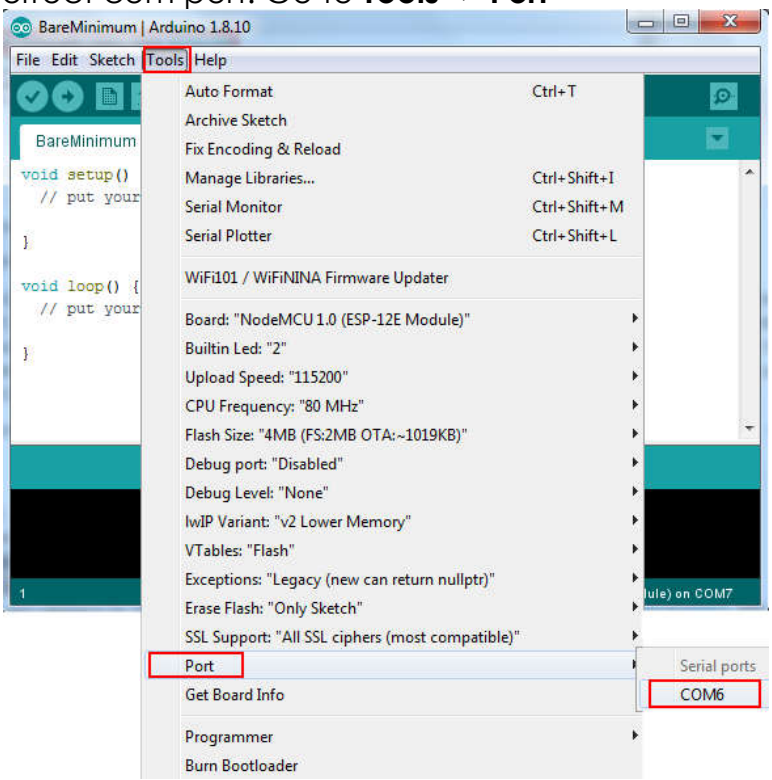


Before uploading the sketch you need to modify the code - type your local network SSID and password.

5. Choose your NodeMCU board. Go to **Tools -> Board -> NodeMCU 1.0 (ESP-12E Module)**

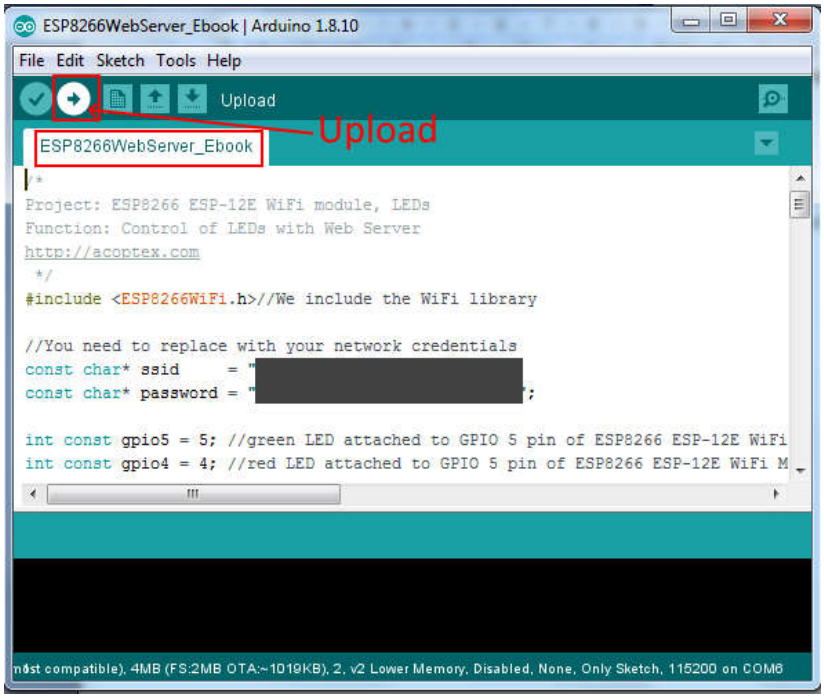


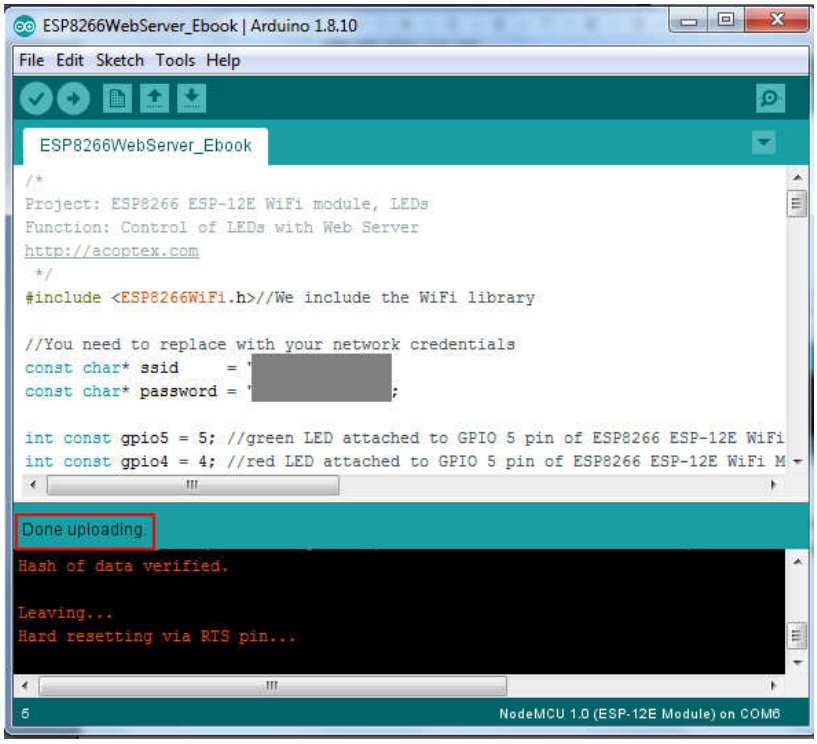
6. Select the correct com port. Go to **Tools -> Port**



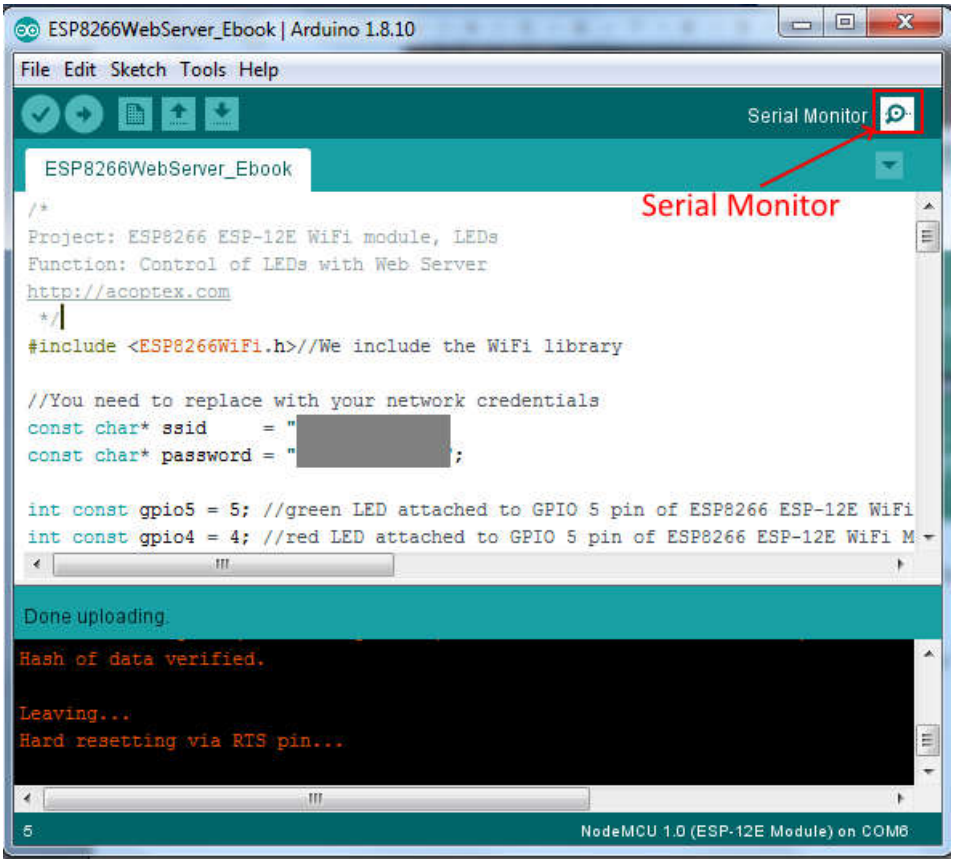
Please note that your COM port is very likely to be different from the preceding screenshot (Port: "COM6"). That is ok, because it doesn't interfere with anything. On the other hand, all the other configurations should look exactly like mine.

7. After checking the configurations, click on **Upload** button in the Arduino IDE and wait a few seconds until you see the message **Done uploading** in the bottom left corner.

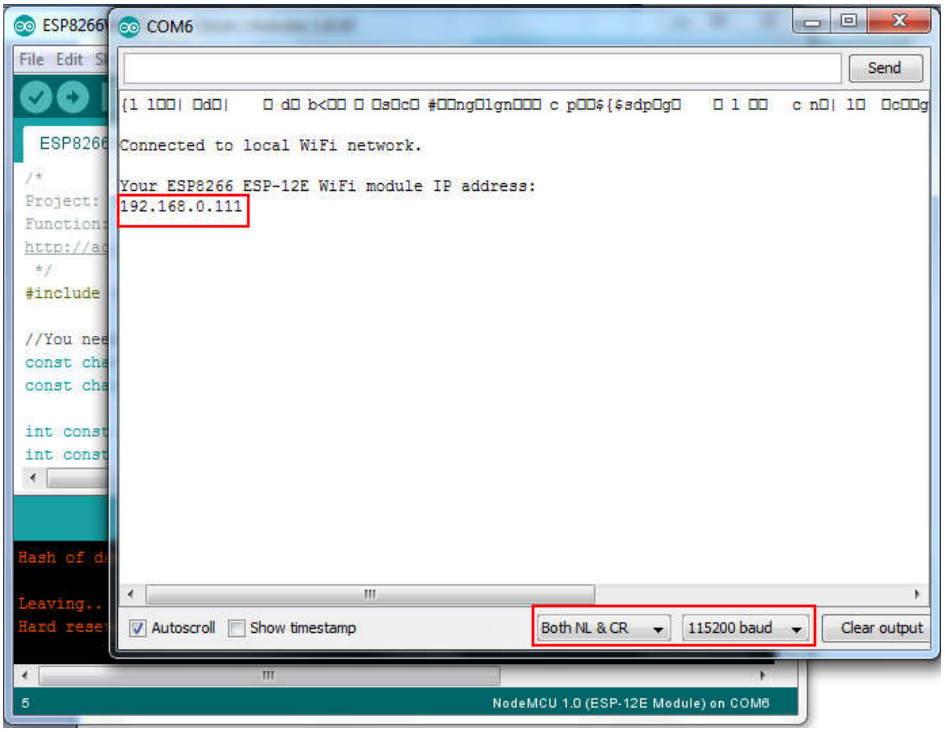




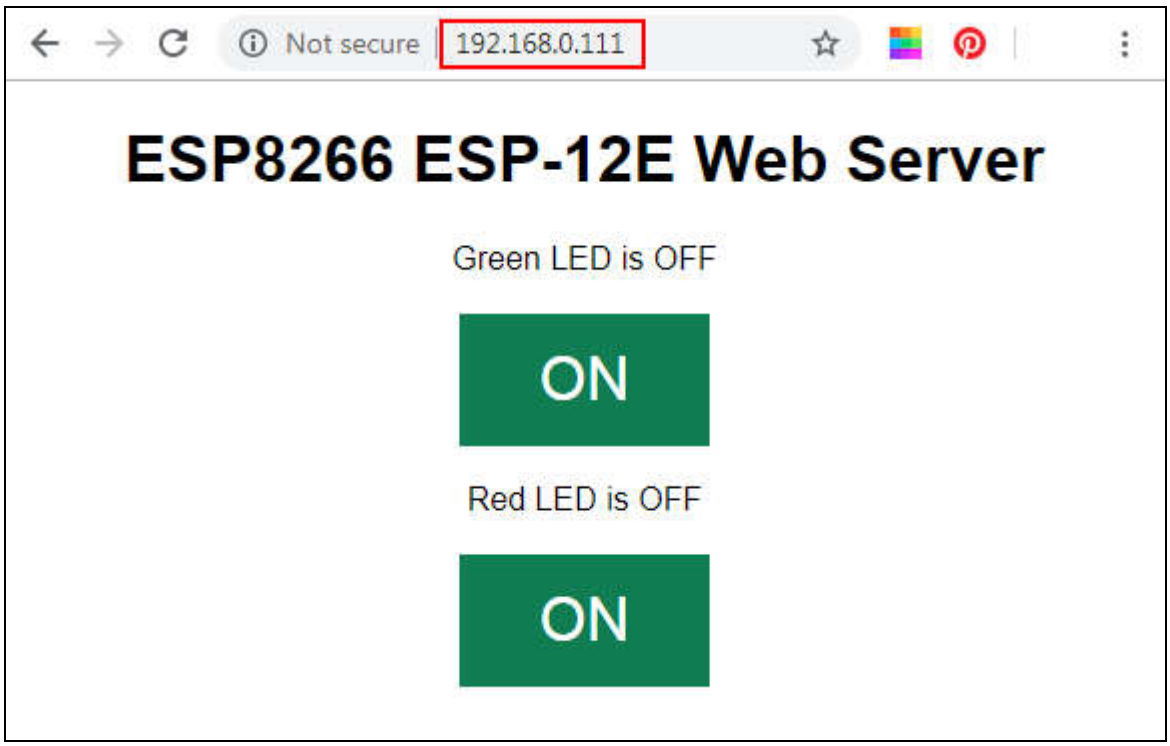
8. Click on **Serial Monitor** button and open Serial Monitor at 115200 bps.



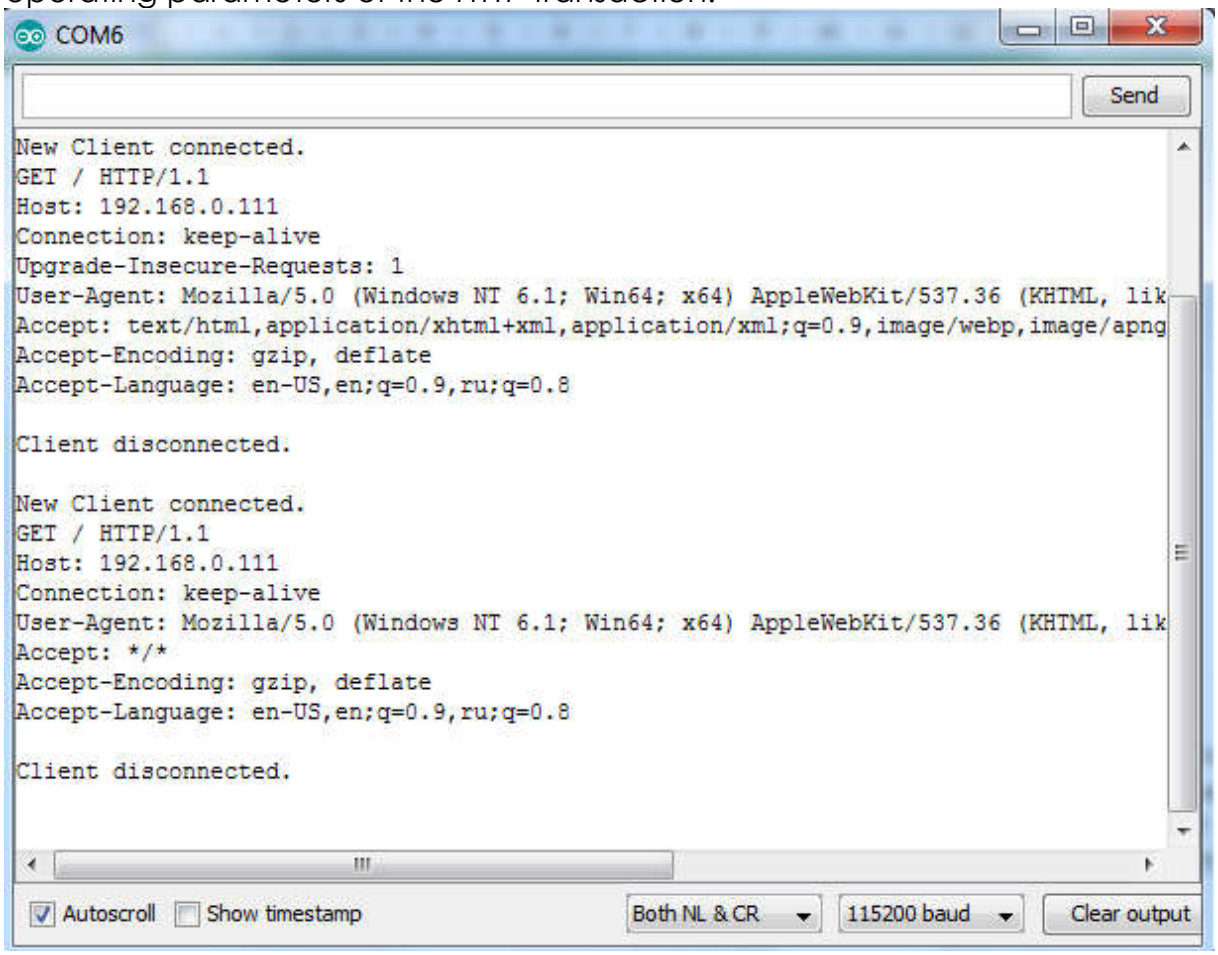
9. Press on-board **RST** button on ESP8266 ESP-12E WiFi module to restart the module. You will see IP address of your ESP8266 ESP-12E WiFi module (for example, I have **192.168.0.111**). Copy this IP address, you will need it to access your Web Server.



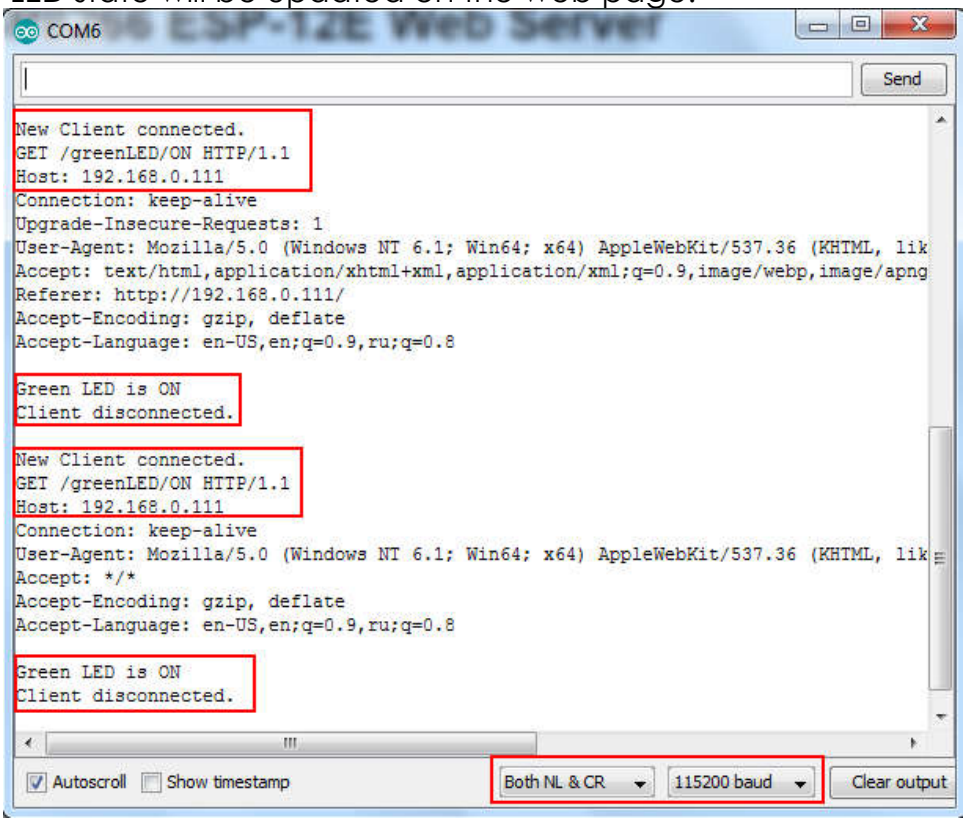
10. Open any web browser (Chrome, Opera, IE...), type the IP address (I have **192.168.0.111**), and you'll see the following page. This page is sent by the ESP8266 ESP-12E WiFi module when you make a request on the IP address.



If you take a look at the Serial Monitor, you can see that the ESP8266 ESP12-E WiFi module receives an HTTP request from a new client (your web browser). You can also see other information - HTTP header fields, which define the operating parameters of the HTTP transaction.



11. Let's click on ON button to turn green LED ON. The ESP8266 ESP12-E WiFi module receives a request on the /greenLED/ON URL, and turns green LED ON. The LED state will be updated on the web page.



12. You can also test red LED button and check that it works same way.

The Code

```
#include <ESP8266WiFi.h>//We include the WiFi library

//You need to replace with your network credentials
const char* ssid     = "YOUR_LOCAL_NETWORK_SSID";
const char* password = "YOUR_LOCAL_NETWORK_PASSWORD";

int const gpio5 = 5; //green LED attached to GPIO 5 pin of ESP8266 ESP-12E WiFi
Module
int const gpio4 = 4; //red LED attached to GPIO 5 pin of ESP8266 ESP-12E WiFi
Module

WiFiServer server(80); //We set web server port number
String header;//We create the string variable to store the HTTP request

//We create variables to store the current output state
String gpio5State = "OFF";
String gpio4State = "OFF";

//Other variables
unsigned long currentTime = millis();
unsigned long previousTime = 0;

const long timeoutTime =2000; //timeout time in milliseconds (2 seconds)

void setup() {//The function only runs once when your ESP8266 ESP12-E boots.
  Serial.begin(115200); //Initialise serial communication at 115200 bps
  pinMode(gpio5, OUTPUT);//We set gpio5 pin as OUTPUT
  pinMode(gpio4, OUTPUT);//We set gpio4 pin as OUTPUT
  digitalWrite(gpio5, LOW);//We set gpio 5 pin to LOW by default
  digitalWrite(gpio4, LOW);//We set gpio 4 pin to LOW by default

  //We connect to local WiFi network
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  //We print in Serial Monitor ESP8266 ESP-12E WiFi module IP address
  Serial.println("");
```

```
  Serial.println("");
  Serial.println("Connected to local WiFi network.");
  Serial.println("");
  Serial.println("Your ESP8266 ESP-12E WiFi module IP address: ");
  Serial.println(WiFi.localIP());

  server.begin();//We start the web server
}

void loop(){//We define what happens when a new client establishes a
connection with the web server.
  WiFiClient client = server.available();   //We listen for incoming clients
/*When a request is received from a client, we will save the incoming data. The
while loop that follows will be running as long as the client stays connected.*/
  if (client) {                             //If a new client connects,
    Serial.println("New Client connected.");//We print a message out in the Serial
Monitor
    String currentLine = "";                //We make a String variable to hold
incoming data from the client
    currentTime = millis();
    previousTime = currentTime;
    while (client.connected() && currentTime - previousTime <= timeoutTime) { //We
have loop while the client's connected
      currentTime = millis();
      if (client.available()) {//If there are bytes to read from the client,
        char c = client.read();//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 two newline characters in a row.
//That's the end of the client HTTP request, so send a response:
          if (currentLine.length() == 0) {
            client.println("HTTP/1.1 200 OK"); //HTTP headers always start with a
response code
            client.println("Content-type:text/html");//and a content-type so the
client knows what's coming,
            client.println("Connection: close");
            client.println();                       //then a blank line
//The next section of if and else statements checks which button was pressed
in your web page, and controls the outputs accordingly
 //We make a request on different URLs depending on the button we click on
            if (header.indexOf("GET /greenLED/ON") >= 0) { //turns the LEDs ON/OFF
```

```
            Serial.println("Green LED is ON");
            gpio5State = "ON";
            digitalWrite(gpio5, HIGH);
          } else if (header.indexOf("GET /greenLED/OFF") >= 0) {
            Serial.println("Green LED is OFF");
            gpio5State = "OFF";
            digitalWrite(gpio5, LOW);
          } else if (header.indexOf("GET /redLED/ON") >= 0) {
            Serial.println("Red LED is ON");
            gpio4State = "ON";
            digitalWrite(gpio4, HIGH);
          } else if (header.indexOf("GET /redLED/OFF") >= 0) {
            Serial.println("Red LED is OFF");
            gpio4State = "OFF";
            digitalWrite(gpio4, LOW);
          }
/*For example, if you have pressed the green LED ON button, the URL changes to
the ESP8266 ESP12-E WiFi module IP address followed by /greenLED/ON, and we
receive that information on the HTTP header. If it contains GET /greenLED/ON, the
code prints a message in the Serial Monitor, changes the gpio5State variable to
ON, and turns the LED ON. It is the same for the other buttons. If you want to add
more outputs, you should modify this part of the code to include them*/
 //Then we make the HTML web page
    client.println("<!DOCTYPE html><html>"); //Indicates that we are sending HTML
     client.println("<head><meta name=\"viewport\" content=\"width=device-width,
initial-scale=1\">");//It makes the web page responsive in any web browser
            client.println("<link rel=\"icon\" href=\"data:,\">"); //We prevent
requests related to the favicon
            //CSS style for ON/OFF buttons
            //You can change the background-color, font-size, make border, change
font color if you want so
            client.println("<style>html { font-family: Helvetica; display: inline-
block; margin: 0px auto; text-align: center;}");
            client.println(".button { background-color: #2E7C4F; border: none;
color: white; padding: 16px 40px;");  //ON button CSS style
            client.println("text-decoration: none; font-size: 30px; margin:
2px;}");
            client.println(".button1 {background-color: #FF4C4F; border: none;
color: white; padding: 16px 33px;");  //OFF button CSS style
            client.println("text-decoration: none; font-size: 30px; margin:
2px;}</style></head>");
            client.println("<body><h1>ESP8266 ESP-12E Web Server</h1>");
```

```
//Web page heading
client.println("<p>Green LED is " + gpio5State + "</p>");
//We show the current state, ON/OFF buttons for green LED
if (gpio5State=="ON") {    //If the gpio5State is OFF, it displays the ON button
              client.println("<p><a href=\"/greenLED/OFF\"><button
class=\"button1\">OFF</button></a></p>");

            } else {
              client.println("<p><a href=\"/greenLED/ON\"><button
class=\"button\">ON</button></a></p>");
            }
            client.println("<p>Red LED is " + gpio4State + "</p>");
//We show the current state, ON/OFF buttons for red LED

            if (gpio4State=="ON") {
//If the gpio4State is OFF, it displays the ON button
              client.println("<p><a href=\"/redLED/OFF\"><button
class=\"button1\">OFF</button></a></p>");
            } else {
              client.println("<p><a href=\"/redLED/ON\"><button
class=\"button\">ON</button></a></p>");
            }
            client.println("</body></html>");
            client.println();              //We add the blank line on the end of
the HTTP response

            break;                         //We 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 = "";                           //We clear the header variable
    client.stop();                         //We close the connection
    Serial.println("Client disconnected.");//We print the message in Serial
Monitor
```

```
 Serial.println();
  }
}
```

The Source code is published on GitHub:
https://github.com/AcoptexCom/Ebooks/blob/master/ESP8266WebServer_Ebook/ESP8266WebServer_Ebook.ino