Build a Multisensor Shield for ESP8266

randomnerdtutorials.com/esp8266-multisensor-shield

4 settembre 2018

In this project you'll discover how to design and create a Multisensor Shield for the ESP8266 Wemos D1 Mini board. The shield has temperature sensor (DS18B20), a PIR motion sensor, an LDR, and a terminal to connect a relay module. We'll start by preparing all the hardware and then program it.

Watch the Video Tutorial

This project is available in video format and in written format. You can watch the video below or you can scroll down for the written instructions. This project is divided in two videos.

Design and Build the ESP8266 WeMos D1 Mini Multisensor Shield – Part 1

In this first video we'll decide the hardware that we're going to use. We'll also take a look at this project's main features and how to design and assemble your own WeMos D1 Mini Multisensor shield.

Watch Video At: https://youtu.be/nKFsNM5Q00k

Program and Test the ESP8266 WeMos D1 Mini Multisensor Shield – Part 2

In this second video, we'll program the Wemos D1 Mini Multisensor Shield with a code that runs a web server that allows you to monitor and control the multisensor shield based on several configurable settings.

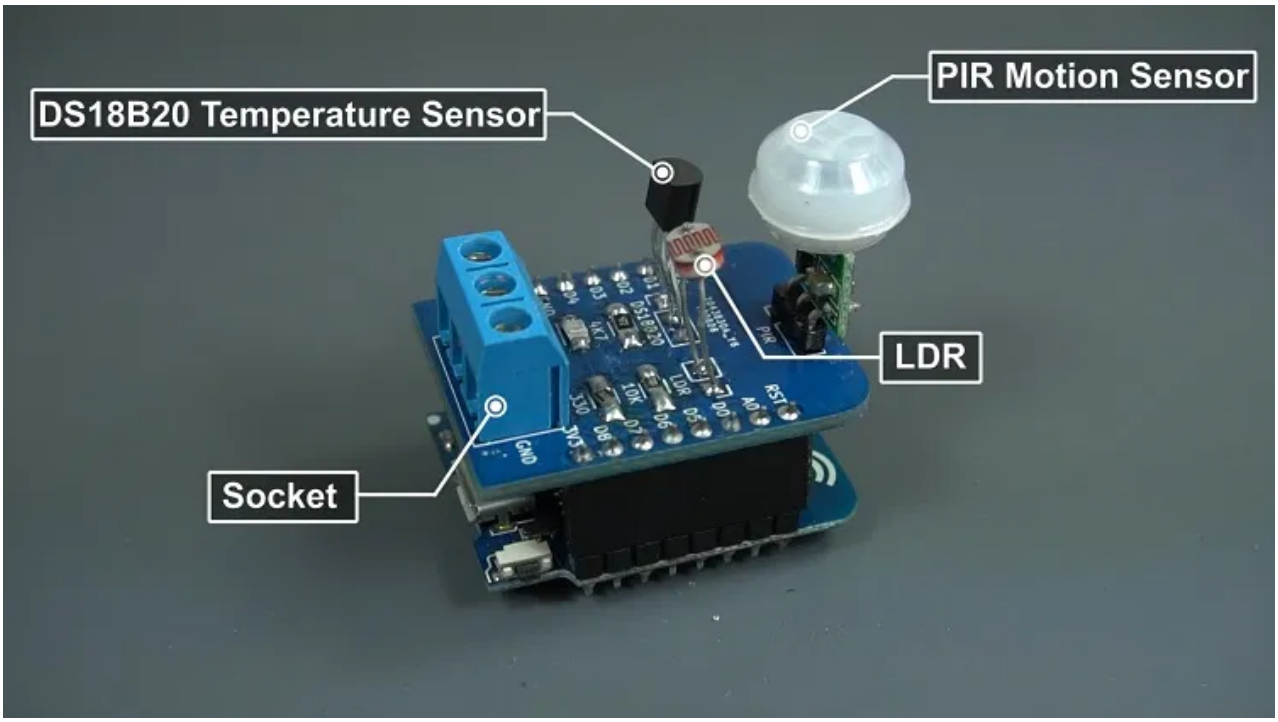
Watch Video At: https://youtu.be/hxRqLIfabqw

Resources

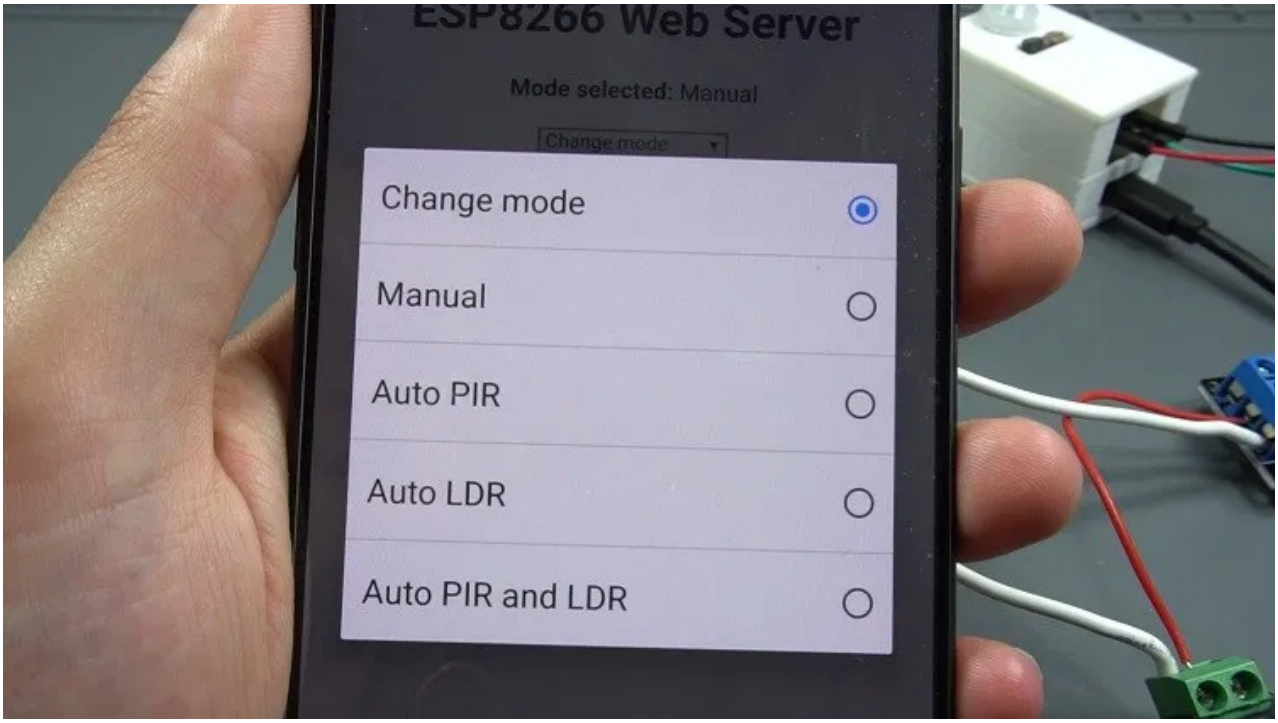
You can find all the resources needed to build this project in the links below (or you can visit the GitHub project):

Project Overview

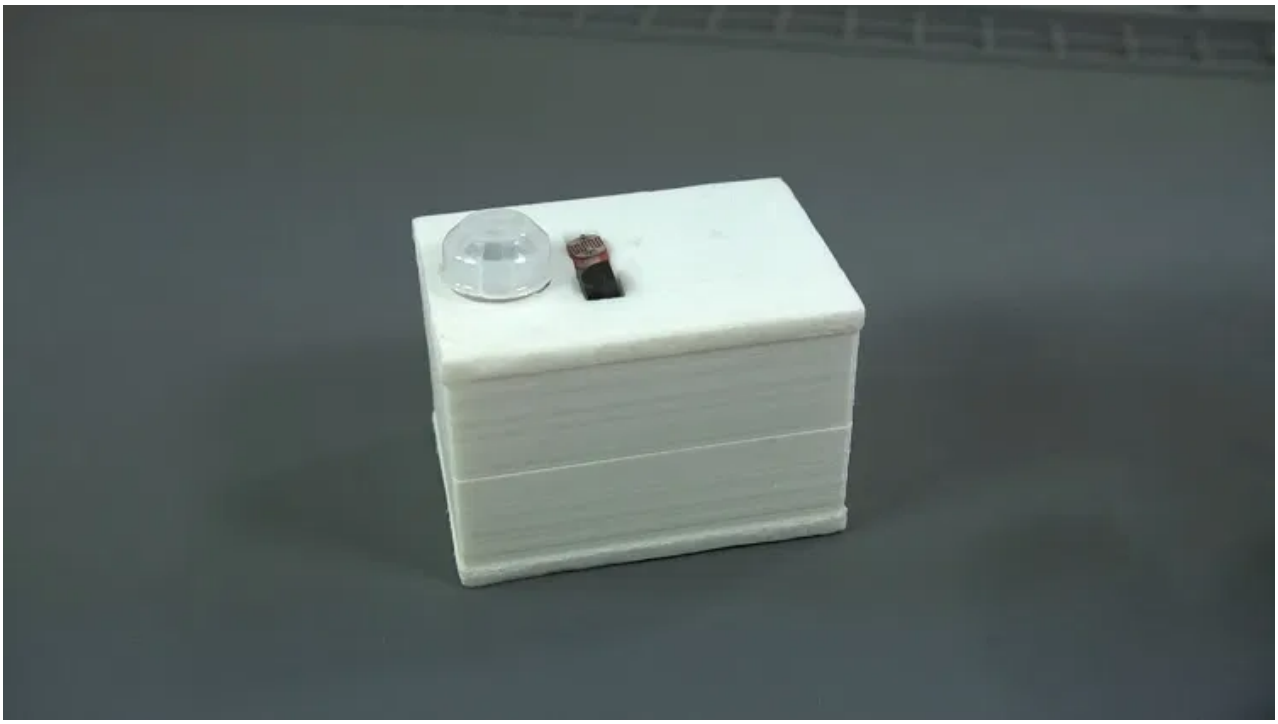
The shield consists of a temperature sensor, a motion sensor, an LDR, and a 3 pin socket where you can connect any output, we'll be using a relay module.



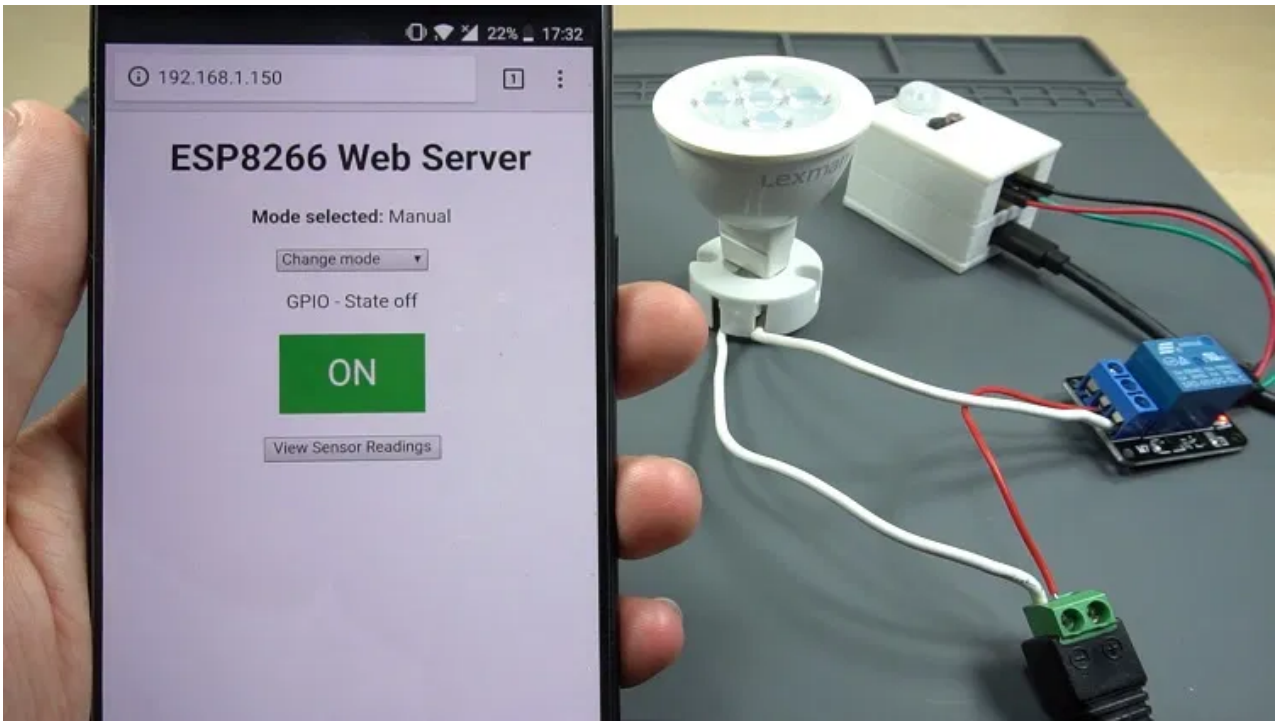
We'll also write a program that runs a web server to control the shield using 4 different modes with several configurable settings.



By the end of this project, you'll have a similar looking gadget.



You'll be able to control the relay on and off in manual mode.



Or you can choose the automatic modes:

- Automatic motion mode, meaning that when it detects motion the relay stays on for a determined number of seconds.
- Or you can use the luminosity mode, so the relay turns on when the light goes below a certain threshold value.

- Finally, there's an option to control the relay based in the current luminosity value and if motion is detected.

We'll show you all these features in action later in this project.

JLCPCB

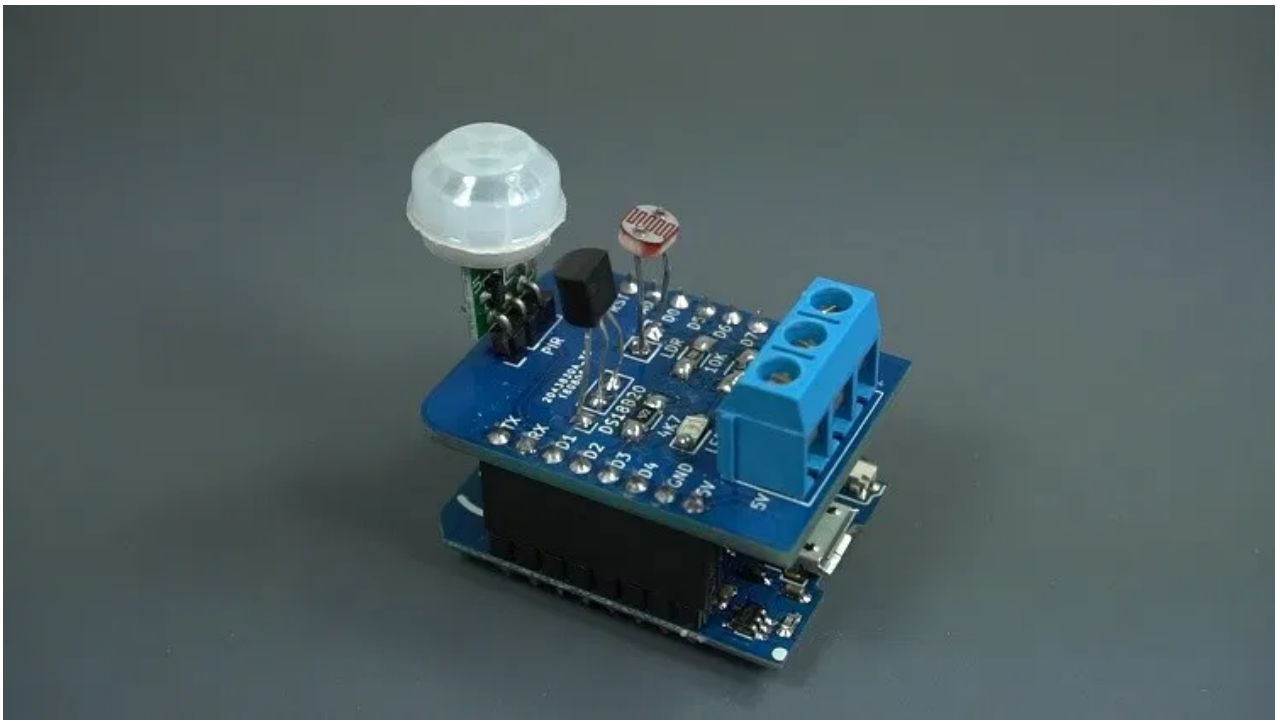
This project was sponsored by JLCPCB. JLCPCB is a well known PCB prototype company in China. It is specialized in quick PCB prototype and small-batch production. You can order a minimum of 10 PCBs for just $2.

If you want to turn your breadboard circuits into real boards and make your projects look more professional, you just have to upload the Gerber files to order high quality PCBs for low prices. We'll show you how to do this later in this blog post.

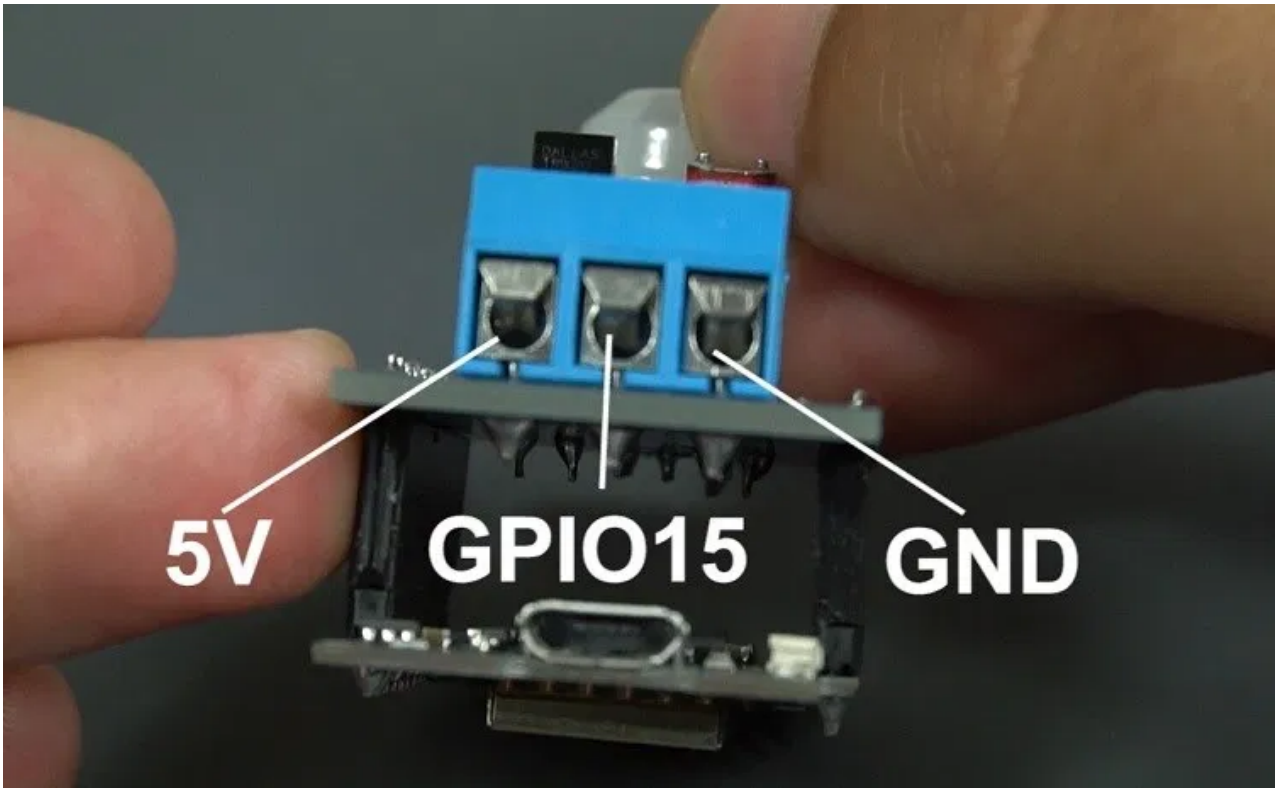
ESP8266 Multisensor Shield Features

The Multisensor Shield features several sensors that can be useful to monitor your house. The shield allows you to control:

- 1x SMD LED to indicate a status
- 1x Light dependent resistor (LDR)
- 1x DS18B20 temperature sensor
- 1x PIR motion sensor



Additionally, it also features a terminal block that gives you access to GND, 5V and GPIO15. That terminal can be used to connect a relay, or any other output you might want to control.



ESP8266 Multisensor Shield Pin Assignment

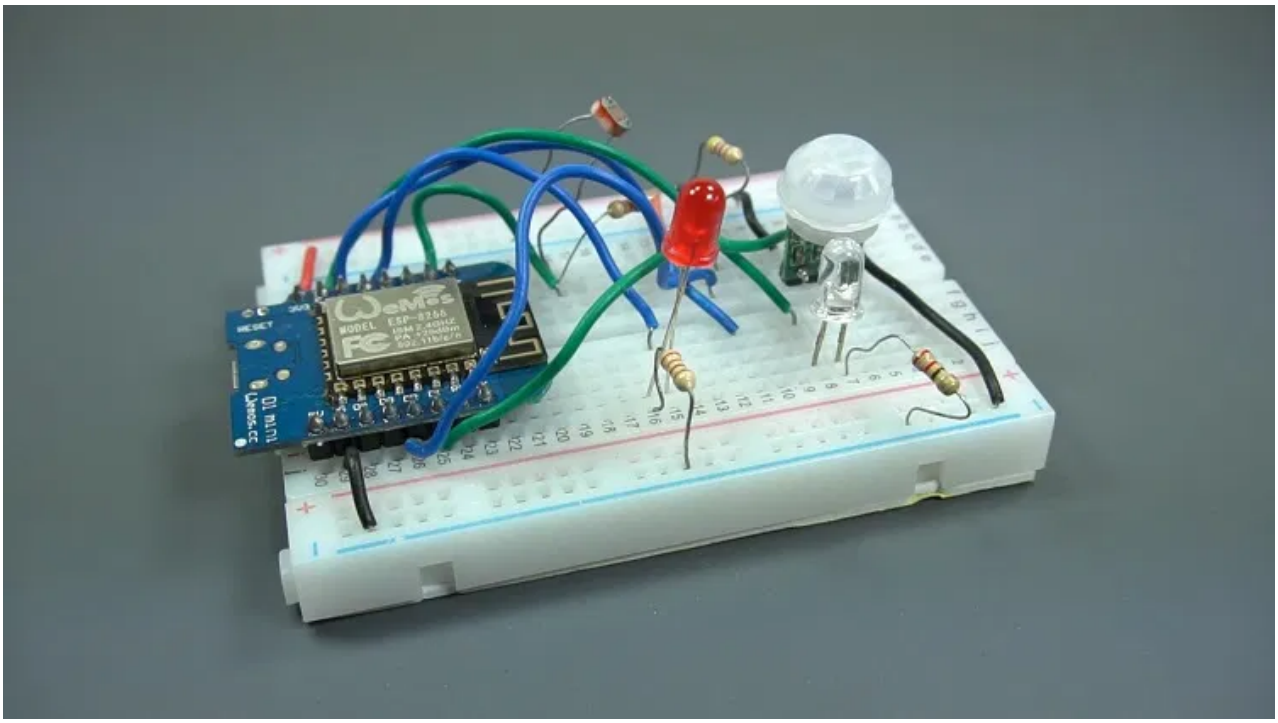
The following table describes the pin assignment for each component of the multisensor shield:

Component	Wemos D1 Mini Pin Assingment
PIR Motion Sensor	GPIO5 (D1)
DS18B20 Temperature Sensor	GPIO4 (D2)
Light Dependent Resistor	A0
LED	GPIO12 (D6)
Additional output	GPIO15 (D8)

If you want to assign and use different pins, read our ESP8266 Pinout Reference Guide.

Testing the Circuit on a Breadboard

Before designing and building the PCB shield, it's important to test the circuit on a breadboard. If you don't want to make a PCB, you can still follow this project by assembling the circuit on a breadboard.



Parts Required

To assemble the circuit on a breadboard you need the following parts:

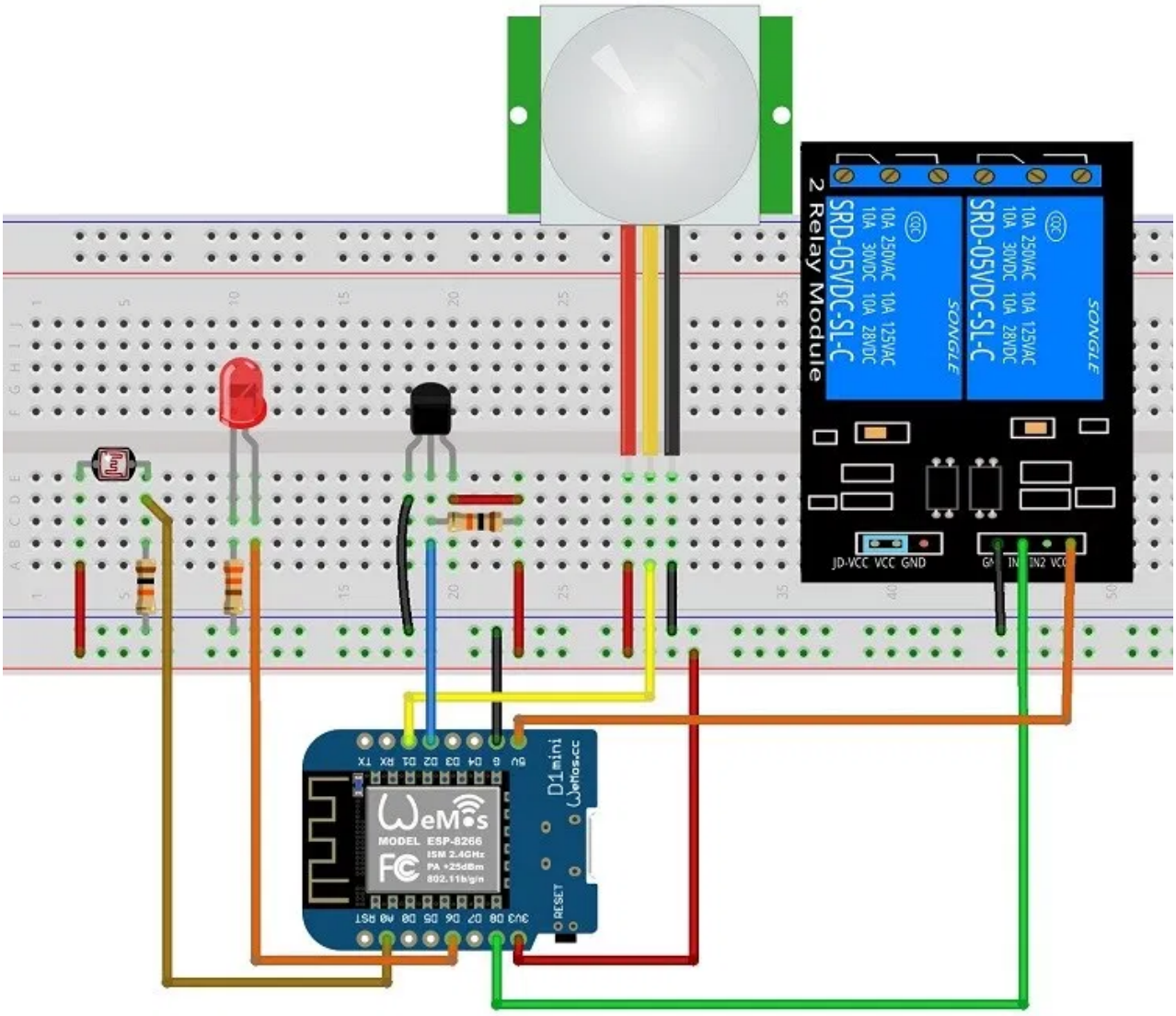
- ESP8266 Wemos D1 Mini – read Best ESP8266 Wi-Fi Development Board
- 1x 5mm LED
- 1x 330 Ohm resistor
- 1x DS18B20 temperature sensor
- 1x mini PIR motion sensor
- 1x light dependent resistor
- 2x 10k Ohm resistor

- 1x relay module
- Breadboard
- Jumper wires

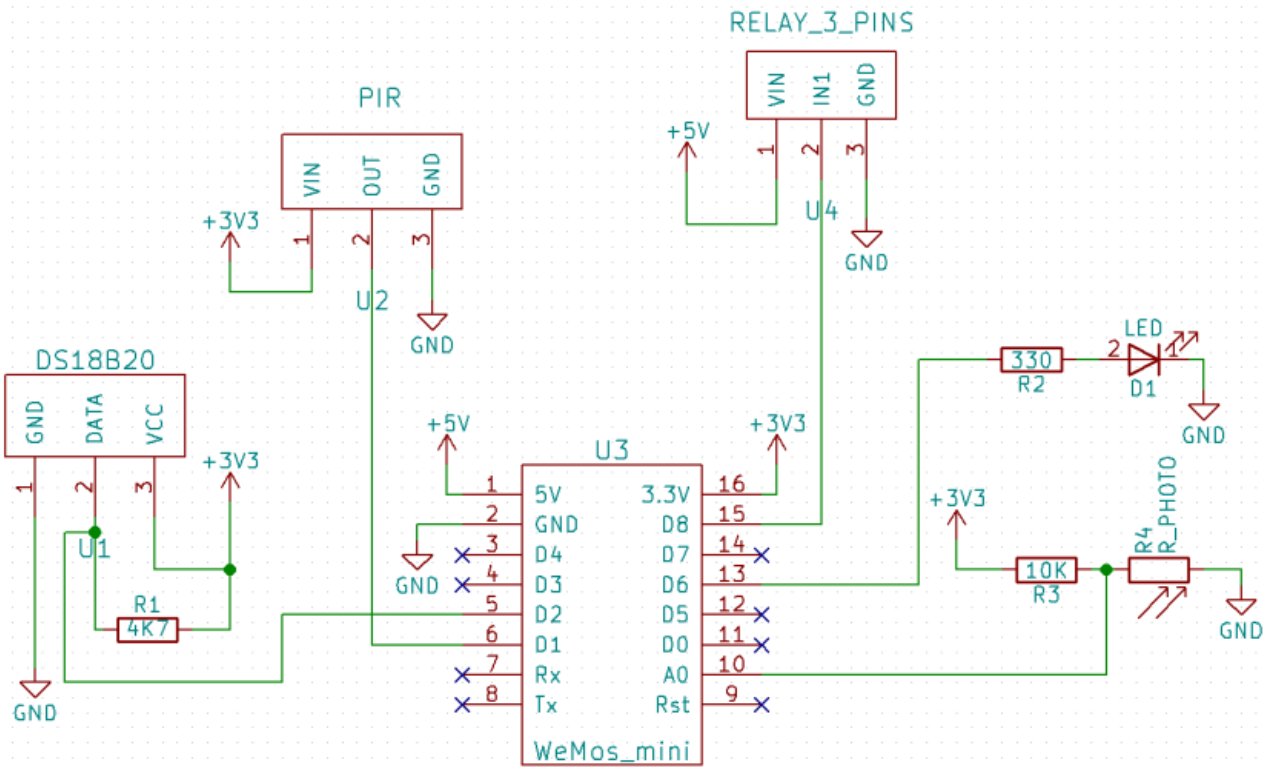
You can use the preceding links or go directly to MakerAdvisor.com/tools to find all the parts for your projects at the best price!

Schematic

After gathering all the parts, assemble the circuit by following the next schematic diagram:

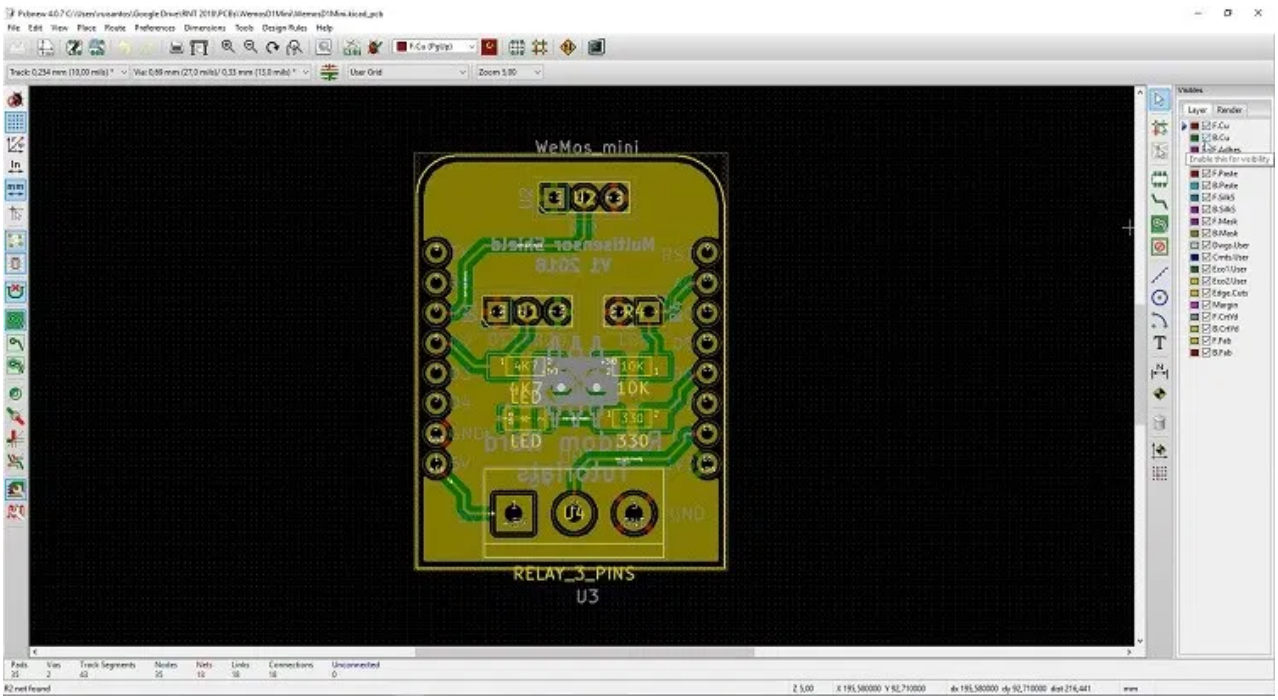


Here's the circuit diagram:



Designing the PCB

After making sure the circuit was working properly on a breadboard, I've designed a PCB on KiCad. KiCad is an open-source software used to design PCBs.



Designing an ESP8266 PCB shield is fairly straightforward. There are already pre-made KiCad parts for the WeMos, and you can find a link to the KiCad parts below:

WeMos D1 Mini KiCad Part

All the other components already exist on KiCad, like the LED, LDR, resistors, terminal blocks, and so on.

Designing the circuit works like in any other circuit software tool, you place some components and you wire them together. When you're happy with your circuit and pins usage, it's time to assign each component to a footprint.

The WeMos D1 Mini was already created, and all the other components were also available by default on the KiCad parts library.

Having the parts assigned, you can start placing each component. When you're happy with the layout, make all the connections and route your PCB.
Once you're done, save your project and export the Gerber files.

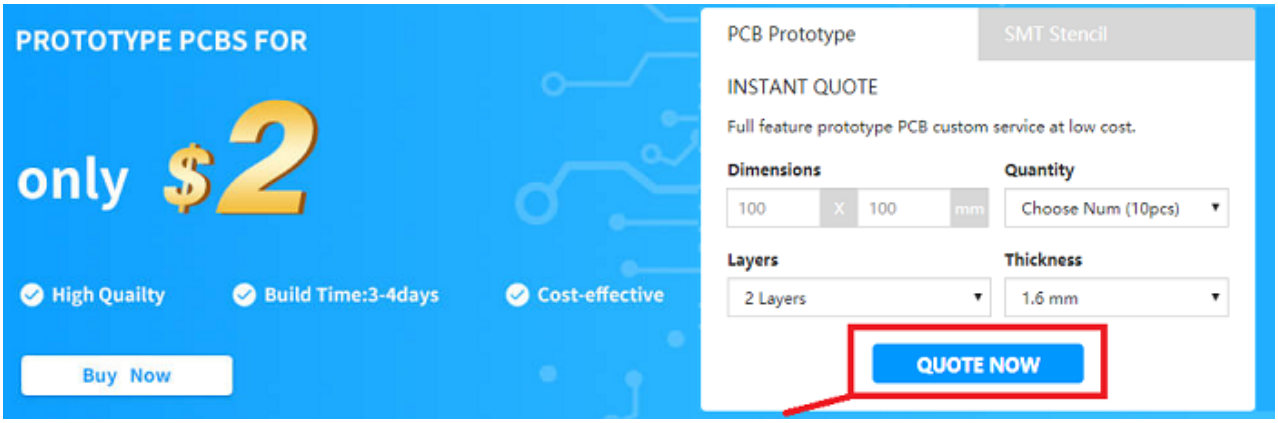
Note: if you're familiar with KiCad, you can grab the project files and edit them to customize the shield for your own needs.

Ordering the PCBs

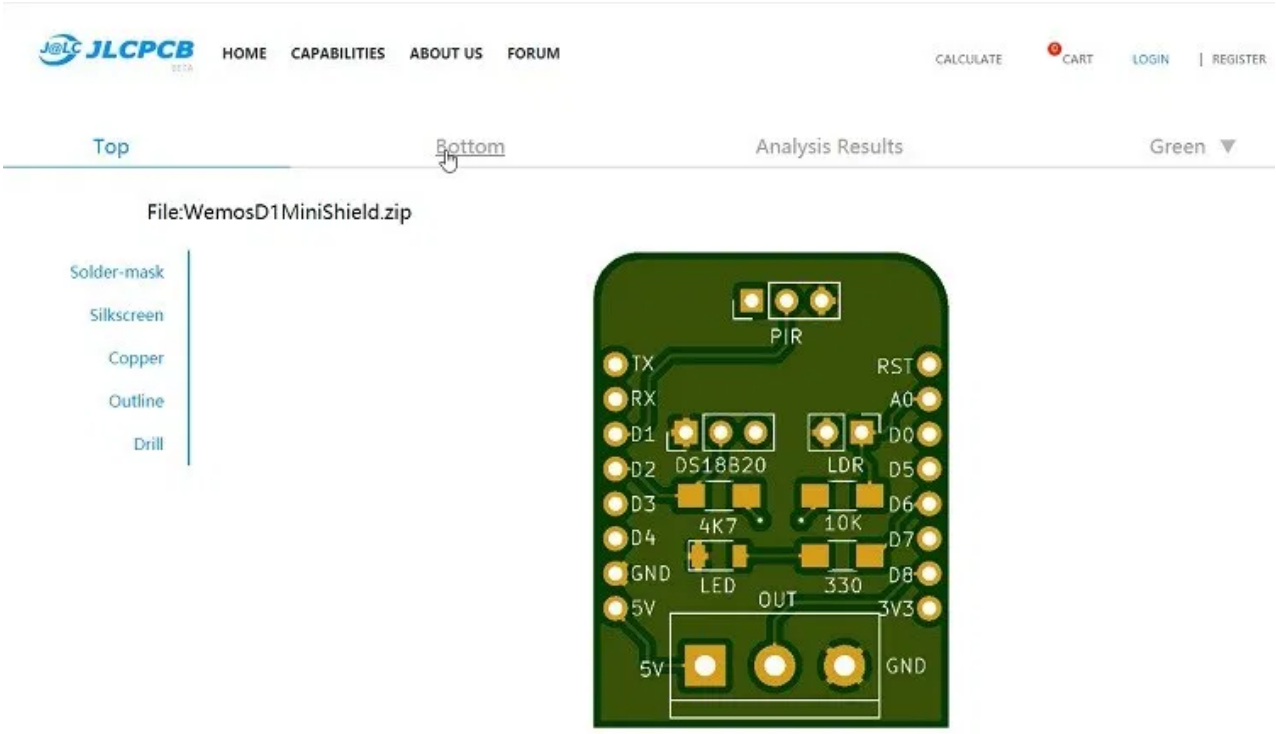
If you don't know how to design the PCB yourself, but you still want to order them, it's very easy. You just have to:

1. Download the Gerber files – click here to download the .zip file

2. Go to JLCPCB.com, click the "**QUOTE NOW**" button, and upload the .zip file you've just downloaded.

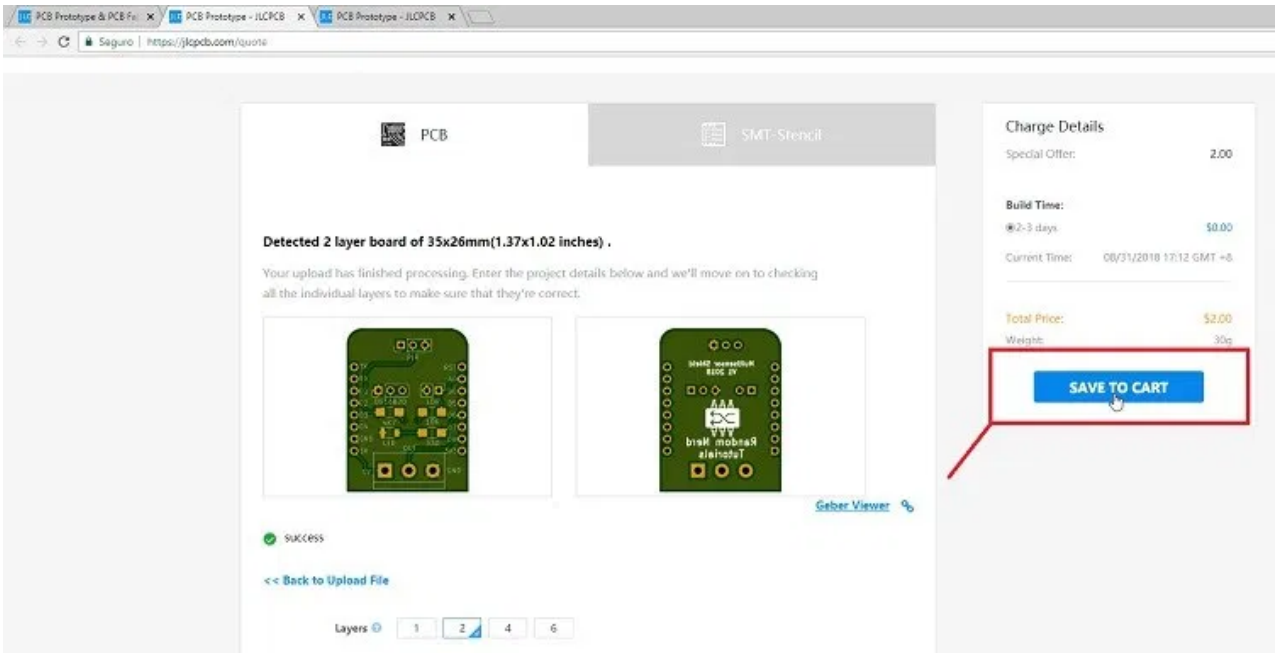


3. You'll see a success message at the bottom. Then, you can use the "Gerber Viewer" link at the bottom right corner to check if everything went as expected. You can view the top and bottom of the PCB. You can view or hide the solder-mask, silkscreen, copper, etc.



With the default settings, you can order 10 PCBs for just $2. However, if you want to select other settings like a different PCB Color it will cost you a few more dollars.

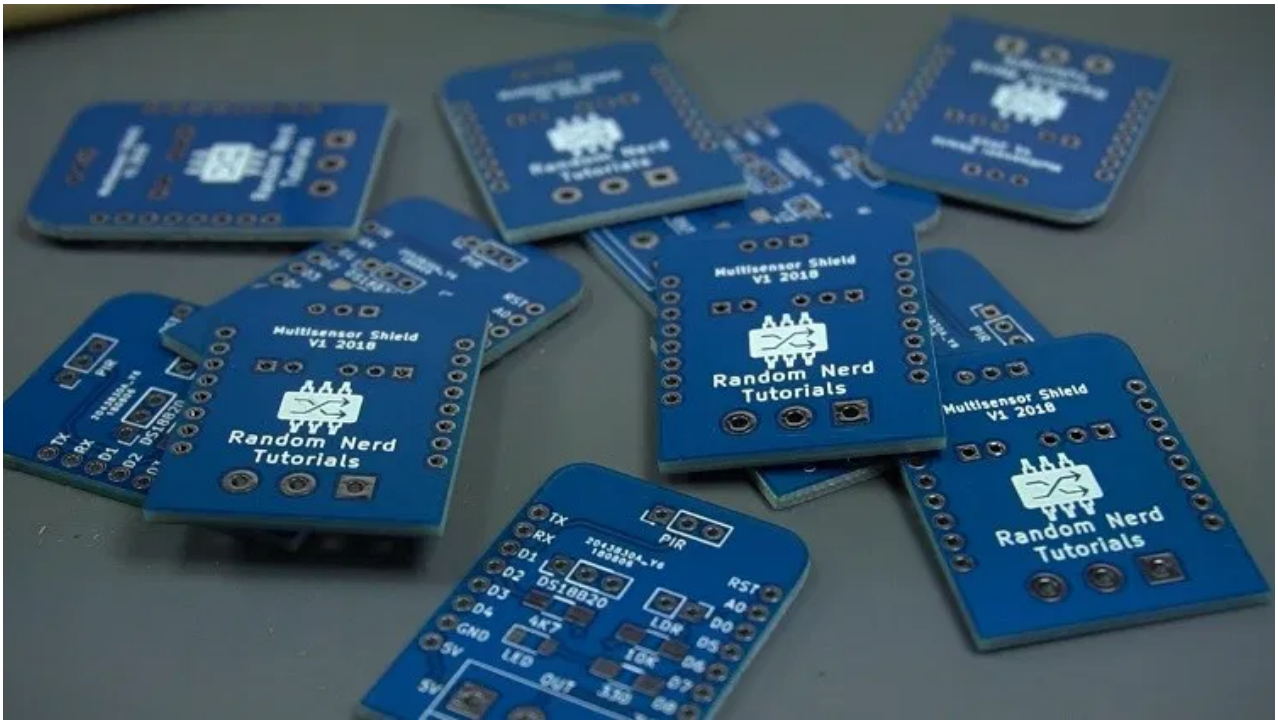
4. When you're happy with your order, click the "**SAVE TO CART**" button to complete the order.



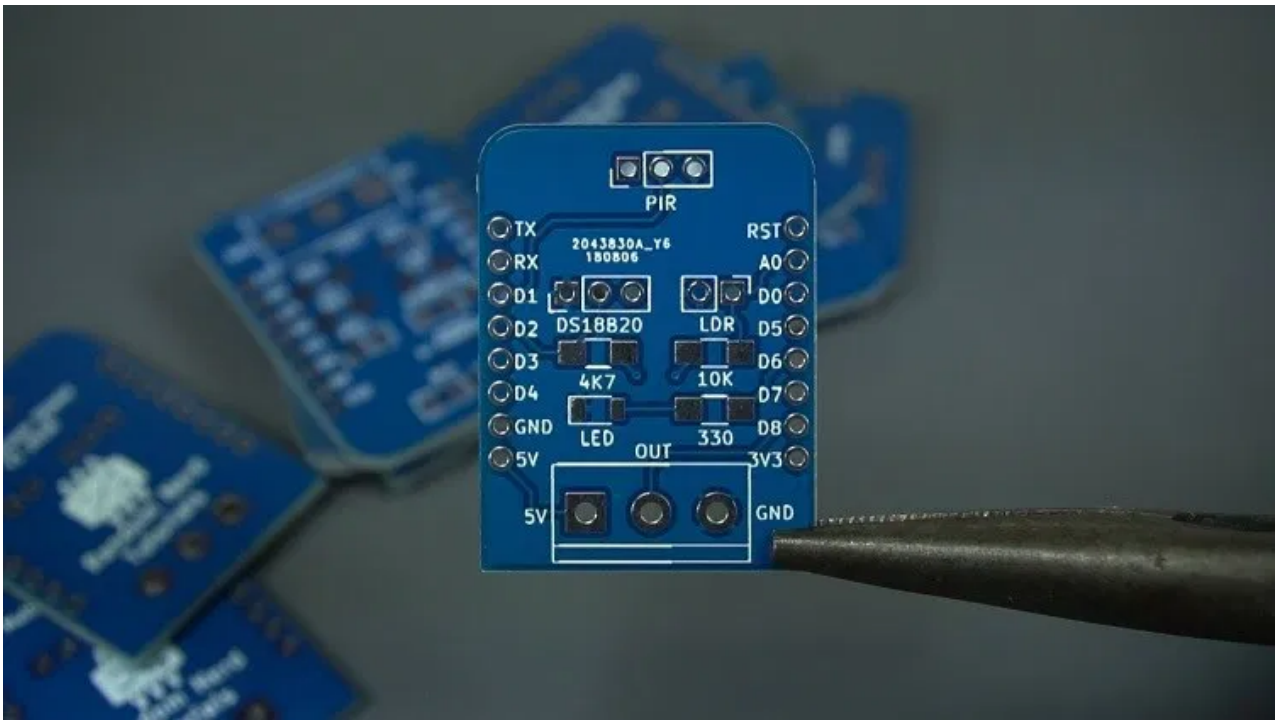
My PCBs took 2 days to be manufactured and they arrived in 3 business days using DHL delivery option.

Unboxing

After approximately one week, I received the PCBs at my office. As usual, everything came well packed, and the PCBs are really high-quality.



The letters on the silkscreen are really well-printed and easy to read. I don't think you can get a better PCB service for this price.



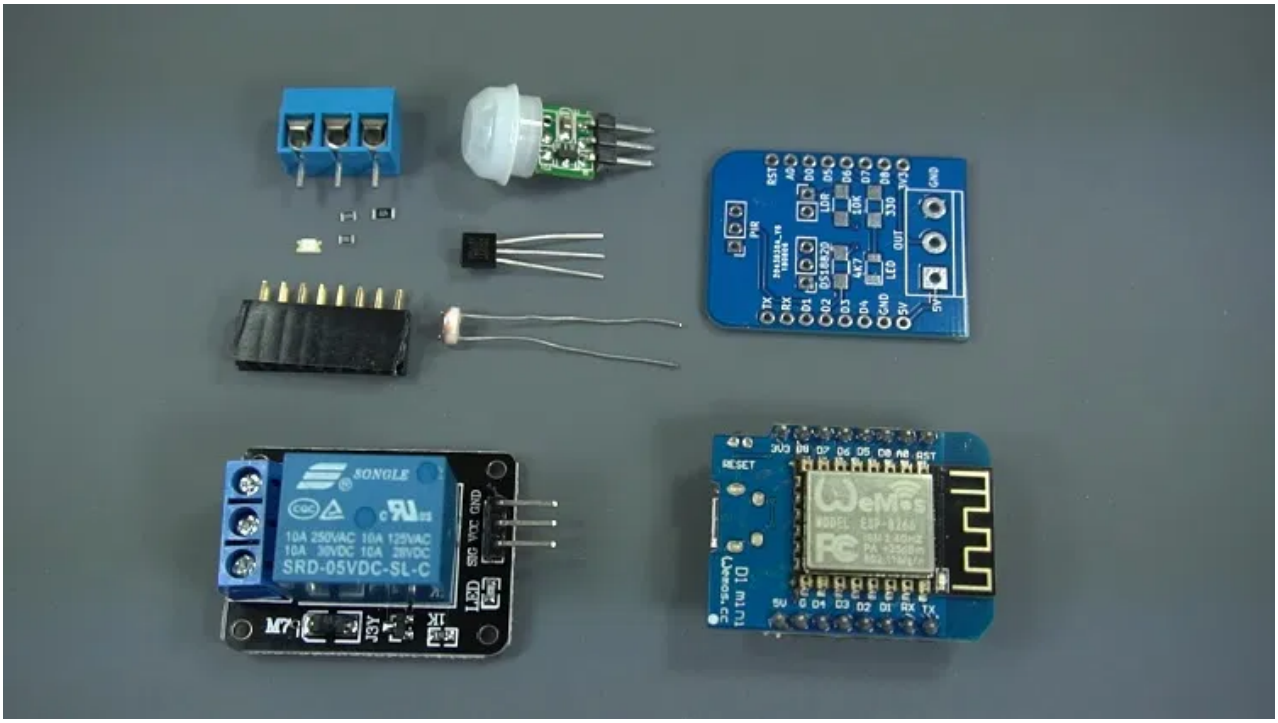
Soldering the Components

The next step is soldering the components to the PCB. I've used an SMD LED and SMD resistors. I know it's a bit difficult to solder SMD components, but they save a lot of space on the PCB.

Here's the soldering tools I've used:

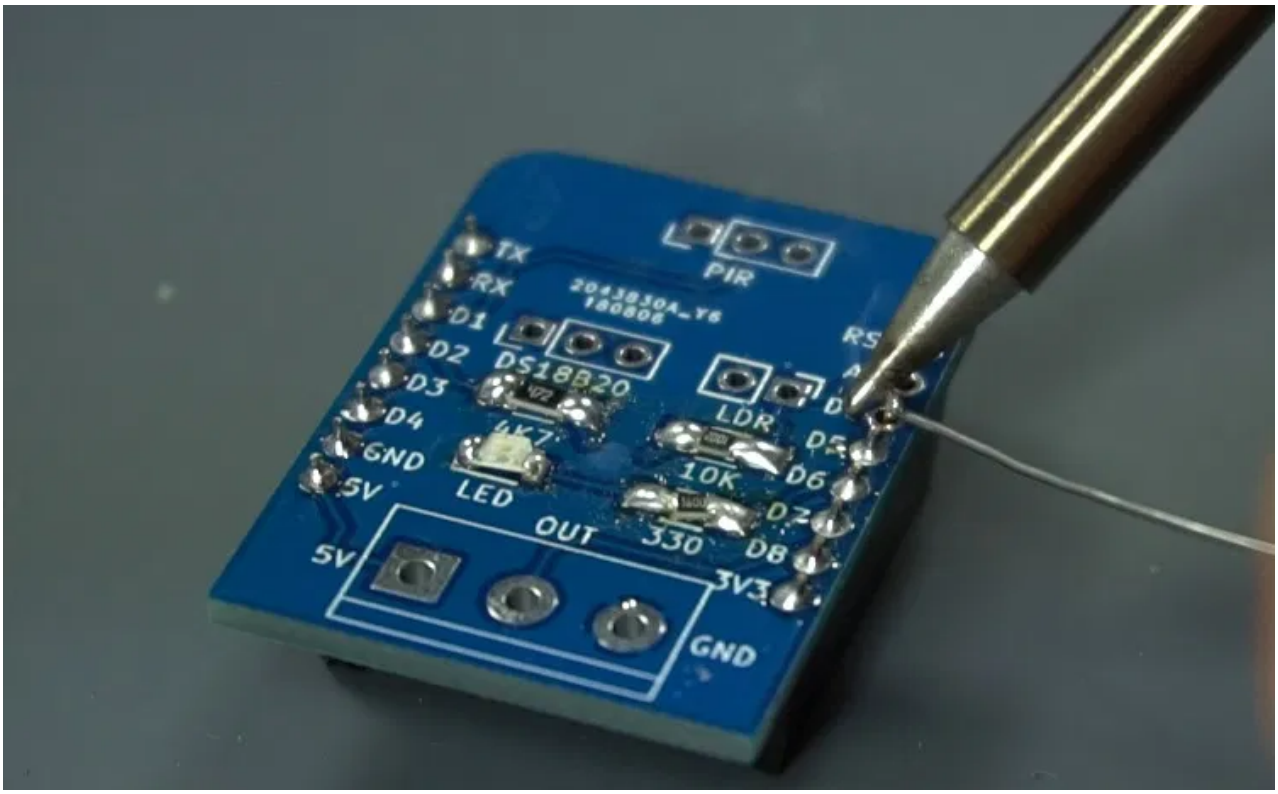
Read our review about the TS100 Soldering Iron: TS100 Soldering Iron Review – Best Portable Soldering Iron.

Here’s a list of all the components you need to solder on the PCB (don’t forget the header pins to attach the shield to the Wemos D1 mini):

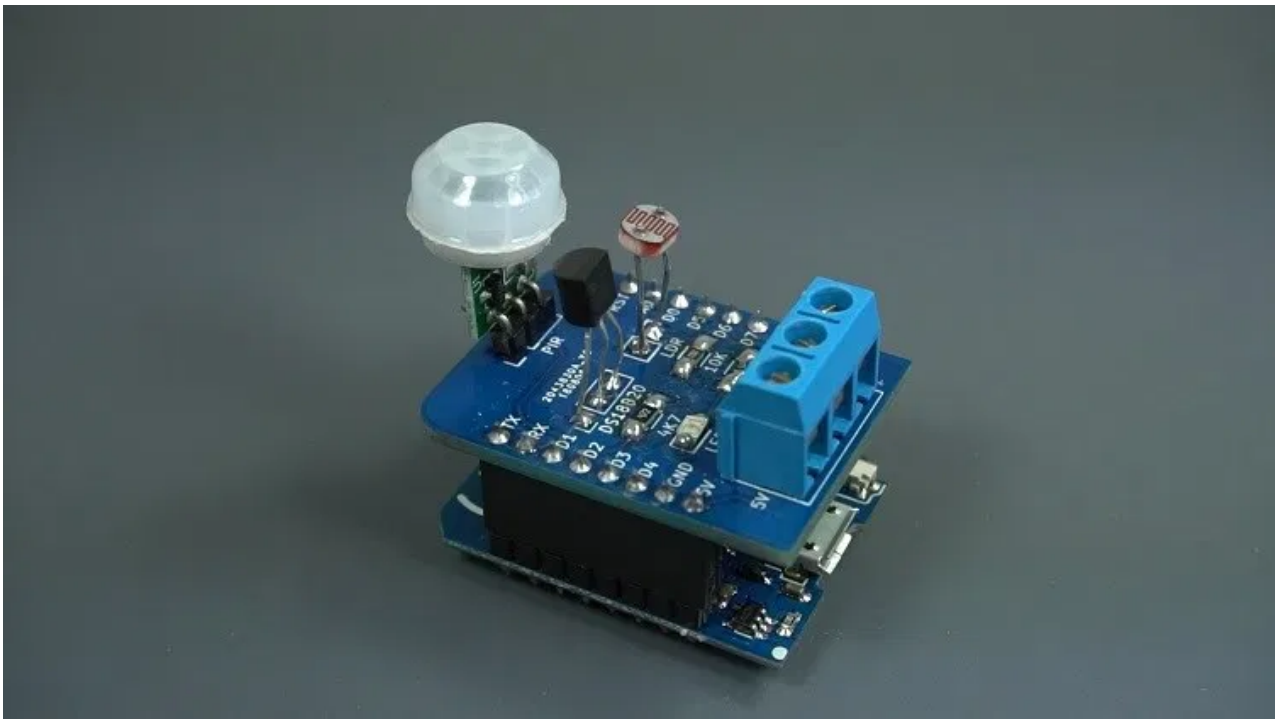


- 1x SMD LED
- 1x 330 Ohm SMD resistors
- 2x 10k Ohm SMD resistor
- 1x DS18B20 temperature sensor
- 1x mini PIR motion sensor
- 1x light dependent resistor
- 1x Screw terminal blocks
- Female pin header socket

Start by soldering the SMD components. Then, solder the header pins. Finally, solder the other components.



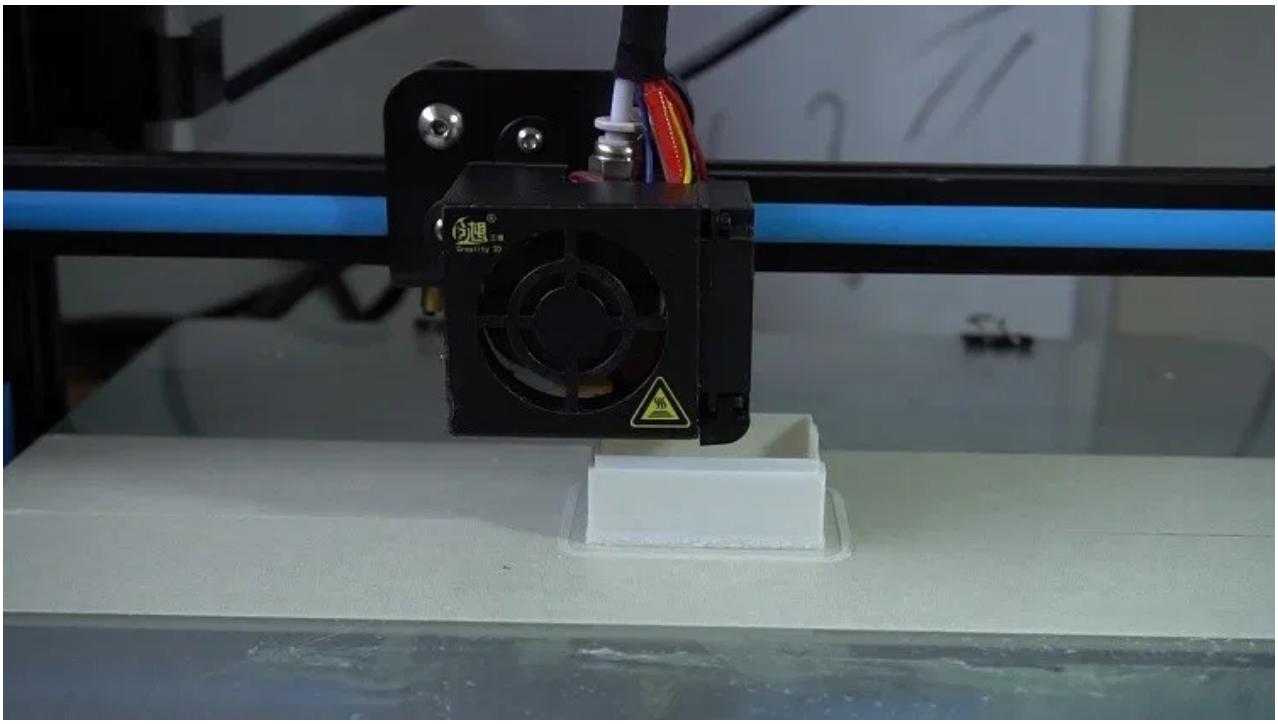
Here's how the WeMos Multisensor Shield looks like after assembling all the parts. It should connect perfectly to the ESP8266 WeMos D1 mini.



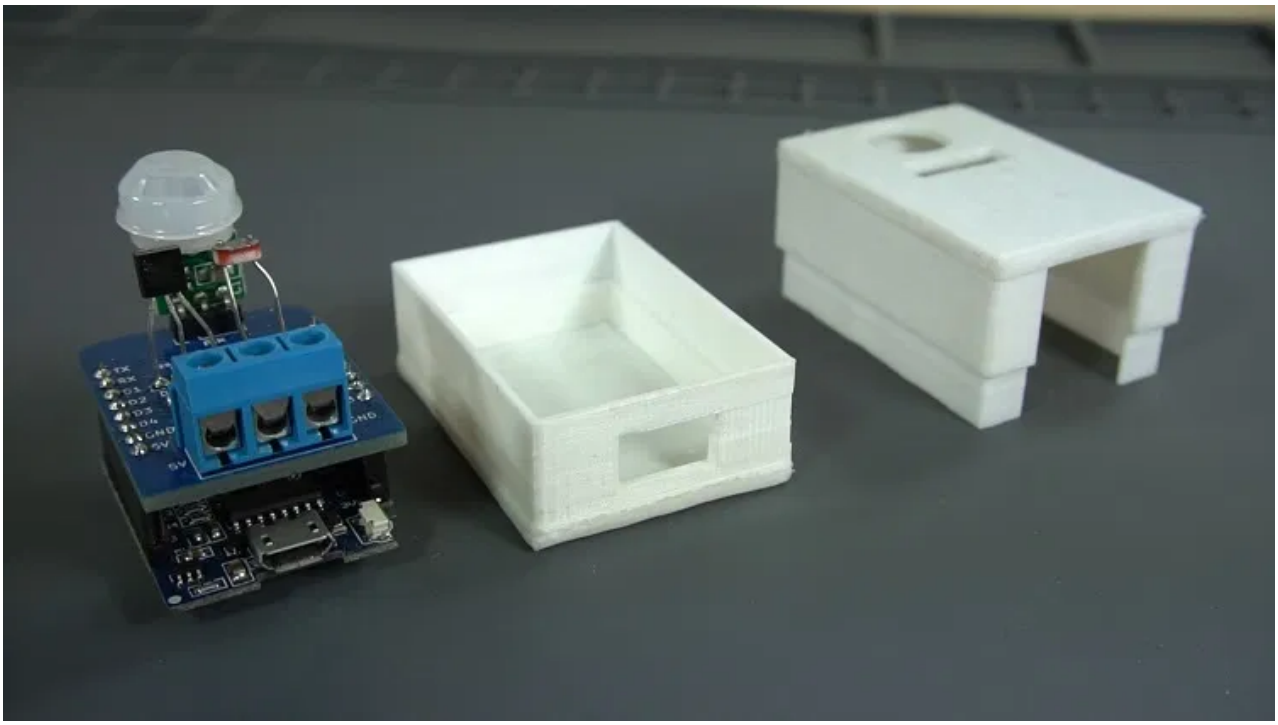
3D Printed Enclosure

Finally, you can buy an enclosure to place your circuit. If you have a 3D printer, you can build your own enclosure.

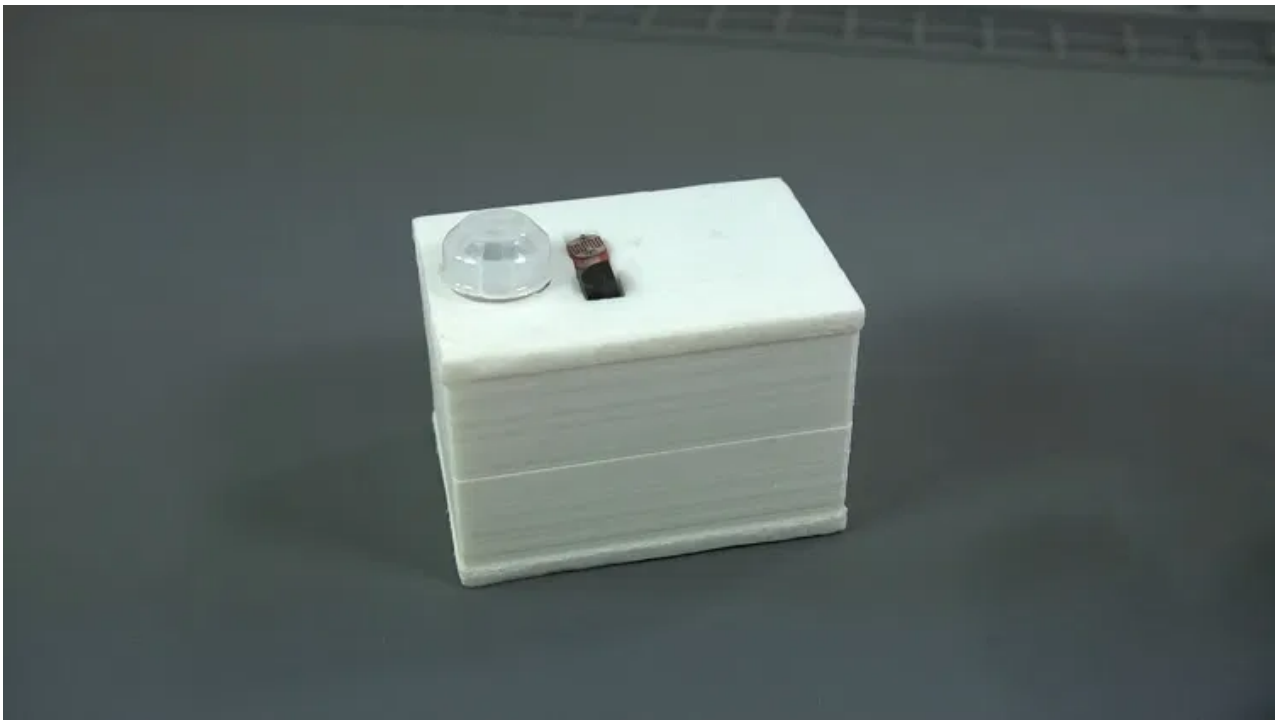
I've 3D printed a simple enclosure to place the multisensor shield using the Creality CR-10 3D printer.



The enclosure consists of two pieces: the bottom and the lid. The lid has a space for the PIR motion sensor, and a rectangular hole for the LDR and temperature sensor. At the side, there's a space for the relay wires and another for the USB cable to go through to power and program the ESP8266.



That's it, the shield is finished!



Now, it's time to write some code.

Programming the Multisensor Shield

The code for this project runs a web server that allows you to monitor and control the multisensor shield based on several configurable settings.

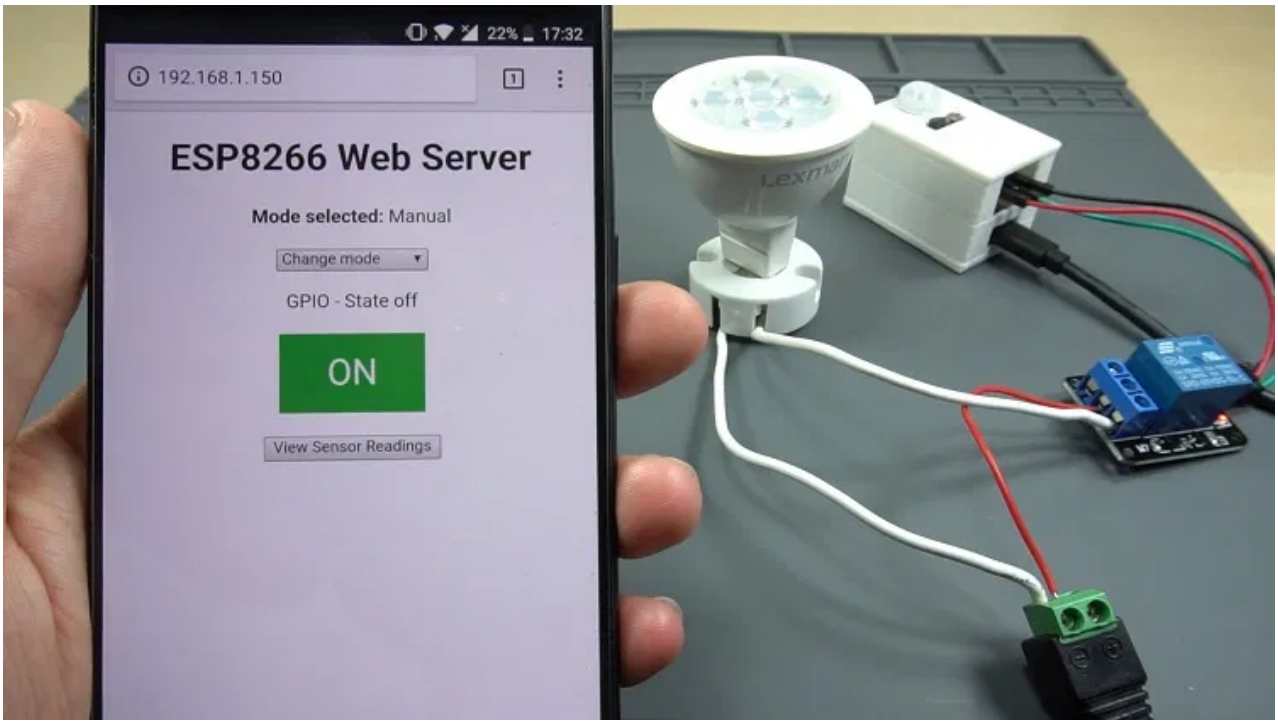
Before proceeding you should have the ESP8266 installed in your Arduino IDE. Follow the next tutorial to install the ESP8266 on the Arduino IDE, if you haven't already.

> How to Install the ESP8266 Board in Arduino IDE

Web Server Overview

Let's continue with this project, the web server we'll build allows you to choose between 4 different modes to control the relay:

- **Manual (mode 0)**: in which you have a button to turn the relay on and off.
- **Auto PIR (mode 1)**: turns the relay on when motion is detected. In this mode there is a field in which you can set the number of seconds the output will be on after motion is detected.
- **LDR (mode 2)**: the relay turns on when the luminosity goes below a certain threshold. You can set an LDR threshold value between 0 and 100%.
- **Auto PIR and LDR (mode 3)**: this mode combines the PIR motion sensor and the LDR. When this mode is selected, the relay turns on when the PIR sensor detects motion and if the luminosity value is below the threshold. In this mode you can set the timer and the LDR threshold value.



In the web server, there's also a button that you can press to request the temperature reading. After requesting the temperature readings, you can press the "Remove Sensor Readings" button to hide the readings to optimize the web server performance.

In every mode there's a label that shows the selected mode, as well as the current output state.

We want the ESP8266 to remember the last output state and the settings, in case it resets or suddenly loses power. So, we need to save those parameters in the ESP8266 EEPROM.

Installing libraries

Before uploading the code, you need to install two libraries in your Arduino IDE: the OneWire library by Paul Stoffregen and the Dallas Temperature library, so that you can use the DS18B20 sensor. Follow the next steps to install those libraries.

OneWire library

1. Click here to download the OneWire library. You should have a .zip folder in your Downloads
2. Unzip the *.zip* folder and you should get **OneWire-master** folder
3. Rename your folder from ~~**OneWire-master**~~ to **OneWire**
4. Move the **OneWire** folder to your Arduino IDE installation **libraries** folder
5. Finally, re-open your Arduino IDE

Dallas Temperature library

1. Click here to download the DallasTemperature library. You should have a .zip folder in your Downloads

2. Unzip the *.zip* folder and you should get **Arduino-Temperature-Control-Library-master** folder
3. Rename your folder from ~~**Arduino-Temperature-Control-Library-master**~~ to **DallasTemperature**
4. Move the **DallasTemperature**folder to your Arduino IDE installation **libraries** folder
5. Finally, re-open your Arduino IDE

Code

Copy the following code to the Arduino IDE.

```
/*********
  Rui Santos
  Complete project details at https://randomnerdtutorials.com
*********/

// Load libraries
#include <ESP8266WiFi.h>
#include <EEPROM.h>
#include <OneWire.h>
#include <DallasTemperature.h>

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

// Auxiliary variables for temperature
static char celsiusTemp[7];
static char fahrenheitTemp[7];
String temperatureString = "";      // Variable to hold the temperature reading

// EEPROM size
// Address 0: Last output state (0 = off or 1 = on)
// Address 1: Selected mode (0 = Manual, 1 = Auto PIR,
// 2 = Auto LDR, or 3 = Auto PIR and LDR)
// Address 2: Timer (time 0 to 255 seconds)
// Address 3: LDR threshold value (luminosity in percentage 0 to 100%)
#define EEPROM_SIZE 4

// Set GPIOs for: output variable, status LED, PIR Motion Sensor, and LDR
const int output = 15;
const int statusLed = 12;
const int motionSensor = 5;
const int ldr = A0;
// Store the current output state
String outputState = "off";

// GPIO where the DS18B20 is connected to
const int oneWireBus = 4;
// Setup a oneWire instance to communicate with any OneWire devices
OneWire oneWire(oneWireBus);
// Pass our oneWire reference to Dallas Temperature sensor
DallasTemperature sensors(&oneWire);

// Timers - Auxiliary variables
unsigned long now = millis();
unsigned long lastMeasure = 0;
boolean startTimer = false;
unsigned long currentTime = millis();
unsigned long previousTime = 0;
const long timeoutTime = 2000;

// Auxiliary variables to store selected mode and settings
int selectedMode = 0;
int timer = 0;
int ldrThreshold = 0;
int armMotion = 0;
int armLdr = 0;
String modes[4] = { "Manual", "Auto PIR", "Auto LDR", "Auto PIR and LDR" };
```

```
// Decode HTTP GET value
String valueString = "0";
int pos1 = 0;
int pos2 = 0;
// Variable to store the HTTP request
String header;
// Set web server port number to 80
WiFiServer server(80);

void setup() {
  // Start the DS18B20 sensor
  sensors.begin();

  // Serial port for debugging purposes
  Serial.begin(115200);

  // PIR Motion Sensor mode, then set interrupt function and RISING mode
  pinMode(motionSensor, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(motionSensor), detectsMovement, RISING);

  Serial.println("start...");
  EEPROM.begin(EEPROM_SIZE);

  // Uncomment the next lines to test the values stored in the flash memory
  /*Serial.println(" bytes read from Flash . Values are:");
  for(int i = 0; i < EEPROM_SIZE; i++) {
    Serial.print(byte(EEPROM.read(i)));
    Serial.print(" ");
  }*/

  // Initialize the output variable and the LED as OUTPUTs
  pinMode(output, OUTPUT);
  pinMode(statusLed, OUTPUT);
  digitalWrite(output, HIGH);
  digitalWrite(statusLed, LOW);
  // Read from flash memory on start and store the values in auxiliary variables
  // Set output to last state (saved in the flash memory)
  if(!EEPROM.read(0)) {
    outputState = "off";
    digitalWrite(output, HIGH);
  }
  else {
    outputState = "on";
    digitalWrite(output, LOW);
  }
  selectedMode = EEPROM.read(1);
  timer = EEPROM.read(2);
  ldrThreshold = EEPROM.read(3);
  configureMode();

  // 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();
}

void loop() {
  WiFiClient client = server.available();   // Listen for incoming clients
  if (client) {                             // If a new client connects,
    currentTime = millis();
    previousTime = currentTime;
    Serial.println("New Client.");          // print a message out in the serial
port
    String currentLine = "";                // make a String to hold incoming data
from the client
    while (client.connected() && currentTime - previousTime <= timeoutTime) {
// loop while the client's connected
      currentTime = millis();
      if (client.available()) {             // if there's 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) {
            // 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();
            // 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 and font-size attributes
to fit your preferences
            client.println("<style>html { font-family: Helvetica; display: inline-
block; margin: 0px auto; text-align: center;}");
            client.println(".button { background-color: #4CAF50; border: none;
color: white; padding: 16px 40px;");
            client.println("text-decoration: none; font-size: 30px; margin: 2px;
cursor: pointer;}");
            client.println(".button2 {background-color: #555555;}</style>
</head>");

            // Request example: GET /?mode=0& HTTP/1.1 - sets mode to Manual (0)
            if(header.indexOf("GET /?mode=") >= 0) {
              pos1 = header.indexOf('=');
              pos2 = header.indexOf('&');
              valueString = header.substring(pos1+1, pos2);
```

```
              selectedMode = valueString.toInt();
              EEPROM.write(1, selectedMode);
              EEPROM.commit();
              configureMode();
            }
            // Change the output state - turn GPIOs on and off
            else if(header.indexOf("GET /?state=on") >= 0) {
              outputOn();
            }
            else if(header.indexOf("GET /?state=off") >= 0) {
              outputOff();
            }
            // Set timer value
            else if(header.indexOf("GET /?timer=") >= 0) {
              pos1 = header.indexOf('=');
              pos2 = header.indexOf('&');
              valueString = header.substring(pos1+1, pos2);
              timer = valueString.toInt();
              EEPROM.write(2, timer);
              EEPROM.commit();
              Serial.println(valueString);
            }
            // Set LDR Threshold value
            else if(header.indexOf("GET /?ldrthreshold=") >= 0) {
              pos1 = header.indexOf('=');
              pos2 = header.indexOf('&');
              valueString = header.substring(pos1+1, pos2);
              ldrThreshold = valueString.toInt();
              EEPROM.write(3, ldrThreshold);
              EEPROM.commit();
              Serial.println(valueString);
            }

            // Web Page Heading
            client.println("<body><h1>ESP8266 Web Server</h1>");
            // Drop down menu to select mode
            client.println("<p><strong>Mode selected:</strong> " +
modes[selectedMode] + "</p>");
            client.println("<select id=\"mySelect\"
onchange=\"setMode(this.value)\">");
            client.println("<option>Change mode");
            client.println("<option value=\"0\">Manual");
            client.println("<option value=\"1\">Auto PIR");
            client.println("<option value=\"2\">Auto LDR");
            client.println("<option value=\"3\">Auto PIR and LDR</select>");

            // Display current state, and ON/OFF buttons for output
            client.println("<p>GPIO - State " + outputState + "</p>");
            // If the output is off, it displays the ON button
            if(selectedMode == 0) {
              if(outputState == "off") {
                client.println("<p><button class=\"button\"
onclick=\"outputOn()\">ON</button></p>");
              }
              else {
                client.println("<p><button class=\"button button2\"
onclick=\"outputOff()\">OFF</button></p>");
              }
            }
```

```
            else if(selectedMode == 1) {
              client.println("<p>Timer (0 and 255 in seconds): <input
type=\"number\" name=\"txt\" value=\"" +
                              String(EEPROM.read(2)) + "\"
onchange=\"setTimer(this.value)\" min=\"0\" max=\"255\"></p>");
            }
            else if(selectedMode == 2) {
              client.println("<p>LDR Threshold (0 and 100%): <input
type=\"number\" name=\"txt\" value=\"" +
                              String(EEPROM.read(3)) + "\"
onchange=\"setThreshold(this.value)\" min=\"0\" max=\"100\"></p>");
            }
            else if(selectedMode == 3) {
              client.println("<p>Timer (0 and 255 in seconds): <input
type=\"number\" name=\"txt\" value=\"" +
                               String(EEPROM.read(2)) + "\"
onchange=\"setTimer(this.value)\" min=\"0\" max=\"255\"></p>");
              client.println("<p>LDR Threshold (0 and 100%): <input
type=\"number\" name=\"txt\" value=\"" +
                               String(EEPROM.read(3)) + "\"
onchange=\"setThreshold(this.value)\" min=\"0\" max=\"100\"></p>");
            }
            // Get and display DHT sensor readings
            if(header.indexOf("GET /?sensor") >= 0) {
              sensors.requestTemperatures();
              temperatureString = " " + String(sensors.getTempCByIndex(0)) + "C  "
+
                          String(sensors.getTempFByIndex(0)) + "F";

              client.println("<p>");
              client.println(temperatureString);
              client.println("</p>");
              client.println("<p><a href=\"/\"><button>Remove Sensor
Readings</button></a></p>");
            }
            else {
              client.println("<p><a href=\"?sensor\"><button>View Sensor
Readings</button></a></p>");
            }
            client.println("<script> function setMode(value) { var xhr = new
XMLHttpRequest();");
            client.println("xhr.open('GET', \"/?mode=\" + value + \"&\", true);");
            client.println("xhr.send(); location.reload(true); } ");
            client.println("function setTimer(value) { var xhr = new
XMLHttpRequest();");
            client.println("xhr.open('GET', \"/?timer=\" + value + \"&\",
true);");
            client.println("xhr.send(); location.reload(true); } ");
            client.println("function setThreshold(value) { var xhr = new
XMLHttpRequest();");
            client.println("xhr.open('GET', \"/?ldrthreshold=\" + value + \"&\",
true);");
            client.println("xhr.send(); location.reload(true); } ");
            client.println("function outputOn() { var xhr = new
XMLHttpRequest();");
            client.println("xhr.open('GET', \"/?state=on\", true);");
            client.println("xhr.send(); location.reload(true); } ");
            client.println("function outputOff() { var xhr = new
XMLHttpRequest();");
```

```
            client.println("xhr.open('GET', \"/?state=off\", true);");
            client.println("xhr.send(); location.reload(true); } ");
            client.println("function updateSensorReadings() { var xhr = new
XMLHttpRequest();");
            client.println("xhr.open('GET', \"/?sensor\", true);");
            client.println("xhr.send(); location.reload(true); }</script></body>
</html>");
            // The HTTP response ends with another blank line
            client.println();
            // Break out of the while loop
            break;
          } 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
        }
      }
    }
    // Clear the header variable
    header = "";
    // Close the connection
    client.stop();
    Serial.println("Client disconnected.");
  }

  // Starts a timer to turn on/off the output according to the time value or LDR
reading
  now = millis();

  // Mode selected (1): Auto PIR
  if(startTimer && armMotion && !armLdr) {
    if(outputState == "off") {
      outputOn();
    }
    else if((now - lastMeasure > (timer * 1000))) {
      outputOff();
      startTimer = false;
    }
  }

  // Mode selected (2): Auto LDR
  // Read current LDR value and turn the output accordingly
  if(armLdr && !armMotion) {
    int ldrValue = map(analogRead(ldr), 0, 1023, 0, 100);
    Serial.println(ldrValue);
    if(ldrValue < ldrThreshold && outputState == "on") {
      outputOff();
    }
    else if(ldrValue > ldrThreshold && outputState == "off") {
      outputOn();
    }
    delay(100);
  }

  // Mode selected (3): Auto PIR and LDR
  if(startTimer && armMotion && armLdr) {
    int ldrValue = map(analogRead(ldr), 0, 1023, 0, 100);
```

```
    Serial.println(ldrValue);
    if(ldrValue < ldrThreshold) {
      outputOff();
      startTimer = false;
      Serial.println("a");
    }
    else if(ldrValue > ldrThreshold && outputState == "off") {
      outputOn();
      Serial.println("b");
    }
    else if(now - lastMeasure > (timer * 1000)) {
      outputOff();
      startTimer = false;
      Serial.println("c");
    }
  }
}

// Checks if motion was detected and the sensors are armed. Then, starts a timer.
ICACHE_RAM_ATTR void detectsMovement() {
  if(armMotion || (armMotion && armLdr)) {
    Serial.println("MOTION DETECTED!!!");
    startTimer = true;
    lastMeasure = millis();
  }
}
void configureMode() {
  // Mode: Manual
  if(selectedMode == 0) {
    armMotion = 0;
    armLdr = 0;
    // Status LED: off
    digitalWrite(statusLed, LOW);
  }
  // Mode: Auto PIR
  else if(selectedMode == 1) {
    outputOff();
    armMotion = 1;
    armLdr = 0;
    // Status LED: on
    digitalWrite(statusLed, HIGH);
  }
  // Mode: Auto LDR
  else if(selectedMode == 2) {
    armMotion = 0;
    armLdr = 1;
    // Status LED: on
    digitalWrite(statusLed, HIGH);
  }
  // Mode: Auto PIR and LDR
  else if(selectedMode == 3) {
    outputOff();
    armMotion = 1;
    armLdr = 1;
    // Status LED: on
    digitalWrite(statusLed, HIGH);
  }
}
```

```
// Change output pin to on or off
void outputOn() {
  Serial.println("GPIO on");
  outputState = "on";
  digitalWrite(output, LOW);
  EEPROM.write(0, 1);
  EEPROM.commit();
}
void outputOff() {
  Serial.println("GPIO off");
  outputState = "off";
  digitalWrite(output, HIGH);
  EEPROM.write(0, 0);
  EEPROM.commit();
}
```

View raw code

This code is quite long to explain, so you can simply replace the following two variables with your network credentials and the code will work straight away.

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

If you want to learn how this code works, continue reading.

How the Code Works

Start by including the necessary libraries.

```
#include <ESP8266WiFi.h>
#include <EEPROM.h>
#include <OneWire.h>
#include <DallasTemperature.h>
```

The *ESP8266WiFi* library is needed to use the ESP Wi-Fi capabilities. The *EEPROM* library allows you to read and write permanent data on the ESP8266 EEPROM, and the OneWire and DallasTemperature libraries allow you to read the temperature from the DS18B20 temperature sensor.

Setting your Network Credentials

You need to add your network credentials in these two variables.

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

These are auxiliary variables to store the temperature in Celsius/Fahrenheit:

```
// Auxiliary variables for temperature and humidity
static char celsiusTemp[7];
static char fahrenheitTemp[7];
static char humidityTemp[7];
String temperatureString = ""; // Variable to hold the temperature reading
```

EEPROM

Next, we define the EEPROM size we want to access.

```
#define EEPROM_SIZE 4
```

We'll need to save four values in the flash memory: the last output state on address 0, the selected mode on address 1, the timer value on address 2, and the LDR threshold value on address 3. So, we need 4 bytes in the flash memory.

- **Address 0**: Last output state (0 = off or 1 = on)
- **Address 1**: Selected mode (0 = Manual, 1 = Auto PIR, 2 = Auto LDR, or 3 = Auto PIR and LDR)
- **Address 2**: Timer (time 0 to 255 seconds)
- **Address 3**: LDR threshold value (luminosity in percentage 0 to 100%)

Defining GPIOs

In this section, we define the GPIOs for the output, the status LED, PIR motion sensor, and the LDR.

```
const int output = 15;
const int statusLed = 12;
const int motionSensor = 5;
const int ldr = A0;
```

We also create a String variable to hold the outputState to be displayed on the web server.

```
String outputState = "off";
```

Temperature Sensor

Next, create the instances needed for the temperature sensor. The temperature sensor is connected to GPIO 4.

```
// GPIO where the DS18B20 is connected to
const int oneWireBus = 4;
// Setup a oneWire instance to communicate with any OneWire devices
OneWire oneWire(oneWireBus);
// Pass our oneWire reference to Dallas Temperature sensor
DallasTemperature sensors(&oneWire);
```

Timers

Next, we create auxiliary variables for the timers:

```
long now = millis();
long lastMeasure = 0;
boolean startTimer = false;
```

Selected Mode and Settings

Here, we initialize variables to store the selected mode and settings:

```
int selectedMode = 0;
int timer = 0;
int ldrThreshold = 0;
int armMotion = 0;
int armLdr = 0;
String modes[4] = { "Manual", "Auto PIR", "Auto LDR", "Auto PIR and LDR" };
```

Setting Variables for the Web Server

The following snippet of code is related to the web server. You can follow the ESP8266 Web Server tutorial to get familiar with a basic web server code.

```
// Decode HTTP GET value
String valueString = "0";
int pos1 = 0;
int pos2 = 0;
// Variable to store the HTTP request
String header;
// Set web server port number to 80
WiFiServer server(80);
```

setup()

In the *setup()*, start by initializing the DS18B20 temperature sensor.

```
sensors.begin();
```

Initialize the Serial Port at a baud rate of 115200 for debugging purposes.

```
Serial.begin(115200);
```

Interrupt

Set the PIR motion sensor as an INPUT_PULLUP, and define it as an interrupt in RISING mode.

```
pinMode(motionSensor, INPUT_PULLUP);
attachInterrupt(digitalPinToInterrupt(motionSensor), detectsMovement, RISING);
```

Flash memory

This part of the code initializes the flash memory with the EEPROM size defined earlier.

```
EEPROM.begin(EEPROM_SIZE);
```

Set the status LED and the output pin as outputs.

```
pinMode(output, OUTPUT);
pinMode(statusLed, OUTPUT);
```

We're controlling a relay with inverted logic, so we start by setting it to HIGH, so it actually starts off.

```
digitalWrite(output, HIGH);
digitalWrite(statusLed, HIGH);
```

Then, set the output to the last saved state. The output state is saved on position 0, so use EEPROM.read(0).

We check if the state saved is on or off to update the output state accordingly.

```
if(!EEPROM.read(0)) {
  outputState = "off";
  digitalWrite(output, HIGH);
}
else {
  outputState = "on";
  digitalWrite(output, LOW);
}
```

We also update all variables that hold settings with the values saved in the EEPROM, like the selected mode, timer, and LDR threshold value.

```
selectedMode = EEPROM.read(1);
timer = EEPROM.read(2);
ldrThreshold = EEPROM.read(3);
```

Then, we call the *configureMode()* function to assign the right values to each mode.

```
configureMode();
```

configureMode() function

Let's take a look on how this function works. If the selected mode is Manual, the motion is not activated (*armMotion*), neither the LDR (*armLdr*). We also set the status LED to LOW to show that we are in manual mode (in the automatic modes, the status LED is on).

```
if(selectedMode == 0) {
  armMotion = 0;
  armLdr = 0;
  // Status LED: off
  digitalWrite(statusLed, LOW);
}
```

A similar process is done to configure the other modes. You change the arm variables to activate or deactivate a sensor. Now, let's go back to the *setup()*.

Wi-Fi connection

Here, we connect to the Wi-Fi network and print the ESP8266 IP address in the Serial Monitor.

```
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();
```

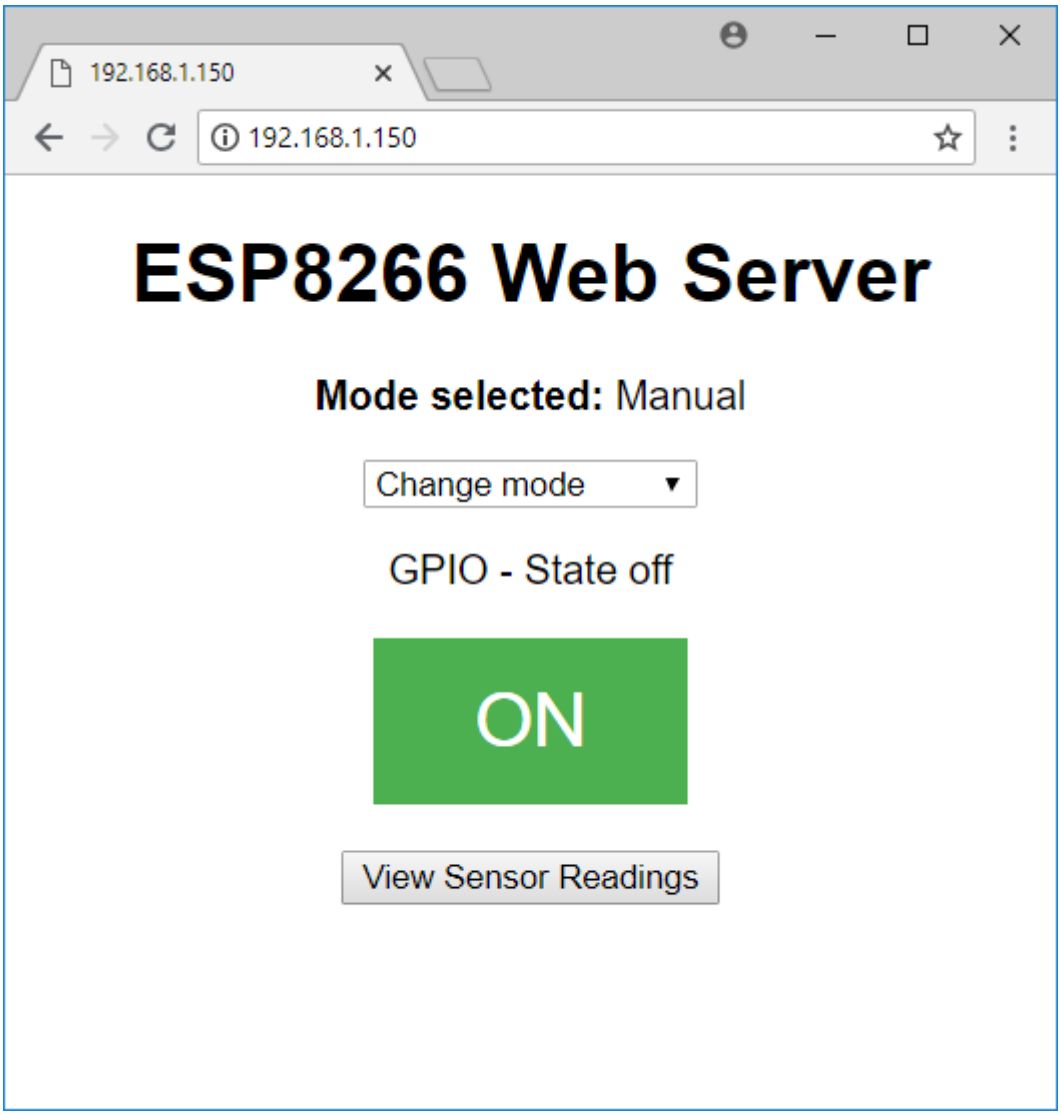
loop()

In the *loop()*, we display the web server and make things happen accordingly to the selected mode and settings.

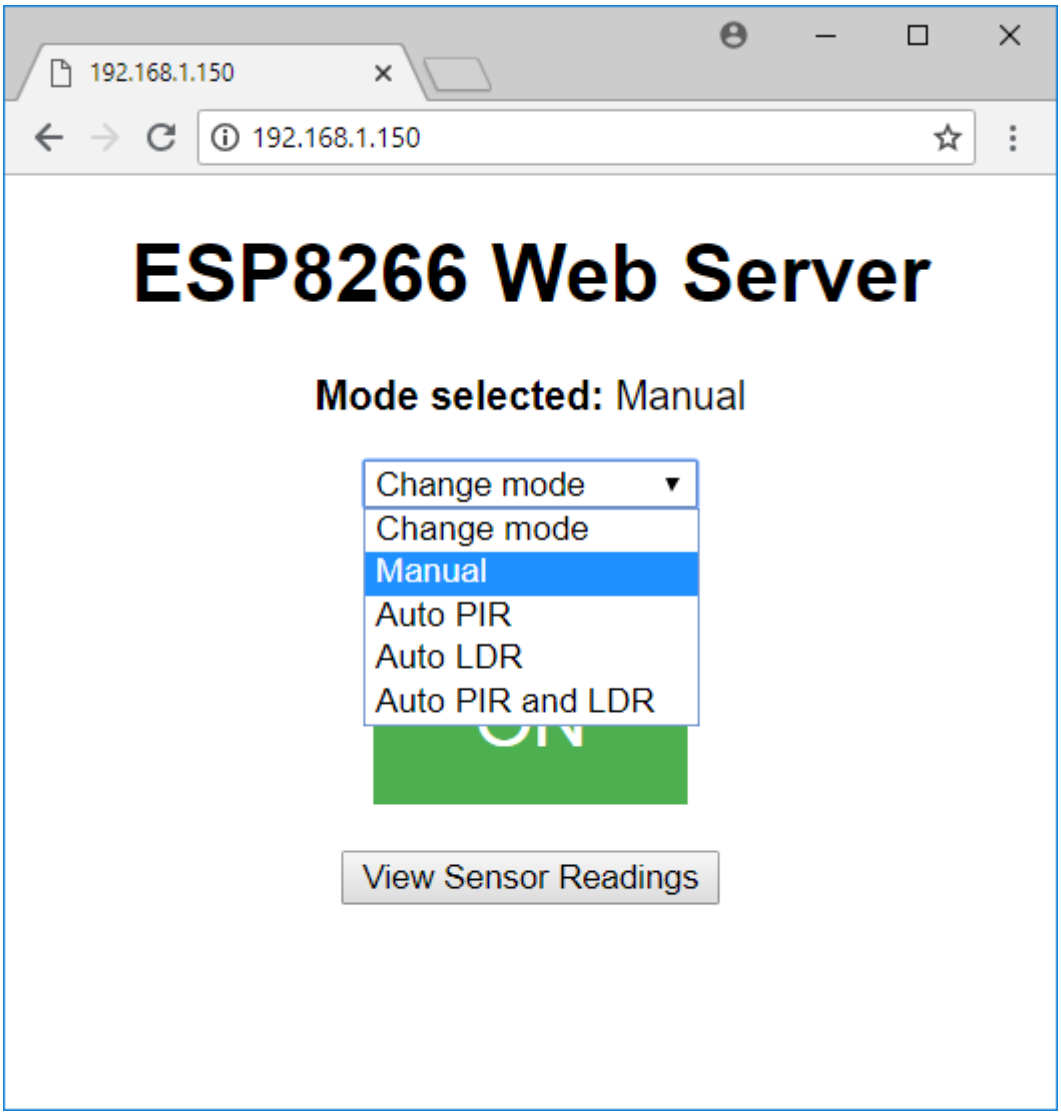
We've covered web servers in great detail in the ESP8266 Web Server Tutorial. So, we'll just take a look at the parts that are more relevant for this project.

This part of the code is easier to understand if we explain what's happening with a live demonstration.

When you access the web server, you'll see a similar web page.



At the top you can select one of these four different modes.



- Manual (mode 0)
- Auto PIR (mode 1)
- Auto LDR (mode 2)
- Auto PIR and LDR (mode 3)

Manual mode

For example, if you choose Manual mode, the following part of the code is being executed.

```
if(header.indexOf("GET /?mode=") >= 0) {
  pos1 = header.indexOf('=');
  pos2 = header.indexOf('&');
  valueString = header.substring(pos1+1, pos2);
  selectedMode = valueString.toInt();
  EEPROM.write(1, selectedMode);
  EEPROM.commit();
  configureMode();
}
```

It saves the selected mode in the *selectedMode* variable and stores it in the flash memory with:

```
EEPROM.write(1, selectedMode);
```

The web page look changes accordingly to the selected mode. In this case, since we've selected the manual mode that corresponds to 0, the following if statement is true and the web page will display two buttons to control the output.

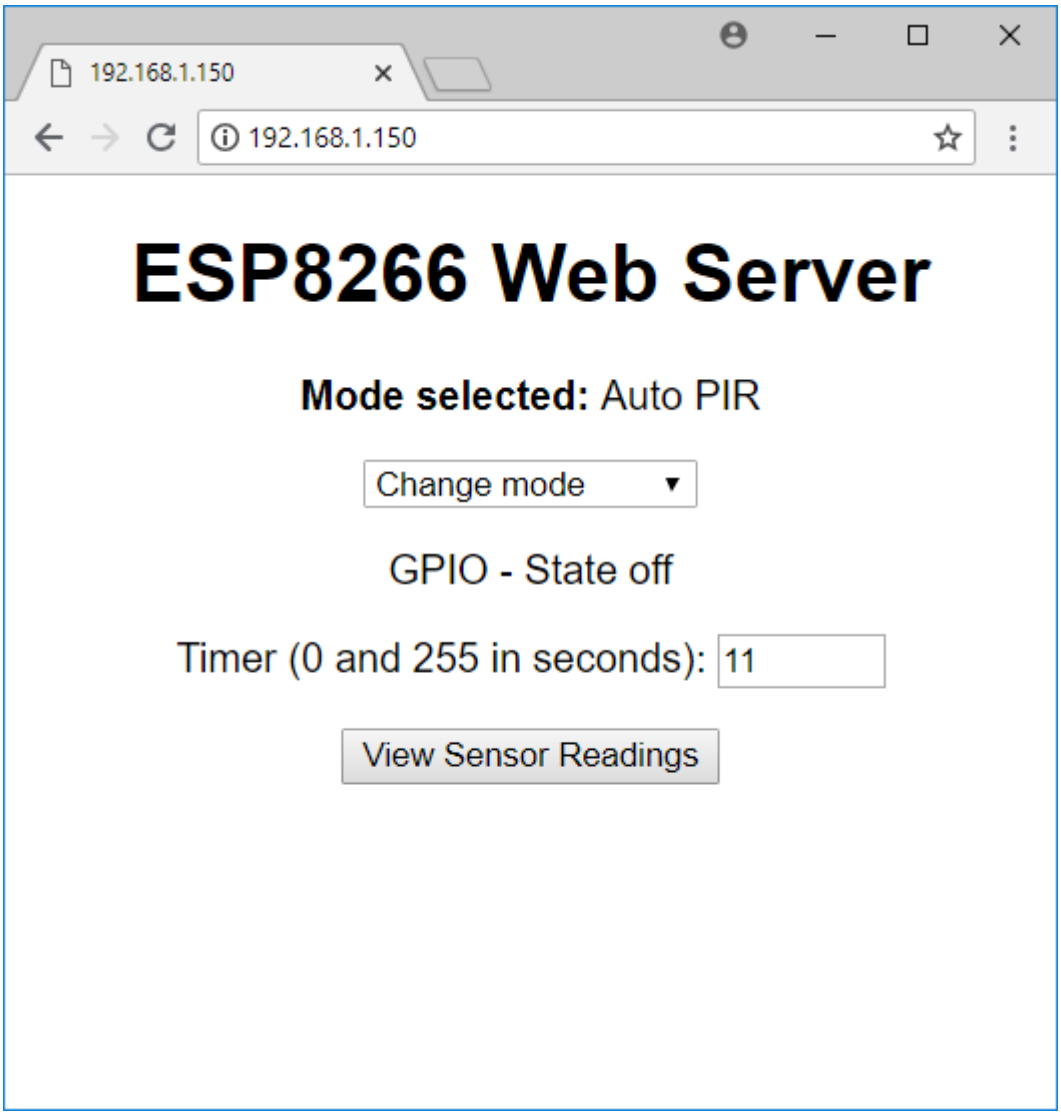
```
if(selectedMode == 0) {
  if(outputState == "off") {
    client.println("<p><button class=\"button\" onclick=\"outputOn()\">ON</button>
</p>");
  }
  else {
    client.println("<p><button class=\"button button2\"
onclick=\"outputOff()\">OFF</button></p>");
  }
}
```

When you click the on and off buttons, the following code runs and one of these two else if statements turns the output on or off.

```
// Change the output state - turn GPIOs on and off
else if(header.indexOf("GET /?state=on") >= 0) {
  outputOn();
}
else if(header.indexOf("GET /?state=off") >= 0) {
  outputOff();
}
```

Auto PIR mode

Now, in the drop-down menu select the Auto PIR mode.



There's a new input field that shows up in the web page. This field allows you to type an int number from 0 to 255 to specify the number of seconds the output should remain on after motion is detected.

When you change the number, it calls the following part of the code and changes the timer variable.

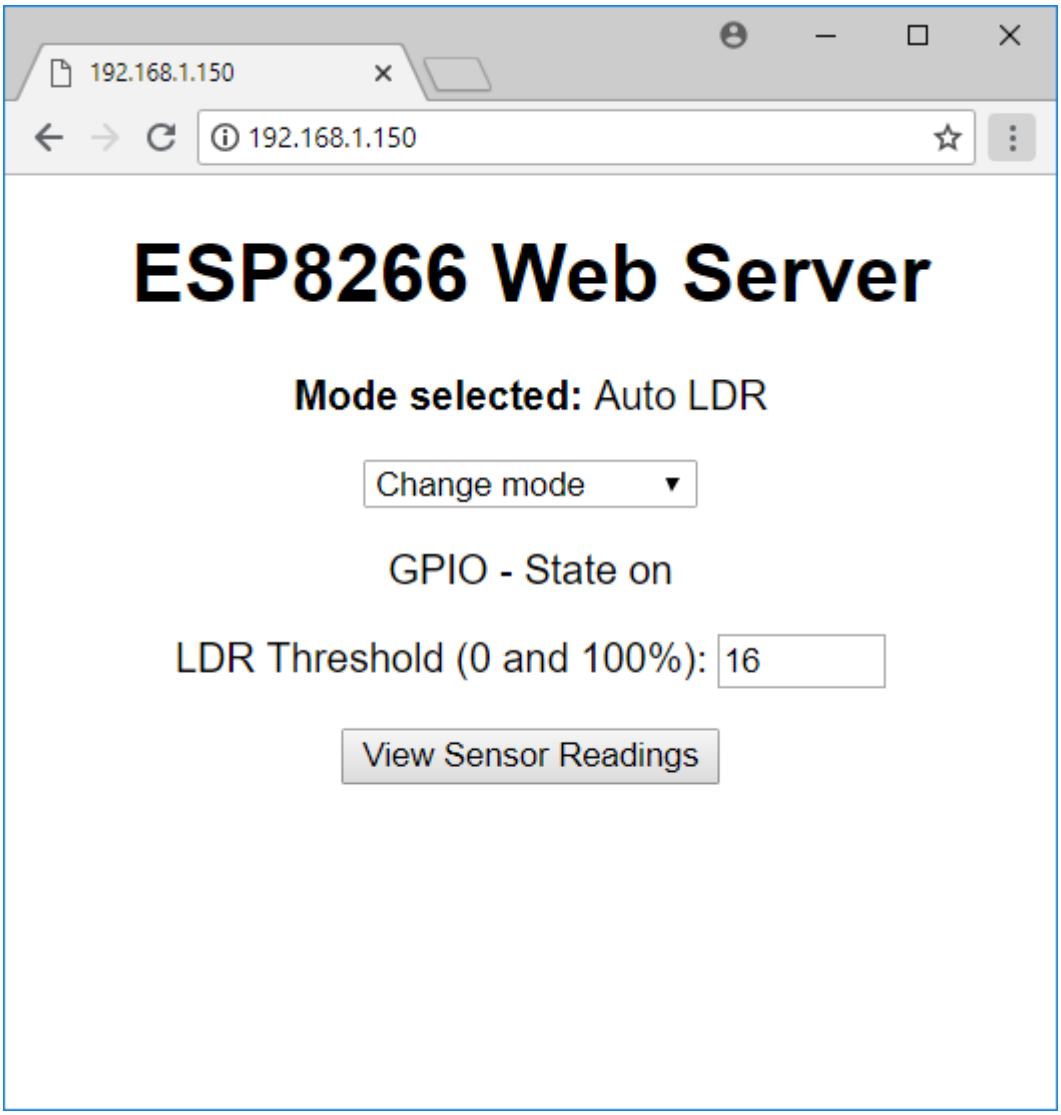
```
else if(header.indexOf("GET /?timer=") >= 0) {
  pos1 = header.indexOf('=');
  pos2 = header.indexOf('&');
  valueString = header.substring(pos1+1, pos2);
  timer = valueString.toInt();
  EEPROM.write(2, timer);
  EEPROM.commit();
  Serial.println(valueString);
}
```

In this mode (mode 1), it only displays the input field for the timer.

```
else if(selectedMode == 1) {
  client.println("<p>Timer (0 and 255 in seconds): <input type=\"number\"
name=\"txt\" value=\"" +
  String(EEPROM.read(2)) + "\" onchange=\"setTimer(this.value)\" min=\"0\"
max=\"255\"></p>");
}
```

Auto LDR mode

Select Auto LDR mode and a new input field appears.



This sets the LDR threshold value and you can enter a number between 0 and 100 to indicate the % of luminosity. When you change this field, it calls the following part of the code to update the LDR threshold value:

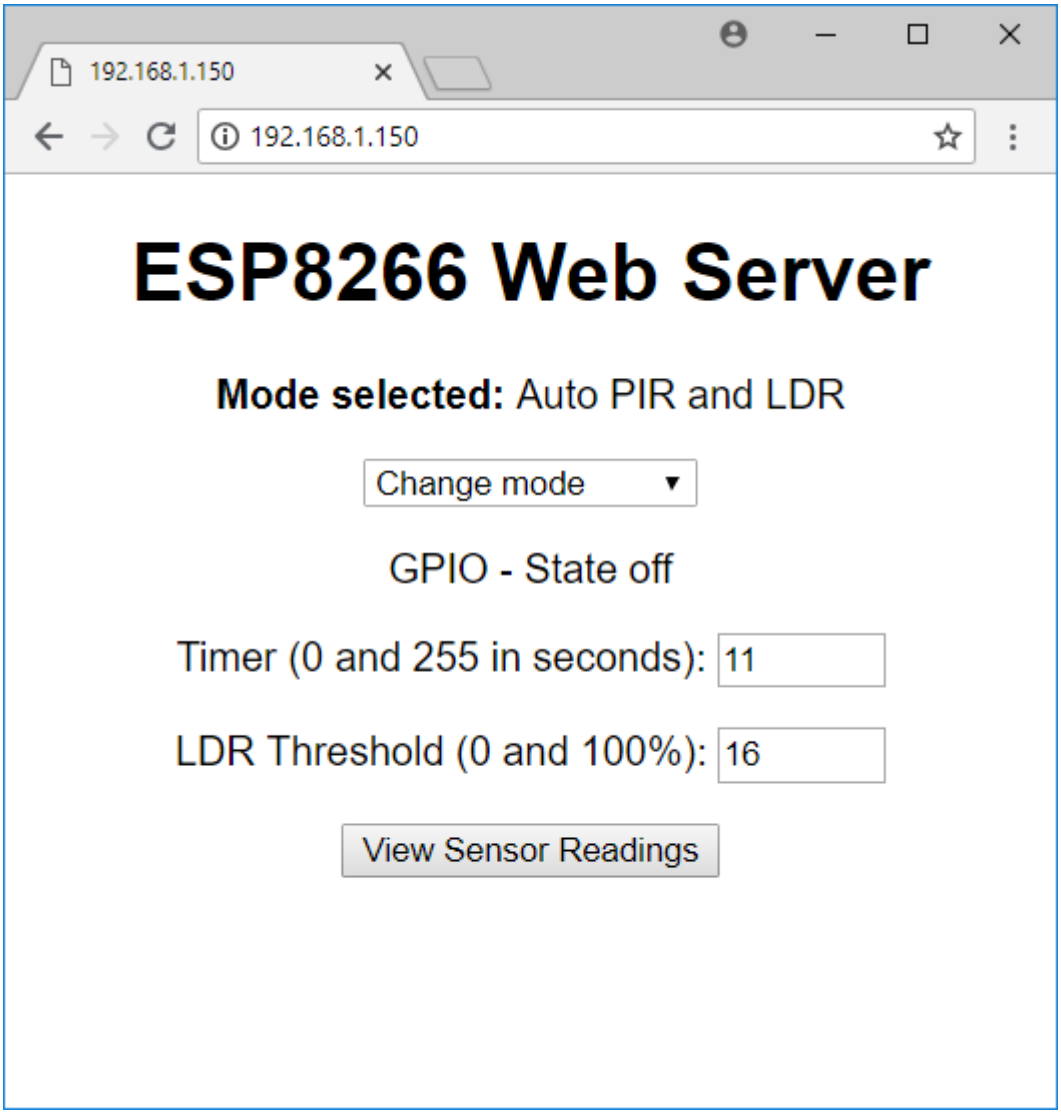
```
else if(header.indexOf("GET /?ldrthreshold=") >= 0) {
  pos1 = header.indexOf('=');
  pos2 = header.indexOf('&');
  valueString = header.substring(pos1+1, pos2);
  ldrThreshold = valueString.toInt();
  EEPROM.write(3, ldrThreshold);
  EEPROM.commit();
  Serial.println(valueString);
}
```

This is mode 2, and it will display the ldr theshold input field.

```
else if(selectedMode == 2) {
  client.println("<p>LDR Threshold (0 and 100%): <input type=\"number\"
name=\"txt\" value=\"" +
  String(EEPROM.read(3)) + "\" onchange=\"setThreshold(this.value)\" min=\"0\"
max=\"100\"></p>");
}
```

Auto PIR and LDR mode

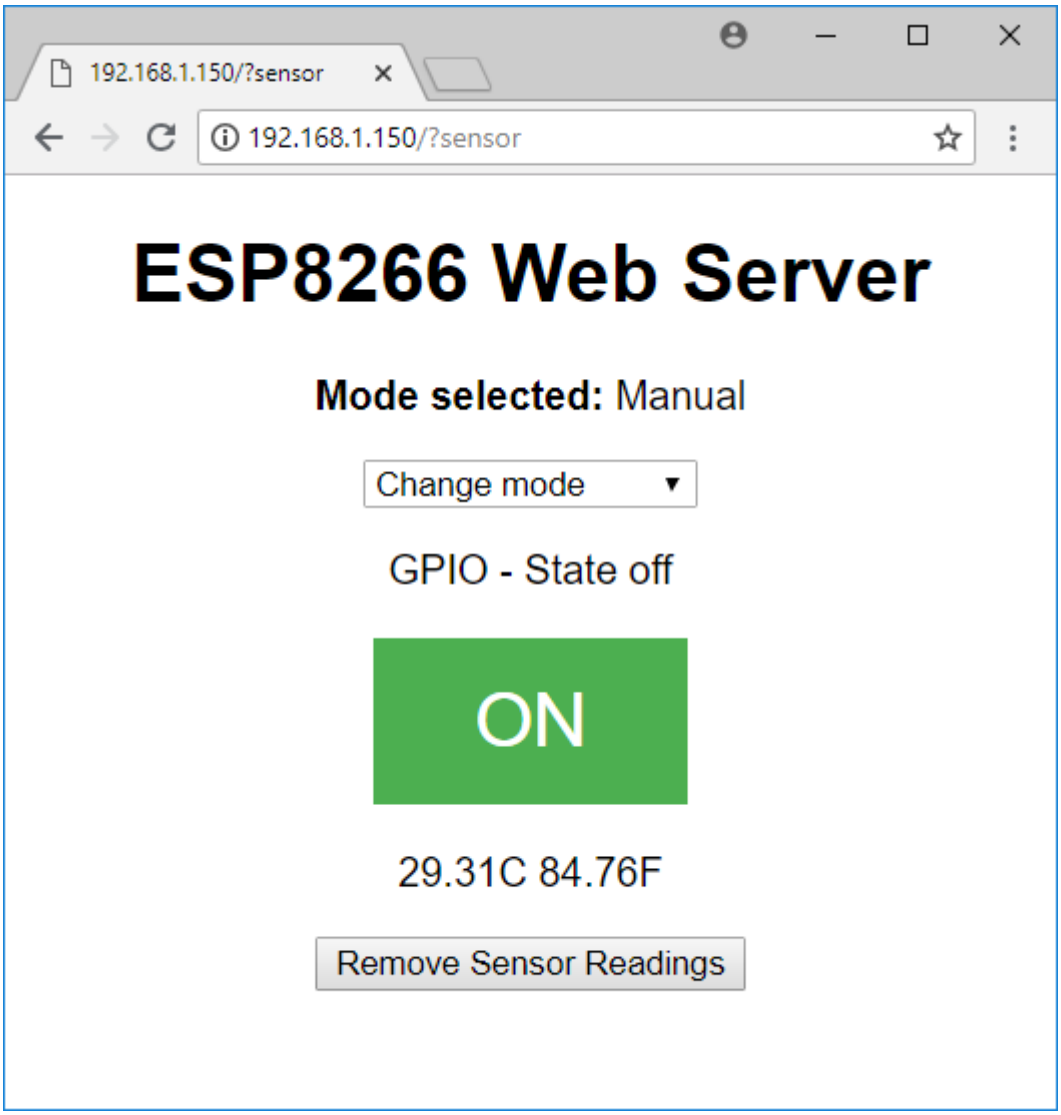
Selecting the Auto PIR and LDR mode activates both the PIR and LDR. It also loads a new web page with two input fields.



Both input fields work the same way as we've described earlier.

Sensor readings

Lastly, there's a button to request and display temperature readings.




```
if(header.indexOf("GET /?sensor") >= 0) {
  sensors.requestTemperatures();
  temperatureString = " " + String(sensors.getTempCByIndex(0)) + "C " +
  String(sensors.getTempFByIndex(0)) + "F";

  client.println("<p>");
  client.println(temperatureString);
  client.println("</p>");
```

There's also a button you can press to remove those readings.

```
client.println("<p><a href=\"/\"><button>Remove Sensor Readings</button></a>
</p>");
```

That's how you configure the settings of your multisensor. Then, accordingly to the mode and settings selected, another part of the *loop()* is running to check whether the output should be on or off.

Controlling the Output State

For example, when motion is detected, it calls the *detectsMovement()* function that starts a timer.

```
void detectsMovement() {
  if(armMotion || (armMotion && armLdr)) {
    Serial.println("MOTION DETECTED!!!");
    startTimer = true;
    lastMeasure = millis();
  }
}
```

Then, depending on the elapsed time, it turns the output on or off.

```
// Mode selected (1): Auto PIR
if(startTimer && armMotion && !armLdr) {
  if(outputState == "off") {
    outputOn();
  }
  else if((now - lastMeasure > (timer * 1000))) {
    outputOff();
    startTimer = false;
  }
}
```

There's also the following section of the code to turn the output on or off accordingly to the luminosity of the threshold value.

```
// Mode selected (2): Auto LDR
// Read current LDR value and turn the output accordingly
if(armLdr && !armMotion) {
  int ldrValue = map(analogRead(ldr), 0, 1024, 0, 100);
  //Serial.println(ldrValue);
  if(ldrValue > ldrThreshold && outputState == "on") {
    outputOff();
  }
  else if(ldrValue < ldrThreshold && outputState == "off") {
    outputOn();
  }
  delay(100);
}
```

Finally, the following snippet of code runs when the auto PIR and LDR mode is selected and motion is detected.

```
// Mode selected (3): Auto PIR and LDR
if(startTimer && armMotion && armLdr) {
  int ldrValue = map(analogRead(ldr), 0, 4095, 0, 100);
  //Serial.println(ldrValue);
  if(ldrValue > ldrThreshold) {
    outputOff();
    startTimer = false;
  }
  else if(ldrValue < ldrThreshold && outputState == "off") {
    outputOn();
  }
  else if(now - lastMeasure > (timer * 1000)) {
    outputOff();
    startTimer = false;
  }
}
```

That's pretty much how the code works, we've also put an effort to write a bunch of comments in the code to make it easier to understand.

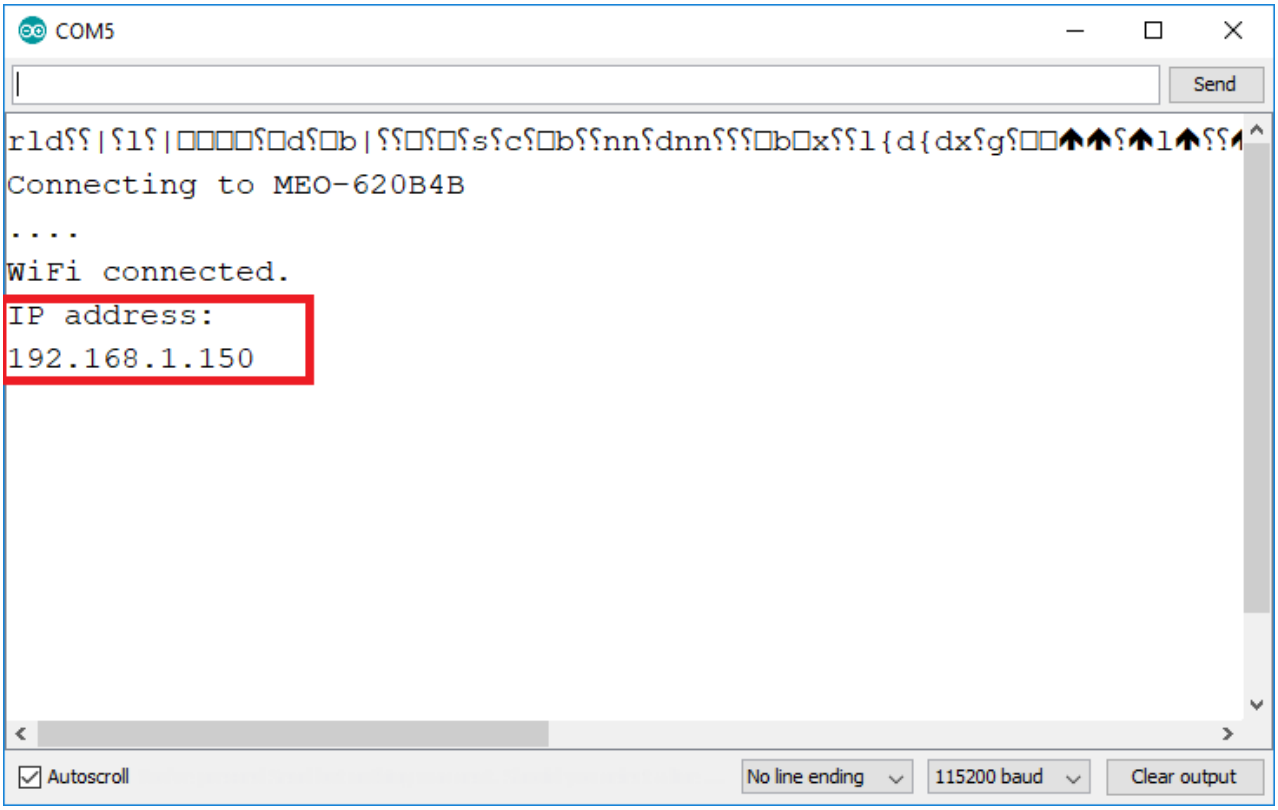
We've programmed the WeMos Multisensor shield with this code, but you can write your own code to integrate with any home automation platform. You just need to take into account the pin assignment of the multisensor shield.

Upload the Code

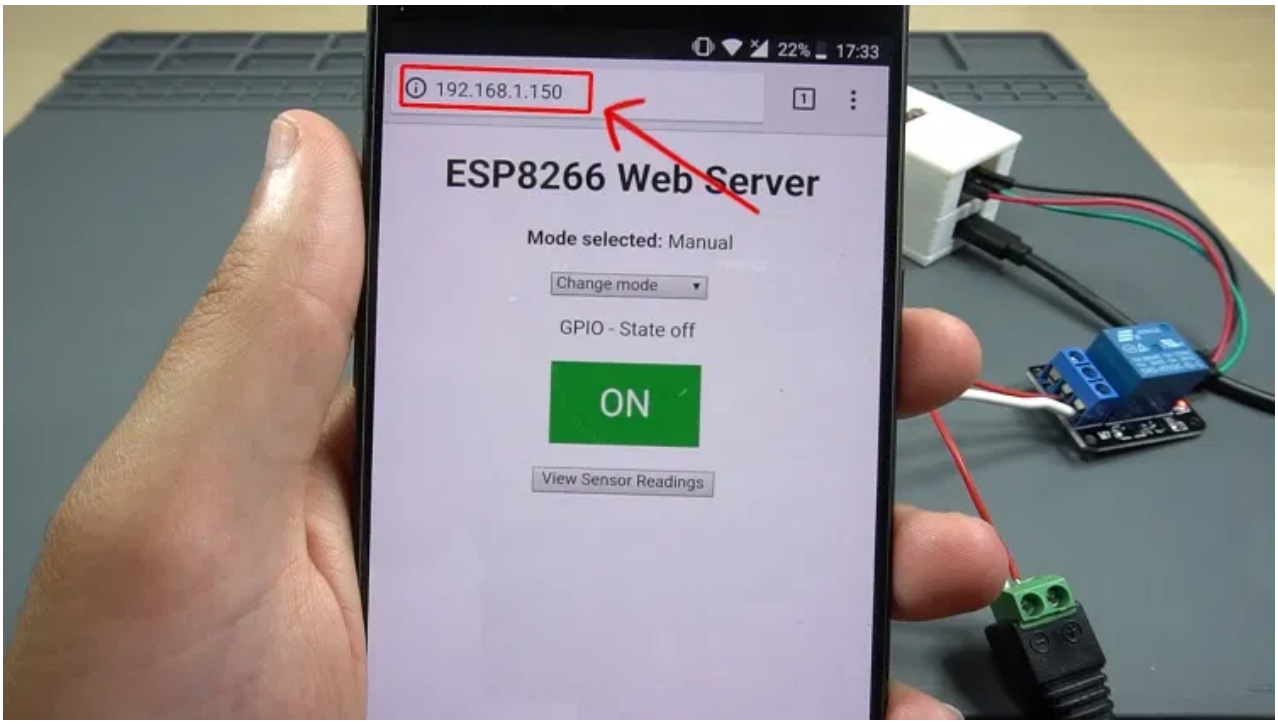
Click the upload button to upload the code to your ESP8266. Make sure you have the right board and COM port selected.

Testing the Multisensor Shield

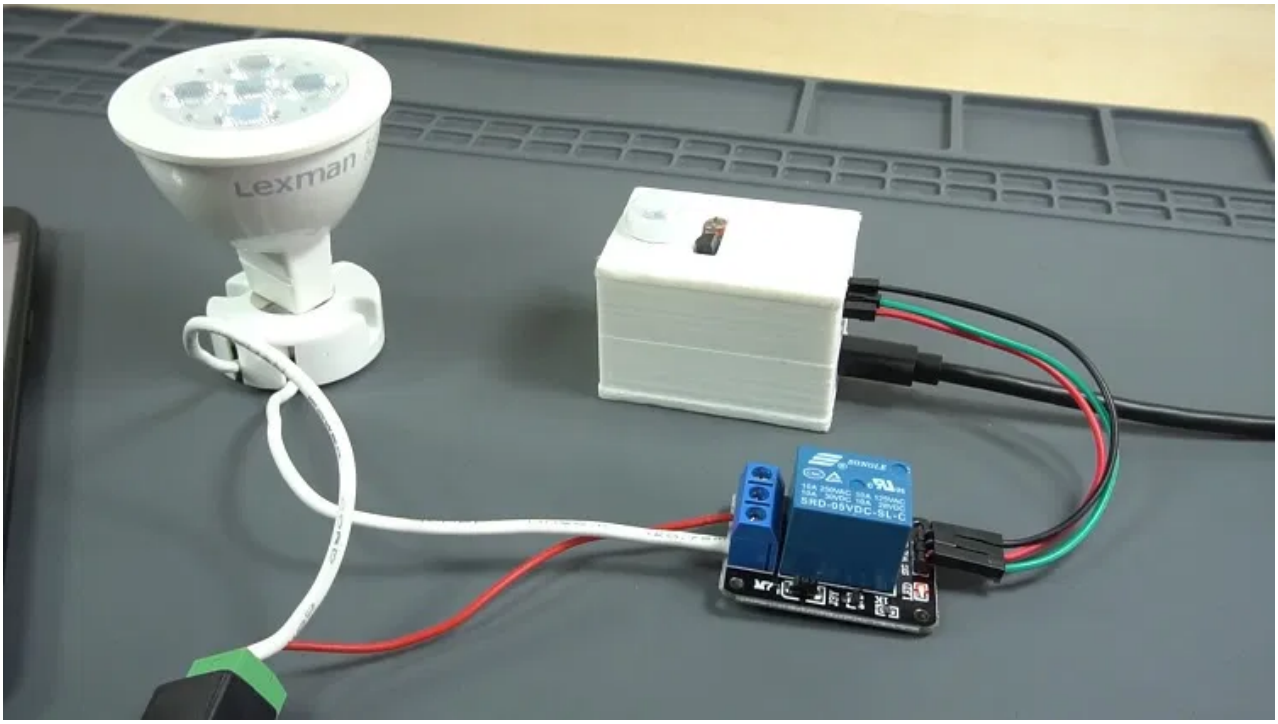
Open the Serial Monitor at a baud rate of 112500. Press the ESP8266 enable button to print the ESP IP address.



Open your browser and type the ESP8266 IP address. The following page should load.

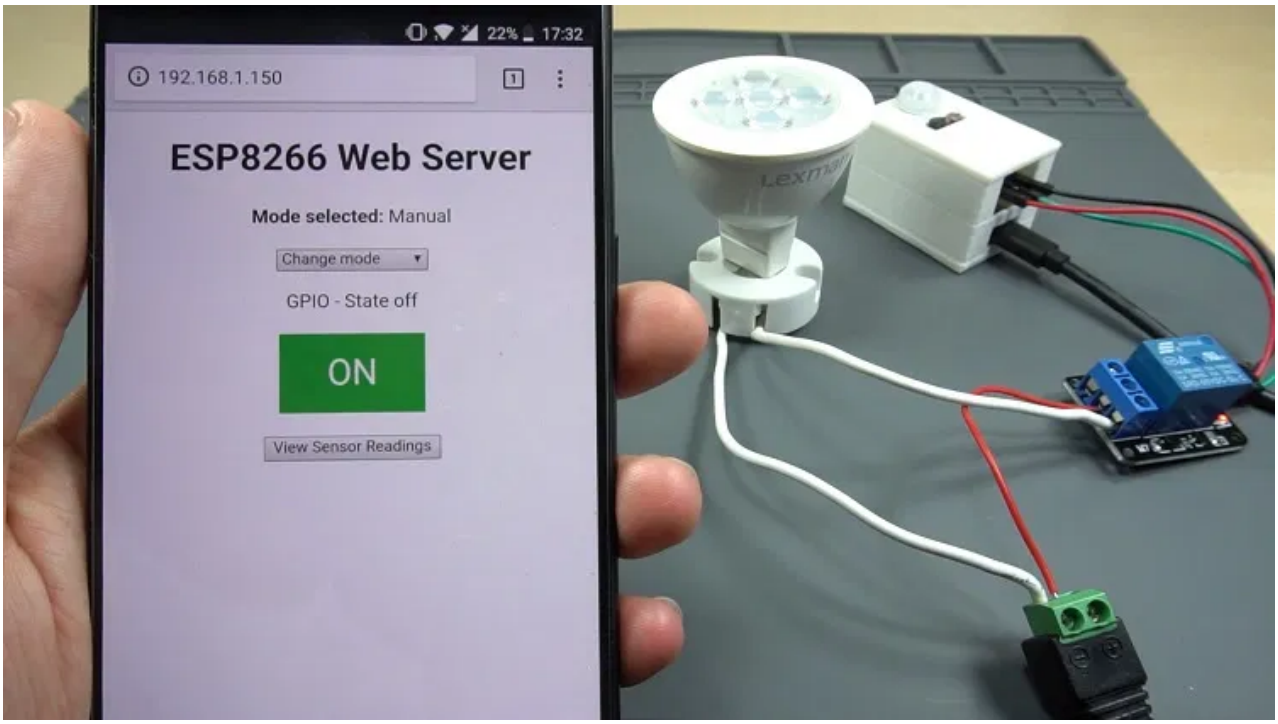


We've connected a relay module to the output terminal socket, so we're controlling a 12V lamp, but you can control any output that you want.

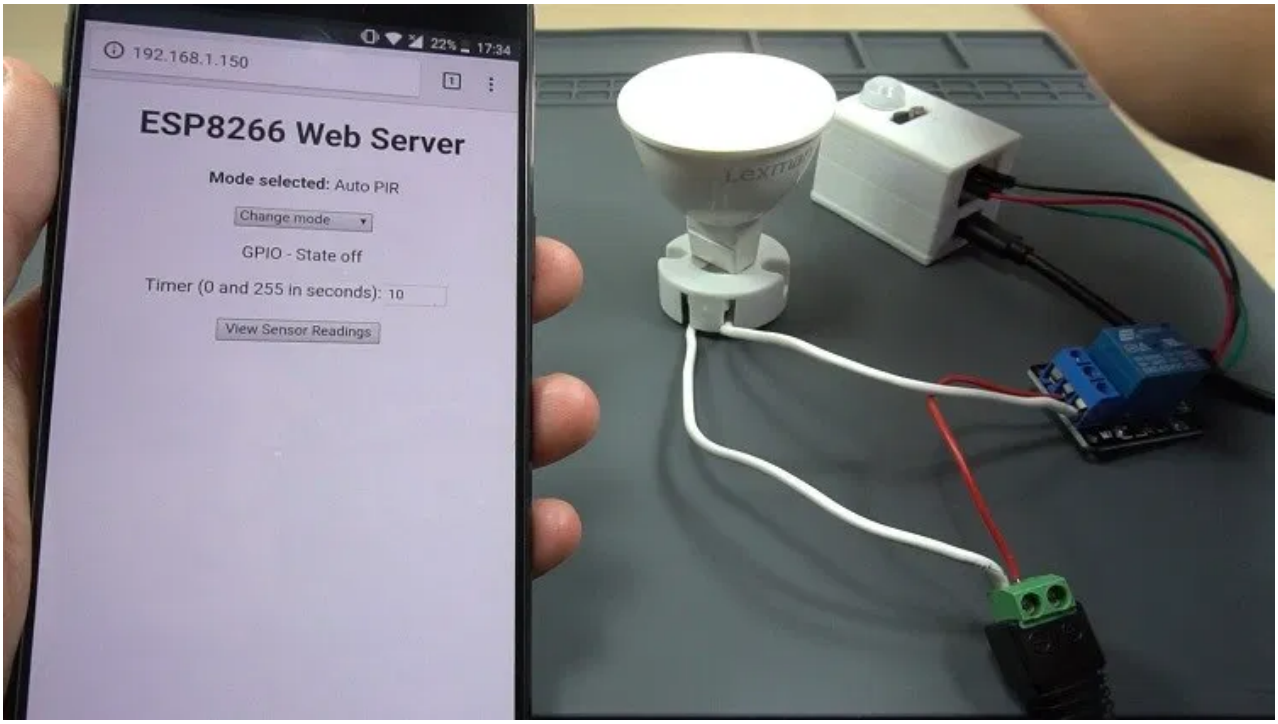


Now, select each mode, try to set different settings to check if everything is working properly.

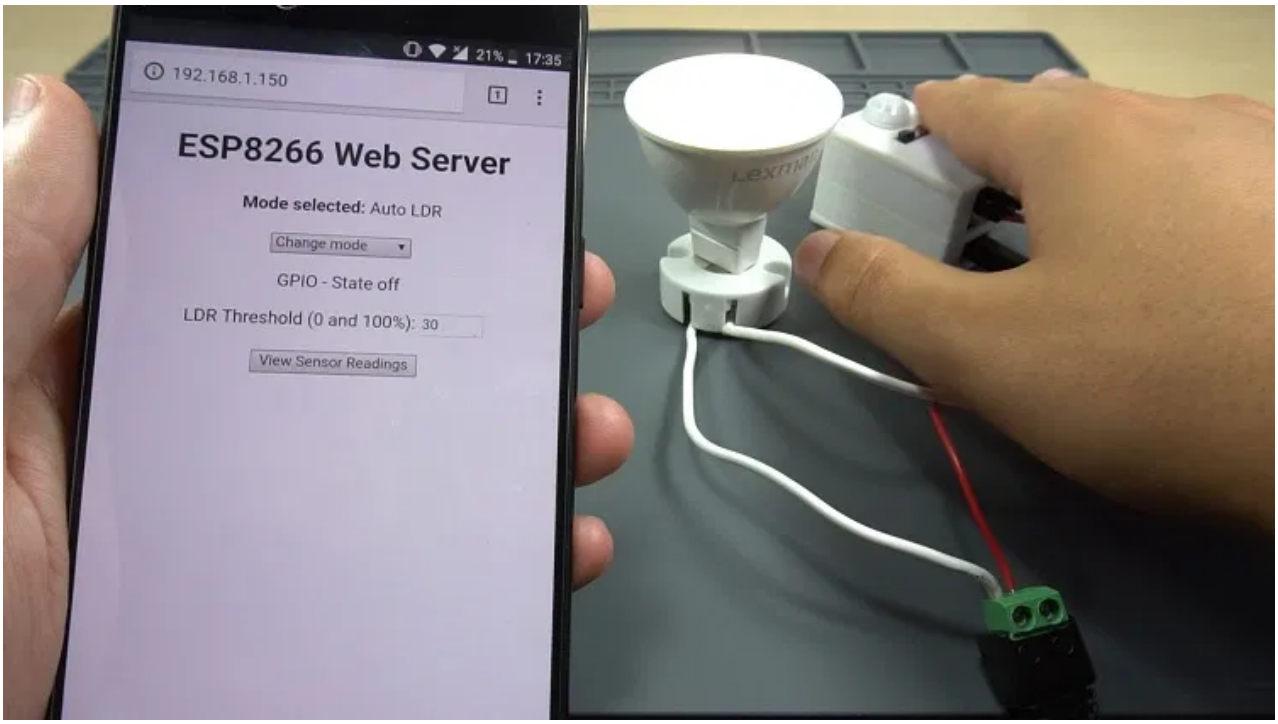
For example, select Manual mode and turn the lamp on and off.



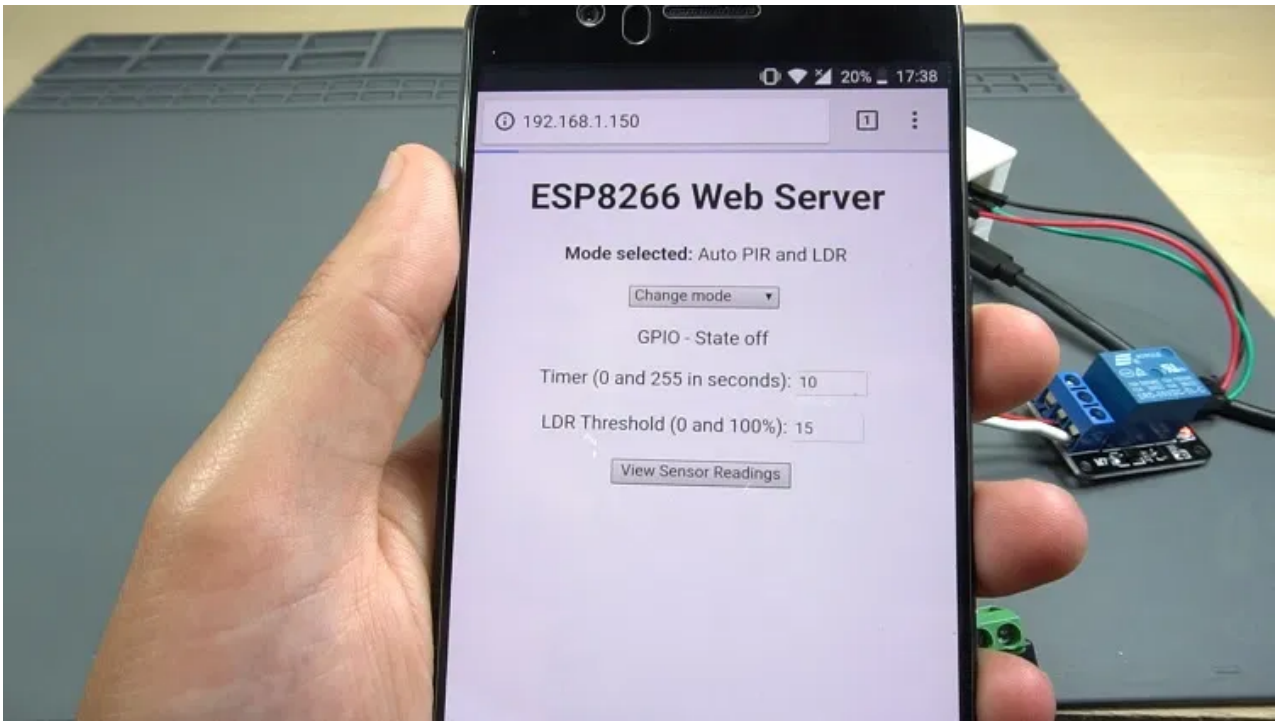
Select the Auto PIR mode. In this mode the lamp turns on for the number of seconds you set, when motion is detected.



On the LDR mode, you can set the threshold value that will make the lamp light up. When I cover the LDR, the luminosity goes below the threshold, and the lamp lights up.

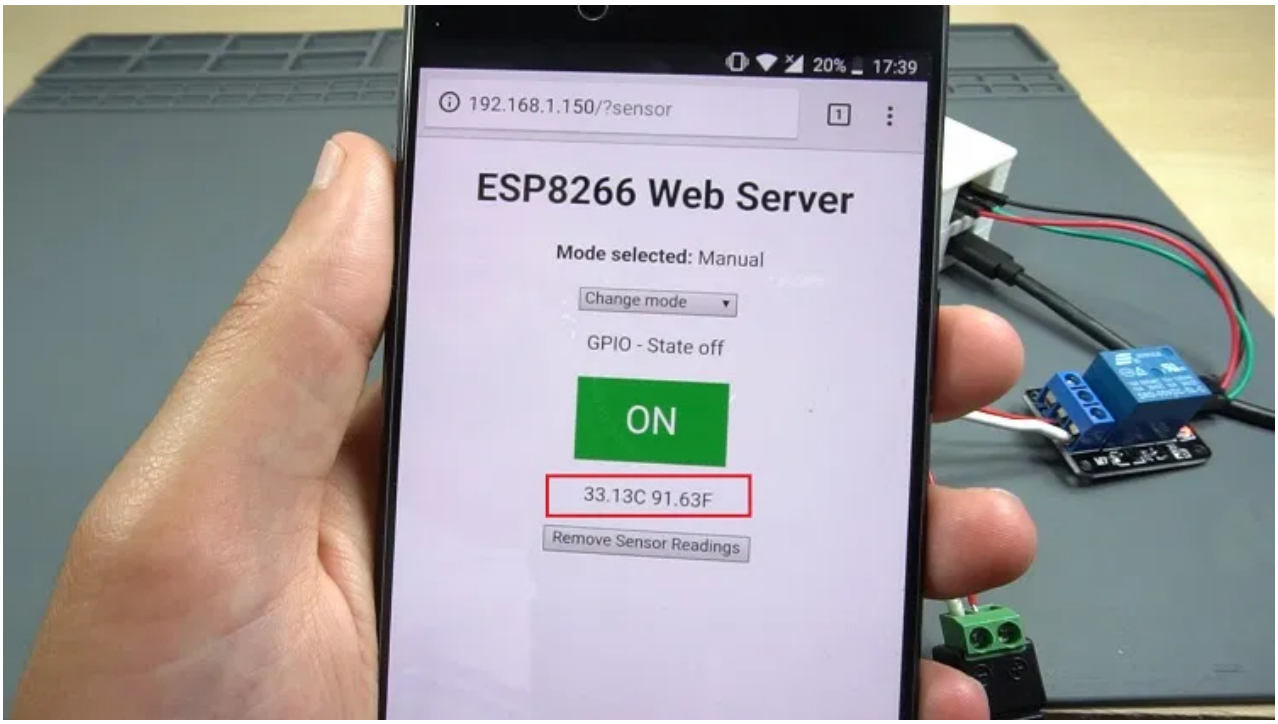


In Auto PIR and LDR mode, I can set the timer and the LDR threshold.



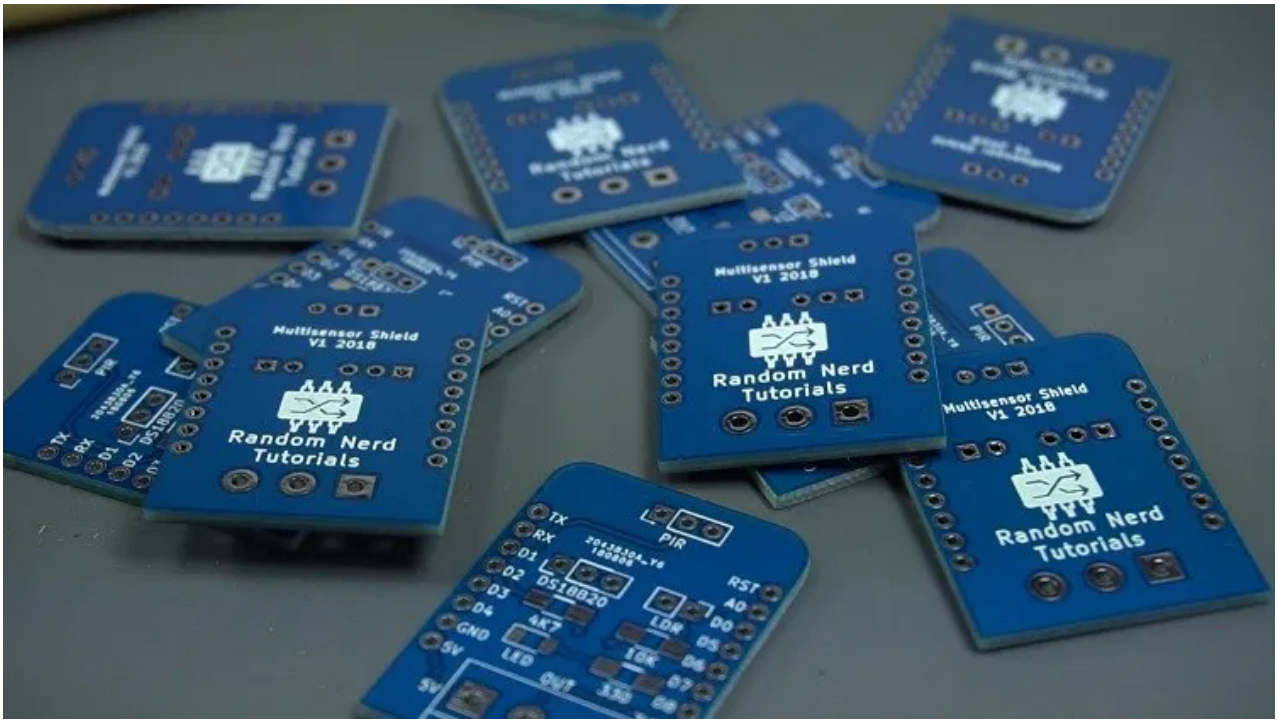
If motion is detected, but the light intensity is above the threshold, nothing happens. But if I cover the LDR, which means there's no light, and motion is detected, the lamp turns on for the number of seconds I've defined in the settings.

You can also click the "View Sensor Readings" button to request the latest temperature readings.



Wrapping Up

We're giving away 5 bare PCBs to someone that posts a comment below! Simply post a comment in this blog post about what you would like to do with the PCB and you're entered for a chance to win one of these bare PCBs. The winners will be announced next week! So, stay tuned! **[Update] the giveaway ended and the winners are: Ram Chivukula, Phillip R. Hickman, Douglas W Murray, Terry Wegener, and Dale Wolver.**



That’s it for this project. We hope you’ve found this project useful and you’re able to build it yourself. You can program the Multisensor Shield with other code suitable for your needs. For example, you can control the output based on the current temperature value. You can also edit the gerber files and add other features to the ESP8266 Multisensor Shield.

If you like this project you may also like other related projects:

Thanks for reading.