Home Automation
with the ESP8266

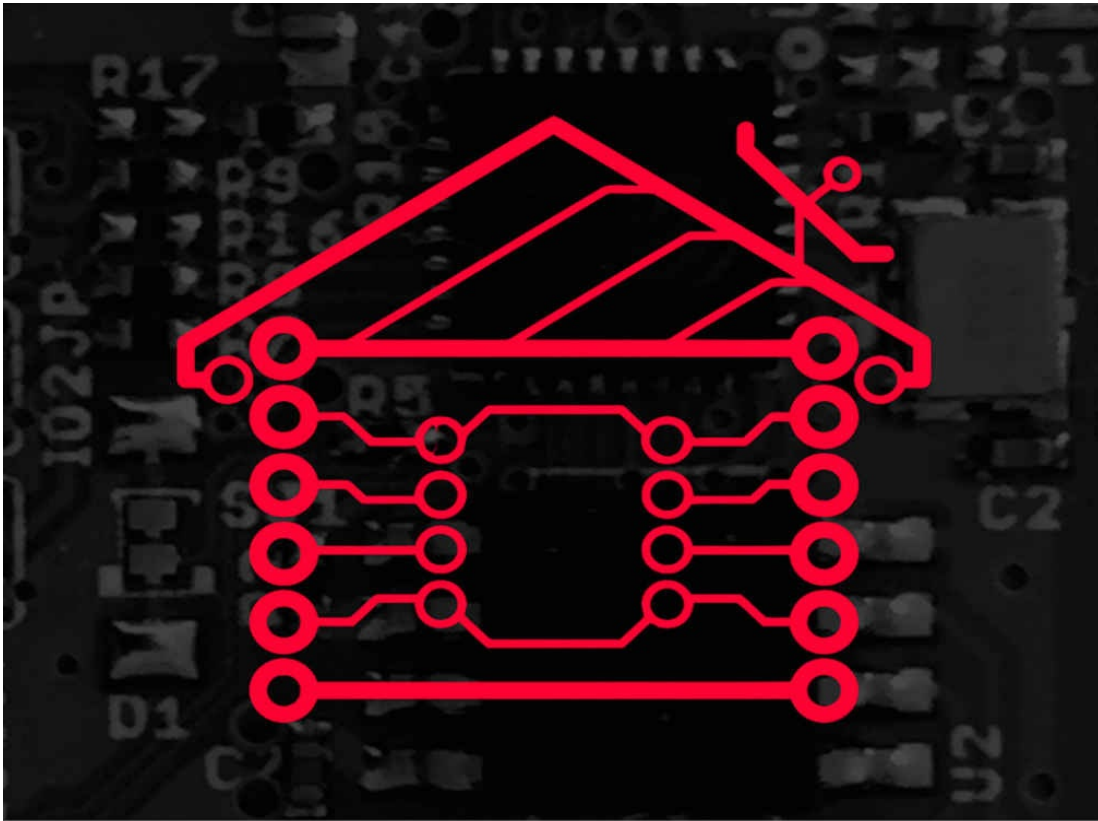Build Home Automation Systems
Using The Powerful & Cheap
ESP8266 WiFi Chip

Open Home Automation   Marco Schwartz

Home Automation
with the ESP8266

Build Home Automation Systems Using The Powerful & Cheap ESP8266 WiFi Chip

Open Home Automation   Marco Schwartz

Home Automation With the ESP8266

Build Home Automation Systems Using the Powerful and Cheap ESP8266 WiFi Chip

Marco Schwartz, PhD

Contents

Chapter 1   Introduction

Chapter 2   ESP8266 Hardware Configuration

Chapter 4   First Projects With the ESP8266

Chapter 6   Control a Lamp Remotely

Chapter 8   Build an Home Automation System Based on the ESP8266

Chapter 9   Conclusion

Legal

First eBook edition: April 2015

About the author

I am Marco Schwartz, and I am an electrical engineer, entrepreneur and author. I have a Master's degree in Electrical Engineering & Computer Science from one of the top Electrical Engineering school in France, and a Master's degree in Micro Engineering from the EPFL university in Switzerland.

I have more than 5 years of experience working in the domain of electrical engineering. My interests gravitate around electronics, home automation, the Arduino platform, open-source hardware projects, and 3D printing.

Since 2011 I have been working full-time as an entrepreneur, running websites with information about open-source hardware and building my own open-source hardware products.

About the companion website

This book has a companion website, Open Home Automation, which you can easily find by going at http://www.openhomeautomation.net. On this website you will find even more projects and resources around home automation and open-source hardware.

All the code that can be found in this book can also be accessed online at https://github.com/openhomeautomation/home-automation-esp8266. This GitHub repository for the book contains all the latest up-to-date code for all the projects you will find in this book.

Chapter 1

Introduction

I first heard about the ESP8266 WiFi chip in 2014. When I heard about it, I was building DIY home automation systems already, mainly using the Arduino platform. At first, I was reluctant to use this chip: yes, it was cheap ($5), but it was very difficult to use and to connect to Arduino for example.

However, everything changed in 2015 with the introduction of a modified version of the Arduino IDE for the ESP8266. With this software, not only it was really easy to use the ESP8266, but it also worked with the onboard processor of the ESP8266, therefore removing the need of an Arduino board next to the chip.

This made it the perfect base to build DIY home automation systems. WiFi is very convenient to use to build connected applications in your home, and at this price, it is a no-brainer to use it in every device in your home. In this book, I will show you how you can do that.

We are going to start by setting up the ESP8266 chip. You will learn how to choose the right module for your project, and get all the additional hardware you need to use the chip. We will also see how to connect the ESP8266 to your computer so you can program it using an USB cable.

Then, we are going to see how to configure & upload code to the ESP8266 chip. For that, we will be using a modified version of the Arduino IDE. This makes using the ESP8266 so much easier, as we will be using a well-known interface & language to configure the chip. We will also be able to use most of the already existing Arduino libraries for our projects.

Then, we will make three simple projects using the ESP8266 chip that you just configured, so you can learn the basics of this chip.

After that, we'll dive into home automation with the ESP8266 WiFi chip: we will connect a temperature & humidity sensor to your ESP8266 WiFi chip, and then display the measurements on a web server ran on the ESP8266 itself! This will allow you to access

measurements from anywhere in your home, for example on your smartphone.

Then, we are going to see how to control a lamp remotely via WiFi. We will also host the interface on the ESP8266 itself, to build a completely autonomous lamp controller. We will also make the interface responsive, so it can be used easily from a smartphone or tablet.

After that, we will connect a motion sensor to the ESP8266, and create a simple alarm system with your computer as the central alarm controller.

Finally, we will use everything we learned in the book to create a small home automation system based on the ESP8266 WiFi chip. We will learn how to integrate several components into a central interface, so you can control everything from the same place.

You don't need to have previous experience with the ESP8266 chip to use the contents of this book. However, a basic knowledge of electronics and programming is required. If you already know a bit about Arduino, it will also help you to understand the examples of this book.

Chapter 2

ESP8266 Hardware Configuration

In this chapter, we are going to see how to set up the ESP8266 so it is correctly powered. We are also going to connect it to your computer so you can program it. Finally, we are going to see how to choose the right module for your project.

2.1   How to Choose Your ESP8266 Module

We are first going to see how to choose the right ESP8266 module for your project. Indeed, there are many modules available on the market, and it is quite easy to get lost between all the choices.

The first one that you probably heard of is the small ESP8266 Serial Wireless Transceiver module:
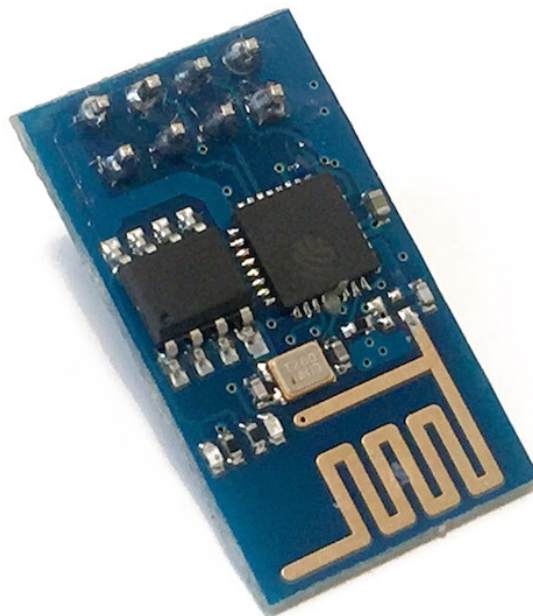
This module is the most famous one, as it is really small and only costs $5. However, the number of accessible GPIO pins (input/output pins) are quite limited. It is also difficult to plug it on a standard breadboard.

If you choose this module, there are some projects of this book that you might not be able to do. For example, you won't be able to do the projects using analog sensors as the analog input pin is not accessible.

But there are many other modules on the market, that gives you access to all the pins of the ESP8266. For example, I really like the ESP8266 Olimex module which is also cheap (around $10):
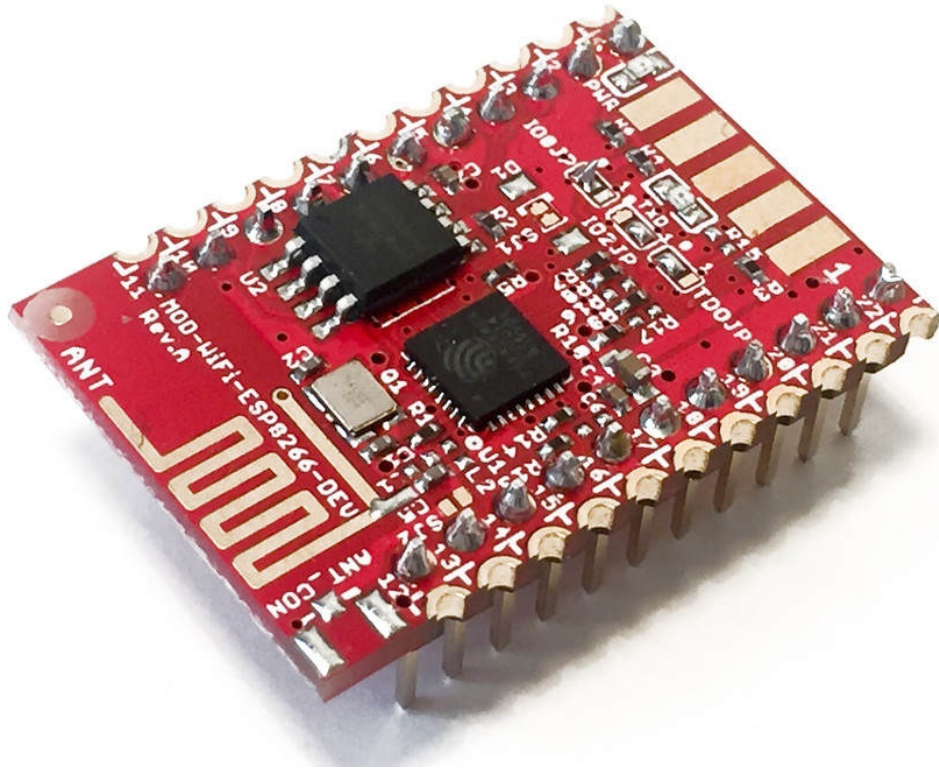
This module can easily be mounted on a breadboard, and you can easily access all the pins of the ESP8266. This is the one I will use for the rest of this book, and therefore I also recommend that you use a similar module. You can also use modules based on the ESP-12 board, which is very similar to the Olimex module.

Another solution is to use the NodeMCU development kit, which is similar to the Olimex board but also have an integrated USB to Serial converter, as well as an onboard power supply. It is even easier to use, but hard to find at the time this guide was written. You can get more information on the NodeMCU website:

http://nodemcu.com/index_en.html

2.2   Hardware Requirements

Let's now see what we need to make the ESP8266 chip work. Indeed, it is usually wrongly assumed that you just need this little chip and nothing else to make it work, and we are going to see that it is not true.

First, you will need some way to program the ESP8266. You can use an Arduino board for that, but for me the really great thing about the ESP8266 is that it can function completely autonomously, using the onboard processor.

So for to program the chip, I will use a USB FTDI programmer. Note that it has to be compatible with the logic level of the ESP8266 chip, so 3.3V. I used a module that can be switched between 3.3V and 5V:
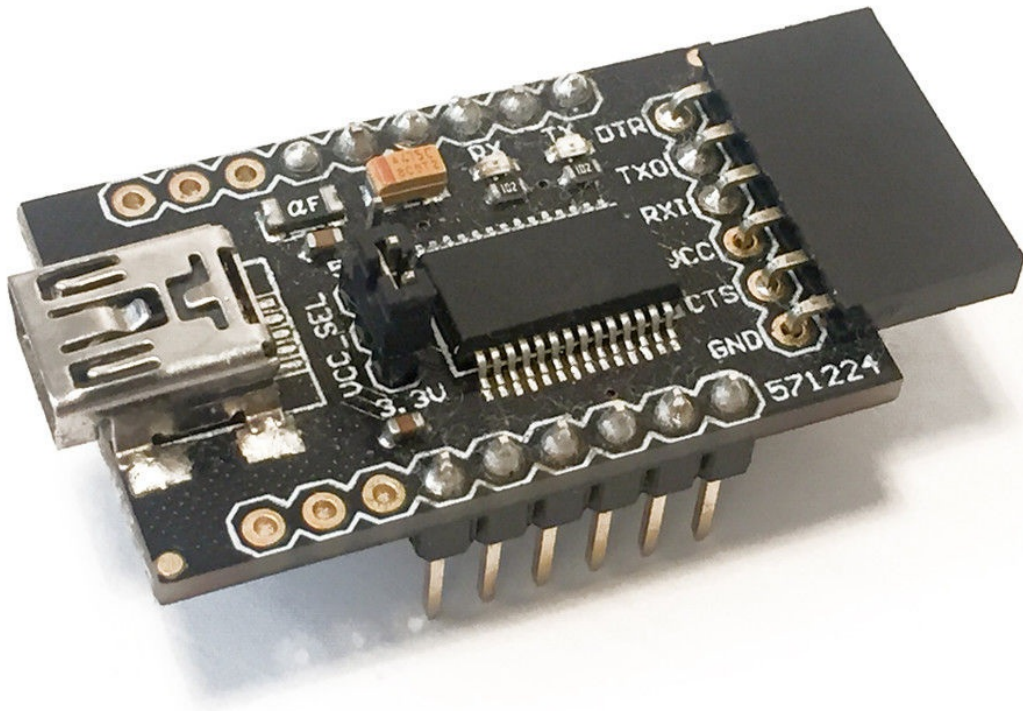
You will also need a dedicated power supply to power the chip. This is a point that is often forgotten and that leads to a lot of issues. Indeed, if you are for example trying to power the ESP8266 chip from the 3.3V coming from the FTDI board or from an Arduino board, it simply won't work correctly.

Therefore, you need a dedicated power supply that can deliver at least 300 mA to be safe. I used a breadboard power supply that can deliver up to 500 mA at 3.3V:
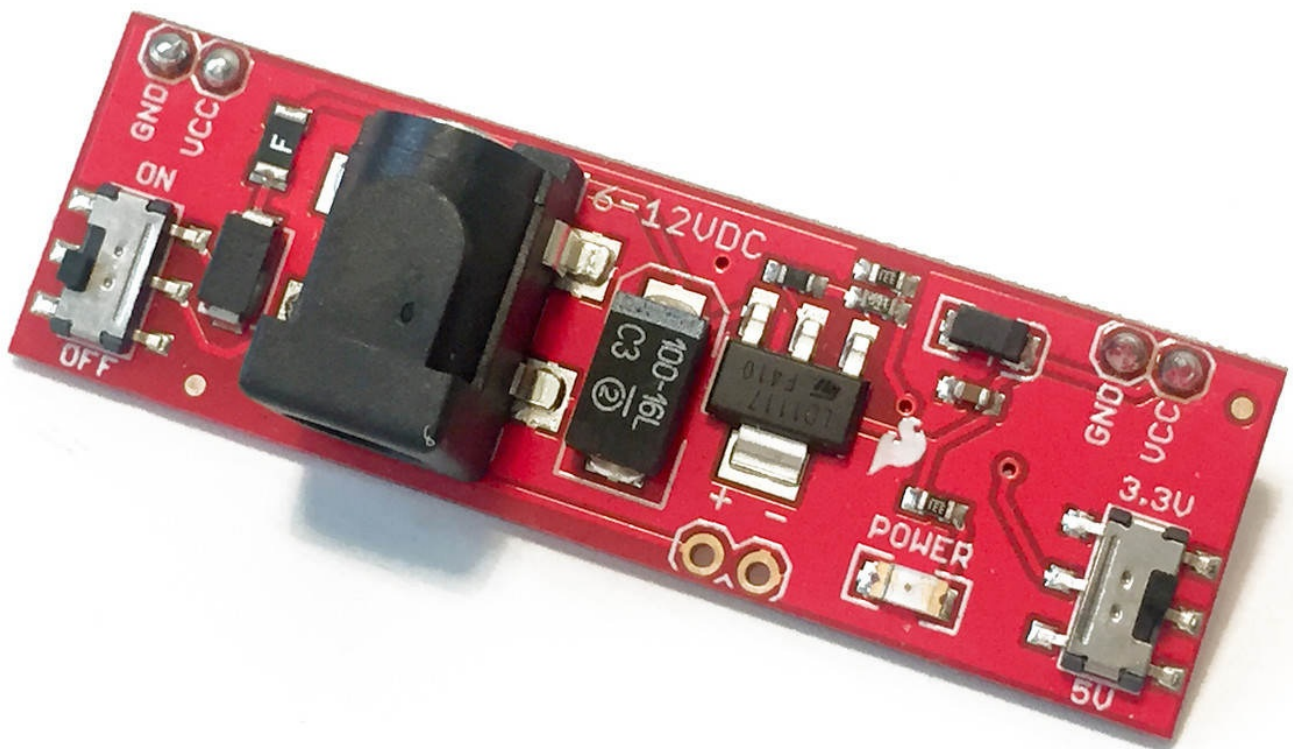
This is a list of all the components that you will need to use the ESP8266 chip:

- ESP8266 Olimex module (https://www.olimex.com/Products/IoT/MOD-WIFI-ESP8266-DEV/)
- Breadboard 3.3V power supply (https://www.sparkfun.com/products/13032)
- 3.3V FTDI USB module (https://www.sparkfun.com/products/9873)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/9194)

2.3 Hardware Configuration

We are now going to see how to configure the hardware for the first use of your ESP8266 board. This is how to connect the different components:
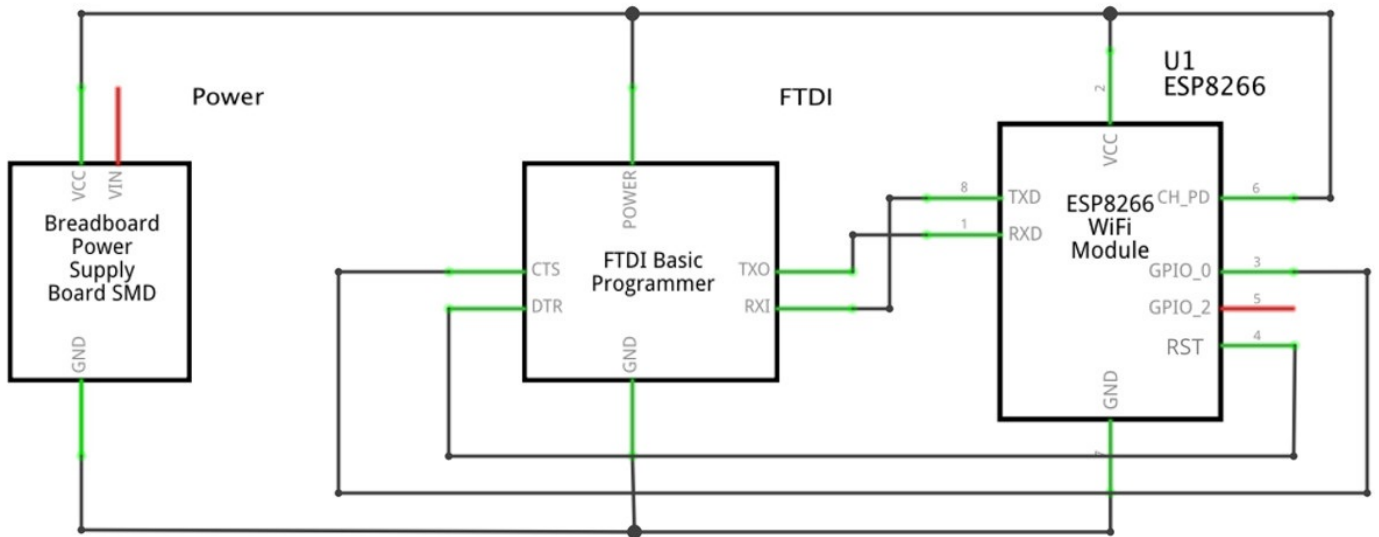
And this is how it will look like at the end:
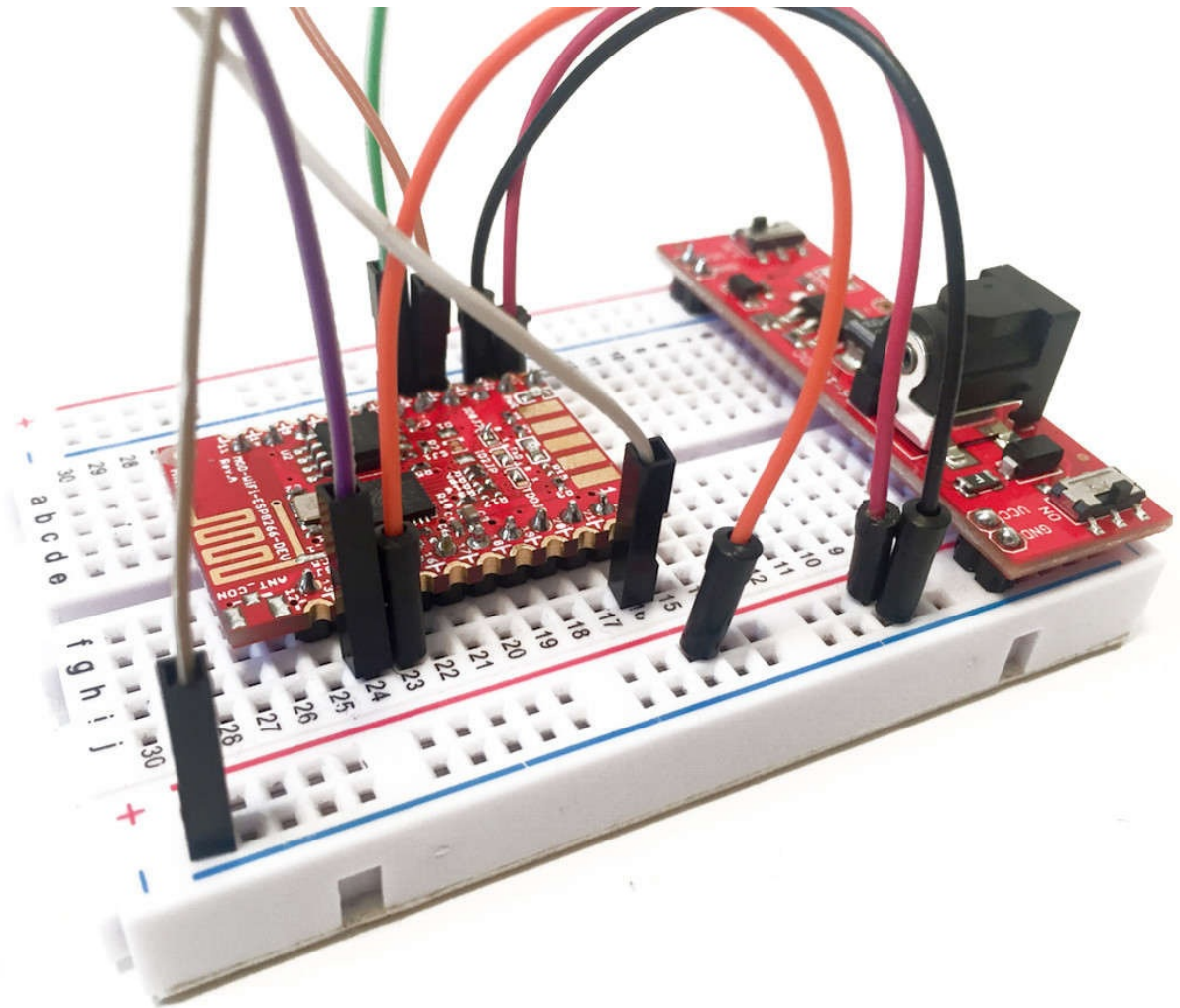
Make sure that you connected everything according to the schematics, or you won't be able to continue. Also make sure that all the switches of your components (FTDI module & power supply) are set to 3.3V, or it will damage your chip.

Your ESP8266 is now ready, and you can now move on to the next step: using the modified version of the Arduino IDE to configure your chip.

Chapter 3

Getting Started With the ESP8266 Arduino IDE

Now that we completely set up the hardware around the ESP8266, we are ready to configure it using a modified version of the Arduino IDE.

Indeed, the most basic way to use the ESP8266 module is to use serial commands, as the chip is basically a WiFi/Serial transceiver. However, this is not convenient, and this is not what I recommend doing.

What I recommend is using the very cool Arduino ESP8266 project, which is a modified version of the Arduino IDE that you need to install on your computer. This makes it very convenient to use the ESP8266 chip as we will be using the well-known Arduino IDE. This is therefore the method we will use in this whole book.

3.1   Installing the Arduino IDE for the ESP8266

We are now going to install the Arduino IDE that has been modified for the ESP8266. You can find all the details of the project on:

https://github.com/esp8266/Arduino

Installing the IDE is really easy as well. You just need to download the corresponding release for your platform, and then follow the onscreen instructions. You can get the latest release from:

https://github.com/esp8266/Arduino/releases

Once the IDE is installed, congratulations, you now have a ESP8266 chip ready to be used with the Arduino IDE!

3.2 Connecting Your Module to Your WiFi Network

Now, we are going to check that the Arduino IDE is correctly working, and connect your chip to your local WiFi network.

To do so, we need to write the code first, and then upload it to the board. The code is will be quite simple: we just want to connect to the local WiFi network, and print the IP address of the board. This is the code to connect to the network:

```cpp
// Import required libraries
#include <ESP8266WiFi.h>

// WiFi parameters
const char* ssid = "your_wifi_name";
const char* password = "your_wifi_password";

void setup(void)
{
  // Start Serial
  Serial.begin(115200);

  // Connect to WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  // Print the IP address
  Serial.println(WiFi.localIP());

}

void loop() {

}
```

You can simply copy the lines of code above, and copy them into the ESP8266 Arduino IDE that you downloaded before. Of course, put your own WiFi name & password in the code. Save this file with a name of your choice.

Now, also go in Tools>Boards, and select "Generic ESP8266 Module". Also select the correct Serial port that corresponds to the FTDI converter your are using.

After that, upload the code to the board, and open the Serial monitor when this is done. Also set the Serial monitor speed to 115200. You should see the following message:

```
WiFi connected
192.168.1.103
```

If you can see this message and an IP, congratulations, your board is now connected to

your WiFi network! You are now ready to build your first projects using the ESP8266 chip.

Chapter 4

First Projects With the ESP8266

Now that your ESP8266 chip is ready to be used & that we could connect it to your WiFi network, we can now build some basic projects with it. This will help you understand the basics of the ESP8266.

We are going to see three projects in this chapter: how to control a LED, how to read data from of a GPIO pin, and finally how to grab the contents from a web page.

4.1 Control a LED

First, we are going to see how to control a simple LED. This will teach you how to use the GPIO pins of the chip as outputs.

The first step is to add an LED to our project. These are the extra components you will need for this project:

- 5mm LED (https://www.sparkfun.com/products/9590)
- 330 Ohm resistor (https://www.sparkfun.com/products/8377)

The next step is to connect the LED with the resistor to the ESP8266 board. To do so, the first thing to do is to place the resistor on the breadboard. Then, place the LED on the breadboard as well, connecting the longest pin of the LED (the anode) to one pin the resistor.

Then, connect the other end of the resistor to the GPIO pin 5 of the ESP8266, and the other end of the LED to the ground.

This is how it should look like at the end:
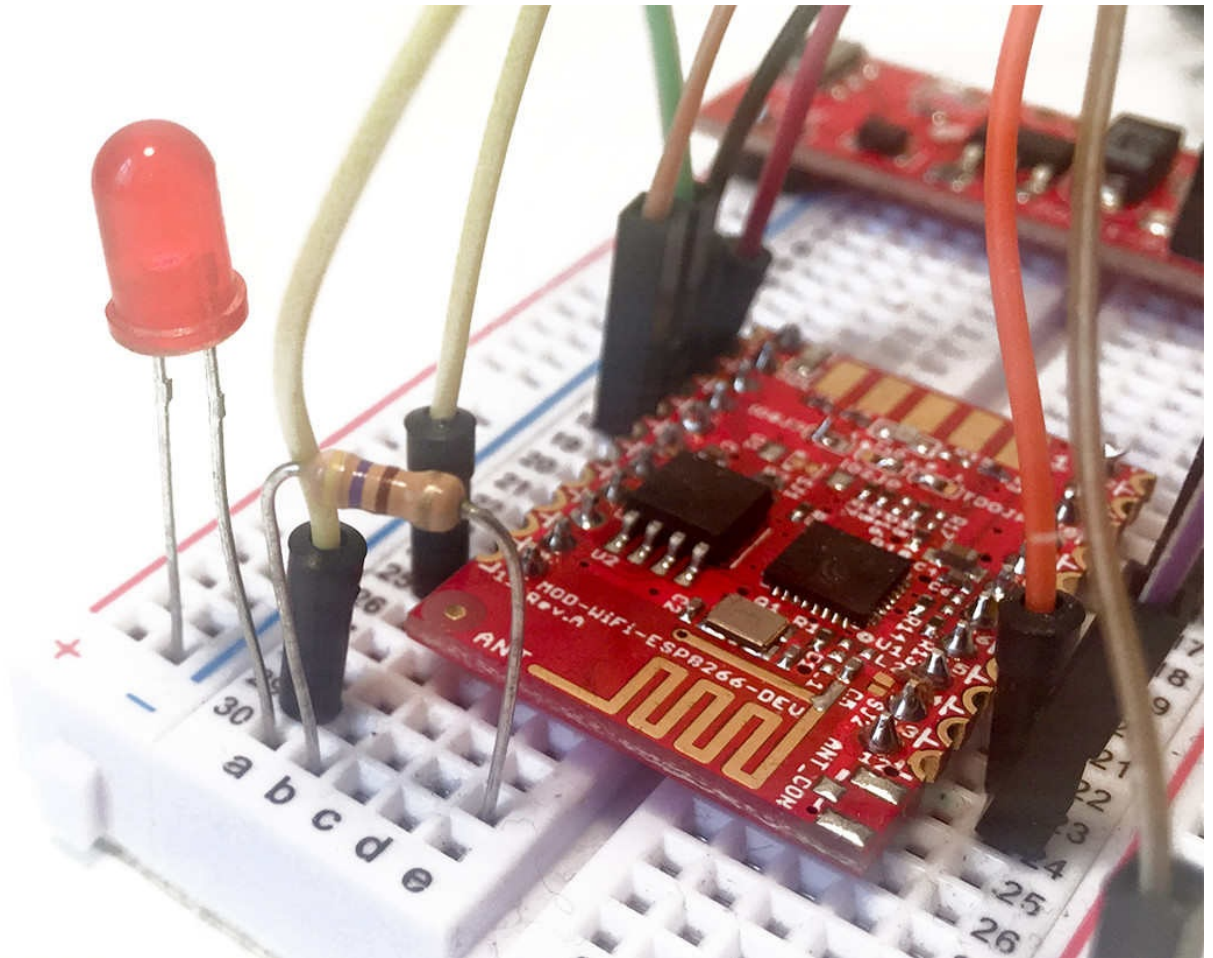
We are now going to light up the LED by programming the ESP8266 chip, just as we did before by connecting it to the WiFi network.

This is the complete code for this section:

```
// Import required libraries
#include <ESP8266WiFi.h>

void setup(void)
{

  // Set GPIO 5 as output
  pinMode(5, OUTPUT);

  // Set GPIO 5 on a HIGH state
  digitalWrite(5, HIGH);

}

void loop() {

}
```

This code simply sets the GPIO pin as an output, and then apply a HIGH state on it.

You can now copy this code and paste it in the Arduino IDE. Then, upload the code to the board. You should immediately see that the LED is lighting up. You can shut it down again by using `digitalWrite(5, LOW)` in the code.

4.2　Read Data From a GPIO Pin

As a second project in this chapter, we are going to read the state of a GPIO pin. For this, we will use the same pin as in the previous project. You can therefore remove the LED & the resistor that we used in the previous project.

Now, simply connect this pin (GPIO 5) of the board to the positive power supply on your breadboard with a wire.

Reading data from a pin is really simple. This is the complete code for this part:

```cpp
// Import required libraries
#include <ESP8266WiFi.h>

void setup(void)
{
  // Start Serial
  Serial.begin(115200);

  // Set GPIO 5 as input
  pinMode(5, INPUT);

}

void loop() {

  // Read GPIO 5 and print it on Serial port
  Serial.print("State of GPIO 5: ");
  Serial.println(digitalRead(5));

  // Wait 1 second
  delay(1000);

}
```

We simply set the pin as an input, and then read the value of this pin, and print it out every second. Copy & paste this code into the Arduino IDE, then upload it to the board.

This is the result you should get in the Serial monitor:

State of GPIO 5: 1

We can see that the returned value is 1 (digital state HIGH), which is what we expected, because we connected the pin to the positive power supply. As a test, you can also connect the pin to the ground, and the state should go to 0.

4.3   Grab the Content From a Web Page

As a last project in this chapter, we are finally going to use the WiFi connection of the chip, to grab the content of a page. We will simply use the www.example.com page, as it's a basic page largely used for test purposes.

This is the complete code for this project:

```cpp
// Import required libraries
#include <ESP8266WiFi.h>

// WiFi parameters
const char* ssid = "your_wifi_network";
const char* password = "your_wifi_password";

// Host
const char* host = "www.example.com";

void setup() {
  // Start Serial
  Serial.begin(115200);
  delay(10);

  // We start by connecting to a WiFi network
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

int value = 0;

void loop() {

  Serial.print("Connecting to ");
  Serial.println(host);

  // Use WiFiClient class to create TCP connections
  WiFiClient client;
  const int httpPort = 80;
  if (!client.connect(host, httpPort)) {
    Serial.println("connection failed");
    return;
  }

  // This will send the request to the server
  client.print(String("GET /") + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n");
  delay(10);

  // Read all the lines of the reply from server and print them to Serial
  while(client.available()){
    String line = client.readStringUntil('\r');
    Serial.print(line);
  }

  Serial.println();
```
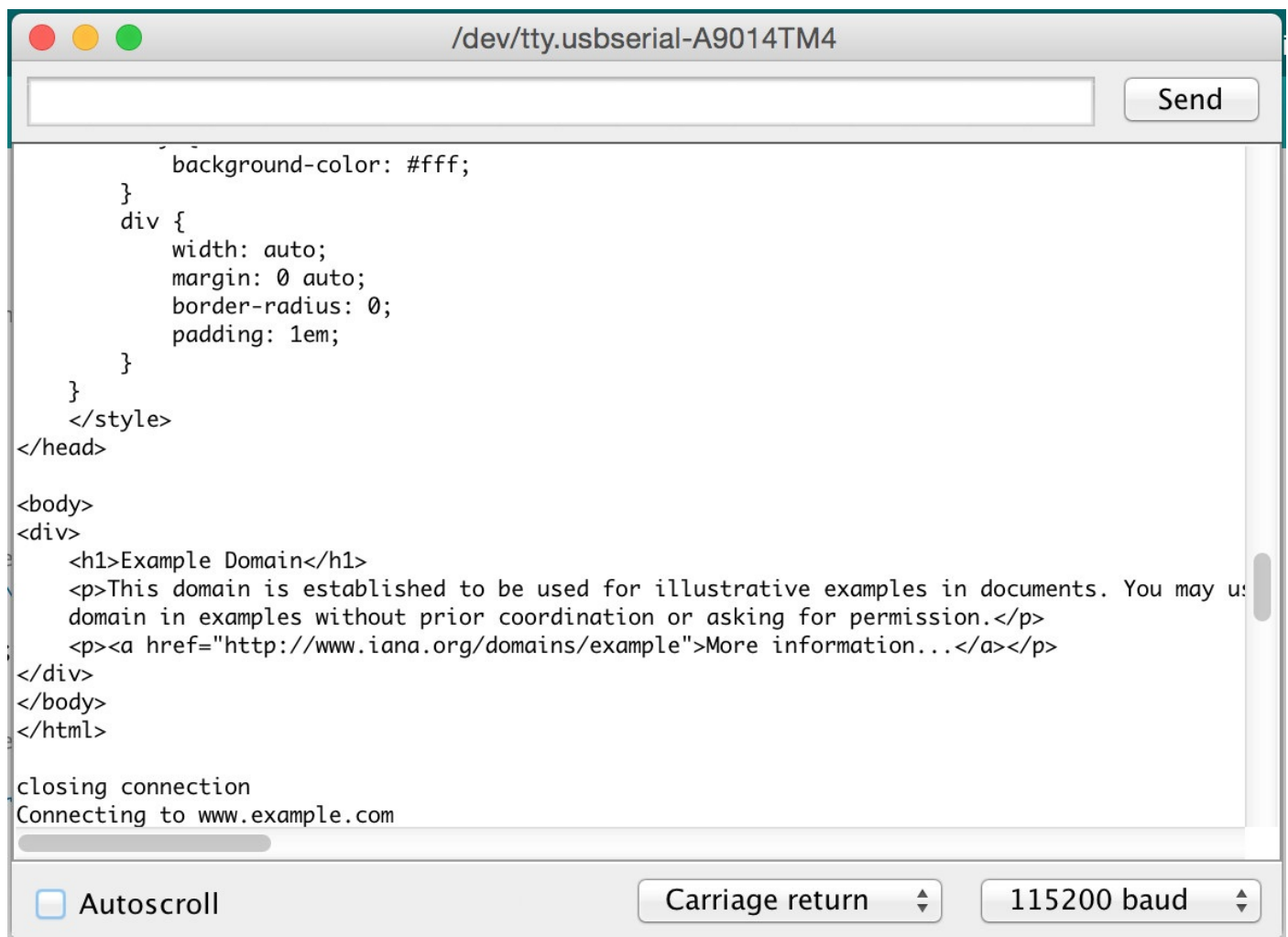
```
    Serial.println("closing connection");
    delay(5000);

}
```

The code is really basic: we first open a connection to the example.com website, and then send a GET request to grab the content of the page. Using the `while(client.available())` code, we also listen for incoming data, and print it all inside the Serial monitor.

You can now copy this code and paste it into the Arduino IDE. Then, upload it to the board. This is what you should see in the Serial monitor:

This is basically the content of the page, in pure HTML code.

Congratulations, you just completed your very first projects using the ESP8266 chip!

Chapter 5

WiFi Weather Measurement Station

In this first chapter of the book, we are going to make a typical home automation project: a WiFi weather measurement station. We will connect a DHT11 sensor to the ESP8266 board, and access the data via WiFi.

To do so, we will run a simple web server on the ESP8266 chip, that will display the results inside a web page. We will also make this web page responsive, so it looks nice even if you are using a smartphone. Finally, we will use the fact that the ESP8266 is already connected to the web to grab weather measurements online & display it in the interface as well.

5.1 Hardware & Software Requirements

For this project, you will of course need an ESP8266 chip. You can for example use an Olimex ESP8266 module.

You will also need a temperature sensor. I used a DHT11 sensor, which is cheap, very easy to use & that will allow us to measure the ambient temperature & humidity.

You will also need a 3.3V FTDI USB module to program the ESP8266 chip. Finally, you will also need some jumper wires & a breadboard.

This is a list of all the components that will be used in this chapter:

- ESP8266 Olimex module (https://www.olimex.com/Products/IoT/MOD-WIFI-ESP8266-DEV/)
- Breadboard 3.3V power supply (https://www.sparkfun.com/products/13032)
- 3.3V FTDI USB module (https://www.sparkfun.com/products/9873)
- DHT11 sensor (https://www.adafruit.com/products/386)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/9194)

On the software side, please refer to the Getting Started with the ESP8266 book (or to the first chapters of this book) to know what software components you need to get.

You will also need the DHT library that you can get from:

https://github.com/adafruit/DHT-sensor-library

To install an Arduino library, simply put the library's folder inside your Arduino 'libraries' folder.

5.2   Hardware Configuration

Again, for most of the hardware configuration please refer to the Getting Started with the ESP8266 book (or to the first chapters of this book). At the end, you will need to have an ESP8266 chip ready to be used with the ESP8266 Arduino IDE.

Once this is done, simply insert the DHT11 sensor on the breadboard. Then, connect the left pin to VCC (red power rail), the right pin to GND (blue power rail), and the pin next to VCC to the GPIO5 pin on your ESP8266 chip. This is the final result, not showing the USB-to-Serial FTDI cables:
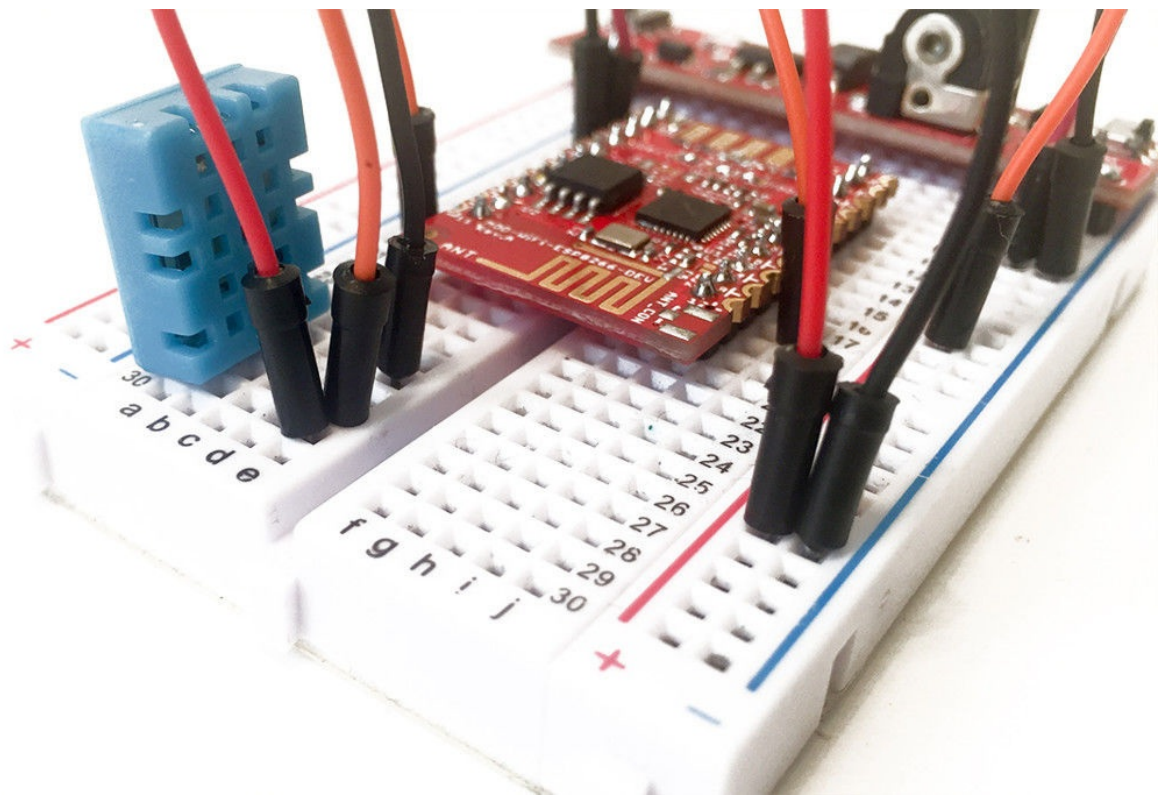
5.3   Testing the Sensor

We are now going to test the sensor. Again, remember that we are using the modified version of the Arduino IDE, so we can code just like we would do using an Arduino board. Here, we will simply print the value of the temperature inside the Serial monitor of the Arduino IDE.

This is the complete code for this part:

```cpp
// Libraries
#include "DHT.h"

// Pin
#define DHTPIN 5

// Use DHT11 sensor
#define DHTTYPE DHT11

// Initialize DHT sensor
DHT dht(DHTPIN, DHTTYPE, 15);

void setup() {

  // Start Serial
  Serial.begin(115200);

  // Init DHT
  dht.begin();
}

void loop() {

  // Reading temperature and humidity
  float h = dht.readHumidity();
  // Read temperature as Celsius
  float t = dht.readTemperature();

  // Display data
  Serial.print("Humidity: ");
  Serial.print(h);
  Serial.print(" %\t");
  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.println(" *C ");

  // Wait a few seconds between measurements.
  delay(2000);

}
```

Let's see the details of the code. You can see that all the measurement part is contained inside the **loop()** unction, which makes the code inside it repeat every 2 seconds.

Then, we read data from the DHT11 sensor, print the value of the temperature & humidity on the Serial port.

Note that the complete code can also be found inside the GitHub repository of the book:

https://github.com/openhomeautomation/home-automation-esp8266

You can now paste this code in the Arduino IDE, and upload it to your board. Then, open the Serial monitor. You should immediately see the temperature & humidity readings inside the Serial monitor. My sensor was reading around 24 degrees Celsius when I tested it, which is a realistic value.

5.4   Accessing the Sensor via WiFi

At this point, we are sure that the sensor is working and that data can be read by the ESP8266 chip. Now, we are going to build the sketch that will connect to your WiFi network, and then serve a web page that will display the results in live.

As this sketch is quite long, I will only detail the most important parts here. You can of course find the complete code for this project inside the GitHub repository of the book.

First, you need to set up your own WiFi network name & password in the code:

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

After that, we create a web server on port 80:

```
WiFiServer server(80);
```

Then, inside the `setup()` function of the sketch, we connect the ESP8266 to the WiFi network:

```
WiFi.begin(ssid, password);

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

Then, we start the server, and print the IP address on the Serial port:

```
// Start the server
server.begin();
Serial.println("Server started");

// Print the IP address
Serial.println(WiFi.localIP());
```

Inside the `loop()` function of the sketch, we check if a client is connected to the ESP8266:

```
WiFiClient client = server.available();
```

Then, we read data from the sensor:

```
// Reading temperature and humidity
float h = dht.readHumidity();

// Read temperature as Celsius
float t = dht.readTemperature();
```

After that, we read the incoming request from the client:

```
String req = client.readStringUntil('\r');
Serial.println(req);
client.flush()
```

Then, we prepare our answer. What we want here is to serve the data to the client in an elegant way. That's why we will use the Bootstrap CSS framework, that will make our page looks pretty. It will also makes it responsive, so it will look great on mobile devices as well.

The first part is to send the Head tag of the HTML page, which includes the Bootstrap CSS file. We also set in this part the refresh rate of the page, which will be automatically refreshed every minute:

```
String s = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n";
s += "<head>";
s += "<meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">";
s += "<meta http-equiv=\"refresh\" content=\"60\" />";
s += "<script src=\"https://code.jquery.com/jquery-2.1.3.min.js\"></script>";
s += "<link rel=\"stylesheet\" href=";
s += "\"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/css/bootstrap.min.css\">";
s += "<style>body{font-size: 24px;} .voffset {margin-top: 30px;}</style>";
s += "</head>";
```

Then, we send the core of the page, which consists in simply displaying the temperature & humidity data:

```
s += "<div class=\"container\">";
s += "<h1>WiFi Weather Station</h1>";

s += "<div class=\"row voffset\">";
s += "<div class=\"col-md-3\">Temperature: </div>";
s += "<div class=\"col-md-3\">" + String(t) + "</div>";
s += "<div class=\"col-md-3\">Humidity: </div>"
s += "<div class=\"col-md-3\">" + String(h) + "</div>";
s += "</div>";

s += "</div>";
```

Finally, we send this to the client, and wait until the client disconnects from our board:

```
client.print(s);
delay(1);
Serial.println("Client disconnected");
```

Note that you can find all the code for this project inside the GitHub repository of the book:

https://github.com/openhomeautomation/home-automation-esp8266

It's now time to test the project. Get the code from the GitHub repository, modify it with your own parameters, and then upload the code to the board.

After that, open the Serial monitor of the Arduino IDE. You should see that the IP address is displayed inside the Serial monitor.

Then, simply go to a web browser and type this IP address. You should immediately see the measured data in your browser:

WiFi Weather Station

Temperature: 23.0   Humidity: 34.0

Note that you can also do the same from your mobile phone or tablet, and it will work just as well!

5.5   Integrating the OpenWeatherMap API

We are going to push this project a little further now. Some weather stations that you can get from a store incorporate an external sensor that you can for example put outside, and then it will display both inside & outside temperatures on the station.

Here, we will use the fact that our project is already connected to the web to get the local outside temperature & humidity from the web. To do so, we will use the OpenWeatherMap API:

http://openweathermap.org/api

It's very easy to test. For example, you can just visit the following link to get the current weather data for London, UK:

http://api.openweathermap.org/data/2.5/weather?q=London,uk

You will get an answer in a JSON container:

```
{
"coord": {
  "lon": -0.13,
  "lat": 51.51
},
"sys": {
    "message": 0.0323,
    "country": "GB",
    "sunrise": 1428384087,
    "sunset": 1428432222
},
"weather": [
{
  "id": 800,
  "main": "Clear",
  "description": "Sky is Clear",
  "icon": "01d"
}
],
"base": "stations",
"main": {
  "temp": 272.73,
  "temp_min": 272.73,
  "temp_max": 272.73,
  "pressure": 1042.28,
  "sea_level": 1050.54,
  "grnd_level": 1042.28,
  "humidity": 87
},
"wind": {
  "speed": 0.96,
  "deg": 43.0004
},
"clouds": {
  "all": 0
},
"dt": 1428388195,
"id": 2643743,
"name": "London",
"cod": 200
}
```

That's good, because it will be really easy to use by our simple HTTP server running on the ESP8266 chip. We will only need to modify the sketch a bit. This is what we need to add for the interface:

```
s += "<div class=\"row voffset\">";
s += "<div class=\"col-md-3\">Exterior temperature: </div>";
s += "<div id=\"ext_temp\" class=\"col-md-3\"></div>";
s += "<div class=\"col-md-3\">Exterior humidity: </div>";
s += "<div id=\"ext_humidity\" class=\"col-md-3\"></div>";
s += "</div>";
```

And after that, we need to add some JavaScript into the answer to the client, so the ESP8266 constantly fetches data from the OpenWeatherMap website:

```
s += "<script>$.ajax({url: ";
s += "\"http://api.openweathermap.org/data/2.5/weather?q=London,uk\",";
s += " crossDomain: true})";
s += ".done(function(result) {";
s += "$(\"#ext_temp\").html((result.main.temp -  273.15).toFixed(2)); ";
s += "$(\"#ext_humidity\").html(result.main.humidity);})";
s += "</script>";
```

Of course, you will need to modify the code accordingly for your own city & country.

Note that you can find all the code for this project inside the GitHub repository of the book:

https://github.com/openhomeautomation/home-automation-esp8266

It's now time to test the project. Get the code from the GitHub repository, modify it with your own parameters, and then upload the code to the board.

Now, go to the same IP address as before with your web browser, and this is what you will see:

WiFi Weather Station

| Temperature: | 23.0 | Humidity: | 34.0 |
| Exterior temperature: | 2.88 | Exterior humidity: | 84 |

5.6   How to Go Further

Let's summarize what we achieved in this project. We built a WiFi weather station based on the ESP8266 WiFi chip. We used a sensor to measure the local temperature & humidity, and also fetched data from the web to get the outside temperature. Then, we displayed everything on a web page that can also be accessed on mobile devices.

There are many things you can do to improve this project. You can for example use more data that you receive from the OpenWeatherMap API & display this data as well on the web page, like the local barometric pressure, or the weather predictions. You can also connect more sensors to the ESP8266 chip, like an ambient light sensor.

Chapter 6

Control a Lamp Remotely

In this chapter, we are going to use the ESP8266 chip to control a lamp remotely via WiFi. However, we won't do what you usually see on the web. Indeed, people usually use a server running on their computers to control devices remotely.

Here, we are going to use the onboard processor of the ESP8266 to host a small web server, that will generate a simple interface on a web page from which you will be able to control the lamp. And we will even make this interface responsive, so it can also be used with your phone or tablet!

6.1 Hardware & Software Requirements

For this project, you will of course need an ESP8266 chip. As for the whole book, I used the Olimex ESP8266 module, but any ESP8266 module will work fine here.

You will also need some way to control your lamp or other devices. I originally used a simple relay for my tests, but I quickly moved to a PowerSwitch Tail Kit which allows to simply & safely plug high voltage devices to your projects.

You will also need a 3.3V FTDI USB module to program the ESP8266 chip.

Finally, you will also need some jumper wires & a breadboard.

This is a list of all the components that will be used in this project:

- ESP8266 Olimex module (https://www.olimex.com/Products/IoT/MOD-WIFI-ESP8266-DEV/)
- Breadboard 3.3V power supply (https://www.sparkfun.com/products/13032)
- 3.3V FTDI USB module (https://www.sparkfun.com/products/9873)
- PowerSwitch Tail Kit (https://www.sparkfun.com/products/10747)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/9194)

Note that you will also need a device to control. I used a simple 30W desk lamp as a test device, but you can also use any other device in your home (if the power rating is lower than the maximum power accepted by the PowerSwitch Tail Kit). You can also just use a simple relay for test purposes.

On the software side, please refer to the Getting Started with the ESP8266 book (or to the first chapters of this book) to know which software components you need to get to make the ESP8266 chip work.

At the end, you will need to have an ESP8266 chip ready to work with the ESP8266 Arduino IDE.

6.2   Hardware Configuration

Again, please refer to the Getting Started with the ESP8266 book (or to the first chapters of this book)

The only thing you need to add to this basic configuration is the PowerSwitch Tail Kit. Connect the two pins on the right (-in and Ground) on the ground of our project (blue power rail), and the +in pin to the GPIO5 pin. If your board doesn't have this pin, you can plug it to the free GPIO pin of your choice, you will just need to modify your code accordingly.

Then, also connect a lamp or any electrical device to the PowerSwitch, and the other end of the PowerSwitch to the mains electricity.

This is the assembled project, without the FTDI module to configure the module:

And this is the project deployed close to the lamp I used as a test:

6.3   Controlling the Lamp Remotely

We are now going to write the code required to control our lamp remotely. Note that we want a completely autonomous operation of the device. The ESP8266 will have to handle requests coming from your browser, display a simple HTML page with two buttons (On & Off), and then control the relay accordingly. As we are using the Arduino IDE to program our ESP8266 we will also be using the well-known Arduino language for this part.

As the code is quite long, I will only cover the important parts here. Of course, you can find all the code inside the GitHub repository of the book.

It starts by include the correct library:

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

Then, you need to enter your WiFi network & password:

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

We also declare a web server running on port 80:

```
WiFiServer server(80);
```

We also declare that we will use GPIO pin 5. Of course if you are using another GPIO pin you will need to modify this value:

```
int output_pin = 5;
```

In the `setup()` function, we declare the pin on which the relay/PowerSwitch is connected:

```
pinMode(output_pin, OUTPUT);
```

After that, we connect to the WiFi network and start the server:

```
WiFi.begin(ssid, password);

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

// Start the server
server.begin();
Serial.println("Server started");
```

Now, in the `loop()` function, we listen to incoming connections on port 80:

```
WiFiClient client = server.available();
```

When we receive some data, we first check if it doesn't contain the **on** or **off** values. If it does, we change the state of the output pin accordingly:

```
if (req.indexOf("/on") != -1){
   digitalWrite(output_pin, 1);
}
else if (req.indexOf("/off") != -1) {
   digitalWrite(output_pin, 0);
}
```

Now, we need to serve and display a basic interface every time we access the board via our browser. We will actually send all the HTML page, line by line. Let's start with the **<head>** element:

```
String s = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n";
s += "<head>";
s += "<meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">";
s += "<script src=\"https://code.jquery.com/jquery-2.1.3.min.js\"></script>";
s += "<link rel=\"stylesheet\" href=\"https://maxcdn.bootstrapcdn.com/";
s += "bootstrap/3.3.4/css/bootstrap.min.css\">";
s += "</head>";
```

You can see that we even import modules like jQuery (to handle the clicks on our buttons) and Bootstrap (to get a nice responsive interface). This is possible on the small ESP8266 chip because we are using the hosted versions of these frameworks, and therefore it doesn't take extra memory space on the chip.

Then, we send the interface itself with two buttons:

```
s += "<div class=\"container\">";
s += "<h1>Relay Control</h1>";
s += "<div class=\"row\">";
s += "<div class=\"col-md-2\"><input class=\"btn btn-block btn-lg btn-primary\" ";
s += "type=\"button\" value=\"On\" onclick=\"on()\"></div>";
s += "<div class=\"col-md-2\"><input class=\"btn btn-block btn-lg btn-danger\" ";
s += "type=\"button\" value=\"Off\" onclick=\"off()\"></div>";
s += "</div></div>";
```

This is also where you can give a name to our project. I put simply 'Relay Control' as I was doing all the tests with a simple relay.

Finally, we send some JavaScript code as well to handle the clicks on the buttons:

```
s += "<script>function on() {$.get(\"/on\");}</script>";
s += "<script>function off() {$.get(\"/off\");}</script>";
```

Note that you can find all the code for this project on the corresponding GitHub repository:

https://github.com/openhomeautomation/home-automation-esp8266

I used again the ESP8266 Arduino IDE software to upload files to the ESP8266 board. Then, open the Serial monitor to get the IP address of your board.

Finally, simply open your favorite web browser, and type in the board IP address. You should see the interface being displayed:
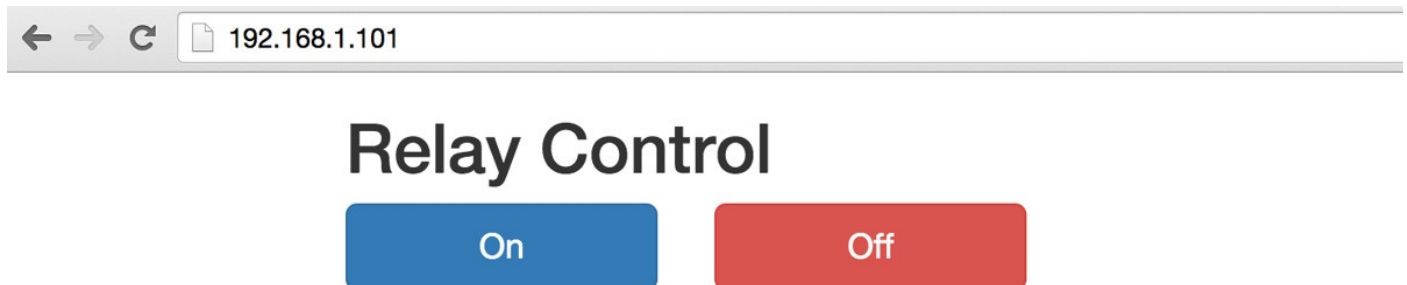
You can now try the buttons: you should notice that the state of the PowerSwitch or the relay is changing instantly, and that the lamp is turning on & off. And note that this is only using the ESP8266 chip here: it's completely independent from your computer!

The interface is also responsive, thanks to the Bootstrap framework. This means it will automatically adapt to the device on which it is used. This is for example the result on my phone:
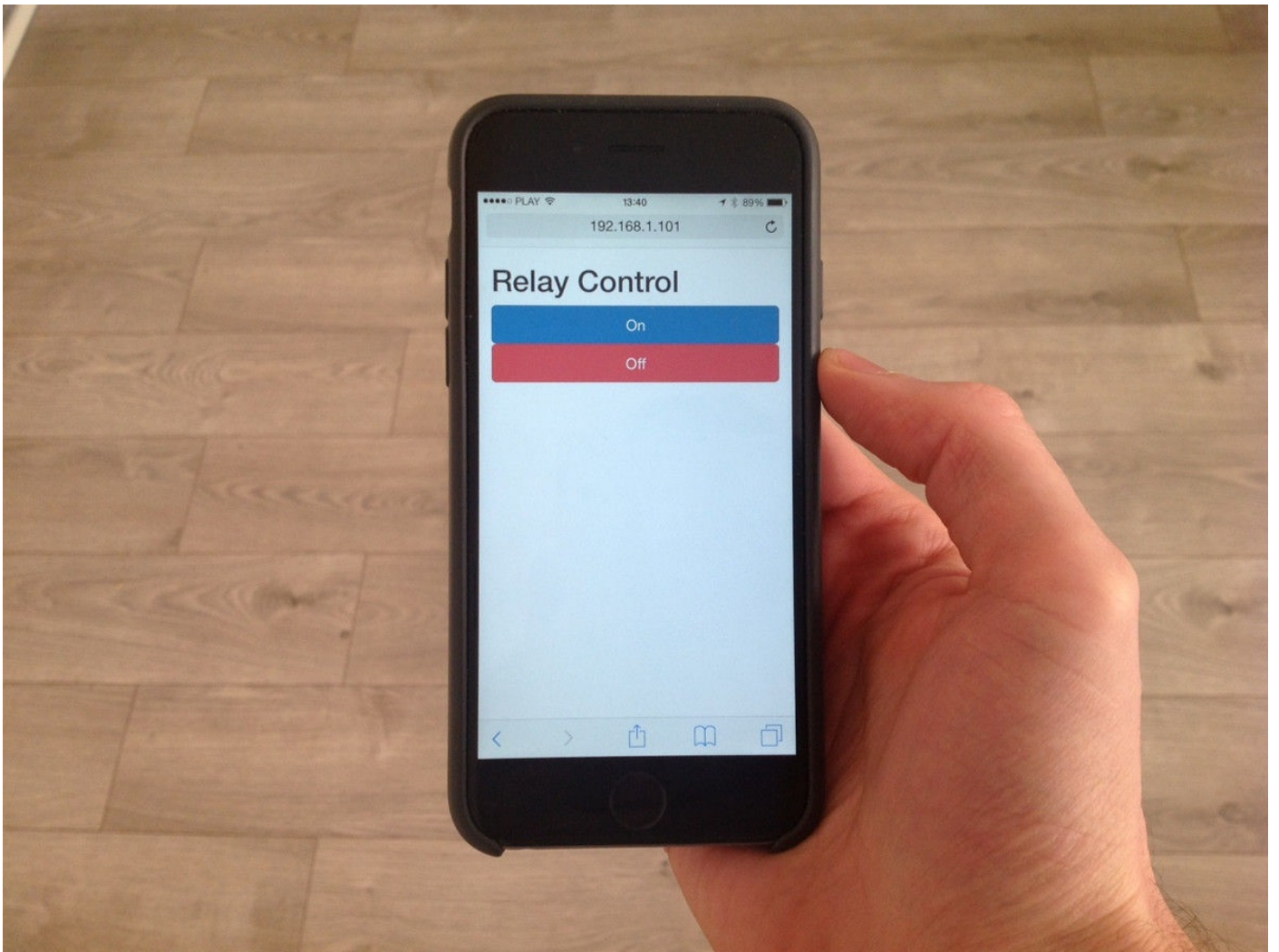
6.4   How to Go Further

In this project, we built a completely autonomous remote lamp controller using the ESP8266 WiFi chip. We made this little chip control a lamp (or any other device) by serving a nice & responsive interface, allowing you to control the device from any terminal within your local WiFi network, like a phone or tablet.

Of course, you can use what you learned in this project to build other home automation systems. You can for example use the same principles to add sensors to the project that measure data and display it all in a single interface, also served by the ESP8266 module.

Chapter 7

Create a Simple WiFi Alarm System

In this chapter, we are going to build a simple alarm system based on the ESP8266. It will be composed of one or several ESP8266 modules coupled with motion sensors, and a central alarm interface running on your computer.

Typically, motion detectors are using low-cost radios to communicate with a central alarm systems, and never WiFi. However, the ESP8266 chip is so cheap that it also makes sense here to use WiFi for motion sensors. Let's dive into the project!

7.1   Hardware & Software Requirements

For this project, you will of course need an ESP8266 chip. As for the whole book, I used the Olimex ESP8266 module, but any ESP8266 module will work fine here.

You will also need a motion sensor. I used a simple & cheap PIR motion sensor for this project. These are exactly the same kind of sensor that are used in commercial home automation systems:
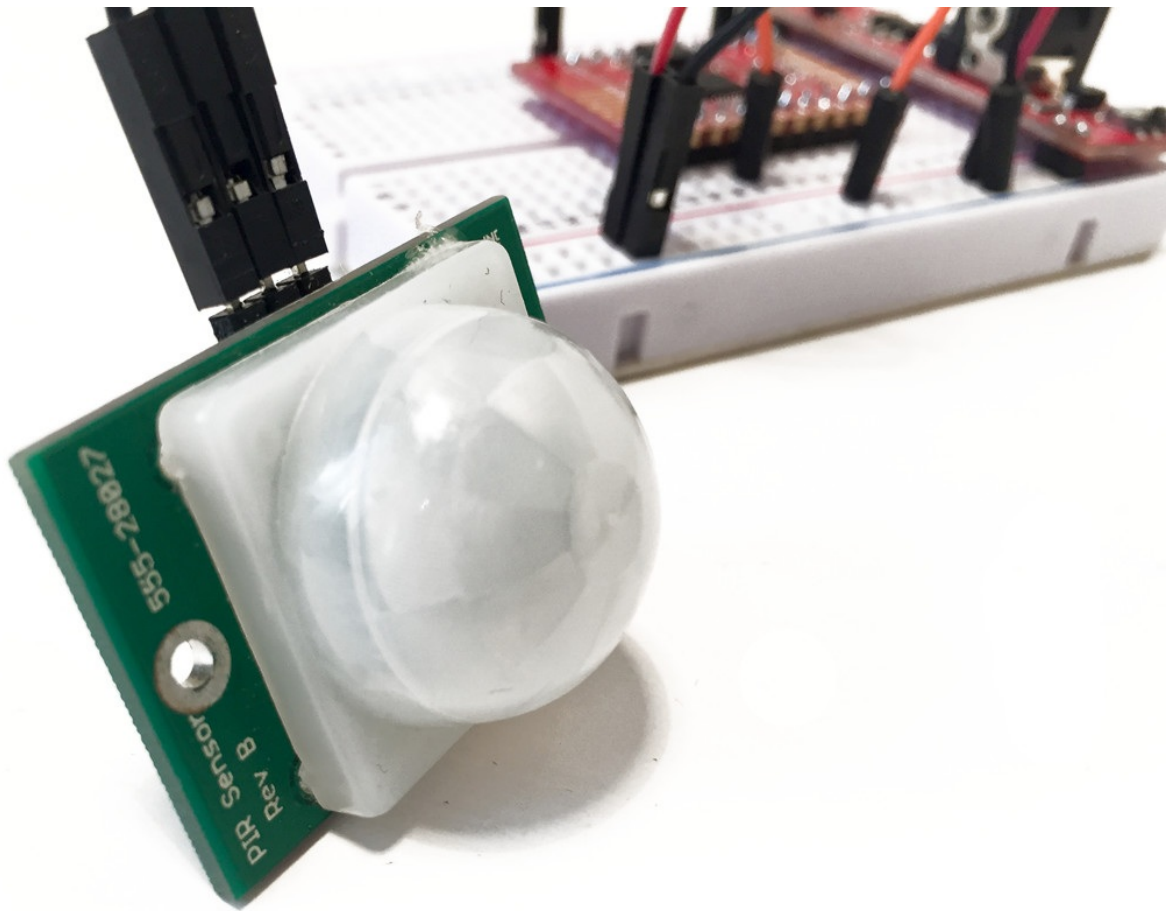
This is a list of all the components that will be used in this project:

- ESP8266 Olimex module (https://www.olimex.com/Products/IoT/MOD-WIFI-ESP8266-DEV/)
- Breadboard 3.3V power supply (https://www.sparkfun.com/products/13032)
- 3.3V FTDI USB module (https://www.sparkfun.com/products/9873)
- PIR motion sensor (https://www.adafruit.com/product/189)
- Breadboard (https://www.sparkfun.com/products/12002)

- Jumper wires (https://www.sparkfun.com/products/9194)

Also note that if you want to have several motion sensors like the one we are going to build here, you will need several ESP8266 chips & PIR sensors as well. As an example, we will only use one motion sensor for the project.

On the software side, please refer to the Getting Started with the ESP8266 book (or to the first chapters of this book) to know which software components you need to get to make the ESP8266 chip work.

At the end, you will need to have an ESP8266 chip ready to work with the ESP8266 Arduino IDE.

7.2   Hardware Configuration

Again, please refer to the Getting Started with the ESP8266 book (or to the first chapters of this book). The only thing you will need to add in this project is the motion sensor.

The PIR motion sensor is really easy to connect to the ESP8266 chip. First, connect the power supply: the VCC pin goes to the red power rail on the breadboard, and the GND pin of the sensor goes to the blue power rail. Finally, connect the SIG pin of the sensor to the GPIO pin 5 of the ESP8266 WiFi chip. This is the final result:
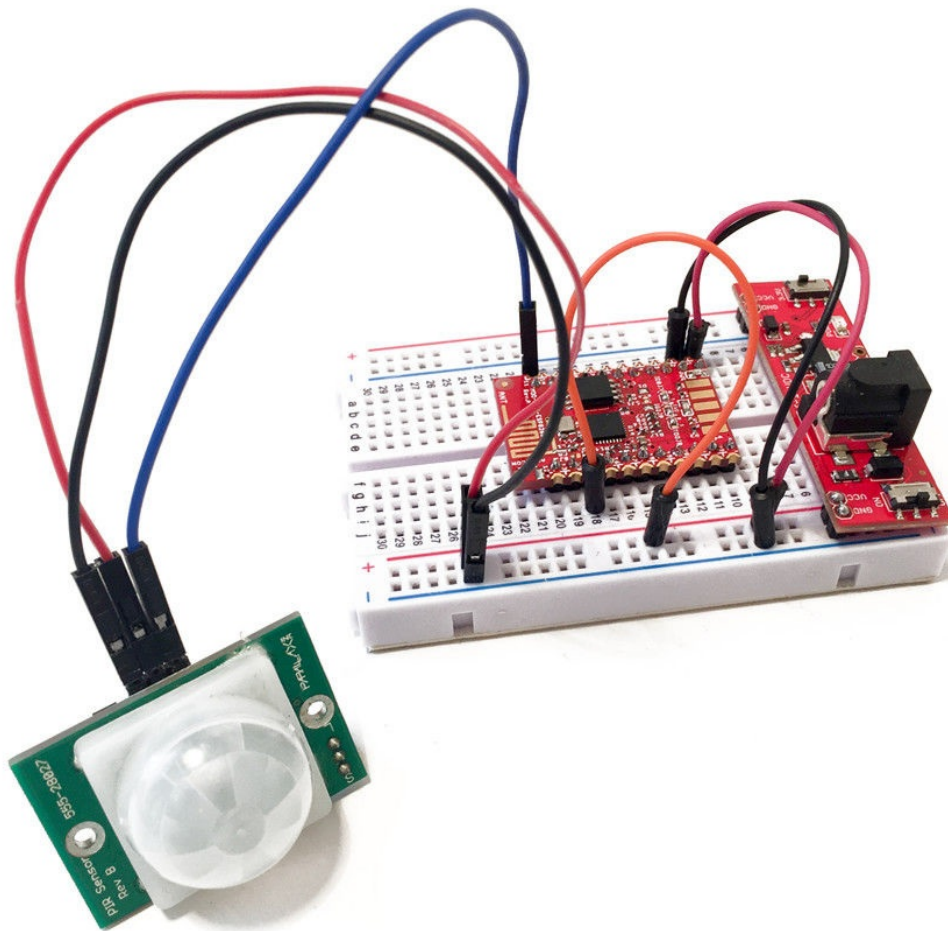
Of course, you can also use another pin for the SIG pin, especially in case you are using an ESP8266 WiFi that doesn't have all the pins exposed. In that case, you will need to modify the code slightly.

7.3   Writing the Motion Sensor Code

We are first going to write the sketch for the motion sensor. This code will basically constantly check the status of the motion sensor. If it detects that the state changed (which means motion has been detected), it will send a message to a web server running on your computer (that we will code later).

As the code is quite long, I will only cover the important parts here. Of course, you can find all the code inside the GitHub repository of the book.

We start by including the required library for the project:

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

Then, we define the IP address of the server that will run the alarm system. In this case, it is the IP address of your computer:

```
const char* host = "192.168.1.100";
```

There are many ways you can get it depending on your operating system, but it is usually found under a menu or panel called "Network Preferences".

We also initialize the motion sensor state to 0:

```
int motion_sensor_state = 0;
```

Then, in the `loop()` function of the sketch, we read data from the pin on which the motion sensor is connected:

```
int new_motion_sensor_state = digitalRead(5);
```

Then, we compare this reading to the old reading, to see if the state changed:

```
if (new_motion_sensor_state != motion_sensor_state)
```

If this is the case, we first assign the new state to the old state:

```
motion_sensor_state = new_motion_sensor_state;
```

Then, we create a client instant to connect to the server running on our computer:

```
WiFiClient client;
const int httpPort = 3000;
if (!client.connect(host, httpPort)) {
  Serial.println("connection failed");
  return;
}
```

After that, we send a POST request to the server that contains the state of the sensor:

```
client.print(String("POST /motion?state=") + String(new_motion_sensor_state) +
" HTTP/1.1\r\n" + "Host: " + host + "\r\n" + "Connection: close\r\n\r\n");
delay(10);
```

After that, we read the answer from the server:

```
while(client.available()){
  String line = client.readStringUntil('\r');
  Serial.print(line);
}
```

Note that you can find the whole sketch inside the GitHub repository of the book:

https://github.com/openhomeautomation/home-automation-esp8266

It's now time to test the project. Get the code from the GitHub repository, modify it with your own parameters, and then upload the code to the board.

Now, we can move to coding the alarm system itself.

7.4   Creating Our Alarm System

We are now going to write the code for the alarm system, that will run on your computer. The server will be really basic: you will be able to monitor the state of the motion sensor from an interface, and also activate or deactivate the alarm mode.

When the alarm mode will be activated, if motion is detected there will be a loud sound coming out of your computer. If it is deactivated, the sensor will keep on communicating with the server, but no sound will be emitted by the computer.

We will use the very popular Node.js framework to code our server. You can now download it and install it from:

https://nodejs.org/

There will basically be 3 files in this project:

- The **app.js** file, that will contain the code for the server itself
- The **interface.jade** file, that will define the interface
- The **interface.js** file, that will make the link between the interface & the server

As the code is quite long, I will only cover the important parts here. Of course, you can find all the code inside the GitHub repository of the book.

Let's start with the main file, which is called **app.js**. It starts by including the Express framework (http://expressjs.com/), which will make it easier to code our web server:

```
var express = require('express');
var app = express();
```

Then, we define some global variables: one for the state of the sensor, and for the state of the alarm:

```
// Global variable for motion sensor
var motion = 0;

// Alarm state
var alarm = 0;
```

Then, we have to define various routes to get & to set the state of these variables:

```
// Change motion sensor state
app.post('/motion', function(req, res) {
  motion = req.query.state;
  res.send('Data received: ' + motion);
});

// Get motion sensor state
```

```javascript
app.get('/motion', function(req, res) {
  res.json({state: motion});
});

// Get alarm state
app.get('/alarm', function(req, res) {
  res.json({state: alarm});
});

// Set alarm state
app.post('/alarm', function(req, res) {
  alarm = req.query.state;
  res.send('Data received: ' + alarm);
});
```

Finally, we start the web server with:

```javascript
app.listen(port);
console.log("Listening on port " + port);
```

Now, let's see the **interface.js** file. This file will make the link between the interface and the server.

We first define a function to query the value of the motion sensor, and change the interface accordingly. If motion is detected and the alarm is not on, we simply change the state of the indicator in the interface. If the alarm is on, we also play a sound.

This is the complete code for this function:

```javascript
function refresh_motion() {
  $.getq('queue', '/motion', function(data) {
    if (data.state == 0) {$('#motion').html("No motion detected");}
    if (data.state == 1) {

      // Change text
      $('#motion').html("Motion detected");

      // Play sound if alarm is on
      $.get('/alarm', function(data) {
        if (data.state == 1) {$.playSound('/audio/alarm');}
      });

    }
  });
}
```

Now, we run this function every 500 milliseconds:

```javascript
refresh_motion();
setInterval(refresh_motion, 500);
```

We also need to initialize a button for the alarm, and change the text & color of this button depending of the state of the alarm:

```javascript
$.get('/alarm', function(data) {
  var alarm_state = data.state;

  if (alarm_state == 1) {
    $('#alarm').html("Alarm On");
    $('#alarm').attr('class', 'btn btn-block btn-lg btn-success');
  }
```

```
  if (alarm_state == 0) {
    $('#alarm').html("Alarm Off");
    $('#alarm').attr('class', 'btn btn-block btn-lg btn-danger');
  }
});
```

Finally, we handle the clicks on the alarm button, and change the state of the alarm accordingly:

```
$('#alarm').click(function() {

  // Get alarm state
  $.get('/alarm', function(data) {
    var alarm_state = data.state;

    if (alarm_state == 0) {
      $.post('/alarm?state=1');
        $('#alarm').html("Alarm On");
        $('#alarm').attr('class', 'btn btn-block btn-lg btn-success');
      }

    if (alarm_state == 1) {
      $.post('/alarm?state=0');
      $('#alarm').html("Alarm Off");
      $('#alarm').attr('class', 'btn btn-block btn-lg btn-danger');
    }
  });
});
```

Note that you can find all the code for this project inside the corresponding GitHub repository:

https://github.com/openhomeautomation/home-automation-esp8266

It's now time to test the project. Get the code from the GitHub repository, and place all the files into a given folder. Then, go to this folder with a terminal. After that, type:

```
sudo npm install express jade
```

And then:

```
node app.js
```

Then, simply go to your favorite web browser, and type:

```
http://localhost:3000/
```

You should see the interface of your simple alarm system, with the alarm off by default:

Alarm System

Motion sensor: Motion detected

You can also now test the motion sensor. Just wave your hand in front of it, and you should see the text changing inside the interface.

Now, try to click on the button. You should see that the button is turning green:

Alarm System

Motion sensor: No motion detected

Now, try to wave your hand in front of the sensor again. This time, you should hear a loud alarm sound coming from your computer.

Congratulations, you now built a simple alarm system with the ESP8266 chip!

7.5   How to Go Further

Let's summarize what we achieved in this chapter. We connected a PIR motion sensor to the ESP8266 chip, and then built a simple alarm system with it.

There are many things you can do to improve this project. One way is to modify the code of this project slightly to accommodate for more motion sensors. On the ESP8266 side, you will need to for example assign an unique ID to each sensor and transmit this ID along with the state of the motion sensor.

Then, on the server side, you will just need to get the ID of the sensor as well, and display the state of the different sensors inside the interface. You could also imagine connecting this project to Twitter, so you receive a Tweet every time motion is detected in your home.

Chapter 8

Build an Home Automation System Based on the ESP8266

For the final chapter of this book, we are going to integrate all the knowledge we acquired in the book so far to build a small home automation system that integrates several components.

As an example, we will use again the temperature & humidity sensor project and the lamp control project, and integrate these projects into a single interface. This will allow you to monitor your home from a single interface. You will also be able to use what you learned in this project and add more components in the future.

8.1 Hardware & Software Requirements

For this project, you basically need the same hardware that we used in two other projects of this book.

For the temperature & humidity sensor module requirements, please check again Chapter 5. For the lamp control module requirements, please check again Chapter 6.

8.2 Hardware Configuration

Again, to configure the hardware for this project, I will redirect you to the corresponding chapters of the book.

For the temperature & humidity sensor module configuration, please check again Chapter 5. For the lamp control module configuration, please check again Chapter 6.

8.3 Writing the Sketches

We are now going to write two sketches: one for the lamp control project, and one for the temperature & humidity sensor module.

Compared to previous chapters, we don't want to have the ESP8266 WiFi serving it's own interface. Here, we just want these modules to respond to commands sent by a central server, that will run on our computer.

To make things easier, we are going to use the aREST API, which implements a RESTful interface for the ESP8266 board. Basically, it means that your board will be fully controllable via HTTP commands, for example coming from a web server. You can find more details about the project here:

http://arest.io

Now, we are going to start with the sketch to control the lamp. This is the complete sketch for this module:

```cpp
// Import required libraries
#include <ESP8266WiFi.h>
#include <aREST.h>

// Create aREST instance
aREST rest = aREST();

// WiFi parameters
const char* ssid = "your_wifi_name";
const char* password = "your_wifi_password";

// The port to listen for incoming TCP connections
#define LISTEN_PORT           80

// Create an instance of the server
WiFiServer server(LISTEN_PORT);

void setup(void)
{
  // Start Serial
  Serial.begin(115200);

  // Give name and ID to device
  rest.set_id("1");
  rest.set_name("lamp_control");

  // Connect to WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  // Start the server
  server.begin();
  Serial.println("Server started");

  // Print the IP address
  Serial.println(WiFi.localIP());
```

```
  }

void loop() {

  // Handle REST calls
  WiFiClient client = server.available();
  if (!client) {
    return;
  }
  while(!client.available()){
    delay(1);
  }
  rest.handle(client);

}
```

Let's now see the important parts of this sketch. First, we need to include the
ESP8266WiFi library and the aREST library:

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

Then, we create an instance of the aREST library:

```
aREST rest = aREST();
```

In the `loop()` function of the sketch, we check if we have a client connected to the
ESP8266. If that's the case, we handle the request with the aREST library:

```
  // Handle REST calls
WiFiClient client = server.available();
if (!client) {
  return;
}
while(!client.available()){
  delay(1);
}
rest.handle(client);
```

It's now time to test the project. Get the code from the GitHub repository, modify it with
your own parameters, and then upload the code to the board.

Now, open the Serial monitor to get the IP address of the board. We'll assume for the rest
of this section that it is 192.168.1.103. Then, go to your favorite web browser, and type:

http://192.168.1.103/mode/5/o

This command will set the GPIO pin 5 to an output. Then, type:

http://192.168.1.103/digital/5/1

This is the command to turn the GPIO pin 5 to a HIGH state. As soon as you press this
command, the LED should turn on. You should also receive a confirmation message in
your browser. You can of course just put a 0 at the end of the command to turn it off
again.

We are now going to see the sketch for the temperature & humidity sensor module. This is the complete sketch for this part:

```cpp
// Import required libraries
#include <ESP8266WiFi.h>
#include <aREST.h>
#include "DHT.h"

// Pin
#define DHTPIN 5

// Use DHT11 sensor
#define DHTTYPE DHT11

// Initialize DHT sensor
DHT dht(DHTPIN, DHTTYPE, 15);

// Create aREST instance
aREST rest = aREST();

// WiFi parameters
const char* ssid = "your_wifi_name";
const char* password = "your_wifi_password";

// The port to listen for incoming TCP connections
#define LISTEN_PORT           80

// Create an instance of the server
WiFiServer server(LISTEN_PORT);

// Variables
float temperature;
float humidity;

void setup(void)
{
  // Start Serial
  Serial.begin(115200);

  // Give name and ID to device
  rest.set_id("2");
  rest.set_name("sensor");

  // Expose variables to API
  rest.variable("temperature", &temperature);
  rest.variable("humidity", &humidity);

  // Connect to WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  // Start the server
  server.begin();
  Serial.println("Server started");

  // Print the IP address
  Serial.println(WiFi.localIP());

  // Init DHT
  dht.begin();

}

void loop() {

  // Reading temperature and humidity
  humidity = dht.readHumidity();
  // Read temperature as Celsius
```

```
  temperature = dht.readTemperature();

  // Handle REST calls
  WiFiClient client = server.available();
  if (!client) {
    return;
  }
  while(!client.available()){
    delay(1);
  }
  rest.handle(client);

}
```

As the code is quite long, I will only go through the important parts that we added compared to the previous sketch.

Just as before, we need to give an ID and a name to our module:

```
rest.set_id("2");
rest.set_name("sensor");
```

Here, what we want to do is to query for the value of the temperature & humidity of the board. This is done with the **variable()** function:

```
rest.variable("temperature", &temperature);
rest.variable("humidity", &humidity);
```

It's now time to test the project. Get the code from the GitHub repository, modify it with your own parameters, and then upload the code to the board.

Then, open the Serial monitor to get the IP address of this module. Let's assume here it is 192.168.1.102. Now, go to a web browser and type:

http://192.168.1.102/temperature

You will then receive an answer from the board with the value of the temperature:

{"temperature": 22, "id": "2", "name": "sensor", "connected": true}

You can of course do the same with the humidity. Congratulations, you now have two working modules in your home automation system based on the ESP8266!

8.4 Creating the Interface

We are now going to code the interface that you will use to control all the modules from a central interface.

As in the previous chapter, we will use the very popular Node.js framework to code our server. You can now download it and install it from:

https://nodejs.org/

There will basically be 3 files in this project:

- The **app.js** file, that will contain the code for the server itself
- The **interface.jade** file, that will define the interface
- The **interface.js** file, that will make the link between the interface & the server

Let's first see the **app.js** file. We start by declaring the Express framework:

```
var express = require('express');
var app = express();
```

We also set the port to 3000:

```
var port = 3000;
```

After that, we set the view engine to Jade, which we will use to code the interface in a very simple way:

```
app.set('view engine', 'jade');
```

Then, we define the main *route* of the interface, which we will use to access the interface of the home automation system:

```
app.get('/', function(req, res){
  res.render('interface');
});
```

In this project will also use the aREST Node.js module, that will make the link between the server running on your computer and all the modules we created before. This is how to use it in the app:

```
var rest = require("arest")(app);
```

Then, we add both devices into the app:

```
rest.addDevice('http','192.168.1.103');
rest.addDevice('http','192.168.1.104');
```

Of course, you will need to change these IP addresses depending on the IP addresses of your modules.

Finally, we start the app with:

```
app.listen(port);
console.log("Listening on port " + port);
```

Now, let's see the content of the **interface.jade** file. This will define the interface and all the components in it.

First, we need to include the Bootstrap CSS framework that we used in previous chapters of this book. We also include jQuery. This is all defined in this piece of code:

```
head
  title Interface
  link(rel='stylesheet',
  href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.0/css/bootstrap.min.css")
  link(rel='stylesheet', href="/css/interface.css")
  meta(name='viewport', content='width=device-width, initial-scale=1')

  script(src="https://code.jquery.com/jquery-2.1.1.min.js")
  script(src="/js/ajaxq.js")
  script(src="/js/interface.js")
```

Then, we define the interface itself in the body tag. We basically create two buttons to control the module with the LED (called lamp control here), and then we create two indicators for the temperature & humidity module:

```
body
  .container
    h1 Dashboard
      .row.voffset
        .col-md-4
          div Lamp Control
        .col-md-4
          button.btn.btn-block.btn-lg.btn-primary#1 On
        .col-md-4
          button.btn.btn-block.btn-lg.btn-danger#2 Off
      .row.voffset
        .col-md-4
          div#temperature Temperature:
        .col-md-4
          div#humidity Humidity:
```

Let's now see the **interface.js** file, which will make the link between the interface and the server. We first need to set the GPIO pin 5 as an output with the following command:

```
$.getq('queue', '/lamp_control/mode/5/o');
```

Then, we create two functions that will handle the clicks on the buttons in the interface:

```
$("#1").click(function() {
```

```
    $.getq('queue', '/lamp_control/digital/5/1');
  });

  $("#2").click(function() {
    $.getq('queue', '/lamp_control/digital/5/0');
  });
```

Finally, we define a function to refresh the temperature & humidity readings. We also repeat this function every 10 seconds:

```
function refresh_dht() {
  $.getq('queue', '/sensor/temperature', function(data) {
    $('#temperature').html("Temperature: " + data.temperature + " C");
  });

  $.getq('queue', '/sensor/humidity', function(data) {
    $('#humidity').html("Humidity: " + data.humidity + " %");
  });
}
refresh_dht();
setInterval(refresh_dht, 10000);
```

It's now time to test the project. Get the code from the GitHub repository, modify it with your own parameters, and then upload the code to the board.

Then, go to the folder where the **app.js** file is located, and type:

```
sudo npm install arest jade express
```

Then, launch the app with:

```
node app.js
```

After that, go to your favorite browser and type:

http://locahost:3000

You should immediately see the interface of your home automation system:

You can already try pressing the button, it should switch the LED on your ESP8266 project on & off. Of course, you can use the exact same interface to control a lamp that would be connected to it for example.

Note that this interface is also fully responsive, which means you can access it from your phone or tablet. This is the result on my phone:
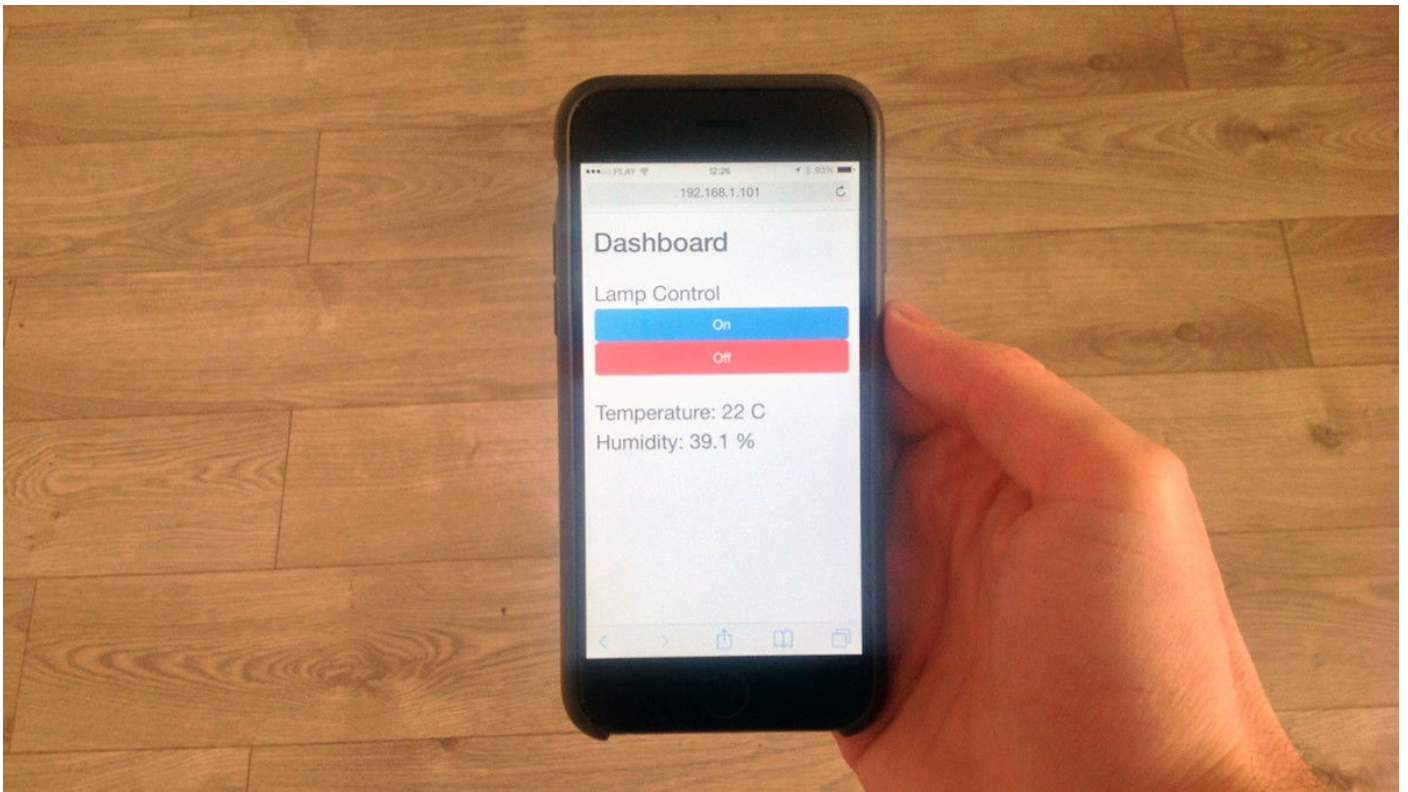
8.5 How to Go Further

Let's summarize what we achieved in this project. We created modules based on the ESP8266 chip, using a REST API that allows us to control these modules via WiFi. Then, we integrated this module into an home automation server running on a computer, so we can control all modules remotely from a single place.

There are many things you can do to improve this project. The first thing to do is to integrate more modules into the project. For example, you could perfectly adapt the alarm system project into this interface.

You can also integrate web services into the same interface, for example to display the outside temperature or barometric pressure, just as we did in Chapter 5.

Chapter 9

Conclusion

If you followed all the projects of this book, congratulations, you now know how to create DIY home automations systems based on the ESP8266 WiFi chip! You learned how to create your own WiFi home automation devices, for example a remote lamp controller that you can use with your mobile phone.

Right now, I recommend that you go again through all the projects of the book, so you can really get all the knowledge that is contained in the book. Then, just experiment! There are virtually no limits on what you can create using the ESP8266 WiFi chip. You can create great home automation systems by using this $5 WiFi chip to automate every aspects oh your home. Have fun!

Chapter 10

Resources

The following is a list of the best ressources concerning open-source home automation with the ESP8266 chip. I organized this chapter in different categories so it is easier for you to find the information you need.

10.1   Learn More About the ESP8266 Chip

- Open Home Automation: The companion website of this book, where you will find other tutorials & ideas about how to use the ESP8266 for home automation projects.
- ESP8266.com: The reference website for the ESP8266 chip, where you will find helpful information about how to use the chip. You will also find a large community there that can help you out with any problems you have with this chip.

10.2   Components

- **SparkFun**: A website selling many sensors & components that can be used with the ESP8266. All their products are open-source and you can download the source files directly from their product descriptions.
- **Adafruit**: A company based in New York that sells high quality products that can be used with the ESP8266.
- **SeeedStudio**: A Chinese company that sells many original products that will also be compatible with the ESP8266. They also offer their own PCB production & assembly services.

Table of Contents

www.ebook777.com