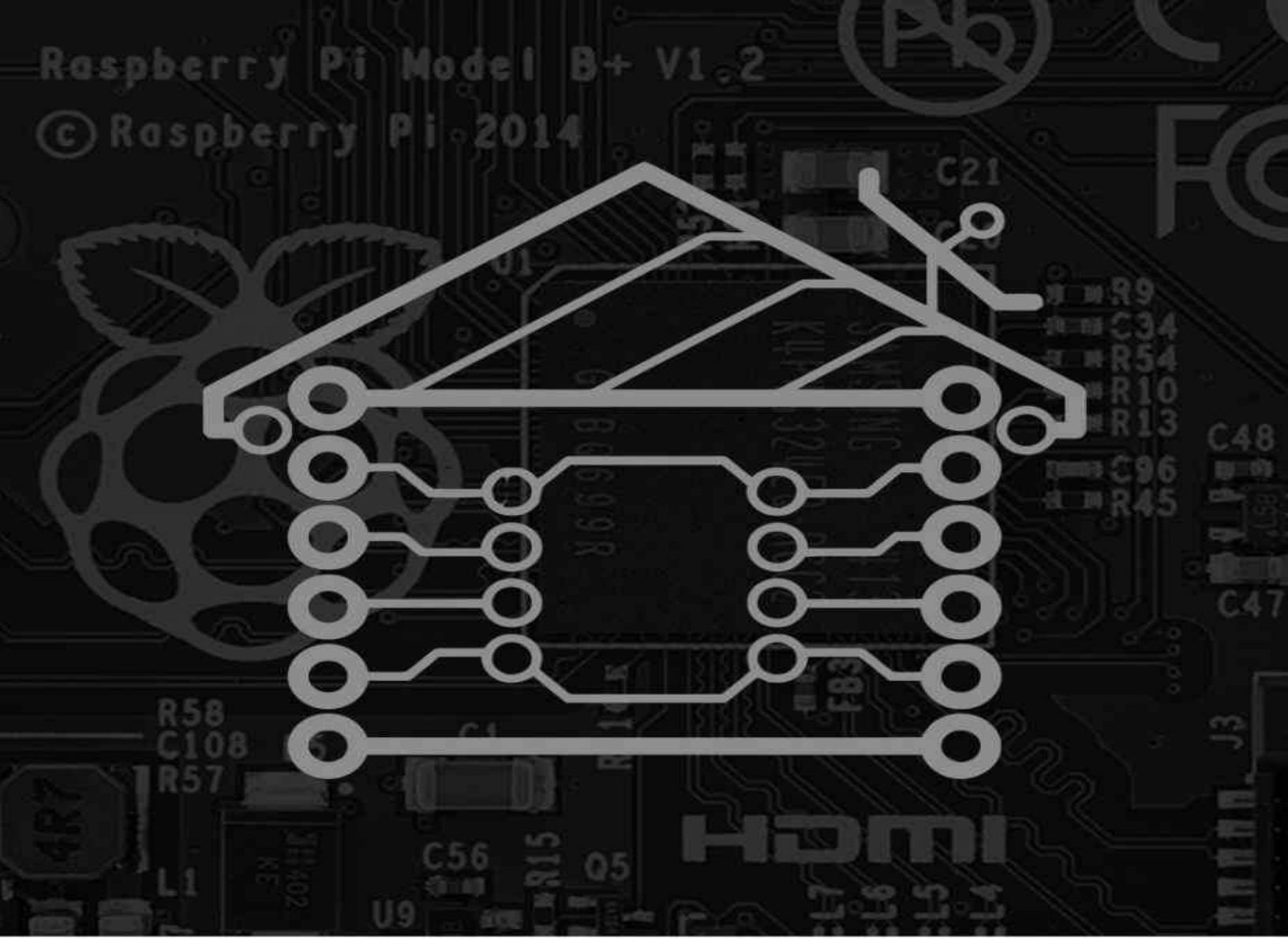

Home Automation Projects with the Raspberry Pi and the ESP8266

Open Home Automation

Marco Schwartz

Home Automation Projects with the Raspberry Pi & the ESP8266

Marco Schwartz

Home Automation Projects with the Raspberry Pi & the ESP8266

Home Automation Projects with the Raspberry Pi & the ESP8266
Legal
About the Author
About the Accompanying Website
1 Introduction
    1.1 Organization Of The Book
    1.2 Who Is This Book For?
    1.3 Prerequisites
2 Introduction to the ESP8266
    2.1 Hardware & Software Requirements
    2.2 Hardware Configuration
    2.3 Configuring Your ESP8266 Chip
    2.4 How to Go Further
3 Build a Digital Doorbell with the ESP8266
    3.1 Hardware & Software Requirements
    3.2 Hardware Configuration
    3.3 Configuring the ESP8266
    3.4 Configuring the Raspberry Pi
    3.5 Testing the Project
    3.6 Playing a Sound with the Doorbell
    3.7 How to Go Further
4 WiFi Temperature & Humidity Sensor
    4.1 Hardware & Software Requirements
    4.2 Hardware Configuration
    4.3 Testing the Sensor
    4.4 Accessing the Sensor via WiFi
    4.5 Connecting the Board to Your Raspberry Pi
    4.6 How to Go Further
5 Control a Remote Lamp from Your Raspberry Pi
    5.1 Hardware & Software Requirements
    5.2 Hardware Configuration
    5.3 Controlling the Lamp Remotely
    5.4 Raspberry Pi Interface
    5.5 How to Go Further
6 Create a Simple WiFi Alarm System
    6.1 Hardware & Software Requirements
    6.2 Hardware Configuration
    6.3 Writing the Motion Sensor Code

6.4 Creating Our Alarm System
6.5 How to Go Further
7 Use the Raspberry Pi as the Hub of ESP8266 Devices
7.1 Hardware & Software Requirements
7.2 Hardware Configuration
7.3 Writing the Sketches
7.4 Creating the Interface
7.5 How to Go Further
8 Conclusion
9 Resources
9.1 General Information
9.2 Components
9.3 Raspberry Pi Resources
9.4 ESP8266 Resources

Home Automation Projects with the Raspberry Pi & the ESP8266

Legal

First eBook edition: January 2016

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 Accompanying Website

This book has an accompanying 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.

Also, if at any moment you are encountering an issue while reading this book, like some code that won't compile or a project that doesn't work, please contact me directly on the following email:

contact@openhomeautomation.net

Also, if you bought this book on the Amazon Kindle platform, it would be amazing if you could leave a review there. It helps me a lot to get feedback about the book, and to create an even better edition of the book in the future. Thanks!

1 Introduction

You probably have heard all about what you can do with the Raspberry Pi. This credit-card sized computer can be plugged into your TV or any HDMI monitor to replace a typical computer. This little device is used in many computer projects, DIY electronics projects and even as a learning tool for kids who want to learn the basics of computer programming.

But among all, the Raspberry Pi is the perfect board for home automation. It is cheap, powerful, and can be interfaced with many sensors and actuators that are usually found in any home automation system. By using the Raspberry Pi, you can build an home automation system that is tailored for your home.

As we will see in this book, it is also very easy to extend the capabilities of the Raspberry Pi using other platforms, like the ESP8266 WiFi chip. This chip is a really cheap & powerful WiFi chip that can be interfaced with sensors and other components, for example relays to control lamps. This is great to make the Raspberry Pi the hub of your home automation system.

Finally, the Raspberry Pi is also the perfect platform to connect your home automation systems to the Internet of Things, as it can easily be interfaced with web services like Twitter.

All of these points make the Raspberry Pi the ideal platform to build home automation systems, and this is exactly what you will learn to do in this book.

1.1 Organization Of The Book

This book will start by an introduction to the ESP8266 WiFi chip, where we will learn how to to use this chip with the Arduino IDE. After that, we will build several home automation projects where we will connect the ESP8266 WiFi chip to the Raspberry Pi board, that will act as the hub of your projects. Finally, at the end of the book we will learn how to connect several ESP8266 devices to your Raspberry Pi, to build a complete home automation system.

1.2 Who Is This Book For?

This book is for all the people who want to use the Raspberry Pi platform to build home automation systems.

This book is also for the people who are currently building home automation systems with other platforms like Arduino, and who want to extend their knowledge, for example by using the Raspberry Pi as the hub of their home automation system.

Finally, this book is also for people who just want to learn more about how to use the Raspberry Pi and get knowledge in electronics & programming.

1.3 Prerequisites

To use this book, you will need to have some basic skills in programming & electronics. It is recommended that you have some basic experience with JavaScript and with client/server interactions.

On the electronics side, a basic experience is required, as you will need to connect sensors to the Raspberry Pi. However, with the detailed explanations that you will find in every chapter of the book, you will be able to follow the different projects without difficulties.

To use all the projects found in this book, you will also need a fully functioning Raspberry Pi, configured with the Raspbian Linux distribution, and connected to the Internet. If you need help doing that, you will find all the required resources on the official Raspberry Pi website:

https://www.raspberrypi.org/

In this book, I will also usually access my Raspberry Pi via SSH, from my own computer. You can learn more about how to do that here:

https://www.raspberrypi.org/documentation/remote-access/ssh/

However, you can perfectly do all the projects of this book directly on your Raspberry Pi, for example if it is connected to an external display.

If you already know a bit about Arduino, it help you as well, as we will be using the Arduino IDE to configure the ESP8266 WiFi chip.

2 Introduction to the ESP8266

The ESP866 is a small WiFi chip that has the huge advantage to come at a ridiculous price point: $5. And the best is that this chip also has a small processor onboard, so it can actually host a complete web server, that can receive commands from other systems.

This makes it the perfect component to complement our Raspberry Pi. We are going to use it to extend the functionalities of the Pi, that will then act as the home automation Hub in our home.

In this chapter, we are going to see how to use the ESP8266 and how to configure your board with a simple sketch.

2.1 Hardware & Software Requirements

There are many boards available on the market for the ESP8266. In the rest of this book, we are going to use the Adafruit ESP8266 breakout board, which makes it very easy to program the chip:

You will also 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. So for to program the chip, I will use a USB FTDI programmer.

For the Adafruit board that we will be using, I recommend a FTDI board that can work with 5V voltage levels. I used a module that can be switched between 3.3V and 5V:

This is a list of all the components that will be used in this chapter, and that you will need for the rest of the book:

- Adafruit ESP8266 Huzzah board (https://www.adafruit.com/product/2471)
- FTDI USB module (https://www.adafruit.com/products/284)

On the software side, you will also need the latest version of the Arduino IDE, that you can get from:

http://www.arduino.cc/en/main/software

2.2 Hardware Configuration

We are now going to see how to configure the hardware for the first use of your ESP8266 board.

Thanks to the Adafruit ESP8266 breakout board, this step will be really simple: you just need to plug the FTDI board like on this picture:

Make sure that the FTDI board is set at 5V, or your Adafruit ESP8266 board might not have enough power to function properly.

2.3 Configuring Your ESP8266 Chip

We are now going to configure your ESP8266 chip using the Arduino IDE. This is a great way to use the chip as you will be able to program it using the well-known Arduino IDE, and also re-use several existing Arduino libraries.

First, you need to take a few steps to be able to configure the ESP8266 with the Arduino IDE:

- Start the Arduino IDE and open the Preferences window.
- Enter the following URL: `http://arduino.esp8266.com/package_esp8266com_index.json` into Additional Board Manager URLs field.
- Open the Boards Manager from the **Tools>Board** menu and install the esp8266 platform.

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:

```
// 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 the board you are using. In our case, we need to select 'Adafruit HUZZAH ESP8266'.

After that, we need to put the board in bootloader mode to program it. Again, this is really easy with the Adafruit board. Simply press on the two buttons on the board, and then release the Reset button first. A LED should light up on the board. Then, release the other button.

Now, 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 & make them interact with your Raspberry Pi.

2.4 How to Go Further

You now have a completely usable ESP8266 module, so you are ready to build projects using this chip & use them with your Raspberry Pi.

In the next chapters, we are going to build several projects based on the ESP8266, and make them interact with your Raspberry Pi which will play the role of the home automation hub in your home.

3 Build a Digital Doorbell with the ESP8266

In this first project using the Raspberry Pi & the ESP8266, we are going to build a simple digital doorbell. We will use the ESP8266 as the doorbell button, which will then transmit a message to the Raspberry Pi whenever it is pressed.

Then, the Raspberry Pi itself will play a sound whenever the button is pressed. Because it works via WiFi, the two elements can be placed anywhere in your home. As a test, we will first use a simple LED on the Raspberry Pi to see if the message has been received correctly. Let's start!

3.1 Hardware & Software Requirements

Let's first see what we need for this project. There are two parts in this project: the button connected to the ESP8266 chip, and the Raspberry Pi that will actually be the 'bell' in the project.

For the ESP8266 part of the project, these are the components you will need:

- ESP8266 Huzzah board (https://www.adafruit.com/product/2471)
- Push button (https://www.adafruit.com/products/367)
- 1K Ohm resistor (https://www.sparkfun.com/products/8980)

This is what you will need for the Raspberry Pi:

- Raspberry Pi (https://www.adafruit.com/products/2358)
- Red LED (https://www.sparkfun.com/products/9590)
- 330 Ohm resistor (https://www.sparkfun.com/products/8377)
- Cobbler Kit (https://www.adafruit.com/product/914)

Finally, these are the components you will need for both parts of the project:

- 2x Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

3.2 Hardware Configuration

It is now time to assemble the project. Let's start with the ESP8266. First, place the ESP8266 board on the breadboard. Then, also place the push button on the breadboard.

Then, connect the resistor to one side of the push button. Connect this same pin to pin 5 on the ESP8266 board. Also connect the other side of the resistor to the GND pin on the ESP8266.

Finally, also connect the other side of the push button to the 3V pin of the ESP8266 board.

This is the fully assembled ESP8266 with the push button:

It's much easier to assemble the project around the Raspberry Pi. As we said earlier, we will use a simple LED to visualize when the message is transmitted from the ESP8266 to the Pi. Simply place the LED in series with the 330 Ohm resistor on the breadboard, with the longest pin of the LED connected to the resistor.

Then, connect the other end of the resistor to pin 4 of the Raspberry Pi, and the other end of the LED to a GND pin of the Pi.

This is the assembled project on the Raspberry Pi:

Finally, for the last section of this project, you will also need to have speakers connected to your Raspberry Pi.

3.3 Configuring the ESP8266

We are now going to configure the ESP8266 board for our project, so it will send a message to the Raspberry Pi whenever the push button is pressed. Remember, the goal here is simply to test the project, and make the LED light up if the button is pressed.

As the sketch is quite long, we are only going to see the most important parts here. Note that you can find the whole sketch inside the GitHub repository of the book.

It starts by importing the ESP8266 WiFi library:

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

Then, you will need to enter your WiFi SSID & password:

```
const char* ssid     = "your-wifi-name";
const char* password = "your-wifi-password";
```

You will also need to enter the IP address of your Raspberry Pi, for example:

```
const char* host = "192.168.0.104";
```

Note that you can get the current IP address of your Pi by using the *ifconfig* command. Then, we define the pin on which the button will be connected to:

```
const int buttonPin = 5;
```

After that, in the setup() function of the sketch, we connect the ESP8266 to your WiFi network:

```
// Connect to WiFi
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());
```

In the loop() function of the sketch, we debounce the state of the button. Debouncing means making sure that when we press the button, only one reading is recorded (mechanical buttons have a tendency to 'bounce' when pressed). This is necessary as we really want one press on the button to send one message to the Raspberry Pi. This is the code to read the state of the button, and to debounce it:

```
// Read the state of the button
int reading = digitalRead(buttonPin);

// If the button state changed, due to noise or pressing:
if (reading != lastButtonState) {
  // reset the debouncing timer
  lastDebounceTime = millis();
}

if ((millis() - lastDebounceTime) > debounceDelay) {

  // If the button state has changed:
  if (reading != buttonState) {
    buttonState = reading;

    // Only send command if state is HIGH
    if (buttonState == HIGH) {
      Serial.println(buttonState);
      ringBell();
    }
  }
}

// Save the reading
lastButtonState = reading;
```

As we can see in the code, every press off the button calls the ringBell() function:

```
sendCommand(1);
delay(1000);
sendCommand(0);
```

This function calls the sendCommand() function two times, with a 1 second delay in between. This will simply light up the LED on the Raspberry Pi, leave it on for a second, and put it off again.

This is the function to actually send the message to the Raspberry Pi:

```
void sendCommand(int state) {

  // Use WiFiClient class to create TCP connections
  WiFiClient client;
  const int httpPort = 3000;
  if (!client.connect(host, httpPort)) {
    Serial.println("connection failed");
    return;
  }

  // We now create a URI for the request
  String url = "/digital/7/" + String(state);

  Serial.print("Requesting URL: ");
  Serial.println(url);

  // This will send the request to the server
  client.print(String("GET ") + url + " HTTP/1.1\r\n" +
```

```
                "Host: " + host + "\r\n" +
                "Connection: close\r\n\r\n");
  delay(100);

  // 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");

}
```

You can now already get the complete code from:

https://github.com/openhomeautomation/rpi-esp8266-book

Open the file with the Arduino IDE, and modify it with your own WiFi name & password. Also enter the IP address of your Raspberry Pi.

Then, put your ESP8266 board in bootloader mode, by following the instructions from the previous chapter. After that, select the ESP8266 board in the Arduino IDE, select the correct Serial port, and then upload the code to the board.

You can open the Serial monitor to make sure that it can connect to the WiFi network, but don't press the button yet.

3.4 Configuring the Raspberry Pi

We are now going to see the other part of the project: configuring the Raspberry Pi. This part of the project will be based on the pi-aREST node.js package, that will greatly simplify the process of controlling the LED.

This is the main Node.js file that will accept the connections from the ESP8266 WiFi chip:

```
// Start
var express = require('express');
var app = express();
var piREST = require('pi-arest')(app);

piREST.set_id('34f5eQ');
piREST.set_name('rpi');

var server = app.listen(3000, function() {
    console.log('Listening on port %d', server.address().port);
});
```

You can now get the code from:

https://github.com/openhomeautomation/rpi-esp8266-book

Put the file into a folder on your Raspberry Pi, and then type the following command to install all the required Node.js modules:

```
sudo npm install pi-arest express
```

Be patient, this can take a while on earlier versions of the Raspberry Pi.

3.5 Testing the Project

It is now time to test the project. First, make sure that the ESP8266 board is correctly configured, and that you installed all the required packages on your Raspberry Pi. Then, go to the folder on your Raspberry where the app.js file is located, and type:

```
sudo node app.js
```

This will start the server on your Pi, waiting for commands from the ESP8266. Then, press the doorbell button on the ESP8266: it should instantly light up the LED on the Pi. Then, after 1 second, the LED should go off again.

3.6 Playing a Sound with the Doorbell

We are now going to take the last step in this project and really transform the project into a doorbell, by a press on a button play a sound on the Raspberry Pi. First, we need to modify the ringBell() function on the ESP8266 code:

```
void ringBell() {

  // Use WiFiClient class to create TCP connections
  WiFiClient client;
  const int httpPort = 3000;
  if (!client.connect(host, httpPort)) {
    Serial.println("connection failed");
    return;
  }

  // We now create a URI for the request
  String url = "/doorbell";

  Serial.print("Requesting URL: ");
  Serial.println(url);

  // This will send the request to the server
  client.print(String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n");
  delay(500);

  // 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");

}
```

Upload this modified version of the code on your ESP8266.

We can see that we defined a new */doorbell* route inside the code. We therefore need to create it inside the app.js file. We also need to include the node-aplay module, that we will use to play sounds from Node.js on the Raspberry Pi.

This is the new file for the Raspberry Pi:

```
// Start
var Sound = require('node-aplay');
var express = require('express');
var app = express();

// Route for ringbell
```

```
app.get('/doorbell', function(req, res) {
  new Sound('doorbell.wav').play();
  res.json({data: "Sound played"});
});

var piREST = require('pi-arest')(app);

// Set parameters
piREST.set_id('34f5eQ');
piREST.set_name('rpi');

var server = app.listen(3000, function() {
    console.log('Listening on port %d', server.address().port);
});
```

By default, this will play a file called doorbell.wav on the Raspberry Pi, but you can of course use any file you want. Just place this file inside the same folder as app.js on your Pi.

Finally, let's test our doorbell. First, install the node-aplay module with:

```
sudo npm install node-aplay
```

Finally, start the project again with:

```
sudo node app.js
```

Now, press the button again on the ESP8266 board, and you should hear the doorbell sound playing on your Raspberry Pi. Congratulations, you just made your own WiFi doorbell!

3.7 How to Go Further

Let's summarize what we did in this chapter. We built a digital WiFi doorbell based on the ESP8266 WiFi chip & the Raspberry Pi.

I hope that this first project using the Raspberry Pi & the ESP8266 chip really showed you what is possible to achieve with those two really different components.

In the next chapters, we are going to explore other ways to integrate the ESP8266 chip by using the Raspberry Pi as a hub.

4 WiFi Temperature & Humidity Sensor

In this chapter, 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 from the Raspberry Pi.

To do so, we will run a simple web server on the ESP8266 chip. The Raspberry Pi will then access this data via WiFi, and display it in a graphical interface.

4.1 Hardware & Software Requirements

For this project, you will of course need an ESP8266 chip. You can for example use the Adafruit ESP8266 board, that we will use in the rest of this book.

You will also need the Raspberry Pi board that we have been using through this whole book.

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.

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

- DHT11 sensor (https://www.adafruit.com/products/386)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/9194)

You will also need the DHT library. You can install it from the Arduino library manager, that you can access from **Sketch>Include Libraries>Manage Libraries** inside the Arduino IDE.

4.2 Hardware Configuration

We are first going to see how to configure the hardware to use the ESP8266 board. First, place the Adafruit ESP8266 board on the breadboard.

Once this is done, simply insert the DHT11 sensor on the breadboard. Then, connect the left pin of the sensor to the 3.3V pin of the ESP8266, the right pin to GND (blue power rail), and the pin next to VCC to the GPIO5 pin on your ESP8266 board. This is the final result, not showing the USB-to-Serial FTDI converter:

Make sure that you connected everything according to the instructions above, or you won't be able to continue.

4.3 Testing the Sensor

We are now going to test the sensor. Again, remember that we are using 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:

```
// 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()` function, 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/rpi-esp8266-book

You can now paste this code in the Arduino IDE. Then, go in **Tools>Boards**, and select the Adafruit ESP8266 board from the list.

After that, we need to put the board in bootloader mode, so we can program it. To do so, refer to the first chapter of this part of the book.

Now, 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 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.

4.4 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 make the sensor data accessible by the Raspberry Pi.

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.

It starts by including the required libraries for this project:

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

Then, 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";
```

We also declare the aREST API, that will help us control the board remotely from the Raspberry Pi:

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

Now, we also declare two variables that will contain the measurements made by the sensor:

```
int temperature;
int humidity;
```

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 expose the two measurement variables to the aREST API, so they can be access from the outside:

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

After that, we set the name & ID of the board:

```
rest.set_id("1");
rest.set_name("sensor_module");
```

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, and handle the request:

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

Then, we read data from the sensor:

```
temperature = dht.readTemperature();
humidity = dht.readHumidity();
```

It's now time to upload the sketch the board. You can grab the complete code from:

https://github.com/openhomeautomation/rpi-esp8266-book

Follow the instructions we saw in the previous section to upload the code to the board. Then, open the Serial monitor to get the IP address of the board, you will need it soon.

4.5 Connecting the Board to Your Raspberry Pi

We are now going to see how to connect the board to your Raspberry Pi, so it can display the data in a nice graphical interface.

The only thing that you need to change to run this interface is the IP address of your ESP8266 board inside the app.js file:

```
rest.addDevice('http','192.168.0.100');
```

You can get all the code from:

https://github.com/openhomeautomation/rpi-esp8266-book

Place all the files into a folder on your Pi, navigate to this folder with a terminal, and type:

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

Be patient, this step can take a while. After that, type:

```
sudo node app.js
```

This will start the application on your Raspberry Pi. Then, navigate to the URL of your Raspberry Pi on port 3000, for example:

```
http://192.168.0.104:3000
```

You should immediately see the interface showing the temperature & humidity readings made by the ESP8266 board:

4.6 How to Go Further

Let's summarize what we achieved in this project. We built a WiFi measurement station based on the ESP8266 WiFi chip. We used a sensor to measure the local temperature & humidity. Then, we displayed all the measurements on a web page that was served by the Raspberry Pi.

There are many things you can do to improve this project. You can for example deploy more of those sensors boards in your home, and display all the measurements on a single page on your Pi.

5 Control a Remote Lamp from Your Raspberry Pi

In this chapter, we are going to use the ESP8266 chip to control a lamp remotely via WiFi. We will use the Raspberry Pi as a hub, that you will access from a web-based interface, and that will relay the commands via WiFi.

Here, we are going to use the onboard processor of the ESP8266 to host a small web server, that will accept commands coming from your Raspberry Pi. These commands will be generated on the Pi itself, using a simple interface that you can access from your computer or a mobile device.

5.1 Hardware & Software Requirements

For this project, you will of course need an ESP8266 chip. As for the whole book, I used the Adafruit 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 5V 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 Huzzah board (https://www.adafruit.com/product/2471)
- FTDI USB module (https://www.adafruit.com/products/284)
- PowerSwitch Tail Kit (https://www.sparkfun.com/products/10747)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/9194)

You will also need the Raspberry Pi that you have been using through this whole book.

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. Also make sure that the switch on the lamp itself is on the 'on' position.

On the software side, if it's not done yet, you will need the latest version of the Arduino IDE that you can get from:

http://www.arduino.cc/en/Main/Software

5.2 Hardware Configuration

We are first going to see how to configure the hardware to use the ESP8266 board. First, place the board on the breadboard.

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 lamp I used as a test:

5.3 Controlling the Lamp Remotely

We are now going to write the code required to control our lamp remotely. The ESP8266 will have to handle requests coming from the Raspberry Pi, and then control the lamp 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 including the correct libraries:

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

After that, we initialize the aREST libary:

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

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);
```

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

```
pinMode(5,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();
```

```
if (!client) {
  return;
}
while(!client.available()){
  delay(1);
}
rest.handle(client);
```

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

https://github.com/openhomeautomation/rpi-esp8266-book

You can now paste this code in the Arduino IDE. Then, go in **Tools>Boards**, and select Adafruit ESP8266 board.

After that, we need to put the board in bootloader mode, so we can program it. Follow the instructions found in the chapter where we learned how to use the ESP8266 chip.

Now, upload the code to the board, and open the Serial monitor when this is done. Also set the Serial monitor speed to 115200.

Then, look in the Serial monitor to get the IP address of your board.

5.4 Raspberry Pi Interface

We are now going to code the interface for this project. The goal is to make a really simple interface, with two buttons, that when pressed will send commands to the ESP8266 to switch the lamp on or off.

Let's start with the main Node.js file. This will use the node-aREST module, that will be in charge of the communication between the Raspberry Pi & the ESP8266. This is the complete file:

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

// View engine
app.set('view engine', 'jade');

// Set public folder
app.use(express.static(__dirname + '/public'));

// node-aREST
var rest = require("arest")(app);
rest.addDevice('http','192.168.0.103');

// Interface routes
app.get('/', function(req, res){
  var devices = rest.getDevices();
  res.render('interface', {devices: devices});
});

// Start server
var server = app.listen(3000, function() {
    console.log('Listening on port %d', server.address().port);
});
```

The only thing you need to change in this file is the address of your ESP8266 chip, that you got before when configuring the chip:

```
rest.addDevice('http','192.168.0.103');
```

Now, let's see the interface file, which is coded in language Jade. The interface is really simple here: we will just define two buttons, one 'on' button, and one 'off' button.

This is the complete interface.js file:

```
doctype
html
  head
    title Home Automation Interface
    link(rel='stylesheet',
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.0/css/bootstrap.min.cs
    link(rel='stylesheet', href="/css/interface.css")
    script(src="https://code.jquery.com/jquery-2.1.1.min.js")
    script(src="/js/interface.js")
  body
```

```
    .container
      h1 Home Automation Interface
      .row.voffset
        .col-md-4
          div Lamp
        .col-md-4
          button.btn.btn-block.btn-lg.btn-primary#on On
        .col-md-4
          button.btn.btn-block.btn-lg.btn-danger#off Off
```

Finally, we define a file called interface.js to handle the clicks on the buttons. A click on a button will call the appropriate command on the ESP8266 chip:

```
$(document).ready(function() {

  // Click on buttons
  $("#on").click(function() {
    $.get('/lamp_control/digital/5/1');
  });

  $("#off").click(function() {
    $.get('/lamp_control/digital/5/0');
  });

});
```

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

https://github.com/openhomeautomation/rpi-esp8266-book

Copy all the files inside a folder on your Pi, go to this folder with a terminal, and type:

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

Be patient, this step can take a while. Once it is done, type:

```
node app.js
```

This will start your app on port 3000. You can now access the interface with your favorite web browser by typing the URL of your Raspberry Pi, followed by port 3000:

```
http://192.168.0.104:3000
```

You should then see the interface:

Now, try clicking on of of those buttons: you should immediately here the click of the PowerSwitch Tail, and the lamp should turn on.

5.5 How to Go Further

In this project, we built a remote lamp controller using the ESP8266 WiFi chip, that we controlled using our Raspberry Pi. In this project, we really used the Raspberry Pi as a home automation hub, hosting the interface to control the lamp.

Of course, you can use what you learned in this project and add more lamps to the project. It's really easy to add more buttons to the interface, and control more lamps from the same central interface.

6 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 Raspberry Pi.

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!

6.1 Hardware & Software Requirements

For this project, you will of course need an ESP8266 chip. As for the whole book, I used the Adafruit ESP8266 breakout board.

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:

You will also need the Raspberry Pi board that you have been using in the whole book. Also, you will need to connect it to a set of external speakers, as we will make it play some alarm sound.

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

- 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.

6.2 Hardware Configuration

We are first going to see how to configure the hardware to use the ESP8266 board. First, place the ESP8266 board on the breadboard.

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 3V pin of the board, and the GND pin of the sensor goes to the GND pin of the board. Finally, connect the SIG pin of the sensor to the GPIO pin 5 of the ESP8266 board. 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.

Make sure that you connected everything according to the instructions above, or you won't be able to continue.

6.3 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/rpi-esp8266-book

It's now time to test the project. You can paste the code in the Arduino IDE. Then, go in **Tools>Boards**, and select the Adafruit ESP8266 board. Also select the correct Serial port that corresponds to the FTDI converter your are using.

After that, we need to put the board in bootloader mode, so we can program it. To do so, please refer to the chapter where we saw how to configure the ESP8266 chip.

We can then move to coding the alarm system itself.

6.4 Creating Our Alarm System

We are now going to write the code for the alarm system, that will run on your Raspberry Pi. 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 Raspberry Pi.

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
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:

```
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:

```
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:

```
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:

```
$.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/rpi-esp8266-book

It's now time to test the project. Get the code from the GitHub repository, and place all the files into a given folder on your Raspberry Pi. 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 the IP address of your Pi:

```
http://192.168.0.104:3000/
```

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

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:

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

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

6.5 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.

7 Use the Raspberry Pi as the Hub of ESP8266 Devices

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 based on the ESP8266, as well as the lamp control project, and integrate these projects into a single interface running on your Pi.

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.

7.1 Hardware & Software Requirements

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

You will of course need the Raspberry Pi board that you have been using in this whole book.

For the temperature & humidity sensor module requirements, please check again the chapter called *WiFi Temperature & Humidity Sensor*. For the lamp control module requirements, please check again the chapter called *Control a Remote Lamp from Your Raspberry Pi*.

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

https://github.com/marcoschwartz/aREST

To install an Arduino library, first download the library from the GitHub repository. Then, go into the Arduino IDE, and click on Sketch>Include Library> Add .ZIP Library. Finally, select the file that you just downloaded.

7.2 Hardware Configuration

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

For the ESP8266 temperature & humidity sensor module configuration, please check again the chapter called *WiFi Temperature & Humidity Sensor*. For the lamp control module configuration, please check again the chapter called *Control a Remote Lamp from Your Raspberry Pi*.

7.3 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.

To make things easier, we are going to use again 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:

```
// 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:

https://github.com/openhomeautomation/rpi-esp8266-book

Then, modify it with your own parameters, and then upload the code to the board, using the instructions we saw in the previous chapters.

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:

```
// 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!

7.4 Creating the Interface

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

This interface will be running on your Raspberry Pi, and will be responsive as well, meaning you can use it from your smartphone or tablet for example.

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 Raspberry Pi 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 of the book:

https://github.com/openhomeautomation/rpi-esp8266-book

Then, modify it with your own parameters, and put all the files in a folder on your Raspberry Pi.

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 the IP address of your Raspberry Pi, followed by port 3000:

```
http://192.168.0.104:3000
```

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:

7.5 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 your Raspberry Pi, 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.

8 Conclusion

I hope you enjoyed reading this book and I also hope you learned a lot by doing so. There are of course many things you can do right now to build home automation systems based on what we saw in this book.

First, I recommend that you really do all the projects of this book yourself. This will give you a good insight of what is possible to realize with the Raspberry Pi along with the ESP8266, and you will learn much more in doing than in just reading.

From there, really ask yourself what you want to do with your Raspberry Pi, and proceed with small steps. First, ask yourself what you can add directly to your Pi: which sensors? Do you need a camera? Do you need to control devices remotely?

Then, think about extending these functions: what do you need to monitor/control in other rooms? How many ESP8266 boards or other systems do you need to do that? With which wireless technology?

Finally, ask yourself the question of which web services do you need. Do you want a simple dashboard to monitor your home? Do you want to receive automated alerts if some motion is detected in your home? This is all possible, and you will have no problem to implement it with what you learned in this book. Good luck!

Enjoyed what you've just read in this book? Discover even more tutorials, books & videos by visiting our website at https://www.openhomeautomation.net/

9 Resources

The following is a list of the best resources concerning home automation with the Raspberry Pi & the ESP8266. I organized this chapter in different categories so it is easier for you to find the information you need.

9.1 General Information

- Open Home Automation (http://www.openhomeautomation.net): The companion website of this book, where you will find many more projects using Raspberry Pi & the ESP8266 to build home automation projects.
- Adafruit Learning System (https://learn.adafruit.com): An online learning platform with a selection of high-quality step-by-step articles on making things in general. Many projects use the Raspberry Pi and the ESP8266, and some are about home automation.

9.2 Components

- SparkFun (https://www.sparkfun.com): A website selling many components that can be interfaced with the Raspberry Pi. All their products are open-source and you can download the source files directly from their product descriptions.
- Adafruit (http://www.adafruit.com): A company based in New York that sells high quality products for the Raspberry Pi platform, as well as ESP8266 boards.
- SeeedStudio (http://www.seeedstudio.com): A Chinese company that sells many original products for the Raspberry Pi platform. They also offer their own PCB production & assembly services.

9.3 Raspberry Pi Resources

- Raspberry Pi Foundation (http://www.raspberrypi.org): The official website of the Raspberry Pi, packed with lots of additional resources and tutorials.
- RasPi.TV (http://raspi.tv/): A website & Youtube channel with a lot of high quality tutorials using the Raspberry Pi.

9.4 ESP8266 Resources

- ESP8266 Community Forum (http://www.esp8266.com): The official website of the Raspberry Pi, packed with lots of additional resources and tutorials.