# Discover the aREST Framework

Marco Schwartz

## Contents

**Legal** **3**

**About the Author** **3**

**About the Accompanying Website** **4**

**Introduction** **4**

0.1 The aREST Framework . . . . . 4

0.2 Structure of the Book . . . . . 5

0.3 Prerequisites . . . . . 6

0.4 Getting Started . . . . . 7

**1 Learning the basics of aREST** **8**

1.1 Control an Arduino board via the Serial port . . . . . 8

1.2 Control an Arduino board Remotely via Ethernet . . . . . 13

1.3 Access Your Arduino Boards Wirelessly via WiFi . . . . . 18

1.4 Use XBee Communications with Arduino & aREST . . . . . 26

1.5 Learn to use Bluetooth LE to control Arduino boards . . . . . 31

1.6 Control your Raspberry Pi board remotely with aREST . . . . 39

1.7 Control an ESP8266 module remotely with aREST . . . . . 44

1.8 Run a User Interface on an ESP8266 module . . . . . 50
1.9 Example project: Control a Lamp Remotely using the ESP8266 55
1.10 How to Go Further . . . . . 59

**2 Build local applications to control your projects 60**

2.1 Control an Arduino Board from a Web Page via Ethernet . . . 60
2.2 Control & Monitor an ESP8266 module from a Web page . . . 67
2.3 Learn to Control & Monitor Arduino boards via Ethernet . . . 72
2.4 Build a Web-based Application to Control Arduino boards via WiFi . . . . . 79
2.5 Get Access to Your Raspberry Pi From a Dashboard . . . . . 89
2.6 Record Data From ESP8266 Devices . . . . . 93
2.7 Deploy the aREST Application on a Raspberry Pi . . . . . 102
2.8 Record & Plot data from Your aREST Devices . . . . . 106
2.9 Example project: Simple Wireless Home Automation System . 115
2.10 How to Go Further . . . . . 130

**3 Access Your Boards From the Cloud 131**

3.1 Access your Arduino Boards from Anywhere . . . . . 131
3.2 Control your Raspberry Pi using a Cloud API . . . . . 136
3.3 Use Your ESP8266 Boards as Internet of Things Modules . . . 141
3.4 Build Your Own Dashboard to Control your Devices From the Cloud . . . . . 146
3.5 Control Several Boards from a Single Dashboard . . . . . 152
3.6 Deploy your own aREST cloud server . . . . . 159
3.7 Example project: Remote Data Monitoring from Multiple Boards 167
3.8 How to Go Further . . . . . 173

**4 Build Mobile Applications using aREST** **173**

4.1 Build Mobile Applications for Android . . . . . 173

4.2 Build Mobile Applications for iOS . . . . . 182

4.3 Example Project: Control Your Home From Your Smartphone 187

4.4 How to Go Further . . . . . 191

**Conclusion** **191**

# 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, aREST.io, which you can easily find by going at http://arest.io/. On this website you will find more resources about the aREST framework.

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@arest.io

# Introduction

## 0.1 The aREST Framework

The story of the aREST framework starts in mid-2014, when I was already building a lot of home automation & Internet of Things projects, mainly using the Arduino platform.

Back then, I was basically reinventing the wheel every time I was creating a new project. For example, to access a measurement done by a board, I was re-writing a lot of code, customized for the project I was working on.

At the same time, I got my first Arduino Yun, and I experimented with a sketch that implemented a REST-like API for the board. This sketch was given you access to the main functions of the board via WiFi, without having to modify the code for every new project you wanted to create.

I was immediately seduced by the concept: have a code that you could write once, upload to the board, and that then give you access to all the functions of the project remotely.

Therefore, I decided to create my own project that would implement the same concept. I added a lot of improvements. The main improvement was to

encapsulate everything in a library, which was much more convenient than having an Arduino sketch containing all the functions and that you needed to copy & paste at every new project.

I also made the library compatible with most of the Arduino boards, as well as most of the ways of communication: WiFi, Serial, Ethernet, XBee ... I then had a library that could be used in any situation, to access your Arduino board using the same set of commands, independently of the hardware you were using. The aREST library was born.

Since then, I added many things to the aREST framework. I for example extended the library outside of the Arduino space, making it compatible with the Raspberry Pi and the ESP8266. I also developed the server-side part of the framework, so you can build nice interfaces to control your boards remotely. Finally, in the end of 2015 I launched the aREST cloud API, which allows anybody to control their aREST projects from anywhere in the world.

More than 100 people try out the library every day now, and at the time the book was written there was more than 200 users of the aREST cloud platform.

By writing this book, I wanted to share with you all the details about the aREST framework, to allow you to create your own applications based on it. In order to teach you the aREST framework, I will use step-by-step examples that illustrates all the aspects of the framework.

Note that most of the examples of this book are in the home automation and IoT space, but you can of course use the aREST framework for all the fields you can imagine, like mobile robots, drones, or healthcare.

## 0.2 Structure of the Book

This book is organized in 4 parts. In each part, you will find step-by-step projects that illustrate essential parts of the framework.

The first part is all about controlling single boards using the aREST framework. That's where you will learn all the commands that you can use inside the framework. For example, you will learn how to control your boards via WiFi, make measurements remotely, and define your own functions that can be called remotely.

The second part of the book is about controlling your aREST projects using graphical interfaces. This is where your will learn to control Arduino boards from a web page, build server-side applications to control many boards, and record & plot data right in your web browser.

In the third part of the book, we will see all the features of the aREST cloud platform. You will learn how to control individual boards from anywhere in the world, monitor them, and build online dashboards to control your projects from anywhere. You'll even learn how to deploy your own aREST-compatible cloud platform!

Finally, the last part of the book is about building mobile applications to control your aREST projects. We will see how to build and deploy Android and iOS applications so you can control your aREST projects from your smartphone or tablet.

## 0.3 Prerequisites

In this book, you will see that I mix several platforms, like Arduino, the Raspberry Pi, and the ESP8266. However, as the aREST framework is universal, you can perfectly apply everything you will learn in this book to Arduino for example.

Therefore, any prior knowledge in at least one of those platforms is really recommend to follow the content of this book. However, you don't need to be an expert in embedded electronics: we'll really keep the hardware part simple, to focus on the aREST framework.

In terms of programming languages, I will mainly use two of them: C/C++ for Arduino & the ESP8266, and JavaScript for the rest (via Node.js and the Meteor platform).

Any knowledge in those languages is also recommended for this book. However, I will really take things from the very basics to more advanced projects, so you can perfectly learn project after project.

In terms of required hardware, I will provide a list of all the required components in every section of the book. I will also give an overview of what you need in terms of software & hardware in the next section.

## 0.4 Getting Started

Before you can actually get started and build your first projects using the aREST framework, we need to install & get a few things.

You will the Arduino IDE for most of the projects of the book. You can get it from:

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

Once it's installed, open it, and go to **Sketch>Include Library>Manage Libraries** and look for the library called *aREST*. Install it, you will use it in nearly all projects of the book.

To use aREST on your Raspberry Pi, you will first need to have a fully functional Raspberry Pi, with Raspbian installed on it. If that's not done yet, follow the instructions from:

https://www.raspberrypi.org/documentation/installation/installing-images/README.md

You will also need to have Node.js installed on your Raspberry Pi. If you don't have it installed yet, install it using the instructions from:

http://weworkweplay.com/play/raspberry-pi-nodejs/

To build server-side applications, you will need the Meteor platform to be installed on your computer. You can get it from:

https://www.meteor.com/install

About the hardware itself, as I mentioned earlier you will get the full list of the required hardware in each project. However, there are three boards that you can be sure you will use in this book several times per chapter.

The first board is the Arduino Uno board. However, you can use other Arduino boards as well, like the Mega.

Then, you will often use a Raspberry Pi. It can be a Raspberry Pi B, B+, or 2 (or even a Zero with a WiFi adapter)

Finally, you will need an ESP8266 board. I will mainly use the Adafruit ESP8266 board, but any ESP8266 board can be used for the projects of this book.

# 1 Learning the basics of aREST

In this first chapter of the book, we are going to learn the foundations of the aREST framework. We are going to learn the basic functions of the framework by going through example projects using aREST. To illustrate all those basic functions of the aREST framework, we are going to use several boards like Arduino, the Raspberry Pi, and the ESP8266 WiFi chip.

At the end of the chapter, you will able to control all your boards remotely using the same aREST commands, via Ethernet, WiFi, or even XBee.

## 1.1 Control an Arduino board via the Serial port

In this very first project of the book, we are going to cover the basics of the aREST framework. To illustrate those basic commands, we are simply going to plug an Arduino board to the USB port of your computer, and control it using the aREST library, via the Serial port. Let's start!

### 1.1.1 Hardware & Software Requirements

For this first project, you will only need a board that is compatible with aREST and Serial communications. In this section, I will be using an Arduino Uno board, but you could also use an Arduino Mega or Due.

Of course, you need to have the latest version of the Arduino IDE installed, as well as the aREST Arduino library. Please go through the *Getting Started* section of the previous chapter if that's not done yet.

### 1.1.2 Hardware Configuration

As we will simply test the aREST library with the USB to Serial communications, we don't actually need to connect any hardware components to our Arduino board. Simply connect the board to your computer using a USB cable.

### 1.1.3 Learning the Basic aREST Commands

We are now going to configure your board, so we can send aREST commands to control it via the Serial port. This is the complete sketch for this part:

```
// Libraries
#include <SPI.h>
#include <aREST.h>

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

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

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

}

void loop() {

  // Handle REST calls
  rest.handle(Serial);

}
```

This sketch is really basic, but already illustrates a lot about the aREST framework. It always starts by including the aREST library:

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

Then, we create the aREST instance:

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

In the setup() function of the sketch, we also set an ID and name that identify the board:

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

Finally, in the loop() function, we process the requests sent to the aREST library:

```
rest.handle(Serial);
```

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

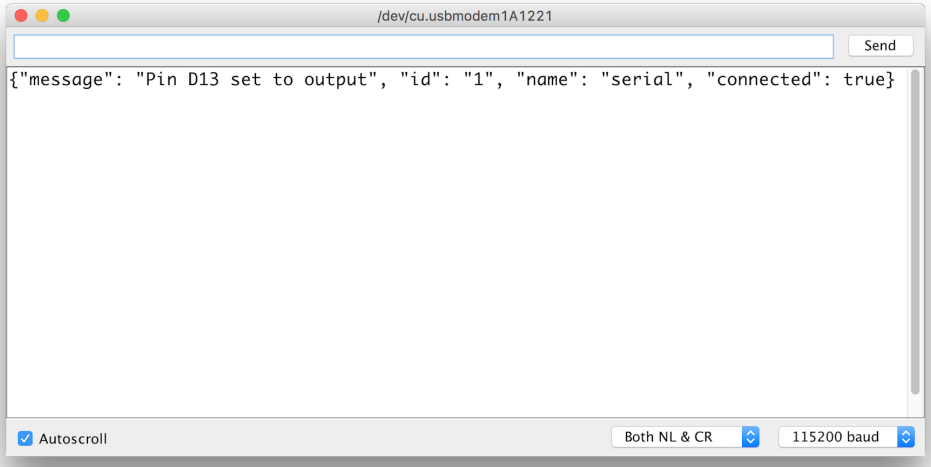
https://github.com/marcoschwartz/discover-arest

You can now open the Arduino IDE, and put this first sketch inside the IDE. Then, upload the sketch to the Arduino board. After, that, open the Serial monitor, and make sure that the Serial speed is set to 115200, and the 'Carriage Return' option is selected.

Now, type the following command and send it to the board:

```
/mode/13/o
```

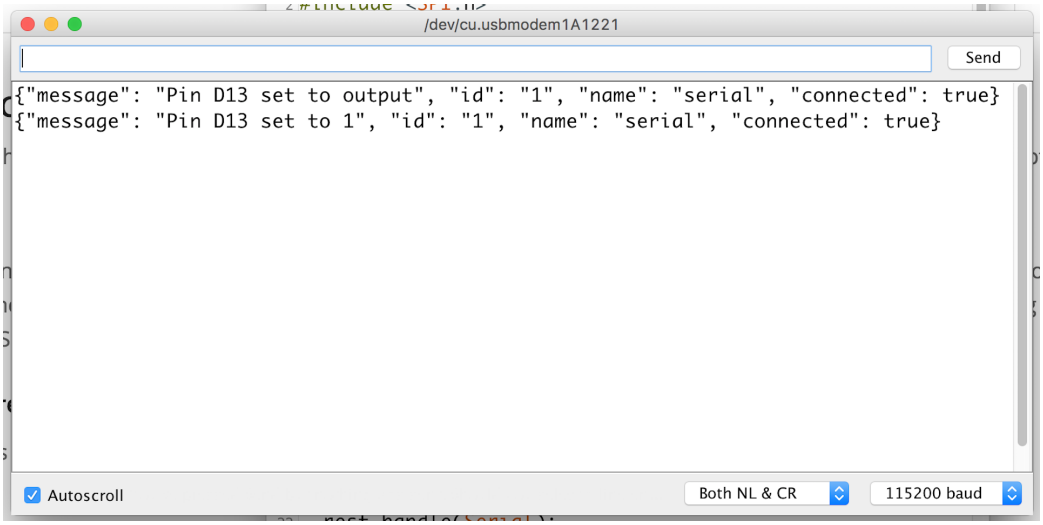
This will make pin number 13, which is connected to a LED on the Arduino Uno board, to an output. You will also see the confirmation inside the Serial monitor:




After that, type:

`/digital/13/1`

This will make a digitalWrite() command to pin 13, and it will turn the LED on. If that works, it means that aREST is working correctly. As earlier, you will also get the confirmation in the serial port:



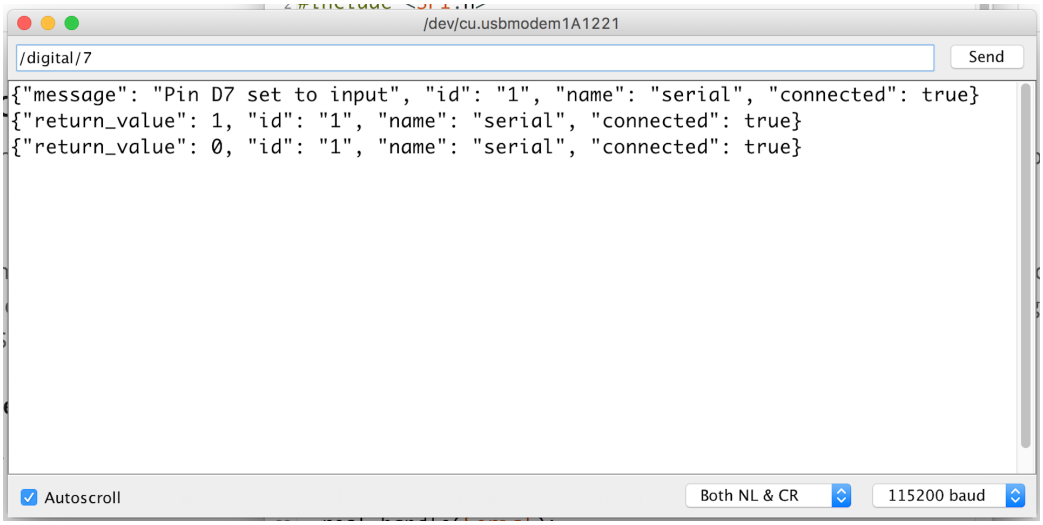
Let's now try to read data from inputs. You can simply read data from pin 7 by typing first:

```
/mode/7/o
```

Followed by:

```
/digital/7
```

This will return the value of pin 7 in the Serial monitor:

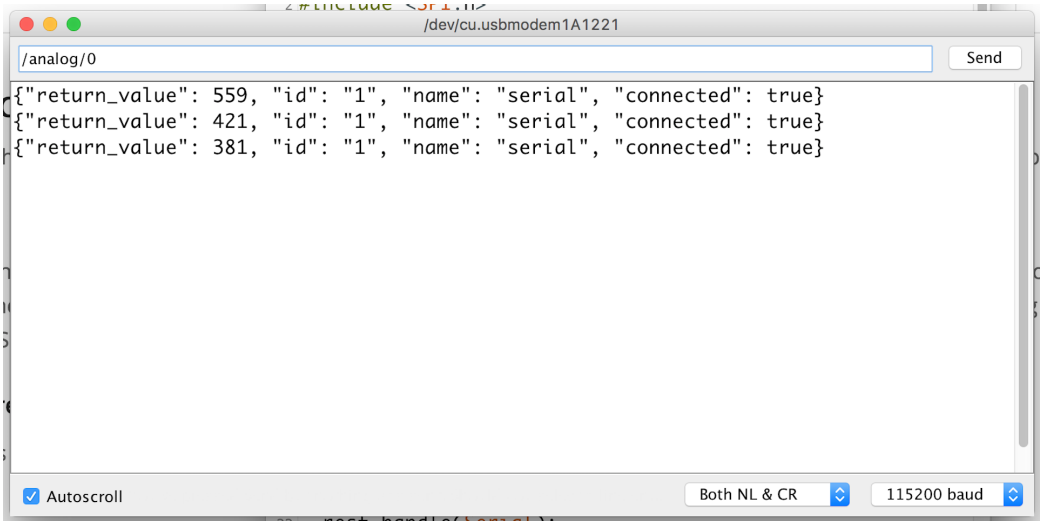



To change the value of pin 7 of the board, you can simply connect a wire cable between the 5V pin and pin 7.

We can also read analog data from the Arduino board via aREST. At the moment there is nothing connect to the Arduino board on analog pins, but you will be able to read out analog fluctuations on this pin. Simply type:

```
/analog/0
```

This will read the value on the analog pin A0, and return a value between 0 and 1023:




From this very first project of the book, you should know already have the foundations of the aREST framework. You can already experiment with those commands now to really get familiar with them, as we will use them in the whole book.

## 1.2 Control an Arduino board Remotely via Ethernet

In this section, we are going to see what really makes the strength of the aREST framework: being able to control boards remotely via a network connection.

We are going to control an Arduino board, but this time using the Arduino Ethernet shield. We will also learn about additional aREST commands.

### 1.2.1 Hardware & Software Requirements

You will need a board that is compatible with the Arduino Ethernet shied, like an Arduino Uno that was used for this guide.

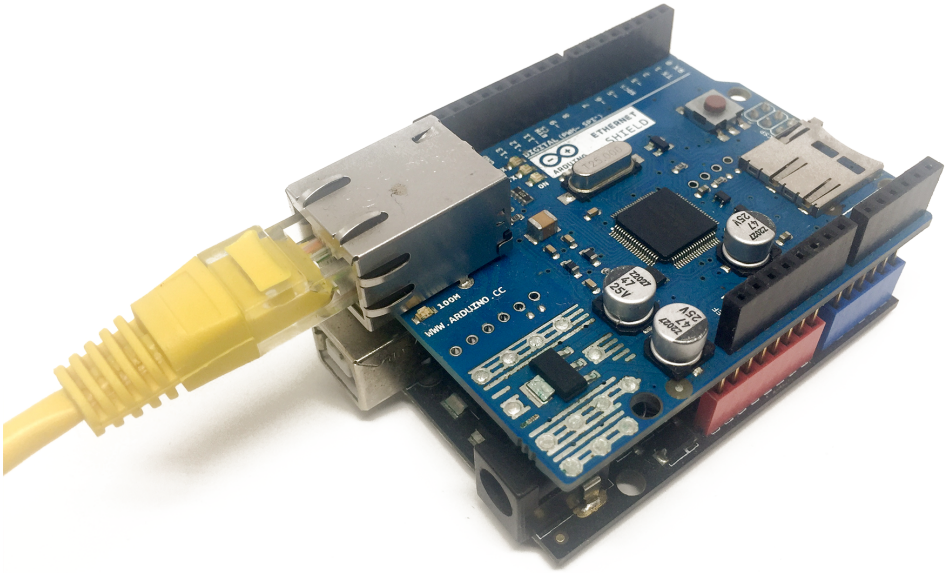
You will also need a the Arduino Ethernet shield or a similar, one LED, one 330 Ohm resistor, a breadboard, and some jumper wires.

- Arduino Uno (https://www.sparkfun.com/products/11021)

- Arduino Ethernet shield (https://www.sparkfun.com/products/11166)
- LED (https://www.sparkfun.com/products/9590)
- 330 Ohm resistor (https://www.sparkfun.com/products/8377)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

### 1.2.2 Hardware Configuration

First, place the Ethernet shield on top of the Arduino board, and plug an Ethernet cable between the board and your router:




For the LED, simply connect it in series with the resistor, with the longest pin of the LED connected to the resistor.

Then, connect the remaining pin of the resistor to pin 6 of the Arduino board, and the remaining pin of the LED to the GND pin of the Arduino board.

### 1.2.3 Controlling the Board via Ethernet

We are now going to configure the Arduino board so it can be access via your local network, using aREST commands. This is the complete sketch for this

part:

```
// Libraries
#include <SPI.h>
#include <Ethernet.h>
#include <aREST.h>

// Enter a MAC address for your controller below.
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0E, 0xFE, 0x40 };

// IP address in case DHCP fails
IPAddress ip(192,168,2,2);

// Ethernet server
EthernetServer server(80);

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

// Variables to be exposed to the API
int temperature;
int humidity;

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

  // Init variables and expose them to REST API
  temperature = 24;
  humidity = 40;
  rest.variable("temperature",&temperature);
  rest.variable("humidity",&humidity);

  // Give name and ID to device
  rest.set_id("008");
  rest.set_name("dapper_drake");
```

```
  // Start the Ethernet connection and the server
  if (Ethernet.begin(mac) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    // no point in carrying on, so do nothing forevermore:
    // try to congifure using IP address instead of DHCP:
    Ethernet.begin(mac, ip);
  }
  server.begin();
  Serial.print("server is at ");
  Serial.println(Ethernet.localIP());

}

void loop() {

  // listen for incoming clients
  EthernetClient client = server.available();
  rest.handle(client);

}
```

As you can notice, there are several similarities between this sketch and the basic Serial example that we saw in the first project.

Here however, there are some details that you need to modify in the code. You need to set the MAC address of your Ethernet shield, which you can find on the bottom of the shield:

```
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0E, 0xFE, 0x40 };
```

It's also time to introduce a new feature of aREST: variables. Variables can be used to automatically fetch values by using aREST commands. This is perfect to get measurements from a sensor for example. Here, to simplify things, we will simply define fixed values for two variables:

```
temperature = 24;
humidity = 40;
rest.variable("temperature",&temperature);
rest.variable("humidity",&humidity);
```

Finally, in the loop() part of the sketch, we handle the request in a very similar fashion to the first example of the book:

```
EthernetClient client = server.available();
rest.handle(client);
```

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

https://github.com/marcoschwartz/discover-arest

You can now configure your board with the code from this section. Don't forget to modify the MAC address inside the code.

Then, open the Serial monitor. You should see the IP address of the board being displayed. I will assume for the rest of this chapter that it is "192.168.115.102". After that, go to your favorite web browser, and type:

```
192.168.115.102/id
```

You should instantly get the confirmation from the board in JSON format:

```
{
  "id": "008",
  "name": "dapper_drake",
  "connected": true
}
```

Congratulations, you can now control your Arduino projects remotely via Ethernet! Let's try more commands, for example to light up the LED on pin 6. First, go back to your browser and type:

```
192.168.115.102/mode/6/o
```

Followed by:

```
192.168.115.102/digital/6/1
```

This should instantly light up the LED connected to pin 6, with the same commands that we used in the first project of this book.

### 1.2.4 More aREST Commands

We'll use this project to illustrate the behavior of additional aREST commands. Remember the variables that we exposed to the API? We can call them easily from the web browser, for example with:

```
192.168.115.102/temperature
```

You should immediately get the answer back in JSON format:

```
{
  "temperature": 24,
  "id": "008",
  "name": "dapper_drake",
  "connected": true
}
```

You can now access any variable that you define on your board via your web browser! This is something you can use just for your own information, for your own web applications, or as we will see in the second chapter of this book, for aREST server-side applications.

## 1.3 Access Your Arduino Boards Wirelessly via WiFi

In this project, we are going to control aREST projects wirelessly, via WiFi. We are going to connect the CC3000 WiFi chip to an Arduino board, and access it wirelessly. We will also use this chapter to explore more functions of the aREST framework.

### 1.3.1 Hardware & Software Requirements

This project is based on Arduino along with a CC3000 WiFi chip. I will use an Arduino Uno and an Adafruit CC3000 breakout board, but you can of course use equivalent components.

This is the list of the components that you will need for this project:

- Arduino Uno (https://www.sparkfun.com/products/11021)
- Adafruit CC3000 breakout board (https://www.adafruit.com/products/1469)
- LED (https://www.sparkfun.com/products/9590)
- 330 Ohm resistor (https://www.sparkfun.com/products/8377)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

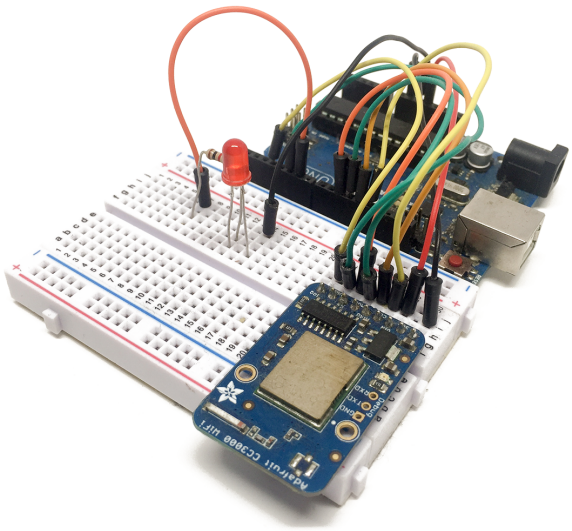
You will also need to install the Adafruit CC3000 WiFi library. You can install it from the Arduino library manager, by going to **Sketch>Include Library>Manage Libraries** and search for the library from there.

### 1.3.2 Hardware Configuration

First, place the CC3000 board on the breadboard. Then, connect the IRQ pin of the CC3000 board to pin number 3 of the Arduino board, VBAT to pin 5, and CS to pin 10. Then, you need to connect the SPI pins to the Arduino board: MOSI, MISO, and CLK go to pins 11,12, and 13, respectively. Finally, take care of the power supply: Vin goes to the Arduino 5V, and GND to GND.

For the LED, simply connect it in series with the resistor, with the longest pin of the LED connected to the resistor. Then, connect the remaining pin of the resistor to pin 6 of the Arduino board, and the remaining pin of the LED to the GND pin.

This is the final result:



### 1.3.3 Controlling the Board via WiFi

We are now going to configure the Arduino board so you can control your board via WiFi. This is the complete sketch for this part:

```
// Import required libraries
#include <Adafruit_CC3000.h>
#include <SPI.h>
#include <aREST.h>
#include <avr/wdt.h>

// These are the pins for the CC3000 chip
#define ADAFRUIT_CC3000_IRQ   3
#define ADAFRUIT_CC3000_VBAT  5
#define ADAFRUIT_CC3000_CS    10

// Create CC3000 instance
Adafruit_CC3000 cc3000 = Adafruit_CC3000(ADAFRUIT_CC3000_CS,
  ADAFRUIT_CC3000_IRQ, ADAFRUIT_CC3000_VBAT, SPI_CLOCK_DIV2);
// Create aREST instance
aREST rest = aREST();
```

```
// Your WiFi SSID and password
#define WLAN_SSID       "your_wifi_name"
#define WLAN_PASS       "your_wifi_password"
#define WLAN_SECURITY   WLAN_SEC_WPA2

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

// Server instance
Adafruit_CC3000_Server restServer(LISTEN_PORT);

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

  // Function to be exposed
  rest.function("led",ledControl);

  // Give name and ID to device
  rest.set_id("008");
  rest.set_name("mighty_cat");

  // Set up CC3000 and get connected to the wireless network.
  if (!cc3000.begin())
  {
    while(1);
  }
  if (!cc3000.connectToAP(WLAN_SSID, WLAN_PASS, WLAN_SECURITY)) {
    while(1);
  }
  while (!cc3000.checkDHCP())
  {
    delay(100);
  }
  Serial.println();
```

```
  // Print CC3000 IP address. Enable if mDNS doesn't work
  while (! displayConnectionDetails()) {
    delay(1000);
  }

  // Start server
  restServer.begin();
  Serial.println(F("Listening for connections..."));

  // Enable watchdog
  wdt_enable(WDTO_4S);
}

void loop() {

  // Handle REST calls
  Adafruit_CC3000_ClientRef client = restServer.available();
  rest.handle(client);
  wdt_reset();

  // Check connection, reset if connection is lost
  if(!cc3000.checkConnected()){while(1){}}
  wdt_reset();

}

// Print connection details of the CC3000 chip
bool displayConnectionDetails(void)
{
  uint32_t ipAddress, netmask, gateway, dhcpserv, dnsserv;

  if(!cc3000.getIPAddress(&ipAddress, &netmask,
    &gateway, &dhcpserv, &dnsserv))
  {
    Serial.println(F("Unable to retrieve the IP Address!\r\n"));
    return false;
  }
  else
```

```
  {
    Serial.print(F("\nIP Addr: ")); cc3000.printIPdotsRev(ipAddress);
    Serial.print(F("\nNetmask: ")); cc3000.printIPdotsRev(netmask);
    Serial.print(F("\nGateway: ")); cc3000.printIPdotsRev(gateway);
    Serial.print(F("\nDHCPsrv: ")); cc3000.printIPdotsRev(dhcpserv);
    Serial.print(F("\nDNSserv: ")); cc3000.printIPdotsRev(dnsserv);
    Serial.println();
    return true;
  }
}

// Custom function accessible by the API
int ledControl(String command) {

  // Get state from command
  int state = command.toInt();

  digitalWrite(6,state);
  return 1;
}
```

In this sketch, you will need to modify some things so it can work. You need to change your WiFi network settings:

```
#define WLAN_SSID       "your_wifi_name"
#define WLAN_PASS       "your_wifi_password"
#define WLAN_SECURITY   WLAN_SEC_WPA2
```

We also introduced a new feature of aREST, functions. Those are functions that you define inside your sketch, and that you then declare to be exposed to the aREST API. You will then be able to call them from a web browser, for example.

First, you need to declare your function in the setup() function of the sketch:

```
rest.function("led",ledControl);
```

Then, implement the details of your function in the sketch. It needs to return an int, and take a String as the input parameters. The parameters in the String are then separated by commas.

For example, in this project I defined a simple function to switch the LED on or off according to the input parameter:

```
int ledControl(String command) {

  // Get state from command
  int state = command.toInt();

  digitalWrite(6,state);
  return 1;
}
```

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

https://github.com/marcoschwartz/discover-arest

Now, put all the code inside your Arduino IDE, and modify your WiFi name & password. After that, upload the code to the board, and open the Serial monitor.

After a while, you should see the IP address of the board inside the Serial monitor. Let's suppose for the rest of this project that it is "192.168.115.104". To check if your board is responding to aREST commands, go to a web browser and type:

```
http://192.168.115.104/id
```

You should immediately see the answer in JSON format:

```
{
  "id": "008",
  "name": "mighty_cat",
  "connected": true
}
```

Now, let's try a new function of the aREST API: the possibility to write analog values on outputs. On the Arduino Uno board, this will have the effect of sending PWM signal on the desired pin. First, set pin 6 as an output:

```
http://192.168.115.104/mode/6/o
```

Then, set the value of pin 6 using:

```
http://192.168.115.104/analog/6/10
```

You should get the confirmation in your web browser:

```
{
  "message": "Pin D6 set to 10",
  "id": "008",
  "name": "mighty_cat",
  "connected": true
}
```

This is basically the equivalent of an analogWrite(6, 10) function inside Arduino. You should see that the LED is now only lightly glowing. You can basically use values from 0 to 255, just as you would do directly in the code.

### 1.3.4 Using aREST Functions

Let's now go back to the function that we exposed to the API via:

```
rest.function("led",ledControl);
```

We will see that it is really easy to access this function remotely. Simply type:

```
http://192.168.115.104/led?params=1
```

You will immediately get the response in your browser:

```
{
  "return_value": 1,
  "id": "008",
  "name": "mighty_cat",
  "connected": true
}
```

As we passed '1' as a parameter, you should also see that the LED is now on. Note that you can pass several parameters to the function, and they need to be separated by commas. You can now define your own aREST functions, and call them remotely!

## 1.4 Use XBee Communications with Arduino & aREST

With the projects we saw so far, we used aREST over several way of communications, like Ethernet or WiFi. But aREST is not limited to HTTP-based communications. In this project, we are going to use XBee radios to communicate between your computer & an Arduino board running aREST. As an example, we'll see how to read data from a remote sensor, via XBee.

### 1.4.1 Hardware & Software Requirements

This project will be composed of two separate devices: the Arduino Uno board, connected to a DHT11 temperature and humidity sensor, along with an XBee radio. The other part will be a USB XBee explorer board, with an XBee radio, connected to your computer.

This is the list of required components for the Arduino & XBee device:

- Arduino Uno (https://www.sparkfun.com/products/11021)
- DHT11 sensor (https://www.adafruit.com/products/386)
- XBee Series 1 module (https://www.sparkfun.com/products/11215)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

This is the list of required components for the device you will connect to your computer:

- XBee explorer board USB (https://www.sparkfun.com/products/11373)
- XBee Series 1 module (https://www.sparkfun.com/products/11215)

You will also need to install the library "Adafruit DHT sensor" via the Arduino library manager.

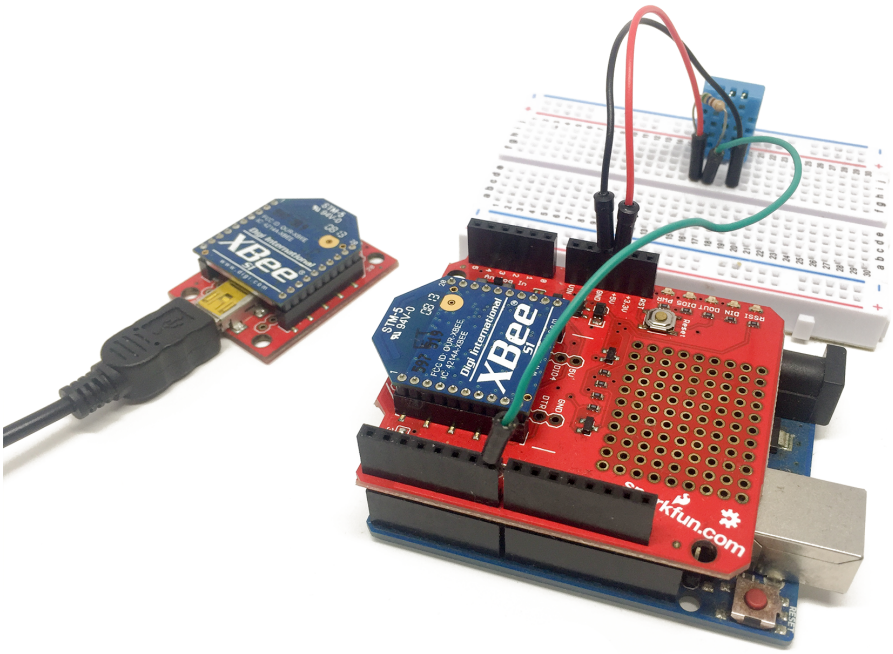
### 1.4.2 Hardware Configuration

Let's first configure the Arduino XBee module. First, place the XBee radio on top of the XBee shield, and then place the shield on the Arduino board.

Let's now connect the DHT11 sensor. The DHT sensor signal pin (pin number 2 of the sensor) has to be connected on pin 7 of the Arduino board. Also connect the VCC pin (pin number 1 of the sensor) of the sensor to the Arduino 5V pin, and GND to GND. Finally, insert the 4.7K Ohm resistor between pin number 1 and 2.

For the XBee explorer board, simply place the XBee radio on top of the USB board, and connect it to your computer via the USB port.

This is the completely assembled hardware:




### 1.4.3 Monitoring Your Board via XBee

Let's now see how to access the measurements made by the sensor via XBee. This is the complete sketch for this part:

```
// Libraries
#include <SPI.h>
#include <aREST.h>
#include "DHT.h"

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

// DHT
#define DHTPIN 7
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
```

```
// Variables
float temperature;
float humidity;

void setup(void)
{
  // Start Serial
  Serial.begin(9600);

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

  // Variables
  rest.variable("temperature", &temperature);
  rest.variable("humidity", &humidity);

  // Start DHT
  dht.begin();

}

void loop() {

  // Read data
  humidity = dht.readHumidity();
  temperature = dht.readTemperature();

  // Handle REST calls
  rest.handle(Serial);

}
```

As you can see, it is really similar to the sketch that we saw in the first project of this book. The only difference is the code for the DHT sensor, as well as the speed of the Serial port (9600).

Note that you can find all the code for this part inside the GitHub repository

of the book:

https://github.com/marcoschwartz/discover-arest

It's now time to program our board. Make sure to set the switch on 'DLINE' on the XBee shield, and upload the sketch. Then, set it again to 'UART'.

After that, select the Serial port corresponding to your XBee explorer board, go to the Serial monitor (make sure the speed is set to 9600), and type:

```
/id
```

The command will be sent to your Arduino board via XBee. You should immediately get the answer inside the Serial monitor:


Then, you can really easily get the temperature data with:

```
/temperature
```

You should immediately get the answer from the board inside the Serial monitor:


You can now control your projects via XBee as well, using the aREST commands!

## 1.5 Learn to use Bluetooth LE to control Arduino boards

In this section, we are going to learn how to control your Arduino projects using Bluetooth Low-Energy (LE) technology. This is really useful if you want to build projects that need to run on batteries.

### 1.5.1 Hardware & Software Requirements

This is the list of the components you will need for this project:

- Arduino Uno (https://www.sparkfun.com/products/11021)
- Adafruit Bluetooth LE breakout (https://www.adafruit.com/products/1697)
- LED (https://www.sparkfun.com/products/9590)
- 330 Ohm resistor (https://www.sparkfun.com/products/8377)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

You will also need the Adafruit nRF8001 library:

https://github.com/adafruit/Adafruit_nRF8001

You can of course also install this library using the Arduino library manager. Finally, if you are using an iOS device you will need to get the Adafruit Bluefruit LE Connect app:

https://itunes.apple.com/app/adafruit-bluefruit-le-connect/id830125974?mt=8

If you are using Android, you need to get the Nordic nRF app:

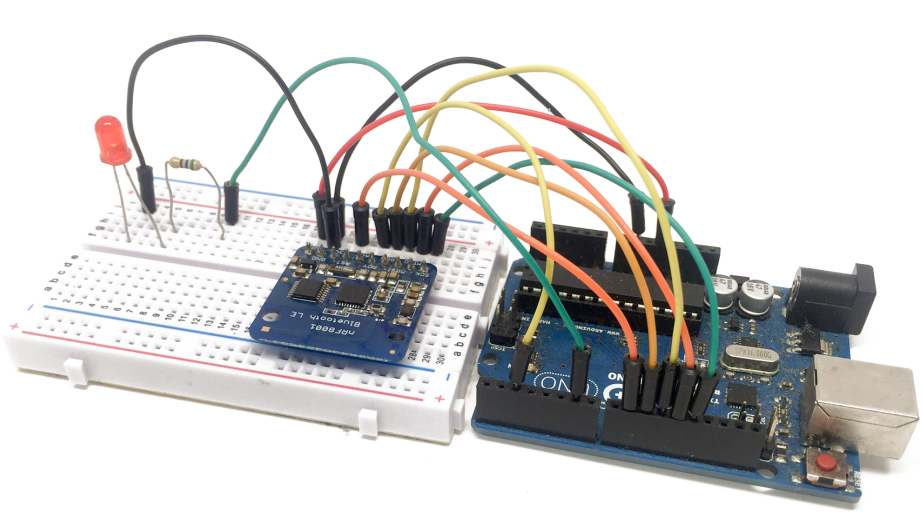
https://play.google.com/store/apps/details?id=com.nordicsemi.nrfUARTv2

### 1.5.2 Hardware Configuration

First, we are now going to connect the Bluetooth LE module. Connect the power supply of the module: GND goes to Arduino GND, and VIN goes to Arduino 5V pin. After that, you need to connect the different wires responsible for the SPI interface: SCK to Arduino pin 13, MISO to Arduino pin 12, and MOSI to Arduino pin 11. Then, connect the REQ pin to Arduino pin 10. Finally, connect the RDY pin to Arduino pin 2, and the RST pin to Arduino pin 9.

For the LED, simply connect it in series with the resistor, with the longest pin of the LED connected to the resistor. Then, connect the remaining pin of the resistor to pin 6 of the Arduino board, and the remaining pin of the LED to the GND pin of the Arduino board.

This is a picture of the final result for this section:



### 1.5.3 Controlling Your Arduino Board via Bluetooth

It's now time to configure our Arduino board so we can control it via Bluetooth. Here, the handling of aREST functions will be quite different from what we saw earlier.

This is the complete sketch for this part:

```
// Libraries
#include <SPI.h>
#include "Adafruit_BLE_UART.h"
#include <aREST.h>

// Pins
#define ADAFRUITBLE_REQ 10
#define ADAFRUITBLE_RDY 2 // This should be an interrupt pin
#define ADAFRUITBLE_RST 9

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

// BLE instance
```

```
Adafruit_BLE_UART BTLEserial = Adafruit_BLE_UART(ADAFRUITBLE_REQ,
  ADAFRUITBLE_RDY, ADAFRUITBLE_RST);

// Variables to be exposed to the API
int temperature;
int humidity;

void setup(void)
{
  // Start Serial
  Serial.begin(115200);
  Serial.println(F("Adafruit Bluefruit Low Energy nRF8001 Print echo demo"));

  BTLEserial.begin();

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

  // Give name and ID to device
  rest.set_id("008");
  rest.set_name("dapper_drake");

  pinMode(6,OUTPUT);

}

aci_evt_opcode_t laststatus = ACI_EVT_DISCONNECTED;

void loop() {

  // Tell the nRF8001 to do whatever it should be working on.
  BTLEserial.pollACI();

  // Ask what is our current status
  aci_evt_opcode_t status = BTLEserial.getState();
  // If the status changed....
  if (status != laststatus) {
    // print it out!
```

```
    if (status == ACI_EVT_DEVICE_STARTED) {
        Serial.println(F("* Advertising started"));
    }
    if (status == ACI_EVT_CONNECTED) {
        Serial.println(F("* Connected!"));
    }
    if (status == ACI_EVT_DISCONNECTED) {
        Serial.println(F("* Disconnected or advertising timed out"));
    }
    // OK set the last status change to this one
    laststatus = status;
  }

  // Handle REST calls
  if (status == ACI_EVT_CONNECTED) {
    rest.handle(BTLEserial);
  }
}
```

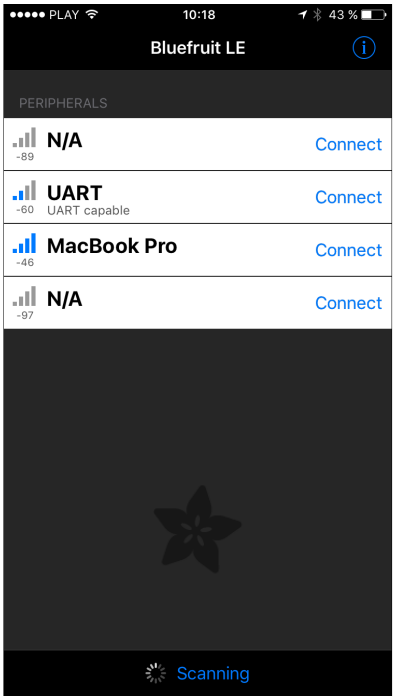
Note that you can find all the code for this part inside the GitHub repository of the book:

https://github.com/marcoschwartz/discover-arest

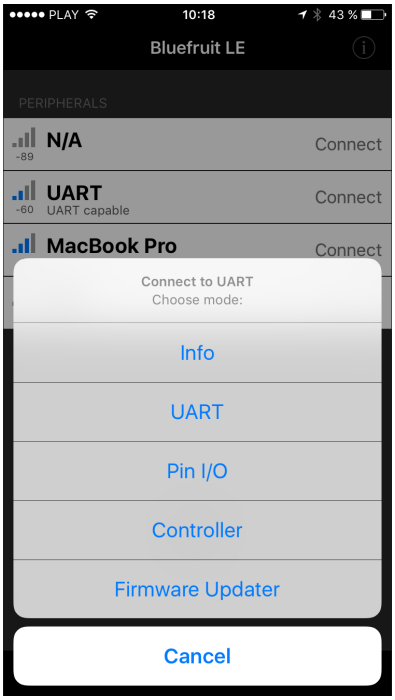
You can now grab the whole code, and also upload it to the board. Also open the Serial monitor. Then, you are going to need some way to communicate with the BLE breakout board. The best is to use the mobile application that you downloaded earlier.

I will use the iOS application as an example, but you can of course use the Android version as well.

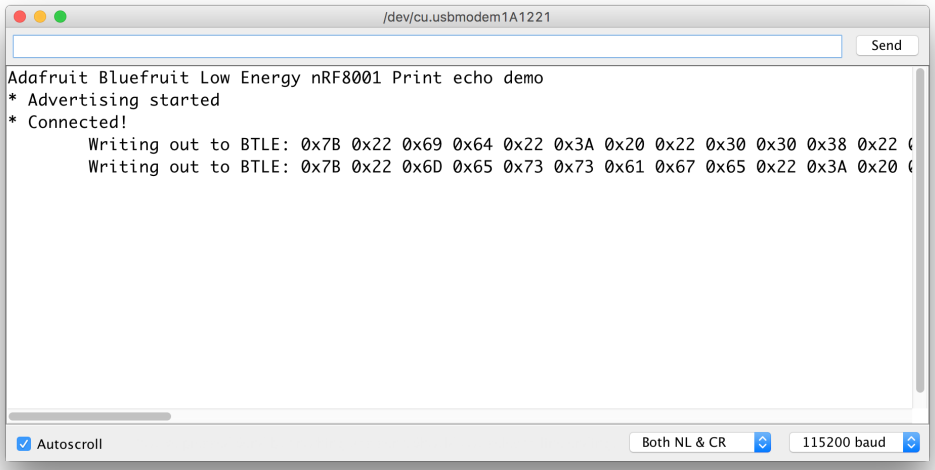
Open the app, and you should see a list of devices listed. Click on 'UART':




Then, click again on ‘UART’:



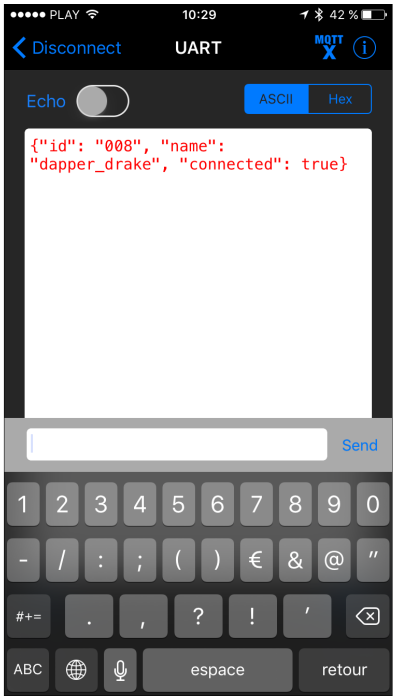
You should see inside the Serial monitor that a device is connected to the BLE module:




Now, send the following command to check if the board is responding:

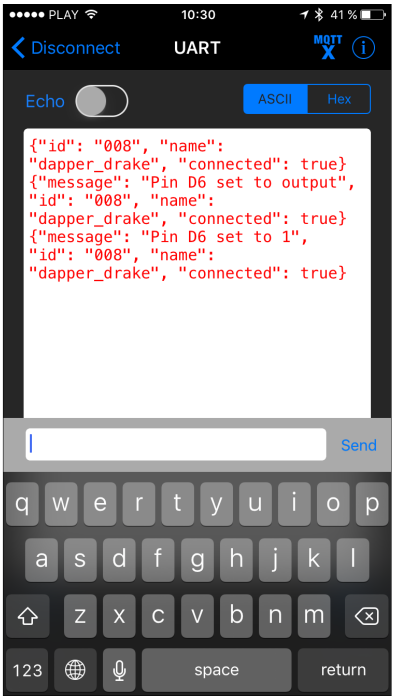
```
/id /
```

You should get the answer inside the application:




Note that here you need to send another '/' character at the end of the command, to indicate that it is the end of the message.

You can now try all the usual aREST commands, you can for example try to turn the LED on:




Congratulations, you can now control your Arduino projects via Bluetooth LE!

## 1.6 Control your Raspberry Pi board remotely with aREST

So far, we only saw how to control Arduino boards using the aREST framework. However, over the years aREST was modified and improved to support more boards from different manufacturers. In this section, we are going to see how to use aREST to control the very popular Raspberry Pi board.

### 1.6.1 Hardware & Software Requirements

Let's first see what you need for this project. I used a Raspberry Pi 2 board, but it will work on any Raspberry Pi that can be connected to your local network.

You will also need some way to connect your Pi to your local network. I used an USB WiFi dongle, but you can also use the Ethernet connection of the Pi.

This is the list of all the components that you will need for this project:

- Raspberry Pi (https://www.adafruit.com/products/2358)
- USB WiFi dongle (https://www.adafruit.com/products/814)
- LED (https://www.sparkfun.com/products/9590)
- 330 Ohm resistor (https://www.sparkfun.com/products/8377)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

You will need your Pi to be completely configured with Raspbian, and with Node.js installed on it. If that's not done yet, please go back to the Getting Started section of the Introduction chapter.

Then, log to your Pi either via an external screen or via SSH. Then, type:

```
ifconfig
```

This will give you the IP address of your Pi, you will need it in a moment.

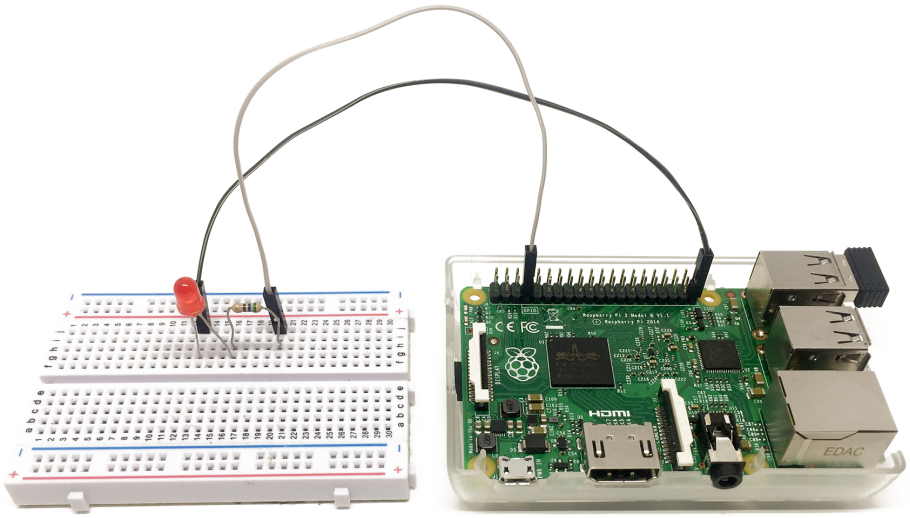
### 1.6.2 Hardware Configuration

It's now time to configure your Raspberry Pi. Simply place the LED in series with the resistor on the breadboard. Then, connect the other end of the resistor to pin 7 of the Raspberry Pi. You can find a list of GPIO pins of the Raspberry Pi 2 at the following link:

http://pi4j.com/images/j8header-2b-large.png

Then, connect the other end of the LED to one GND pin of the Pi.

This is the final result:



### 1.6.3 Controlling Your Raspberry Pi Remotely

We are now going to configure the Raspberry Pi so it can be controlled via aREST. Here, we of course can't use the Arduino aREST library to configure our board. This is why I created a dedicated library called pi-aREST for the Raspberry Pi, that is using Node.js.

This is the complete code for this part:

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

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

temperature = 24;
humidity = 40;
piREST.variable('temperature',temperature);
piREST.variable('humidity',humidity);
```

```
var server = app.listen(80, function() {
    console.log('Listening on port %d', server.address().port);
});
```

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

https://github.com/marcoschwartz/discover-arest

You can now take the file to a folder on your Raspberry Pi, go this folder via a terminal, and type:

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

This will install all the required components for pi-aREST. Then, start the software with:

```
sudo node pi_arest.js
```

The aREST API is now available on your Pi as well. Let's assume for the rest of this chapter that the IP address of your board is '192.168.115.106'. You can test the aREST API by simply going to your web browser, and type:

```
http://192.168.115.106/id
```

You should immediately receive the answer in JSON format:

```
{
  "id":"34f5eQ",
  "name":"my_new_Pi",
  "hardware":"rpi",
  "connected":true
}
```

Let's now try to light up the LED. Here, there is a difference with the Arduino aREST software: we don't need to set the pin as an output first, this is done by the Raspberry Pi software for you. Simply type:

```
http://192.168.115.106/digital/7/1
```

You should that the LED is now on, and you should also get the confirmation in the browser:

```
{
  "id":"34f5eQ",
  "name":"my_new_Pi",
  "hardware":"rpi",
  "connected":true,
  "message":"Pin 7 set to 1"
}
```

Then, we will also test the variable command of aREST, which also works with the Raspberry Pi. This is done by defining variables in the code with:

```
piREST.variable('temperature',temperature);
piREST.variable('humidity',humidity);
```

Now, type the following command in your browser:

```
http://192.168.115.106/temperature
```

You will immediately get the answer in your browser:

```
{
  "id":"34f5eQ",
  "name":"my_new_Pi",
  "hardware":"rpi",
  "connected":true,
  "temperature":24
}
```

Here, we just set values to those variables, but you could of course link them to measurements coming from a sensor.

## 1.7 Control an ESP8266 module remotely with aREST

In this section, we are going to use aREST on yet another type of hardware: the ESP8266 WiFi chip. This chip is a $5 component that embeds a processor along with a WiFi chip, making it the perfect board for cheap and powerful wireless projects. As an example, we are going to run aREST on the board and grab data from a sensor wirelessly.

### 1.7.1 Hardware & Software Requirements

For this project, the most important component will of course be the ESP8266 WiFi chip. For the rest of this book, I will use the Adafruit ESP8266 board, as it's really convenient to use. Of course, you can use any ESP8266 module that you wish, most of them are compatible with aREST.

This is the list of components for this section:

- Adafruit ESP8266 breakout board (https://www.adafruit.com/products/2471)
- DHT11 sensor (https://www.adafruit.com/products/386)
- FTDI USB converter (https://www.adafruit.com/products/284)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

To program the board, we are going to use the well-known Arduino IDE, which will be very convenient. You just need to install the board definition for the ESP8266. To do that, follow this procedure:

- Start the Arduino IDE and open the Preferences window.
- Enter the following URL into the *Additional Board Manager URLs* field:

```
http://arduino.esp8266.com/package_esp8266com_index.json
```

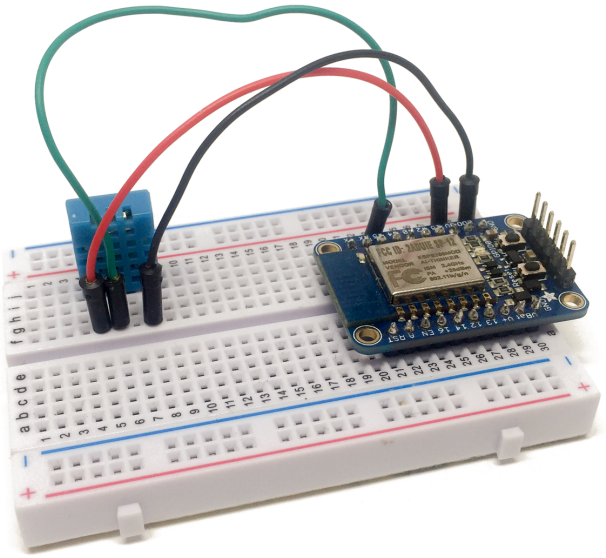
- Open Boards Manager from Tools > Board menu and install the esp8266 platform.

You will also need to install the Adafruit DHT library, that you can install from the Arduino library manager.

### 1.7.2 Hardware Configuration

Let's now assemble the hardware for this project. Simply place your ESP8266 module on the breadboard, and the DHT sensor next to it. Then, connect the first pin of the DHT11 sensor to the VCC pin, the second pin to pin number 5, and the last pin of the sensor to one GND pin of the ESP8266.

This is the final result:



### 1.7.3 Control ESP8266 Remotely

We are going to see that the sketch for the ESP8266 is really similar to the one for the Arduino boards. This is the complete sketch for this part:

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

// DHT11 sensor pins
#define DHTPIN 5
```

```
#define DHTTYPE DHT11

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

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

// WiFi parameters
const char* ssid = "wifi-name";
const char* password = "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 to be exposed to the API
float temperature;
float humidity;

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

  // Init DHT
  dht.begin();

  // Init variables and expose them to REST API
  rest.variable("temperature",&temperature);
  rest.variable("humidity",&humidity);

  // Give name and ID to device
  rest.set_id("1");
  rest.set_name("esp8266");
```

```
  // 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() {

  //rest.glow_led();

  // Reading temperature and humidity
  humidity = dht.readHumidity();
  temperature = dht.readTemperature();

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

}
```

Just as with Arduino with the CC3000 WiFi chip, you will need to set your WiFi name and password:

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

Compared to an Arduino sketch, only the part to handle requests is different:

```
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 part inside the GitHub repository of the book:

https://github.com/marcoschwartz/discover-arest

It's now time to upload the sketch. Make sure you opened the file in the Arduino IDE, and modify your WiFi name and password. Then, connect the FTDI USB converter to the board, put your board into bootloader mode. You can check the documentation of your board to know how to do this.

Now, upload the sketch to the board, and open the Serial monitor. Then, reset the board. You should see the IP address of the board displayed in the Serial monitor.

After that, go to your favorite web browser, and type:

```
http://192.168.115.102/temperature
```

You should immediately see the answer in JSON format:

```
{
  "temperature": 25.00,
  "id": "1",
  "name": "living_room",
  "connected": true
}
```

As you can see, it's really easy to use the same aREST commands that we used so far on the ESP8266 chip.

### 1.7.4 Access all Variables on Your Board

We are going to use the ESP8266 board to illustrate one more feature of the aREST framework: getting all variables on the board, using a single request.

For that, simply type the IP address of the board, without any command:

```
http://192.168.115.102/
```

You will get the following answer:

```
{
  "variables":
  {
    "temperature": 25.00,
    "humidity": 32.00
  },
  "id": "1",
  "name": "esp8266",
  "connected": true
}
```

As you can see, you can now access all the variables on the board, using a single command. Note that this is also available on other boards running the aREST framework!

## 1.8 Run a User Interface on an ESP8266 module

We are going to see one more basic feature of the aREST framework before moving on to the first example of this book. We are going to see how to run a graphical user interface (GUI) right on the ESP8266 chip itself, so you can control outputs and monitor inputs from a web page, without the need of any other components.

For that, we are going to use another library that is built on top of aREST, called aREST UI. This will allow us to easily create interface elements within the Arduino code.

### 1.8.1 Hardware & Software Requirements

For this project, the most important component will of course be the ESP8266 WiFi chip. For the rest of this book, I will use the Adafruit ESP8266 board, as it's really convenient to use. Of course, you can use any ESP8266 module that you wish, they are all compatible with aREST.

This is the list of components for this section:

- Adafruit ESP8266 breakout board (https://www.adafruit.com/products/2471)
- DHT11 sensor (https://www.adafruit.com/products/386)
- LED and 330 Ohm resistor
- FTDI USB converter (https://www.adafruit.com/products/284)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

At this point, you should already have the ESP8266 board definitions installed into the Arduino IDE, as well as the DHT11 library.

You will need to install one more library: aREST UI. You can do so by looking for 'aREST_UI' inside the Arduino library manager.

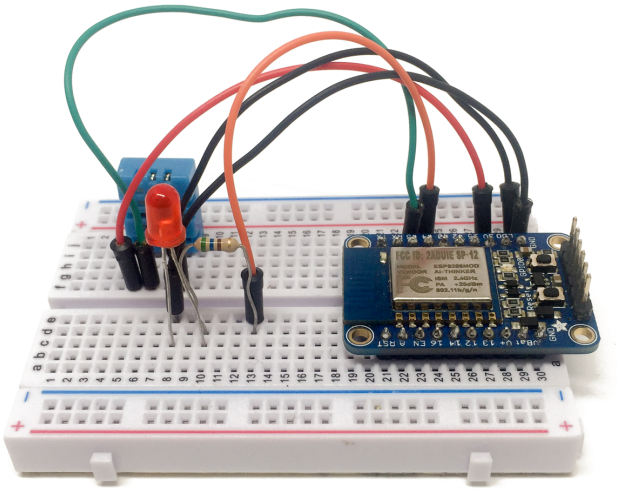
### 1.8.2 Hardware Configuration

Let's now assemble the hardware for this project. Simply place your ESP8266 module on the breadboard, and the DHT sensor next to it. Then, connect the

first pin of the DHT11 sensor to the VCC pin, the second pin to pin number 5, and the last pin of the sensor to one GND pin of the ESP8266.

For the LED, simply place it on the breadboard in series with the resistor, the longest pin in contact with the resistor. Then, connect the other end of the resistor to pin number 4 of the ESP8266, and the other end of the LED to one GND pin.

This is the final result:



### 1.8.3 Controlling Your ESP8266 with a Graphical Interface

Let's now configure the ESP8266 so we can run the aREST UI interface on it. This is the complete sketch for this part:

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

// DHT11 sensor pins
#define DHTPIN 5
#define DHTTYPE DHT11
```

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

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

// WiFi parameters
const char* ssid = "wifi-name";
const char* password = "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 to be exposed to the API
float temperature;
float humidity;

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

  // Init DHT
  dht.begin();

  // Init variables and expose them to REST API
  rest.variable("temperature",&temperature);
  rest.variable("humidity",&humidity);

  // Give name and ID to device
  rest.set_id("1");
  rest.set_name("esp8266_ui");

  // 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());

  // Create UI
  rest.title("ESP8266 UI");
  rest.button(4);
  rest.label("temperature");
  rest.label("humidity");

}

void loop() {

  // Reading temperature and humidity
  humidity = dht.readHumidity();
  temperature = dht.readTemperature();

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

```
}
```

Let's now see the differences between this new sketch and other sketches using aREST. You need to include the aREST UI library:

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

Then, instead of creating an aREST instance, we create an instance of aREST_UI:

```
aREST_UI rest = aREST_UI();
```

After that, in the setup() function, we define the interface elements of our sketch. We set a title, a button to control the LED, and two labels for the sensor measurements:

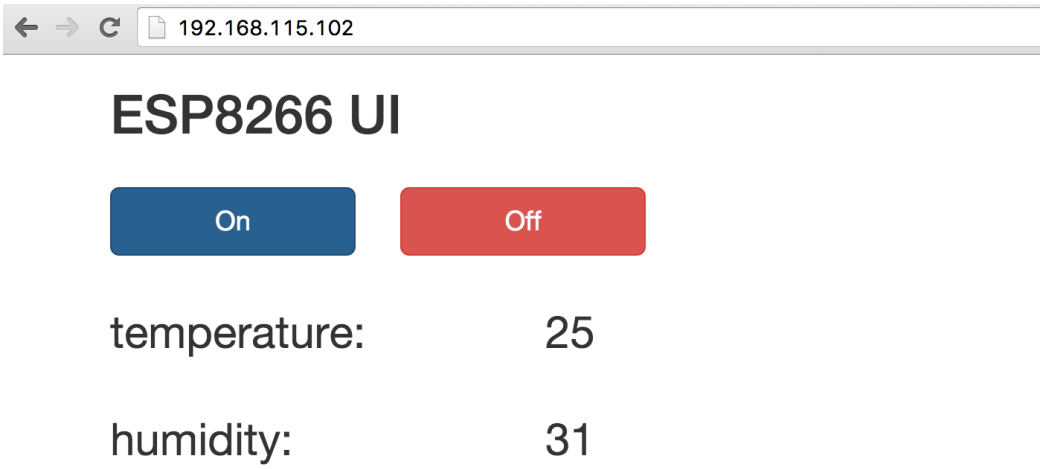
```
rest.title("ESP8266 UI");
rest.button(4);
rest.label("temperature");
rest.label("humidity");
```

Now, grab the code again from the GitHub repository of the book, and upload it to the board, following the instructions we saw earlier.

Then, open the Serial monitor to grab the IP address of the board. Then, you can simply type this IP address in a web browser, to access the interface, like:

```
http://192.168.115.102/
```

You should immediately see the interface inside your web browser:




You can try it now: for example, just click on the 'ON' button. The LED should instantly turn on. You can now create your own interfaces that are running on your ESP8266 board, without the help of any external component or service!

## 1.9 Example project: Control a Lamp Remotely using the ESP8266

Before ending this chapter about the basics of the aREST framework, we are going to see a real-life example on how to use aREST. We are going to build a simple lamp controller, that will allow the user to switch a lamp on and off via WiFi.

### 1.9.1 Hardware & Software Requirements

For this project, the most important component will of course be the ESP8266 WiFi chip. For the rest of this book, I will use the Adafruit ESP8266 board, as it's really convenient to use. Of course, you can use any ESP8266 module that you wish, they are all compatible with aREST.

This is the list of components for this section:

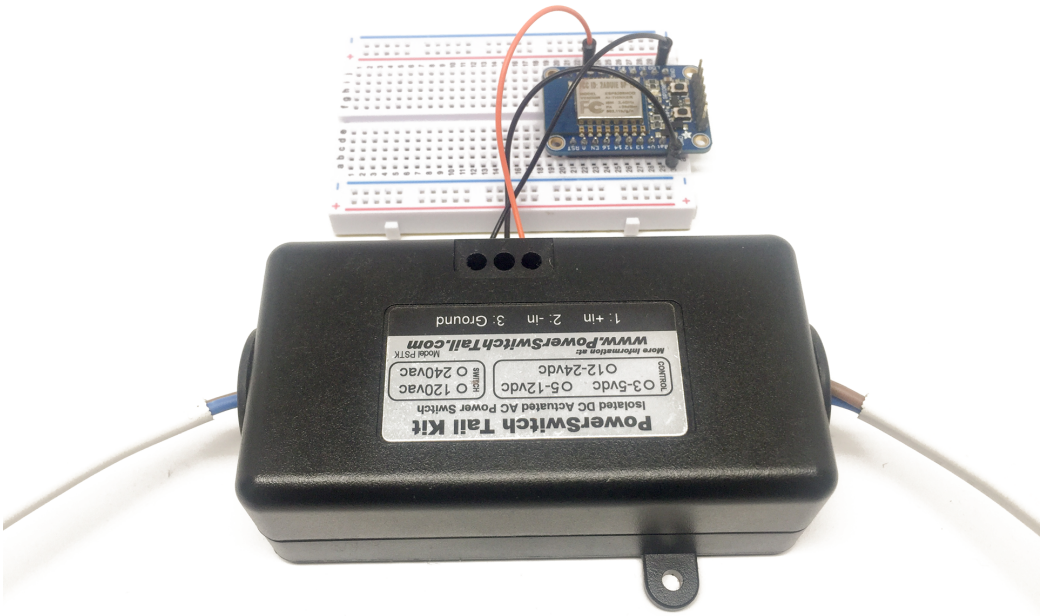
- Adafruit ESP8266 breakout board (https://www.adafruit.com/products/2471)
- Powerswitch Tail Kit (https://www.adafruit.com/products/268)
- FTDI USB converter (https://www.adafruit.com/products/284)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

At this point, you should already have the aREST UI library installed, as well as the ESP8266 board definitions. If that's not done yet, have a look at the previous chapters to know how to do this.

### 1.9.2 Hardware Configuration

The configuration for this project is very simple: first, place the ESP8266 module on the breadboard. Then, connect the Vin+ pin of the PowerSwitch Tail to the pin number 5 of the ESP8266. Finally, connect the two remaining pins of the PowerSwitch Tail to the GND pins of the ESP8266.

This is the result at this point:



Now, you can connect an electrical appliance to the PowerSwitch Tail Kit, for example a lamp. I used a 30W lamp for this project. Also connect the other part of the PowerSwitch Tail to the mains electricity.

### 1.9.3 Controlling a Lamp Remotely with aREST

We are now going to see how to configure the ESP8266 board so you can control the lamp via WiFi. We are again going to use the aREST UI library to create the interface. This is the complete sketch for this part:

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

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

// WiFi parameters
const char* ssid = "wifi-name";
const char* password = "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());

  // Create UI
  rest.title("Lamp control");
  rest.button(5);

}

void loop() {

  // Handle REST calls
  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 part inside the GitHub repository of the book:

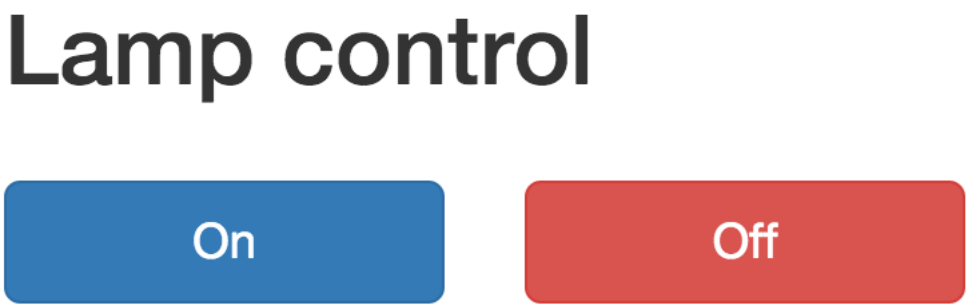
https://github.com/marcoschwartz/discover-arest

Now, grab the complete code, modify your WiFi network parameters, and upload the code to the board using the instructions we saw earlier. Also open the Serial monitor to get the IP address of the board. You might need to reset the board at this point.

Then, go to your web browser, and type the IP address of the board:

```
http://192.168.115.102/
```

You should see the graphical interface in your browser:




You can now try the interface: just by pressing a button, you can switch the lamp on and off. Congratulations, you can now built home automation projects using the aREST framework!

## 1.10 How to Go Further

In this first chapter of the book, we learned about the basics of the aREST framework. We learned about those basics by using a wide variety of hardware and communication mediums, like Ethernet or WiFi communications. We also saw that even those basic knowledge can allow you to create complex projects, like graphical interfaces hosted on the boards themselves.

In the next chapter, we are going to see how to go further, and create web-based applications that will allow you to easily control your boards running aREST.

# 2 Build local applications to control your projects

In this chapter, you will learn how to build applications to control your aREST boards from your computer. We will start by controlling projects from a single web page, and then we will move to building more complex applications based on the Meteor framework.

This will be the chapter where you will learn to control several boards running aREST from the same application, and do more complex projects like recording data measured by the aREST boards, and plotting this data in live inside your web browser.

## 2.1 Control an Arduino Board from a Web Page via Ethernet

In this first project of the chapter, we are going to use a library for JavaScript called aREST.js to control aREST devices from a web page. This is the perfect library if you want to easily control aREST devices from a graphical interface, in the case where you don't need to record and store data for example.

As an example, we are going to use an Arduino Ethernet board running aREST, that we will completely control from a simple HTML page using aREST.js.

### 2.1.1 Hardware & Software Requirements

Let's first see what we need for this project. You will need a board that is compatible with the Arduino Ethernet shied, like an Arduino Uno that I will use for this section.

You will also need a the Arduino Ethernet shield or a similar, one LED, one 330 Ohm resistor, a breadboard, and some jumper wires.

This is the list of components that were used in this project:

- Arduino Uno (https://www.sparkfun.com/products/11021)

- Arduino Ethernet shield (https://www.sparkfun.com/products/11166)
- LED (https://www.sparkfun.com/products/9590)
- 330 Ohm resistor (https://www.sparkfun.com/products/8377)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

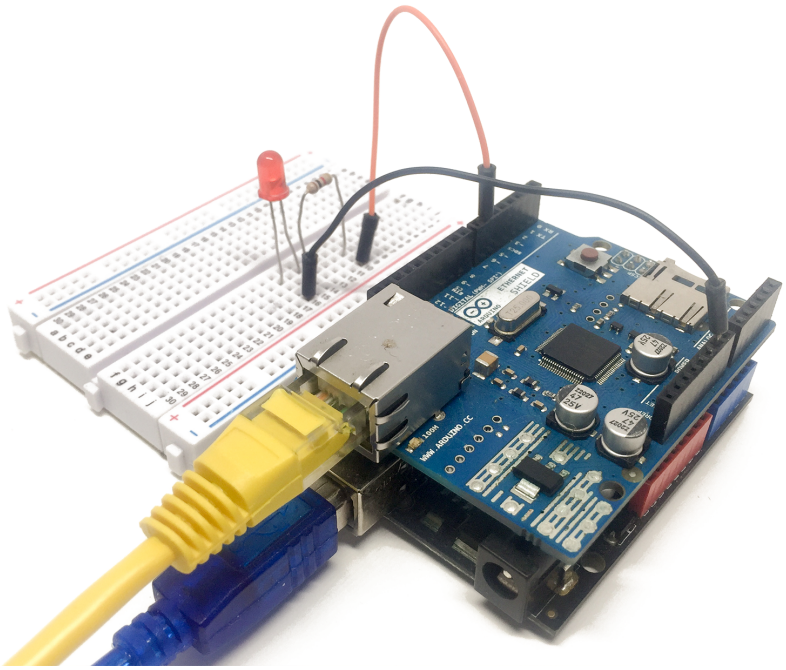
### 2.1.2 Hardware Configuration

First, place the Ethernet shield on top of the Arduino board, and plug an Ethernet cable between the board and your router.

For the LED, simply connect it in series with the resistor, with the longest pin of the LED connected to the resistor.

Then, connect the remaining pin of the resistor to pin 6 of the Arduino board, and the remaining pin of the LED to the GND pin of the Arduino board.

This is the final result:



### 2.1.3 Control Your Ethernet Board from a Web Page

Let's now see how to control the board from a web page. Here is the complete sketch for this part:

```
// Libraries
#include <SPI.h>
#include <Ethernet.h>
#include <aREST.h>

// Enter a MAC address for your controller below.
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0E, 0xFE, 0x40 };

// IP address in case DHCP fails
IPAddress ip(192,168,2,2);

// Ethernet server
EthernetServer server(80);

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

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

  // Give name and ID to device
  rest.set_id("008");
  rest.set_name("dapper_drake");

  // Start the Ethernet connection and the server
  if (Ethernet.begin(mac) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    // no point in carrying on, so do nothing forevermore:
    // try to congifure using IP address instead of DHCP:
    Ethernet.begin(mac, ip);
  }
```

```
  server.begin();
  Serial.print("server is at ");
  Serial.println(Ethernet.localIP());

}

void loop() {

  // listen for incoming clients
  EthernetClient client = server.available();
  rest.handle(client);

}
```

This is something we already saw in the previous chapter, so I won't go into details here. Just make sure to modify the MAC address in the sketch, and upload it to your Arduino board. Then, open the Serial monitor to get the IP address of the board.

We are now going to see the to build the interface for this project. It will be composed of two parts: an HTML page, and a JavaScript file.

Let's first see the content of the HTML page. In the header tag, you need to import jQuery, AjaxQ, and of course the aREST.js library:

```
<script type="text/javascript"
  src="https://code.jquery.com/jquery-2.1.4.min.js">
</script>
<script type="text/javascript"
  src="https://cdn.rawgit.com/Foliotek/AjaxQ/master/ajaxq.js">
</script>
<script type="text/javascript"
  src="https://cdn.rawgit.com/marcoschwartz/aREST.js/master/aREST.js">
  </script>
<script type="text/javascript" src="script.js"></script>
```

You also need to import the script.js file, that will contain our JavaScript code.

Then, we define a simple interface where we have two buttons to control the LED on the board:

```
<div class='row'>

  <div class="col-md-2">Digital pin 6: </div>

  <div class="col-md-2">
    <button id='on' class='btn btn-primary btn-block' type="button">
      On
    </button>
  </div>

  <div class="col-md-2">
    <button id='off' class='btn btn-danger btn-block' type="button">
      Off
    </button>
  </div>

</div>
```

I also added several other elements to illustrate all the functions of aREST. This is an element to use PWM commands on pin 6:

```
<div class='row'>

  <div id='label' class="col-md-2">PWM pin 6: </div>
  <div class="col-md-4">
    <input id='slider' type="range" value="0" max="255" min="0" step="5">
  </div>
</div>
```

Finally, we also create an element that will contain the readings of the analog pin A0 on the board:

```
<div class='row'>
```

```
  <div class="col-md-3">Analog pin A0:
    <span id='A0'></span>
  </div>

</div>
```

Let's now see the script.js file. The first step is to create a new aREST device, with the IP address of the board:

```
var device = new Device("192.168.115.105");
```

We also set pin 6 as an output:

```
device.pinMode(6, "OUTPUT");
```

Next, we define the controls on the buttons:

```
// Button
$('#on').click(function() {
  device.digitalWrite(6, 1);
});

$('#off').click(function() {
  device.digitalWrite(6, 0);
});
```

As you can see, each button calls a digitalWrite() function, which is really similar to the function that you can use on an Arduino sketch. The first command will simply call a '/digital/6/1' function on the aREST API, which in return will call a digitalWrite(6, 1) function on the board.

For the PWM control of the same pin, I created a slider:

```
$('#slider').mouseup(function() {
  var val = $('#slider').val();
  device.analogWrite(6, val);
});
```

This will call the analogWrite() function on the board every time the slider is modified.

Finally, we create a loop to read data from the analog pin A0, every 5 seconds:

```
device.analogRead(0, function(data) {
  $("#A0").html(data.return_value);
});
setInterval(function() {
  device.analogRead(0, function(data) {
    $("#A0").html(data.return_value);
  });
}, 5000);
```

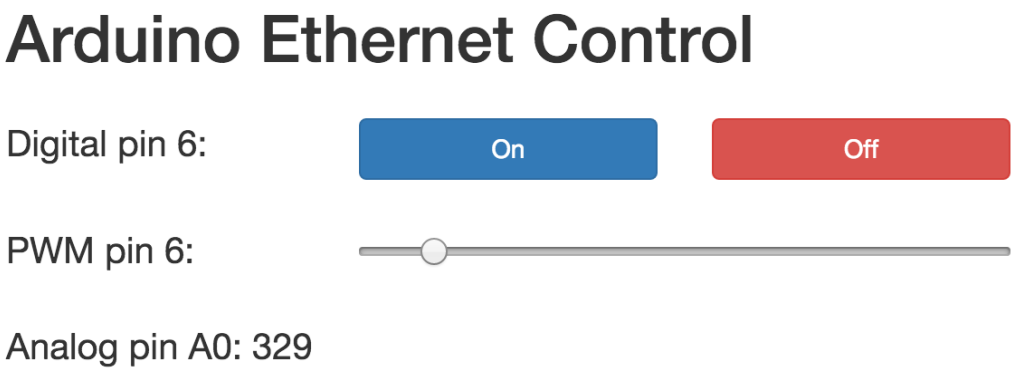
As you can see, we use the analogRead() function, which does exactly the same thing as its Arduino counterpart.

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

https://github.com/marcoschwartz/discover-arest

It's now time to test our project! Make sure to grab the code from the GitHub repository, and modify the script with the IP address of your board.

Then, simply open the HTML file with any web browser. You should immediately see the interface in your browser:




You can now try the different controls, and the LED should react accordingly. Especially use the slider to modify the luminosity of the LED.

We can now build simple interfaces for your aREST projects. Later in this book, we are going to see how to build more complex applications to also record data locally.

## 2.2 Control & Monitor an ESP8266 module from a Web page

In this section, we are going to continue exploring the aREST.js library, this time by using it to read data from a sensor connected to the ESP8266 module.

### 2.2.1 Hardware & Software Requirements

For this example project, you will need components that we already used in the previous chapter, like an ESP8266 module, a DHT sensor, and breadboard & jumper wires.

This is the list of the required components for this project:

- ESP8266 Huzzah module (https://www.adafruit.com/products/2471)
- FTDI USB converter (https://www.adafruit.com/products/70)
- DHT11 sensor (https://www.adafruit.com/products/386)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

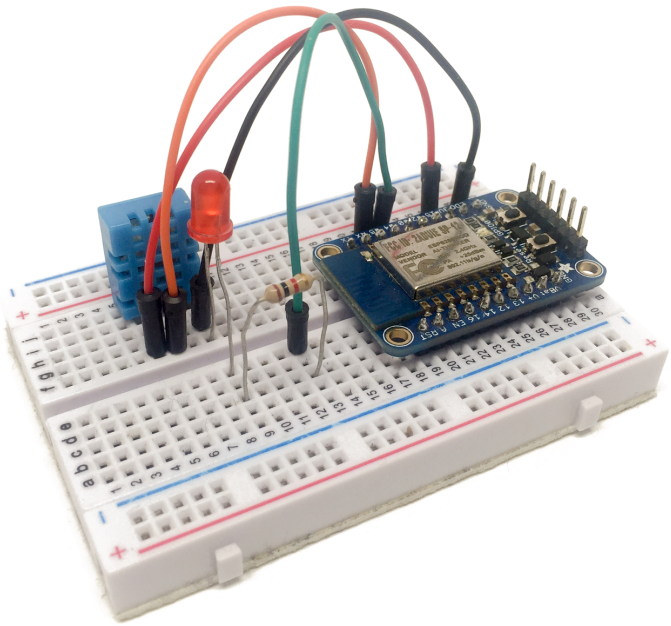
You will also need to have the board definitions installed for the ESP8266, as well as the DHT library from Adafruit. It should already be done at this point, but if that's not the case refer to the previous chapter before moving on.

### 2.2.2 Hardware Configuration

Let's now see how to configure this project. First, place the ESP8266 module on the breadboard, and also place the DHT11 sensor on it.

The DHT sensor signal pin (pin number 2 of the sensor) has to be connected on pin 5 of the ESP8266. Also connect the VCC pin (pin number 1 of the sensor) of the sensor to the ESP8266 3.3V pin, and GND to GND.

This is the final result:



### 2.2.3 Control Your ESP8266 from a Web Page

Let's now see how to control the ESP8266 board from a web page, and grab data from the sensor. This is the complete sketch for this part:

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

// DHT11 sensor pins
#define DHTPIN 5
#define DHTTYPE DHT11

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

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

// WiFi parameters
const char* ssid = "wifi-name";
const char* password = "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 to be exposed to the API
float temperature;
float humidity;

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

  // Init DHT
  dht.begin();

  // Init variables and expose them to REST API
  rest.variable("temperature",&temperature);
  rest.variable("humidity",&humidity);

  // Give name and ID to device
  rest.set_id("1");
  rest.set_name("esp8266");

  // 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() {

  // Reading temperature and humidity
  humidity = dht.readHumidity();
  temperature = dht.readTemperature();

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

}
```

As you can see, it's a typical sketch for the ESP8266 that we already saw earlier. Just make sure to modify the WiFi network name and password, and upload the sketch to the board. Then, open the Serial monitor to grab the IP address of the board.

Let's now see the details of the interface. As in the previous section, it will be composed of an HTML page and a JavaScript file. This is the part in the

HTML page to display the temperature & humidity:

```
<div class='row'>

  <div class="col-md-2">Temperature:
    <span id='temperature'></span>
  </div>

  <div class="col-md-2">Humidity:
    <span id='humidity'></span>
  </div>
</div>
```

To update those fields, we need to define some code inside the JavaScript file:

```
// Temperature display
device.getVariable('temperature', function(data) {
  $('#temperature').html(data.temperature);
});

// Humidity display
device.getVariable('humidity', function(data) {
  $('#humidity').html(data.humidity);
});
```

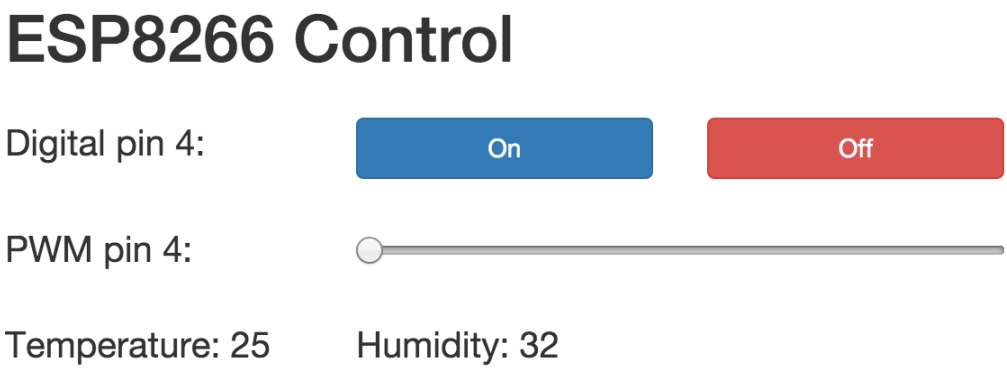
Those commands are basically the equivalent of calling a variable exposed by the aREST API. Note that we need to use callbacks here, as we can only update the value of the indicator in the interface when we have the answer from the board.

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

https://github.com/marcoschwartz/discover-arest

It's now time to test our project! Make sure to grab the code from the GitHub repository, and modify the JavaScript file with the IP address of your board.

Then, simply open the HTML file with any web browser. You should immediately see the interface in your browser:




As you can see, the temperature & humidity data is immediately displayed inside the interface. Here, this is just the instant values from the board. We are going to learn how to store measurements data on your computer later in this chapter.

## 2.3 Learn to Control & Monitor Arduino boards via Ethernet

So far in this chapter, we saw how to control aREST devices using a simple web page and some JavaScript code. This is great for simple applications, however it is not suited for more complex projects. For example, using a simple web page you can't really plan automated actions, connect with external APIs, or store data locally.

This is why we need to use a real web server to connect to aREST devices for more complex applications. To help you out in this task, I developed a module for the very popular Meteor framework that makes it really easy. You can learn more about Meteor at http://meteor.com/.

To illustrate the basics of this Meteor aREST module, we will use again an Arduino Uno board that we will control via Ethernet.

### 2.3.1 Hardware & Software Requirements

You will need an Arduino board, like an Arduino Uno that was used for this project. You will also need an Ethernet shield for Arduino. You will also need one LED, one 220 Ohm resistor, a breadboard, and some jumper wires.

This is the list of the required components for this part:

- Arduino Uno (https://www.sparkfun.com/products/11021)
- Arduino Ethernet shield (https://www.sparkfun.com/products/11166)
- DHT11 sensor + 4.7k resistor (https://www.adafruit.com/products/386)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

You will need to have the Meteor framework installed on your computer. If that's not done yet, follow the instructions from the Getting Started section in the Introduction.

### 2.3.2 Hardware Configuration

The hardware configuration for this project is quite simple, as this is something we already saw earlier in the book. First, place the shield on top of the Arduino board.

The DHT sensor signal pin (pin number 2 of the sensor) has to be connected on pin 7 of the Arduino board. Also connect the VCC pin (pin number 1 of the sensor) of the sensor to the Arduino 5V pin, and GND to GND. Finally, insert the 4.7K Ohm resistor between pin number 1 and 2.

### 2.3.3 Configure Your Arduino Ethernet Board

We first need to configure the Arduino board so we can access it via Ethernet. This is the complete sketch for this part:

```
// Libraries
#include <SPI.h>
#include <Ethernet.h>
#include <aREST.h>
#include "DHT.h"

// DHT 11 sensor
#define DHTPIN 7
```

```
#define DHTTYPE DHT11

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

// Enter a MAC address for your controller below.
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0E, 0xFE, 0x40 };

// IP address in case DHCP fails
IPAddress ip(192,168,2,2);

// Ethernet server
EthernetServer server(80);

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

// Variables to be exposed to the API
int temperature;
int humidity;

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

  // Init variables and expose them to REST API
  rest.variable("temperature",&temperature);
  rest.variable("humidity",&humidity);

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

  // Start the Ethernet connection and the server
  if (Ethernet.begin(mac) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    // no point in carrying on, so do nothing forevermore:
```

```
    // try to congifure using IP address instead of DHCP:
    Ethernet.begin(mac, ip);
  }
  server.begin();
  Serial.print("server is at ");
  Serial.println(Ethernet.localIP());

}

void loop() {

  // Make measurements
  humidity = (int)dht.readHumidity();
  temperature = (int)dht.readTemperature();

  // listen for incoming clients
  EthernetClient client = server.available();
  rest.handle(client);

}
```

This is a sketch we already saw several times in this book, so I won't detail it. Just make sure to modify the MAC address of the board, and upload the code to the board. Then, open the Serial monitor to grab the IP address of the board.

Finally, go to a web browser, and check that the board is online with:

```
http://192.168.115.104/id
```

You should get the answer in the browser:

```
{
  "id": "1",
  "name": "arduino",
  "connected": true
}
```

### 2.3.4 Build a Web Application to Control Your Boards

Now that the board is configured, we are going to dive into the core of this section: building an application based on Meteor.

Compared to other frameworks like Node.js that separate the server and the client parts, Meteor allow you to code everything into a single framework.

We'll first define an HTML file containing the interface:

```
<template name='home'>

  <body>

    <div class='container'>

      <h3>Arduino Monitoring</h3>

      <div class='row'>
        <div class='col-md-3'>
          Temperature: <span id='temperature'></span> C
        </div>
        <div class='col-md-3'>
          Humidity: <span id='humidity'></span>%
        </div>
      </div>

    </div>

  </body>

</template>
```

This file is really basic, as it only contains two containers for the temperature and humidity.

Then, inside a JavaScript file, we will define both the code for the server and the client:

```
if (Meteor.isServer) {
  Meteor.startup(function () {

    // Add device
    aREST.addDevice('http', '192.168.115.104');

  });
}

if (Meteor.isClient) {

  // Main route
  Router.route('/', {name: 'home'});

  // Rendered
  Template.home.rendered = function() {

    // Refresh temperature & humidity
    Meteor.call('getVariable', 1, 'temperature', function(err, data) {
      console.log(data);
      $('#temperature').text(data.temperature);
    });

    Meteor.call('getVariable', 1, 'humidity', function(err, data) {
      $('#humidity').text(data.humidity);
    });

  }

}
```

Let's see the detail of this code. On the server side, we need to call the aREST module to add the board to the application database:

```
aREST.addDevice('http', '192.168.115.104');
```

Don't worry about the database itself: Meteor is taking care about it for you. Then, on the client side, we define a route for our application:

```
Router.route('/', {name: 'home'});
```

This will simply load the 'home' template whenever we access the root of the server. Then, still on the client side, we call the 'getVariable' method, which is also included into the Meteor aREST module.

To call this function, we simply need to send the ID of the device we set in the code, and the variable we want to get:

```
Meteor.call('getVariable', 1, 'temperature', function(err, data) {
  console.log(data);
  $('#temperature').text(data.temperature);
});

Meteor.call('getVariable', 1, 'humidity', function(err, data) {
  $('#humidity').text(data.humidity);
});
```

In the same time, we update the indicators in the interface inside the callbacks of those methods.

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

https://github.com/marcoschwartz/discover-arest

It's now time to test the project! Grab all the code from the GitHub repository, and put all the files in a folder on your computer. Then, go to this folder with a terminal, and type:

```
meteor create .
```

This will initialize a new Meteor project inside this folder. Then, install the aREST Meteor module with:

```
meteor add marcoschwartz:arest
```

Finally, start the project with:

```
meteor
```

Now, navigate to the app with a web browser at:

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

You should immediately see the interface in your web browser, with the measurements made by the board.

At this point, you can wonder why we are doing all this, as we could do the same without Meteor. However, the difference is that we now have a much more solid infrastructure for our project, from which we could easily record data for example.

## 2.4 Build a Web-based Application to Control Arduino boards via WiFi

In the previous section, we saw the basics of the aREST module for Meteor. In this one, we are going to learn more about the aREST module for Meteor. We are going to build a Meteor application again, this time to control a board connected to your local network via WiFi.

### 2.4.1 Hardware & Software Requirements

For this project, we are going to use components we already used earlier: an Arduino Uno board, and the Adafruit CC3000 breakout board. Here, we'll also add a 5V relay to the project, so we can control an output from the Meteor application.

This is the list of the required components for this project:

- Arduino Uno (https://www.sparkfun.com/products/11021)
- Adafruit CC3000 breakout board (https://www.adafruit.com/products/1469)
- DHT11 (https://www.adafruit.com/products/386)
- Relay (https://www.pololu.com/product/2480)

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

Also make sure that the Adafruit CC3000 library is installed. If not, you can grab it using the Arduino IDE library manager.

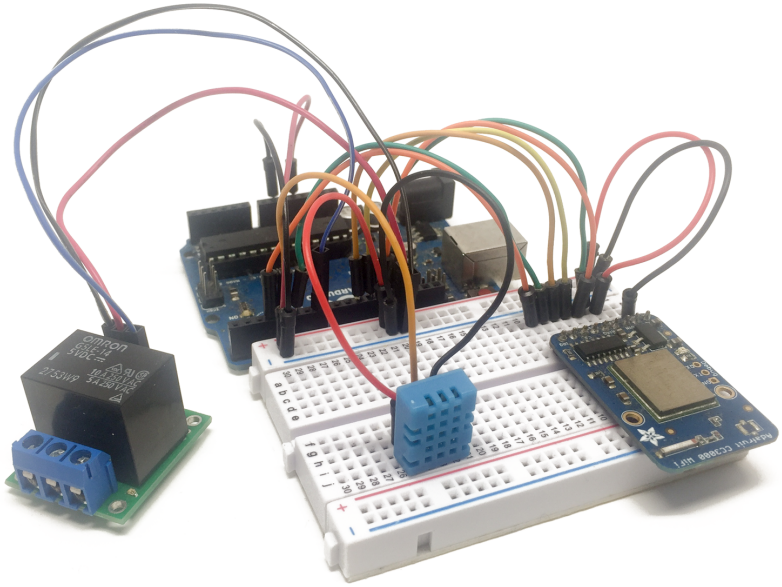
### 2.4.2 Hardware Configuration

First, place the CC3000 board on the breadboard. Then, connect the IRQ pin of the CC3000 board to pin number 3 of the Arduino board, VBAT to pin 5, and CS to pin 10. Then, you need to connect the SPI pins to the Arduino board: MOSI, MISO, and CLK go to pins 11,12, and 13, respectively. Finally, take care of the power supply: Vin goes to the Arduino 5V, and GND to GND.

The DHT sensor signal pin (pin number 2 of the sensor) has to be connected on pin 4 of the Arduino board. Also connect the VCC pin (pin number 1 of the sensor) of the sensor to the Arduino 5V pin, and GND to GND. Finally, insert the 4.7K Ohm resistor between pin number 1 and 2.

For the relay, connect the IN (or SIG) pin to pin number 7 of the Arduino board. Then, connect VCC to the 5V pin of the Arduino board, and GND to the GND pin of the Arduino.

This is the final result:



### 2.4.3 Configure Your WiFi Board

Let's now configure the Arduino board. This is a sketch we are already familiar with at this point:

```
// Import required libraries
#include <Adafruit_CC3000.h>
#include <SPI.h>
#include <aREST.h>
#include <avr/wdt.h>
#include "DHT.h"

// DHT11 sensor pins
#define DHTPIN 4
#define DHTTYPE DHT11

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

```
// These are the pins for the CC3000 chip
#define ADAFRUIT_CC3000_IRQ   3
#define ADAFRUIT_CC3000_VBAT  5
#define ADAFRUIT_CC3000_CS    10

// Create CC3000 instance
Adafruit_CC3000 cc3000 = Adafruit_CC3000(ADAFRUIT_CC3000_CS,
  ADAFRUIT_CC3000_IRQ, ADAFRUIT_CC3000_VBAT,SPI_CLOCK_DIV2);

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

// Your WiFi SSID and password
#define WLAN_SSID       "wifi-name"
#define WLAN_PASS       "wifi-password"
#define WLAN_SECURITY   WLAN_SEC_WPA2

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

// Server instance
Adafruit_CC3000_Server restServer(LISTEN_PORT);

// Variables
int temperature;
int humidity;

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

  // Init DHT
  dht.begin();

  // Give name and ID to device
  rest.set_id("1");
  rest.set_name("cc3000");
```

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

  // Set up CC3000 and get connected to the wireless network.
  Serial.print(F("CC3000 init ... "));
  if (!cc3000.begin())
  {
    while(1);
  }

  Serial.println(F(" done"));

  if (!cc3000.connectToAP(WLAN_SSID, WLAN_PASS, WLAN_SECURITY)) {
    while(1);
  }
  while (!cc3000.checkDHCP())
  {
    delay(100);
  }
  Serial.println();

  // Print CC3000 IP address. Enable if mDNS doesn't work
  while (! displayConnectionDetails()) {
    delay(1000);
  }

  // Start server
  restServer.begin();
  Serial.println(F("Listening for connections..."));

  // Enable watchdog
  wdt_enable(WDTO_4S);
}

void loop() {
```

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

  // Handle REST calls
  Adafruit_CC3000_ClientRef client = restServer.available();
  rest.handle(client);
  wdt_reset();

  // Check connection, reset if connection is lost
  if(!cc3000.checkConnected()){while(1){}}
  wdt_reset();

}

// Print connection details of the CC3000 chip
bool displayConnectionDetails(void)
{
  uint32_t ipAddress, netmask, gateway, dhcpserv, dnsserv;

  if(!cc3000.getIPAddress(&ipAddress, &netmask,
    &gateway, &dhcpserv, &dnsserv))
  {
    Serial.println(F("Unable to retrieve the IP Address!\r\n"));
    return false;
  }
  else
  {
    Serial.print(F("\nIP Addr: ")); cc3000.printIPdotsRev(ipAddress);
    Serial.print(F("\nNetmask: ")); cc3000.printIPdotsRev(netmask);
    Serial.print(F("\nGateway: ")); cc3000.printIPdotsRev(gateway);
    Serial.print(F("\nDHCPsrv: ")); cc3000.printIPdotsRev(dhcpserv);
    Serial.print(F("\nDNSserv: ")); cc3000.printIPdotsRev(dnsserv);
    Serial.println();
    return true;
  }
}
```

Make sure to modify the WiFi network name and password, and upload the code to the board. Then, open the Serial monitor to grab the IP address of the board, you will need it later.

### 2.4.4 Create the Meteor Application

It's now time to build the web-based application to control the board via a graphical interface. Here again, we are going to use the Meteor aREST module.

This is the template that we will use for the interface:

```
<template name='home'>

  <body>

    <div class='container'>

      <h3>Arduino WiFi Control & Monitoring</h3>

      <div class='row'>

        <div class="col-md-2">
          <button id='on' class='btn btn-primary btn-block' type="button">
            On
          </button>
        </div>

        <div class="col-md-2">
            <button id='off' class='btn btn-danger btn-block' type="button">
            Off
          </button>
        </div>

      </div>

      <div class='row'>
        <div class='col-md-3'>Temperature:
```

```
          <span id='temperature'></span> C</div>
        <div class='col-md-3'>Humidity:
          <span id='humidity'></span>%</div>
      </div>

    </div>

  </body>

</template>
```

If we compare this file to the interface we created in the previous section, we can notice that we added two buttons, that we will use to control the relay.

Let's now see the JavaScript file, that will contain all the code for the server & the client.

For the server, we just need to set the IP address of the target board using the aREST Meteor module:

```
if (Meteor.isServer) {
  Meteor.startup(function () {

    // Add device
    aREST.addDevice('http', '192.168.115.105');

  });
}
```

Now, inside the client code, we need to define two *events*, that will link the buttons inside the interface to actual commands sent to the board.

This is done by the following piece of code:

```
Template.home.events({

  'click #on': function() {
    Meteor.call('digitalWrite', 1, 7, 1);
  },
```

```
  'click #off': function() {
    Meteor.call('digitalWrite', 1, 7, 0);
  }

});
```

As you can see, you can call the digitalWrite() method inside Meteor, just as you would call the function inside an Arduino sketch.

Finally, we also need to do some things at the moment the page is loaded. This is done inside the rendered function of Meteor. This is the equivalent of the setup() function of Arduino.

Inside this function, we will set pin 7 as an output, and also refresh the measurements of the sensor:

```
Template.home.rendered = function() {

  // Set pin
  Meteor.call('pinMode', 1, 7, 'o');

  // Refresh temperature & humidity
  Meteor.call('getVariable', 1, 'temperature', function(err, data) {
    console.log(data);
    $('#temperature').text(data.temperature);
  });

  Meteor.call('getVariable', 1, 'humidity', function(err, data) {
    $('#humidity').text(data.humidity);
  });

}
```

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

https://github.com/marcoschwartz/discover-arest

It's now time to test the project! Grab all the code from the GitHub repository, and put all the files in a folder on your computer. Then, go to this folder with a terminal, and type:

```
meteor create .
```

This will initialize a new Meteor project in this folder. Then, install the aREST Meteor module with:

```
meteor add marcoschwartz:arest
```

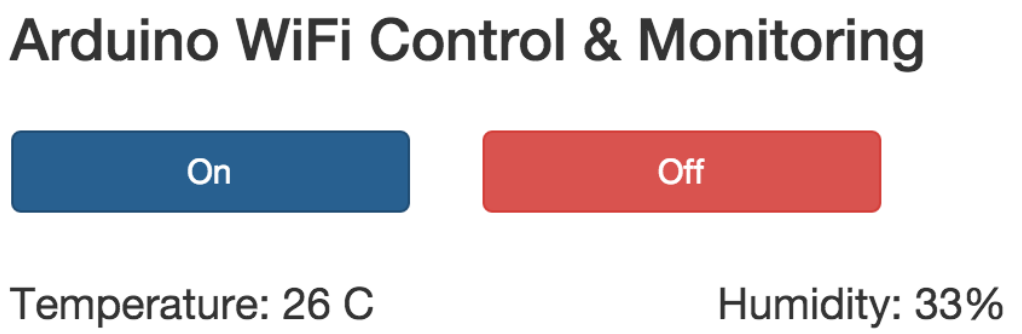
Finally, start the project with:

```
meteor
```

Now, navigate to the application with a web browser at:

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

You should immediately see the interface displayed on your computer:




You can try it now, for example to control the relay. We could stop here, but we are going to do an extra thing here. Because we now have a solid application running with the Meteor framework, it's easy for us to define triggers, for example to take a given action when a condition is met.

As an example, we'll set the relay to the 'On' state whenever the temperature falls below 20 degrees. This could simulate the behavior of a thermostat, for example.

To do that, we need to implement it on the server side. Indeed, you don't want to have the page opened in your browser so the application can check if the condition is met.

To implement that, we will the SyncedCron module for Meteor, that allows us to create automated tasks that will be repeated automatically on the server in the background.

This is the code for this part:

```
SyncedCron.add({
  name: 'Check temperature',
  schedule: function(parser) {
    return parser.text('every 1 minute');
  },
  job: function() {
    Meteor.call('getVariable', 1, 'temperature', function(err, data) {

      // Check temperature value
      if (data.temperature < 20) {

        // Activate relay
        Meteor.call('digitalWrite', 1, 7, 1);
      }
    });
  }
});
```

As you can see, whenever the temperature falls below 20 degrees, we simply switch the relay to on.

You can now build more complex applications based on the aREST Meteor module, by defining triggers that are executed when a given condition is met.

## 2.5 Get Access to Your Raspberry Pi From a Dashboard

In this chapter, we saw how to control Arduino boards as well as ESP8266 modules using web-based applications. But that's not everything that can be done by the aREST framework: you can also control Raspberry Pi boards right from the web application. And this is exactly what we are going to do in this project.

### 2.5.1 Hardware & Software Requirements

For this project, we are going to need components that we already used in this book: mainly a Raspberry Pi, that can be a Raspberry Pi B, B+, or 2.

This is the list of all the required components for this project:

- Raspberry Pi (https://www.adafruit.com/product/2358)
- LED (https://www.sparkfun.com/products/9590)
- 330 Ohm resistor (https://www.sparkfun.com/products/8377) (https://www.adafruit.com/products/386)
- Breadboard & jumper wires (https://www.adafruit.com/products/64)

### 2.5.2 Hardware Configuration

For this project, we are simply going to use a LED as an indicator for our Raspberry Pi. Simply place the LED in series with the resistor on a breadboard, the longest pin of the LED in contact with the resistor.

Then, connect the other end of the resistor to the GPIO pin 7 of the Raspberry Pi, and the other end of the LED to one GND pin of the Raspberry Pi.

### 2.5.3 Configure Your Raspberry Pi

Let's now see how to configure your Raspberry Pi, so it can be controlled from a web application. We are going to use again the pi-aREST module for this project.

This should be some code you are already familiar with:

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

piREST.set_id('34f5eQ');
piREST.set_name('my_new_Pi');
```

```
var server = app.listen(80, function() {
    console.log('Listening on port %d', server.address().port);
});
```

Save this code inside a JavaScript file on your Pi, and then navigate to the folder with a terminal. Then, type:

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

This will install all the required modules for this project. Then, start the software with:

```
sudo node arest.js
```

Your Raspberry Pi is now ready to be controlled remotely by the Meteor application.

### 2.5.4 Control Your Raspberry Pi from a Dashboard

Let's now see how to build the Meteor application to control your Raspberry Pi remotely. This is the code for the HTML file, that will define the interface:

```
<template name='home'>
  <body>
    <div class='container'>
      <h3>Raspberry Pi Control</h3>
      <div class='row'>
        <div class="col-md-2">
          <button id='on' class='btn btn-primary btn-block' type="button">
            On
          </button>
        </div>
        <div class="col-md-2">
            <button id='off' class='btn btn-danger btn-block' type="button">
            Off
          </button>
```

```
        </div>
      </div>
    </div>
  </body>
</template>
```

Then, inside the JavaScript file, we add the Raspberry Pi board to the Meteor application:

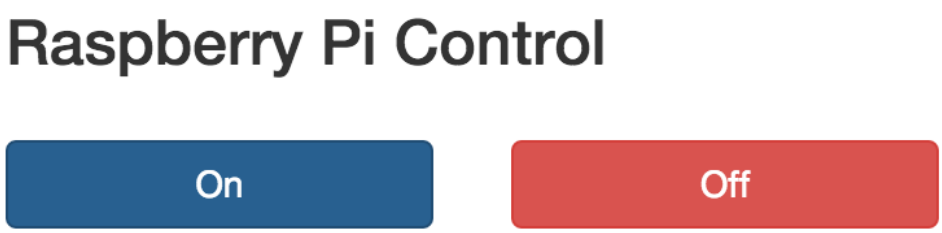
```
if (Meteor.isServer) {
  Meteor.startup(function () {

    // Add device
    aREST.addDevice('http', '192.168.115.107');

  });
}
```

For the client part, we link the buttons to the digitalWrite function of the aREST Meteor module:

```
if (Meteor.isClient) {

  // Main route
  Router.route('/', {name: 'home'});

  // Events
  Template.home.events({

    'click #on': function() {
      Meteor.call('digitalWrite', "34f5eQ", 7, 1);
    },
    'click #off': function() {
      Meteor.call('digitalWrite', "34f5eQ", 7, 0);
    }

  });
```

```
}
```

Now, repeat the procedure that we saw earlier to start the Meteor application, and then visit the server with the web browser. You should immediately have access to the interface:




## 2.6 Record Data From ESP8266 Devices

We are now able to control any board from a Meteor application running on your computer. In this section, we are now going to learn how to actually record data on the server.

As an example, we are going to make an ESP8266 board measure data, and send it to the server so it can be recorded. Instead of the server constantly querying the board for data, we are going to introduce a new feature of aREST: the publish() function, allowing the board to send data by itself to the server, that will then record this data inside a database.

### 2.6.1 Hardware & Software Requirements

You will need an ESP8266 board, like the Adafruit ESP8266 board that was used for this guide. To program the board, you will need a FTDI breakout board. You will also a DHT11 sensor, a breadboard, and some jumper wires.

This is the list of the required components for this part:

- Adafruit ESP8266 breakout board (https://www.adafruit.com/products/2471)

- DHT11 sensor (https://www.adafruit.com/products/386)
- FTDI USB converter (https://www.adafruit.com/products/284)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

### 2.6.2 Hardware Configuration

Let's now see how to assemble this project. First, place the ESP8266 module and the DHT11 sensor on the breadboard.

The DHT sensor signal pin (pin number 2 of the sensor) has to be connected on pin 7 of the ESP8266 board. Also connect the VCC pin (pin number 1 of the sensor) of the sensor to the ESP8266 VCC pin, and GND to GND.

### 2.6.3 Configure Your ESP8266 module

We are now going to configure the ESP8266 so it send data regularly to the server. We will make it publish the temperature and humidity measured by the sensor, at regular intervals.

This is the complete sketch for this part:

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

// DHT11 sensor pins
#define DHTPIN 5
#define DHTTYPE DHT11

// Remote server
char* servername = "192.168.115.101";
int port = 3000;

// Create aREST instance
aREST rest = aREST(servername, port);
```

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

// WiFi parameters
const char* ssid = "wifi-name";
const char* password = "wifi-password";

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

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

// Counter
int counter;

// Measurement interval
int measurement_interval = 60 * 1000; // 1 minute

// Variables
float temperature;
float humidity;

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

  // Init DHT
  dht.begin();

  // Give name and ID to device
  rest.set_id("1");
  rest.set_name("esp8266");

  // 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());

  // Start counter
  counter = millis();

}

void loop() {

  // Reading temperature and humidity
  humidity = dht.readHumidity();
  temperature = dht.readTemperature();

  // Publish
  if ( (millis() - counter) > measurement_interval) {
    if (restClient.connect(servername, port)) {
      rest.publish(restClient, "temperature", temperature);
      rest.publish(restClient, "humidity", humidity);
    }
    counter = millis();
  }

  // Handle REST calls
  WiFiClient client = server.available();
  if (!client) {
    return;
```

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

}
```

This sketch is quite similar to what we already saw in previous projects, with some differences. First, we define a server name and port at the beginning of the sketch:

```
char* servername = "192.168.115.101";
int port = 3000;
```

The servername variable is basically the IP address of your computer. This will tell the ESP8266 to send data to the server running on your computer. You need to modify this variable with the IP address of the computer where your Meteor server will run.

Then, we define the interval between two measurements made by the board:

```
int measurement_interval = 60 * 1000; // 1 minute
```

In the loop() function of the sketch, we actually publish the data at the interval we defined before:

```
if ( (millis() - counter) > measurement_interval) {
    if (restClient.connect(servername, port)) {
      rest.publish(restClient, "temperature", temperature);
      rest.publish(restClient, "humidity", humidity);
    }
    counter = millis();
  }
```

As you can see, the publish() function is quite simple to use: we just pass the client, the name of the event (here the variables names), and the value to publish.

You can now already upload this code to the board. However, it won't do anything until we have the server running on your computer.

### 2.6.4 Record Data to Your Meteor Application

We are now going to see how to record data inside a Meteor application, and how to access this data. Thanks to the Meteor aREST module, most of it will happen automatically in the background.

What we are going to do is to display all the measurements on the main page. This is the code for the interface of the main page:

```
<template name='home'>

  <body>

    <div class='container'>

      <h3>ESP8266 Monitoring</h3>

      <div class='row row-title'>
        <div class='col-md-3'>Time</div>
        <div class='col-md-1'>Device</div>
        <div class='col-md-2'>Variable</div>
        <div class='col-md-2'>Data</div>
      </div>

      {{#each events}}
        {{> event}}
      {{/each}}

    </div>

  </body>

</template>
```

You can see that we use a template called event, and this is what we need to define now. This is the definition of the event template, which is located inside the same file:

```
<template name='event'>
  <div class='row'>
    <div class='col-md-3'>
      <span class='label label-primary'>
        {{livestamp timestamp}}
      </span>
    </div>
    <div class='col-md-1'>{{device}}</div>
    <div class='col-md-2'>{{variable}}</div>
    <div class='col-md-2'>{{value}}</div>
  </div>
</template>
```

For each event (or each datapoint published by the board and recorded on the server), we will print the timestamp, the device ID, the variable name, and the value of the recorded data.

On the server side, we add the device on startup:

```
if (Meteor.isServer) {
  Meteor.startup(function () {
    aREST.addDevice('http', '192.168.115.104');
  });
}
```

For the client, we create a helper called event to return all the events recorded by the server:

```
if (Meteor.isClient) {

  // Main route
  Router.route('/', {name: 'home'});

  Template.home.helpers({

    events: function() {
      return Events.find({}, {sort: {timestamp: -1}});
```

```
    }

  });

}
```

It's now time to test the project! Make sure that you grab the code from the GitHub repository, and navigate to the folder where the files are located. Then, type:

```
meteor create .
```

This will initialize a new Meteor project in this folder. Then, install the aREST Meteor module with:

```
meteor add marcoschwartz:arest
```

Finally, start the project with:

```
meteor
```

Now, navigate to the app with a web browser at:

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

You should immediately see the interface displayed on your computer, with no recordings at the moment:


You should see that after a few seconds, some data will appear on the main page:

## ESP8266 Monitoring

| Time | Device | Variable | Data |
|---|---|---|---|
| a few seconds ago | 1 | humidity | 32.00 |
| a few seconds ago | 1 | temperature | 26.00 |

Every minute, you will see another couple of points being added to the page:

## ESP8266 Monitoring

| Time | Device | Variable | Data |
|---|---|---|---|
| a few seconds ago | 1 | humidity | 32.00 |
| a few seconds ago | 1 | temperature | 25.00 |
| 2 minutes ago | 1 | humidity | 32.00 |
| 2 minutes ago | 1 | temperature | 25.00 |
| 3 minutes ago | 1 | humidity | 32.00 |
| 3 minutes ago | 1 | temperature | 25.00 |
| 4 minutes ago | 1 | humidity | 32.00 |
| 4 minutes ago | 1 | temperature | 25.00 |
| 5 minutes ago | 1 | humidity | 32.00 |
| 5 minutes ago | 1 | temperature | 25.00 |
| 6 minutes ago | 1 | humidity | 32.00 |
| 6 minutes ago | 1 | temperature | 25.00 |
| 7 minutes ago | 1 | humidity | 32.00 |
| 7 minutes ago | 1 | temperature | 25.00 |

You can also use the API of the aREST Meteor module to get all events as a JSON array. Just go to:

```
http://localhost:3000/1/events
```

This will immediately return all the events that were recorded by this device so far on this server. You can now record events on your Meteor server, allowing you to build your own applications using those recordings! Of course, you can use those events as you wish inside your own applications.

## 2.7 Deploy the aREST Application on a Raspberry Pi

In this section, we are going to learn how to run an aREST server on the Raspberry Pi itself, without the need of any external component or service. We are also going to use the official Raspberry Pi camera and display the pictures taken by the Raspberry Pi right on a graphical interface.

### 2.7.1 Hardware & Software Requirements

Let's first see what we need for this project. You will need components we already used earlier in this book, like a Raspberry Pi board (B/B+/2), a DHT11 sensor, and a PowerSwitch tail kit. You will also need the official Raspberry Pi camera.

This is the list of all the components you will need for this project:

- Raspberry Pi (https://www.adafruit.com/product/2358)
- Powerswitch tail kit (https://www.adafruit.com/products/268)
- DHT11 sensor + 4.7k Ohm resistor (https://www.adafruit.com/products/386)
- Raspberry Pi camera (https://www.adafruit.com/products/1367)
- Breadboard & jumper wires (https://www.adafruit.com/products/64)

You will of course need to have your Raspberry Pi fully configured with Node.js installed, which should be the case at this point.

### 2.7.2 Hardware Configuration

Let's now see how to assemble the hardware for this project. First, place the DHT sensor on the breadboard. The DHT sensor signal pin (pin number 2 of the sensor) has to be connected on pin 4 of the Raspberry Pi. Also connect

the VCC pin (pin number 1 of the sensor) of the sensor to the RPi 3.3V, and GND to GND. Finally, insert the 4.7K Ohm resistor between pin number 1 and 2.

For the PowerSwitch Tail Kit, connect the Vin+ pin of the kit to the Raspberry Pi GPIO pin 7. Then, connect the two other pins to GND pins of the Raspberry Pi.

For the Raspberry Pi camera, the procedure is really simple. Simply insert the Raspberry Pi camera connector into the dedicated connector on the Raspberry Pi board. Then, secure the connection with the little piece of plastic on the Pi.

### 2.7.3 Create a Dashboard on Your Pi

We are now going to see how to create a graphical user interface for your Pi, that will run on the Raspberry Pi itself. This will be based on the pi-aREST module, that we already used before.

Inside the main app.js file of a typical pi-aREST project, you simply need to add a route for the interface file that we will code in moment:

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

Now, let's see the interface, defined in interface.jade. Jade is basically a templating language for HTML, that can allow us to easily create HTML files. This is the header of this file:

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

Then, this is how we define the interface, with buttons to control the Power-Switch Tail Kit, indicators for sensor data, and a space for the picture taken by the camera:

```
body
  .container
    h1 Raspberry Pi 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
    .row.voffset
      .col-md-6
        div#temperature Temperature:
      .col-md-6
        div#humidity Humidity:
    .row.voffset
      .col-md-6
        img#camera(width=640, height=360)
```

Now, we will define the behavior of the interface in a file called interface.js. First, we need to link the buttons of the interface to the corresponding aREST commands:

```
// Click on buttons
$("#on").click(function() {
  $.get('/digital/7/1');
});

$("#off").click(function() {
  $.get('/digital/7/0');
});
```

Then, we define functions to refresh the data coming from the sensor:

```
function refreshSensors() {

  $.get('/temperature', function(json_data) {
    $("#temperature").text('Temperature: ' + json_data.temperature + ' C');

    $.get('/humidity', function(json_data) {
      $("#humidity").text('Humidity: ' + json_data.humidity + ' %');
    });
  });

}
```

We also execute this function every 5 seconds:

```
refreshSensors();
setInterval(refreshSensors, 5000);
```

Now, let's see how to take a picture from the camera at regular intervals. This is something that is also integrated into the pi-aREST module, using the '/camera/snapshot' command. We define a function inside the script to take a new picture every 10 seconds, and we use this picture to refresh the picture inside the interface:

```
setInterval(function() {

  // Take picture
  $.get("/camera/snapshot");

}, 10000);

setInterval(function() {

  // Reload picture
  d = new Date();
  $("#camera").attr("src","/pictures/image.jpg?" + d.getTime());

}, 1000);
```

It's now time to test the project! Make sure to grab the code from the GitHub repository of the book, put all the files in a folder, and navigate to this folder using a terminal. Then, type:

```
sudo npm install node-dht-sensor express pi-arest
```

Be patient, this can take a while on older Raspberry Pi boards. Then, start the project with:

```
sudo node app.js
```

Now, you can access the interface by going to the IP address of your Pi, followed by port 3000, and the interface route. For example:

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

You should immediately see the interface inside your browser, with the buttons, sensor data, and also the picture taken by the Pi. Note that this interface is really running on the Pi itself, and does not require an external computer to work.

## 2.8 Record & Plot data from Your aREST Devices

In this section, we are going to learn how to record & plot live data coming from an Arduino board. We will connect a simple sensor to an Arduino Ethernet board, and make it send data to an aREST Meteor server. Then, on your computer we are going to use this data to make a live plot of the recorded data.

### 2.8.1 Hardware & Software Requirements

Let's first what you need for this project. We are going to use components that we already used before, like an Arduino Uno, and Ethernet shield, a a DHT11 sensor.

This is the list of all components we will use in this project:

- Arduino Uno (https://www.sparkfun.com/products/11021)
- Arduino Ethernet shield (https://www.sparkfun.com/products/11166)
- DHT11 sensor (https://www.adafruit.com/products/386)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

### 2.8.2 Hardware Configuration

First, place the DHT sensor on the breadboard. The DHT sensor signal pin (pin number 2 of the sensor) has to be connected on pin 7 of the Arduino board. Also connect the VCC pin (pin number 1 of the sensor) of the sensor to the Arduino 5V pin, and GND to GND. Finally, insert the 4.7K Ohm resistor between pin number 1 and 2.

### 2.8.3 Configuring Your Ethernet Board

Let's now configure the board. This is a sketch we already more or less, as it uses the same publish() function that we discovered earlier in the chapter.

This is the complete sketch for this part:

```
// Libraries
#include <SPI.h>
#include <Ethernet.h>
#include <aREST.h>
#include "DHT.h"

// DHT 11 sensor
#define DHTPIN 7
#define DHTTYPE DHT11

// Remote server
char* servername = "192.168.115.101";
int port = 3000;

// DHT sensor
DHT dht(DHTPIN, DHTTYPE);
```

```
// Enter a MAC address for your controller below.
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0E, 0xFE, 0x40 };

// IP address in case DHCP fails
IPAddress ip(192,168,2,2);

// Ethernet server & client
EthernetServer server(80);
EthernetClient restClient;

// Create aREST instance
aREST rest = aREST(servername, port);

// Counter
int counter;

// Measurement interval
int measurement_interval = 60 * 1000; // 1 minute

// Variables to be exposed to the API
int temperature;
int humidity;

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

  // Init variables and expose them to REST API
  rest.variable("temperature",&temperature);
  rest.variable("humidity",&humidity);

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

  // Start the Ethernet connection and the server
```

```
  if (Ethernet.begin(mac) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    // no point in carrying on, so do nothing forevermore:
    // try to congifure using IP address instead of DHCP:
    Ethernet.begin(mac, ip);
  }
  server.begin();
  Serial.print("server is at ");
  Serial.println(Ethernet.localIP());

  // Start counter
  counter = millis();

}

void loop() {

  // Make measurements
  humidity = (int)dht.readHumidity();
  temperature = (int)dht.readTemperature();

  // Publish
  if ( (millis() - counter) > measurement_interval) {
    if (restClient.connect(servername, port)) {
      rest.publish(restClient, "temperature", temperature);
      rest.publish(restClient, "humidity", humidity);
    }
    counter = millis();
  }

  // Listen for incoming clients
  EthernetClient client = server.available();
  rest.handle(client);

}
```

There are some things you will need to modify in this sketch. First, modify the MAC address and put the one from your Ethernet shield.

Then, you need to set the IP address of the server inside the sketch:

```
char* servername = "192.168.115.101";
int port = 3000;
```

In the loop() function, we publish data (temperature and humidity) to the server at regular intervals:

```
if ( (millis() - counter) > measurement_interval) {
  if (restClient.connect(servername, port)) {
    rest.publish(restClient, "temperature", temperature);
    rest.publish(restClient, "humidity", humidity);
  }
  counter = millis();
}
```

You can now already upload the code to the board. Also open the Serial monitor to get the IP address of the board. At this point it won't do anything, as you need the server to be running on your computer.

### 2.8.4 Recording & Plotting Data

We are now going to see how to configure a server using the Meteor aREST module, so it can record data, and plot it in live straight into your web browser.

Let's first define the interface. This is the home template, where we define two graphs:

```
<template name='home'>

  <body>

    <div class='container'>

      <h3>Ethernet Monitoring</h3>
```

```
      <div class='row'>

        <div class='col-md-6'>
          {{> highchartsHelper chartId="temperature"
            chartWidth="100%" charHeight="100%"
            chartObject=temperatureChart}}
        </div>
        <div class='col-md-6'>
          {{> highchartsHelper chartId="humidity"
            chartWidth="100%" charHeight="100%"
            chartObject=humidityChart}}
        </div>
      </div>

    </div>

  </body>

</template>
```

The most important elements in this interface are the charts, defined by the following line (here for temperature):

```
{{> highchartsHelper chartId="temperature"
  chartWidth="100%" charHeight="100%"
  chartObject=temperatureChart}}
```

For this project, we are going to use HighCharts (http://www.highcharts.com/) to plot the data recorded on the server. This is a very convenient library to plot data in JavaScript applications, and it is easy to integrate in Meteor.

On the server side, we need to add the board in our project:

```
if (Meteor.isServer) {
  Meteor.startup(function () {
    aREST.addDevice('http', '192.168.115.104');
  });
}
```

Then, in the rendered function, we need to format the data so it can be used by the charts elements. This is done by the following piece of code:

```
Template.home.rendered = function() {

  Tracker.autorun(function () {

    // Get all events for temperature
    eventsTemperature = Events.find({variable: 'temperature'}).fetch();
    eventsHumidity = Events.find({variable: 'humidity'}).fetch();

    // Split in arrays
    timestamps = [];
    temperatureData = [];
    humidityData = [];
    for (i = 0; i < eventsTemperature.length; i++){
      timestamps.push(moment(eventsTemperature[i].timestamp).format('HH:mm'));
      temperatureData.push(parseFloat(eventsTemperature[i].value));
      humidityData.push(parseFloat(eventsHumidity[i].value));
    }

    // Set session variables
    Session.set('timestamps', timestamps);
    Session.set('temperatureData', temperatureData);
    Session.set('humidityData', humidityData);

  });

}
```

Especially notice this function:

```
Tracker.autorun()
```

This will ensure that the data is refreshed whenever a new measurement point comes in, and therefore that we will plot live data.

Then, we can finally link this data to the charts we defined before. This is for example for the temperature chart:

```
Template.home.temperatureChart = function() {

    return {
        title: {
            text: 'Temperature',
            x: -20 //center
        },
        xAxis: {
            categories: Session.get('timestamps'),
            tickInterval: 10
        },
        yAxis: {
            title: {
                text: 'Temperature (°C)'
            },
            plotLines: [{
                value: 0,
                width: 1,
                color: '#808080'
            }]
        },
        tooltip: {
            valueSuffix: '°C'
        },
        series: [{
            name: 'Temperature',
            data: Session.get('temperatureData')
        }]
    };
};
```

It's now time to test the project! First, grab all the code from the GitHub repository of the book:

https://github.com/marcoschwartz/discover-arest

Then, navigate to the folder where the files are located with a terminal, and type:

```
meteor create .
```

After that, add the aREST Meteor module and the HighCharts module with:

```
meteor add maazalik:highcharts marcoschwartz:arest
```

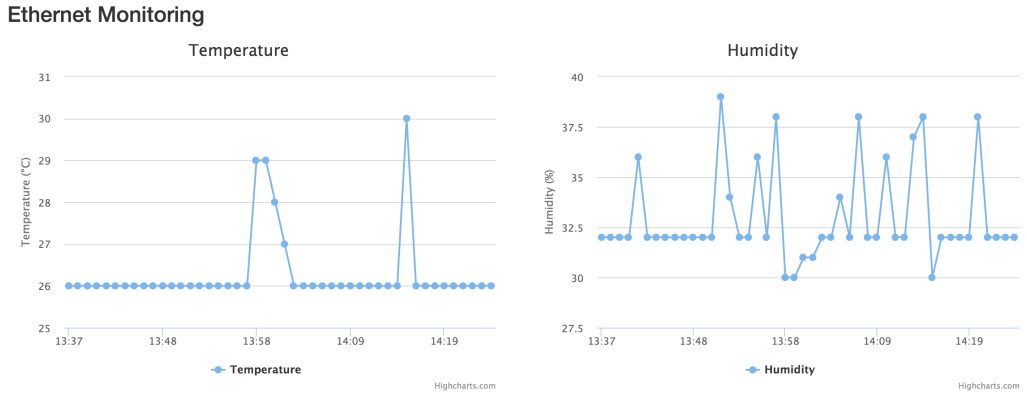
Finally, start the application with:

```
meteor
```

To access the plots, go to:

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

You should immediately see the live recording of the data being displayed on your screen. This is the result after several minutes of recording:




You can of course build on this project to make other recording applications. For example, you can personalize the graphs for your own needs, by visiting the HighCharts website and looking at the examples there. You can also add more boards to the projects, and plot all the data in live inside the application.

## 2.9 Example project: Simple Wireless Home Automation System

As an example for this chapter, we are going to build another home automation project. This time, we are going to use what we learned in this chapter to build a simple home automation system based on the ESP8266 board.

We will use three different boards to make a simple motion sensor, a lamp controller, and temperature & humidity sensor. Then, we will integrate everything into a single dashboard.

### 2.9.1 Hardware & Software Requirements

First, let's see what we need for this project. There will be three parts in this project: a temperature & humidity measurement module, a lamp controller module, and a motion sensor module.

These are the required components for the temperature & humidity sensor module:

- Adafruit ESP8266 breakout board (https://www.adafruit.com/products/2471)
- DHT11 sensor (https://www.adafruit.com/products/386)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

These are the required components for the lamp controller module:

- Adafruit ESP8266 breakout board (https://www.adafruit.com/products/2471)
- PowerSwitch tail (https://www.adafruit.com/products/268)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

These are the required components for the motion sensor module:

- Adafruit ESP8266 breakout board (https://www.adafruit.com/products/2471)

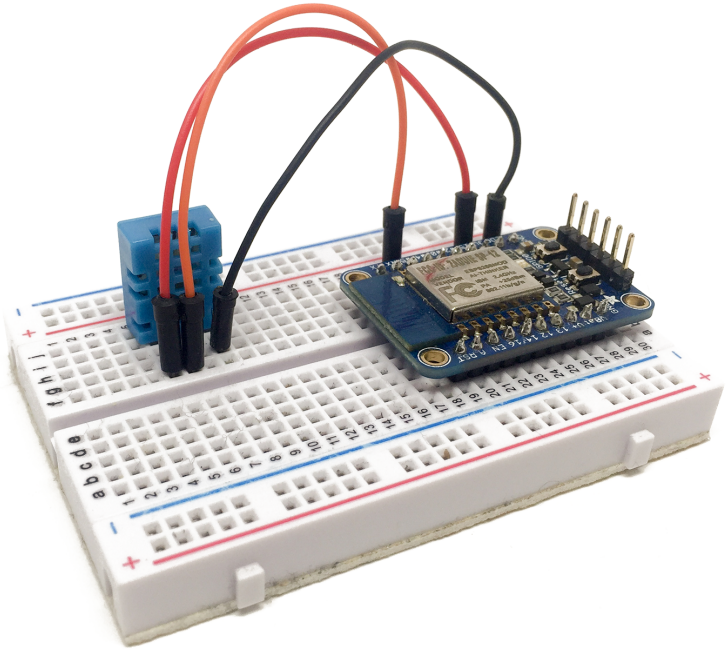
- PIR motion sensor (https://www.adafruit.com/products/189)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

For all modules, you will need a FTDI USB converter to program the ESP8266 chip:

- FTDI USB converter (https://www.adafruit.com/products/284)

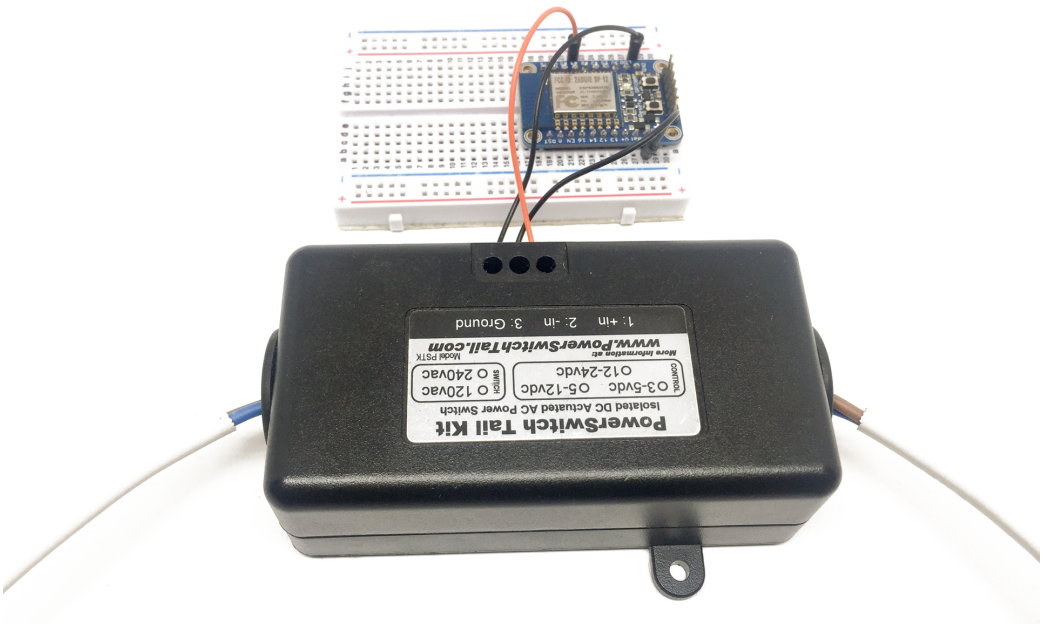
### 2.9.2 Hardware Configuration

Let's first assemble the sensor board. Connect the DHT sensor signal pin (pin number 2 of the sensor) has to be connected on pin 4 of the ESP8266. Also connect the VCC pin (pin number 1 of the sensor) of the sensor to the ESP8266 3.3V pin, and GND to GND. This is the completely assembled sensor module:



Now, for the lamp controller, connect the Vin+ pin of the kit to the ESP8266 pin 4. Then, connect the two other pins to GND pins of the ESP8266.

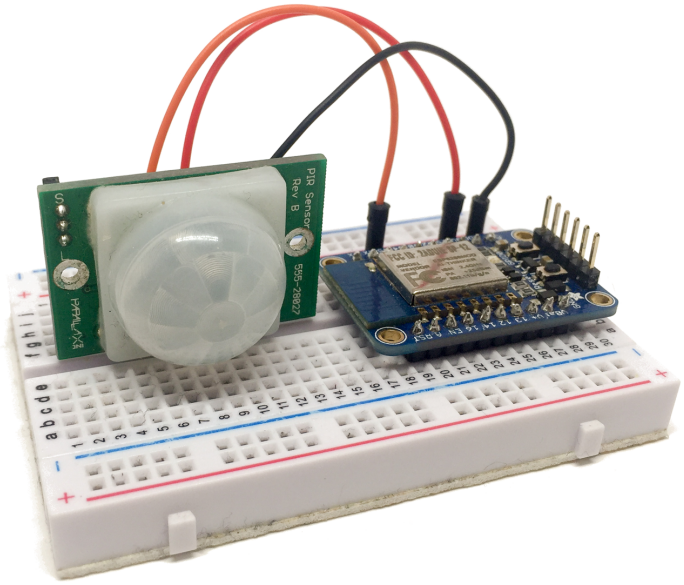
This is the assembled lamp controller:




Also connect a lamp or another appliance to the PowerSwitch Tail, and the other end of the PowerSwitch Tail to the mains electricity.

The motion sensor module is quite easy to assemble. Connect the OUT (or SIG) pin of the motion sensor to pin number 5 of the ESP8266. Then, connect the VCC pin of the sensor to the 3.3V, and GND to GND.

This is the completely assembled motion sensor:




### 2.9.3 Configure Your Sensor Board

We are now going to configure one board after the other, starting by the sensor board. This is the complete code for this module:

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

// DHT11 sensor pins
#define DHTPIN 5
#define DHTTYPE DHT11

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

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

```
// WiFi parameters
const char* ssid = "wifi-name";
const char* password = "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);

  // Init DHT
  dht.begin();

  // Give name and ID to device
  rest.set_id("1");
  rest.set_name("esp8266_control");

  // Expose variables
  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());

}

void loop() {

  // Reading temperature and humidity
  humidity = dht.readHumidity();
  temperature = dht.readTemperature();

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

}
```

This is already a sketch with saw several times, so I won't detail it here. Just make sure to change your WiFi name and password, and upload the sketch to the board. Then, open the Serial monitor and reset the board to get the IP address of the board.

Now, just check that the board is responding to aREST commands by typing the following command in your web browser, of course by changing the IP address:

```
http://192.168.115.104/temperature
```

You should get the answer in JSON format:

```
{
  "temperature": 25.00,
  "id": "1",
  "name": "esp8266_control",
  "connected": true
}
```

### 2.9.4 Configure Your Lamp Control Board

Let's now move on to the lamp controller module. This is the complete sketch for this part:

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

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

// WiFi parameters
const char* ssid = "wifi-name";
const char* password = "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("2");
  rest.set_name("esp8266_lamp");

  // 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);

}
```

This is already a sketch with saw several times, so I won't detail it here. Just make sure to change your WiFi name and password, and upload the sketch to the board. Then, open the Serial monitor and reset the board to get the

IP address of the board.

Now, just check that the board is responding to aREST commands by typing the following command in your web browser, of course by changing the IP address:

```
http://192.168.115.105/mode/4/o
```

Then, switch on the lamp or the appliance connected to the PowerSwitch Tail by typing:

```
http://192.168.115.105/digital/4/1
```

### 2.9.5 Configure the Motion Sensor Board

Finally, we are going to configure the motion sensor board. This is the complete sketch for this part:

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

// Remote server
char* servername = "192.168.115.101";
int port = 3000;

// Create aREST instance
aREST rest = aREST(servername, port);

// WiFi parameters
const char* ssid = "wifi-name";
const char* password = "wifi-password";

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

// Create an instance of the server
```

```
WiFiServer server(LISTEN_PORT);
WiFiClient restClient;

// Variable
bool motion = false;

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

  // Give name and ID to device
  rest.set_id("3");
  rest.set_name("esp8266_motion");

  // 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());

  // Set motion sensor pin
  pinMode(5, INPUT);

}

void loop() {
```

```
  // Reading temperature and humidity
  bool motionStatus = digitalRead(5);

  // Publish
  if (motionStatus != motion) {
    if (restClient.connect(servername, port)) {
      rest.publish(restClient, "motion", motionStatus);
    }
  }
  motion = motionStatus;

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

}
```

This is already a sketch with saw several times, so I won't detail it too much here. What changes here is that we only publish data to the server when we detect a change on the motion detector, which is done by the following piece of code:

```
// Reading temperature and humidity
bool motionStatus = digitalRead(5);

// Publish
if (motionStatus != motion) {
  if (restClient.connect(servername, port)) {
    rest.publish(restClient, "motion", motionStatus);
  }
}
motion = motionStatus;
```

Just make sure to change your WiFi name and password, as well as the IP address of your computer, and upload the sketch to the board. Then, open the Serial monitor and reset the board to get the IP address of the board.

### 2.9.6 Build Your Home Automation Dashboard

It's now time to build the home automation dashboard that will allow us to control everything within the same interface. It will again be based on the Meteor framework, along with the aREST Meteor module.

We first define the interface of the project:

```
<template name='home'>
  <body>
    <div class='container'>
      <h3>Home Automation System</h3>

      <div class='row'>
        <div class='col-md-3'>Temperature:
          <span id='temperature'></span> C</div>
        <div class='col-md-3'>Humidity:
          <span id='humidity'></span>%</div>
      </div>

      <div class='row'>
        <div class="col-md-2">
          <button id='on' class='btn btn-primary btn-block' type="button">
            On
          </button>
        </div>
        <div class="col-md-2">
            <button id='off' class='btn btn-danger btn-block' type="button">
            Off
          </button>
        </div>
      </div>

      <div class='row'>
```

```
        <div class="col-md-2">Motion sensor:</div>
        <div class="col-md-2"><span id='motion'>{{motionStatus}}</span></div>
      </div>

    </div>
  </body>
</template>
```

The interface is composed of three elements: buttons to control the lamp, text indicators for the sensor readings, and finally another indicator for the motion sensor status.

Then, on the server side, we add the three modules to the server:

```
if (Meteor.isServer) {
  Meteor.startup(function () {

    // Add device
    aREST.addDevice('http', '192.168.115.104');
    aREST.addDevice('http', '192.168.115.105');
    aREST.addDevice('http', '192.168.115.106');

  });
}
```

Then, in the rendered function, we set pin 4 to an output, and also refresh the measurements from the sensors:

```
Template.home.rendered = function() {

  // Set pin
  Meteor.call('pinMode', 2, 4, 'o');

  // Refresh temperature & humidity
  Meteor.call('getVariable', 1, 'temperature', function(err, data) {
    console.log(data);
    $('#temperature').text(data.temperature);
```

```
  });

  Meteor.call('getVariable', 1, 'humidity', function(err, data) {
    $('#humidity').text(data.humidity);
  });

}
```

We also create a dedicated helper for the motion sensor status. As this is directly linked to the database of Meteor, this will be automatically refreshed whenever a new status is recorded by the application. This is the code to do that:

```
Template.home.helpers({

  motionStatus: function() {

    // Motion sensor
    var sensorState = Events.find({}, {sort: {timestamp: -1}}).fetch()[0];
    console.log(sensorState);

    if (sensorState.value == '1') {
      return 'Motion detected';
    }
    if (sensorState.value == '0') {
      return 'No motion';
    }

  }

});
```

Finally, we define two events to link the buttons inside the interface:

```
Template.home.events({

  'click #on': function() {
```

```
    Meteor.call('digitalWrite', "2", 4, 1);
  },
  'click #off': function() {
    Meteor.call('digitalWrite', "2", 4, 0);
  }

});
```

Finally, we define a route for the page containing the interface:

```
Router.route('/', {name: 'home'});
```

Now, grab all the code from the GitHub repository of the book, and put it all inside a folder on your computer. Then, initialise a new Meteor project with:

```
meteor create .
```

Then, install the aREST meteor module with:

```
meteor add marcoschwartz:arest
```

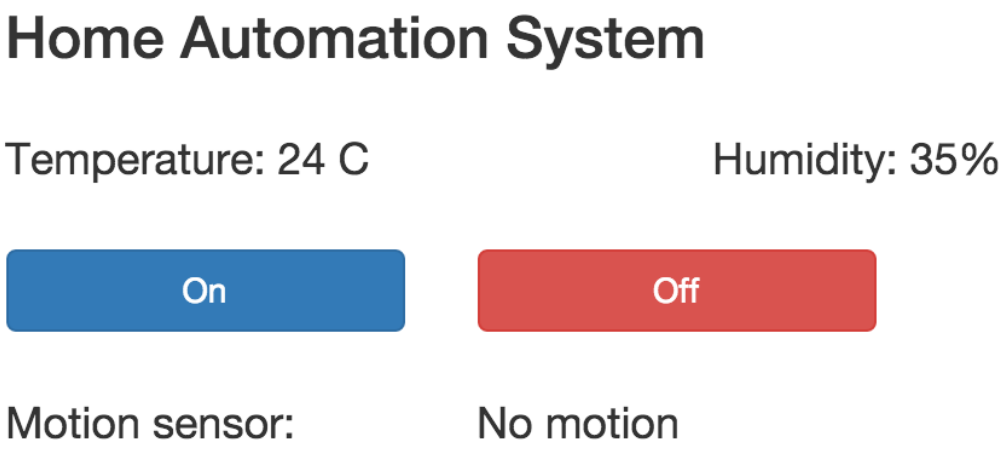
Finally, start the project using:

```
meteor
```

You can now go to your favorite browser and type:

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

You should see the interface of your home automation system:




You can now try it, for example by clicking on a button. You can also pass your hand in front of the motion sensor: the indicator inside the interface will change immediately.

You can of course now connect more modules to the interface, as it's really easy to do using the Meteor aREST module. At the end, you can have a complex home automation system based on the aREST framework.

## 2.10 How to Go Further

In this chapter, we learned how to control your aREST projects using graphical interfaces running on your computer. We learned how to control your projects from a simple web page, and also from a web server. Finally, as an example for this chapter, we built a complete home automation system based on aREST.

You can now experiment with what you learned in this chapter, and create your own projects based on it. What we saw in this chapter is especially useful when you want to create projects with several aREST boards, that you want to control & monitor within the same interface.

In the next chapter, we are going to see another important part of aREST that was introduced in 2015 into the framework: the cloud access. You will learn how to access any of your aREST projects from anywhere in the world, and control your projects from cloud dashboards.

# 3 Access Your Boards From the Cloud

This chapter will be focused on the cloud access of the aREST framework. Indeed, aREST makes it really easy to control your projects from anywhere in the world. We will learn how to control individual boards from a cloud server, and then how to control several boards from a cloud dashboard. At the end of the chapter, you will even learn how to deploy your own aREST cloud server!

## 3.1 Access your Arduino Boards from Anywhere

aREST can be used to control Arduino boards from anywhere in the world, by communicating with the board via MQTT commands through the Ethernet library. The Arduino board basically acts as a web client that accepts incoming commands from the cloud.arest.io website, and process those commands via the library. You can then control your Arduino board via any web browser, anywhere in the world, via cURL calls, or via web-based applications.

### 3.1.1 Hardware & Software Requirements

You will need an Arduino board, like an Arduino Uno that was used for this project. You will also need an Ethernet shield for Arduino. You will also need one LED, one 220 Ohm resistor, a breadboard, and some jumper wires.

- Arduino Uno (https://www.sparkfun.com/products/11021)
- Arduino Ethernet shield (https://www.sparkfun.com/products/11166)
- LED (https://www.sparkfun.com/products/9590)
- 330 Ohm resistor (https://www.sparkfun.com/products/8377)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

You will also need to install the PubSubClient library via the Arduino library manager.

### 3.1.2 Hardware Configuration

Simply place the Arduino Ethernet shield on your Arduino board, and connect the shield to your local network using an Ethernet cable.

For the LED, simply connect it in series with the resistor, with the longest pin of the LED connected to the resistor. Then, connect the remaining pin of the resistor to pin 5 of the Arduino board, and the remaining pin of the LED to the GND pin.

### 3.1.3 Control Your Boards from the Cloud

We are now going to configure your Arduino board so it can be accessed from anywhere in the world. As we will see, the sketch is mainly similar to the one we used to control our aREST projects within your own local network.

This is the complete sketch for this part:

```
// Libraries
#include <SPI.h>
#include "Ethernet.h"
#include <PubSubClient.h>
#include <aREST.h>

// Enter a MAC address for your controller below.
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0E, 0xFE, 0x40 };

// Clients
EthernetClient ethClient;
PubSubClient client(ethClient);

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

// Unique ID to identify the device for cloud.arest.io
char* device_id = "9u2co4";

void setup() {
```

```
  // Open serial communications and wait for port to open:
  Serial.begin(115200);
  Serial.println("Hello aREST !");

  // Set callback
  client.setCallback(callback);

  // Give name and ID to device
  rest.set_id(device_id);
  rest.set_name("arduino");

  // Start the Ethernet connection
  if (Ethernet.begin(mac) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
  }

  // Set output topic
  char* out_topic = rest.get_topic();
}


void loop() {

  // Connect to the cloud
  rest.loop(client);

}

// Handles message arrived on subscribed topic(s)
void callback(char* topic, byte* payload, unsigned int length) {

  rest.handle_callback(client, topic, payload, length);

}
```

Let's see what we modified compared to other aREST sketches we saw so far. The cloud version of aREST uses MQTT t communicate with a server

deployed in the cloud, so we'll need to set several things related to MQTT communications.

First, we included the PubSubClient library in addition to the other required libraries:

```
#include <SPI.h>
#include "Ethernet.h"
#include <PubSubClient.h>
#include <aREST.h>
```

We also need to client a PubSubClient client instance, and to pass it to aREST:

```
PubSubClient client(ethClient);
aREST rest = aREST(client);
```

Then, in the setup() function, we set the callback. We'll see what that does later:

```
client.setCallback(callback);
```

We also set an ID for our aREST device. This is very important, as it will identify the device on the cloud server. You will use this ID later to control your board from the cloud server. Please modify this ID here so you have your own unique ID on the server:

```
char* device_id = "9u2co4";
```

We also set the output topic for the aREST cloud server communications:

```
char* out_topic = rest.get_topic();
```

In the loop() function, we simply need to pass the client instance to aREST:

```
rest.handle(client);
```

Finally, we define the callback that we set before:

```
void callback(char* topic, byte* payload, unsigned int length) {

  rest.handle_callback(client, topic, payload, length);

}
```

This function will simply handle the data received by the board via MQTT from the cloud server, and answer accordingly.

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

https://github.com/marcoschwartz/discover-arest

You can now open the Arduino IDE and open the code for this project. Save the sketch somewhere, and modify MAC address inside the sketch. You will find the MAC address below the Ethernet shield. Also give an unique ID to your board. Then, upload the sketch to the Arduino board.

Go to your favorite web browser, and type:

```
cloud.arest.io/9u2co4/id
```

You should immediately get the answer from the server:

```
{
  "id": "9u2co4",
  "name": "arduino",
  "connected": true
}
```

Now, try to power down the device, wait around 10 seconds, and type the same command again. You should get the following answer:

```
{
  "message": "Requested device is not online",
  "connected": false,
  "id": "9u2co4"
}
```

Next, activate the device again, and type the following inside your browser:

```
cloud.arest.io/9u2co4/mode/5/o
```

This will make pin number 5, which is connected to the LED, to an output. After that, type:

```
cloud.arest.io/9u2co4/digital/5/1
```

You should get the following answer:

```
{
  "message": "Pin D5 set to 1",
  "id": "9u2co4",
  "name": "arduino",
  "connected": true
}
```

This will make a digitalWrite() command to pin 5 and it will turn the LED on. If that works, it means that aREST is working correctly, and that you can access your board from anywhere in the world.

## 3.2 Control your Raspberry Pi using a Cloud API

So far in this book, we saw how to control your projects based on the Raspberry Pi from within your local network, using the aREST framework. In this section, we are going to take another approach. We are going to see how to connect your Raspberry Pi to the aREST cloud server, so it can be controlled from anywhere in the world.

You will then be able to call the aREST API from any web browser or web application. This will allow you to control your Raspberry Pi from the dashboard that comes with the API, but also from your own web applications if you wish. As an example, we will see how to read the data from a sensor & how to control a lamp.

### 3.2.1 Hardware & Software Requirements

In order to illustrate the functionalities of the project in this chapter, we are going to use a setup that we already saw earlier in this book: we will connect a DHT11 sensor to the Raspberry Pi to measure temperature & humidity, and also a PowerSwitch Tail Kit to control a lamp or other electrical devices.

This is the list of the components you will need for this project:

- Raspberry Pi (https://www.adafruit.com/product/2358)
- DHT11 sensor + 4.7k Ohm resistor (https://www.adafruit.com/products/386)
- Powerswitch tail kit (https://www.adafruit.com/products/268)
- Breadboard & jumper wires (https://www.adafruit.com/products/64)

### 3.2.2 Hardware Configuration

The hardware configuration for this project is quite simple, as this is something we already saw earlier in the book. For this project, you can either use a cobbler kit as we did in earlier chapters, or just wire directly the Pi to the breadboard.

The first step is to connect the pin number 1 of the PowerSwitch to pin number 13 of the Raspberry Pi. Finally, connect the two other pins of the PowerSwitch to a GND pin of the Raspberry Pi.

The DHT sensor signal pin (pin number 2 of the sensor) has to be connected on pin 4 of the Raspberry Pi. Also connect the VCC pin (pin number 1 of the sensor) of the sensor to the RPi 3.3V, and GND to GND. Finally, insert the 4.7K Ohm resistor between pin number 1 and 2.

### 3.2.3 Connecting to the Cloud Server

It's now time to connect our Raspberry Pi project to the aREST cloud server, that we will use to control it remotely.

As usual, we will use Node.js to connect our Pi to the aREST live server which is deployed in the cloud. This is the complete code for this part:

```
// Libraries
var express = require('express');
var app = express();
var piREST = require('pi-arest')(app);
var sensorLib = require('node-dht-sensor');

// Set ID & name
piREST.set_id('p5dgwt');
piREST.set_name('pi_cloud');

// Connect to cloud.aREST.io
piREST.connect();

// Start server
var server = app.listen(80, function() {
    console.log('Listening on port %d', server.address().port);
});

// Sensor readout
var sensor = {
  initialize: function () {
    return sensorLib.initialize(11, 4);
  },
  read: function () {

    // Read
    var readout = sensorLib.read();
    temperature = readout.temperature.toFixed(2);
    humidity = readout.humidity.toFixed(2);

    // Set variables
    piREST.variable('temperature', temperature);
    piREST.variable('humidity', humidity);
    console.log('Temperature: ' + temperature + 'C, ' +
        'humidity: ' + humidity + '%');

    // Repeat
    setTimeout(function () {
```

```
        sensor.read();
    }, 2000);
  }
};

// Init sensor
if (sensor.initialize()) {
  sensor.read();
} else {
  console.warn('Failed to initialize sensor');
}
```

Let's now see the most important parts of this code. First, we declare the Node.js modules that we will use for this project:

```
var express = require('express');
var app = express();
var piREST = require('pi-arest')(app);
var sensorLib = require('node-dht-sensor');
```

Then, we set an ID & a name to our project. Make sure to modify the ID, as it will identify your Raspberry Pi project on the aREST server:

```
piREST.set_id('p5dgwt');
piREST.set_name('pi_cloud');
```

After that, the rest of the code is dedicated to connecting to the cloud server. At the end of the code, we initialise the DHT11 sensor:

```
if (sensor.initialize()) {
  sensor.read();
} else {
  console.warn('Failed to initialize sensor');
}
```

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

https://github.com/marcoschwartz/discover-arest

It's now time to test it. Put all the code into a file (named here cloud.js), and inside the folder where the file is located type:

```
sudo npm install pi-arest express node-dht-sensor
```

This will take a while, especially on the first versions of the Raspberry Pi. After that, type:

```
sudo node cloud.js
```

This will connect the Raspberry Pi to the cloud API, and you should see the confirmation in the console.

We are now going to use the cloud API 'by hand' before using it from a dashboard later in this book. For that, go to your favorite web browser, and type:

```
https://cloud.arest.io/p5dgwt/temperature
```

Of course, replace the ID of the board in the URL by the one you set in the code. You should get the answer of the Pi in JSON format:

```
{
  "id": "p5dgwt",
  "name": "pi_cloud",
  "hardware": "rpi",
  "connected": true,
  "temperature": 24
}
```

As you can see, the temperature is returned as a field in the JSON object. This will be useful for later to grab the temperature from a dashboard.

We can now do the same to turn on any electrical device connected to the PowerSwitch tail kit:

```
https://cloud.arest.io/p5dgwt/digital/13/1
```

You will get the confirmation as a JSON message:

```
{
  "id": "p5dgwt",
  "name": "pi_cloud",
  "hardware": "rpi",
  "connected": true,
  "message": "Pin 13 set to 1"
}
```

It's really important here to notice that those commands are called from a cloud server, and that they can be called from anywhere in the world!

## 3.3 Use Your ESP8266 Boards as Internet of Things Modules

aREST can be used to control ESP8266 boards from anywhere in the world, by communicating with the ESP8266 via MQTT commands. The ESP8266 board basically acts as a web client that accepts incoming commands from the cloud.arest.io website, and process those commands via the library. You can then control your ESP8266 board via any web browser, anywhere in the world, via cURL calls, or via web-based applications.

### 3.3.1 Hardware & Software Configuration

You will need an ESP8266 board, like the Adafruit ESP8266 board that was used for this guide. To program the board, you will need a FTDI breakout board. You will also need one LED, one 220 Ohm resistor, a breadboard, and some jumper wires.

- Adafruit ESP8266 breakout board (https://www.adafruit.com/products/2471)
- LED (https://www.sparkfun.com/products/9590)

- 330 Ohm resistor (https://www.sparkfun.com/products/8377)
- FTDI USB converter (https://www.adafruit.com/products/284)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

You will need to get & install the aREST Arduino library and the PubSub library. You can install those libraries using the Arduino library manager.

You will also need to install the ESP8266 boards definitions for Arduino. You can find the procedure at the following link:

https://github.com/esp8266/Arduino

### 3.3.2 Hardware Configuration

Simply place the ESP8266 board on your breadboard, and then connect the FTDI breakout board to it.

For the LED, simply connect it in series with the resistor, with the longest pin of the LED connected to the resistor. Then, connect the remaining pin of the resistor to pin 5 of the ESP8266 board, and the remaining pin of the LED to the GND pin.

### 3.3.3 Controlling Your ESP8266 Remotely

We are now going to connect your ESP8266 board to the aREST cloud server. This is the complete code for this part:

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

// Clients
WiFiClient espClient;
PubSubClient client(espClient);

// Create aREST instance
```

```
aREST rest = aREST(client);

// Unique ID to identify the device for cloud.arest.io
char* device_id = "9u2co4";

// WiFi parameters
const char* ssid = "wifi-name";
const char* password = "wifi-password";

// Declare functions to be exposed to the API
int ledControl(String command);

// Functions
void callback(char* topic, byte* payload, unsigned int length);

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

  // Set callback
  client.setCallback(callback);

   // Function to be exposed
  rest.function("led",ledControl);

  // Give name and ID to device
  rest.set_id(device_id);
  rest.set_name("esp8266");

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

```
  // Set output topic
  char* out_topic = rest.get_topic();

}

void loop() {

  // Connect to the cloud
  rest.loop(client);

}

// Handles message arrived on subscribed topic(s)
void callback(char* topic, byte* payload, unsigned int length) {

  rest.handle_callback(client, topic, payload, length);

}

// Custom function accessible by the API
int ledControl(String command) {

  // Get state from command
  int state = command.toInt();

  digitalWrite(5,state);
  return 1;
}
```

As you can see, the code is quite similar to what we saw earlier in this book. What you need to change here is the device ID, which identifies the device on the network:

```
char* device_id = "9u2co4";
```

Then, modify the WiFi network name and password in the code:

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

You can now open the Arduino IDE, and grab the code from the GitHub repository of the book. Save the sketch somewhere, and modify the WiFi name & password inside the sketch. Also give an unique ID to your board. Then, upload the sketch to the ESP8266 board. Go to your favourite web browser, and type:

```
cloud.arest.io/9u2co4/id
```

You should immediately get the answer in JSON format:

```
{
  "id": "9u2co4",
  "name": "esp8266",
  "connected": true
}
```

It's also easy to call a function present on the board, that has been exposed to the aREST API. First, type:

```
cloud.arest.io/9u2co4/mode/5/o
```

After that, type the following command to call the function:

```
cloud.arest.io/9u2co4/led?params=1
```

Then, you will get the answer in JSON format:

```
{
  "return_value": 1,
  "id": "9u2co4",
  "name": "esp8266",
  "connected": true
}
```

You can now control your ESP8266 boards from anywhere! As they only cost around $5, it makes it the perfect solution for your Internet of Things projects.

## 3.4 Build Your Own Dashboard to Control your Devices From the Cloud

We are now able to control any board from the cloud using the aREST cloud API. In this section, we are going to see how to create our first online dashboard to control an ESP8266 board from the cloud, using aREST.

You will be able to build your own dashboard to control a LED remotely. We'll also see how to read data from your board and display it inside the dashboard. Let's start!

### 3.4.1 Hardware & Software Requirements

The hardware requirements for this section are really similar to other projects in which we already used the ESP8266 chip.

This is the list of the required hardware for this chapter:

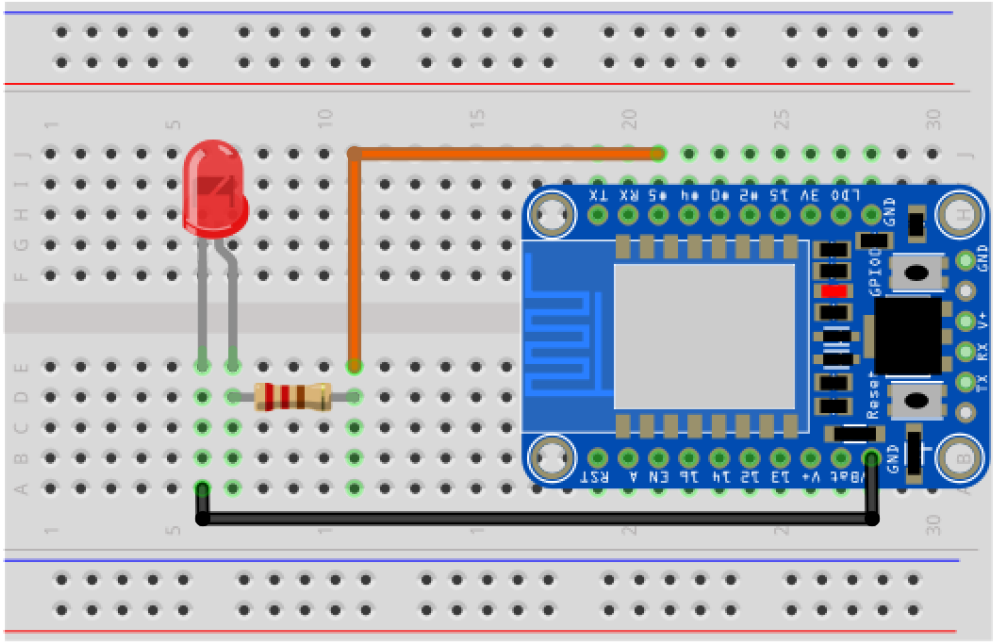
- ESP8266 Huzzah module (https://www.adafruit.com/products/2471)
- FTDI USB converter (https://www.adafruit.com/products/70)
- LED (https://www.sparkfun.com/products/9590)
- 330 Ohm resistor (https://www.sparkfun.com/products/8377)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

For the software part, you will need to instal the PubSubClient library for the Arduino IDE.

You also need to have the ESP8266 board definitions installed inside your Arduino IDE software.

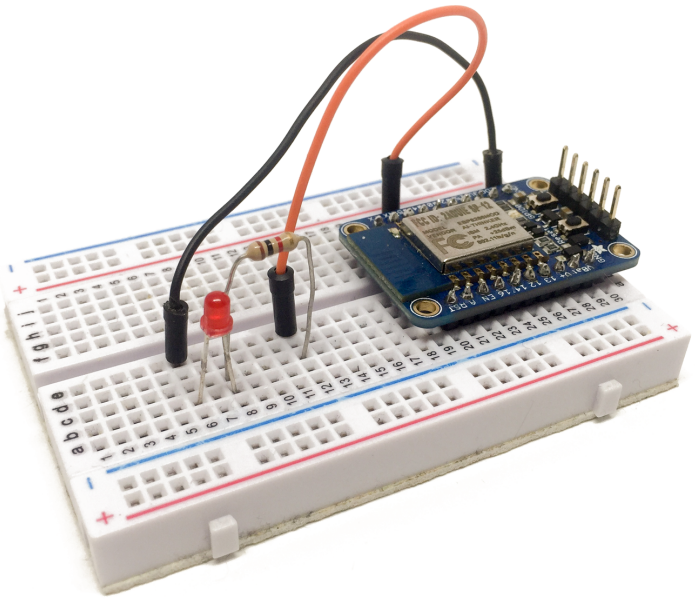
### 3.4.2 Hardware Configuration

Let's now assemble the hardware for this chapter. To help you out, this is the schematic for this chapter:



First, place the ESP8266 board on the breadboard. Then, place the resistor in series with the LED. Connect the resistor to pin number 5 of the ESP8266 board, and the other end of the LED to one GND pin of the ESP8266.

This is the final result:



### 3.4.3 Setting Up Your ESP8266 Board

It's now time to configure the ESP8266 board so it can connect to the aREST cloud server. This is the complete code for this chapter:

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

// Clients
WiFiClient espClient;
PubSubClient client(espClient);

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

// Unique ID to identify the device for cloud.arest.io
```

```
char* device_id = "3g83df";

// WiFi parameters
const char* ssid = "your-wifi-name";
const char* password = "your-wifi-password";

// Variables to be exposed to the API
int temperature;
int humidity;

// Functions
void callback(char* topic, byte* payload, unsigned int length);

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

  // Set callback
  client.setCallback(callback);

  // Init variables and expose them to REST API
  temperature = 24;
  humidity = 40;
  rest.variable("temperature",&temperature);
  rest.variable("humidity",&humidity);

  // Give name and ID to device
  rest.set_id(device_id);
  rest.set_name("esp8266");

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

```
  Serial.println("WiFi connected");

  // Set output topic
  char* out_topic = rest.get_topic();

}

void loop() {

  // Connect to the cloud
  rest.loop(client);

}

// Handles message arrived on subscribed topic(s)
void callback(char* topic, byte* payload, unsigned int length) {

  rest.handle_callback(client, topic, payload, length);

}
```

For this section, I didn't want to complicate things by using a sensor, so I just defined two variables that contains data:

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

In the setup() function, I assigned some arbitrary values to those variables:

```
temperature = 24;
humidity = 40;
rest.variable("temperature",&temperature);
rest.variable("humidity",&humidity);
```

Of course, you could perfectly connect a sensor to your board, and use the readings from this sensor to feed the aREST variables that we declared in the sketch.

As usual, you will also need to change the ID of the device in the sketch, as well as you WiFi name & password.

After that, put the board in bootloader mode, and upload the code to the board. You can quickly check in the Serial monitor that the board is indeed connected to aREST.io.

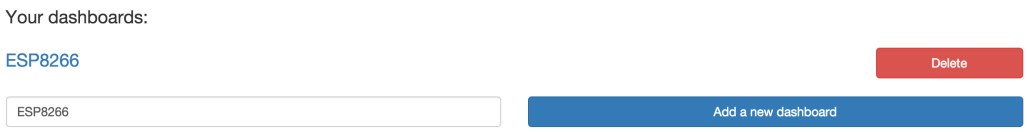
### 3.4.4 Controlling Output

Let's now see how to control the LED from a dashboard that we will create. To do that, we need to introduce a new tool inside the aREST framework: the cloud dashboard service, which is available at:

http://dashboard.arest.io/

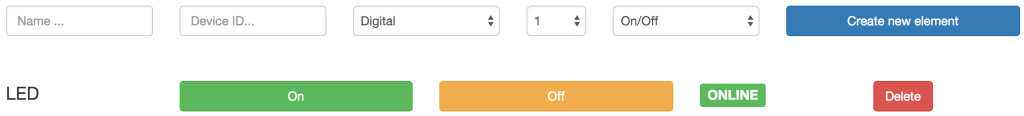
Go to this URL, and create an account. You will instantly be taken to the page where you can create a new dashboard:


After creating a new dashboard, you will see it in the main window:




Click on your newly created dashboard, and create a first element to control the LED on the board. Put a name, the ID of your device, and also assign pin 5 to this dashboard element. This is how it looks like:


Then, add the element by clicking on the **Create** button, and you will see that the element has been added to the dashboard:




Finally, it's time to test your dashboard element! Simply click on the **ON** button: you should immediately see the LED turning on on your board.

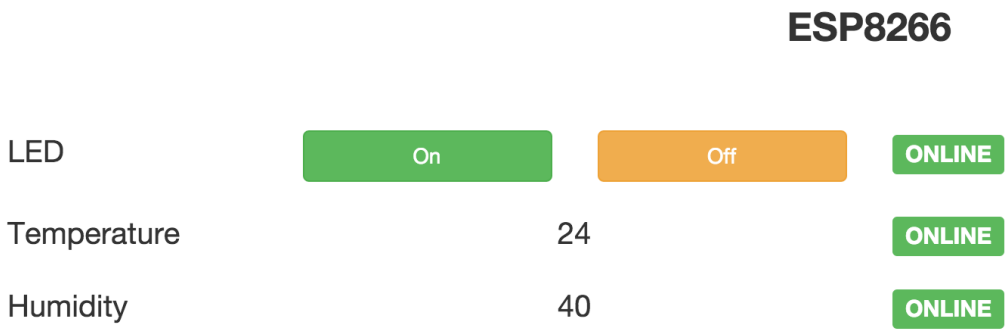
Also note that every time you create a new dashboard element to control a digital pin, this pin is automatically set as an output by the cloud API.

### 3.4.5 Displaying Data

We are now going to move further with the dashboard we created earlier, and display the data that is present on the board inside the dashboard.

Go back to the dashboard, and create a new element with the same device ID, of the *variable* type, linked to the temperature variable:


You will rapidly see the value of the variable displayed in your dashboard. You can now do the same for the temperature variable:




You can now create a dashboard to control a single board, in the cloud, using the aREST framework. You can of course use what you learned in this chapter to control all kind of on/off devices from your dashboard, for example a relay.

You can also use the cloud dashboard service to display temperature, humidity, or measurements coming from other sensors.

## 3.5 Control Several Boards from a Single Dashboard

In this section, we are going to expand on what we learned in the previous project. Instead of controlling a single board, we are going to see how to control several boards from the same online dashboard. This is great if you

want to have several home automation projects based on the Raspberry Pi, all located in different rooms in your home.

We are first going to setup two Raspberry Pi boards, and connect them to the same cloud API we used in the previous chapter. Then, we will control those boards within the same online dashboard. Let's start!

### 3.5.1 Hardware & Software Requirements

We are first going to see what we need for this section. There will be basically two different projects: one Raspberry Pi will be attached to the DHT11 sensor that already used earlier in this book, and the other Raspberry Pi to a PowerSwitch Tail Kit so you can control electrical appliances.

For the Raspberry Pi that will be used to control electrical appliances, this is the list of the components you will need:

- Raspberry Pi (https://www.adafruit.com/product/2358)
- Powerswitch tail kit (https://www.adafruit.com/products/268)
- Breadboard & jumper wires (https://www.adafruit.com/products/64)

For the Raspberry Pi that will be used to measure data, this is the list of the components you will need:

- Raspberry Pi (https://www.adafruit.com/product/2358)
- DHT11 sensor + 4.7k Ohm resistor (https://www.adafruit.com/products/386)
- Breadboard & jumper wires (https://www.adafruit.com/products/64)

Both of the Raspberry Pi boards should of course be configured as we already configured all of them before. This means having Raspbian installed, along with Node.js, and connected to your local network & to the Internet.

### 3.5.2 Hardware Configuration

The hardware configuration for this project is quite simple, as this is something we already saw earlier in the book. There are two projects to configure here, so I will treat them separately.

For the Raspberry Pi that will be used to control electrical appliances, the first step is to connect the pin number 1 of the PowerSwitch to pin number 13 of the Raspberry Pi. Finally, connect the two other pins of the PowerSwitch to a GND pin of the Raspberry Pi.

For the other Raspberry Pi, the DHT sensor signal pin (pin number 2 of the sensor) has to be connected on pin 4 of the Raspberry Pi. Also connect the VCC pin (pin number 1 of the sensor) of the sensor to the RPi 3.3V, and GND to GND. Finally, insert the 4.7K Ohm resistor between pin number 1 and 2.

This is a close-up of the final result:


### 3.5.3 Connecting Your Boards the Cloud Server

Let's now connect both boards to the cloud server. Again, we will use the cloud API from the aREST framework. We'll first take care about the Raspberry Pi that is attached to the DHT11 sensor:

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

```
var sensorLib = require('node-dht-sensor');

// Set ID & name
piREST.set_id('p5dgwt');
piREST.set_name('pi_sensor');

// Connect to cloud.aREST.io
piREST.connect();

// Start server
var server = app.listen(80, function() {
    console.log('Listening on port %d', server.address().port);
});

// Sensor readout
var sensor = {
  initialize: function () {
    return sensorLib.initialize(11, 4);
  },
  read: function () {

    // Read
    var readout = sensorLib.read();
    temperature = readout.temperature.toFixed(2);
    humidity = readout.humidity.toFixed(2);

    // Set variables
    piREST.variable('temperature', temperature);
    piREST.variable('humidity', humidity);
    console.log('Temperature: ' + temperature + 'C, ' +
        'humidity: ' + humidity + '%');

    // Repeat
    setTimeout(function () {
        sensor.read();
    }, 2000);
  }
};
```

```
// Init sensor
if (sensor.initialize()) {
  sensor.read();
} else {
  console.warn('Failed to initialize sensor');
}
```

As for the previous chapter, make sure to give an unique ID to the board. Also, make sure to give a name to the board here, it will be used in the dashboard to identify which boards is connected to the dashboard's functions.

It's now time to connect this Raspberry Pi to the cloud server. Put all the code into a file (named here cloud.js), and inside the folder where the file is located type:

```
sudo npm install pi-arest express node-dht-sensor
```

This will take a while, especially on the first versions of the Raspberry Pi. After that, type:

```
sudo node cloud.js
```

This will connect the Raspberry Pi to the cloud API, and you should see the confirmation in the console.

Let's now deal with the Raspberry Pi that will control an electrical appliance. This is the code for this Pi:

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

// Set ID & name
piREST.set_id('r6hwcy');
piREST.set_name('pi_lamp');
```

```
// Connect to cloud.aREST.io
piREST.connect();

// Start server
var server = app.listen(80, function() {
    console.log('Listening on port %d', server.address().port);
});
```

As with the other Pi, make sure to modify the ID & name in the code.

Again, we will connect this Pi to the cloud API. Put all the code into a file (named here cloud.js), and inside the folder where the file is located type:

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

This will take a while, especially on the first versions of the Raspberry Pi. After that, type:

```
sudo node cloud.js
```

This will connect the Raspberry Pi to the cloud API, and you should see the confirmation in the console.

Both our Raspberry Pi boards are now connected to the cloud API. You can test that a board is actually connected by typing the ID in the board after the URL of the cloud API server. For example:

```
https://cloud.arest.io/p5dgwt
```

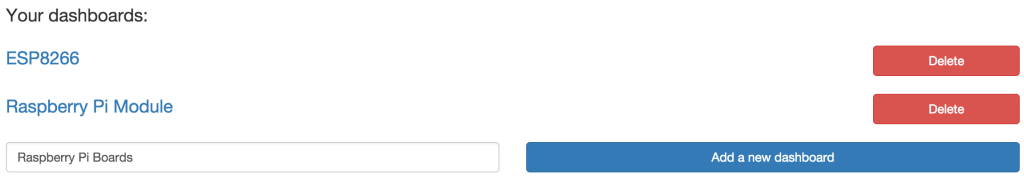
If you can see the device answering with it's name and ID, it means that it is currently online.

### 3.5.4 Adding all Boards to a Cloud Dashboard

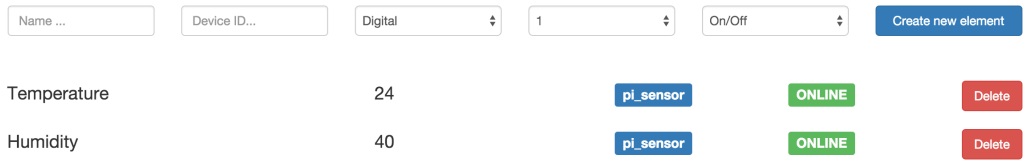
We are now going to see how to control all those boards using a cloud dashboard. We will use the same aREST dashboard service that we used in the previous section. Just head over to:

http://dashboard.arest.io/

After creating an account there, add a new dashboard:

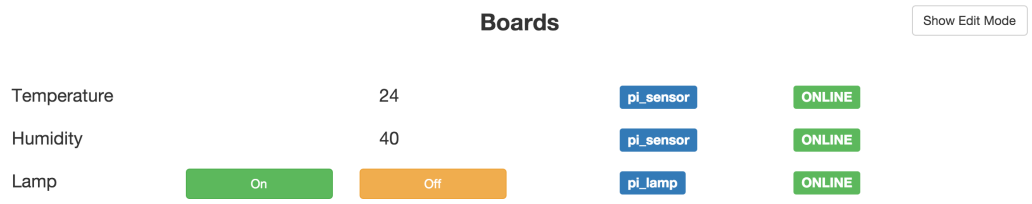



From there, you will be able to access all the variables from your Pi board and control them remotely. First, add the temperature & humidity indicators from the Raspberry board with the DHT sensor:




As you can see, the name of the device will appear on the dashboard, so you can easily identify to which device the data corresponds.

Finally, add an On/Off element for the second Raspberry Pi. This is is the final result inside the dashboard:




Of course, feel free to try the button for example. Any electrical appliance connected to the second Raspberry Pi should immediately turn on. Congratulations, you can now control several devices from your a cloud dashboard, from anywhere in the world!

In this section, we integrated several Raspberry Pi boards into a single dashboard, so you can control your home automation projects from anywhere. Of course, you could perfectly add more boards to the project, and control them all from a single interface. And you can of course pack several sensors and controllers on every board as well!

## 3.6 Deploy your own aREST cloud server

So far in this chapter, we saw how to connect your devices to the aREST cloud server. This is a very convenient way to instantly connect any device to the cloud and access it from anywhere.

However, there are cases where you want to use your own server, for safety reasons for example. Luckily for you, aREST is completely open-source, and you can actually use the code on your own cloud server.

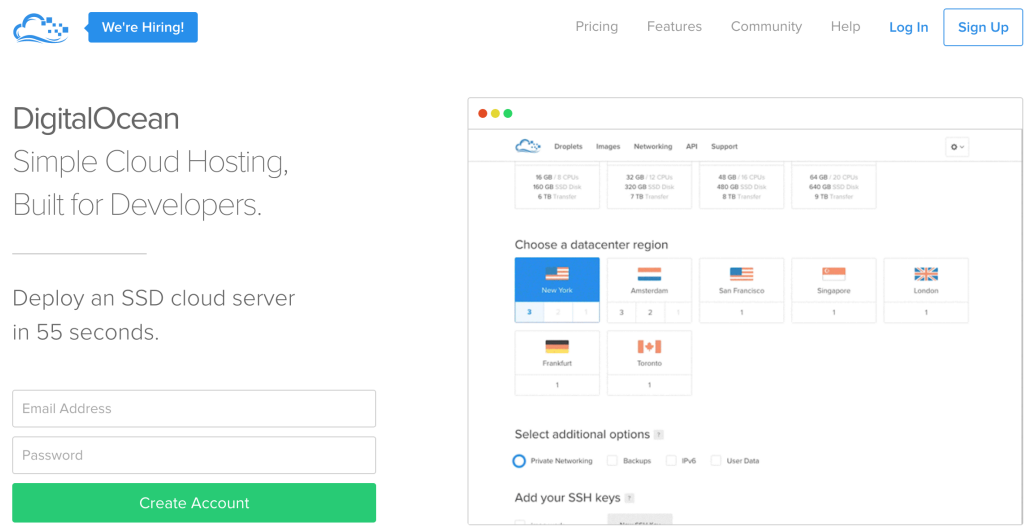
In this section, we are going to see how to deploy your own aREST-based cloud server and connect a device to it.

### 3.6.1 Create & Configure Your Digital Ocean Account

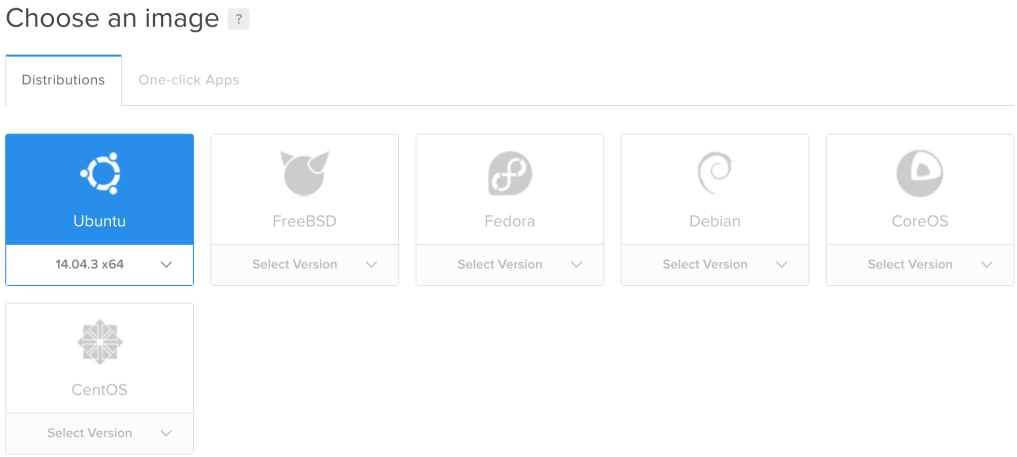
The first step for this section is to have somewhere where you can deploy your server code. I will use Digital Ocean for that, which is a very convenient and cheap service to create a server in the cloud. You can go create an account at:

https://www.digitalocean.com/

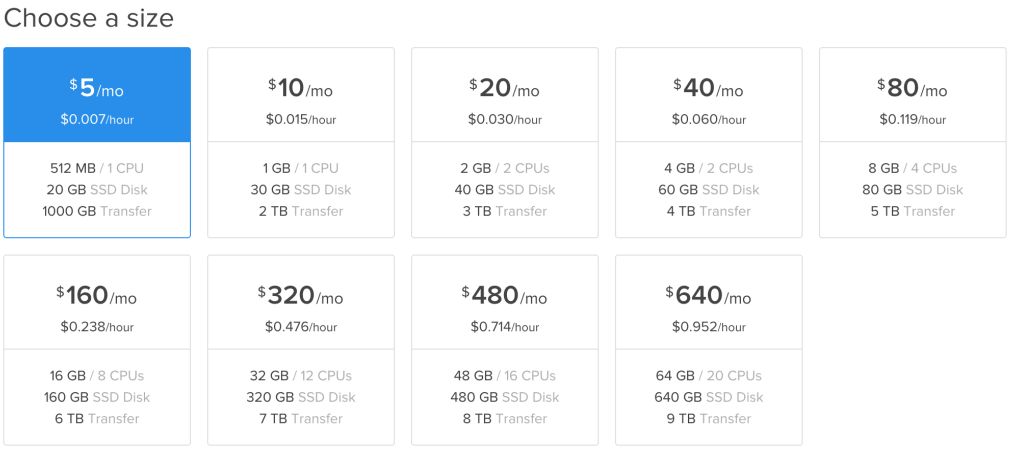
Once this is done, create a Droplet, which is a server instance in the cloud:



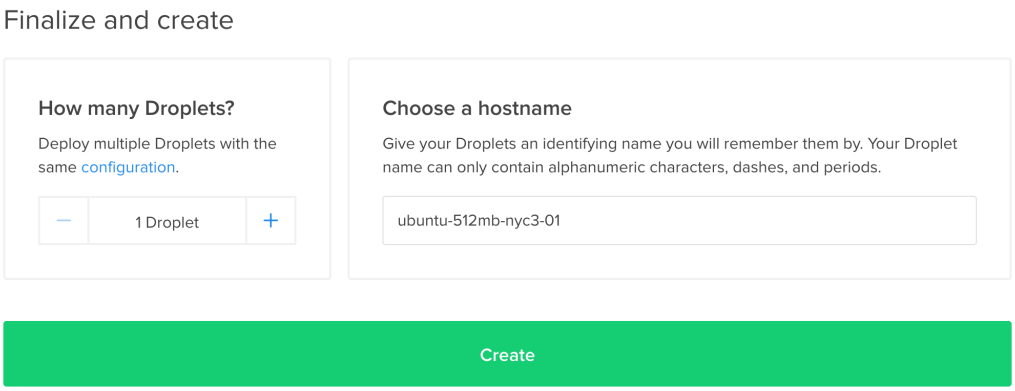

As the image, choose 'Ubuntu':



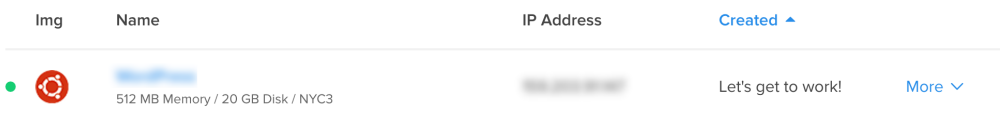

You can choose the cheapest plan, as we won't need a lot of computing power:




Finally, create the Droplet:




You will see the Droplet in the list of Droplets on your account, along with the IP address of the Droplet:




You will need this IP address in a moment. Now, you should also have received a password via email. However, I recommend setting up SSH keys on your computer, to access your server without password. You can learn how to do it using the tutorial found on this page:

https://www.digitalocean.com/community/tutorials/how-to-set-up-ssh-keys–2

### 3.6.2 Deploy Your Cloud Server

We are now going to deploy the aREST code on your freshly created cloud server. First, grab the code from:

https://github.com/marcoschwartz/meteor-arest-mqtt

Put the code in a folder on your computer, and type:

```
meteor create .
```

This will initialise a new Meteor project. Then, install the required Meteor modules with:

```
meteor add percolate:synced-cron iron:router http
```

Then, you need to install all the required NPM packages for the project to work. You can do this by typing:

```
sudo meteor npm install
```

After that, simply test the project with:

```
meteor
```

If everything goes well, stop the local Meteor server. We are now going to install a tool called Meteor Up to easily deploy our project on your Digital Ocean server. For that, type:

```
npm install -g mupx
```

Followed by:

```
mupx init
```

This will create several files in your folder, including a file called mup.json. Open this file, and paste the following code:

```
{
  // Server authentication info
  "servers": [
    {
      "host": "159.203.92.147",
      "username": "root",
      //"password": "password"
      // or pem file (ssh based authentication)
      "pem": "~/.ssh/id_rsa"
    }
  ],

  // Install MongoDB in the server,
  // does not destroy local MongoDB on future setup
  "setupMongo": true,

  // WARNING: Node.js is required! Only skip
  // if you already have Node.js installed on server.
  "setupNode": true,

  // WARNING: If nodeVersion omitted will setup 0.10.36 by default.
  // Do not use v, only version number.
  "nodeVersion": "0.10.43",

  // Install PhantomJS in the server
  "setupPhantom": true,

  // Show a progress bar during the upload of the bundle to the server.
  // Might cause an error in some rare cases if set to true,
  // for instance in Shippable CI
  "enableUploadProgressBar": true,

  // Application name (No spaces)
  "appName": "arest",

  // Location of app (local directory)
  "app": ".",
```

```
  // Configure environment
  "env": {
    "ROOT_URL": "http://localhost",
    "PORT": 3000
  },

  // Meteor Up checks if the app comes online just after the deployment
  // before mup checks that, it will wait for no. of seconds configured below
  "deployCheckWaitTime": 15
}
```

The only thing you need to modify here is the IP address of your Droplet, and also insert your password if you are using the password method to access your server.

Then, you can configure the server with:

```
mupx setup
```

When this is done, deploy your application with:

```
mupx deploy
```

You should see the following in the terminal:

```
[159.203.92.147] - Uploading bundle
[159.203.92.147] - Uploading bundle: SUCCESS
[159.203.92.147] - Setting up Environment Variables
[159.203.92.147] - Setting up Environment Variables: SUCCESS
[159.203.92.147] - Invoking deployment process
[159.203.92.147] - Invoking deployment process: SUCCESS
```

Once this is done, congratulations, your deployed your own aREST server on the cloud!

### 3.6.3 Connect a Device to Your Cloud Server

This is great, but it won't be finished unless we actually connect a device to the server. Let's see how to do that. We are again going to use an ESP8266 board for this example.

This is the complete code for this part:

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

// Clients
WiFiClient espClient;
PubSubClient client(espClient);

// Create aREST instance
aREST rest = aREST(client, "159.203.92.147");

// Unique ID to identify the device for cloud.arest.io
char* device_id = "9u2co4";

// WiFi parameters
const char* ssid = "wifi-name";
const char* password = "wifi-password";

// Declare functions to be exposed to the API
int ledControl(String command);

// Functions
void callback(char* topic, byte* payload, unsigned int length);

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

```
  // Set callback
  client.setCallback(callback);

  // Give name and ID to device
  rest.set_id(device_id);
  rest.set_name("esp8266");

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

  // Set output topic
  char* out_topic = rest.get_topic();

}

void loop() {

  // Connect to the cloud
  rest.loop(client);

}

// Handles message arrived on subscribed topic(s)
void callback(char* topic, byte* payload, unsigned int length) {

  rest.handle_callback(client, topic, payload, length);

}
```

As you can see, the code is really similar to the one we saw earlier in this chapter. The only difference is this part:

```
aREST rest = aREST(client, "159.203.92.147");
```

As you can see, you need to pass the IP address of the server as a second argument. This will indicate to aREST to connect to your own server and not the aREST cloud server.

Now, configure the board with the code we just saw, by making sure to insert the IP address of your server, and also to enter your WiFi name and password.

Then, upload the code to the board. You can now check the connection with your own server by going to the IP address of the Droplet, followed by port 3000. For example:

```
http://159.203.92.147:3000/9u2co4/id
```

You should receive the confirmation from the server:

```
{
  "id": "9u2co4",
  "name": "esp8266",
  "connected": true
}
```

Congratulations, you just connected a board to your own cloud server! You can now do everything we did so far in this chapter, but this time by using your own server. You can now also buy a domain name if you wish, and redirect it to the IP address of your Droplet.

## 3.7 Example project: Remote Data Monitoring from Multiple Boards

As an example for this chapter, we are going to see how to deploy a remote data monitoring system based on the ESP8266 module. We will see how to configure the boards, and then monitor them within a single cloud dashboard.

This is great for real-life applications, like the monitoring of a remote house or cabin, where you need to monitor data at several places in the location you want to monitor.

### 3.7.1 Hardware & Software Requirements

For this example project, you will need components that we already used in previous sections of this chapter, like an ESP8266 module, a DHT sensor, and breadboard & jumper wires.

This is the list of the required components for one module:

- ESP8266 Huzzah module (https://www.adafruit.com/products/2471)
- FTDI USB converter (https://www.adafruit.com/products/70)
- DHT11 sensor (https://www.adafruit.com/products/386)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

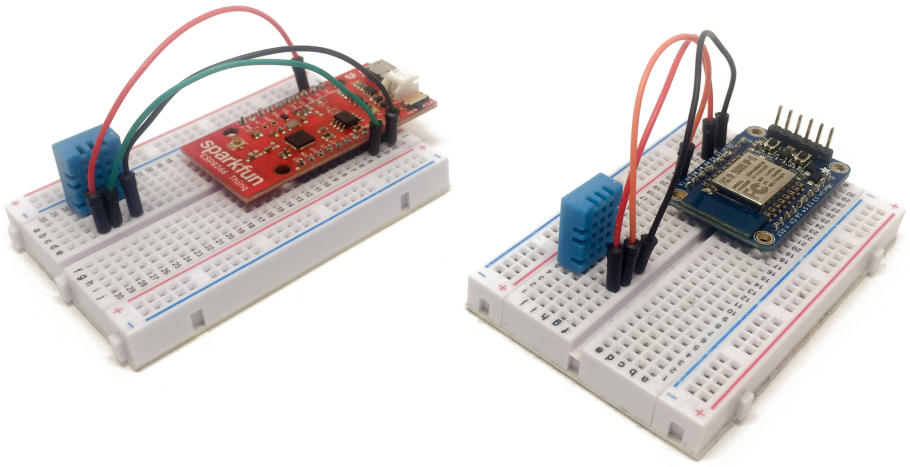
I used two modules for this example, but you can of course use as many as you wish.

### 3.7.2 Hardware Configuration

Let's now see how to configure a given module. First, place the ESP8266 module on the breadboard, and also place the DHT11 sensor on it.

The DHT sensor signal pin (pin number 2 of the sensor) has to be connected on pin 5 of the ESP8266. Also connect the VCC pin (pin number 1 of the sensor) of the sensor to the ESP8266 3.3V pin, and GND to GND.

This is the final result, using two monitoring modules based on the ESP8266:



### 3.7.3 Configuring the Boards

Let's now configure our modules. This is the complete code for this part:

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

// DHT11 sensor pins
#define DHTPIN 5
#define DHTTYPE DHT11

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

// Clients
WiFiClient espClient;
PubSubClient client(espClient);
```

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

// Unique ID to identify the device for cloud.arest.io
char* device_id = "9u2co4";

// WiFi parameters
const char* ssid = "your-wifi-name";
const char* password = "your-wifi-password";

// Variables to be exposed to the API
float temperature;
float humidity;

// Functions
void callback(char* topic, byte* payload, unsigned int length);

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

  // Init DHT
  dht.begin();

  // Set callback
  client.setCallback(callback);

  // Init variables and expose them to REST API
  rest.variable("temperature",&temperature);
  rest.variable("humidity",&humidity);

  // Give name and ID to device
  rest.set_id(device_id);
  rest.set_name("sensor2");

  // Connect to WiFi
  WiFi.begin(ssid, password);
```

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

  // Set output topic
  char* out_topic = rest.get_topic();

}

void loop() {

  // Reading temperature and humidity
  humidity = dht.readHumidity();
  temperature = dht.readTemperature();

  // Connect to the cloud
  rest.loop(client);

}

// Handles message arrived on subscribed topic(s)
void callback(char* topic, byte* payload, unsigned int length) {

  rest.handle_callback(client, topic, payload, length);

}
```

There are some things you will need to change here before uploading the code. For each module, you need to set your own WiFi network name and password. Note that you could perfectly use different WiFi networks for different modules: as they will all connect to the same cloud server, this won't be an issue.

Then, you need to give an unique ID to every module, defined by the line:

```
char* device_id = "9u2co4";
```

Now, upload the code to each module. You can also verify that each module is connected to the aREST cloud server by checking the Serial monitor or by sending them the 'id' command via the aREST cloud server.

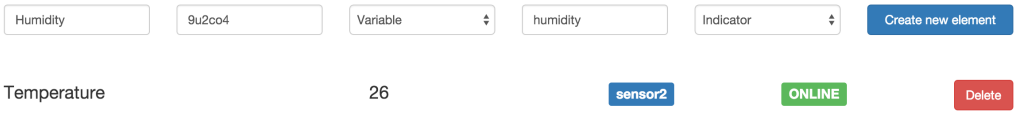
### 3.7.4 Monitoring Data from a Dashboard

We are now going to create a new dashboard from which you will be able to constantly monitor your boards. First, if that's not done create a dashboard at:

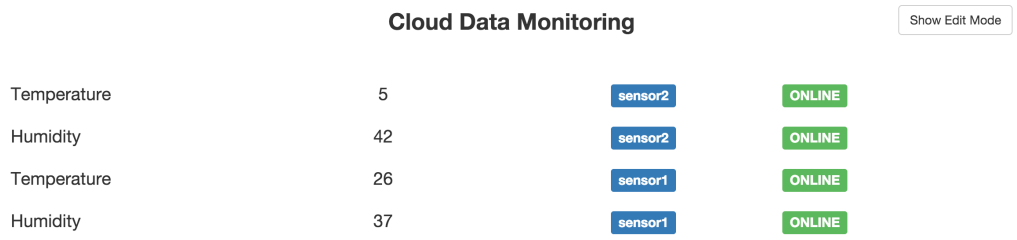
http://dashboard.arest.io/

Then, create a new dashboard:


For each module, create two entries, one for the temperature, and one for the humidity:




This will be the final result in your online cloud dashboard:




You can now monitor several boards from anywhere in the world, allowing you to monitor the complete status of your home or a cabin while you are away! Of course, you can add other sensors to each module to monitor even more data using aREST.

## 3.8 How to Go Further

In this chapter, we saw how to use the cloud capabilities of the aREST framework. We learned how to control devices from anywhere in the world, wether it is an Arduino board, a Raspberry Pi, or ESP8266 boards. We also saw how to create online dashboards to control and monitor your projects from anywhere.

You can now apply this to all your projects that require to be accessible from anywhere in the world. It can be home automation projects like the one we built in this chapter, but you can also think about projects in the industrial space. I heard for example from someone monitoring their farm remotely using the aREST framework, even as they were living 10 kms away! You can also imagine controlling a mobile robot remotely, from anywhere in the world.

In the next chapter, we are going to see another important part of the aREST framework: how to build mobile applications based on the aREST framework.

# 4 Build Mobile Applications using aREST

In the last chapter of this book, we are going to learn how to build mobile applications to control your aREST projects. You will learn how to easily build applications for Android and iOS, so you can control your aREST projects from your mobile devices.

## 4.1 Build Mobile Applications for Android

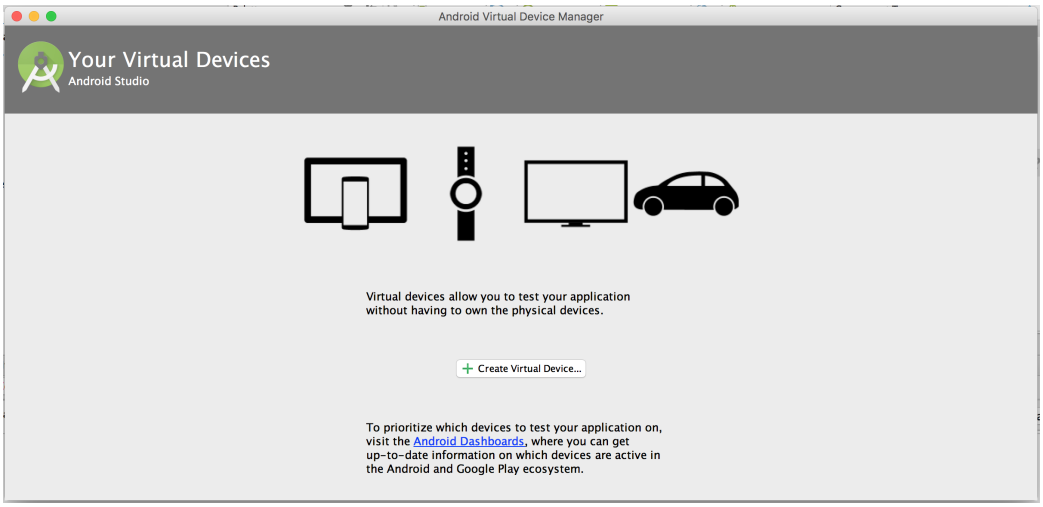
In the first section of this chapter, we are going to see how to build a native application for the Android platform. For that, we are again going to use Meteor, which is a framework that we already saw earlier in this book. We will see that from there, it is really easy to build an Android application.

As for the hardware, you can basically use any project you will found in this book. I used a simple ESP8266 board, configured with a basic aREST sketch, and with a LED connected to pin 5. You can refer to previous chapter if you need instructions on how to build the hardware.

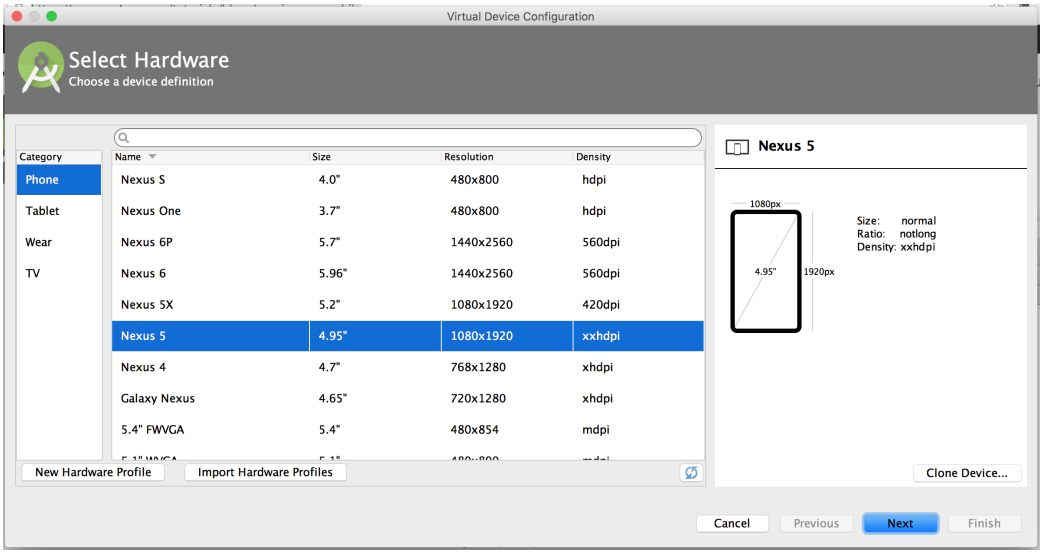
But first, we need to install several tools so you can build Android applications from Meteor. The easiest way to do that is to install Android Studio:

http://developer.android.com/sdk/index.html

Once that's done, open it, and look for the Android Virtual Device Managed (AVD), and open it:

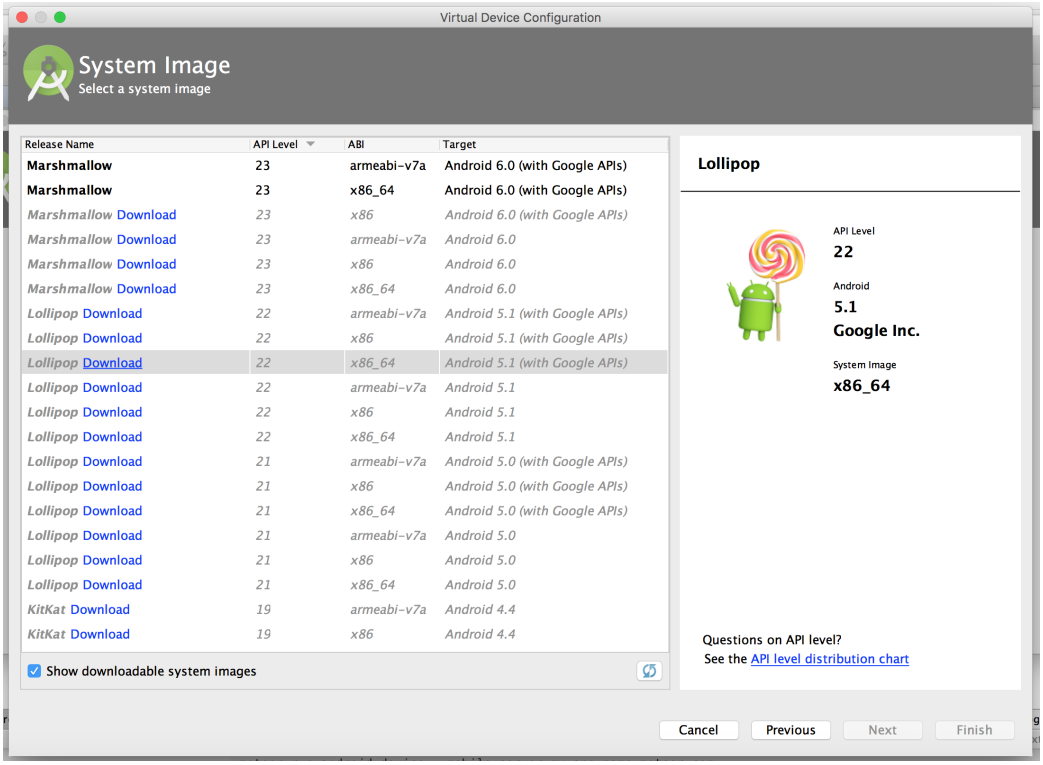



From there, we will be able to create a virtual device on which we will run the application. From the menu, select the default device:

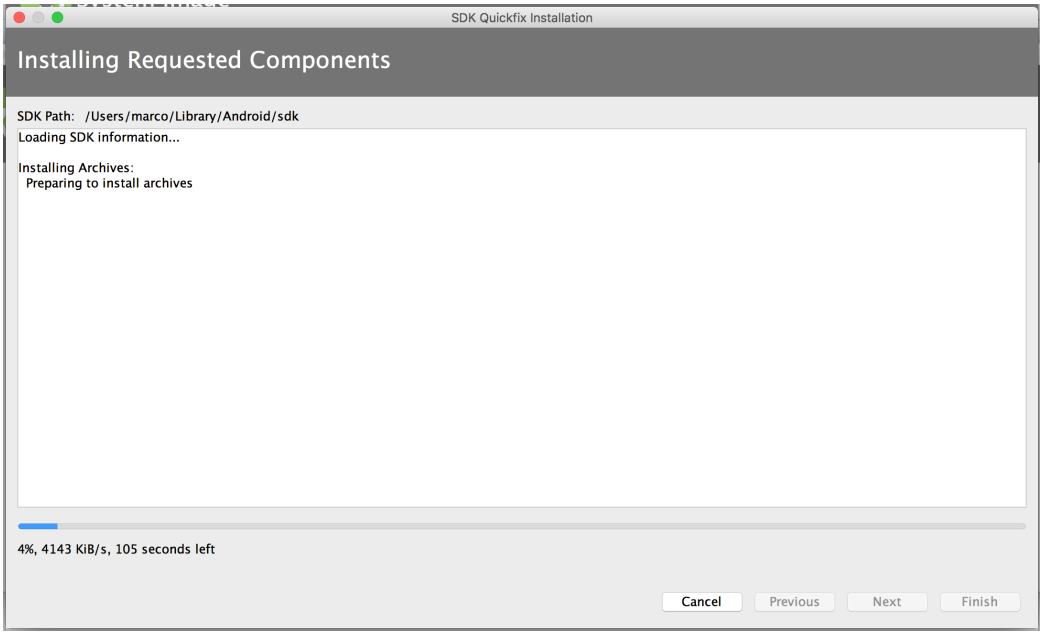



Then, in the next menu, you will need to select the x86_64 image for Android

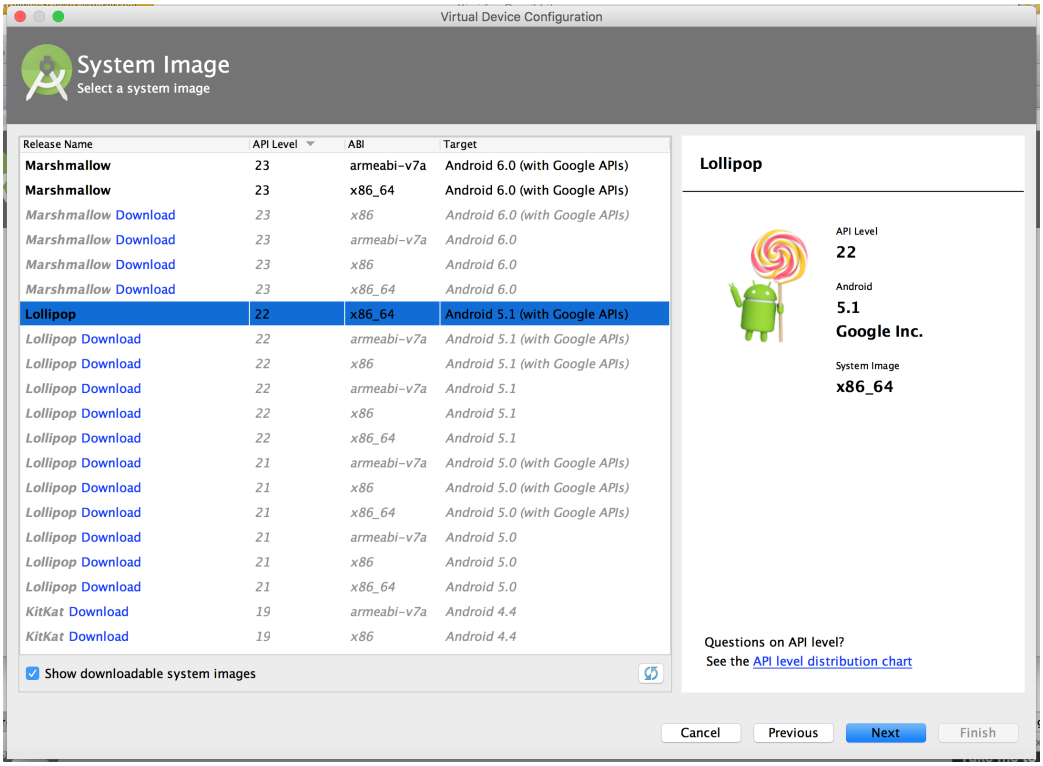
5.1. Indeed, at the time this book was written, Meteor didn't support Android 6.0. Click on the Download button:



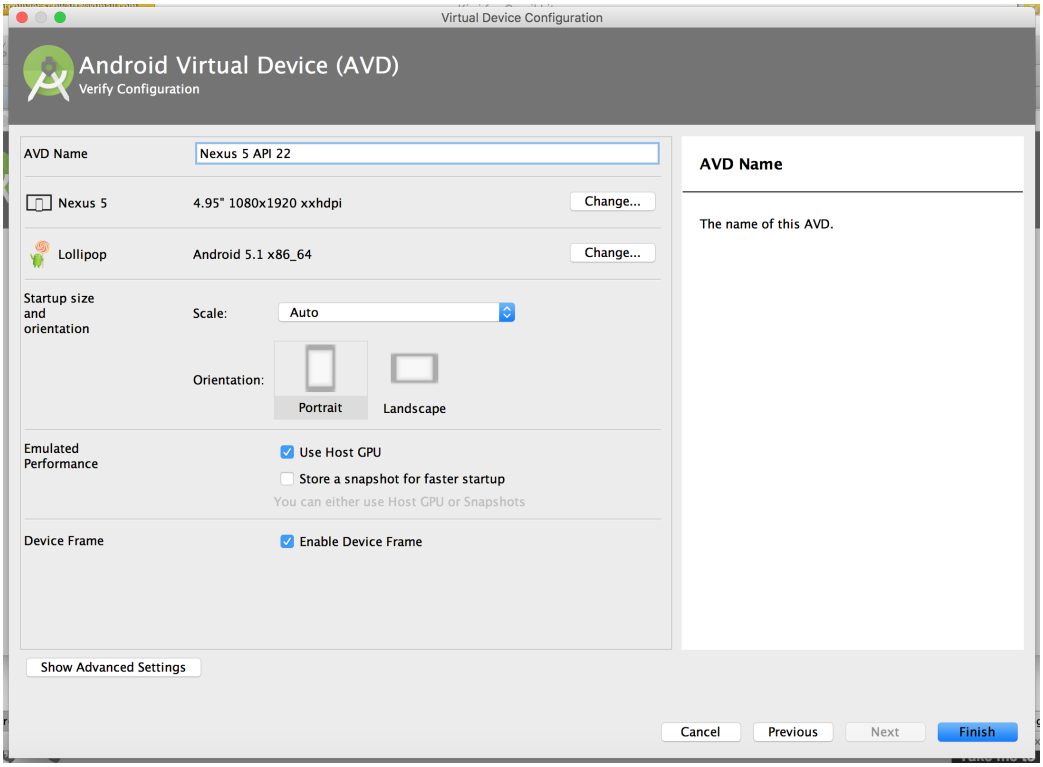

Now, wait for the package to be installed:



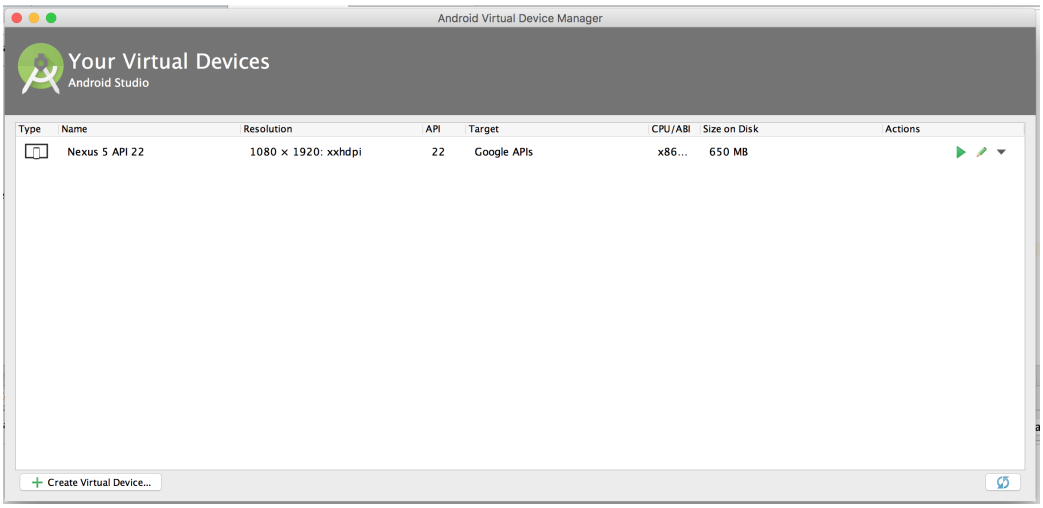

After that, select the image you just downloaded:




Finally, validate the creation of the device:




You should the virtual device inside the virtual devices manager:




Now, we are going to see the code for the application we will deploy on the Android device.

First, the interface. We will simply use an interface with two buttons to turn the LED on or off:

```
<head>
  <title>Android</title>
  <meta name="viewport" content="width=device-width,
    initial-scale=1, maximum-scale=1, user-scalable=no">
</head>

<body>
  <h1>ESP8266 Android Control</h1>

  {{> home}}
</body>

<template name="home">
  <div class='container'>
    <div class='row'>
      <div class='col-md-3'>
        <button id='on' class='btn btn-block btn-primary'>
          On
        </button></div>
      <div class='col-md-3'>
        <button id='off' class='btn btn-block btn-warning'>
          Off
      </button></div>
    </div>
  </div>
</template>
```

Note here that we inserted this line, that we didn't see earlier in this book:

```
<meta name="viewport" content="width=device-width,
  initial-scale=1, maximum-scale=1, user-scalable=no">
```

This will allow the interface to be perfectly scaled for your mobile device.

Then, let's see the JavaScript code for the client:

```
if (Meteor.isClient) {

  Template.home.rendered = function() {

    // Pin mode
    Meteor.call('pinMode', "192.168.115.105", 5, 'o');

  }

  // Events
  Template.home.events({
    'click #on': function() {
      Meteor.call('digitalWrite', "192.168.115.105", 5, 1);
    },
    'click #off': function() {
      Meteor.call('digitalWrite', "192.168.115.105", 5, 0);
    }
  });
}
```

As you can see, we simply call Meteor methods at each click of a button.

As the Meteor aREST module doesn't support mobile applications yet, we need to define those methods in the code:

```
if (Meteor.isServer) {

  Meteor.methods({

    digitalWrite: function(ipAddress, pin, state) {
      HTTP.get('http://' + ipAddress + '/digital/' + pin + '/' + state);
    },
    pinMode: function(ipAddress, pin, state) {
      HTTP.get('http://' + ipAddress + '/mode/' + pin + '/' + state);
    }

  });

}
```

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

https://github.com/marcoschwartz/discover-arest

Then, navigate to the folder where the code is located, and type:

```
meteor create .
```

Next, install the required modules with:

```
meteor add twbs:bootstrap http
```

Now, add the Android platform using:

```
meteor add-platform android
```

Finally, run the project on the Android virtual device with:

```
meteor run android
```

This will open the Android virtual device emulator, from which you will be able to use the interface. Try to press on a button: it should instantly turn the LED on or off.

Of course, you can also deploy your application on your own Android device! Simply make sure your phone is connected to your computer, and type:

```
meteor run android-device
```

Congratulations, you can now build Android applications using aREST!

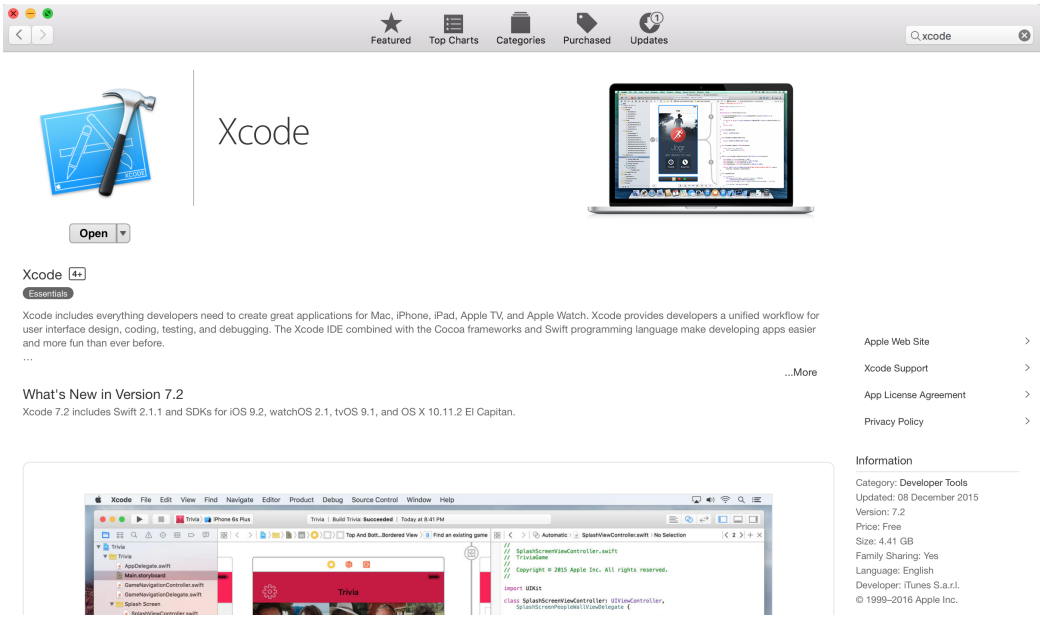
## 4.2 Build Mobile Applications for iOS

In this section, we are going to build a mobile application to allow to control your aREST devices using an iOS device.

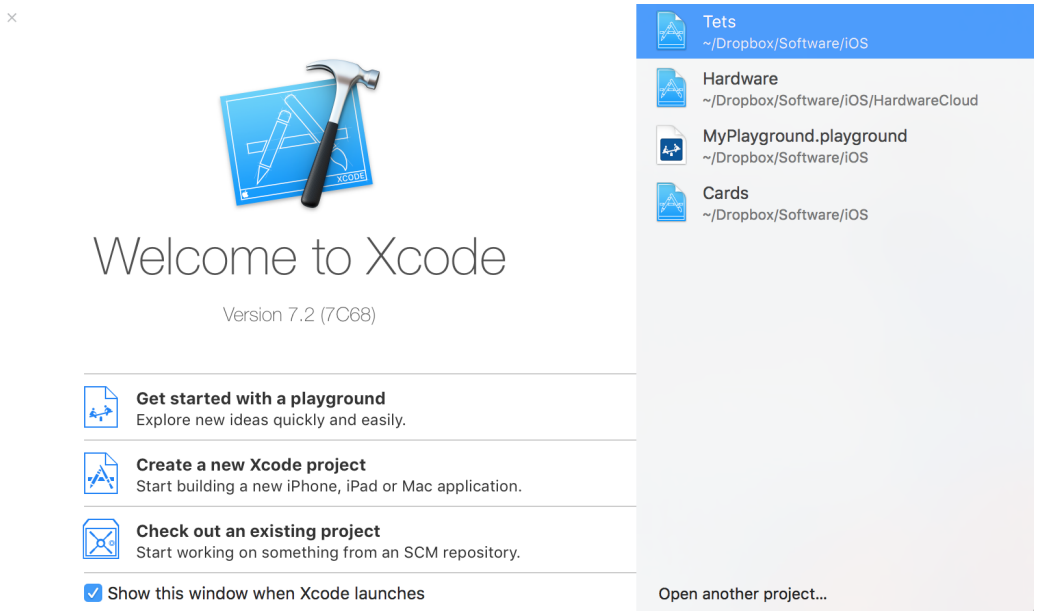
Note that this section is only for OS X users. Otherwise, you won't be able to follow the instructions of this section.

As for the hardware, you can basically use any project you will found in this book. I used a simple ESP8266 board, configured with a basic aREST sketch, and with a LED connected to pin 5. You can refer to previous chapters if you need instructions on how to build the hardware.

The first step is to install Xcode, using the Mac App Store:




Then, you need to open Xcode at least once, so you can install the required components and to accept the Xcode terms. The procedure is over once you get to the Xcode main screen:




Once this is done, you can close Xcode.

Let's now see the code for this section. First, the interface. We will simply use an interface with two buttons to turn the LED on or off:

```
<head>
  <title>iOS</title>
  <meta name="viewport" content="width=device-width,
    initial-scale=1, maximum-scale=1, user-scalable=no">
</head>

<body>
  <h1>ESP8266 iOS Control</h1>

  {{> home}}
</body>

<template name="home">
  <div class='container'>
    <div class='row'>
      <div class='col-md-3'>
        <button id='on' class='btn btn-block btn-primary'>
```

```
          On
        </button></div>
      <div class='col-md-3'>
        <button id='off' class='btn btn-block btn-warning'>
          Off
        </button></div>
    </div>
  </div>
</template>
```

Note here that we inserted this line, that we didn't see earlier in this book:

```
<meta
  name="viewport"
  content="width=device-width,
    initial-scale=1,
    maximum-scale=1,
    user-scalable=no">
```

This will allow the interface to be perfectly scaled for your mobile device.

Then, let's see the JavaScript code for the client:

```
if (Meteor.isClient) {

  Template.home.rendered = function() {

    // Pin mode
    Meteor.call('pinMode', "192.168.115.105", 5, 'o');

  }

  // Events
  Template.home.events({
    'click #on': function() {
      Meteor.call('digitalWrite', "192.168.115.105", 5, 1);
    },
```

```
    'click #off': function() {
      Meteor.call('digitalWrite', "192.168.115.105", 5, 0);
    }
  });
}
```

As you can see, we simply call Meteor methods at each click of a button.

As the Meteor aREST module doesn't support mobile applications yet, we need to define those methods in the code:

```
if (Meteor.isServer) {

  Meteor.methods({

    digitalWrite: function(ipAddress, pin, state) {
      HTTP.get('http://' + ipAddress + '/digital/' + pin + '/' + state);
    },
    pinMode: function(ipAddress, pin, state) {
      HTTP.get('http://' + ipAddress + '/mode/' + pin + '/' + state);
    }

  });

}
```

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

https://github.com/marcoschwartz/discover-arest

Make sure to enter the IP address of the board you want to control inside the Meteor code.

Then, navigate to the folder where the code is located, and type:

```
meteor create .
```

After that, install the required modules for the project:

```
meteor add twbs:bootstrap http
```

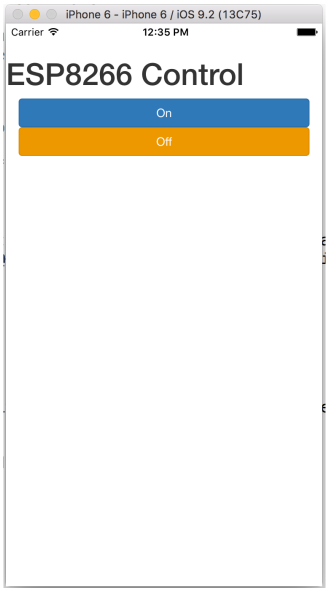
Then, add the iOS platform with:

```
meteor add-platform ios
```

You are now ready to test the project! You can simply start it by typing:

```
meteor run ios
```

After a while, the iOS emulator will start, and your application will be loaded:




You can use it by pressing one of the buttons: the LED connected to the ESP8266 board should immediately respond.

If you now want to deploy this application on your iOS device, you will need to have a paid Apple iOS Developer Account.

If that's the case, simply make sure your device is connected to your computer, and type:

```
meteor run ios-device
```

After a while, you will have your application ready to be used on your iOS device. Congratulations, you can now build iOS applications to control aREST devices!

## 4.3 Example Project: Control Your Home From Your Smartphone

For this last example project of the book, we are going to continue build mobile applications to control aREST projects, and this time we are going to build an application that could be used to control a smart home.

In your home, you could want to control a lamp for example, along with wanting to know the current temperature and humidity in a room. This is exactly what we are going to do in this project.

For the hardware, we are going to re-use a project that we built in previous chapters. The hardware will based on an ESP8266 running aREST, along with a PowerSwitch Tail (connected to a lamp) and a DHT11 sensor. I invite you to go back to the previous chapters to learn more about how to build those projects.

I will build this application for iOS as I am an iOS and OS X user, but you could perfectly build it for Android as well.

It always start by coding the interface:

```
<template name="home">

  <div class='row'>
    <div class='col-md-3'>
      <button id='on' class='btn btn-block btn-primary'>On</button>
    </div>
    <div class='col-md-3'>
      <button id='off' class='btn btn-block btn-warning'>Off</button>
    </div>
  </div>

  <div class='row'>
    <div class='col-md-3'>
      Temperature: <span id='temperature'></span> C
    </div>
    <div class='col-md-3'>
      Humidity: <span id='humidity'></span>%
    </div>
```

```
  </div>

</template>
```

Compared to previous projects in this chapter, you can see that we added indicators for temperature & humidity.

Therefore, we will need to update this data inside the code for the client:

```
if (Meteor.isClient) {

  // Events
  Template.home.events({
    'click #on': function() {
      Meteor.call('digitalWrite', "192.168.115.105", 5, 1);
    },
    'click #off': function() {
      Meteor.call('digitalWrite', "192.168.115.105", 5, 0);
    }
  });

  Template.home.rendered = function() {

    // Pin mode
    Meteor.call('pinMode', "192.168.115.105", 5, 'o');

    // Variables
    Meteor.call('variable', "192.168.115.105",
      'temperature', function (err, data) {
      $('#temperature').text(data.temperature);
    });

    Meteor.call('variable', "192.168.115.105",
      'humidity', function (err, data) {
      $('#humidity').text(data.humidity);
    });

  }
```

```
}
```

Of course, you will need to change the IP address for the one of the board you are using.

Then, for the server code, we simply define all the function we need to communicate with the aREST API running on the board:

```
if (Meteor.isServer) {

  Meteor.methods({

    digitalWrite: function(ipAddress, pin, state) {
      HTTP.get('http://' + ipAddress + '/digital/' + pin + '/' + state);
    },
    pinMode: function(ipAddress, pin, state) {
      HTTP.get('http://' + ipAddress + '/mode/' + pin + '/' + state);
    },
    variable: function(ipAddress, variable) {
      var answer = HTTP.get('http://' + ipAddress + '/' + variable);
      return answer.data;
    }

  });

}
```

Now, grab all the files for this project, and navigate to the folder with a terminal. Then, initialise a new Meteor project with:

```
meteor create .
```

After that, install the required components:

```
meteor add twbs:bootstrap http
```

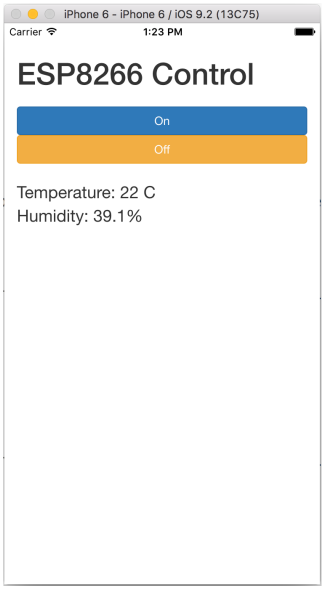
Then, add the iOS platform:

```
meteor add-platform ios
```

Then, run the application on the virtual iOS device with:

```
meteor run ios
```

You should see the following application inside the emulator:




As you can see, the measurement from the sensor immediately appeared inside the application. You can also deploy it on your own iOS device (if you are part of the iOS developer program) with:

```
meteor run ios-device
```

You now have all the required knowledge to build an application to control your smart home! You can for example add more devices, and control them all within the same application.

## 4.4 How to Go Further

In this chapter, we saw how to build applications for Android & iOS, to control your projects running aREST. We saw that's it is really easy to do using the Meteor framework, that can be used to automatically generate mobile applications from existing Meteor code.

You can literally take projects that you built earlier in the second chapter of this book, and use them as applications on your mobile device. Of course, you can also use this chapter to build applications in other fields than home automation, for example to control a mobile robot from your phone.

# Conclusion

This is already the end of this book, and I really hope you enjoyed learning about the aREST framework! Before I give you some parting advices, let's summarize what we learned in this book.

In the first part of the book, we saw the basics of the aREST framework, and you learned how to control individual boards using aREST. In the second part of the book, we saw how to build graphical interfaces & server-side applications to control your aREST projects.

In the third part of the book, we learned how to control your projects from anywhere in the world. Finally, in the last part of the book you learned how to build mobile applications to control your aREST projects.

You now have all the tools to build your own projects using the power of the aREST framework. There are limitless possibilities, but here are three project ideas that could use the aREST framework:

1. Inexpensive remote monitoring and control of a mountain cabin using the aREST cloud platform and ESP8266 boards
2. Control of a mobile robot based on the Raspberry Pi using the aREST meteor application and a nice graphical interface, using the Raspberry Pi camera as the robot's eyes
3. Web-connected healthcare sensors based on the ESP8266, that send results in real-time to your own aREST cloud server

If you create a project using the aREST framework and you would like to share it with the community, or if you have feedback about this book or aREST itself, don't hesitate to send me an email at:

contact@arest.io

Thanks again for reading this book, and have fun creating amazing projects with the aREST framework!