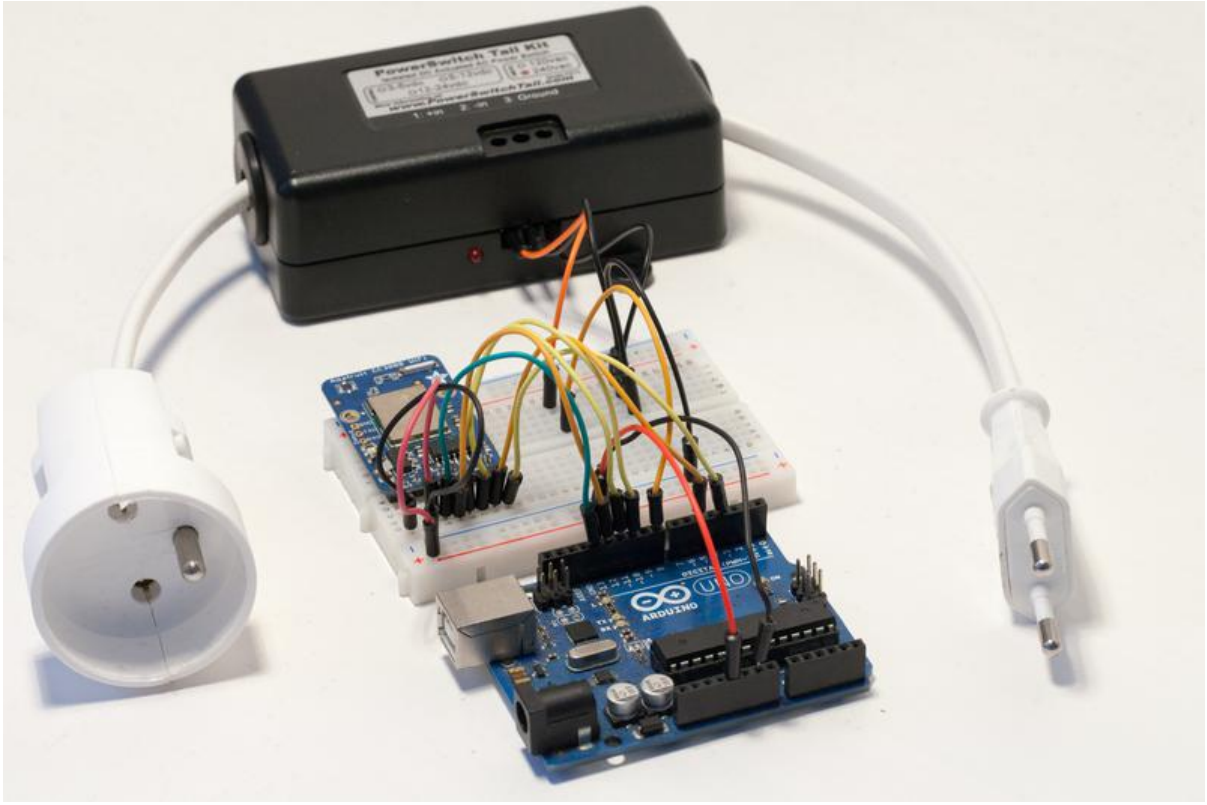
Wireless Power Switch with Arduino & the CC3000 WiFi Chip

Created by Marc-Olivier Schwartz



https://learn.adafruit.com/wireless-power-switch-with-arduino-and-the-cc3000-wifi-chip

Last updated on 2023-08-29 02:33:52 PM EDT

Table of Contents

Introduction

In this project, we are going to build an open-source version of WiFi power switches that you can buy in many stores. These switches can usually be controlled from a

smartphone or tablet, and give you the ability to switch on or off any device that is connected to the switch.

We are going to use Arduino, the CC3000 WiFi chip, and a powerswitch module to make an open-source version of such WiFi power switches. With the project we are going to build in this guide, you will be able to control this switch from your computer, or from any device connected to your local WiFi network.

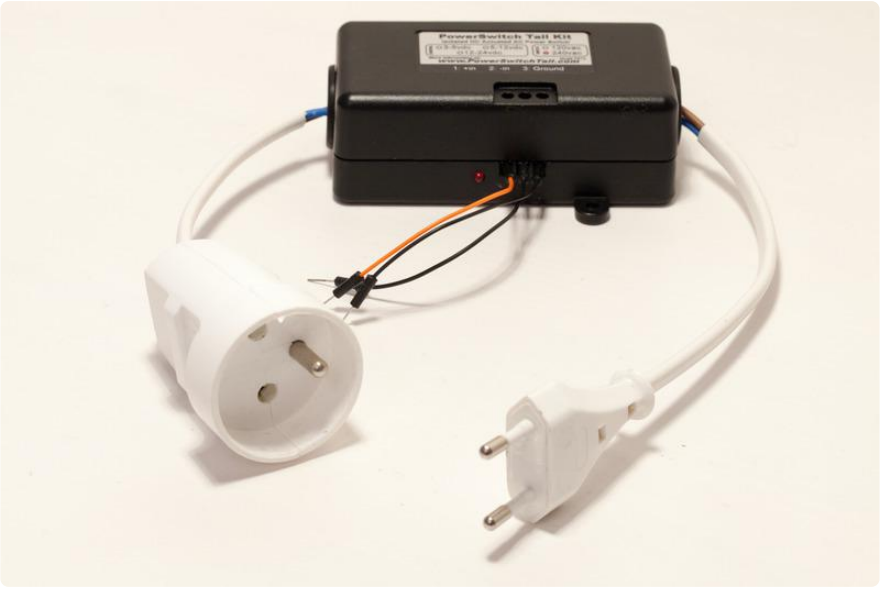
You will be able to plug a 110v or 230v device into the switch (I took a standard 30W desk lamp as an example) and switch this device on or off via WiFi. Using the interface that we are going to build, you'll also be able to wirelessly control the device that is connected to the switch from the click of a button. Let's dive in!

Hardware & Software Requirements

First, I used an Arduino Uno board for the core of this project.

For WiFi connectivity, I used an Adafruit CC3000 breakout board. You could also use an Adafruit CC3000 WiFi shield, the code would be exactly the same.

For the power switch itself, I used a powerswitch tail 2 kit, which allows to easily make the connections between the device you want to control and the mains elecricity. Then, it is really easy to control the device using a microcontroller board like Arduino. This is a picture of the kit I used (in its EU version on the picture):



Finally, I used a breadboard and some jumper wires to make the different electrical connections.

On the software side, you need to have the latest version of the Arduino IDE installed on your computer.

You will also need to have Node.js installed on your computer, to run the interface on your computer. You can download it by going at the following address:

http://nodejs.org/ ()

You will also need the aREST library for Arduino which you can find at the following link:

https://github.com/marcoschwartz/aREST ()

This project also requires having the CC3000 chip library:

https://github.com/adafruit/Adafruit_CC3000_Library ()

And the CC3000 mDNS library:

https://github.com/adafruit/CC3000_MDNS ()

To install a given Arduino library, simply extract the folder in your Arduino /libraries folder (or create this folder if it doesn't exist yet).

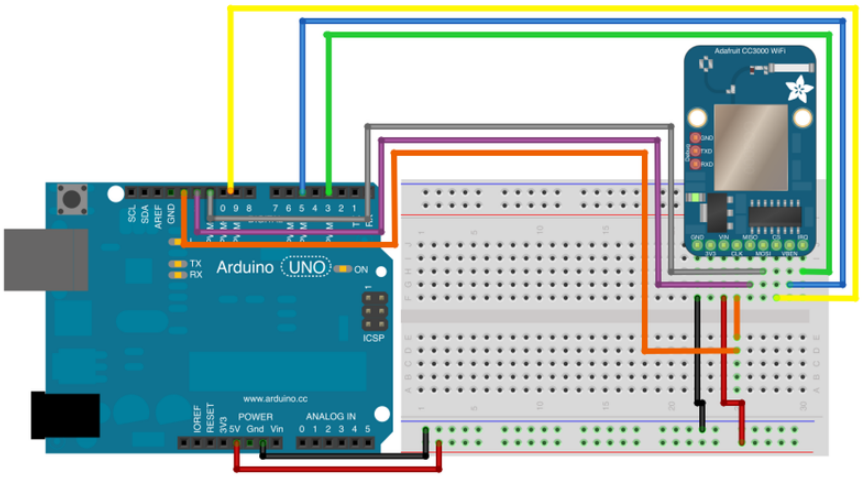
Hardware Configuration

Let's now assemble the hardware for this project. We will do so in two parts. We will first connect the different components like the WiFi module to the Arduino board, and then we will connect the powerswitch tail.

The hardware connections for the first part are actually quite simple: we just have to connect the WiFi module. First, connect the Arduino Uno +5V pin to the red rail on the breadboard, and the ground pin to the blue rail.

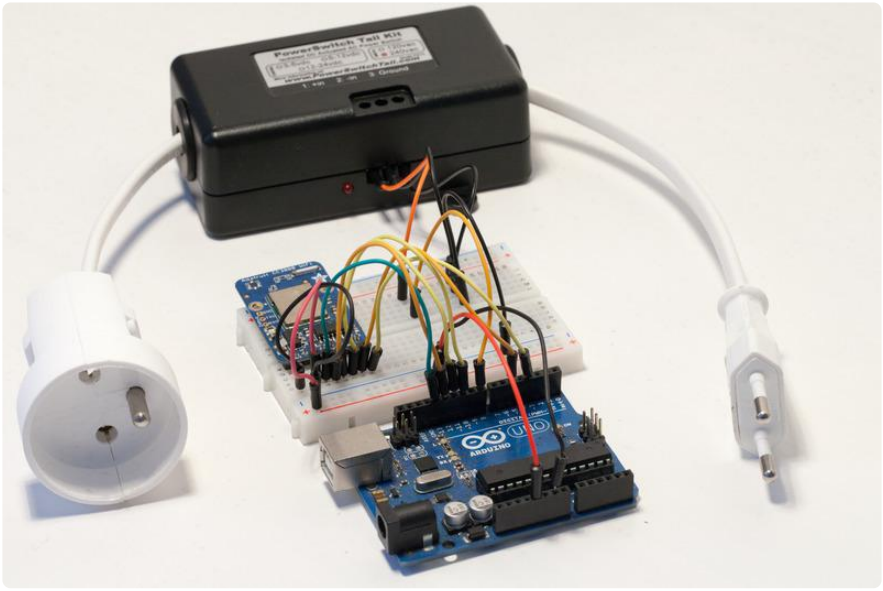
Now, the WiFi module. First, connect the IRQ pin of the CC3000 board to pin number 3 of the Arduino board, VBAT to pin 5, and CS to pin 10. Then, you need to connect the SPI pins to the Arduino board: MOSI, MISO, and CLK go to pins 11,12, and 13, respectively. Finally, take care of the power supply: Vin goes to the Arduino 5V (red power rail), and GND to GND (blue power rail).

This is a schematic of the project, without the connections to the powerswitch tail:



Then, you need to connect the Arduino board to the powerswitch tail. Simply connect the +In pin of the powerswitch to Arduino pin 8, and the two remaining pins of the powerswitch to the Arduino ground.

The following picture is an overview of the whole project fully assembled:



Finally, connect the male power plug to the mains elecricity, the device you want to control to the female power plug, and finally the Arduino board to your computer using an USB cable.

This is a picture of the assembled project, with the device to be controlled (a standard 30W desk lamp) connected to the switch:

Note that as this project involves dangerous voltages (110v or 230v), make sure that you never touch the project after it is plugged into an electrical socket.

Testing the Project

It is now time to test the project. As the most important part of the project is the relay module, that's what we are going to test here. We are simply going to switch the powerswitch on and off continuously every 5 seconds, just to check that it is working.

At this point, you should already plug a device (like the lamp I used as an example) into the project and connect the switch to an electrical socket, to see that all the connections are correctly made.

It starts by declaring on which pin the switch is connected to:

```
const int relay_pin = 8;
```

In the setup() function of the sketch, we set this pin as an output:

```
pinMode(relay_pin,OUTPUT);
```

Then, in the loop() function of the sketch, we set this pin to a HIGH state, switching on the powerswitch:

```
digitalWrite(relay_pin, HIGH);
```

And wait for 5 seconds:

```
delay(5000);
```

We then switch it off again:

```
digitalWrite(relay_pin, LOW);
```

And wait for 5 seconds before repeating the loop() function:

```
delay(5000);
```

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

https://github.com/marcoschwartz/arduino-wifi-powerswitch ()

It's now time to test the sketch. Make sure that the lamp is correctly connected to the project, and that the male plug is plugged to the mains electricity. Then, upload the

Arduino sketch to the board. You should see that every 5 seconds, the powerswitch is switching, turning the device connected to it on and off.

Remote Control

In this section, we are going to build the Arduino sketch that we will use to control the switch via WiFi.

To do so, we are going to use the aREST library that implements a REST API for Arduino. This way, we can have an easy access to the pins of the Arduino board, and also to the variable in which the power measurement is stored.

Note that only the most important parts of the code are detailed here, please get to the complete sketch on GitHub to have the complete code that you can upload to the Arduino board.

It starts by including all the required libraries:

```
#include &lt;Adafruit_CC3000.h&gt;
#include &lt;SPI.h&gt;
#include &lt;CC3000_MDNS.h&gt;
#include &lt;aREST.h&gt;
```

And declaring the pin on which the powerswitch module is connected:

```
const int relay_pin = 8;
```

Then, we create the instance of the aREST object that we will use to handle the requests coming via the WiFi connection:

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

We also need to define on which port we want to the WiFi chip to listen to. For convenience, we will use the port 80, so you can directly command your Arduino board from a web browser:

```
#define LISTEN_PORT  80
```

We also create an instance of the CC3000 server:

```
Adafruit_CC3000_Server restServer(LISTEN_PORT);
```

We also need to create an instance of the MDNS server, so we can send commands to the Arduino board without having to type the board IP address to access it:

```
MDNSResponder mdns;
```

After that, we declare the relay pin as an output:

```
pinMode(relay_pin,OUTPUT);
```

We can also set a name and an ID for the device, that will be returned at each call of the board via the aREST library:

```
rest.set_id("001");
rest.set_name("smart_lamp");
```

After this step, we set a name for the Arduino board on the network, for example “arduino”. This means that the board will be accessible by the name arduino.local on your local network:

```
if (!mdns.begin("arduino", cc3000)) {
  while(1);
}
```

Finally, still in the setup() function we start the CC3000 server and wait for incoming connections:

```
restServer.begin();
Serial.println(F("Listening for connections..."));
```

In the loop() function of the sketch, we update the MDNS server:

```
mdns.update();
```

And process any incoming connection using the aREST library:

```
Adafruit_CC3000_ClientRef client = restServer.available();
rest.handle(client);
```

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

https://github.com/marcoschwartz/arduino-wifi-powerswitch ()

It’s now time to test this sketch on our project. Download the code from the GitHub repository, and make sure that you modify the name and the password of the WiFi

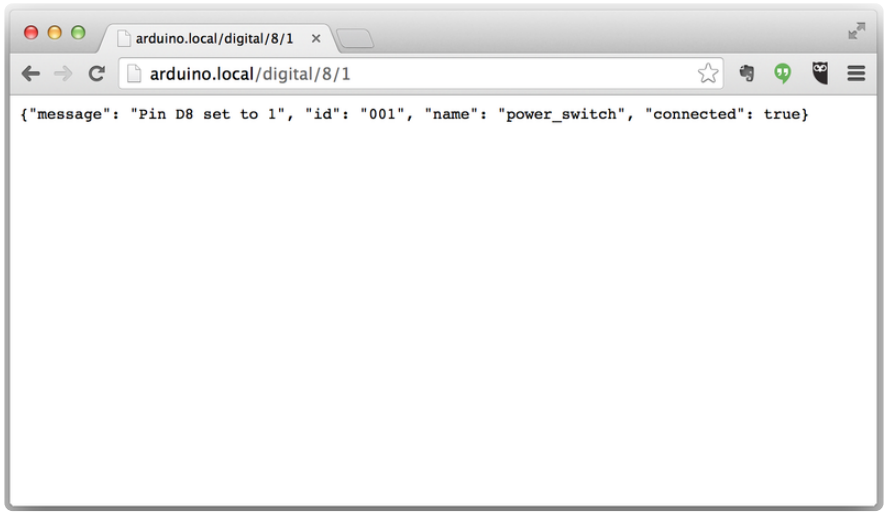
network on which the WiFi chip will connect to. Then, upload the code to the Arduino board, and open the Serial monitor. This is what you should see:

Listening for connections...

Now, close the Serial monitor, and open your web browser. You can now make direct calls to the REST API running on the board to control the pins of the Arduino board. For example, to turn the switch on, just type:

http://arduino.local/digital/8/1

You should hear the relay switching, the device connected to the project should turn on and you should have a confirmation message inside your browser:

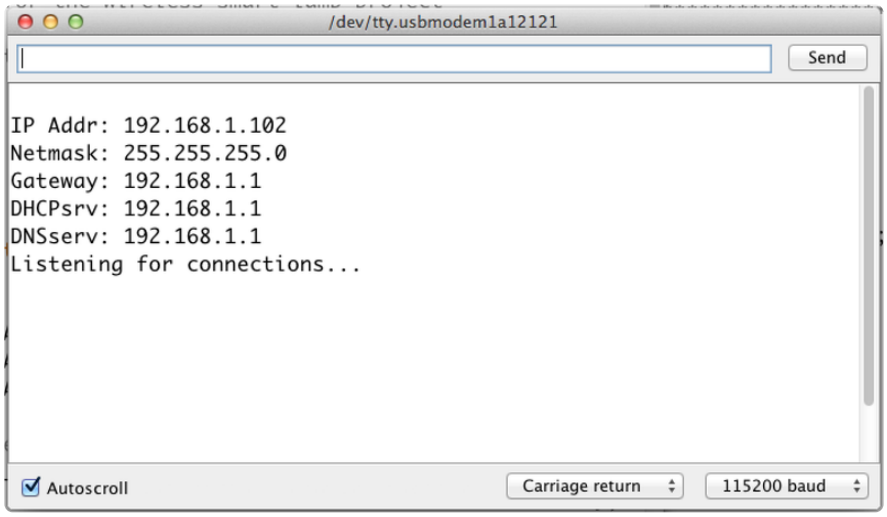


To switch the device off again, just type:

http://arduino.local/digital/8/0

Note that these commands will work from any devices connect to the same local network as the Arduino board. For example, you can use your smartphone to control your Arduino board with the same commands.

If it doesn't work, the first thing to do is to use the IP address of the board in place of the arduino.local name. You can get the IP address of the board by looking at the messages displayed on the Serial monitor when starting the project:



Building the Interface

We are now going to build the interface that you will use to control the project from your computer. Using this interface, you will be able to control the switch via WiFi just by pressing buttons.

The interface we are going to develop is based on Node.js that will basically run a web server on your computer. You don't need to have any experience with Node.js to make this project work: the code walkthrough that you will find in this section is only there to make you understand how the different pieces of the interface works together.

First, we are going to code the main file called app.js, which we will run later using the node command in a terminal. It starts by importing the required modules:

```
var express = require('express');
var path = require('path');
var arest = require('arest');
```

Then, we create our app based on the express framework, and the set the port to 3700:

```
var app = express();
var port = 3700;
```

We also need to tell our software where to look for the graphical interface that we are going to code later, and also where to find the code for the interface:

```
app.use(express.static(path.join(__dirname, 'public')));
app.use(express.static(path.join(__dirname, 'views')));
```

Now, we are going to build the different routes of our server. The first one concerns the interface itself, and this is the URL we are going to use when we want to access the graphical interface of the project. We define this route by linking the /interface URL to the corresponding HTML file:

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

We also define the URL that we will use to send commands to our Arduino project. To do so, we link the /send URL to the corresponding function in the aREST Node.js module:

```
app.get("/send", function(req, res){
  arest.send(req,res);
});
```

Finally, still in this app.js file, we start the app with the port we defined before, and write a message in the console:

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

This was for the main server file. Now, we are going to build the interface itself. Let's see the content of the HTML file first. This file is located in the /view folder of our project.

The file starts by importing the different JavaScript files that will handle the click on the interface, and send the correct commands to the Arduino board:

```
&lt;script type="text/javascript" src="/js/jquery-2.0.3.min.js"&gt;&lt;/script&gt;
&lt;script type="text/javascript" src="/js/interface.js"&gt;&lt;/script&gt;
&lt;script type="text/javascript" src="/js/arest.js"&gt;&lt;/script&gt;
```

We also include some css files to give a better look at our interface:

```
&lt;LINK href="/css/interface.css" rel="stylesheet" type="text/css" /&gt;
&lt;LINK href="/css/flat-ui.css" rel="stylesheet" type="text/css" /&gt;
```

Then, we need to create two buttons: one to turn the relay on, and another one to switch it off again. This is the code for the “On” button:

```
&lt;button class="btn btn-block btn-lg btn-primary"
type="button" id="1" onClick="buttonClick(this.id)"&gt;On&lt;/button&gt;
```

You can see that a click on this button is calling another function. We will define this function in the JavaScript file of this project that we will see in a moment.

Finally, we also need to create a text field to display an indicator to check that the project is online:

```
&lt;div class="status" id="status"&gt;Offline&lt;/div&gt;
```

Now, we are going to see the contents of the interface.js file, located in the /js folder of the project. We first need to define the type of communication between the Node.js server and the Arduino board (here, wifi) and the address of the board.

If you used the code from the GitHub repository of this project, you can leave that untouched. If you are using another name or using the IP address of the board, you need to change the address of the board here:

```
type = 'wifi';
address = 'arduino.local';
```

Then, we will define this function called buttonClick that is triggered whenever a button is clicked in the interface. Depending on which button is clicked (identified by the id of the button), we send a different command to the Arduino board. This is the complete code for this function:

```
function buttonClick(clicked_id){

  if (clicked_id == "1"){
    send(type, address, "/digital/8/1");
  }

  if (clicked_id == "2"){
    send(type, address, "/digital/8/0");
  }
}
```

Then, we will define this function called buttonClick that is triggered whenever a button is clicked in the interface. Depending on which button is clicked (identified by

the id of the button), we send a different command to the Arduino board. This is the complete code for this function:

```
setInterval(function() {
```

To get the status of the board, we use the same function as before, by sending the /id command. However this time, we store the data in a variable:

```
json_data = send(type, address, '/id');
```

This JSON data contains some information of the status of the board. If the “connected” field is present, we set the status indicator to online, and set the color to green. If the data is not present or corrupted, we set it to offline and apply a red color to this indicator:

```
if (json_data.connected == 1){
  $("#status").html("Device Online");
  $("#status").css("color","green");
}
else {
  $("#status").html("Device Offline");
  $("#status").css("color","red");
}
```

Finally, we repeat this action every 5 seconds:

```
}, 5000);
```

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

https://github.com/marcoschwartz/arduino-wifi-powerswitch ()

It’s now time to test the interface. Make sure that you download all the files from the GitHub repository, and update the code with your own data if necessary, like the Arduino board address. Also, make sure that the Arduino board is programmed with the code we saw earlier in this guide.

Then, go to the folder of the interface with a terminal, and type the following command to install the aREST module:

sudo npm install arest

And also type the following command to install the express module:

sudo npm install express

Finally, you can start the Node.js server by typing:

node app.js

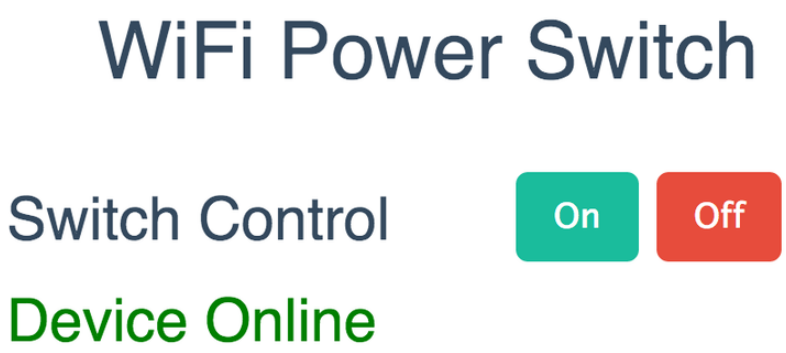
You should be greeted with the following message in the terminal:

Listening on port 3700

You can now go to the your web browser, and type:

localhost:3700/interface

You should see the interface being displayed inside your browser, with the buttons to control the switch. Don't worry, when your first open the interface the project should appear as offline and the indicator should not display any data. After 5 seconds, the interface will make a query to the Arduino board and update the data accordingly:



You can now also test the different buttons of the interface. By default, the switch is turned off, so click on the "On" button to turn it on instantly. You should also hear the "click" coming from the relay.

How to Go Further

Let's summarise what we did in this guide. We learned how to build a WiFi power switch and to control it from your web browser. We were able to control a 110v or 230v device wirelessly, for example a lamp, and monitor its status via WiFi. Finally, we built a software to control the whole project from a graphical interface within your web browser.

There are of course many ways to go further with this project. You can of course add more sensors to the Arduino board and access these measurements wirelessly. For example, you could perfectly add a temperature sensor to the project and display the data on the graphical interface as well. You can also add another powerswitch to this project, and control both independently.

By changing the code running on your computer, inside the JavaScript file, you can also define more complex behaviours for the switch. Not only you can control the switch from the interface but you can also use the fact that your computer is connected to the web to create more complex behaviours. For example, you can automatically switch it off after a given time in the evening, and switch it on again in the early morning.

Finally, you can have several of these projects in your home, simply by giving different names to your Arduino boards, and adding more elements in the graphical interface. You can also create more behaviours and buttons for your project, for example a button to automatically switch off all devices in your home with a simple click.