DATA ACQUISITION TECHNOLOGIES AND SENSOR NETWORKS

Project Report

Smart Outlet: Access and Control Your Power Outlets Remotely

By Eno Ciraku, Diogo Cosin, & Ralph Florent

December 11, 2019

Abstract

This document provides a detailed explanation about how the Smart Outlet project is built, used, and tested. This explanation spans from the specifications of the electronic modules to the very constructed prototype. It also includes information on the methodology used as well as the final results. The results being a device being switched on or off remotely by an end-user.

This document also serves as a guide for describing the different techniques used to implement the high level (software-based) part of this project.

1 Introduction

In this report, the solution, specification, and outcomes of the project of a Smart Outlet are presented as part of the course *Data Acquisition Technologies and Sensor Networks* grading criteria. The course proposes to design a sensor network communicating wirelessly via a web interface with a database. The recorded data should be visualized graphically through a web interface.

In this section, we describe our proposed solution, its motivation, and comment on this report organization.

1.1 Solution Proposal

The so-called Smart Outlet (SO) is a power outlet remotely controlled via a webpage interface. The solution offers a switch *on/off*, which can be toggled through the previously mentioned webpage. The same webpage also provides a monitoring feature that allows the users to verify the states of the SO throughout time tabular and graphically.

The components of the macro solution of the project are given follows:

- **Hardware component:** the power outlet, the controller device, and the wireless communication interface;
- **Webpage:** web-interface for controlling the SO states;
- **Database:** relational database for storing the states of the SO.

Further details of the solution are introduced in Sections 3 and 4.

In this project, we decided to constrain the SO range to a Wireless Local Area Network (WLAN). However, this limitation can be overcome by exposing the SO solution in a public HTTP interface. This feature may be implemented in a possible next step of this project (out of scope in this report).

1.2 Motivation

Besides being a requirement to the successful completion of the course *Data Acquisition Technologies and Sensor Networks*, building the SO offers many advantages to a possible end user. For these reasons, we decided to implement a prototype for this solution. These advantages are described as follows:

- **Convenient usability:** the SO can be controlled from any place considering that the webpage may be exposed publicly via HTTP requests;
- **Energy saving:** the user can track the SO state and turn it off when considering that it has been unnecessarily used;

- **Multiple users:** the solution enables the access of different users through different platforms by only requiring a device that can access and visualize webpages exposed on the internet. In other words, users with computers, smartphones, and tablets can control the SO.

1.3 Note on the Report Structure

Before the next Sections of the report, in which technical details and results are presented, let us briefly enlighten how this report is structured. The structure follows:

- **Theoretical review:** theoretical and technical details of the chosen hardware components are briefly presented;
- **Instrumentation & resources:** the integration of the hardware and software of the SO prototype is detailed;
- **Methodology:** the techniques, strategies, and procedural methods used to implement the core functionality of the project are presented;
- **Results & discussions:** results obtained with the SO prototype implementation.

2 Theoretical Review

This section briefly provides information about the hardware components used in this project. If desired, more information can be found on their respective bibliographical references, also provided in this section.

2.1 Arduino Mega 2560 Rev3

The Arduino Mega 2560 Rev3, showed in Figure 1, is a microcontroller based on the the Atmel's microcontroller ATmega2560. In this subsection, we limitate ourselves to the ArduinoMega 2560 Rev3. However, general and technical details about the microcontroller ATmega2560 can be found in [1].

A microcontroller is an electronic device and a small computer placed on a single metal-oxide-semiconductor (MOS) integrated circuit. It contains at least one Central Processing Unit (CPU), memory, and input/output peripherals which interface in different voltage levels, typically 3.3V and 5V. Program memory is also included in this type of device. This way, developers can adapt the device logically according to the desired application. They are normally deployed in embedded systems such as automobile engine control systems, implantable medical devices, remote controls, office machines,

Figure 1: An Arduino Mega Rev3 unity.

among others. With the new Internet of Things (IoT) trend, these devices act in data collection, sensing, and actuating components in the physical world [2].

Besides the microcontroller itself, the Arduino Mega 2560 Rev3 also contains 54 digital input/output pins, 16 analog inputs, 4 UARTs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. Through external shields, the technical capabilities may be extended for different purposes, such as Bluetooth, ZigBee, Wi-Fi, and GPS connections, among others [1].

Through its Integrated Development Environment (IDE), developers can program this microcontroller according to the required application. The Arduino programming language is a collection of C and C++ functions called as specified by the developers. Behind the hoods, these functions are directed to a C/C++ compiler. Nevertheless, it worths mentioning that developers can also use different software to program Arduino, however, some extra steps are required [3].

2.2 Elegoo 4-Channel Relay

The Elegoo's 4-channel relay GE-EL-SM-006, showed in Figure 2, is a set of 4 relays 5V active low. It also provides 4 Ligh-emitting diodes (LEDs) for verification of the relays' status.

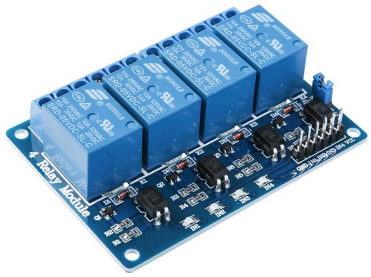

Figure 2: The Elegoo's 4-channel relay GE-EL-SM-006.

A relay is an electrical switch that controls its status (open or closed) according to the input signal.

In its traditional form, a magnetic field is created to attract a coil and consequently close the contact, hence the switch behavior. In addition, other operating principles have been created, as in solid-state relays that use semiconductor features for control. These devices are commonly applied as electrical switches, as in the project presented in this report, but also protective systems [4].

This module also allows the interface with different controllers, among them the Arduino (briefly introduced in the Subsection 2.1). These controllers can be then particularly programmed to command the 4-Channel relay states.

2.3 Espressif ESP8266EX

The Espressif's ESP8266EX, showed in Figure 3, is a System on a Chip (SoC), an integrated circuit, which offers Wi-Fi networking capability as well as microcontroller capabilities. It can be deployed in standalone applications, given that it gives CPU, programmable Random-access memory (RAM), and Read-only memory (ROM), that is, capabilities that a microcontroller provides. Not only that, but the ESP8266EX can also be employed in applications as the slave to a host microcontroller. For example, the device can be applied to any microcontroller as a Wi-Fi adaptor, as designed in this report's project [5].

Figure 3: The Espressif's ESP8266EX.

Some key capabilities of the Espressif's ESP8266EX follows [5]:

- **Wi-Fi support:** 802.11 b/g/n, 802.11 n support (2.4 GHz), up to 72.2 Mbps;
- **CPU:** Tensilica L106 32-bit Reduced instruction set computer (RISC) processor;
- **Memory:** includes memory controller and memory units including SRAM and ROM;
- **External flash:** external SPI flash to store user programs, supporting up to 16MB;
- **Clock:** clock generated from internal crystal oscillator and extenal crystal;
- **General Purpose Input/Output Interface (GPIO):** 17 GPIO pins which can be customized according to the program logic;

- **I2C:** interface with other microcontrollers and other peripheral equipments performed via software programming.

Others technical features of the ESP8266EX include: Universal Asynchronous Receiver Transmitter (UART), Pulse-Width Modulation (PWM), Infrared (IR) Remote Control, Analog-to-Digital Converter (ADC) [5].

3 Intrumentation & Resources

Recalling that the SO prototype is the complete integration of a set of hardware and software, we detail in this section more in-depth specifications on them. Below are enlisted the tools used to set up and carry out successfully the current version of the project.

3.1 Hardware and other materials

The SO hardware refers to the physical components that add up to the *Webduino* module. Although there are many possible options in choosing different kinds of hardware to build up the circuitry, we opt for the most reasonable[1] choice, which is to reuse already-prebuilt modules and integrate them into one. Hence, in Table 1 are listed the items with their corresponding details.

Hardware & Other Materials			
	Models	**Quantity**	**Cost**
Microcontroller Board	MEGA 2560 R3	1	€ 14,99
4-Channel Relay	GE-EL-SM-006	1	€ 6,99
Wi-Fi Module	ESP8266-01	1	€ 6,99

Table 1: Detailed information on materials and hardware used for the SO prototype.

Additionally, other useful materials such as:

- 12x male-to-male jumper cables (20 cm)
- 12x male-to-female jumper cables (20 cm)
- 12x female-to-female jumper cables (20 cm)

[1]Building a separate module from scratch requires time and work, plus other tasks to refine a working component.

- 1x access point or router (tp-link TL-WR940N)
- 2x computing devices or laptop (Lenovo T460p and Macbook Air)

remain useful to connect the different components and have them work as one stable module. More technical specifications are given on each one of the materials and devices, including their working conditions, in Section 2.

Notes: Observe that the monetary budget reaches the sum of €28,97 for only one SO prototype that eventually includes the monitoring and control of 4 (four) power outlets. Most of the materials were bought online on Amazon.de and some of them, provided by the *Data Acquisition* lab.

Considering that the main purpose of building this prototype is educational, we do not account for commercial goals. That leaves room for improvement in the future if we want to aim for industrialization of the project. That signifies that, at some point, we might need to rethink the choice of tools and software to optimize the build of the hardware module, which may, in turn, favor a cost reduction in our financial records.

3.2 Tools and software

As for the programming tools used to create, debug, maintain the code of the Smart Outlet at both hardware- and software-level, we use free programs yet efficient that are available for most of the OS[2] platforms. These programs are of different types such as code editors (e.g Visual Studio Code), compilers (e.g LaTeX), online platforms (e.g GitHub), and so forth. With that being said, we present the set of tools and software used at the time of implementing the prototype:

- Operating systems (GNU/Linux, Mac OS, and Windows)
- Visual Studio Code (lightweight text editor)
- Git[3] (version control)
- GitHub (web-based hosting service for Git versioning system)
- Jupyter Notebook (workspace for scripting and simulation)
- Arduino IDE (development environment for Arduino boards)

Regarding the software versions, it is highly recommended to use the exact versions mentioned in Table 2 to avoid conflicts and compiling errors. On the other hand, the developer may want to dig into the breaking changes (if that is the case) that most of the time require to refactor parts of

[2]OS: Operating Systems like Windows, Mac OS, Linux, and so on.

[3]Git is also available as a bash emulation for other platforms for free (e.g., Git Bash for Windows).

Tools & Software			
	Versions	**Sources**	**Cost**
Visual Studio Code	1.40.1	See [6]	Free
Git	2.7.4	Built-in Linux program	Free
GitHub	N/A	See [7]	5 free users
Python	3.7.1	See [8]	Free
Jupyter Notebook	5.7.4	See [9]	Free
Arduino IDE	1.8.10	See [10]	Free

Table 2: Detailed information on the tools and software used for the Smart Outlet coding procedure and the technical documentation.

the implementation to have a fully working prototype. However, it is also recommended to check the *changelog* of the updates or releases, if there are any.

Note that some tools mentioned above are just a matter of personal preferences. Other preferred options are more than welcome as long as the developer keeps in mind the development speed and productivity. For example, many developers would choose Sublime over *Visual Studio Code*. But they both end up facilitating the same routine: text edition.

3.3 Programming languages

Finally, we use the following programming languages:

- C/C++ (for the webduino)
- Python with the micro web framework Flask (for the web API service)
- Angular Framework - JavaScript/TypeScript (for the web application)

Important: *Though we highly recommend that the exact versions of the software and the exact models of the materials and devices are used to test out or replicate this project, keep in mind that these hardware might no longer be available in the market as well as the software components might be outdated at some point in time. If that is the case, stay alert to the updates as we intend to support this project until 2022.*

4 Methodology

In this section, we describe the techniques and strategies, as well as the procedural methods used to implement the core functionality of this project. This description includes the components of the system, the workflow diagram, the third-party libraries, the algorithm and data structure, and finally, the code implementation.

It is essential to stand out the fact that we use external resources to come up with daily results in terms of programming tools and full-on working devices. That is, before adventuring into using the materials and devices as well as the appropriate software tools, different research sources were consulted. Some of these resources are the datasheets of both the prebuilt modules and the materials, the assistance of the instructor and teaching assistants (TAs), search engines (mainly Google Search/Internet), and, finally, some related books and references (e.g. materials provided by the professor).

Being exceptionally helpful, these resources have been the source of truth for any decision-making regarding the correct use of the hardware modules.

4.1 Components of the prototype

Roughly speaking, Smart Outlet comprises 3 (three) principal modules:

- **Webduino:** a low-level, modular circuitry formed by an Arduino and a network of sensors. The Arduino board, a microcontroller, acts as a supervisor of microtasks. Being the core component of the hardware systems, it controls the different input/output functions of the connected chips.
- **Web API:** an API service attending HTTP requests from the *webduino* (its only consumer). It coordinates the communication between the Wi-Fi module of the webduino (client) over the air medium (wireless) and the available API resources on an HTTP server. The API service contains various layers of interactions, including the database for data persistence.
- **Web APP:** (short for web application) a single page application (SPA) to visualize the historical content or executed actions during the webduino's operations. It allows user-friendly interactions between an end-user and the prototype itself. The web app is also responsive. That is, it can be accessed and used via mobile devices (tablets, smartphones, etc.).

Each one of these modules deep down contains a set of characteristics that requires more than a brief description to highlight their corresponding functionality. However, in this document, we intend to explain only how to connect them and make them work properly as a whole. We indeed provide full access to the online repository as specified in Appendix A so that anyone can dig any further into the datasheets if needs be.

4.2 Workflow diagram

The Smart Outlet project has an easy-to-understand workflow. This workflow explains how each module is connected (not fully as in a mesh connection) and performs a specific task. In this case, we provide a diagram and explain the corresponding role of the inner contents taking part in that workflow in other to achieve the complete functionality of Smart Outlet as a whole.

As shown in Figure 4, the workflow diagram consists of the following parts:

- **Devices accessing the web application:** a user can use smartphones or desktops (laptops) to access and load the web application. This end-user device should be connected to the local area network (LAN) with this IP: `192.168.0.0/24`.
- **Server infrastructure:**
  - an HTTP server to serve the website when the user browses its web address[4];
  - an API service to attend the requests from consumers (webduino and web app).
- **Webduino:** the hardware module or prototype connected to the same network waiting to update the states of the outlets, if any.
- **Router:** working like a switch, assures connectivity and communication between the other components of the system.

Important: *Although every connected device lands on the same network, it is highly important to understand that no direct connectivity between the web application and the webduino is established. Refer to UML diagram in Figure 5 to get a better idea of how this interaction is made. To illustrate this point, if we had architected the diagramming system this way, we would confront serious problems at the time of taking the web service to public. That is, routing to a local IP (when there are some many security concerns to consider) would be troublesome.*

How do the components interact with each other? Observe the *UML (Unified Modeling Language)* in Figure 5 to grasp the idea faster as it is a visual aid to display the sequence of events occurring when the prototype is operating.

1. Given the initial conditions, all the components are considered up and running properly;[5]

[4]Note that a web address, in this case, is the associated IP address and the port where the Apache 2.0 service is running. (e.g 192.168.0.105:4200)

[5]The devices (database, API web service, webduino) should be powered on and running. Each device should be assigned a local IP address and should be reachable by other devices over the network.

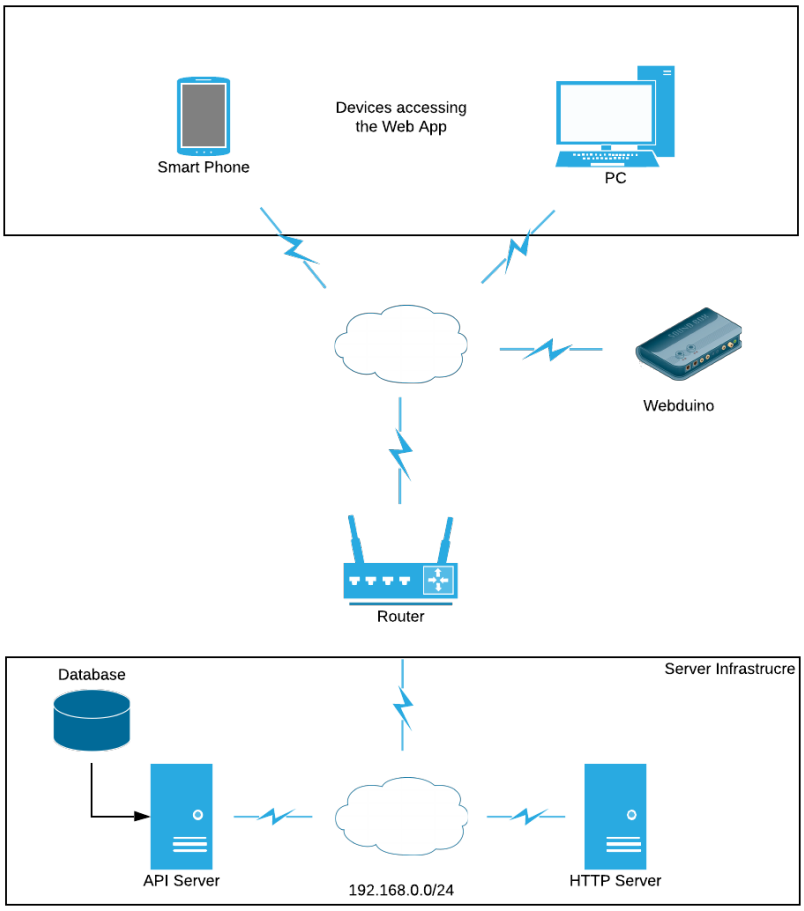

Figure 4: Workflow diagram - this is an architecture diagram that displays how every component of the system is interconnected in a real-world situation. (credits: made with *Lucidchart*)

2. A user accessing a web browser may load and view the last updates of the power outlets. When a user updates[6] the state of an outlet, the changes are saved into the database through the API service and immediately (2 seconds) reflected in the webduino.

3. The webduino will constantly attempt to request the last states of the power outlets from the web API service. Once obtained the data, it proceeds by updating the states of the relays, which in turn will open or close the switch of the corresponding power outlet(s).

4. The historical records can also be visualized on separate views using both a tabular form and a scatter plot.

[6]Set an update for a power outlet is basically to change its state from ON-OFF or vice-versa.

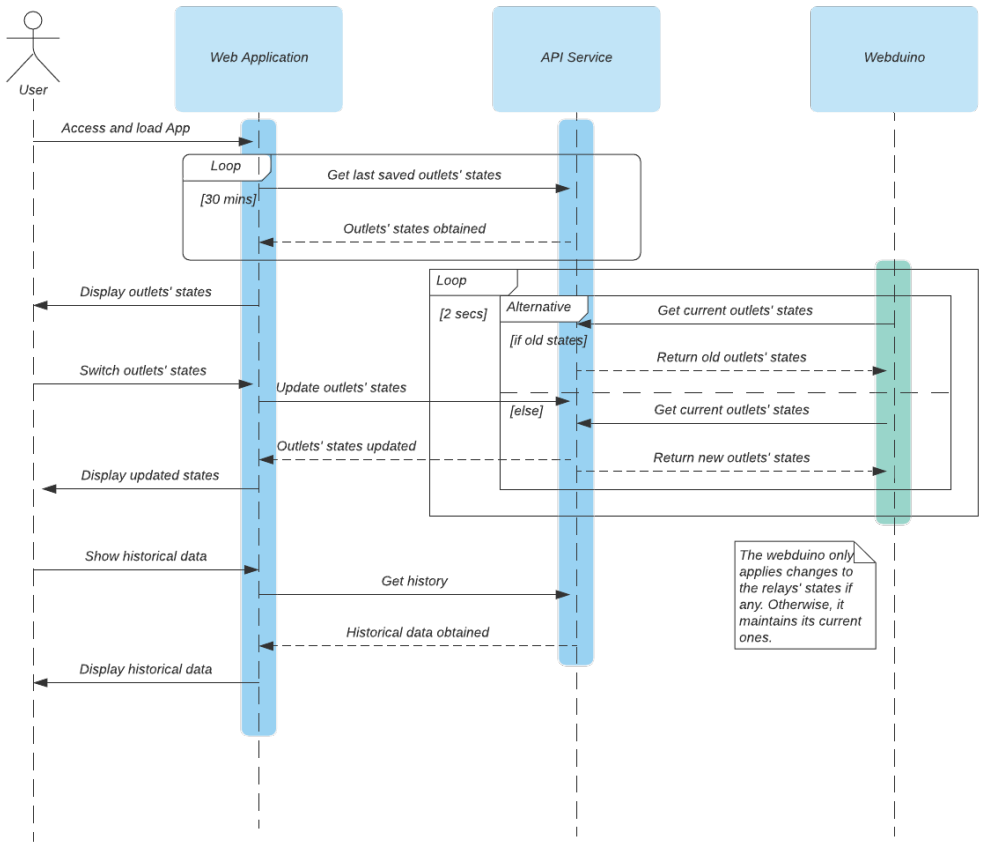

Figure 5: UML sequence diagram - this shows how every object (web app, API, webduino) within the Smart Outlet system interacts with each other and the order where these interactions take place. (credits: made with *Lucidchart*)

4.3 Algorithm and data structure

Recall that the project touches different development environments. These environments require different architectures, techniques of programming as well as the programming languages used. As the line of thinking is distinct for every module, this section discusses each environment separately and eventually shows how they relate to each other.

4.3.1 Webduino

The Smart Outlet prototype, also known as *webduino*, is the final integrated piece of hardware obtained when combining the microcontroller (Arduino board), the relay module, and the Wi-Fi (ESP8266-01) module. Since we are dealing with different kinds of prebuilt submodules, we proceed by evaluating their functionality under a pilot conceptual test. More details on these tests are provided in Section 5.

Arduino supports coding in C/C++ and the concept of Object-Oriented Programming (OOP). This capability gives us an advantage on how to better organize the code. After different attempts to optimize the code, we end up using a class-based structure that facilitates the handling of the relay, that is, the *ON-OFF* switch, and finally, convert it into a reusable library. Observe for example in Listing 1 the class definition representing an outlet.

```
#ifndef Outlet_h
#define Outlet_h

class Outlet {
  public:
    // constructors
    Outlet();
    Outlet(char, boolean);

    // setters
    void setId(char);
    void setState(boolean);

    // getters
    char getId();
    boolean getState();

    // methods
    void begin();   // initialize arduino setup
    void turnOn();  // actuates a digital switch from 0 (off) to 1 (on)
    void turnOff(); // actuates a digital switch from 1 (on) to 0 (off)

  private:
    char _id;
    boolean _state;
};

#endif
```

Listing 1: Example of class definition of Outlet

We create and use other utilities as well as helper functions to achieve the final implementation. Similarly, we use this external library *ArduinoJson* to parse JSON data coming from the server.

The overall logic is quite simple: when the Arduino starts, the submodules' setups are initialized and then fall in an infinite *loop* while executing an updating status check from the API service. Note the same coded logic in Listing 2. Keep in mind that the entire code is publicly available and can be consulted for further reading and understanding, as specified in Appendix A.

```
// Omit previous scripts (imports, setup, global variables, etc.)
void loop () {
  if (connect(0, SERVER_API)) {
    bool isSent = sendRequest("GET", ENDPOINT, "", 0, SERVER_API);
    if (isSent) {
      if (DEBUG) Serial.println("Request has been sent successfully!");
      bool parsed = parseRelayStates();
      if (parsed) updateRelays();
    } else {
      if (DEBUG) Serial.println("Request couldn't be sent...");
    }
    disconnect(0);

  } else {
    if (DEBUG) Serial.println("Connection to the server " + String(SERVER_API) + " failed!");
  }
}
// Omit other scripts (helpers)
```

Listing 2: Continuous checks for updating relays' states

After running the sample tests for every separate module, we put them all together and test them out as one module. And to run the webduino independently, we make the response data available by using a mock API.

4.3.2 Web API

The Web API (abbreviation for Application Programming Interface) provides read and write access to the states of the relays, stored in the relational database MySQL. For that, three endpoints are designed with the aid of the Python micro web framework Flask. Their functions are given as follows:

- **status:** GET interface to provide the status of the outlets in JSON format (see Figure 6 for the response format);
- **statusino:** GET interface to provide the status of the outlets in JSON format in a lighter version. This endpoint is specifically designed for the communication with the Arduino to save data transmission. This way, the field *updatedOn* is ommited from the response as it is not necessary for the correct control of the outlets;
- **update:** POST interface to update the status of the outlets in the database;
- **history:** GET interface to provide the historical data for the status of the outlets (see Figure 7 for the response format).

```
{
  "data": [
    {
      "outlet": 1,
      "state": 0,
      "updatedOn": "Wed, 13 Nov 2019 12:03:55 GMT"
    },
    {
      "outlet": 2,
      "state": 1,
      "updatedOn": "Wed, 13 Nov 2019 12:02:22 GMT"
    }
  ]
}
```


Figure 6: An example of the endpoint *status* response.


```
{
  "data": [
    {
      "id": 1,
      "outlet": 2,
      "state": 2,
      "updatedOn": "Tue, 05 Nov 2019 13:40:32 GMT"
    },
    {
      "id": 2,
      "outlet": 2,
      "state": 2,
      "updatedOn": "Tue, 05 Nov 2019 13:44:05 GMT"
    },
    {
      "id": 3,
      "outlet": 2,
      "state": 2,
      "updatedOn": "Tue, 05 Nov 2019 13:44:35 GMT"
    },
    {
      "id": 4,
      "outlet": 2,
      "state": 2,
      "updatedOn": "Tue, 05 Nov 2019 13:49:36 GMT"
    },
    {
      "id": 5,
      "outlet": 2,
      "state": 2,
      "updatedOn": "Tue, 05 Nov 2019 13:50:58 GMT"
    },
```


Figure 7: An example of the endpoint *history* response.

4.3.3 Web Application

The Web App (short for web application), also known as the *Web UI* module of this project, is an Angular-based[7] application. Angular is a JavaScript framework used for building web applications in a fast yet effective manner. It contains appealing features and is powered by Google.

As a single web application, the routing done via views. The main components or views of the web app are:

- **home:** a splash screen page serving as the entry point of the web app (see Figure 8). It also describes the web page purposes;

[7] Consult the README.md markdown file to learn about the installation and running process when using Angular.

- **outlets:** a page to perform updates on the outlets' states (See Figure 9). A user can both view and change the states of the currently available outlets;
- **history:** a page to have a basic overview of the past saved data in tabular (see Figure 10) and graph format (see Figure 11).

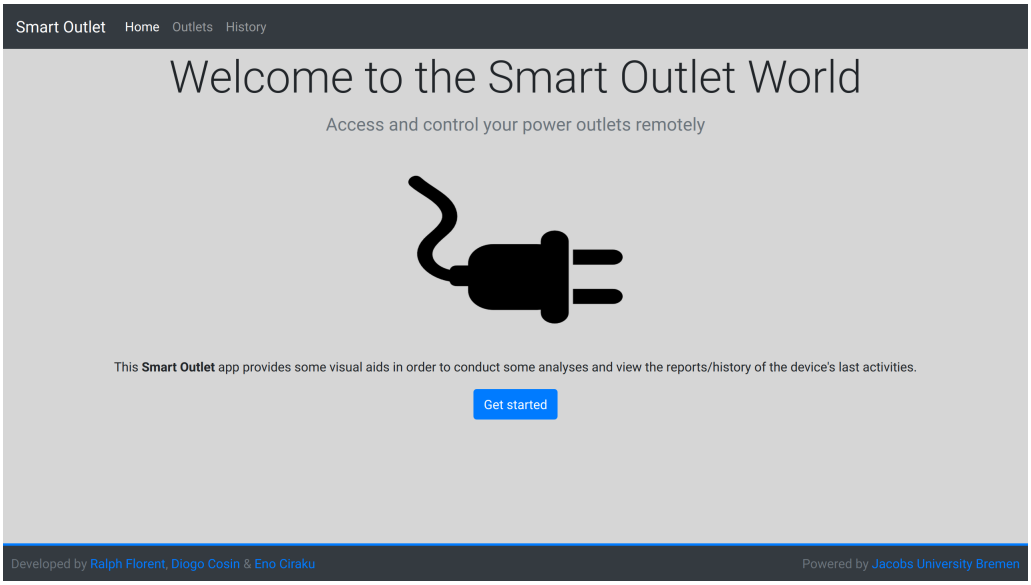

Figure 8: View of the home page

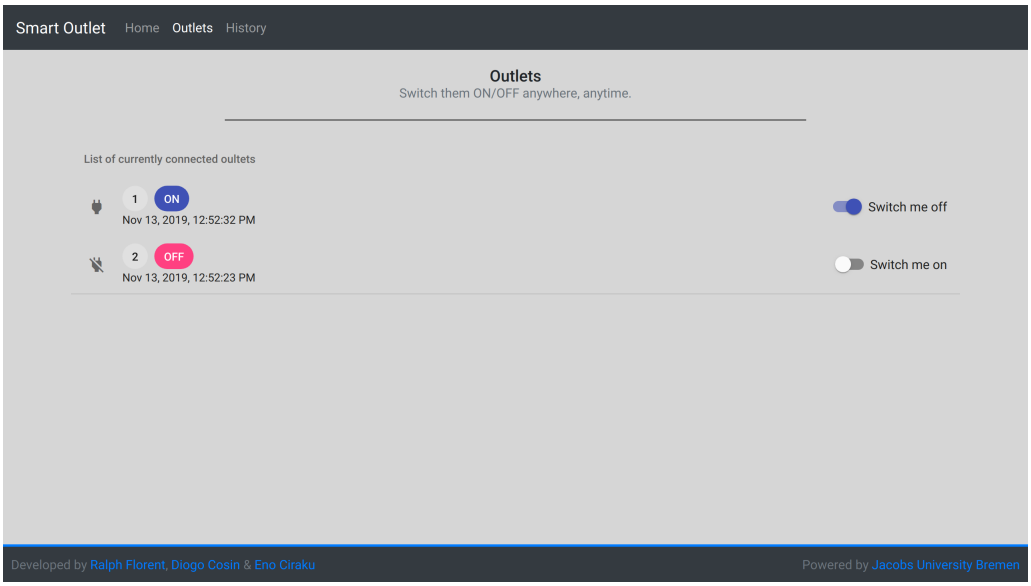

Figure 9: View of the outlets page

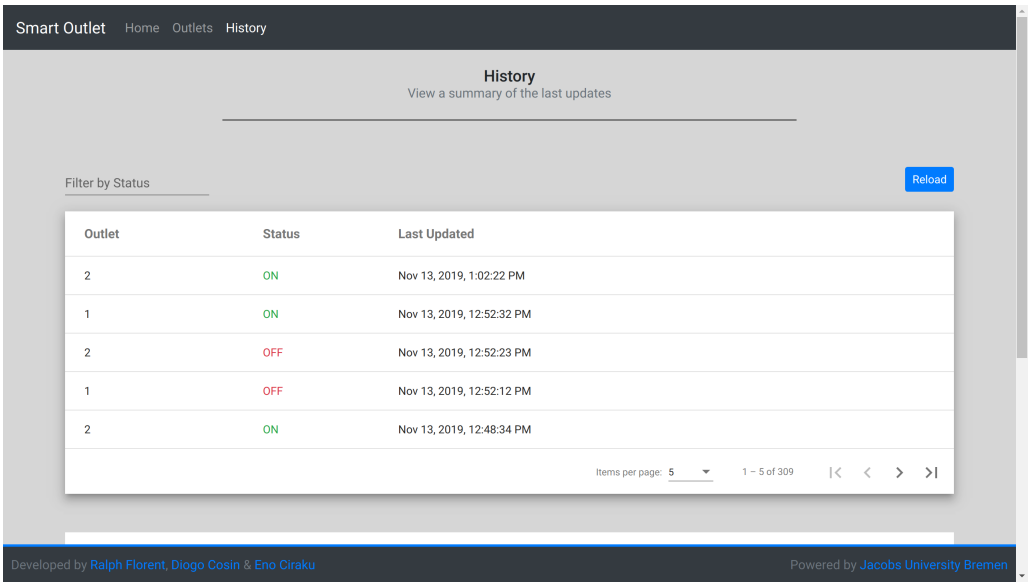

Figure 10: View of the history page

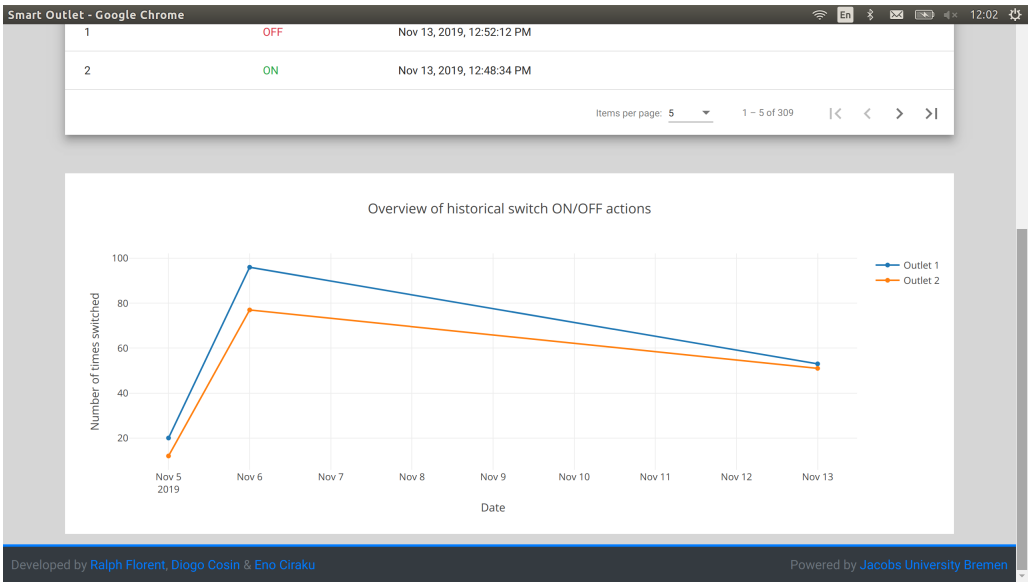

Figure 11: View of the plot page

We do not detail much about the algorithm and data structure used for the web application since the main focus is not on how to build a web page efficiently. However, any curious developer is welcome to check the *README* content of the web page to learn more about how to take a web app from development to production. Be aware that the programming techniques are well advanced and robust.

In addition to that, the web app is unit-tested and dockerized.

5 Results and Discussions

In this report, the different, relevant aspects of the project prototype called Smart Outlet have been presented and described. The central component of the project is the Arduino, which acts as micro-task supervisor and controls the input/output functions of the chips as described in Section 4. Figure 12 shows the prototype assembled in the laboratory environment.

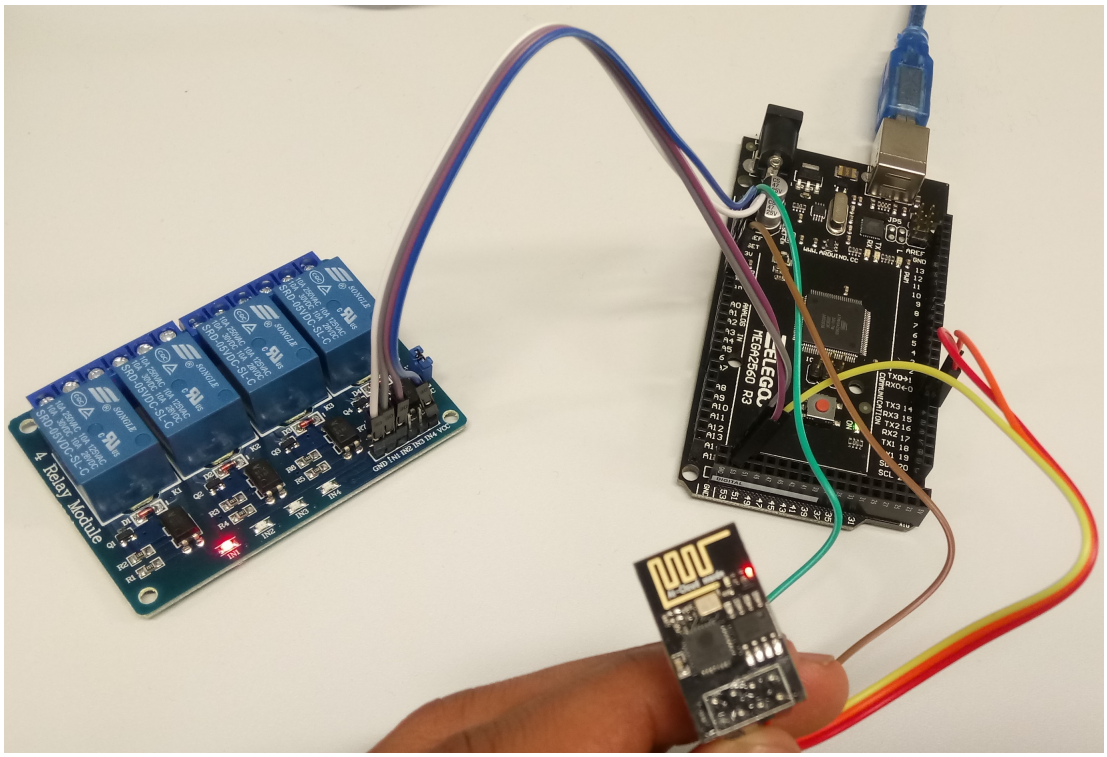

Figure 12: Assembled prototype.

Other important aspects of this work include the architecture and design of the different components used to make the prototype work. Knowledge of Computer Networking, Web API, Web APP and Sensor Circuitry were fundamental in building a complex module with the intended functionality.

To complete the project, trial and error approaches were employed in gaining a greater control over each of our different components and then we proceeded to merge the functionalities of the components. This part also required trial and error as the interconnection of the modules not always worked out the way they were expected to.

The prototype works as described in Section 4. The user is able, either through a smartphone or a PC to manipulate the Smart Outlet by switching it ON/OFF with the ultimate goal being a better energy consumption. The user can also view the status of the outlet in real-time and further view a history of the working of the outlet as shown in Figure 10.

References

[1] Arduino. *Arduino*. Dec. 2019. URL: https://store.arduino.cc/arduino-mega-2560-rev3.

[2] Wikipedia. *Microcontroller*. Dec. 2019. URL: https://en.wikipedia.org/wiki/Microcontroller. (Last updated December 07, 2019).

[3] Arduino. *Arduino - FAQ*. Dec. 2019. URL: https://www.arduino.cc/en/main/FAQ. (Last updated December 06, 2019).

[4] Wikipedia. *Relay*. Dec. 2019. URL: https://en.wikipedia.org/wiki/Relay. (Last updated December 06, 2019).

[5] Espressif Systems. *ESP8266EX*. Feb. 2018. URL: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf. (Last updated February, 2018).

[6] Microsoft Corporation. *Visual Studio Code*. Oct. 2019. URL: https://code.visualstudio.com/.

[7] GitHub Inc. *The world's leading software development platform*. Nov. 2019. URL: https://github.com/.

[8] Python Software Foundation. *Welcome to Python.org*. Mar. 2019. URL: https://www.python.org/.

[9] Project Jupyter. *Project Jupyter*. Jan. 2019. URL: https://jupyter.org/. (Last updated April 12, 2019).

[10] Arduino. *Arduino - Software*. Oct. 2019. URL: https://www.arduino.cc/en/main/software. (Last updated October 20, 2019).

Appendix A Code Repository

All the code implemented during the execution of the prototype described in this report is available on the GitHub repository https://github.com/ralflorent/smart-outlet.

Appendix B Understand the Repository

It is highly recommended to check and read the markdown files (e.g. *README.md*) to encourage further understanding of every part of the repository. It is a self-sufficient, self-explanatory repository where every taken step is detailed with sustainable reasons.

We also try to follow the best practices by using the recommendations of the open-source community. For example, observe how we use the commit messages, the file structures, the naming conventions, and so forth.

Finally, we use up-to-date tools and software so that we can take advantage of their fully-available features. To illustrate our point, we use the last version Angular Framework to develop the web application. That is, supporting and upgrading this framework is indisputably and relatively easy.

Appendix C Prototype Demo

A video was recorded demonstrating the prototype fully operational. The video can be found at the link https://youtu.be/EptGgv-rxUE.