

PROYECTO DE FINAL DE CARRERA

INGENIERÍA TELEMÁTICA

FACULTAD DE INGENIERÍA

UNIVERSIDAD DE MONTEVIDEO



UNIVERSIDAD DE MONTEVIDEO



Iván BABIC (ibabic@correo.um.edu.uy)

Alexis ZECHARIES (azecharies@correo.um.edu.uy)

Tutores:

Rafael Sotelo

Thomas Hobbins

Marzo de 2018

Índice general

1 Resumen Ejecutivo	1
1.1 Antecedentes	1
1.2 Objetivos	1
1.3 Metodología	1
1.4 Objetivos cumplidos	2
2 Introducción	3
2.1 Objetivo General	3
2.2 Objetivos Específicos	3
2.3 Descripción del Producto	3
2.4 Necesidades satisfechas	4
2.5 Justificación de impacto	4
2.6 Antecedentes tecnológicos	4
3 Marco Teórico	7
3.1 Arquitectura del sistema	7
3.2 Git	8
3.3 PlatformIO	9
3.4 Protocolos IoT	10
3.4.1 HTTP	10
3.4.2 CoAP	10
3.4.3 MQTT	11
3.5 Sonoff	14
3.5.1 SPIFFS	14
3.6 USB TTL Serial	16
3.6.1 USB TTL Serial Adapter	16
3.6.2 USB TTL Serial Cable	17
3.7 Mini Procesadores	17

ÍNDICE GENERAL

3.7.1	Raspberry Pi 3	17
3.7.2	Raspberry Pi 2 B	18
3.7.3	Banana Pi 3	18
3.7.4	BeagleBone Black	19
3.8	Python	20
3.8.1	pip	20
3.8.2	virtualenvs	20
3.9	Flask	21
3.10	MongoDB	22
3.11	React	23
3.12	Cron	23
3.13	Apache HTTP Server	24
3.14	Amazon EC2	25
3.15	Mycroft	25
4	Marco Metodológico	27
4.1	Tareas comprendidas y no comprendidas	27
4.2	Entregables	28
4.3	Restricciones: Tiempo, presupuesto, tecnología	29
4.4	Supuestos y riesgos	29
4.5	Estructura de Desglose de Trabajo (EDT)	30
4.6	Desglose de Paquetes de Trabajo en Actividades	30
4.7	Procedimientos	30
4.7.1	Protocolos	30
4.7.2	Librerías	31
4.7.3	Hardware	32
5	Estructura del Contenido	35
5.1	Arquitectura	35
5.2	Módulos del Sistema	38
5.2.1	Módulo MQTT Mesh	38
5.2.2	Módulo de Control	39
5.2.3	Módulo de interfaces de usuario	40
5.3	Desarrollo	42
5.3.1	Módulo MQTT Mesh	42

5.3.2	Módulo Controlador	62
5.3.3	Módulo de interfaces de usuario	65
5.3.4	Aportes a librerías de terceros	85
6	Conclusiones	87
7	Recomendaciones	89
Anexos		93
I	Entregables	95
I.1	Justificación de avance de 50 %	95
II	Código Fuente	103
II.1	Sonoff	103
II.1.1	main.cpp	103
II.1.2	credentials.h.j2	119
II.1.3	platformio.ini	119
II.2	Raspberry Pi	120
II.2.1	mosquitto.conf	120
II.2.2	api.py	120
II.2.3	mongo_db_utils.py	134
II.2.4	device_template.json	140
II.3	Aplicación Web	140
II.3.1	index.js	140
II.3.2	App.js	140
II.3.3	LightControlConsole.js	141

Índice de figuras

3.1	<i>Arquitectura general del sistema construido.</i>	7
3.2	<i>Interfaz de Atom con PlatfromIO.</i>	10
3.3	<i>Capaz de stack de protocolos CoAP [18].</i>	11

ÍNDICE DE FIGURAS

3.4	<i>Diagrama de red MQTT.</i> [22].	11
3.5	<i>Composición de un topic MQTT.</i> [23].	12
3.6	<i>Uso de la wildcard -.</i> [23].	13
3.7	<i>Topics ejemplo.</i> [23].	14
3.9	<i>Diagrama de SPIFFS en memoria</i>	14
3.10	<i>USB Serial Adapter.</i> [39].	16
3.11	<i>USB Serial Cable.</i> [40].	17
3.12	<i>Raspberry Pi 3.</i> [34].	18
3.13	<i>Raspberry Pi 2.</i> [33].	18
3.14	<i>Banana Pi M3.</i> [7].	19
3.15	<i>BeagleBone Black.</i> [8].	19
4.1	<i>Estructura de Desglose de Trabajo.</i>	30
5.1	<i>Redes en el sistema.</i>	37
5.2	<i>Componentes del Módulo MQTT Mesh.</i>	38
5.3	<i>Componentes del Módulo de Control.</i>	39
5.4	<i>Componentes del Módulo de UI para Aplicación Móvil.</i>	40
5.5	<i>Componentes del Módulo de UI para Asistente Virtual.</i>	41
5.6	<i>Conexiones del ESP8266 expuestas por el Sonoff.</i>	47
5.7	<i>Sonoff modificado.</i>	47
5.8	<i>Conexion de interruptores con Sonoff.</i>	48
5.9	<i>Vista de página principal provisoria.</i>	67
5.10	<i>Flujo de autenticación con Auth0.</i>	68
5.11	<i>Selectores jquery-cron.</i>	70
5.12	<i>Página de inicio.</i>	72
5.13	<i>D.U.D.E. Auth0 Log In</i>	72
5.14	<i>Página de configuración de dispositivo central.</i>	73
5.15	<i>Página de inicio con sesión iniciada.</i>	73
5.16	<i>Vistas de barra lateral.</i>	74
5.17	<i>Cambiar estado de dispositivo.</i>	75
5.18	<i>Cambiar estado de grupo.</i>	75
5.19	<i>Vistas de selección.</i>	76
5.20	<i>Agregar acción programada.</i>	77
5.21	<i>Agregar dispositivo.</i>	77

5.22 Agregar grupo.	78
5.23 Editar dispositivo.	79
5.24 Editar grupo.	79
5.25 Pantalla de error.	80
5.26 Elección de SO.	80
5.27 Elección de máquina virtual.	81
5.28 Elección de grupo de seguridad.	81
5.29 Creación de grupo de seguridad.	81
5.30 Elección de llave.	82
5.31 Panel informativo.	82

Índice de cuadros

3.1 Comparación de conceptos MongoDB vs BDD relacionales..	22
4.1 Entregables: descripción y fecha	28
5.1 Tabla de topics de comunicación con dispositivos Sonoff	55

Capítulo 1

Resumen Ejecutivo

1.1. Antecedentes

Este proyecto nace por el interés en la automatización de los objetos cotidianos. En la actualidad tanto los componentes electrónicos y como las plataformas y protocolos de IoT han avanzado a tal punto que es posible su implementación a un costo accesible para las masas. Se desea que este proyecto contribuya en esta proximidad creciente y hacer de las casas inteligentes un sueño factible para cualquier interesado.

1.2. Objetivos

Se creó un sistema de domótica altamente adaptable, de fácil instalación y uso. El mismo provee módulos reusables para permitir la construcción de interfaces que consuman sus servicios. Nuestro objetivo final fue lograr un producto con un costo por el cual sea posible automatizar la totalidad de los elementos electrónicos de un hogar, oficina o empresa, a un precio razonable y brindando un servicio que compita con las soluciones actuales, dando la opción de no utilizar aplicaciones o asistentes virtuales de empresas que registren los comportamientos de los usuarios.

1.3. Metodología

Se programó los dispositivos Sonoff, y se utilizó un Raspberry Pi como plataforma central para contener los servicios de coordinación y configuración, permitiendo que el proceso de instalación y uso sea lo más simple e intuitivo posible. Se proveen dos métodos de interacción

con el sistema, siendo estos reemplazables en un futuro con soluciones de terceros como Google Home o Alexa.

1.4. Objetivos cumplidos

(No aplica en esta entrega)

Capítulo 2

Introducción

2.1. Objetivo General

Se llevó a cabo un sistema de domótica de bajo costo utilizando tecnologías emergentes en el mundo de las TIC, haciendo foco en el área de IoT, y así adquirir experiencia tanto teórica como práctica en los componentes de hardware y software del state-of-the-art de esta área.

2.2. Objetivos Específicos

Se busca llevar a cabo un sistema de domótica modular, utilizando la mayor cantidad de código open source y hardware orientado al desarrollo. De esta manera se permite el mayor control y conocimiento del sistema, dejando la libertad de utilizar los protocolos que se encuentre más adecuados para cada módulo y de elegir los componentes de hardware que sea posible obtener, teniendo siempre como meta la obtención de un sistema adaptable y de bajo costo.

2.3. Descripción del Producto

El producto cuenta con un módulo central que expone servicios a distintas interfaces de usuario. Este módulo se encarga de formar la red de dispositivos de control, administrarlos y comunicarse con los mismos, tanto para consultar estados como enviar instrucciones. Como se mencionó anteriormente, se obtuvo una arquitectura modular, la cual permite, además de mantener un orden en el desarrollo e investigación, la capacidad de adaptar los distintos

componentes a servicios ya establecidos, ya sean de código abierto o de API's configurables. Un ejemplo de cada uno es Home Assistant [17] y Google Home [15]. Estas adaptaciones quedaron fuera del alcance inicial de este proyecto, pero se desarrollaron interfaces propias, una en forma de aplicación web (fácilmente adaptable a aplicación móvil) y otra como asistente personal (específicamente se utilizó el asistente open source Mycroft [27]).

2.4. Necesidades satisfechas

El producto busca acercar a toda persona interesada en sistemas de automatización a la posibilidad de contar con un servicio de este tipo, ya sea en el hogar, oficinas o cualquier locación con acceso a Internet. Con esta implementación se aumenta tanto la comodidad del usuario como la eficiencia en el uso de los componentes eléctricos controlados por los interruptores inteligentes. Algunos de los casos de uso en los que esto se ve reflejado son:

2.5. Justificación de impacto

Se encontró carencia en el mercado actual, ya que se ha detectado una tendencia de reemplazo de componentes, y se busca con este desarrollo brindar la capacidad de adaptación de elementos que no pueden ser controlados remotamente.

2.6. Antecedentes tecnológicos

En la actualidad existen muchas soluciones para automatización de casas, su gran debilidad: la gran mayoría se enfoca en el cambio de los elementos a automatizar, es así en las luces Phillips [30] o su competencia con precios más económicos IKEA [19]. Existen también soluciones de adaptación de dispositivos, como el mismo dispositivo Sonoff, pero su código cerrado no deja realizarle modificaciones a la implementación. Uno de los problemas a los que se enfrentan estos dispositivos es que el alcance de WiFi nunca cubre la totalidad del área habitada donde está instalado, algo que se busca solucionar con este sistema. Además de problemas técnicos, hemos notado un movimiento de concientización por parte de desarrolladores open source, los cuales se están esforzando para brindar opciones a los dispositivos de

automatización de grandes empresas, las cuales utilizan datos obtenidos para sacar provecho de los mismos.

Capítulo 3

Marco Teórico

3.1. Arquitectura del sistema

En la figura [3.1] se pueden apreciar los componentes del sistema y cómo interactúan entre ellos. Además se puede observar los protocolos que utilizan para estas interacciones, a continuación se presentarán los conceptos básicos del sistema.

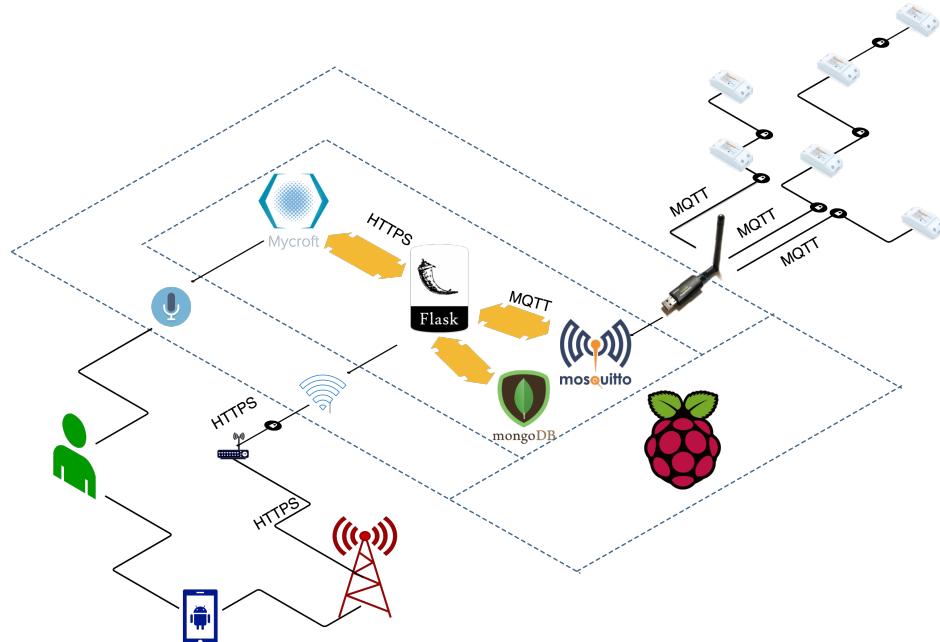


Figura 3.1: *Arquitectura general del sistema construido.*

El usuario interactúa con el servidor Flask a través de alguna de las interfaces brindadas. Luego el servidor se encarga de interactuar con los interruptores inteligentes según corresponda.

3.2. Git

Para el desarrollo de código grupal, es necesario utilizar un controlador de versiones del mismo, ya que es posible que varios desarrolladores quieran efectuar cambios en el mismo archivo, perdiéndose o dañándose parte del código. Esta herramienta fue diseñada por Linus Torvalds, conocido por iniciar y mantener el desarrollo del kernel de Linux, y en la actualidad es ampliamente utilizada en proyectos con gran extensión de código y varios desarrolladores. Algunos conceptos básicos a manejar del mismo son:

- Branch: es una versión paralela a un repositorio, contenida en el mismo pero permite realizar cambios sin modificar la versión “oficial”, una vez terminados y testeados los cambios se pueden aplicar a la versión utilizada por los demás desarrolladores.
- Clone: es una copia del repositorio que se encuentra alojada en el computador del desarrollador en vez de en el servidor donde se guarda el repositorio.
- Commit: es un cambio individual de un archivo o varios de ellos, es el equivalente a guardar un cambio, pero en vez de ser en el sistema local de almacenamiento es en git. Esto permite llevar un registro de los cambios y quién los hizo, esta tarea es usualmente facilitada con un mensaje adherido a cada commit con una breve descripción de los cambios. Este registro permite volver a estados anteriores y deshacer cambios incorrectos.
- Push: Este comando envía los cambios registrados en los commits anteriores, reflejando los mismos en el repositorio remoto para que sean visibles para los demás usuarios.
- Pull request: son cambios propuestos a un repositorio, realizados por un usuario y aceptados o rechazados por los colaboradores del repositorio en cuestión. Cada pull request cuenta con su propio foro de discusión y es posible requerir cambios antes de aceptar o rechazar el mismo.

Para este proyecto se utilizó una estrategia de 2 branches principales y un branch por nueva funcionalidad, las cuales se mencionan a continuación:

- master: Esta branch aloja checkpoints del proyecto, por lo que el código en la misma debe ser testeado antes de trasladarse y debe tomarse como puntos de referencia en

casos de fallo del sistema. Ningún usuario tiene autorización para realizar push a esta branch, por lo que hasta los administradores del proyecto deben abrir un pull request y esperar que este sea aprobado para efectuar cambios.

- dev: Branch de desarrollo, sale de el último commit de master y de ella se deben abrir las branches de nuevas funcionalidades, es una branch de adaptación de código, por lo que su estado no es tan estable como master. Los usuarios tampoco tienen permitido realizar push directos, por lo que deberán abrir un pull request para mergear las branches de funcionalidades.
- branches de funcionalidades: son la estrategia a seguir para desarrollar nuevas funcionalidades, los desarrolladores deben ir pusheando sus cambios y al terminar una tarea, recién solicitar reflejar estos cambios en dev.

3.3. PlatformIO

Es un IDE creado para un ambiente de desarrollo en IoT. Satisface una necesidad de los programadores a la hora de embarcarse en proyectos de IoT la cual es que hay muchas tecnologías diferentes con las que debe trabajar, esto hace que lograr la configuración correcta para cada una de ellas pueda ser un dolor de cabeza si no se maneja con este IDEs o similares. PlatformIO logra integrar todo dentro del mismo ambiente de programación. [41]

Por ejemplo, para programar en un board Arduino típicamente se debe descargar el IDE de Arduino en donde se compila y sube el código. Si para cada board hubiera que hacer lo mismo, y además saber que configuración crearle a cada uno para que funcione con las diferentes librerías que se desea integrar al proyecto, lo cual representa un gran contratiempo. Lo que brinda PlatformIO es la posibilidad de descargar diferentes boards (más de 200) de desarrollo (como Arduino), así como integrar diferentes librerías para cada plataforma y todo de forma intuitiva y fácil desde una interfaz gráfica.

Otro factor positivo es que se integra fácilmente con Atom [4], que es un IDE conocido y previamente usado por los integrantes del grupo.

La librería mesh que elegimos tiene como dependencias otras librerías y usa una plataforma llamada espressif [5]. Para configurar esto alcanza descargar el board en PlatformIO,

agregar una línea en el archivo de configuración diciendo cual se usa y desde la interfaz gráfica descargar las librerías que son dependencia.

PlatformIO incluso se encarga de mantener actualizadas las plataformas y librerías en su última versión. Hacer esto manualmente es una pérdida de tiempo.

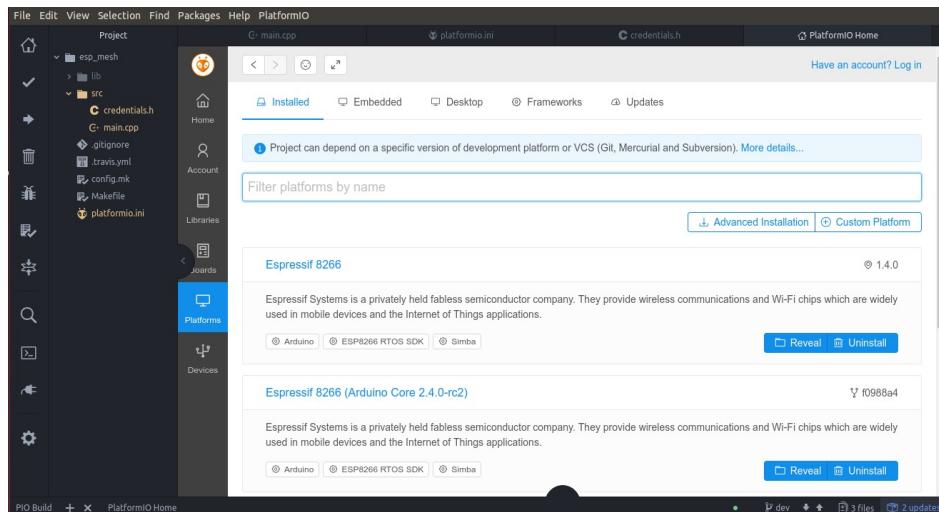


Figura 3.2: Interfaz de Atom con PlatfromIO.

3.4. Protocolos IoT

3.4.1. HTTP

Hypertext Transfer Protocol o HTTP es un protocolo de capa de aplicación que posibilita el intercambio de comunicación entre cliente y servidor. La última versión fue liberada en el 2015, descripta en la RFC 7540. Funciona por arriba de la capas TCP/IP y es muy popular. Se basa en operaciones de solicitud/respuesta. Una vez que un cliente se conecta al servidor empieza el intercambio de solicitudes y respuestas entre ambos.

3.4.2. CoAP

CoAP (Constrained Application Protocol) está definido en la RFC 7228 y fue creado con la concepción de ser un protocolo ligero apto para la transferencia de datos en nodos con un procesamiento limitado y redes también limitadas, como todo protocolo de IoT. Así como

HTTP, este protocolo está basado en una arquitectura REST (Representational State Transfer), pero tiene una diferencia fundamental frente a HTTP la cual es que HTTP utiliza como capa de transporte TCP y CoAP UDP.

Para ilustrar lo anteriormente dicho:



Figura 3.3: Capaz de stack de protocolos CoAP [18].

3.4.3. MQTT

Message Queue Telemetry Transport (MQTT) es un estándar ISO (ISO/IEC PRF 20922) que funciona sobre TCP/IP y está pensado para correr en dispositivos con bajo nivel de procesamiento como los ESP8266.

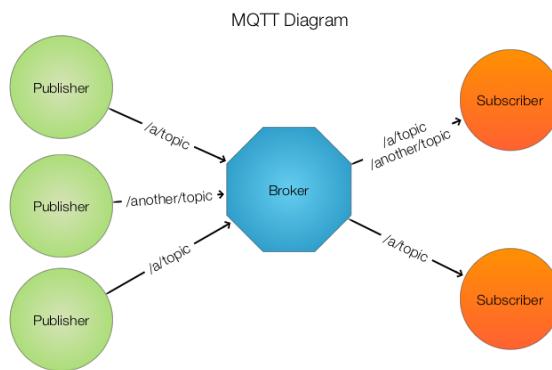


Figura 3.4: Diagrama de red MQTT. [22].

Como se ve en la figura [3.4] la arquitectura de este protocolo consiste de 3 partes fundamentales: el publisher, el broker y el subscriber. El broker es el encargado de recibir los mensajes procesar a qué topic está dirigido ese mensaje que envía el publisher y mandarlos

hacia los subscribers que estén escuchando los mensajes de ese topic particular. Un topic es un String como puede ser "this/is/a/topic" y subscribers que escuchan los mensajes de ese topic son los que están suscritos a ese topic. Así como CoAP, este protocolo es M2M por lo que sirve para conectar los clientes que son los nodos al servidor.

Hay conceptos introductorios que son necesarios aclarar a la hora de entender el funcionamiento del protocolo. Estos son:

Cliente:

En MQTT un cliente es cualquier dispositivo que esté conectado a la red MQTT.

Patrón Publish/Subscribe:

En este patrón, el dispositivo que publica (publisher) un mensaje desconoce qué dispositivo/s escuchan este mensaje. El cliente publica a un broker que es el encargado de mandar ese mensaje a los subscribers, que son los dispositivos están interesados en escuchar ese mensaje. Esto quiere decir que los publishers no saben de la existencia de los subscribers y viceversa. Por lo general los clientes pueden publicar mensajes y suscribirse a topics.

Topics:

Un topic es un string en formato UTF-8, el cual es usado por el broker para filtrar los mensajes para cada cliente, el mismo consiste de varios niveles que se usan para poder publicar y suscribirse a varios topics, de forma general o específica, de manera fácil e intuitiva. Cada nivel está separado por un separador (/).



Figura 3.5: Composición de un topic MQTT. [23].

Esta solución es extremadamente liviana y no requiere previa inicialización por parte del broker, por lo que el mismo aceptará suscripciones y publicaciones a cualquier topic válido. Algunos detalles a tener en cuenta:

- Los niveles no pueden ser un string vacío: "mqtt//topic/"
- Los mismos son case-sensitive: "mqtt/topic != mqtt/Topic"
- Un separador marca un nuevo nivel: "mqtt/topic != mqtt/topic/"

Wildcards:

En algunos casos es útil suscribirse a topics con determinadas características, pero que no se sabe con exactitud el topic completo, para esto existen las llamadas wildcards, que representan uno o varios niveles de nombre desconocido.

- De un nivel: "+"



Figura 3.6: Uso de la wildcard "-". [23].

Cualquier topic que respete la estructura con un string en lugar de la wildcard será reconocido como un match, por ejemplo si se suscribe a "myhome/groundfloor/+ / temperature" se recibirán o no los mensajes publicados a los siguientes topics:

- Multinivel: "#"

A diferencia la wildcard de un nivel, esta wildcard sólo puede ser colocada al final de un topic y suscribe a todo topic que comience igual que el topic suscrito, sin importar que tantos niveles tenga a continuación.

- Topics que comienzan con "\$"

A parte de los caracteres mencionados, el otro que tiene un funcionamiento especial es el "\$", la función que cumple es que, cuando un topic comienza con este carácter,

- ✓ myhome / groundfloor / livingroom / temperature
- ✓ myhome / groundfloor / kitchen / temperature
- ✗ myhome / groundfloor / kitchen / brightness
- ✗ myhome / firstfloor / kitchen / temperature
- ✗ myhome / groundfloor / kitchen / fridge / temperature

Figura 3.7: Topics ejemplo. [23].



entonces no será posible suscribirse al mismo utilizando los wildcards anteriores, esto es así ya que estos topics son reservados para estadísticas internas de MQTT, y los clientes no tienen permitido publicar en dichos topics.

3.5. Sonoff

3.5.1. SPIFFS

A pesar de que en el chip ESP8266 [11] el sistema de ficheros(o filesystem, FS) este guardado en el mismo chip que el firmware, cambiar el programa por un nuevo script no modificará el FS. Esto es gracias a la sectorización del espacio virtual del chip como se muestra en el diagrama [3.9].

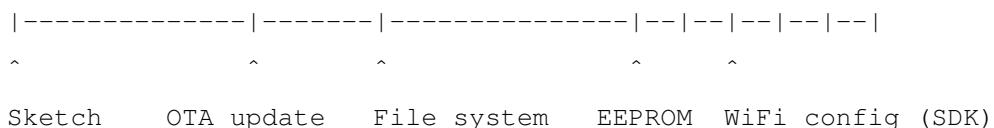


Figura 3.9: Diagrama de SPIFFS en memoria

En el caso del chip que viene integrado en los dispositivos Sonoff que se utilizaron en este proyecto, el tamaño total del chip es de 1Mb y los tamaños que se pueden asignar al FS son 64kb, 128kb, 256kb, 512kb.

Algunas de las funciones básicas para el manejo de SPIFFS son:

begin

```
SPIFFS.begin()
```

Este método monta el sistema de archivos SPIFFS, debe ser invocado antes de intentar utilizar ninguna API del FS. Retorna `true` o `false` según si el montaje fue correcto o no.

format

```
SPIFFS.format()
```

Este método formatea el FS, puede ser llamado antes o después de **begin**.

open

```
SPIFFS.open(path, mode)
```

Abre un archivo. `path` debe ser la ruta absoluta empezando con una barra (eg. `/dir/filename.txt`). `mode` es un string que especifica el modo de acceso, los valores posibles son `"r"`, `"w"`, `"a"`, `"r+"`, `"w+"`, `"a+"`. Significando estos lo mismo que para `fopen` en C.

Devuelve un archivo, para chequear que un archivo se haya abierto correctamente, se debe usar un operador booleano.

```
File f = SPIFFS.open("/f.txt", "w");
if (!f) {
    Serial.println("file open failed");
}
```

exists

```
SPIFFS.exists(path)
```

Devuelve `true` si `path` existe, `false` en caso contrario.

remove

```
SPIFFS.remove(path)
```

Remueve un archivo dado su path. Retorna true o false según si el archivo fue correctamente removido o no.

3.6. USB TTL Serial

3.6.1. USB TTL Serial Adapter

Para borrar el código inicial con el que vienen instalados los Sonoff era necesaria una conexión desde el USB de la computadora o dispositivo donde se encuentra el código hacia la UART del ESP8266.

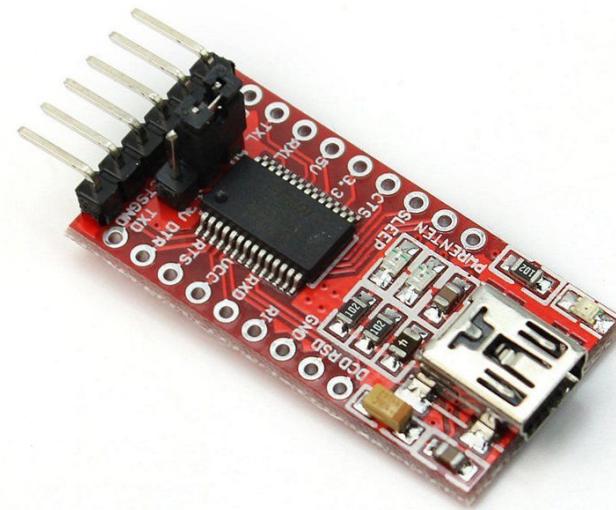


Figura 3.10: *USB Serial Adapter.* [39].

Para lograrlo, se compraron dispositivos llamados adaptadores USB a TTL serial. De un lado se le conecta un cable mini USB y del otro lado se conectan jumpers que salen de los pines VCC, GRD, Tx, Rx, mientras que se dejan otros 2 pines libres que son el DTR y el CTS. Este chip tiene un jumper para elegir si la salida es de 3,3V o 5V.

3.6.2. USB TTL Serial Cable

Otra posibilidad que se consideró fueron cables USB a TTL serial que como se ve en la figura (x) no tienen ningún elemento activo.



Figura 3.11: *USB Serial Cable*. [40].

3.7. Mini Procesadores

3.7.1. Raspberry Pi 3

Los Raspberry Pi (RPi) son computadoras que vienen en un solo circuito (single-board computer), incluyen el microprocesador, la memoria, los input/outputs, la conexión HDMI. Fueron creados en el Reino Unido educar acerca de computación así como para integrar sistemas donde se necesite hardware de bajo costo. Raspberry Pi ha ganado popularidad por todas las prestaciones que da, para el poco costo que tiene.

Se han desarrollado varios modelos, el que se utiliza en el proyecto es el Raspberry Pi 3 Model B es más reciente. Requiere una tarjeta microSD, su procesador es de 1,2GHz, memoria RAM de 1Gb y cuenta con las siguientes características entrada Ethernet (RJ-45), 17 GPIOs, alimentación de 5V a través de micro USB o GPIO, salida HDMI, 1 GB de RAM, CPU 64-bit quad-core ARMv8, 4 puertos USB. Este modelo trae integrado WiFi 802.11n y Bluetooth 4.1.



Figura 3.12: *Raspberry Pi 3*. [34].

Su precio es de aproximadamente USD 40.

3.7.2. Raspberry Pi 2 B

La tarjeta de memoria que se le coloca cambia de SD a microSD y el procesador es de 900MHz. Este modelo no cuenta con módulo de WiFi ni Bluetooth integrado, por lo que se debe conectar un dongle en caso de requerir estas características. A pesar de esto, su precio es igual o mayor que el RPi 3.



Figura 3.13: *Raspberry Pi 2*. [33].

3.7.3. Banana Pi 3

Este single-board computer fue influenciado fuertemente por el RPi en su diseño, este modelo es más potente que el RPi 3, con un procesador de 1,8 GHz y memoria RAM de 2Gb,

además de contar con memoria flash integrada de 8Gb, lo cual se ve reflejado en su precio que asciende a USD 80.



Figura 3.14: *Banana Pi M3*. [7].

3.7.4. BeagleBone Black

Esta plataforma cuenta con similares características que las anteriores, procesador de 1Ghz, memoria de 512MB, incluyendo también una aceleradora gráfica de 3D y un acelerador NEON de punto flotante, haciéndolo altamente performante para cálculos vectoriales. Sus conexiones son USB, Ethernet, HDMI , 2x 46 pin headers. Su valor es de aproximadamente USD 63.

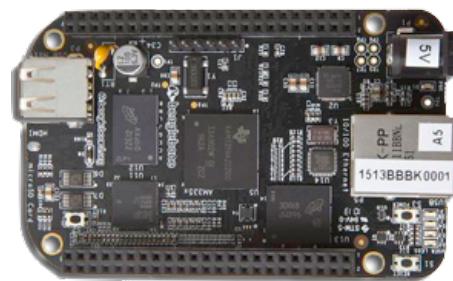


Figura 3.15: *BeagleBone Black*. [8].

3.8. Python

3.8.1. pip

Es un sistema de gestión de paquetes utilizado para instalar y administrar paquetes escritos en Python. Presenta una manera fácil y rápida de instalar paquetes, siendo su interfaz amigable e intuitiva. Algunos comandos son:

- Instalar un paquete:

```
$ pip install <nombre del paquete>
```

- Desinstalar un paquete:

```
$ pip uninstall <nombre del paquete>
```

- Mostrar paquetes instalados:

```
$ pip list
```

- Instalar paquetes indicados en un archivo de requerimientos:

```
$ pip install -r requirements.txt
```

3.8.2. virtualenvs

Esta es una herramienta ampliamente utilizada por desarrolladores Python, la misma permite aislar el ambiente de desarrollo del resto de los paquetes Python instalados en el computador del desarrollador. Esto facilita el control de versión de paquetes y permite que el ambiente de desarrollo y producción sean tan similares como sea posible. La creación y uso de un virtual env (venv) es muy sencilla, a continuación se muestran los comandos a utilizar:

- Instalar la herramienta virtualenvwrapper:

```
$ pip install virtualenvwrapper  
$ export WORKON_HOME=~/Envs  
$ mkdir -p $WORKON_HOME  
$ source /usr/local/bin/virtualenvwrapper.sh
```

- Crear un ambiente virtual:

```
$ mkvirtualenv env1
```

- Para instalar un paquete en el venv se debe primero activar el mismo:

```
$ workon env1
```

- Y luego instalar el paquete normalmente:

```
$ pip install <nombre del paquete>
```

- Para salir de un venv:

```
$ deactivate
```

3.9. Flask

Se puede leer la descripción "Flask is a micro web development framework for Python.-[12] en la página oficial de Flask, pero qué significa esto?

"Micro" no significa que toda la aplicación web deba estar contenida en un sólo archivo de Python (aunque bien podría estarlo), ni significa que Flask carece de funcionalidades. El término "micro" se refiere a que Flask tiene como objetivo mantener el núcleo simple pero extensible, lo que es sumamente útil en casos como el de este proyecto, donde se tiene componentes de hardware con capacidades limitadas, y Flask permite agregar sólo las funcionalidades necesarias, ahorrando recursos que otras soluciones menos flexibles y customizables con funcionalidades fijas consumirían. Una frase que describe muy bien a esta implementación es que "Flask puede ser todo lo que necesitas y nada más." Para lograr esto, Flask soporta extensiones para agregar todas las funcionalidades que se necesites como si fueran implementadas por él mismo. Algunas de las librerías disponibles permiten integración con bases de datos, validación de formularios, manejo de subida de archivos, varias tecnologías open source de autenticación, entre otras.

Otra de las características de Flask que lo hacen atractivo para proyectos de desarrollo es la facilidad con la que se agregan o intercambian estos componentes, ya que se pueden tomar decisiones en el proceso de desarrollo pudiendo reutilizar gran parte del código, por ejemplo

MySQL	MongoDB
Table	Collection
Row	Document
Column	Field
Joins	Embedded documents, linking

Cuadro 3.1: *Comparación de conceptos MongoDB vs BDD relacionales..*

un cambio de que base de datos utilizar afectaría al servidor únicamente en modificar la extensión a utilizar.

3.10. MongoDB

Hay muchas bases de datos de las que elegir. La razón de esta cantidad de opciones es que difieren en sus objetivos: algunas son buenas realizando consultas complejas, otras tienen un tiempo de consultas muy bajo, algunas pueden escalar fácilmente y llegar a contener petabytes de información. Estas son una parte importante en cualquier aplicación y la gran decisión a tomar es la de utilizar una clásica base de datos relacional o el nuevo tipo, usualmente llamadas "NoSQL". MongoDB es una base de datos NoSQL cuya etimología proviene de la palabra "humongous" que significa enorme, que permite desarrollar soluciones ágiles y escalables, características que se adhieren perfectamente con el cometido de este proyecto. A diferencia de las bases de datos relacionales, guarda los datos en documentos en formato JSON con un esquema dinámico. Esto significa que no se debe definir la estructura de estos documentos antes de utilizar la base de datos y que además los mismos pueden ir variando con el tiempo, posibilitando la reutilización de los datos si se decide agregar nuevos tipos de elementos.

Algunos conceptos que se podrían comparar en estas arquitecturas son los siguientes [21]:

3.11. React

React es una librería de JavaScript desarrollada por Facebook para construir interfaces de usuario. Se basa en la construcción mediante componentes reutilizables, los cuales la librería se encarga de actualizar según cambie el estado al que están relacionados. Estas características hacen de React una opción muy atractiva a la hora de desarrollar aplicaciones web, ya que es fácil de debuggear el código y replicar estados de falla de las aplicaciones.

Otro aspecto de esta librería que la coloca en la cima de las opciones de las empresas, tanto startups como empresas del Fortune 500 [36], es la capacidad de construir aplicaciones móviles nativas con React Native [35], utilizando como lenguaje de desarrollo JavaScript [20], uno de los lenguajes más populares actualmente, además de las librerías React y React Native. La ventaja de esta forma de desarrollar aplicaciones móviles es que a diferencia de PhoneGap y otros wrappers de aplicaciones web, con react native no se construye una aplicación híbrida, sino que se desarrolla con las mismas piezas que utilizando Objective-C o Java, sólo que estos bloques se consolidan usando JavaScript y React. Esto permite acceder a más funcionalidades específicas de los dispositivos móviles y hace que la aplicación sea apreciablemente más veloz y performante.

3.12. Cron

Cron es un administrador de procesos que es utilizado en Unix. Se encarga de que estos procesos se corran a intervalos constantes de tiempo o a horas específicas ambos determinados por las personas que tengan permisos para crear o editar estos crones. Solo los usuarios que están en el archivo cron.allow son los que pueden crear crones en sus respectivos crontabs.

El archivo crontab contiene los comandos que se ejecutan, así como la hora o frecuencia con que se corren, como se puede ver a continuación:

```
12 22 * * * curl -H "Authentication-Token:  
f1a8659701bbc7196940761c1d55c3b9a25fb6569a863df" -X POST -d '{"state  
": "0", "device_id": "12395613"}' http://localhost:5000/set_led_state
```

Donde la estructura es la siguiente:

```
.----- minuto (0-59)
| .----- hora (0-23)
| | .---- dia del mes (1-31)
| | | .--- mes (1-12)
| | | | .-- dia de la semana (0-6) (domingo=0 o 7)
| | | |
* * * * comando a ejecutar
```

3.13. Apache HTTP Server

Apache [3] es un servidor web HTTP de código abierto que tiene más de 20 años de creación. Es extremadamente popular, siendo el servidor HTTP más usado y es altamente configurable. A su vez, es multiplataforma abarcando entre otras, a Unix. Al ser de código abierto y habiendo estado tanto tiempo en funcionamiento hace que sea una opción confiable del punto de vista de la seguridad y esto también hace que sea fácil conseguir ayuda al tener un problema.

Los carpetas básicas donde se guarda la información de configuración son: `sites-available`, `sites-enabled`. Archivos básicos de configuración son: `apache2.conf` (nombre en Ubuntu), `ports.conf`.

En el archivo `ports.conf` se puede encontrar los puertos en los que escucha el servidor Apache. Por defecto solo está habilitado el puerto 80.

El archivo `apache2.conf` puede contener toda la configuración del sistema. Por un tema de modularidad y orden se acostumbra a diseminar la configuración en otros archivos.

La carpeta `sites-available` contiene archivos que definen los hosts virtuales del servidor. El contenido de estos archivos son lo que direccionan el pedido del usuario cuando accede al servidor. Esto depende de cómo hace el llamado (URL, puerto, ruta).

La carpeta `sites-enabled` contiene los archivos disponibles de `sites-available`. Son los únicos que tiene en cuenta Apache.

Finalmente, hay una carpeta que no es de configuración pero es de gran importancia. Esta

es `/var/www/html/` que es donde se ubica todo el código del servidor.

3.14. Amazon EC2

Amazon EC2 [1]

Amazon Elastic Compute Cloud (Amazon EC2 [1]) es un servicio brindado por Amazon Web Services. Es un entorno virtualizado para trabajar en la nube. Se puede elegir de un conjunto de imágenes de máquinas virtualizadas con las cuales empezar a trabajar. A su vez, se brinda la posibilidad de elegir las características de esa computadora como: memoria, almacenamiento y capacidad de red. A esto se le llama tipo de instancia. También, cuenta con un firewall virtual que se le aplica a una o varias de esas máquinas virtuales. Se brinda una IP estática para ese equipo junto con un dominio.

Al firewall virtual se lo llama Amazon EC2 Security Groups y al crear uno se lo puede aplicar a varias instancias (computadoras virtuales). A este firewall se le pueden añadir un conjunto de reglas para que empleadas por varias instancias. Por defecto, solo viene habilitado el puerto 22 para que el administrador se pueda conectar por SSH al equipo.

Este servicio es ofrecido de forma gratuita por el primer año para computadoras con procesamiento, ram y red limitada. De esta forma, si se necesita un servidor con pocos requerimientos, se puede usar este servicio para virtualizar el contenido del mismo y no tener que preocuparse por la alimentación u otros elementos físicos del servidor.

3.15. Mycroft

Mycroft es el primer asistente virtual open source, y está diseñado para ser ejecutado en una computadora personal y hasta en un Raspberry Pi. Este software open source está a total disposición para ser modificado, extendido o mejorado. La instalación del mismo incluye algunos skills básicos, algunas órdenes que se pueden hacer incluyen:

- “Hey Mycroft, what time is it?”
- “Hey Mycroft, what time is it in Paris?”

- “Hey Mycroft, what time is it in Melbourne?”
- “Hey Mycroft, what’s the weather?”
- “Hey Mycroft, set an alarm for 30 minutes.”
- “Hey Mycroft, what’s my IP address?”
- “Hey Mycroft, tell me a joke.”
- “Hey Mycroft, tell me about chocolate.”
- “Hey Mycroft, tell me about Linux.”

Se pueden agregar nuevos skills del repositorio generado con las contribuciones de la comunidad. [24]

Capítulo 4

Marco Metodológico

4.1. Tareas comprendidas y no comprendidas

Tareas comprendidas:

- Componente principal con servicios web, endpoints exponiendo scripts de control y configuración de interruptores inteligentes, además de la inclusión del asistente personal.
- Creación de red con permisos de autenticación, permitiendo la interacción segura entre los componentes.
- Aplicaciones web y móvil para control y configuración de los periféricos.
- Firmware optimizado a bajo nivel para permitir su ejecución en los sistemas de recursos limitados.
- Investigación de tecnologías de comunicación, lenguajes de programación, microcomputadoras y controladores orientados a IoT.
- La presentación del dispositivo será en el protector normal del raspberry.

Tareas no comprendidas:

- Configuración inalámbrica de interruptores inteligentes.
- Proceso de flasheo en masa de módulos de comunicación.
- Compatibilidad de aplicación móvil con más de un SO.
- Plan de importación de hardware para producción.

- Investigación de patentes.

4.2. Entregables

Título	Descripción	Tareas comprendidas	Fecha de entrega
Entrega de Cálculos / Rutinas con justificación de avance de 50 %	Conectividad de interruptores y central de control.	Se investigaron las opciones para implementar la comunicación y control de los dispositivos sin una interfaz amigable y desarrollo del servidor web para conexión con Sonoff. Pruebas con Sonoff.	15/11/2017
Entrega Borrador del Informe del PFC	Realización de documento donde se registran las investigaciones, decisiones y desarrollos.	Se realizó la conexión con la aplicación móvil, así como una implementación mínima del asistente personal.	14/12/2017
Entrega de Cálculos Rutinas con justificación de avance de 100 %	Prototipo del sistema finalizado	Conexión de los periféricos con el servidor central, órdenes de servidor para apagar y prender los periféricos, integración del servidor con el asistente personal, descubrimiento de todos los dispositivos de la red, scipts de voz generales.	14/02/2018

Cuadro 4.1: *Entregables: descripción y fecha*

4.3. Restricciones: Tiempo, presupuesto, tecnología

Entre las restricciones más relevantes para el alcance del proyecto se encuentran la necesidad de obtener como resultado un interruptor inteligente de tamaño reducido, capaz de ser colocado en el espacio destinado a las llaves de corriente convencionales y coste competitivo con los productos en el mercado actual.

Sumado a esto, se tiene que los módulos de comunicación que cumplen las características anteriores utilizan protocolos en desarrollo, algunos de los cuales funcionan con frecuencias similares a las reservadas para otros servicios, por lo que se deberá estudiar cuál de estos es posible utilizar en Uruguay.

Otra consideración a tener en cuenta es el rango máximo de comunicación entre dispositivos, siendo que dependiendo del protocolo y módulo elegido se deberá balancear entre velocidad de transmisión, penetración en estructuras físicas (paredes, muebles, rejillas) y consumo de energía.

Se deberá utilizar hardware y software con licencias de libre modificación y distribución, ya que el uso de productos pagos repercutiría de manera negativa en la accesibilidad del producto al público en general, provocando que el impacto en la sociedad.

Al contar con un límite de 4 pedidos de un máximo de USD 200, esta realidad conlleva a que, en caso de no ser capaces de incluir algún módulo o componente, nos veremos obligados a descartar dicha alternativa.

4.4. Supuestos y riesgos

El itinerario y alcance del proyecto se planean con el supuesto de la puntualidad de los pedidos al exterior; ya que, tratándose de state-of-the-art hardware, el mismo no está disponible en Uruguay, o en caso de estarlo, su precio es varias veces mayor al cual se comercializa en el exterior.

Dentro de los posibles interesados en el proyecto se encuentra UTE. Al ser un ente público se consideró que puede demorar en brindar la información que se solicita para adaptar el

modelo a este interesado. Si este fuere el caso, se desistirá de este ente para realizar en el modelado del producto.

También es posible que alguno de los componentes se rompa por un mal uso dada la poca experiencia con este tipo de dispositivos o que el mismo venga fallado. Si esto sucediese se deben comprar dispositivos nuevos lo cual conllevará contratiempos por el envío de los materiales. Para aliviar riesgo, dado que los elementos de este proyecto no son caros y pueden ser reusados, se compraran de forma redundante para que el fallo en alguno de los elementos no golpee al proyecto.

4.5. Estructura de Desglose de Trabajo (EDT)

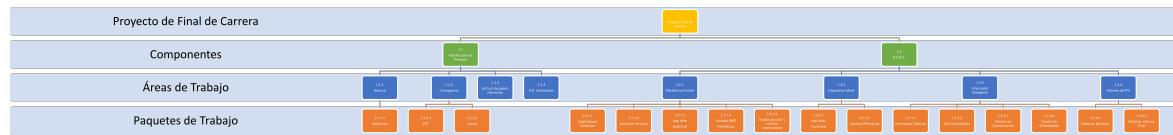


Figura 4.1: *Estructura de Desglose de Trabajo*.

4.6. Desglose de Paquetes de Trabajo en Actividades

4.7. Procedimientos

4.7.1. Protocolos

La elección de protocolos consistió en dos rondas, donde el primer acercamiento fue absolutamente teórico, se basó en la investigación por medio de papers y de la documentación oficial en Internet de los protocolos, para descartar varios de ellos, y así reducir el número de posibilidades. Luego se investigó implementaciones y hardware disponible para los protocolos elegidos, siendo esta una investigación más práctica, donde se tuvo en cuenta la calidad y madurez de las librerías desarrolladas para cada uno, así como el acceso y precio de los módulos correspondientes. Otro punto importante al momento de decidir fue la

comunidad creada alrededor de los protocolos y plataformas de desarrollo, ya que se sabía que su apoyo sería indispensable para reducir la curva de aprendizaje y evitar caer en errores comunes sin tener una fuente rápida y efectiva para despejar dudas.

4.7.2. Librerías

Respecto a las librerías, se buscó trabajar librerías maduras y mantenidas, que no estén abandonadas por sus mantenedores y cuyos issues hayan o estén siendo resueltos. La razón de por qué utilizar librerías y no desarrollar las funcionalidades por nosotros mismos son simples, en el caso de encontrar una librería que se adecúe a las necesidades del proyecto y con las características buscadas mencionadas, no tiene caso reinventar la rueda. Esto es teniendo en cuenta la revisión de las mismas y que hay un entendimiento de cómo funcionan, y no es simplemente una caja negra. Es una de las intenciones la mejora de las mismas, ya que se interactuó con sus mantenedores, abriendo issues y ayudando a la mejora de estas. El procedimiento para la investigación y adaptación de las librerías en el sistema sigue los siguientes lineamientos:

- Análisis del estado de la librería:
 - Cantidad de commits.
 - Issues abiertos y resueltos.
 - Últimas actualizaciones.
 - Cantidad de versiones.
 - Comentarios y stars que tenga el repositorio.
 - Calidad de documentación.
 - Cantidad y calidad de ejemplos.
 - Menciones y utilización de la librería en proyectos.
 - Requerimientos de hardware.
- Revisión de código y ejemplos:
 - Entendimiento general de las funciones brindadas por la librería.

- Lectura de ejemplos relevantes para el proyecto, usualmente comenzando con el "Hello World", para entender el uso básico de la librería y luego proseguir con uno que utilice las funcionalidades que se utilizarán en el proyecto.
- Instalación de la librería y experimentación con ejemplos:
 - Instalación de librería en un ambiente virtual para evitar conflictos de librerías utilizadas por la que se está investigando y las que utilizan distintas partes del sistema.
 - Utilizar el ejemplo básico para comprobar que la instalación fue correcta.
 - Experimentar con el ejemplo que utilice las funciones en las que se está interesado.
 - Implementar una solución para el sistema con la librería.
 - Acoplar dicha solución al sistema.

4.7.3. Hardware

Los procedimientos de desarrollo con hardware fueron los menos consistentes debido a las grandes diferencias de los mismos, por lo que se detallan por separado.

Raspberry Pi

El componente era previamente conocido y es el más fácil de manejar en este contexto, ya que es posible utilizar git para clonar y actualizar el código existente en el componente. Además de esto es posible modificar los archivos desde una sesión ssh, permitiendo probar cambios en el dispositivo antes de realizar commits. El procedimiento que se siguió para desarrollar código en este dispositivo es:

- Desarrollo y testeo de código en el computador personal.
- Si se está desarrollando un nuevo módulo: subir el código a GitHub y hacer un pull desde el Raspberry Pi.
- En caso de realizar modificaciones pequeñas, se copia el código por ssh al archivo del Raspberry Pi y se testea antes de versionar los cambios.

- Una vez terminado el testeo, proceder a realizar un Pull Request y volver a hacer un pull de la branch dev en el Raspberry Pi.

Sonoff

Los pasos para su preparación fueron los siguientes:

- Compra y espera de entrega. Se realizaron dos compras, una express de 2 unidades y otra de envío gratuito con 9 unidades, una de ellas un Sonoff Dual.
- Adaptación física para desarrollo, se soldaron las conexiones y se confeccionó un switch físico para poder reiniciar fácilmente el dispositivo.
- Una vez conectado por el adaptador USB/Serial, se borran los SPIFFS con la herramienta esptool.
- Se resetea el Sonoff con el botón presionado para entrar en modo configuración.
- Con Atom y el plug-in de PlatformIO se flashea el firmware desarrollado al componente.

Capítulo 5

Estructura del Contenido

5.1. Arquitectura

El proyecto cuenta con tres módulos principales, cuyos componentes son parte exclusiva de uno de los módulos o, en caso del servidor Flask, de todos ellos. Este servidor funciona como puente y permite la unificación de los 3 módulos en un único sistema. Un factor decisivo a la hora de elegir esta arquitectura fue la seguridad de la red MQTT.

Añadir seguridad en redes de IOT es un tema complejo dada la naturaleza limitada de las redes y dispositivos que las integran. El costo de overhead y procesamiento inherente a la seguridad en el intercambio de mensajes entre broker y los nodos puede ser mayor al beneficio obtenido. Es por esto que se debe analizar alternativas al uso de SSL/TLS de los que se habló anteriormente.

Si el diseño de la arquitectura de la red MQTT es malo y, además, no se utiliza SSL/TLS, es decir, los mensajes se mandan en texto plano, es muy simple que un hacker inyecte mensajes en la red. [28] Por ejemplo, si el broker estuviera conectado a los nodos en una red wifi abierta, un hacker podría conectarse a la red, sniffear los mensajes, ver el formato de los mensajes entre broker y nodos logrando así entender que topics y payloads se utilizan para realizar ciertas acciones como prender y apagar el relé y publicar las acciones en vez del broker. A través de este ejemplo es fácil darse cuenta es extremadamente importante tener una contraseña para la red MQTT y con un nivel de seguridad alto como WPA/WPA2, para que sea computacionalmente difícil de que un hacker se conecte a la red e infecte mensajes. [42]

Otro posible problema es que la red MQTT sea la misma que la red de la persona, lo que puede dar lugar a que si esta persona no tiene una buena contraseña en su red, esté expuesta a que un hacker pueda conectarse a la red utilizando contraseñas populares o que

una persona que se haya conectado alguna vez tenga luego malas intenciones.

Para mejorar las arquitecturas de seguridad mencionadas anteriormente se propone:

- Tener 4 contraseñas involucradas en arquitectura:
 1. La que se debe crear el usuario para registrarse en la base de datos mongo y así tener acceso a las funciones de sistema como el apagado y prendido de los relé. Esta autenticación hace que no cualquier persona en la red local del hogar pueda tener acceso a esas funciones, solo los que se autentifican.
 2. La segunda contraseña es la de la red local de la casa a la cual se debe conectar el Raspberry Pi luego de la instalación inicial.
 3. La tercera es la de la red MQTT. Esta red tendrá un nombre y contraseña luego de la instalación del sistema de la cual el usuario nunca se deberá preocupar, será transparente para el mismo. Esta contraseña es elegida con una combinación entre la MAC del Raspberry Pi donde se alberga el broker y la hora a la que se crea la misma. Todos los nodos que se conecten tendrán que guardar esa contraseña en sus SPIFFS y adquirirla de ahí al prenderse para conectarse.
 4. La última contraseña es con la que se comunican entre los nodos entre ellos cuando alguno no llega a recibir la señal del broker. Esta contraseña también debe ser única y crearse con un criterio similar a la de la red MQTT. Esta contraseña permite que si dos clientes independientes instalarán el producto en lugares cercanos, los mensajes de uno y otro no se mezclen. De la misma manera, un hacker tampoco podría ingresar mensajes a la red usando nodos con el software del sistema.
- Como se puede ver en la figura [5.1], se hace una separación tanto física como lógica de la red MQTT con la red del usuario. Una está dedicada a la red MQTT y otra es la red local del usuario que manda las órdenes al Flask y este traduce a órdenes MQTT. Esta arquitectura resuelve los problemas expuestos anteriormente dado que se tiene control sobre la red MQTT. Esta red es transparente para el usuario y esto permite que se puedan crear una contraseña WPA/WPA2 muy segura utilizando la dirección MAC del dispositivo y la hora de creación de la misma, como se explicó anteriormente. Crear una red MQTT separada de la red del usuario no solo mejora la seguridad. Mientras menos se dependa de una red local donde personas ajenas al proyecto pueden cambiar su configuración, mayor control se tiene sobre el ambiente en el que corre el sistema.

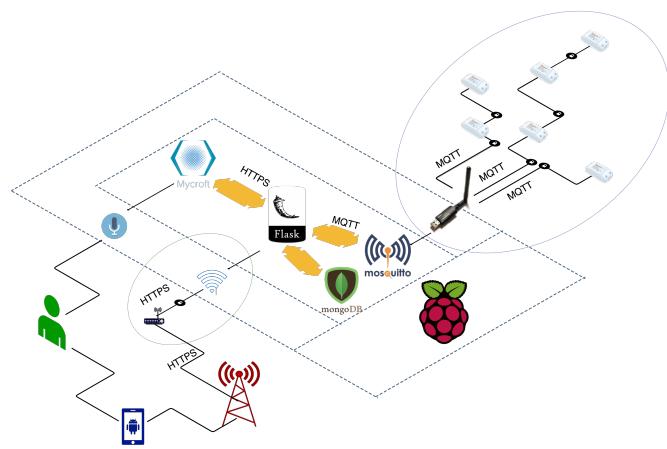


Figura 5.1: Redes en el sistema.

5.2. Módulos del Sistema

5.2.1. Módulo MQTT Mesh

Componentes:

- Interruptores Sonoff
- Raspberry Pi: Específicamente el access point configurado en el dongle Panda, el broker Mosquitto y el servidor Flask.

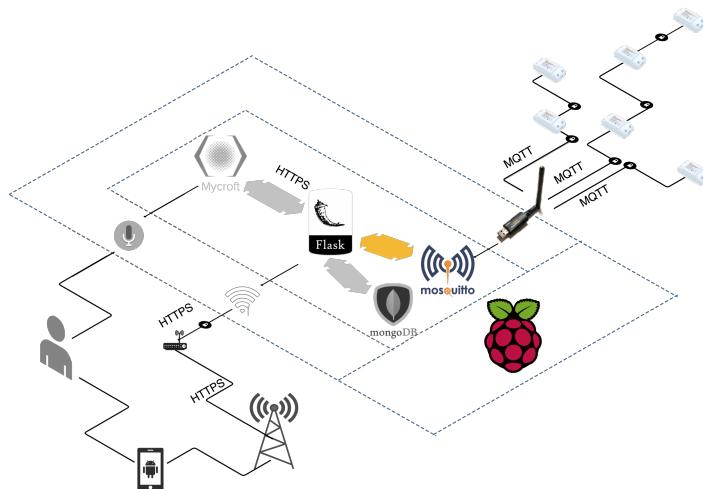


Figura 5.2: Componentes del Módulo MQTT Mesh.

Descripción:

Este módulo es la base del sistema y su funcionamiento es el siguiente:

- El broker Mosquitto se encarga de la distribución de los mensajes MQTT a los componentes conectados al Raspberry Pi por el Access-Point configurado con el dongle.
- El software en los interruptores sonoff permiten la conexión al broker directamente o a través de otro interruptor perteneciente a la misma red. Estos dispositivos son

configurables a través de la aplicación web, permitiendo así configurar tanto el nombre como contraseña de la red.

- El servidor Flask se suscribe y publica en topics, siendo el único componente de la red además de los interruptores inteligentes.

5.2.2. Módulo de Control

Componentes:

- Raspberry Pi: Específicamente la base de datos MongoDB y el servidor Flask.

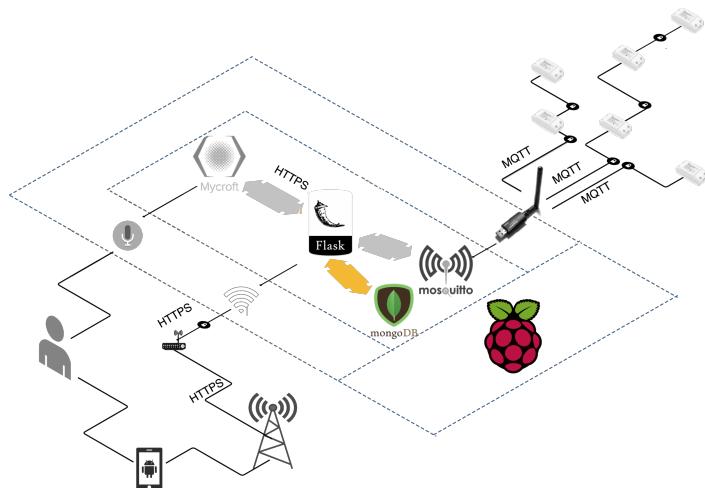


Figura 5.3: Componentes del Módulo de Control.

Descripción:

El servidor Flask se conecta a la red MQTT, permitiendo así suscribirse y publicar en distintos topics. Este servidor será accesible por las interfaces de usuario como único punto de acceso a los componentes de la red MQTT, se encargará de traducir los mensajes enviados por estas interfaces y de manejar otras estructuras de datos necesarias para la interacción con los componentes. Llevará un registro en la MongoDB de todos los componentes conectados

al broker, junto con los topics para poder interactuar con ellos y a los demás datos de cada componente, así como nombre, habitación o grupo al que pertenecen.

5.2.3. Módulo de interfaces de usuario

Aplicación web

Componentes:

- Aplicación contenida en el dispositivo móvil.

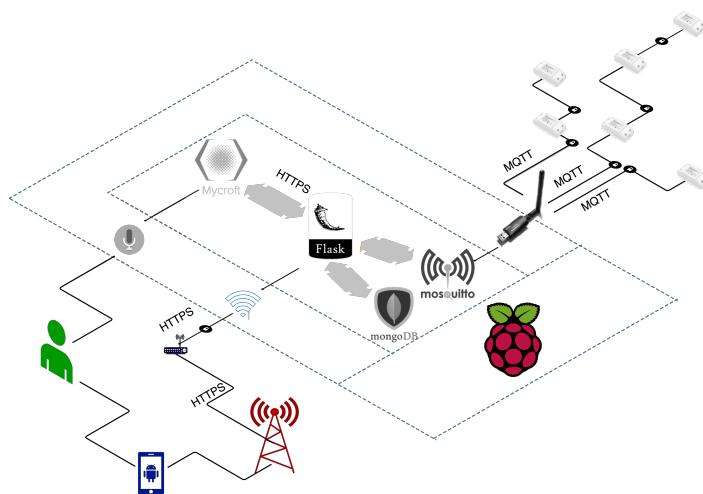


Figura 5.4: Componentes del Módulo de UI para Aplicación Móvil.

Descripción:

- Esta aplicación permite interactuar de una manera sencilla e intuitiva con los servicios proveídos por los módulos anteriormente mencionados.
- Una de las funciones básicas es conectarse a los interruptores inalámbricos en modo configuración, y así modificar las credenciales necesarias para acceder a la red proveída por el Raspberry Pi.

- También expone los dispositivos pertenecientes a la red, permitiendo apagar, prender y programar una de las anteriores acciones para un momento determinado.

Asistente virtual

Componentes:

- Asistente Mycroft, localizado en Raspberry Pi.
- Micrófono, conectado al Raspberry Pi.

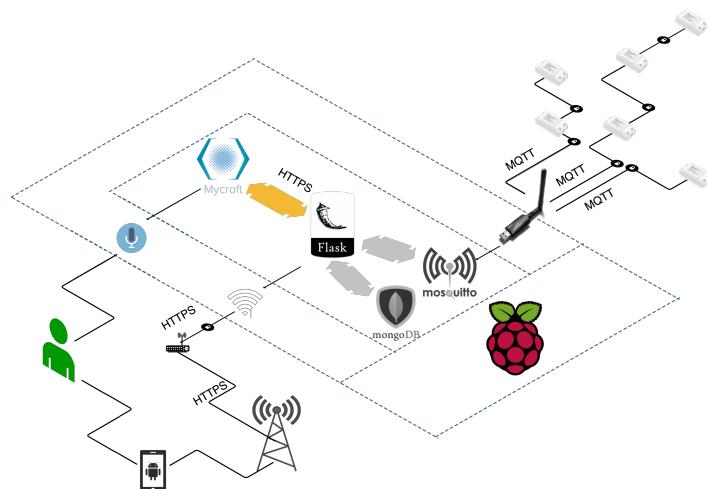


Figura 5.5: Componentes del Módulo de UI para Asistente Virtual.

Descripción:

- Utilizando Mycroft, se permite el control a través de voz a los dispositivos ya configurados en el sistema.

5.3. Desarrollo

5.3.1. Módulo MQTT Mesh

Este módulo cuenta con el broker mosquitto localizado en el Raspberry Pi y el firmware desarrollado para este proyecto en los dispositivos Sonoff.

Mosquitto broker:

Primero se debe acceder al Raspberry Pi a través de ssh e instalar los componentes:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install mosquitto
$ sudo apt-get install mosquitto-clients
```

Luego se debe realizar un link simbólico del archivo de configuración proveído en el repositorio de GitHub del proyecto, dentro de `MQTT_Server/mosquitto`. Para lograr esto, se debe posicionar en la carpeta del Raspberry Pi que contiene el archivo de configuración creado en la instalación de mosquitto, la dirección predeterminada es `/etc/mosquitto`, luego se debe borrar este archivo y crear un link con el archivo anteriormente mencionado.

```
$ sudo rm mosquitto.conf
$ sudo ln -s <cloned repo directory>/Tesis/MQTT_Server/mosquitto/
    mosquitto.conf mosquitto.conf
```

El contenido del archivo al cual se crea el link es el siguiente:

```
pid_file /var/run/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log
allow_anonymous false
password_file /etc/mosquitto/pwfile
listener 1883
```

Algunos puntos importantes de esta configuración son:

persistence : Si es seteado a `True`, toda conexión suscripción y mensajes serán escritos en el disco, en una base de datos llamada `mosquitto.db`, localizada en la carpeta indicada por `persistence_location`. Cuando mosquitto se reinicia, carga toda la información contenida en esta base de datos. Los datos se guardan en la base de datos cuando mosquitto se cierra o en intervalos de tiempo definidos por `autosave_interval`. Se puede forzar un guardado en la base de datos enviando a mosquitto la señal `SIGUSR1`.

allow_anonymous : También de valor booleano, determina si los clientes deben proveer o no nombre de usuario y contraseña al conectarse. Si se setea a `false`, se debe crear otro medio de autenticación para el acceso de clientes. Su valor predeterminado es `true`.

Al indicar que sólo aceptaremos conexiones que brinden credenciales, debemos crear la contraparte verificadora de las mismas en el broker, para esto, se debe correr el siguiente comando:

```
$ sudo mosquitto_passwd -c /etc/mosquitto/pwfile <username>
```

Para testear que esto haya funcionado correctamente se puede inicializar un cliente con las credenciales configuradas en el archivo recién creado de la siguiente forma:

```
$ mosquitto_sub -d -u <username> -P <passwd> -t test/topic
```

Y se debería obtener el siguiente resultado:

```
$ Client mosqsub/8161-picroft sending CONNECT
$ Client mosqsub/8161-picroft received CONNACK
$ Client mosqsub/8161-picroft sending SUBSCRIBE (Mid: 1, Topic: /test/
topic, QoS: 0)
$ Client mosqsub/8161-picroft received SUBACK
$ Subscribed (mid: 1): 0
```

Luego utilizando la misma herramienta publicar un mensaje en el topic al que nos suscribimos (utilizaremos las mismas credenciales ya que son las únicas que se configuran, aunque esto no es necesario) :

```
$ mosquitto_pub -d -u <username> -P <passwd> -t test/channel -m
whatever
```

En la consola del cliente suscrito se debería obtener:

```
$ pi@picroft:~ $ mosquitto_pub -d -t /test/channel -m whatever
Client mosqpub/8164-picroft sending CONNECT
Client mosqpub/8164-picroft received CONNACK
Client mosqpub/8164-picroft sending PUBLISH (d0, q0, r0, m1, '/test/
channel', ... (8 bytes))
Client mosqpub/8164-picroft sending DISCONNECT

$ pi@picroft:/etc/mosquitto $ mosquitto_sub -d -t /test/channel

Client mosqsub/8163-picroft sending CONNECT
Client mosqsub/8163-picroft received CONNACK
Client mosqsub/8163-picroft sending SUBSCRIBE (Mid: 1, Topic: /test/
channel, QoS: 0)
Client mosqsub/8163-picroft received SUBACK
Subscribed (mid: 1): 0
Client mosqsub/8163-picroft received PUBLISH (d0, q0, r0, m0, '/test/
channel', ... (8 bytes))
whatever
```

Access Point:

Se conectó un dongle de wifi al Raspberry Pi para poder crear una red dedicada a la comunicación entre los elementos de la red, ya que se quería independizar la misma de la red wifi del lugar de instalación. [38]

- Primero instalar componentes:

```
sudo apt-get update
sudo apt-get install hostapd isc-dhcp-server
sudo apt-get install iptables-persistent
```

Aparecerán dos ventanas de configuración, elegir "Yes." en las dos.

- Configurar el DHCP server Editaremos el archivo /etc/dhcp/dhcpd.conf.

```
sudo nano /etc/dhcp/dhcpd.conf
```

Encontrar las siguientes líneas y comentarlas:

```
option domain-name "example.org";  
option domain-name-servers ns1.example.org, ns2.example.org;
```

Encontrar las siguientes líneas y descomentar .^authoritative":

```
# If this DHCP server is the official DHCP server for the local  
# network, the authoritative directive should be uncommented.  
# authoritative
```

Por último se recorre hasta el final del archivo y se agrega:

```
subnet 192.168.42.0 netmask 255.255.255.0 {  
    range 192.168.42.10 192.168.42.50;  
    option broadcast-address 192.168.42.255;  
    option routers 192.168.42.1;  
    default-lease-time 600;  
    max-lease-time 7200;  
    option domain-name "local";  
    option domain-name-servers 8.8.8.8, 8.8.4.4;  
}
```

Ejecutar:

```
sudo nano /etc/default/isc-dhcp-server  
Y buscar INTERFACES="", agregar el nombre de la interfaz del  
adaptador de WiFi.
```

- Luego se fija una IP estática para esta interfaz, para esto debemos modificar el archivo /etc/network/interfaces:

```
sudo nano /etc/network/interfaces
```

Se agrega las siguientes líneas luego de allow-hotplug wlan0:

```
iface wlan0 inet static  
address 192.168.42.1  
netmask 255.255.255.0
```

- Configurar Access Point.

Se crea un nuevo archivo `/etc/hostapd/hostapd.conf` con la configuración:

```
interface=wlan0
driver=n180211
ssid=Pi_AP
country_code=US
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=Raspberry
wpa_key_mgmt=WPA-PSK
wpa_pairwise=CCMP
wpa_group_rekey=86400
ieee80211n=1
wme_enabled=1
```

Por último se debe indicar al RPi dónde encontrar el archivo de configuración:

```
sudo nano /etc/default/hostapd
```

En la línea que dice `#DAEMON_CONF=""` sustituir por:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

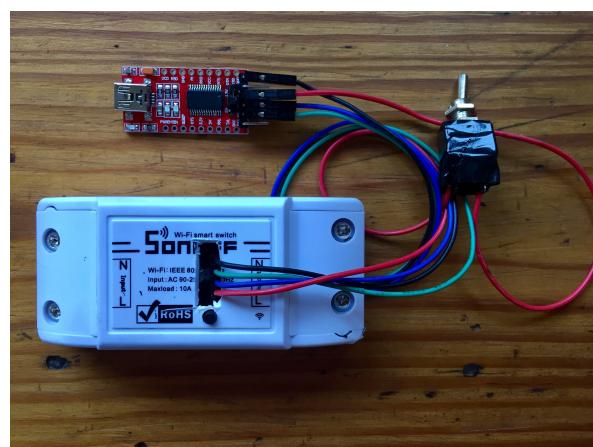
Dispositivos Sonoff

Modificaciones físicas y subida de firmware A pesar de que este componente está pensado para ser hackeado, es necesario realizarle algunas modificaciones físicas poder modificar el firmware del mismo.

En la figura [5.8] podemos apreciar que los conectores del chip ESP8266 integrado en el Sonoff son fácilmente accesibles pero no cuentan con cabezales para conectar los cables del adaptador Serial-USB, por esta razón es necesario soldar los mismos. Luego se debe conectar al adaptador USB/Serial para poder subir el firmware desde un computador.



Figura 5.6: Conexiones del ESP8266 expuestas por el Sonoff.



(a)) Vista superior..



(b)) Vista de lado.

Figura 5.7: Sonoff modificado.

Adaptación de interruptores analógicos

Debido a que se busca la mayor comodidad del lado del usuario y se intenta que la

integración del sistema no conlleve a limitaciones, se configuraron los dispositivos Sonoff para poder ser controlados por interruptores analógicos. Ya sea del tipo utilizado comúnmente en instalaciones eléctricas como interruptores digitales. Esto permitirá que el usuario utilizar tanto los medios usuales como las interfaces desarrolladas para interactuar con el sistema. Esto se logra conectando el interruptor entre el pin `GPIO14` del Sonoff y tierra, logrando así una interacción lógica con el relay interno.



Figura 5.8: Conexión de interruptores con Sonoff.

Esto permite que si el usuario apaga las luces con llave en la pared, luego las enciende con la aplicación y a continuación oprime la llave en la pared, la luz volverán a apagarse. Este comportamiento permite una interacción más natural e intuitiva, siendo la falta de esta funcionalidad una debilidad clave en las soluciones de la competencia.

Para lograr que el interruptor tenga comportamiento de botón que cambie de estado el relay, se debió configurar el pin en modo `INPUT_PULLUP`, para luego vincular este pin con una interrupción que dispare la función `switch_state_changed`. Esta función cambia de estado al relay y envía una notificación al servidor central para informar el estado del dispositivo. Este mensaje se envía en el topic `device_out/sub_topic/state_changed` siendo la estructura del mensaje enviado:

```
{
  "_id": {_id},
  "state": "0/1"
}
```

La configuración y vinculación de la interrupción se logran con las siguientes líneas:

```
//Setup the Switch :
pinMode(SWITCH_PIN, INPUT_PULLUP);
```

```
attachInterrupt(digitalPinToInterrupt(SWITCH_PIN), switch_state_changed
, CHANGE);
```

Como se puede observar la función se ejecuta con la condición CHANGE, por lo que no diferencia entre interruptor cerrado o abierto. Observando la función llamada, podemos notar algunas singularidades, por ejemplo, es necesario utilizar una lógica de semáforo por temporizador, ya que debido a la imperfección de los interruptores analógicos, el chip detecta diferentes cantidades de cambios en el interruptor antes de cambiar de estado, esto es conocido como efecto rebote.

```
void switch_state_changed () {
    unsigned long interrupt_time = millis();
    if (interrupt_time - previousInterrupt > 200)
    {
        previousInterrupt = interrupt_time;
        int state = digitalRead(RELAY_PIN);
        digitalWrite(RELAY_PIN, !state);
        char msg[255];
        strlcpy(msg, "{", sizeof(msg));
        strlcat(msg, "\"_id\":", sizeof(msg));
        strlcat(msg, ID.c_str(), sizeof(msg));
        strlcat(msg, ",\"state\":", sizeof(msg));
        if(state==0)strlcat(msg, "\"0\"", sizeof(msg));
        if(state==1)strlcat(msg, "\"1\"", sizeof(msg));
        strlcat(msg, "}", sizeof(msg));
        mesh->publish(
            "state_changed",
            msg
        );
    }
}
```

Cabe mencionar que este efecto no es provocado solamente por cambios de estado del interruptor externo conectado al Sonoff, sino que al realizar pruebas y modificar el estado del relay desde la aplicación móvil, se descubrió que al prenderse el relay, los cambios de tensión disparaban la interrupción. Por lo que se debió activar el semáforo en caso de recibir órdenes remotas, para evitar que esta situación entorpezca el funcionamiento del dispositivo.

```

if(sTopic == "Power"){//Relay HIGH to turn on
    previousInterrupt = millis();
    ...
}

```

Análisis de Código

Librería FS [3.5.1]

Esta librería es usada por la librería ESP8266MQTTMesh [??] y también en el código que se realizó. Para lo primero que se utilizó como se puede ver a continuación, es para leer de los SPIFFS y así ver si existe una variable llamada `mqtt_server` donde se almacena la IP del broker MQTT. Si no encuentra este archivo se conecta a la red MQTT con la IP por default que tiene el broker en el archivo `credentials.h`.

```

broker=getInformationFromSPIFFS("/broker/");
if(broker!=""){//getInformationFromSPIFFS("/broker/")
    mqtt_server=broker.c_str();
} else{
    mqtt_server= mqtt_default_server;
}

String getInformationFromSPIFFS(String baseString){
    Dir dir = SPIFFS.openDir(baseString);
    if(!dir.next()){
        return "";
    } else{
        File brokerFile=SPIFFS.open(dir.fileName(), "w");
        char* str=(char*)brokerFile.name();
        char * pch;
        pch = strtok (str,"/");
        int counter=0;
        String broker;
        while (pch != NULL) {
            if(counter==1){
                broker=String(pch);

```

```

        }

        Serial.println("entre");
        pch = strtok (NULL, "/");
        counter=counter+1;
    }

    return broker;
}

}

```

Si se quisiera, también se podría usar el mismo concepto con otras variables que se encuentran en el archivo `credentials.h` como `MESH_PASSWORD`, `MQTT_DEFAULT_SERVER`, `NETWORK_LIST`. Esto hace que los valores del archivo `credentials.h` sean flexibles.

Esta librería también se utiliza para variables que no están en `credentials.h`. Por ejemplo, para determinar si dejar pasar la electricidad en el relé al momento de prender el Sonoff. Este se fija si existe la el directorio `/ON_When_Power_On/`, si existe, deja pasar la electricidad. De la misma manera se fija si existe el directorio `/OFF_When_Power_On/` para determinar no deja pasar la electricidad. Se utiliza el método `SPIFFS.exists(path)` de forma trivial. Ya se explicó anteriormente su funcionamiento básico.[[3.5.1]]

```

boolean bONExists = check_if_power_ON_Exists();
boolean bOFFExists = check_if_power_OFF_Exists();

//check if poweroff or on state exists in spiffs
if (!bONExists && !bOFFExists) {
    File f = SPIFFS.open("/ON_When_Power_On/", "w"); //por defecto esta ON
    f.close();
    digitalWrite(RELAY_PIN, HIGH);
} else {
    if (bONExists) {
        digitalWrite(RELAY_PIN, HIGH);
    }else{
        if (bOFFExists) {
            digitalWrite(RELAY_PIN, LOW);
        }
    }
}

```

Librería Bounce2

Esta es una librería de Arduino que sirve para cubrirse de los problemas típicos de rebotes asociados a la utilización de botones en el hardware. Se setea el pin asociado al botón como una entrada pull up de la siguiente manera:

```
pinMode(BUTTON_PIN, INPUT_PULLUP);
```

Luego se crea el objeto asociado a esta librería:

```
Bounce debouncer = Bounce();
```

Se crea un vínculo entre el objeto y el pin asociado al botón y se setea el tiempo de rebote en milisegundos:

```
debouncer.attach(BUTTON_PIN);
debouncer.interval(5); // interval in ms
```

Dentro del loop se actualiza el objetivo para que tome el valor del pin donde está el botón y se lee ese valor:

```
debouncer.update();
int value = debouncer.read();
```

Esto se usa para ver si el usuario aprieta el botón más de 2 segundos, si esto pasa es porque se quiere entrar en el modo de configuración del Sonoff.

Librería ESP8266MQTTMesh

Esta librería es la base del proyecto. La misma se encarga de setear una IP única para cada nodo de la red. Hace esto basándose en una propiedad que tiene el broker mosquitto que consiste en que cada vez que un dispositivo se suscribe a él. Este publica la bssid de todos los nodos que se han conectado. La bssid se crea en base a la MAC del dispositivo que se suscribió con un número identificador de la siguiente forma:

```
Client mosqsub/2372-picroft received PUBLISH (d0, q0, r1, m0, '/device_in
/bssid/2E:3A:E8:11:60:A3', ... (1 bytes))
```

```

Client mosqsub/2372-picroft received PUBLISH (d0, q0, r1, m0, '/device_in
/bssid/5E:CF:7F:B4:69:06', ... (1 bytes))
5

Client mosqsub/2372-picroft received PUBLISH (d0, q0, r1, m0, '/device_in
/bssid/2E:3A:E8:0F:43:95', ... (1 bytes))
7

```

Es así que el nodo toma esas BSSID y ve si la suya coincide con alguna de las publicadas. Si es así, se setea la IP 192.168.<subdominio>.1 siendo subdominio el número único expuesto anteriormente. Si no coincide con ninguna, hace una iteración desde 4 hasta 256 hasta que encuentra un número no coincide con ninguno de los números únicos y se setea ese como subdominio. Esta IP es almacenada en las SPIFFS por si la próxima vez que se inicie no llega al broker para conseguir su IP, toma la que tiene guardada en SPIFFS.

En el nodo, la red mesh se inicializa de la siguiente manera:

```

mesh = ESP8266MQTTMesh::Builder(networks, network_password, (char*)
    mqtt_server, mqtt_port)
.setVersion(FIRMWARE_VER, FIRMWARE_ID)
.setMeshPassword(mesh_password)
.setBaseSSID(base_ssid)
.setMeshPort(mesh_port)
.setTopic(in_topic, out_topic)
.buildptr();
mesh->setCallback(callback);
mesh->begin();
}

```

Los argumentos de la función `Builder()` son:

- `networks`: es una lista de redes posibles para conectarse (generalmente es solo una), la misma debe terminar con un ítem vacío ()�.
- `network_password`: es la contraseña de una red.
- `mqtt_server`: es la IP del server MQTT.
- `mqtt_port`: es el puerto de esa IP al cual conectarse.

- `FIRMWARE_VER, FIRMWARE_ID`: los utiliza para ver si la versión de la librería es la correcta.
- `mesh_password`: es la contraseña con la que los nodos se comunican entre si cuando no alguno no tiene acceso al broker.
- `base_ssid`: es la `SSID` del nodo que luego comprarte con el broker.
- `in_topic, out_topic`: son los prefijo que siempre van a tener los mensajes a los que se suscribe y se publica respectivamente antes de conectarse al broker. Luego de conectarse a este, el prefijo pasa a ser:
 - `/<out_topic>/<BASE_SSID>-<DEVICE_#>` para publicar.
 - `/<in_topic>/<BASE_SSID>-<DEVICE_#>` para suscribirse.

Como se ve en el código, también se le setea una función de callback, que es a la que se llama cuando se reciben mensajes que le corresponden desde el broker. Las entradas de esta función son el topic y el payload del mensaje MQTT.

```
void callback(const char *topic, const char *msg) { }
```

La implementación de esta función chequea si el topic coincide con alguno de la tabla. [5.1] Si esto sucede, se verifica que el payload sea válido. Si es así, se realiza la acción que se especifica en la descripción.

Otra parte fundamental de esta librería es la conexión entre los nodos cuando alguno de ellos no llega al broker. Un nodo puede conectarse al broker directamente o a través de otro nodo. En este caso, el intermediario va a actuar de mensajero entre broker y nodo fuera del alcance del broker. Cada nodo puede cumplir la función de mensajero hasta con 4 otros nodos. Si un nodo pierde la conexión al broker o a su nodo mensajero, inmediatamente se desconecta de todos los nodos que estén mandando mensajes a través de él. Esto tiene un efecto en cascada. Un nodo solo permite que otros nodos los usen de mensajero si él tiene una conexión consolidada con el broker o con otro nodo que le hace de mensajero.

El comportamiento del nodo es escuchar los mensajes que le llegan desde el broker o el nodo que le este haciendo de mensajero, procesarlo y luego pasarlo a los nodos que están conectados a él. A su vez, este nodo escucha a los nodos que están conectados a él y reenvía

<i>Topic</i>	<i>Payloads válidos</i>	<i>Descripción</i>
LedPower	0 / off	Apaga el LED.
LedPower	1 / on	Prende el LED.
LedState		Publica si el estado actual del LED.
Power;jx;		Publica el estado actual del relé;jx;.
Power	0 / off	Apaga el relé;jx;
Power	1 / on	Prende el relé;jx;
PowerOnState		Publica el estado actual del relé.
PowerOnState	0 / off	Keep relay(s) off after power on
PowerOnState	1 / on	Turn relay(s) on after power on

Cuadro 5.1: *Tabla de topics de comunicación con dispositivos Sonoff*

los paquetes que le llegan al broker o nodo mensajero. Por lo tanto los nodos envían mensajes directamente al broker o lo hacen a través de un nodo mensajero.

El protocolo que usa un nodo para mandarle mensajes a otro es TCP/IP. Se manda el topic y mensaje en el payload del paquete. [??]

Librería WiFiManager

Esta librería fue usada para recibir los datos de configuración inicial de SSID y contraseña de la red MQTT del RPi de forma transparente para el usuario. La forma en la que se logra esto es al almacenar esos datos en el dispositivo del usuario. Cuando se configura por primera vez un dispositivo, se mandan en formato JSON al mismo.

Lo primero que hace la librería cuando se llama dentro del Sonoff es levantar una red Wifi llamada AutoConnectAP para que el usuario se conecte a la misma. Se implementó que solo se llame a la librería cuando no se tiene registro en los SPIFF de que hubo una conexión

a la red MQTT.

```
boolean check_if_first_connection_Exists() {
    if (SPIFFS.exists("/first_connection/")) {
        Serial.print("first_connection OK");
        return true;
    } else {
        return false;
    }
}

boolean bFirstConnection = check_if_first_connection_Exists();
WiFiManager wifiManager;

if (!bFirstConnection) {
    wifiManager.setAPCallback(configModeCallback);
    wifiManager.autoConnect("AutoConnectAP");
}
```

La librería bloquea hasta que el usuario ingrese datos válidos para la conexión a una red Wifi, también levanta una interfaz gráfica en el dispositivo del usuario para se ingresen estos datos. Sin embargo, en la solución de este producto no se quiere que el usuario ingrese los datos ni que se levante una interfaz gráfica, dado que eso lo manda directo el dispositivo de forma transparente para el usuario.

Por este motivo, se utilizó una versión modificada de la librería que permite seleccionar si se desea activar la interfaz web al realizarse una conexión a la red Wifi del Sonoff. La modificación consta de la siguiente línea.

```
wifiManager.setCaptivePortalEnable(false);
```

Sin embargo, de esta manera todavía no se puede mandar los datos desde el dispositivo del usuario dado que la librería no posibilita llamados AJAX por restricciones de CORS. Por lo tanto, se debió modificar el código fuente de la librería para que los aceptara. Se hizo de la siguiente manera:

```
server->sendHeader("Access-Control-Allow-Origin", "*");
```

Al mandar ese encabezado en la request del servidor, el dispositivo está habilitado para

mandar el AJAX por más que no haya pedido la interfaz web al servidor para setear los datos.

Cuando le llega ese AJAX al dispositivo, verifica si la `SSID` y contraseña que se le manda son válidos para conectarse a una red y en caso de serlo, escribe en los SPIFFS que pudo conectarse, así como esa `SSID` y contraseña y se reinicia. Cuando se vuelve a iniciar, como lee de los `SPIFF` que se pudo conectar una vez, no llama a la librería Wifi Manager y obtiene el usuario y contraseña de los `SPIFF`.

Acciones Programadas

Se eligió la opción de crear crons a la hora de realizar el scheduler. Se tomó esta decisión por dos motivos:

- Los crones son algo fácil de manipular en linux dado que están en un archivo de texto.
- La flexibilidad que brinda tener una implementación propia para el solucionar el problema.

La otra opción que se consideró fue la de utilizar una librería de Python especializada en la administración de crones. Adaptar esta librería a nuestra necesidad (crones por grupo) era más compleja y no brindaba ningún beneficio en particular.

Se desarrollaron dos scripts en bash, uno para borrar crones y otro para agregarlos. Estos scripts se llaman `makeCron.sh` y `removeCron.sh` respectivamente. `makeCron.sh` toma dos parametros de entrada, el primero el cron que se quiere ejecutar. Esto incluye tanto el horario o repetición que se le quiere dar como el comando que se quiere ejecutar. El segundo argumento es el id del device al que se le quiere asignar ese cron. `removeCron.sh` recibe como paramentro el id del dispositivo al que se le quieren eliminar los crones.

`makeCron.sh` funciona de la siguiente manera:

```
mkdir -p ~/crons/$2  
cd ~/crons/$2
```

Se toma el `id` del device y se crea una carpeta con ese número dentro la carpeta crons en el el home del usuario pi (que viene por default). Si ya existe esa carpeta, no hace nada.

Resultando en la siguiente estructura:

- crons
 - 123415616
 - 123516123

Luego entra a la carpeta en cuestión.

A su vez, se agrega la línea con el cron nuevo del device. Como se mencionó anteriormente, esta línea está en el primer parámetro de entrada del bash. Esta línea se agrega a la lista de crones de la siguiente manera:

```
crontab -l > mycron.txt  
echo "$1" >> mycron.txt  
crontab mycron.txt  
rm mycron.txt
```

Se toman los crones que ya existen, dado que `crontab -l` devuelve todos los crones en `crontab`, y se escriben en un archivo temporal. A este archivo temporal se le concatena la línea nueva. A continuación se toma este archivo de texto como el nuevo `crontab` y se elimina el archivo temporal.

Para finalizar, se agrega ese cron a un archivo de texto llamado `deviceCrons` en la carpeta a la que se había entrado. Este archivo se usa para mantener un registro de los crones agregados al `crontab` para cada dispositivo.

Por otra parte, el bash `removeCron.sh` hace lo siguiente:

Se borra el archivo `crontab`:

```
crontab -r
```

Se elimina el archivo donde se registran los crones para ese dispositivo.

```
rm /home/pi/crons/$1/deviceCrons
```

Se deja uno vacío,

```
cd /home/pi/crons/$1
touch deviceCrons
```

Se entra a la carpeta crons y se itera en todas las carpetas para concatenar el archivo deviceCrons de todos los dispositivos a un archivo temporal (ya habiendo eliminado el que el usuario ordenó),

```
cd /home/pi/crons/
for dir in */ ; do
    cat $dir/deviceCrons >> ~/mycron
done
```

Una vez finalizada esa iteración, ese archivo temporal se copia el crontab para luego ser eliminado. De esta manera, ahora en el crontab quedan los crones de todos los dispositivos menos del que se pretendía eliminar.

```
crontab ~/mycron
rm ~/mycron
```

Estos dos scripts bash son llamados el flask en api.py. Los dos endpoints que los llaman son /delete_crons y /add_cron. Funcionan de la siguiente manera:

Endpoint add_cron:

Requiere de 4 datos que son: id del device, la acción (apagar/prender), cuando es que se quiere realizar la acción, el tipo de cron (si es de grupo o de un solo dispositivo).

Se pregunta de que tipo es el cron y dependiendo de eso, llama a métodos diferentes:

```
if(cron["type"]=="device"):
    add_cron(str(cron["_id"]), cron["action"], cron["cron"])
else:
    add_group_cron(str(cron["_id"]), cron["action"], cron["cron"])
```

El método add_cron toma los parámetros **id**, **action** y **cron**, y los concatena en un string que representa al bash que se quiere ejecutar en determinado momento. Este bash en un curl que hace un POST a localhost (ya que es donde se encuentra el servidor flask), llamando así

al endpoint `set_switch_state` que apaga o prende la luz dependiendo de la acción que contenga el POST.

```
str="" + cron + " curl -H \"\"\"Authentication-Token:  
f1a8659701bbbc7196940761c1d55c3b9a25fb6569a863df\"\"\" -X POST -d  
\' {\"state\":\""+action+"\", \"device_id\":\""+id+"\"}  
\' http://localhost:5000/set_switch_state\""
```

Luego se concatena ese string con el del device:

```
str2=str+id
```

Quedando algo de la forma:

```
38 9 * * 4 curl -H "Authentication-Token:  
f1a8659701bbbc7196940761c1d55c3b9a25fb6569a863df" -X POST -d '{"state  
":"0","device_id":"12409587"}' http://localhost:5000/set_switch_state  
12409587
```

Luego se llama a la librería subprocess que es una librería para correr scripts de bash en python:

```
subprocess.call("$(pwd)/makeCron.sh "+str2, shell=True)
```

Asimismo, el método `add_group_cron` tiene como parámetros `groupId`, `action` y `cron`. Toma de la base de datos el grupo con el ese `groupId` y se itera por los devices de ese grupo. Para cada device se llama al método `add_cron` con el mismo criterio que se describió anteriormente.

```
group = get_group_from_db(app, mongo, groupId)
for device in group.get("devices"):
    add_cron(str(device["_id"]), action, cron)
```

Endpoint `delete_crongs`:

Este endpoint requiere de la variable `type` y `id`. La variable `type` puede ser `device` o `group`, dependiendo de cuál de las dos sea, se llamará los métodos `delete_crongs` o `delete_group_crongs`.

```
check_fields(data, DELETE_CRONS_REQUIRED_FIELDS)
if (data["type"]=="device"):
```

```

    delete_crons(str(data["_id"]))

else:
    delete_group_crons(str(data["_id"]))

```

De forma similar al código de `add_group_cron` y `add_cron`, `delete_crons` llama al bash `removeCron` con el `id` del dispositivo.

```

def delete_crons(id):
    str1="device "+id
    subprocess.call("${pwd}/removeCron.sh "+str1, shell=True)

```

A su vez, `delete_group_crons` toma de la base de datos el grupo con el `id` que se le solicita y luego itera en los devices de ese grupo llamando a `delete_crons` para que borre los crons de ese dispositivo.

```

def delete_group_crons(groupId):
    group = get_group_from_db(app, mongo, groupId)
    for device in group.get("devices"):
        delete_crons(str(device["_id"]))

```

credentials.h

Este archivo contiene información sensible para la implementación del sistema (contraseñas y configuraciones de seguridad). Por lo que el mismo no se incluye en el informe ni está versionado en el repositorio, en cambio se brinda un template del mismo, el cual indica qué información es necesaria para el funcionamiento de los scripts. [II.1.2]

main.cpp [II.1.1]

En este script hay muchos flujos lógicos mezclados, por lo que se mencionarán ordenados según su nivel de importancia en un dispositivo que es encendido por primera vez.

- Configuración de registro de dirección de broker MQTT [5.3.1].
- Registro de dispositivo en la red mesh, tanto en el broker como en la base de datos. [5.3.2]
- Restauración de estado anterior del relé.
- Interpretación de mensajes, ya sean órdenes o consultas de estado.

5.3.2. Módulo Controlador

Para el desarrollo de este módulo se realizó la instalación y configuración de varios componentes en el Raspberry Pi.

Setup del servidor Flask, MongoDB

Para instalar el servidor se recomienda hacerlo en un virtual environment, para así evitar conflicto de librerías entre aplicaciones.

Antes de comenzar la instalación de los paquetes de Python en el Raspberry Pi se deben instalar las siguientes librerías:

```
sudo apt-get install libevent-dev  
sudo apt-get install python-all-dev
```

Luego instalaremos MongoDB, ya que será utilizada por el servidor Flask, para esto se ejecuta:

```
sudo apt-get install mongodb-server
```

Luego se crea la base de datos mqtt_device:

```
mongo  
use mqtt_device
```

Luego instalar los requerimientos del proyecto:

```
pip install -r requirements.txt
```

Ejecutar el servidor desde la carpeta que lo contiene:

```
export FLASK_APP=api.py  
flask run --host 0.0.0.0
```

Desarrollo

Como se mencionó anteriormente, Flask cuenta con varias extensiones para agregarle funcionalidades, en este caso fue necesario utilizar dos de ellas, `flask_mqtt` [13] y `flask_pymongo` []. `flask_mqtt` [13]: es un wrapper de la librería `paho-mqtt` [29] que permite conectarse a un broker MQTT, simplificando al integración de este protocolo en las aplicaciones web, maneja los eventos de recibir mensajes, loguear eventos de la red y permite suscribirse y publicar a topics del broker a la que se encuentra conectado. `flask_pymongo` [14]: esta librería también es un wrapper de otra para facilitar su uso y configuración en un servidor Flask, en este caso la librería adaptada es `PyMongo` [32]. Esta librería aporta herramientas para trabajar con MongoDB en Python, presentando los documentos de las colecciones de MongoDB como un repositorio persistente y en el que es posible realizar búsquedas.

El código del servidor se puede encontrar en el anexo [II.2.2], aquí se revisarán las funcionalidades más importantes del mismo.

Configuración de parámetros:

Para configurar los parámetros requeridos para conectarse al broker MQTT se optó por guardar esta configuración en un documento de la MongoDB, aprovechando el hecho de que ya es necesario la conexión con la misma para el manejo de dispositivos. Este documento se puede obtener y modificar a través del endpoint `/config`, realizando un GET o POST respectivamente, en el caso de querer modificarla, se realiza una verificación de que el request contenga los siguientes parámetros:

```
CONFIG_REQUIRED_FIELDS = [
    'MQTT_BROKER_URL', 'MQTT_USERNAME',
    'MQTT_PASSWORD', 'MQTT_BROKER_PORT',
]
```

Las funciones de obtención y actualización del documento de configuración en MongoDB se pueden ver en el anexo [II.2.3].

Agregar dispositivos:

Cuando un nuevo dispositivo se conecta al broker, publica un mensaje al topic /new_device . Debido a la implementación de la librería [??], los topics a los que se publica tienen un prefijo configurado /device_out, y, luego de que se le asigne un número al dispositivo, también contará con otro nivel, siendo /device_out/<BASE_SSID>-<DEVICE_#>. Es por esto que el servidor se suscribe al siguiente topic:

```
MQTT_CHANNELS = {
    'new_device': '/device_out/+/new_device'
}
```

Siendo que hasta que el dispositivo no cuente con el número de dispositivo, el topic no coincidirá con el topic suscrito. Una vez asignado, la wildcard + será ocupada por < BASE_SSID>-<DEVICE_#>.

Una vez recibido el mensaje a ese canal, se crea un nuevo JSON a partir de device_template . json [II.2.4], se chequea que el mensaje recibido sea de la siguiente forma:

```
{
    "_id": {{ ESP chip ID }},
    "type": {{ 1 for sonoff, 2 for dual}}
}
```

Por último se obtiene el subtopic, que es el nivel en el que se hace match con la wildcard, para así poder comunicarnos con ese dispositivo.

Envío de órdenes:

Si un el servidor recibe un mensaje de una interfaz de usuario con las credenciales necesarias de autenticación, y con contenido válido, entonces redirige el mismo a la red mesh.

5.3.3. Módulo de interfaces de usuario

Aplicación web

Aplicación provisoria: Para crear la aplicación provisoria se eligió utilizar la librería React [3.11] junto con `create-react-app` [9], una librería que ayuda en el setup del ambiente de desarrollo. Algunos componentes como los botones y el contenedor en el que están ubicados se importaron de la librería `reactstrap` [37]

Para instalar los componentes necesarios:

```
$ npm install -g create-react-app
```

Luego de instalada la librería, crear una aplicación es tan fácil como:

```
$ create-react-app dude_web_app
```

Sólo hace falta iniciar la misma para comprobar que todo haya marchado bien y empezar a desarrollar código:

```
$ cd my-app  
$ npm start
```

Si todo salió bien se debería observar lo siguiente en la consola:

```
Compiled successfully!
```

```
You can now view dude_web_app in the browser.
```

```
Local:          http://localhost:3000/  
On Your Network:  http://192.168.1.7:3000/
```

```
Note that the development build is not optimized.
```

```
To create a production build, use yarn build.
```

Y la aplicación por defecto se debería ver en el navegador.

Especificaciones de este primer acercamiento:

- Desarrollo en React
- Funcionalidades:
 - Configurar la dirección IP del broker.
 - Especificar el topic al que se desea enviar las órdenes.
 - Órdenes implementadas:
 - Apagar, prender y consultar estado de LED.
 - Apagar, prender y consultar estado de relé.
 - Setear en apagado o prendido y consultar valor de estado al prenderse el Sonoff.

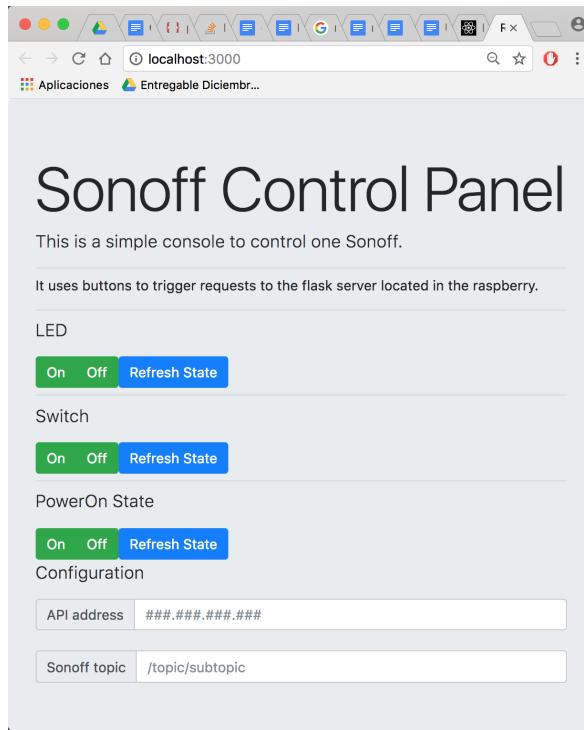


Figura 5.9: Vista de página principal provisoria.

En la figura [5.9] se puede observar los botones para prender y apagar tanto el LED como el relé, el último sirve para indicar el estado del relé cuando el Sonoff inicia. Todos los botones reflejan el estado actual del componente, actualizándose si el mismo cambia por un motivo externo a la misma aplicación (corte de luz, cambio de estado mediante otra sesión se la aplicación o mediante otra interfaz de usuario).

El código de la misma se puede observar bajo el anexo [II.3]

Aplicación Web y Móvil:

Para el desarrollo final se utilizaron las herramientas y librerías Auth0 [6], Phonegap [31] y Android Studio [2].

Para la autenticación de la aplicación celular y web se utilizó la herramienta Auth0, ya que provee funcionalidades muy completas que no podrían haber sido implementadas debido a los tiempos manejados y siendo que no es este el foco del proyecto. Esta herramienta terceriza el servicio de autentificación, pudiendo crearse usuarios propios del servicio o utilizando varias cuentas de otras aplicaciones, entre ellas Facebook, Github y Google. El

flujo de autenticación conlleva una redirección a una página brindada por Auth0, donde el usuario es identificado por alguno de los medios permitidos por la aplicación que solicita la autenticación. Una vez autenticado el usuario, Auth0 redirige al usuario nuevamente hacia la aplicación, brindando información del usuario y un token que expira en un tiempo determinado para aumentar la seguridad.

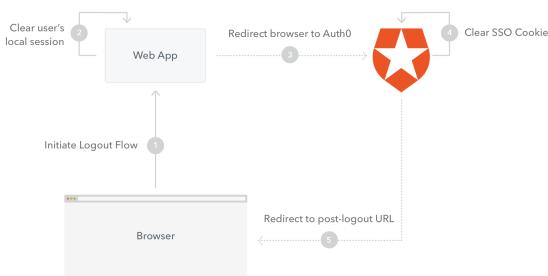


Figura 5.10: Flujo de autenticación con Auth0.

Otras funciones interesantes que permite utilizar sin necesidad de ser implementadas son:

- Recuperación de contraseña.
- Notificaciones vía mail de ingreso sospechoso.
- Validación de cuentas por medio de e-mail o mensaje de texto.
- Activación de autenticación de tres vías, utilizando aplicaciones como Google Authenticator o Authy.

El desarrollo se realizó con tecnologías web, permitiendo así que Phonegap compile las versiones web y móviles sin problemas, algunas configuraciones difieren en la configuración de la autenticación por parte de Auth0, ya que se realizan callbacks diferentes, siendo el de la aplicación móvil uno especial considerado por Auth0. El resto de la aplicación funciona como se espera a pesar de la necesidad de ajustes de tamaño estéticos y de usabilidad debido a las diferencias proporcionales de las pantallas. Para crear el proyecto en Phonegap se instala el paquete con la herramienta ‘npm’.

```
npm install -g phonegap
```

Luego se inicializa el proyecto con ‘phonegap create web_app’, que crea el esqueleto del proyecto, que incluye entre los componentes más importantes:

- node_modules: carpeta con los módulos de node instalados
- www: carpeta donde se contendrán los archivos de la página web.
- config.xml: archivo que contiene metadata de la aplicación así como librerías y plataformas que se han habilitado para el empaquetamiento.

Una vez agregado código en la carpeta www, se compila para la plataforma Android ejecutando:

```
phonegap build android
```

Y para la aplicación web:

```
phonegap build browser
```

Para poder ir visualizando los resultados del desarrollo, se sirve la aplicación en un servidor local para desarrollar la versión web y emular un móvil Android gracias a Android Studio. Para levantar el servidor local basta con ejecutar `phonegap serve`, levantando el servicio en el puerto 3000. En modo default cuenta con una configuración que permite la recarga automática ante cambios en el código que hicieron de esta la herramienta principal para el desarrollo de la aplicación. A diferencia de la aplicación web, la aplicación móvil hizo necesaria la instalación del Android Studio para contar con el servicio de emulación, ya que no se contaban con móviles a disposición. Una vez instalado android studio se debe entrar al proyecto e inicializar una instancia de un celular, pudiéndose elegir entre múltiples modelos. Luego se debe seleccionar y descargar el sistema operativo, en nuestro caso utilizamos Android Oreo en Pixel 2. Luego se compila y sube la aplicación corriendo el emulador y el comando `phonegap run android`.

Antes de comenzar el desarrollo el código de la aplicación web, se realizó un bosquejo de este en Balsamic, que permite además de dibujar sencillamente las páginas, también permite definir links que funcionan como botones, cambiando las vistas para poder definir flujos intuitivos y determinar las páginas necesarias. Una vez planificada la estructura de la página se comenzó con la implementación práctica.

Para el desarrollo de la aplicación se decidió adquirir un template para no tener que invertir tiempo en la parte estética de la aplicación. Para la elección del mismo se tuvo en cuenta las librerías que utilizan y la adaptabilidad a versiones móviles. El tema elegido fué MegaMobile de Enabled [10], debido a su interfaz vistosa y el estilo de las barras laterales y botones, los cuales se adecuaban a las pantallas diseñadas en Balsamic.

Librería jquery-cron:

Debido a la utilización de cron del lado del servidor, es necesario traducir la hora, día, etc a la que el usuario quiere realizar el apagado y prendido de los dispositivos o de los grupos a el formato de cron. Para esto se utilizó una librería llamada `jquery-cron` que simplifica esta tarea. La misma consiste en una interfaz muy simple, como se ve a continuación, que cambia el `String` del cron generado dependiendo de lo que el usuario ingrese. A su vez, `jquery-cron` es complementada por `jquery-gentleSelect` que decora los bordes de los selectores.



Figura 5.11: Selectores jquery-cron.

Se adaptó esta librería creando un fork del proyecto para que todos los nombres de las opciones y textos queden en español, esta modificación quedó disponible para la utilización de la comunidad. Para usar esta librería alcanza con tener dos divs, uno para el scheduler y otro para texto resultante. El margen izquierdo del div de texto es negativo para que no aparezca en la pantalla, dado que el cron debe ser transparente para el usuario.

```
<div id="selector_scheduler"></div>

<div class='example-text' style="margin: 0px 0px 0px -40%;" id='example1-val'></div>
```

Se debe cargar la librería antes de usarla, junto con jquery-gentleSelect.

```
<script type="text/javascript" src="js/jquery-cron.js"></script>
<script type="text/javascript" src="js/jquery-gentleSelect.js"></script>
```

Luego, se crea la interfaz:

```
$('#selector_scheduler').cron({
    initial: "42 3 * * 5",
    onChange: function() {
        $('#example1-val').text($(this).cron("value"))
    },
    useGentleSelect: true
})
```

Initial es el valor inicial que toma el scheduler. Onchange toma una función para ejecutar cuando el usuario selecciona un nuevo valor en alguno de los selectores. Lo que hace la función es traducir lo que el usuario eligió y setearlo en el div de texto. UseGentleSelect se pone en true para habilitar el uso de la librería jquery-gentleSelect.

Interfaz móvil y web

Cuando el usuario abre por primera vez la aplicación, solo tiene la posibilidad de iniciar sesión. Esto se realiza mediante el servicio autenticación Auth0. Si no se cuenta con un token de autorización no expirado, la aplicación requerirá que el usuario se identifique.

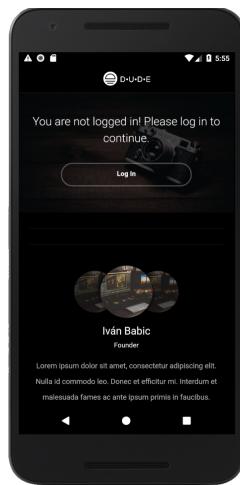


Figura 5.12: Página de inicio.

Una vez que se presiona el botón de LogIn, este lleva al servicio de google Auth0 para que el usuario pueda autenticarse.



Figura 5.13: *D.U.D.E. Auth0 Log In*

Una vez que el usuario rellena sus datos y logra iniciar su sesión, la aplicación retorna con sesión iniciada, y checkea que se cuente con la configuración del Raspberry Pi. En caso de que no esté configurado, se muestra la interfaz de configuración, la cual permite ingresar la IP del Raspberry y al oprimir el botón enviarle un request de la configuración. Es totalmente necesaria la conexión entre dispositivo y raspberry para el funcionamiento de la aplicación, ya que es ahí donde se encuentra la base de datos con la información de dispositivos, y es sólo a través del dispositivo central que se puede interactuar con los demás dispositivos configurados.

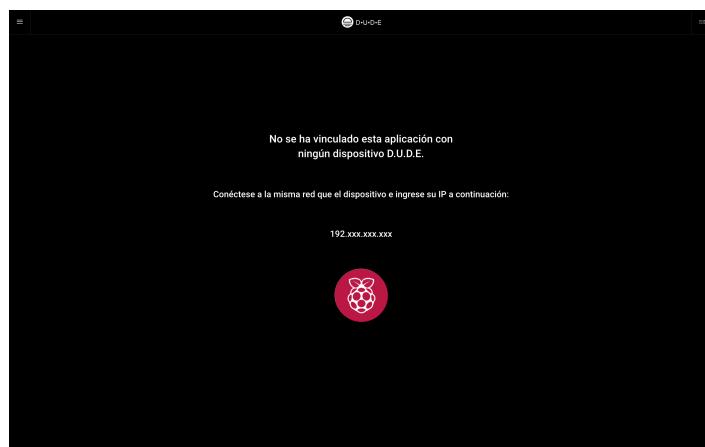


Figura 5.14: *Página de configuración de dispositivo central.*

Si ya se contaba con esta configuración, o luego de realizarla, se vuelve a la página principal. Aunque en este caso se cuenta con un mensaje que indica que la sesión, un botón

para salir de ésta y un ícono en la esquina superior izquierda.

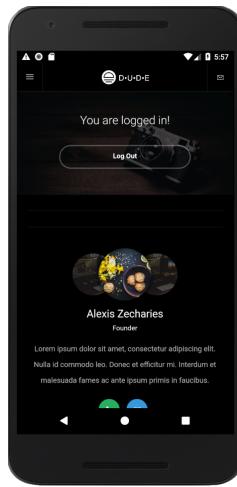
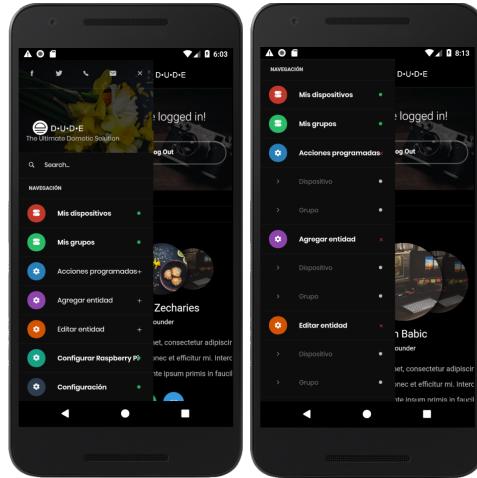


Figura 5.15: Página de inicio con sesión iniciada.

Al presionar este ícono, se desliza la barra lateral desde la izquierda. Esta contiene todas las funciones del sistema:

- Mis dispositivos
- Mis grupos
- Acciones programadas
 - Dispositivo
 - Grupo
- Agregar entidad
 - Dispositivo
 - Grupo
- Editar entidad
 - Dispositivo
 - Grupo
- Configuración.

Al seleccionar Acciones programadas, Agregar entidad o Editar entidad se abre un submenú en cada una que posibilita realizar estas acciones para dispositivos o para grupos.



(a)) *Funciones.* (b)) *Opc. expandidas.*

Figura 5.16: *Vistas de barra lateral.*

Funciones del sistema

Mis dispositivos

Vista que permite modificar el estado de un dispositivo previamente agregado, la lista información de dispositivos es obtenida desde el Raspberry. se muestra un ícono de una lámpara apagada o prendida dependiendo del estado actual del dispositivo. Al presionar el botón con la bombilla se envía la orden al dispositivo central, quien publica la orden y cambia el estado en la base de datos.

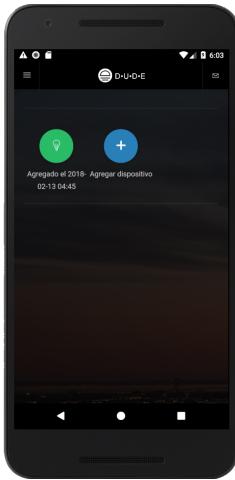


Figura 5.17: *Cambiar estado de dispositivo.*

Mis grupos

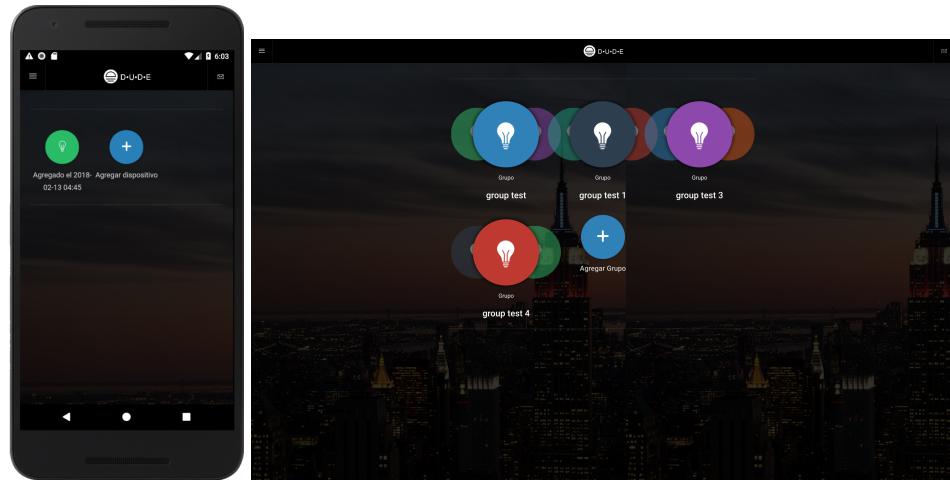
Vista que permite modificar el estado de un grupo existente. Ya que estos no tienen un estado de prendido o apagado porque los dispositivos pueden encontrarse en ambos estados aunque pertenezcan al mismo grupo, se le da la opción al usuario de cambiar de estado a todos los dispositivos del grupo mediante dos botones, uno con una bombilla apagada y otro con una bombilla prendida.



Figura 5.18: *Cambiar estado de grupo.*

Elegir dispositivo

En este las funciones Acciones programadas y Editar entidad se debe elegir el grupo o dispositivo al que se quiere programar una accion o editar. Esto se logra con las siguientes vistas.

(a)) *Dispositivo*.(b)) *Grupo*.Figura 5.19: *Vistas de selección*.

Acciones programadas

La interfaz para agregar una acción cuenta con 3 botones, así como una interfaz gráfica brindada por la librería `jquery-cron` explicada anteriormente para la elección de la hora, día, mes, año, día de la semana, minuto de cada hora que el usuario puede elegir para programar la acción. En la parte superior aparece el nombre del dispositivo o grupo al que se le quiere aplicar esta acción.

Los 3 botones son: apagar, prender, eliminar. Dependiendo de si se eligió dispositivo o grupo, se mandará el id del dispositivo o grupo elegido. Si se selecciona la opción eliminar se eliminan todos los cronos para el dispositivo o grupo. Si se elige prender o apagar, se mandará esa acción junto con el cron que traduce la librería que eligió el usuario y el `id` que se explicó previamente.



Figura 5.20: Agregar acción programada.

Agregar entidad

Esto da la posibilidad de agregar un dispositivo o un grupo.

- **Dispositivo:** Para agregar un dispositivo tiene que haber un Sonoff en modo descubrimiento, es decir, un Sonoff que tenga la luz de su led parpadeando. Esto quiere decir que está creando una red llamada ConnectAP. Teniendo esto en cuenta, al seleccionar la pestaña de agregar dispositivo aparece en pantalla un texto que indica al usuario que se conecte a la red ConnectAP. Cuando este se conecta a esa red, la aplicación lo lleva a la pantalla de inicio dado que que mandaron los datos correctamente a al dispositivo y este queda correctamente agregado.



Figura 5.21: Agregar dispositivo.

- **Grupo:** presenta una interfaz gráfica que lista todos los dispositivos agregados por el usuario, permitiendo seleccionar los que se desea incluir en el grupo y un elegir un nombre. Al texto que introduce el usuario se modifica para evitar problemas. Algunas de las correcciones son: sustituir espacios por uno solo, borra tabs y new lines, impone mayúscula a la primer letra del nombre. Una vez que el usuario presiona el botón confirmar, se envían esos datos al dispositivo central para que los agregue a la base de datos.



Figura 5.22: Agregar grupo.

Editar entidad

Las opciones que se brindan son editar dispositivo o editar grupo.

- **Dispositivo:** lleva a una interfaz con todos los dispositivos registrados y la opción de agregar un dispositivo nuevo [[5.19]a)]. Una vez que se elige uno de ellos, se va a la interfaz de edición de dispositivo. En esta se muestra el nombre del dispositivo, que ser presionado permite la edición del mismo. Se despliegan todos los grupos registrados en la base de datos. Si el dispositivo ya pertenece a alguno de ellos, aparece con un tick a la izquierda. Todos pueden seleccionarse o dejarse de seleccionar según las preferencias del usuario. El botón de confirmar manda los Id de todos los grupos seleccionados para ese dispositivo y el nombre que está escrito en esa interfaz, este cambiado o no. También se manda el **id** del dispositivo al que se le aplicaron los cambios y se vuelve al menú inicial.

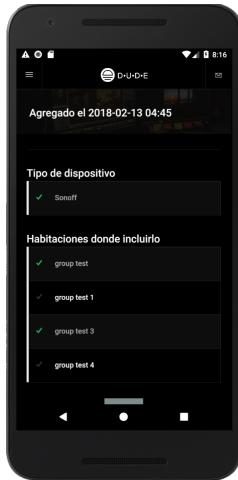


Figura 5.23: *Editar dispositivo.*

- Grupo: lleva a una interfaz con todos los grupos registrados además de la opción de agregar un nuevo grupo [[5.19]b)]. Una vez elegido el grupo, aparece una interfaz equivalente a la de nuevo grupo. La única diferencia es que en esta interfaz, si hay dispositivos seleccionados previamente para integrar ese grupo, aparecerán con un tick a la izquierda de su nombre en la lista de dispositivos. De forma similar a lo explicado para el editar grupo, al presionar el botón confirmar se manda el **id** del grupo editado, una lista con los Id de los dispositivos elegidos y el nombre del grupo que aparece en pantalla.



Figura 5.24: *Editar grupo.*

Error

En caso de haber un error de conexión con el flask, el usuario verá una pantalla de error que es la siguiente.

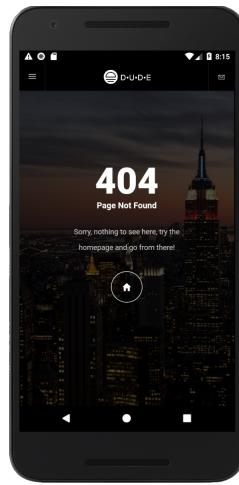


Figura 5.25: Pantalla de error.

Hosting de aplicación web

Para que la aplicación sea accesible y su mantenimiento no sea una sobrecarga, se decidieron utilizar AWS (Amazon Web Services) donde se levantó una máquina virtual con EC2 que sirve un Apache HTTP Server.

Amazon EC2

Para crear una máquina virtual, primero se necesita elegir que sistema operativo. En este proyecto, como solo se debe mostrar un prototipo, se eligió una de las opciones gratis que brinda AWS con Ubuntu de arranque. Se optó por Ubuntu dado que se tenía experiencia instalado servidores Apache ahí.

 SUSE Linux Enterprise Server 12 SP3 (HVM), SSD Volume Type - ami-75143f10	Select	
SUSE Linux Free tier eligible	SUSE Linux Enterprise Server 12 Service Pack 3 (HVM), EBS General Purpose (SSD) Volume Type. Public Cloud, Advanced Systems Management, Web and Scripting, and Legacy modules enabled. Root device type: ebs Virtualization type: hvm	64-bit
 Red Hat Enterprise Linux 7.4 (HVM), SSD Volume Type - ami-0b1e356e	Select	
Red Hat Free tier eligible	Red Hat Enterprise Linux version 7.4 (HVM), EBS General Purpose (SSD) Volume Type Root device type: ebs Virtualization type: hvm	64-bit
 Ubuntu Server 16.04 LTS (HVM), SSD Volume Type - ami-965e6bf3	Select	
Ubuntu Server Free tier eligible	Ubuntu Server 16.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services). Root device type: ebs Virtualization type: hvm	64-bit

Figura 5.26: Elección de SO.

Luego de elegir el sistema operativo, hay que elegir cuanta RAM, procesador, característica de red se quiere tener. En este caso, solo había una opción gratis y que a su vez tenía todo lo que se necesitaba para el servidor. Por lo que se eligió esa.

Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)								
	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
<input type="checkbox"/>	General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
<input checked="" type="checkbox"/>	General purpose	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.large	2	8	EBS only	-	Low to Moderate	Yes

Figura 5.27: Elección de máquina virtual.

Finalmente se le debe asignar un grupo de seguridad a esa instancia como se ve a continuación.

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more about Amazon EC2 security groups.](#)

Assign a security group: Create a new security group
 Select an existing security group

Security Group ID	Name	Description	Actions
<input type="checkbox"/> sg-10e57a7b	default	default VPC security group	Copy to new
<input type="checkbox"/> sg-ab8c13c0	dude	dude	Copy to new
<input type="checkbox"/> sg-489d0223	launch-wizard-1	launch-wizard-1 created 2018-03-03T13:14:46.923-03:00	Copy to new

Figura 5.28: Elección de grupo de seguridad.

Por lo tanto, antes de esto se creó un grupo de seguridad acorde para este servicio. Se habilitó el ping tanto como las conexiones TCP en el puerto 80. El SSH en el puerto 22 ya estaba habilitado previamente. Todo el tráfico saliente ya está habilitado por defecto.

Type	Protocol	Port Range	Source	Description	
HTTP	TCP	80	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop	
HTTP	TCP	80	Custom ::/0	e.g. SSH for Admin Desktop	
SSH	TCP	22	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop	
SSH	TCP	22	Custom ::/0	e.g. SSH for Admin Desktop	
All ICMP - IPv4	ICMP	0 - 65535	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop	
All ICMP - IPv6	ICMP	0 - 65535	Custom ::/0	e.g. SSH for Admin Desktop	

Figura 5.29: Creación de grupo de seguridad.

Luego de crear la instancia, AWS da al usuario una llave para que pueda conectarse con el servidor en cuestión. La llave se puede elegir de otra ya existente o crear una nueva. El usuario guarda su llave privada, mientras que la instancia que acaba de crear se queda con la llave pública.

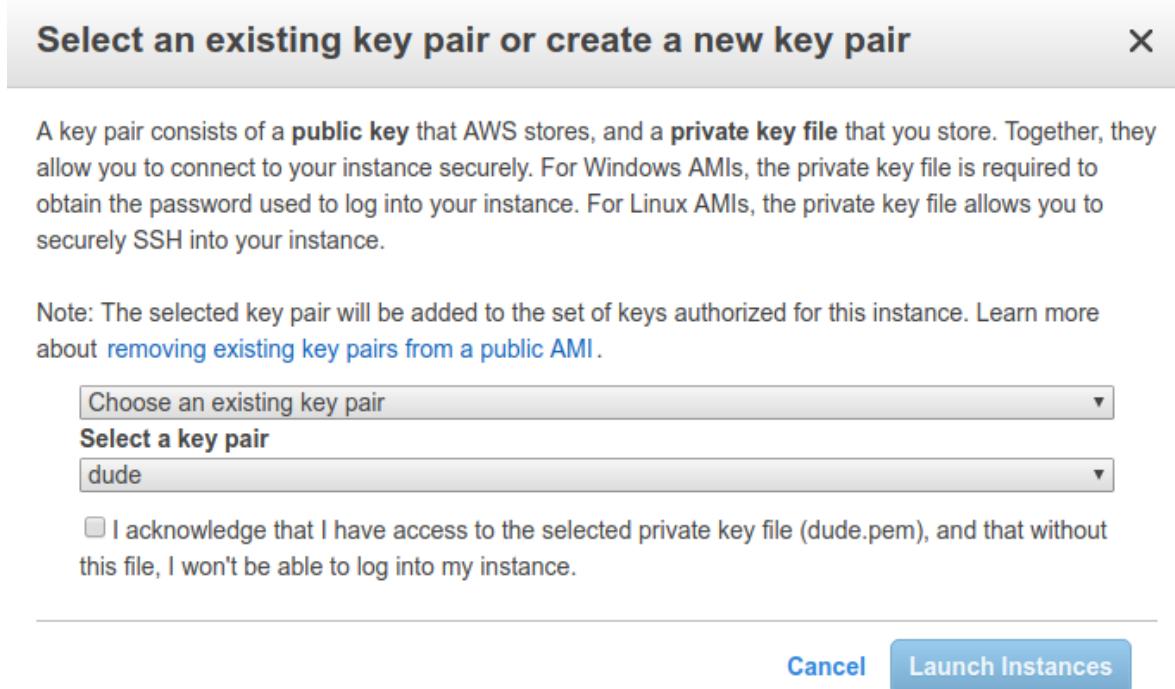


Figura 5.30: Elección de llave.

Terminado esto queda la máquina virtual corriendo. Hay un panel web que permite ver el estado de la misma, su IP su DNS público como se ve a continuación.

	Name	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IP)	IPv4 Public IP	IPv6 IPs
<input checked="" type="checkbox"/>	dude	i-061...	t2.micro	us-east-2c	running	2/2 checks ...	None	ec2-52-14-23...	52.14.237.175

Figura 5.31: Panel informativo.

Apache HTTP Server

Primero se instala Apache mediante el gestor de paquetes de Ubuntu de la siguiente manera:

```
apt-get install apache2
```

Luego, todo el código generado por phonegap para la plataforma de navegador se puso en

la carpeta `/var/www/html/dude`. Asimismo, se necesita que todos los llamados que van al puerto 80 sea atendido por el servidor Apache y este llame a los archivos en `/var/www/html/dude`.

Para lograr esto en el archivo `ports.conf` se tiene que configurar que apache escuche en el puerto 80 (lo cual se hace por defecto) de la siguiente manera:

Listen 80

Una vez que se escucha lo que llega a ese puerto, hay que direccionar el tráfico a la carpeta `/var/www/html/dude`. Para esto, en `sites-available` hay que modificar el archivo por defecto de servidor para que el tráfico que llega a ese puerto se direccione a `/var/www/html/dude` y no a `/var/www/html`, como hace por defecto. El archivo por defecto modificado es `000-default.conf`.

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html/dude

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Finalmente, hay que habilitar ese archivo y reiniciar el servidor apache

```
sudo a2ensite 000-default.conf
sudo service apache2 restart
```

Mycroft

Se desarrollaron skills para el asistente virtual Mycroft [27] como una de las interfaces que permitirán al usuario interactuar con el sistema, pero a diferencia de las aplicaciones móvil y web, esta interfaz tiene funcionalidades más básicas, ya que la interfaz por vos no resulta cómoda para acciones complejas, como agregar dispositivos o modificar los mismos. Mycroft cuenta con dos versiones en la actualidad, una para Linux y otra pensada específicamente para Raspberry Pi, llamado Picroft. La segunda contiene a la implementación de Linux además de otros archivos y configuraciones necesarios para lograr la compatibilidad

de Mycroft con esta plataforma y sus drivers. Debido a problemas en actualizaciones en los servidores de Mycroft y debido a que el proyecto Picroft se ha enfocado más en la compatibilidad con los productos Mark 1 y Mark 2 (hardware compuesto por Raspberry Pi con micrófonos y parlantes montados), ocurrieron errores con las configuraciones de Raspberry's más generales. Siendo imposible la instalación correcta y funcional de Picroft, se consultó en foros de la comunidad y nos pusimos en contacto con desarrolladores del proyecto, aportando logs de nuestras instalaciones para ayudar a descubrir y solucionar el problema ([25], [26]). A pesar de su apoyo no se logró el cometido, siendo que descubrimos que era un problema general y contamos con nuestros topics entre los más comentados y vistos a la semana de realizar la publicación. Luego de estos intentos se decidió que se debía optar por una alternativa, considerando las herramientas de Google y Amazon para esto, a pesar de que se alejaban del perfil open source y sin monetización de datos del cliente. Por estas razones y sumada la confianza en el equipo de Mycroft y en la comunidad para que se resuelvan los problemas en Picroft, se decidió por desarrollar los skills con Mycroft instalado en Linux, ya que cuando eventualmente Picroft vuelva a un estado estable, se pueden utilizar los mismos skills.

Debido a los contratiempos se desarrollaron únicamente skills de cambio de estado de luces, cuyos llamados son:

- Prender luces:
 - turn on the lights
 - turn the lights on
 - make it shine
 - light me up
- Apagar luces:
 - turn off the lights
 - turn the lights off
 - darkness please

Para el desarrollo y entendimiento de la implementación de skills se siguió la guía introductoria brindada por el equipo de Mycroft [16]. La estructura básica de un skill es:

- Directorios

- dialog: contiene una carpeta por idioma, que a su vez contienen los archivos con frases de respuesta para skills, de los que Mycroft elige de forma aleatoria dentro de un archivo.
- test: contiene archivos para realizar tests de intents.
- vocab: similar al directorio dialog, solo que los archivos contienen frases que determinan la intención de utilizar un skill.

- Archivos

- `__init__.py`: Este es el cuerpo principal del skill, donde se definen los intents y las definiciones handler, que contienen la lógica de la función a ejecutar.
- LICENSE: Archivo que determina el tipo de licencia bajo la que se rige el skill.
- README.md: Archivo que explica la utilización del skill.
- requirements.sh y requirements.txt: Archivos que definen las dependencias necesarias para el funcionamiento de los skills.

5.3.4. Aportes a librerías de terceros

A medida que se avanzó en el proyecto se encontraron problemas u oportunidades de mejora en algunas librerías que fueron utilizadas para llevar a cabo el proyecto. Esto generó que, a pesar que no fuera un objetivo desde el comienzo, se mejoró o ayudó a mejorar estas librerías.

ESP8266MQTTMesh

Se encontró un problema que provocaba que no se pudiera conectar a la red MQTT el dispositivo si no se conseguían los datos como el nombre de red y contraseña desde un archivo de texto que se cargaba con el código. De esta manera, estos parámetros no podían ser guardados en los SPIFFS para conseguirlos luego. Esto se reportó en el issue [26] y se consiguió una rápida respuesta de parte del programador a cargo. Esto llevó a que la librería ahora pueda integrarse con proyectos como WiFiManager, que brindan una conexión con el ESP para que se pueda elegir la SSID a la cual conectarse

y la contraseña.<https://github.com/PhracturedBlue/ESP8266MQTTMesh/issues/29> Esto fue implementado por en este proyecto e tesis y casualmente otra persona lo solicitó <https://github.com/PhracturedBlue/ESP8266MQTTMesh/issues/38>. Por este motivo, al terminar el proyecto se va a agregar un código de ejemplo a la librería ESP8266MQTTMesh para que otras personas puedan integrar ambas librerías, haciéndola una opción muy atractiva.

Finalmente, también se ayudó a mejorar la librería desde el aspecto del debuggeo. Se tuvo un problema cuya causa fue una elección errónea de versión de plataforma. A partir de esto, el programador encargado del proyecto agregó código que no deja compilar si la elección de la plafotma no es la adecuada. De esta forma se ahorró horas de debuggeo de issues cuyo problema no sea el código en sí. <https://github.com/PhracturedBlue/ESP8266MQTTMesh/issues/19>

jquery-cron

Se creó un fork de la librería con la traducción al español de la misma. Este fork es visible para todo el mundo. <https://github.com/alexiszecharies/jquery-cron> <https://github.com/shawnchin/jquery-cron/issues/33>

WiFiManager

Se ayudó en el testeo de un fork en estado de development de esta librería. Se encontró un error que luego fue corregido por el encargado de ese fork. <https://github.com/tzapu/WiFiManager/issues/1> A su vez, se modificó el código fuente para cumplir con uno de los requisitos del proyecto y se dejó constancia de cómo se hizo para lograr esto en ese issue.

Capítulo 6

Conclusiones

Se cumplió con todos los objetivos previamente establecidos. El sistema cuenta con una API que puede ser llamada por distintas interfaces de forma transparente para el usuario. Esto puede verse mediante la utilización del asistente por voz Mycroft, cuya función es interpretar comandos de voz para llamar a la API del sistema. Es un sistema muy flexible, el lugar de Mycroft podría estar ocupado por sensores de movimiento, sensores de calor, etc. Esta flexibilidad hace al sistema atractivo para empresas que necesitan una solución a medida.

A su vez, el respeto por la privacidad del usuario es clave en el sistema. Muchas soluciones de domótica hacen pasar sus request por servidores que registran las acciones del usuario. De esta forma pueden obtener datos como el horario en que el usuario está en su casa, cuando duerme. También se debe destacar que la seguridad es un punto fuerte en este sistema. El poco poder de procesamiento en los dispositivos de IoT suele ser un problema para la seguridad de los mismos, sin embargo, se eligió una arquitectura que aunque hizo el desarrollo más complejo, esto se tradujo a seguridad.

Respecto a la competitividad del producto, vale la pena mencionar que una de las motivaciones para la elección de este tema fué el interés por la domótica y las ansias de poseer este tipo de tecnología. El problema con las opciones a disposición son varios, aunque la mayor barrera resultaba ser los elevados precios de los productos. Por ejemplo los productos Hue de Phillips promocionados por el asistente virtual de Google, cuyos precios más modestos son de USD 50 por un pack de 4 luces blancas de intensidad fija.

Otro punto a tener en cuenta es que este método donde la bombilla contiene la lógica resulta en que un artefacto luminoso con varias bombillas aumente sustancialmente el gasto necesario para automatizar un hogar u oficina. Siendo también necesario el que el artefacto en cuestión sea compatible con la conexión de la bombilla, quitando libertad en la elección de

las mismas y teniendo que reemplazar las previamente instaladas.

Nuestra solución aporta la filosofía de automatizar elementos previamente existentes y no se limita a artefactos luminosos, ya que es posible controlar termotanques u otros electrodomésticos como calderas eléctricas.

Sumado a esto se buscó que el sistema no dificulte el funcionamiento actual de las instalaciones, ya que debido a que el dispositivo Sonoff modificado es conectado a los interruptores eléctricos colocado en las paredes, el usuario puede controlar el dispositivo tanto analógicamente como con nuestro sistema sin problemas. En el caso presentado por la mayoría de la competencia, es imposible encender una luz cuya llave no brinde corriente a la misma, siendo que nuestro sistema siempre se encuentra encendido, sin importar el estado del relay que controla el estado de la corriente.

Asimismo, el uso de una librería basada en la arquitectura mesh hace posible que el sistema pueda usarse en lugares con alcance de la red WiFi limitada. Los estudiantes sienten que haber trabajado con este tipo de arquitecturas les sirvió mucho desde el punto de vista técnico, dado que este tipo de arquitectura es muy novedosa y trae muchos beneficios.

Este proyecto fue muy provechoso para los alumnos. Estos pudieron trabajar con tecnologías y lenguajes de programación con las cuales no habían tenido oportunidad durante la carrera. Esto representó un gran desafío, pero a la vez el resultado superó lo que ellos esperaban cuando empezaron. El tener que adaptar muchas de las librerías que se utilizaron para la necesidades del proyecto hizo que los estudiantes tuvieran que estar codo con los programadores de las mismas tanto buscando errores como codificando. Incluso se aportó soluciones a problemas y se aportaron ideas que quedan como referencia para las personas que elijan estas librerías. Esta experiencia profesional es invaluable a la hora de salir al mercado laboral.

Capítulo 7

Recomendaciones

deberiamos hablar que somos dependientes de las actualizaciones de mycroft? tendriamos que averiguar eso bien, por si en la muestra nos preguntan que pasa si vendemos el producto. puede que mycroft actualice y se rompa todo?

configuracion de RPi

conexion entre Rpi y celular es manual, deberia ser mas transparente capaz

randomizacion de contrasena de la mesh

emisor rf pi zero que se conecte como un sonoff al pi zero con la bd

agregar otros tipos de sonoff

mayor cantidad de skills para mycroft

Bibliografía

- [1] *AmazonEC2*. https://docs.aws.amazon.com/es_es/AWSEC2/latest/UserGuide/concepts.html. Accedido: 2018-03-03.
- [2] *Android Studio Website*. <https://developer.android.com/studio/index.html>. Accedido: 2018-03-01.
- [3] *Apache*. https://en.wikipedia.org/wiki/Apache_HTTP_Server/. Accedido: 2018-03-03.
- [4] *Atom Website*. <https://atom.io/>. Accedido: 2017-08-20.
- [5] *Atom Website*. <http://docs.platformio.org/en/latest/platforms/espressif8266.html>. Accedido: 2017-08-20.
- [6] *Auth0 Website*. <https://auth0.com/>. Accedido: 2018-03-01.
- [7] *Banana Pi M3*. <http://www.banana-pi.org/m3.html>. Accedido: 2017-12-01.
- [8] *BeagleBone Black*. <https://beagleboard.org/black>. Accedido: 2017-12-01.
- [9] *create-react-app Repository*. <https://github.com/facebookincubator/create-react-app>. Accedido: 2017-12-01.
- [10] *Enabled Website*. <https://www.enableds.com/>. Accedido: 2018-03-01.
- [11] *ESP8266 Community Forum*. <http://www.esp8266.com/>. Accedido: 2017-12-01.
- [12] *Flask Docs*. <http://flask.pocoo.org/>. Accedido: 2017-12-01.
- [13] *flask-mqtt Docs*. <http://flask-mqtt.readthedocs.io/en/latest/>. Accedido: 2017-12-01.
- [14] *flask-pymongo Docs*. <https://flask-pymongo.readthedocs.io/en/latest/>. Accedido: 2017-12-01.
- [15] *Google Home Docs*. <https://support.google.com/googlehome/?hl=en#topic=7029677>. Accedido: 2017-08-20.
- [16] *GuiaMycroft*. <https://mycroft.ai/documentation/skills/introduction-developing-skills/>. Accedido: 2018-03-01.
- [17] *Home Assistant Docs*. <https://home-assistant.io/>. Accedido: 2017-08-20.

- [18] *HTTP CoAP Layers*. <https://www.cse.wustl.edu/~jain/cse574-14/ftp/coap/index.html>. Accedido: 2017-12-01.
- [19] *IKEA Lighting Website*. <http://www.ikea.com/us/en/catalog/categories/departments/lighting/36812/>. Accedido: 2017-12-01.
- [20] *JavaScript Website*. <https://www.javascript.com/>. Accedido: 2017-12-01.
- [21] *MongoDB Docs*. <https://docs.mongodb.com/manual/mongo/>. Accedido: 2017-12-01.
- [22] *MQTT Diagram*. <https://2016.angularattack.com/entries/506-armyofone>. Accedido: 2017-12-01.
- [23] *MQTT Essentials part 5*. <https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices>. Accedido: 2017-12-01.
- [24] *Mycroft Community Skill Repository*. <https://github.com/MycroftAI/mycroft-skills/blob/master/README.md#community-contributed-skill-list>. Accedido: 2017-12-01.
- [25] *Mycroft Topic 1*. [https://community.mycroft.ai/t/cant-open-cli-in-picroft-0-9/2886/](https://community.mycroft.ai/t/cant-open-cli-in-picroft-0-9/2886). Accedido: 2018-03-01.
- [26] *Mycroft Topic 2*. [https://community.mycroft.ai/t/with-picroft-0-8b-picroft-is-crashing-at-start/2887/2/](https://community.mycroft.ai/t/with-picroft-0-8b-picroft-is-crashing-at-start/2887/2). Accedido: 2018-03-01.
- [27] *Mycroft Website*. <https://mycroft.ai/>. Accedido: 2017-08-20.
- [28] Lavinia Năstase. «Security in the Internet of Things: A Survey on Application Layer Protocols». En: (2017).
- [29] *paho-mqtt Docs*. <https://pypi.python.org/pypi/paho-mqtt/1.1>. Accedido: 2017-12-01.
- [30] *Phillips Lighting Website*. <http://www.lighting.philips.com/main/systems/system-areas>. Accedido: 2017-12-01.
- [31] *Phonegap Website*. <https://phonegap.com/>. Accedido: 2018-03-01.
- [32] *PyMongo Docs*. <http://api.mongodb.com/python/current/>. Accedido: 2017-12-01.
- [33] *Raspberry Pi 2*. <https://www.raspberrypi.org/products/raspberry-pi-2-model-b/>. Accedido: 2017-12-01.

- [34] *Raspberry Pi 3*. <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>. Accedido: 2017-12-01.
- [35] *React Native Website*. <https://facebook.github.io/react-native/>. Accedido: 2017-12-01.
- [36] *ReactJS Website*. <https://reactjs.org/>. Accedido: 2017-12-01.
- [37] *reactstrap Website*. <https://reactstrap.github.io/>. Accedido: 2017-12-01.
- [38] *RPi Access Point Configuration*. <https://cdn-learn.adafruit.com/downloads/pdf/setting-up-a-raspberry-pi-as-a-wifi-access-point.pdf>. Accedido: 2017-12-01.
- [39] *USB to Serial Adapter*. http://product.hstatic.net/1000157790/product/nex00015_converter_usb_to_serial_ttl_master.jpg. Accedido: 2017-12-01.
- [40] *USB to Serial Cable*. http://www.robotshop.com/media/catalog/product/cache/1/image/900x900/9df78eab33525d08d6e5fb8d27136e95/f/t/ftdi-usb-to-ttl-serial-cable-3-3v_1.jpg. Accedido: 2017-12-01.
- [41] *What is PlatformIO*. <http://docs.platformio.org/en/latest/what-is-platformio.html>. Accedido: 2017-08-20.
- [42] *WiFi Attacks*. <https://security.stackexchange.com/questions/6946/is-it-possible-to-get-all-the-data-i-send-through-wifi/6947#6947>. Accedido: 2017-11-20.

Anexo I

Entregables

I.1. Justificación de avance de 50 %

Proyecto Final de Carrera

Justificación de Avance de 50%

D.U.D.E

Compras en el exterior

Dentro de las actividades referentes a las compras, se realizaron todas y se tuvo en cuenta traer los productos con una cierta redundancia, dado que si suceden imprevistos como roturas (por la poca experiencia con los dispositivos con los que se trabajan) o extravíos, estos no significaran un retraso que pusiera en riesgo la planificación de tiempos. Dentro de los supuestos y riesgos especificados en el Informe de Proyecto Final de Carrera se tuvo en cuenta posibles retrasos en la llegada de los dispositivos que se pretendían traer desde Estados Unidos, pero desafortunadamente sucedió un hecho imprevisible que fue la llegada de un huracán a Miami , donde se encontraba el courier, provocando demoras MUCHO mayores a las previstas y haciendo que se tuviera que seguir muy de cerca este tema para que los objetos llegaran lo más rápido posible.

Que se haya alargado esta actividad hizo que consecuentemente se alargará la investigación, que era una actividad extra. Esto fue extremadamente provechoso, dado que se pudo indagar más a fondo acerca de los diferentes protocolos y arquitecturas para realizar el proyecto.

Investigación

En la etapa de planificación y definición de arquitecturas se contó con una gran carga de investigación; ya que muchas de las alternativas que se manejaron para cada componente del sistema eran poco o nada conocidas al principio de este proyecto, lo que dificultó una elección temprana de la solución a escoger. Al presentarse esta problemática en muchos frentes se tuvo que estudiar cada tecnología en profundidad para no sólo utilizar las que representarán mayores ventajas individuales, sino también como partes de un sistema compuesto por todas ellas. Debido a esta situación se optó por una estrategia iterativa,

donde además de investigar las tecnologías se hicieron pruebas de protocolos e investigación de demos y proyectos similares para obtener una perspectiva práctica y entender las implementaciones actuales para cada solución; en esta búsqueda se hizo foco en ver la estabilidad y madurez de las librerías y la comunidad que las rodeaba, ya que en el mundo open-source esta es una pieza fundamental, ya que es mejor una librería con errores que sea activamente mantenida que una con menos imperfecciones pero abandonada.

WBS	Tarea
1.1	Planificación de Proyecto
1.2.1.1.1	Inv. Single-Board Computer
1.2.1.2.1	Inv. Asistentes Personales
1.2.3.4.1.1	Baterías
1.2.3.4.1.2	Cargadores de celular
1.2.3.4.1.3	Otras estrategias
1.2.3.3.1.1	Comparación de Tecnologías
1.2.3.3.1.2	Necesidades de la Red
1.2.3.3.1.3	Capacidad de Módulos (ver si alcanzaban para las necesidades)
1.2.3.2.1.1	Comparación de Modelos
1.2.3.2.1.2	Proveedores
1.2.3.2.1.3	Carriers
1.2.3.2.1.4	Periféricos disponibles
1.2.2.2.2.1	Licencias de Desarrollo de Stores
1.2.2.2.2.2	Decisión de Plataformas
1.2.2.2.2.3	Plug-in's

Dentro de los papers que se leyeron y trabajaron se analizaron más en profundidad, incluso resumiendo, se encuentran:

On the features and challenges of security and privacy in 2013 Computer Networks.

Security in the Internet of Things- A Survey on Application Layer Protocols.

De los que solo se leyeron, pero por no resultar tan útiles como los anteriores no fueron resumidos se encuentran:

Survey of Standardized Protocols for the Internet of Things.

A su vez, se vieron videos de un curso provisto por Rafael al comienzo del curso.

Set up de los componentes por separado

Periféricos

Al principio del proyecto se analizó distintas formas de hacer que los enchufes se apagaran remotamente haciendo especial hincapié en que fuera una solución muy barata. Se comenzó pensando en relés alimentados por algún microcontrolador que se encargara de hacer de cerebro, cortando o proveyendo electricidad al relé según fuera la orden que se le daba. Dentro de los problemas que se encontraron a la hora de hacer esto era la alimentación que necesitaría un elemento como este. Si se localizaba una batería detrás de las fichas de luz, hacer el cambio de las baterías sería molesto para las personas que lo instalen y además resultaba extraño que estando al lado de una fuente de energía como un cable de 220 V se usaría batería. Por esto se siguió la idea de hacer un circuito que se alimentara de los 220 V, lo que hubiese significado un trabajo extenso y tedioso de electrónica. Se siguió investigando acerca del tema y se terminó encontrando un dispositivo que se encargaba de esta electrónica y usa un microcontrolador llamado ESP8266 que está ganando lugar en el mercado, tiene una gran comunidad detrás, es muy barato y la empresa (ITEAD) que vende esta solución (Sonoff), tiene una librería open source para desarrollar. La solución del Sonoff es más barato que comprar todos los elementos por separado.

Una vez que llegaron los Sonoff se empezó a realizar las tareas que se tenían previstas para los periféricos. Estas son:

WBS	Tarea
1.2.3.2.2.1.1	Pedido pruebas
1.2.3.2.3.1.1	Configuración de primera conexión
1.2.3.2.3.2	Publish-Subscribe prueba
1.2.3.1	Interruptor Eléctrico
1.2.3.3	Módulo de Comunicación
1.2.3.4	Fuente de alimentación

Servidor REST

Se desarrolló un servidor hosteado en el Raspberry Pi (Componente central) el cual se encargaría de aislar la red mesh-MQTT de los demás componentes del sistema, el mismo recibirá pedidos HTTP de componentes que deseen interactuar con los interruptores y traducirá estos pedidos a mensajes MQTT si los requerimientos de autenticación son satisfechos. También se encargará en un futuro de llevar una base de datos con los elementos registrados en la red y la configuración del usuario para cada interruptor (cuarto al que pertenece, nombre, etc.).

Luego de la obtención del primer prototipo se encontró que la configuración de un access-point dedicado a esta tarea debería haber sido considerado ya que el Raspberry Pi cuenta con un sólo módulo WiFi que se utilizó para la conexión con la red del hogar donde se encuentran los demás dispositivos del sistema D.U.D.E.; otro problema en la planificación de tareas fue el de no considerar el set up inicial de la configuración de credenciales de este servidor.

WBS	Tarea

1.2.1.4.2	Workspace Set-Up
1.2.1.4.3.1	Endpoints
1.2.1.5.2	Scripts para control de interruptores
1.2.1.6	Configuración de AP para red mesh

Aplicación Web/Móvil

Se optó por desarrollar la aplicación con tecnologías web debido a su fácil portabilidad a cualquier plataforma. Se había programado el comienzo del desarrollo de la interfaz web para etapas más tardías del proyecto pero los problemas de entregas de componentes provocaron la necesidad de adelantar esta tarea, obteniéndose así el primer prototipo de panel de control de los componentes totalmente integrado a los demás componentes.

WBS	Tarea
1.2.2.1.2	Workspace Set-Up
1.2.2.1.3.1	Integración con Back-End
1.2.2.1.3.2	Diseño de Interfaz Provisoria de Prendido/Apagado

Integración de componentes

Luego del desarrollo y testeo de los módulos se debió integrarlos y definir sus interacciones, esta tarea conllevó algunos problemas debido a la naturaleza mixta del proyecto, ya que, al utilizar software distribuido en componentes de hardware de distintas capacidades y características, se debió corregir problemas de configuración no detectados con las herramientas de desarrollo utilizadas; además de la dificultad de debuggear código de ejecución asíncrona donde los errores son logueados en terminales separadas.

Tareas Realizadas:

WBS	Tarea
1.2.1.4.3.3	Scripts para ctrl y conf de interruptores
1.2.1.4.3.2	Integración con Scripts controladores
1.2.1.3.3.2	Integración con Servidor REST

Conclusión

Al lograrse un circuito completo entre la interfaz gráfica de una computadora y los scripts controladores que activan las acciones del Sonoff ,corriendo cada componente en donde funcionará al final del proyecto y todas las investigaciones previas que se necesitaron para llegar a esto , así como los cambios en la arquitectura que fueron surgiendo a lo largo del semestre (mesh, broker fijo ahorrando un router con dongle en raspberry pi), se considera que se logró llegar a un 50% del proyecto.

Anexo II

Código Fuente

II.1. Sonoff

II.1.1. main.cpp

```
#include <FS.h>
#include <Arduino.h>
#include "credentials.h"
#include <ESP8266MQTTMesh.h>
#include <ESP8266WiFi.h>
#include <Bounce2.h>

#include <DNSServer.h>
#include <ESP8266WebServer.h>
#include <WiFiManager.h>

#include <Ticker.h>

#define FIRMWARE_ID      0x1337
#define FIRMWARE_VER     "0.1"

Ticker ticker;

const char* networks[2];
const char* network_password;
const char* mesh_password    = MESH_PASSWORD;
const char* base_ssid        = BASE_SSID;
const char* mqtt_server;
const char* mqtt_default_server = MQTT_DEFAULT_SERVER;
```

```
const int mqtt_port      = MQTT_PORT;
const int mesh_port       = MESH_PORT;
const int LED_PIN         = LED_BUILTIN;
const int RELAY_PIN        = RELAY_BUILTIN;
const int BUTTON_PIN       = BUTTON_BUILTIN;
const int SWITCH_PIN       = 14;

String ID = String(ESP.getChipId());
```



```
const char *out_topic    = OUT_TOPIC;
const char *in_topic     = IN_TOPIC;
unsigned long previousMillis = 0;
unsigned long previousInterrupt = 0;
const long interval = 1000;
int cnt = 0;
char status[3] = "ON";
```



```
String broker;
String ssid;
String pass;
```



```
char type[2] = "1"; //1 if sonoff 2 if sonoff dual
```



```
ESP8266MQTTMesh* mesh;
```



```
void callback(const char *topic, const char *msg);
```



```
boolean check_if_power_ON_Exists() {
    if (SPIFFS.exists("/ON_When_Power_On/")) {
        Serial.print("On");
        digitalWrite(LED_PIN, LOW);
        return true;
    } else {
        return false;
    }
}
```



```
boolean check_if_power_OFF_Exists()
```

```
if (SPIFFS.exists("/OFF_When_Power_On/")) {
    Serial.print("off");
    digitalWrite(LED_PIN, HIGH);
    return true;
} else{
    return false;
}

boolean check_if_first_connection_Exists(){
    if (SPIFFS.exists("/first_connection/")) {
        Serial.print("first_connection OK");
        return true;
    } else{
        return false;
    }
}

void tick()
{
    //toggle state
    int state = digitalRead(LED_PIN); // get the current state of GPIO1
    pin
    digitalWrite(LED_PIN, !state); // set pin to the opposite state
}

String getInformationFromSPIFFS(String baseString){//ej el base string de
    /broker/algo es broker
    Dir dir = SPIFFS.openDir(baseString);
    if(!dir.next()){
        Serial.println("no hay broker");
        return "";
    } else{
        Serial.print("Hay broker, es");
        File brokerFile=SPIFFS.open(dir.fileName(), "w");
        char* str=(char*)brokerFile.name(); // hay que parsear el file name
        que es un CONST char* ( un string incambiable) a un char* para que
        se puedan hacer operaciones
```

```
//const char * punt = dir.fileName();  
  
char * pch;  
pch = strtok (str,"/");  
int counter=0;  
String broker;  
while (pch != NULL) {  
    if(counter==1){  
        broker=String(pch);  
    }  
    Serial.println("entre");  
    pch = strtok (NULL, "/");  
    counter=counter+1;  
}  
Serial.println(broker);  
return broker;  
}  
  
}  
  
void sign_up_to_broker() {  
    char msg[255];  
    strlcpy(msg, "{", sizeof(msg));  
    strlcat(msg, "\"_id\"::", sizeof(msg));  
    strlcat(msg, ID.c_str(), sizeof(msg));  
    strlcat(msg, "\",\"type\"::", sizeof(msg));  
    strlcat(msg, type, sizeof(msg));  
    strlcat(msg, "}", sizeof(msg));  
    Serial.println("Sending sign in");  
    mesh->publish(  
        "new_device",  
        msg  
    );  
}  
  
void configModeCallback (WiFiManager *myWiFiManager) {  
    Serial.println("Entered config mode");  
    Serial.println(WiFi.softAPIP());  
    //if you used auto generated SSID, print it  
    Serial.println(myWiFiManager->getConfigPortalSSID());
```

```
//entered config mode, make led toggle faster
ticker.attach(0.2, tick);
}

void switch_state_changed () {
    unsigned long interrupt_time = millis();
    if (interrupt_time - previousInterrupt > 200)
    {
        previousInterrupt = interrupt_time;
        int state = digitalRead(RELAY_PIN);
        digitalWrite(RELAY_PIN, !state);
        char msg[255];
        strlcpy(msg, "{", sizeof(msg));
        strlcat(msg, "\_id\:", sizeof(msg));
        strlcat(msg, ID.c_str(), sizeof(msg));
        strlcat(msg, ",\\"state\:", sizeof(msg));
        if(state==0)strlcat(msg, "\"0\"", sizeof(msg));
        if(state==1)strlcat(msg, "\"1\"", sizeof(msg));
        strlcat(msg, "}", sizeof(msg));
        mesh->publish(
            "state_changed",
            msg
        );
    }
}

// Instantiate a Bounce object
Bounce debouncer = Bounce();
boolean threeSecondsTimmer=false;
unsigned long millisTime;

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

    // always use this to "mount" the filesystem
    if (! SPIFFS.begin()) {
        SPIFFS.format();
    }
}
```

```
Serial.println("Formatting");
if (! SPIFFS.begin()) {
    Serial.println("No esta andando los SPIFFS");
}
}

boolean bFirstConnection = check_if_first_connection_Exists();

//WiFiManager
//Local intialization. Once its business is done, there is no need to
keep it around
WiFiManager wifiManager;

//Setup the LED :
pinMode(LED_PIN,OUTPUT);
digitalWrite(LED_PIN, LOW);

//Setup the RELAY :
pinMode(RELAY_PIN,OUTPUT);
digitalWrite(RELAY_PIN, LOW);

//Setup the Switch :
pinMode(SWITCH_PIN, INPUT_PULLUP);
attachInterrupt(digitalPinToInterrupt(SWITCH_PIN), switch_state_changed
, CHANGE);

//passed the blocking method, managed to connect
if (!bFirstConnection) {
    // start ticker with 0.5 because we start in AP mode and try to
    connect
    ticker.attach(0.6, tick);

    wifiManager.setAPCallback(configModeCallback);

    wifiManager.setCaptivePortalEnable(false);
}
```

```
//fetches ssid and pass from eeprom and tries to connect
//if it does not connect it starts an access point with the specified
//name
//here "AutoConnectAP"
//and goes into a blocking loop awaiting configuration
 wifiManager.autoConnect("AutoConnectAP");

//if you get here you have connected to the WiFi
Serial.println("connected...yeey :)");
String sSsid = "/ssid/" + WiFi.SSID();
String sPass = "/pass/" + WiFi.psk();
Serial.println(sSsid);
Serial.println(sPass);

File fSsid = SPIFFS.open(sSsid.c_str(), "w");
if (!fSsid) { //chequeo si se grabo bien
    Serial.print("Couldn't write file ssid");
} else{
    fSsid.close();
    Serial.println("Se creo el file ssid");
}

File fPass = SPIFFS.open(sPass.c_str(), "w");
if (! fPass) { //chequeo si se grabo bien
    Serial.print("Couldn't write file password");
} else{
    fPass.close();
    Serial.println("Se creo el file password");
}

File fConnected = SPIFFS.open("/first_connection/", "w"); //por
//defecto esta ON
if (! fConnected) {
    Serial.println("Couldn't write /first_connection/");
}
fConnected.close();
```

```
bFirstConnection=true;

Serial.println("Registered first connection");

delay(500);

ESP.restart();

}else{

    Serial.println("Already Registered first connection");
}

boolean bONExists = check_if_power_ON_Exists();
boolean bOFFExists = check_if_power_OFF_Exists();

//ALEXIS
Serial.print("My MAC is: ");
Serial.println(WiFi.softAPmacAddress().c_str());

// Setup the button with an internal pull-up :
pinMode(BUTTON_PIN, INPUT_PULLUP);
// After setting up the button, setup the Bounce instance :
debouncer.attach(BUTTON_PIN);
debouncer.interval(5); // interval in ms

//check if poweroff or on state exists in spiffs
if (!bONExists && !bOFFExists) {

    File f = SPIFFS.open("/ON_When_Power_On/", "w");//por defecto esta ON
    if (! f) {
        Serial.print("Couldn't write /ON_When_Power_On/");
    }
    Serial.println(SPIFFS.exists("/ON_When_Power_On/"));
    f.close();
    digitalWrite(LED_PIN, LOW);
    Serial.println("!bONExists && !bOFFExists");
} else {
    if (bONExists) {
        Serial.println("ONExists");
    }
}
```

```
    digitalWrite(LED_PIN, LOW);

}else{
    if (bOFFExists) {
        Serial.println("OFExists");
        digitalWrite(LED_PIN, HIGH);
    }else{
        Serial.println("Se rompio algo");
    }
}

}

broker=GetInformationFromSPIFFS("/broker/");// importante que sea
// global esta variable, si no se rompe todo
if(broker!=""){//getInformationFromSPIFFS("/broker/")
    mqtt_server=broker.c_str();
    Serial.println("es distinto de nulo");
    Serial.print("el c_str es ");
    Serial.println(broker.c_str());
}else{
    mqtt_server= mqtt_default_server;
    Serial.println("es vacio, agarro el default");
}

ssid=GetInformationFromSPIFFS("/ssid/");// importante que sea global
// esta variable, si no se rompe todo
if(ssid!=""){
    Serial.print("el c_str es de ssid es: ");
    Serial.println(ssid.c_str());
    networks[0] = ssid.c_str();
    networks[1] = NULL;
}else{
    Serial.println("es vacio el ssid!! error");
}

pass=GetInformationFromSPIFFS("/pass/");// importante que sea global
// esta variable, si no se rompe todo
network_password=pass.c_str();
Serial.print("el c_str es de pass es: ");
```

```
Serial.println(pass.c_str());  
  
Serial.print("El server es ");  
Serial.println(mqtt_server);  
Serial.print("El broker es ");  
Serial.println(broker);  
  
mesh = ESP8266MQTTMesh::Builder(networks, network_password, (char*)  
    mqtt_server, mqtt_port)  
.setVersion(FIRMWARE_VER, FIRMWARE_ID)  
.setMeshPassword(mesh_password)  
.setBaseSSID(base_ssid)  
.setMeshPort(mesh_port)  
.setTopic(in_topic, out_topic)  
.buildptr();  
Serial.println("hi");  
Serial.println(NETWORK_PASSWORD);  
Serial.println(mqtt_server);  
mesh->setCallback(callback);  
mesh->begin();  
  
}  
  
void loop() {  
  
    unsigned long currentMillis = millis();  
  
    if (currentMillis - previousMillis >= interval) {  
        if (mesh->connected()) {  
            if (!SPIFFS.exists("/signed_in_broker/")) {  
                sign_up_to_broker();  
                previousMillis = currentMillis;  
            }  
            String cntStr = String(cnt);  
            String msg = "hello from " + ID + " cnt: " + cntStr;  
            mesh->publish(ID.c_str(), msg.c_str());  
            cnt++;  
        }  
    }  
}
```

```
    } else {
        //return;
        Serial.println("Not Connected to mesh");
    }
    previousMillis = currentMillis;
}

// Update the Bounce instance :
debouncer.update();
// Get the updated value :
int value = debouncer.read();

millisTime=millis();

int LedStateBeforeButtonPressed;
// Turn on or off the LED as determined by the state :
if ( value == LOW ) {
    LedStateBeforeButtonPressed=digitalRead(LED_PIN);
    digitalWrite(LED_PIN, HIGH );
    float secondsOnWhile=0.0;
    int contadorSegundos=0;
    //count amount of seconds with button pressed
    while(value == LOW && contadorSegundos<2 ){
        debouncer.update();
        value = debouncer.read();
        if((millis()-millisTime)/1000.0>1.0){
            contadorSegundos=contadorSegundos+1;
            millisTime=millis();
        }
        delay(100); //hace que no se active el watchdog
    }
    if(contadorSegundos==2){
        Serial.println("boton apretado 2 segundos");

        SPIFFS.remove("/first_connection/");

        //si llego aca seguro que hay una ssid y pass
        Dir dSsid = SPIFFS.openDir("/ssid/");
    }
}
```

```
Dir dPass = SPIFFS.openDir("/pass/");

if(dSsid.next()) {
    File ssidFile=SPIFFS.open(dSsid.fileName(), "w");
    Serial.print("Estoy tratando de borrar este file: ");
    String m=dSsid.fileName();
    Serial.println(dSsid.fileName());
    SPIFFS.remove(dSsid.fileName());
    Serial.print("Estoy tratando de borrar este file: ");
    Serial.print("Este es el resultado del exists: ");
    Serial.println(SPIFFS.exists(m));
}

else{
    Serial.println("Something went wrong");
}

if(dPass.next()) {
    File passFile=SPIFFS.open(dPass.fileName(), "w");
    String m=dSsid.fileName();
    Serial.print("Estoy tratando de borrar este file: ");
    Serial.println(dPass.fileName());
    SPIFFS.remove(dPass.fileName());
    Serial.print("Este es el resultado del exists: ");
    Serial.println(SPIFFS.exists(m));
}

else{
    Serial.println("Something went wrong");
}

WiFiManager wifiManager;

wifiManager.resetSettings();

delay(500);

ESP.restart();
```

```
/*Dir dir = SPIFFS.openDir("/broker/");

if(!dir.next()) {
    //no encontre un broker
    Serial.println("no hay broker");

    //leo que hay en buffer y si hay algo lo pongo en el inputBuffer
    String input=Serial.readString(); //se lo queda en buffer,
    recordar escribir en la terminal antes de hacer lo de los 2
    segundos

    Serial.println(input);

    if(input!=""){//en un broker en el input para escribir
        if(input=="format") {
            SPIFFS.remove("/signed_in_broker/");
        }else{
            char filename[100];
            File f = SPIFFS.open(input.c_str(), "w");
            if (!f) {//chequeo si se grabo bien
                Serial.print("Couldn't write broker");
            }else{
                f.close();
                Serial.print("Se creo el broker");
                //rebooteo para que se conecte a ese broker
                ESP.restart(); //the boot mode:(1,7) problem is known and
                only happens at the first restart after serial flashing.
                //if you do one manual reboot by power or RST
                //pin all will work more info see: https://github.com/esp8266/Arduino/issues/1017
            }
        }
    }else{//si no tengo nada escrito, necesito input para grabar el
        broker en los SPIFFS
        Serial.println("escribi algo en la terminal!!");
    }
}else{
    //elimino el broker de los SPIFSS
    Serial.print("Hay broker, es");
    Serial.println(dir.fileName());
    File brokerFile=SPIFFS.open(dir.fileName(), "w");
}
```

```
    Serial.print("lo voy a borrar");
    SPIFFS.remove(dir.fileName());
} */
}

digitalWrite(LED_PIN, LedStateBeforeButtonPressed );//restore button
state to the state prior to button being pressed
}

}

void callback(const char *topic, const char *msg) { //entered callback
    because last digit in device_in/mesh_esp8266-X matched
    // topic is the command and msg the payload
    //reference in tasmota messages for command and payload https://github.
    com/arendst/Sonoff-Tasmota/wiki/Commands
    int iDigitalReadLED;
    char sDigitalReadLED[8];
    int iDigitalReadRelay;
    char sDigitalReadRelay[8];
    String sTopic=String(topic);
    String sPayload=String(msg);
    if (digitalRead(LED_PIN) == LOW)
    {
        iDigitalReadLED=0;
    }else{
        iDigitalReadLED=1;
    }
    if (digitalRead(RELAY_PIN) == LOW)
    {
        iDigitalReadRelay=0;
    }else{
        iDigitalReadRelay=1;
    }
    itoa(iDigitalReadLED, sDigitalReadLED, 10);
    itoa(iDigitalReadRelay, sDigitalReadRelay, 10);

    if(sTopic == "LedState") {
        if(sPayload == "") {

```

```
if(iDigitalReadLED==0)mesh->publish(ID.c_str(),"ON");//aca no va a
    ir ID.c_str() creo
if(iDigitalReadLED==1)mesh->publish(ID.c_str(),"OFF");//aca no va a
    ir ID.c_str() creo
}
}

if(sTopic == "Power"){//Relay HIGH to turn on
previousInterrupt = millis(); // a change in the relay sseems to
    trigger the interruption, this avoids a change
if(sPayload == "") mesh->publish(ID.c_str(),sDigitalReadRelay);
if(sPayload == "0"){
    digitalWrite(RELAY_PIN, LOW);
    mesh->publish(ID.c_str(),"PoweredOFF");
}
if(sPayload == "1"){
    digitalWrite(RELAY_PIN, HIGH);
    Serial.println(RELAY_PIN);
    mesh->publish(ID.c_str(),"PoweredON");
}
}

if(sTopic == "PowerOnState"){
if(sPayload == ""){
    boolean bONExists=check_if_power_ON_Exists();
    boolean bOFFExists=check_if_power_OFF_Exists();
    if (bONExists){
        Serial.println("ONExists");
        mesh->publish(ID.c_str(),"ON_When_Power_On");
    }else{
        if (bOFFExists){
            Serial.println("OFExists");
            mesh->publish(ID.c_str(),"OFF_When_Power_On");
        }else{
            Serial.println("Se rompio algo");
        }
    }
}

if(sPayload == "0"){


```

```
SPIFFS.remove("/ON_When_Power_On/");

File f = SPIFFS.open("/OFF_When_Power_On/", "w");
if (!f) {
    Serial.print("Couldn't write /OFF_When_Power_On/");
}else{
    f.close();
    mesh->publish(ID.c_str(), "Power_On_Changed");
}

if(sPayload == "1"){
    SPIFFS.remove("/OFF_When_Power_On/");
    File f = SPIFFS.open("/ON_When_Power_On/", "w");
    if (!f) {
        Serial.print("Couldn't write /ON_When_Power_On/");
    }else{
        f.close();
        mesh->publish(ID.c_str(), "Power_On_Changed");
    }
}

if(sTopic == "LedPower"){//LED LOW to turn on
    if(sPayload == "") mesh->publish(ID.c_str(), sDigitalReadLED);
    if(sPayload == "0"){
        digitalWrite(LED_PIN, HIGH);
        mesh->publish(ID.c_str(), "PoweredOFF");
    }
    if(sPayload == "1"){
        digitalWrite(LED_PIN, LOW);
        mesh->publish(ID.c_str(), "PoweredON");
    }
}

if (sTopic == "correctly_signed_in"){ // ojo que esto es un else if y
    no un if como los demas, no creo que aporte nada que sea un else if
    y no un if

    File f = SPIFFS.open("/signed_in_broker/", "w");
    if (!f) {
        Serial.print("Couldn't write /signed_in_broker/");
    }
}
```

```

    f.print("1\n");
    f.close();
}
}

```

II.1.2. credentials.h.j2

```

#define NETWORK_LIST \
    "{{ SSID_1 }}", \
    "{{ SSID_2 }}", \
    "", \
    // this need to be ended by an empty string

#define NETWORK_PASSWORD "{{ NETWORK_PASSWORD }}"
#define MESH_PASSWORD    "{{ MESH_PASSWORD }}"
#define BASE_SSID        "{{ BASE_SSID }}"
#define MQTT_DEFAULT_SERVER "{{ MQTT_DEFAULT_SERVER }}"
#define MQTT_PORT         {{ MQTT_PORT }}
#define MESH_PORT         {{ MESH_PORT }}
#define OUT_TOPIC         "{{ OUT_TOPIC }}"
#define IN_TOPIC          "{{ IN_TOPIC }}"
#define LED_BUILTIN        {{ LED_BUILTIN }}
#define RELAY_BUILTIN      {{ RELAY_BUILTIN }}
#define BUTTON_BUILTIN     {{ BUTTON_BUILTIN }}

```

II.1.3. platformio.ini

```

[platformio]
env_default = nossl

[env:ssl]
platform = espressif8266@4f59cef
framework = arduino
board = esp01_1m

#build_flags = -DASYNC_TCP_SSL_ENABLED=1 -DGATEWAY_ID=10499051 -DLED_PIN
=2 -g
build_flags = -DASYNC_TCP_SSL_ENABLED=1
lib_deps = ESP8266MQTTMesh

```

```
[env:nossal]
platform = https://github.com/platformio/platform-espressif8266.git#
    feature/stage
framework = arduino
board = esp01_1m
lib_deps = ESP8266MQTTMesh
board_flash_mode = dout
#build_flags = -DLED_PIN=2 -g
```

II.2. Raspberry Pi

II.2.1. mosquitto.conf

```
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

pid_file /var/run/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log
# allow_anonymous false
password_file /etc/mosquitto/pwfile
listener 1883
```

II.2.2. api.py

```
"""
```

The server in charge of receiving HTTP requests and transfer them to the devices via MQTT.

```
"""

import datetime
import eventlet
import json
import logging
import pickle
import subprocess

from flask import Flask, render_template, request, jsonify
from flask_mqtt import Mqtt
from flask_pymongo import PyMongo

from logging.handlers import RotatingFileHandler
from mongo_db_utils import *
from crossdomain import crossdomain
from auth_decorator import requires_auth
from settings import settings

eventlet.monkey_patch()

CONFIG_REQUIRED_FIELDS = settings["CONFIG_REQUIRED_FIELDS"]

NEW_DEVICE_REQUIRED_FIELDS = settings["NEW_DEVICE_REQUIRED_FIELDS"]

NEW_GROUP_REQUIRED_FIELDS = settings["NEW_GROUP_REQUIRED_FIELDS"]

MQTT_CHANNELS = settings["MQTT_CHANNELS"]

SET_LED_STATE_REQUIRED_FIELDS = settings["SET_LED_STATE_REQUIRED_FIELDS"]

SET_SWITCH_STATE_REQUIRED_FIELDS = settings[
    "SET_SWITCH_STATE_REQUIRED_FIELDS"]

SET_GROUP_STATE_REQUIRED_FIELDS = settings["]"
```

```
SET_GROUP_STATE_REQUIRED_FIELDS"]  
  
ADD_CRON_REQUIRED_FIELDS = settings["ADD_CRON_REQUIRED_FIELDS"]  
  
DELETE_CRONS_REQUIRED_FIELDS = settings["DELETE_CRONS_REQUIRED_FIELDS"]  
  
# Parameters for SSL enabled  
# app.config['MQTT_BROKER_PORT'] = 8883  
# app.config['MQTT_TLS_ENABLED'] = True  
# app.config['MQTT_TLS_INSECURE'] = True  
# app.config['MQTT_TLS_CA_CERTS'] = 'ca.crt'  
  
app = Flask(__name__)  
app.config['MONGO_DBNAME'] = settings["MONGO_DBNAME"]  
mongo = PyMongo(app)  
  
config = get_config_doc(app, mongo)  
app.config['SECRET'] = settings["MONGO_SECRET"]  
app.config['TEMPLATES_AUTO_RELOAD'] = True  
app.config['MQTT_BROKER_URL'] = config.get('MQTT_BROKER_URL', '0.0.0.0')  
app.config['MQTT_BROKER_PORT'] = int(config.get('MQTT_BROKER_PORT', 1883))  
)  
app.config['MQTT_USERNAME'] = config.get('MQTT_USERNAME', None)  
app.config['MQTT_PASSWORD'] = config.get('MQTT_PASSWORD', None)  
app.config['MQTT_REFRESH_TIME'] = 1 # refresh time in seconds  
app.config['MQTT_TLS_ENABLED'] = False  
print(app.config)  
app.logger.info(app.config)  
mqtt = Mqtt(app)  
mqtt.subscribe('device_out/test')  
print(MQTT_CHANNELS['new_device'])  
mqtt.subscribe(MQTT_CHANNELS['new_device'])  
mqtt.subscribe(MQTT_CHANNELS['state_changed'])  
mqtt.publish('device_out', 'message')  
  
  
class InvalidRequest(Exception):  
    pass
```

```
@app.errorhandler(InvalidRequest)
def handle_invalid_usage(error):
    return jsonify({'status': 'error', 'error': str(error)}), 400

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/publish', methods=['POST', 'OPTIONS'])
@crossdomain(origin='*')
def handle_publish():
    if not ('Authentication-Token' in request.headers and
            request.headers.get('Authentication-Token', '') == settings['
                FLASK_AUTH_TOKEN']):
        return jsonify({'status': 'Unauthorized attempt'}), 401
    data = request.get_json(force=True)
    mqtt.publish(data['topic'], data['message'])
    return jsonify({'status': 'OK'}), 200

@app.route('/config', methods=['GET', 'POST', 'OPTIONS'])
@crossdomain(origin='*')
def handle_config():
    if request.method == 'POST':
        if not ('Authentication-Token' in request.headers and
                request.headers.get('Authentication-Token', '') ==
                settings['FLASK_AUTH_TOKEN']):
            return jsonify({'status': 'Unauthorized attempt'}), 401
        config = request.get_json(force=True)
        app.logger.info(config)
        print(config)
        check_fields(config, CONFIG_REQUIRED_FIELDS)
        update_config_doc(app, mongo, config)
    return jsonify({'status': 'Configuration updated correctly!'})
```

```
    200

else:
    return jsonify(get_config_doc(app, mongo)), 200

@app.route('/subscribe', methods=['POST'])
def handle_subscribe():
    data = request.get_json(force=True)
    mqtt.subscribe(data['topic'])
    return jsonify({'status': 'subscribed'}), 200

@app.route('/get_device', methods=['GET', 'OPTIONS'])
@crossdomain(origin='*')
def get_device():
    if not ('Authentication-Token' in request.headers and
        request.headers.get('Authentication-Token', '') == settings['
            FLASK_AUTH_TOKEN']):
        return jsonify({'status': 'Unauthorized attempt'}), 401
    try:
        _id = request.args.get("_id")
        if not _id:
            raise InvalidRequest("Query string param _id is necessary
                for this request.")
        else:
            device = get_device_from_db(app, mongo, _id, True)
    except Exception as e:
        raise InvalidRequest(e)
    return jsonify(device), 200

@app.route('/get_devices', methods=['GET', 'OPTIONS'])
@crossdomain(origin='*')
def get_devices():
    if not ('Authentication-Token' in request.headers and
        request.headers.get('Authentication-Token', '') == settings['
            FLASK_AUTH_TOKEN']):
        return jsonify({'status': 'Unauthorized attempt'}), 401
```

```
try:
    devices = get_devices_from_db(app, mongo, True)
except Exception as e:
    raise InvalidRequest(e)
return jsonify(devices), 200

@app.route('/create_group', methods=['POST', 'OPTIONS'])
@crossdomain(origin='*')
def handle_create_group():
    if not ('Authentication-Token' in request.headers and
            request.headers.get('Authentication-Token', '') == settings['
                FLASK_AUTH_TOKEN']):
        return jsonify({'status': 'Unauthorized attempt'}), 401
    try:
        data = request.get_json(force=True)
        return create_group(data)
    except Exception as e:
        raise InvalidRequest(e)

@app.route('/get_group', methods=['GET', 'OPTIONS'])
@crossdomain(origin='*')
def handle_get_group():
    if not ('Authentication-Token' in request.headers and
            request.headers.get('Authentication-Token', '') == settings['
                FLASK_AUTH_TOKEN']):
        return jsonify({'status': 'Unauthorized attempt'}), 401
    try:
        _id = request.args.get("_id")
        if not _id:
            raise InvalidRequest("Query string param _id is necessary
for this request.")
        else:
            group = get_group_from_db(app, mongo, _id)
    except Exception as e:
        raise InvalidRequest(e)
    return jsonify(group), 200
```

```
@app.route('/get_groups', methods=['GET', 'OPTIONS'])
@crossdomain(origin='*')
def handle_get_groups():
    if not ('Authentication-Token' in request.headers and
            request.headers.get('Authentication-Token', '') == settings['
                FLASK_AUTH_TOKEN']):
        return jsonify({'status': 'Unauthorized attempt'}), 401
    try:
        groups = get_groups_from_db(app, mongo, False)
    except Exception as e:
        raise InvalidRequest(e)
    return jsonify(groups), 200

@app.route('/get_groups_with_devices', methods=['GET', 'OPTIONS'])
@crossdomain(origin='*')
def handle_get_groups_with_devices():
    if not ('Authentication-Token' in request.headers and
            request.headers.get('Authentication-Token', '') == settings['
                FLASK_AUTH_TOKEN']):
        return jsonify({'status': 'Unauthorized attempt'}), 401
    try:
        groups = get_groups_from_db(app, mongo, True)
    except Exception as e:
        raise InvalidRequest(e)
    return jsonify(groups), 200

@app.route('/edit_device', methods=['POST', 'OPTIONS'])
@crossdomain(origin='*')
def handle_edit_device():
    if not ('Authentication-Token' in request.headers and
            request.headers.get('Authentication-Token', '') == settings['
                FLASK_AUTH_TOKEN']):
        return jsonify({'status': 'Unauthorized attempt'}), 401
    try:
```

```
device = request.get_json(force=True)
print(device)
app.logger.info(device)
result = edit_device(app, mongo, device)
except Exception as e:
    raise InvalidRequest(e)
if result:
    return jsonify({'status': 'OK'}), 200
else:
    return jsonify({'status': 'Device not found.'}), 400

@app.route('/edit_group', methods=['POST', 'OPTIONS'])
@crossdomain(origin='*')
def handle_edit_group():
    if not ('Authentication-Token' in request.headers and
            request.headers.get('Authentication-Token', '') == settings['
                FLASK_AUTH_TOKEN']):
        return jsonify({'status': 'Unauthorized attempt'}), 401
    try:
        group = request.get_json(force=True)
        print(group)
        app.logger.info(group)
        result = edit_group(app, mongo, group)
    except Exception as e:
        raise InvalidRequest(e)
    if result:
        return jsonify({'status': 'OK'}), 200
    else:
        return jsonify({'status': 'Group not found.'}), 400

@app.route('/set_led_state', methods=['POST', 'OPTIONS'])
@crossdomain(origin='*')
def handle_change_led_state():
    if not ('Authentication-Token' in request.headers and
            request.headers.get('Authentication-Token', '') == settings['
                FLASK_AUTH_TOKEN']):

```

```

        return jsonify({'status': 'Unauthorized attempt'}), 401
    try:
        data = request.get_json(force=True)
        check_fields(data, SET_LED_STATE_REQUIRED_FIELDS)
        change_led_state(data['device_id'], data['state'])
    except Exception as e:
        raise InvalidRequest(e)
    return jsonify({'status': 'OK'}), 200

@app.route('/set_switch_state', methods=['POST', 'OPTIONS'])
@crossdomain(origin='*')
def handle_change_switch_state():
    if not ('Authentication-Token' in request.headers and
            request.headers.get('Authentication-Token', '') == settings['
                FLASK_AUTH_TOKEN']):
        return jsonify({'status': 'Unauthorized attempt'}), 401
    try:
        data = request.get_json(force=True)
        check_fields(data, SET_SWITCH_STATE_REQUIRED_FIELDS)
        change_switch_state(data['device_id'], data['state'])
    except Exception as e:
        raise InvalidRequest(e)
    return jsonify({'status': 'OK'}), 200

@app.route('/set_group_state', methods=['POST', 'OPTIONS'])
@crossdomain(origin='*')
def handle_set_group_state():
    if not ('Authentication-Token' in request.headers and
            request.headers.get('Authentication-Token', '') == settings['
                FLASK_AUTH_TOKEN']):
        return jsonify({'status': 'Unauthorized attempt'}), 401
    try:
        data = request.get_json(force=True)
        check_fields(data, SET_GROUP_STATE_REQUIRED_FIELDS)
    except Exception as e:
        raise InvalidRequest(e)

```

```
    return change_group_state(data['group_id'], data['state'])

@app.route('/add_cron', methods=['GET', 'POST', 'OPTIONS'])
@crossdomain(origin='*')
def handle_add_cron():
    if request.method == 'POST':
        if not ('Authentication-Token' in request.headers and
                request.headers.get('Authentication-Token', '') ==
                settings['FLASK_AUTH_TOKEN']):
            return jsonify({'status': 'Unauthorized attempt'}), 401
        cron = request.get_json(force=True)
        check_fields(cron, ADD_CRON_REQUIRED_FIELDS)
        if(cron["type"]=="device"):
            add_cron(str(cron["_id"]),cron["action"],cron["cron"])
        else:
            add_group_cron(str(cron["_id"]),cron["action"],cron["cron"])
    return jsonify({'status': 'Cron added correctly!'}), 200
else:
    return jsonify({'status': 'OK'}), 200

@app.route('/delete_crongs', methods=['GET', 'POST','OPTIONS'])
@crossdomain(origin='*')
def handle_delete_crongs():
    if request.method == 'POST':
        if not ('Authentication-Token' in request.headers and
                request.headers.get('Authentication-Token', '') ==
                settings['FLASK_AUTH_TOKEN']):
            return jsonify({'status': 'Unauthorized attempt'}), 401
        data = request.get_json(force=True)
        check_fields(data, DELETE_CRONGS_REQUIRED_FIELDS)
        if(data["type"]=="device"):
            delete_crongs(str(data["_id"]))
        else:
            delete_group_crongs(str(data["_id"]))
    return jsonify({'status': 'Cron deleted correctly!'}), 200
else:
```

```

    return jsonify({'status': 'OK'}), 200

@mqtt.on_message()
def handle_mqtt_message(client, userdata, message):
    print(message.payload)
    try:
        json.loads(message.payload)
        data = dict(
            topic=message.topic,
            payload=json.loads(message.payload)
        )
        topic_split = message.topic.split('/')
        sub_channel = topic_split[1]
        app.logger.info("message: {}".format(data))
        print("message: {}".format(data))
        if len(topic_split) == 3:
            if topic_split[2] == 'new_device':
                check_fields(data["payload"], NEW_DEVICE_REQUIRED_FIELDS)
                mqtt.publish(
                    '{}/{}/correctly_signed_in'.format(settings["IN_TOPIC"],
                                                       sub_channel),
                    202)
        return add_device(data['topic'], data['payload'])
    elif topic_split[2] == 'state_changed':
        update_device_state(app, mongo, data["payload"]["_id"],
                            data["payload"]["state"])
    except Exception as e:
        print("error: {}".format(e))

@mqtt.on_log()
def handle_logging(client, userdata, level, buf):
    print(client, userdata, level, buf)
    app.logger.info("{} {} {} {}".format(client, userdata, level, buf))

def add_device(topic, payload):
    """
    topic: String containing topic. eg: device_out/something

```

```
payload: JSON eg: {
    "_id": {{ ESP chip ID }},
    "type": {{ 1 for sonoff, 2 for dual}}
}

```
check_fields(payload, NEW_DEVICE_REQUIRED_FIELDS)
with app.app_context():
 sub_channel = topic.split('/')[-1]
 if mongo.db.device.find({'_id': payload.get('_id')}).count() == 0:
 device = json.loads(open(settings['DEVICE_TEMPLATE_FILE']).read())
 device['name'] = datetime.datetime.today().strftime("Agregado el %Y-%m-%d %H:%M")
 device['_id'] = payload['_id']
 device['type'] = payload['type']
 device['sub_channel'] = sub_channel
 mongo.db.device.insert(device)
 mqtt.publish(
 '{}/{}/{}{}/correctly_signed_in'.format(settings["IN_TOPIC"]),
 sub_channel),
 200)
 else:
 mqtt.publish(
 '{}/{}/{}{}/correctly_signed_in'.format(settings["IN_TOPIC"]),
 sub_channel),
 202)

def change_led_state(_id, state):
 device = get_device_from_db(app, mongo, _id, False)
 mqtt.publish(
 '{}/{}/{}{}'.format(settings['IN_TOPIC'], device["sub_channel"],
 settings['LED_STATE_TOPIC']),
 state
)
 update_device_state(app, mongo, _id, state)
```

```

def change_switch_state(_id, state):
 device = get_device_from_db(app, mongo, _id, False)
 mqtt.publish(
 '{}/{}{}'.format(settings['IN_TOPIC'], device["sub_channel"],
 settings['SWITCH_STATE_TOPIC']),
 state
)
 update_device_state(app, mongo, _id, state)

def change_group_state(_id, state):
 try:
 group = get_group_from_db(app, mongo, _id)
 for device in group["devices"]:
 change_switch_state(device["_id"], state)
 return jsonify({'status': 'OK.'}), 200
 except Exception as e:
 raise InvalidRequest(e)

def create_group(data):
 check_fields(data, NEW_GROUP_REQUIRED_FIELDS)
 with app.app_context():
 group = json.loads(open(settings['GROUP_TEMPLATE_FILE']).read())
 group['name'] = str(data['name']).lower()
 group = create_group_if_necessary(app, mongo, group)
 if data.get('devices'):
 for device in data['devices']:
 try:
 _id = int(device)
 add_device_to_group(app, mongo, _id, group)
 except Exception as e:
 continue
 return jsonify({'status': 'OK.'}), 200

def add_cron(id, action, cron):
 str=""" + cron + " curl -H \\\"Authentication-Token:"
```

```
f1a8659701bbbc7196940761c1d55c3b9a25fb6569a863df\\\" -X POST -d
\'{\\"state\\\":\\\""+action+"\\\",\\\"device_id\\\":\\\""+id+
\\\"}\\' http://localhost:5000/set_switch_state\\"
str2=str+" "+id
subprocess.call("${pwd}/makeCron.sh "+str2, shell=True)

def delete_crons(id):
 str1="device "+id
 subprocess.call("${pwd}/removeCron.sh "+str1, shell=True)

def add_group_cron(groupId,action,cron):
 group = get_group_from_db(app, mongo, groupId)
 for device in group.get("devices"):
 add_cron(str(device["_id"]),action,cron)

def delete_group_crons(groupId):
 group = get_group_from_db(app, mongo, groupId)
 for device in group.get("devices"):
 delete_crons(str(device["_id"]))

def check_fields(data_json, req_fields):
 for field in req_fields:
 value = data_json.get(field, None)
 if value is None:
 raise InvalidRequest(
 u'Request body needs a "{}" field.'.format(field))
 if not str(value).strip():
 raise InvalidRequest(
 u'The "{}" field cannot be empty or spaces.'.format(field
)))
if __name__ == '__main__':
 handler = RotatingFileHandler('/tmp/dude_flask.log', maxBytes=10000,
```

```

 backupCount=1)

handler.setLevel(logging.INFO)
app.logger.addHandler(handler)
app.run(host='0.0.0.0', port=5000, use_reloader=True, debug=True)

```

### II.2.3. mongo\_db\_utils.py

```

import json

from bson.objectid import ObjectId

def check_fields(data_json, req_fields):
 for field in req_fields:
 value = data_json.get(field, None)
 if value is None:
 raise InvalidRequest(
 u'Request body needs a "{}" field.'.format(field))
 if not str(value).strip():
 raise InvalidRequest(
 u'The "{}" field cannot be empty or spaces.'.format(field))
))

def format_object_id(objects):
 if isinstance(objects, list):
 for obj in objects:
 if obj.get("_id"):
 obj["_id"] = str(obj.get("_id"))
 obj["_id"] = obj["_id"].split(".") [0]
 else:
 if objects.get("_id"):
 objects["_id"] = str(objects.get("_id"))
 objects["_id"] = objects["_id"].split(".") [0]
 return objects

def update_config_doc(app, mongo, config):
 with app.app_context():
 mongo.db.config.update(
 { '_id': 1 },
 {

```

```
'$set': {
 "MQTT_BROKER_URL": config["MQTT_BROKER_URL"],
 "MQTT_USERNAME": config["MQTT_USERNAME"],
 "MQTT_PASSWORD": config["MQTT_PASSWORD"],
 "MQTT_MESH_PASSWORD": config["MQTT_MESH_PASSWORD"],
 "MQTT_BROKER_PORT": config["MQTT_BROKER_PORT"],
 "DONGLE_SSID": config["DONGLE_SSID"],
 "DONGLE_PASSWORD": config["DONGLE_PASSWORD"],
 "REGISTERED_USERS": config["REGISTERED_USERS"]
}
},
upsert=True
)

def get_config_doc(app, mongo):
 config = {}
 with app.app_context():
 config_cursor = mongo.db.config.find({'_id': 1})
 if config_cursor.count() > 0:
 config = next(config_cursor)
 return config

def get_groups_from_db(app, mongo, with_devices):
 groups = []
 with app.app_context():
 for group in mongo.db.device_group.find():
 groups.append(group)
 groups = format_object_id(groups)
 if with_devices:
 for group in groups:
 group["devices"] = get_group_devices(app, mongo, group['_id'])
 return groups

def get_group_from_db(app, mongo, _id):
 groups = []
 with app.app_context():
 for group in mongo.db.device_group.find({'_id': { '$in': [_id,
```

```
 ObjectId(_id)] }) :
 groups.append(group)
group = format_object_id(groups[0])
group["devices"] = get_group_devices(app, mongo, group['_id'])
return group

def get_devices_from_db(app, mongo, external):
 devices = []
 with app.app_context():
 if external:
 mongo_resp = mongo.db.device.find({}, {'sub_channel': 0})
 else:
 mongo_resp = mongo.db.device.find()
 for device in mongo_resp:
 devices.append(device)
 devices = format_object_id(devices)
 return devices

def get_device_from_db(app, mongo, _id, external):
 devices = []
 with app.app_context():
 if external:
 print('external')
 mongo_resp = mongo.db.device.find(
 {
 '_id': int(_id)
 },
 {
 'sub_channel': 0
 }
)
 else:
 mongo_resp = mongo.db.device.find({'_id': int(_id)})
 for device in mongo_resp:
 devices.append(device)
 device = {}
 if devices:
 device = format_object_id(devices[0])
 else:
```

```
 print("No devices found")
 return device

def get_group_devices(app, mongo, group_id):
 devices = []
 with app.app_context():
 mongo_resp = mongo.db.device.aggregate(
 [
 {'$unwind': '$groups'},
 {
 '$match': {
 'groups': group_id
 }
 },
 {
 '$project': {
 'state': '$state',
 'name': '$name',
 'groups': '$groups',
 '_id': '$_id',
 'type': '$type'
 }
 }
]
)
 for device in mongo_resp:
 devices.append(device)
 devices = format_object_id(devices)
 return devices

def create_group_if_necessary(app, mongo, group):
 name = group["name"]
 groups = []
 with app.app_context():
 for group in mongo.db.device_group.find({'name': name}):
 groups.append(group)
 groups = format_object_id(groups)
 if len(groups) == 0:
 group["_id"] = mongo.db.device_group.insert(group)
```

```
group = format_object_id(group)
else:
 group = groups[0]
return group

def add_device_to_group(app, mongo, _id, group):
 device = get_device_from_db(app, mongo, _id, False)
 device_groups = device["groups"]
 device_groups.append(group["_id"])
 device_groups = list(set(device_groups))
 with app.app_context():
 mongo.db.device.update(
 {'_id': int(_id)},
 { '$set': { 'groups' : device_groups}}
)

def update_device_state(app, mongo, _id, state):
 state = False if state == "0" else True
 device = get_device_from_db(app, mongo, _id, False)
 with app.app_context():
 mongo.db.device.update(
 {'_id': int(_id)},
 { '$set': { 'state' : state}}
)

def edit_device(app, mongo, device_new):
 print(device_new["_id"])
 device = get_device_from_db(app, mongo, device_new["_id"], False)
 if not device:
 return False
 with app.app_context():
 mongo.db.device.update(
 {'_id': int(device_new["_id"])},
 { '$set':
 {
 'groups' : device_new["groups"],
 'name' : device_new["name"]
 }
 }
)
```

```
 }
)
 return True

def edit_group(app, mongo, group):
 group_exists = get_group_from_db(app, mongo, group["_id"])
 if group_exists:
 with app.app_context():
 mongo.db.device_group.update(
 {'_id': { '$in': [group['_id'], ObjectId(group['_id'])] } },
 { '$set':
 {
 'name': group["name"]
 }
 }
)
 mongo.db.device.update(
 { },
 { '$pull':
 {
 'groups':
 {
 '$in': [group['_id']]
 }
 }
 },
 multi=True
)
 for _id in group['devices']:
 mongo.db.device.update(
 { '_id': int(_id) },
 { '$push':
 {
 'groups': group['_id']
 }
 },
 multi=True
)
```

```

)
 return True
else:
 return False

```

#### II.2.4. device\_template.json

```
{
 "_id": null,
 "sub_channel": null,
 "type": null,
 "groups": [],
 "state": false,
 "name": ""
}
```

### II.3. Aplicación Web

#### II.3.1. index.js

```

import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import registerServiceWorker from './registerServiceWorker';
import 'bootstrap/dist/css/bootstrap.css';

ReactDOM.render(<App />, document.getElementById('root'));
registerServiceWorker();

```

#### II.3.2. App.js

```

import React, { Component } from 'react';
import { LightControlConsole } from './LightControlConsole';
import './App.css';

class App extends Component {

```

```
 render() {
 return (
 <LightControlConsole/>
);
 }
}

export default App;
```

### II.3.3. LightControlConsole.js

```
import React from 'react';
import { Jumbotron, Button, ButtonGroup, InputGroup, InputGroupAddon,
 Input } from 'reactstrap';
import axios from 'axios';

export class LightControlConsole extends React.Component {
 constructor(props) {
 super(props);
 this.state = {
 ledState: true,
 switchState: true,
 powerOnState: true,
 api_address: "",
 sonoff_topic: ""
 };
 }

 handleLEDClick = (value) => {
 this.setState({ledState: value});
 let message = value ? "1" : "0";
 let payload = {
 "topic": `${this.state.sonoff_topic}/LedPower`,
 "message": message
 };
 let url = `http://${this.state.api_address}/publish`;
 console.log(url);
 console.log(payload);
```

```
axios.post(
 url,
 payload
).then(function(response) {
 console.log(response)
});

};

handleSwitchClick = (value) => {
 this.setState({switchState: value});
 let message = value ? "1" : "0";
 let payload = {
 "topic": `${this.state.sonoff_topic}/Power`,
 "message": message
 };
 let url = `http://${this.state.api_address}/publish`;
 console.log(url);
 console.log(payload);
 axios.post(
 url,
 payload
).then(function(response) {
 console.log(response)
 });
};

handlePOSClick = (value) => {
 this.setState({powerOnState: value});
 let message = value ? "1" : "0";
 let payload = {
 "topic": `${this.state.sonoff_topic}/PowerOnState`,
 "message": message
 };
 let url = `http://${this.state.api_address}/publish`;
 console.log(url);
 console.log(payload);
 axios.post(
 url,
```

```
payload
).then(function(response) {
 console.log(response)
}) ;
};

handleRefreshClick = () => {

};

handleAPIChange = (event) => {
 this.setState({
 api_address: event.target.value
 })
};

handleTopicChange = (event) => {
 this.setState({
 sonoff_topic: event.target.value
 })
};

render() {
 let btn_led_color = this.state.ledState ? "success" : "danger";
 let btn_switch_color = this.state.switchState ? "success" : "danger";
 let btn_PO_color = this.state.powerOnState ? "success" : "danger";
 return (
 <div>
 <Jumbotron>
 <h1 className="display-3">Sonoff Control Panel</h1>
 <p className="lead">This is a simple console to control one Sonoff.</p>
 <hr className="my-2" />
 <p>It uses buttons to trigger requests to the flask server located in the raspberry.</p>
 <hr className="my-2" />
 <p className="lead">LED</p>
 <ButtonGroup className="lead">
```

```
<Button color={btn_led_color} onClick={() => this.
 handleLEDClick(true)}>On</Button>{' '}
<Button color={btn_led_color} onClick={() => this.
 handleLEDClick(false)}>Off</Button>
</ButtonGroup>
<Button color="primary" onClick={() => this.
 handleRefreshClick(true)}>Refresh State</Button>{' '}
<hr className="my-2" />
<p className="lead">Switch</p>
<ButtonGroup className="lead">
 <Button color={btn_switch_color} onClick={() => this.
 handleSwitchClick(true)}>On</Button>{' '}
 <Button color={btn_switch_color} onClick={() => this.
 handleSwitchClick(false)}>Off</Button>
</ButtonGroup>
<Button color="primary" onClick={() => this.
 handleRefreshClick(true)}>Refresh State</Button>{' '}
<hr className="my-2" />
<p className="lead">PowerOn State</p>
<ButtonGroup className="lead">
 <Button color={btn_PO_color} onClick={() => this.
 handlePOSClick(true)}>On</Button>{' '}
 <Button color={btn_PO_color} onClick={() => this.
 handlePOSClick(false)}>Off</Button>
</ButtonGroup>
<Button color="primary" onClick={() => this.
 handleRefreshClick(true)}>Refresh State</Button>{' '}
<p className="lead">Configuration</p>
<InputGroup>
 <InputGroupAddon>API address</InputGroupAddon>
 <Input placeholder="#.#.##.##.##" value={this.state.
 api_address} onChange={this.handleAPIChange}/>
</InputGroup>

<InputGroup>
 <InputGroupAddon>Sonoff topic</InputGroupAddon>
 <Input placeholder="/topic/subtopic" value={this.state.
 sonoff_topic} onChange={this.handleTopicChange}/>
```

```
</InputGroup>

</Jumbotron>
</div>
);
}
}
```