

# PROYECTO DE FINAL DE CARRERA

INGENIERÍA TELEMÁTICA

FACULTAD DE INGENIERÍA

UNIVERSIDAD DE MONTEVIDEO



UNIVERSIDAD DE MONTEVIDEO



Iván BABIC ([ibabic@correo.um.edu.uy](mailto:ibabic@correo.um.edu.uy))

Alexis ZECHARIES ([azecharies@correo.um.edu.uy](mailto:azecharies@correo.um.edu.uy))

Tutores:

Rafael Sotelo

Thomas Hobbins

Marzo de 2018

# Índice general

## 1 Estructura del Contenido

1.1	Arquitectura . . . . .
1.2	Módulos del Sistema . . . . .
1.2.1	Módulo MQTT Mesh . . . . .
1.2.2	Módulo de Control . . . . .
1.2.3	Módulo de interfaces de usuario . . . . .
1.3	Desarrollo . . . . .
1.3.1	Módulo MQTT Mesh . . . . .
1.3.2	Módulo Controlador . . . . .
1.3.3	Módulo de interfaces de usuario . . . . .

# Capítulo 1

## Estructura del Contenido

### 1.1. 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. [20] 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. [33]

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 [1.1], 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.

## 1. ESTRUCTURA DEL CONTENIDO

---

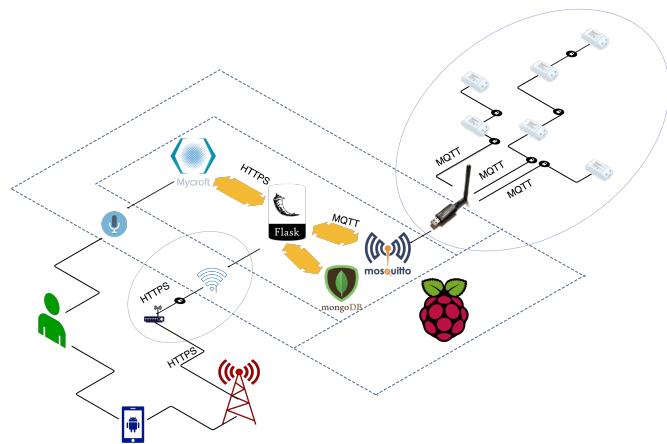


Figura 1.1: Redes en el sistema.

## 1.2. Módulos del Sistema

### 1.2.1. Módulo MQTT Mesh

#### Componentes:

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

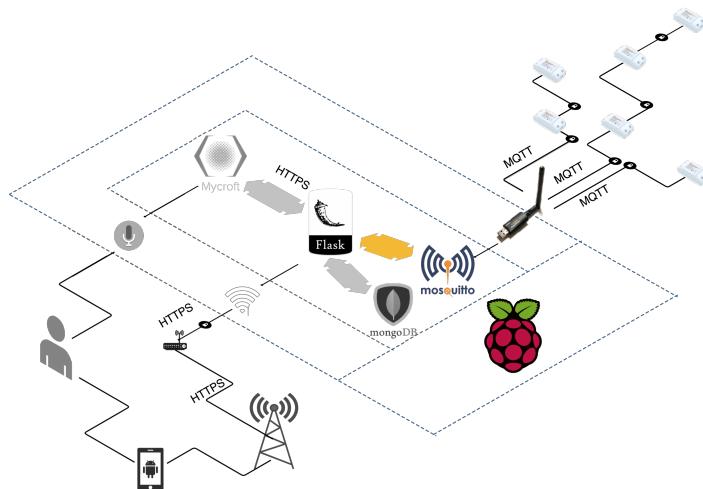


Figura 1.2: Componentes del Módulo MQTT Mesh.

#### 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

## 1. ESTRUCTURA DEL CONTENIDO

---

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.

### 1.2.2. Módulo de Control

#### Componentes:

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

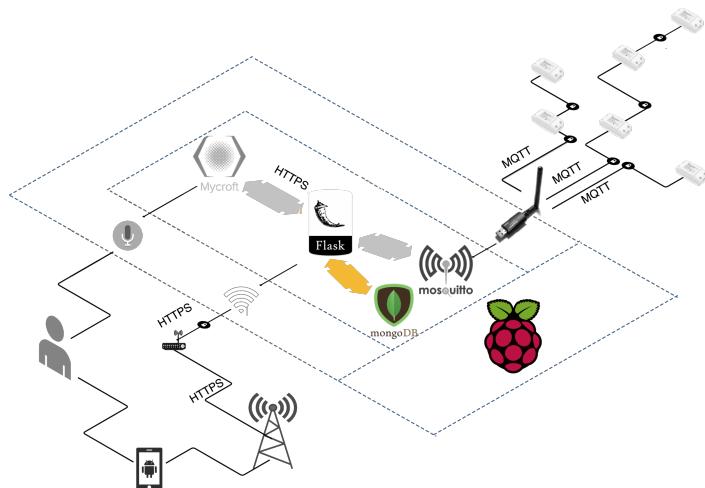


Figura 1.3: Componentes del Módulo de Control.

#### 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.

### 1.2.3. Módulo de interfaces de usuario

#### Aplicación web

#### Componentes:

- Aplicación contenida en el dispositivo móvil.

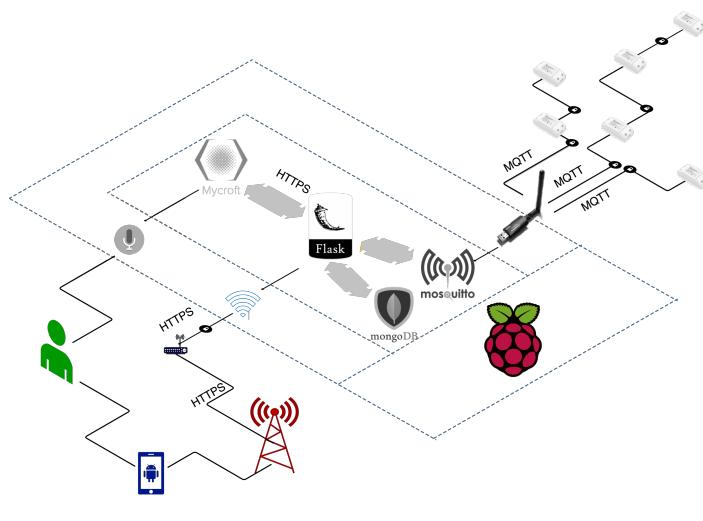


Figura 1.4: Componentes del Módulo de UI para Aplicación Móvil.

#### 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.

## 1. ESTRUCTURA DEL CONTENIDO

---

- 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.

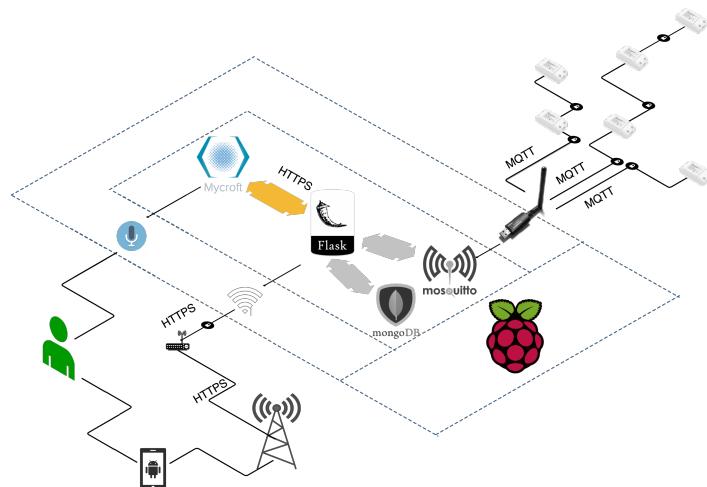


Figura 1.5: Componentes del Módulo de UI para Asistente Virtual.

#### Descripción:

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

## 1.3. Desarrollo

### 1.3.1. 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
```

## 1. ESTRUCTURA DEL CONTENIDO

---

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. [29]

- 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.

## 1. ESTRUCTURA DEL CONTENIDO

---

- 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.

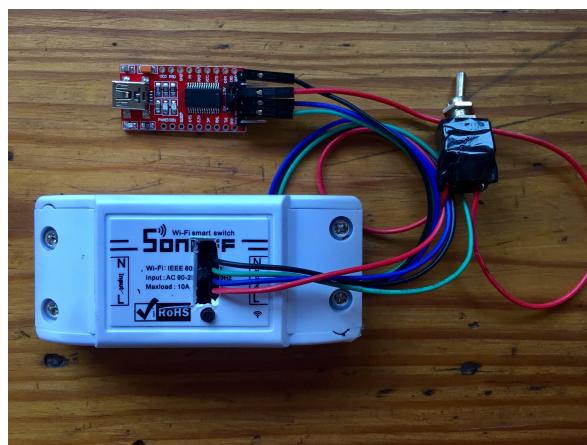
En la figura [1.8] podemos apreciar que los conectores del chip ESP8266 integrado en el Sonoff son fácilmente accesibles pero no cuentan con cabezales para conectar los cables del adaptador Serial-USB, por esta razón es necesario soldar los mismos. Luego se debe conectar al adaptador USB/Serial para poder subir el firmware desde un computador.

## 1. ESTRUCTURA DEL CONTENIDO

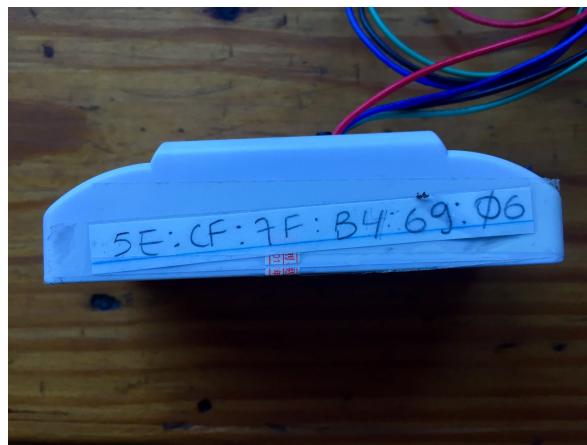
---



Figura 1.6: Conexiones del ESP8266 expuestas por el Sonoff.



(a)) Vista superior..



(b)) Vista de lado.

Figura 1.7: 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.

cos. 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 1.8: 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);
```

## 1. ESTRUCTURA DEL CONTENIDO

---

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 [??]

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;
        }
    }
}
```

## 1. ESTRUCTURA DEL CONTENIDO

---

```
    }
    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.[[\[??\]](#)]

```
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 es 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-picrft received PUBLISH (d0, q0, r1, m0, '/device_in
/bssid/2E:3A:E8:11:60:A3', ... (1 bytes))
4
Client mosqsub/2372-picrft received PUBLISH (d0, q0, r1, m0, '/device_in
/bssid/5E:CF:7F:B4:69:06', ... (1 bytes))
5
Client mosqsub/2372-picrft 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

## 1. ESTRUCTURA DEL CONTENIDO

---

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 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. [1.1] Si esto sucede, se verifica que el payload sea válido. Si es así, se realiza la acción que se especifica en la descripción.

<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.
Powerxjx;		Publica el estado actual del reléxjx;.
Power	0 / off	Apaga el reléxjx;
Power	1 / on	Prende el reléxjx;
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 1.1: Tabla de topics de comunicación con dispositivos Sonoff

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

## 1. ESTRUCTURA DEL CONTENIDO

---

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 lo 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 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:

## 1. ESTRUCTURA DEL CONTENIDO

---

- 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.

## 1. ESTRUCTURA DEL CONTENIDO

---

```
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\_crons:

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\_crons o delete\_group\_crons.

```
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"):
```

## 1. ESTRUCTURA DEL CONTENIDO

---

```
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. [??]

**main.cpp** [??] 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 [1.3.1].
- Registro de dispositivo en la red mesh, tanto en el broker como en la base de datos. [1.3.2]
- Restauración de estado anterior del relé.
- Interpretación de mensajes, ya sean órdenes o consultas de estado.

### 1.3.2. 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 [8] y flask\_pymongo [9]. flask\_mqtt [8]: es un wrapper de la librería paho-mqtt [21] 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 [9]: 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 [23]. 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 [??], aquí se revisarán las funcionalidades más importantes del mismo.

## 1. ESTRUCTURA DEL CONTENIDO

---

### 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 [??].

### 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` [??], 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.

### 1.3.3. Módulo de interfaces de usuario

#### Aplicación web

**Aplicación provisoria:** Para crear la aplicación provisoria se eligió utilizar la librería React [??] junto con `create-react-app` [5], 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` [28]

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:

## 1. ESTRUCTURA DEL CONTENIDO

---

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.

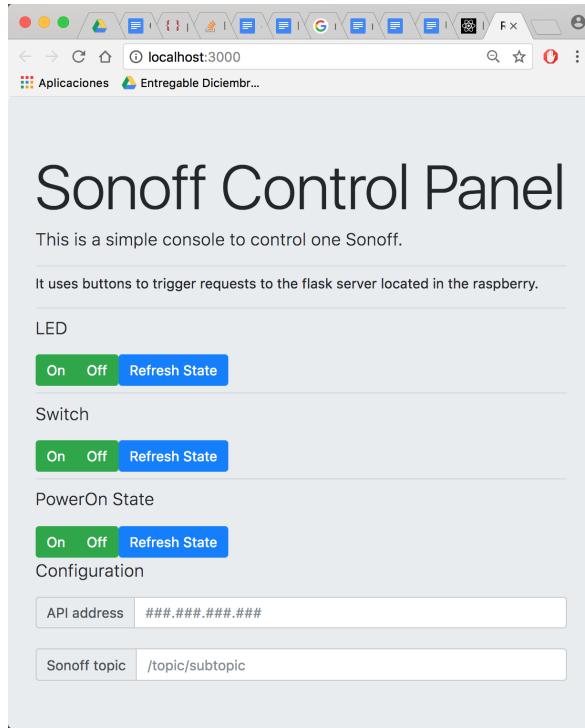


Figura 1.9: Vista de página principal provisoria.

En la figura [1.9] 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 en la aplicación o mediante otra interfaz de usuario).

El código de la misma se puede observar bajo el anexo [??]

**Aplicación Web y Móvil:** Para el desarrollo final se utilizaron las herramientas y librerías Auth0 [**Auth0**], Phonegap [**Phonegap**] y Android Studio [**AndroidStudio**].

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

## 1. ESTRUCTURA DEL CONTENIDO

---

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.

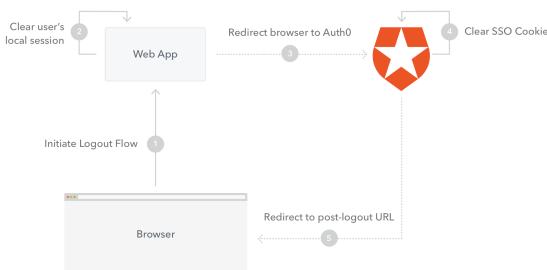


Figura 1.10: Flujo de autenticación con Auth0.

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

## 1. ESTRUCTURA DEL CONTENIDO

---

MegaMobile de Enabled [Enabled], debido a su interfaz vistosa y el estilo de las barras laterales y botones, los cuales se adecuaban a las pantallas diseñadas en Balsamic.

### 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.

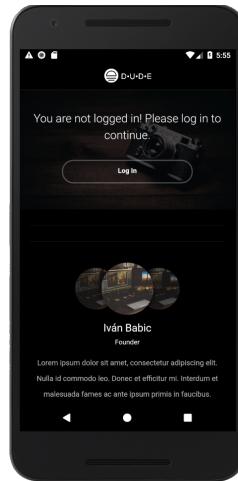


Figura 1.11: Página de inicio.

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

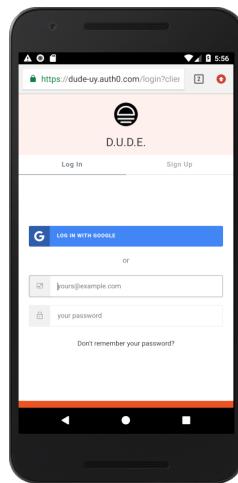


Figura 1.12: D.U.D.E. Auth0 Log In

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.

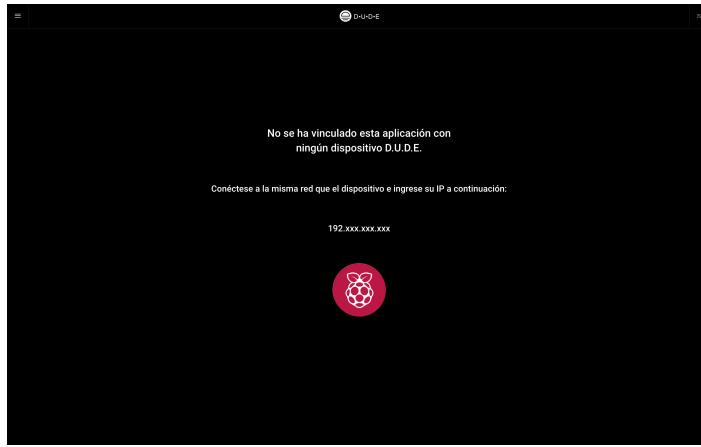


Figura 1.13: Página de configuración de dispositivo central.

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.

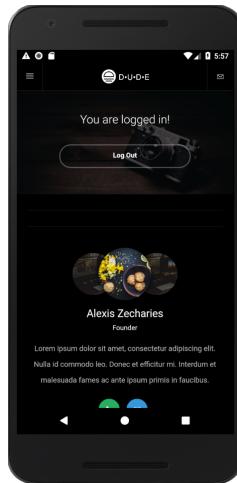


Figura 1.14: Página de inicio con sesión iniciada.

## 1. ESTRUCTURA DEL CONTENIDO

---

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 1.15: *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.

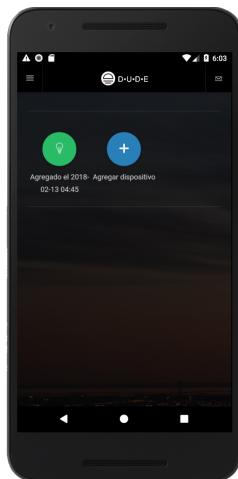


Figura 1.16: *Cambiar estado de dispositivo.*

### 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.

## 1. ESTRUCTURA DEL CONTENIDO

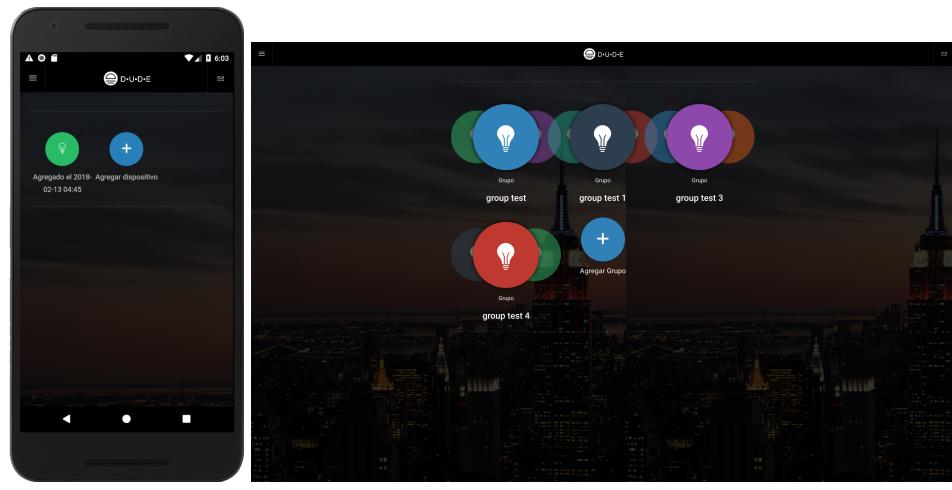
---



Figura 1.17: *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 1.18: *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 1.19: 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.

## 1. ESTRUCTURA DEL CONTENIDO

---



Figura 1.20: 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 1.21: Agregar grupo.

### Editar entidad

La 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 [[1.18 ]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 al 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.

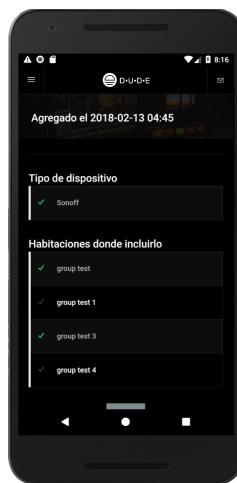


Figura 1.22: *Editar dispositivo*.

- **Grupo:** lleva a una interfaz con todos los grupos registrados además de la opción de agregar un nuevo grupo [[1.18 ]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.

## 1. ESTRUCTURA DEL CONTENIDO

---



Figura 1.23: *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.

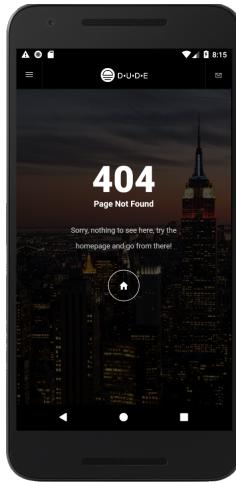


Figura 1.24: *Pantalla de error.*

### Mycroft

Se desarrollaron skills para el asistente virtual Mycroft [19] 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 voz 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 ([Topic1](#) [[Topic1](#)], [Topic2](#) [[Topic2](#)]). 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

## 1. ESTRUCTURA DEL CONTENIDO

---

Para el desarrollo y entendimiento de la implementación de skills se siguió la guía introductoria brindada por el equipo de Mycroft [[GuiaMycroft](#)]. 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.



# Bibliografía

- [1] *Atom Website*. <https://atom.io/>. Accedido: 2017-08-20.
- [2] *Atom Website*. <http://docs.platformio.org/en/latest/platforms/espressif8266.html>. Accedido: 2017-08-20.
- [3] *Banana Pi M3*. <http://www.banana-pi.org/m3.html>. Accedido: 2017-12-01.
- [4] *BeagleBone Black*. <https://beagleboard.org/black>. Accedido: 2017-12-01.
- [5] *create-react-app Repository*. <https://github.com/facebookincubator/create-react-app>. Accedido: 2017-12-01.
- [6] *ESP8266 Community Forum*. <http://www.esp8266.com/>. Accedido: 2017-12-01.
- [7] *Flask Docs*. <http://flask.pocoo.org/>. Accedido: 2017-12-01.
- [8] *flask-mqtt Docs*. <http://flask-mqtt.readthedocs.io/en/latest/>. Accedido: 2017-12-01.
- [9] *flask-pymongo Docs*. <https://flask-pymongo.readthedocs.io/en/latest/>. Accedido: 2017-12-01.
- [10] *Google Home Docs*. <https://support.google.com/googlehome/?hl=en#topic=7029677>. Accedido: 2017-08-20.
- [11] *Home Assistant Docs*. <https://home-assistant.io/>. Accedido: 2017-08-20.
- [12] *HTTP CoAP Layers*. <https://www.cse.wustl.edu/~jain/cse574-14/ftp/coap/index.html>. Accedido: 2017-12-01.
- [13] *IKEA Lighting Website*. <http://www.ikea.com/us/en/catalog/categories/departments/lighting/36812/>. Accedido: 2017-12-01.
- [14] *JavaScript Website*. <https://www.javascript.com/>. Accedido: 2017-12-01.
- [15] *MongoDB Docs*. <https://docs.mongodb.com/manual/mongo/>. Accedido: 2017-12-01.
- [16] *MQTT Diagram*. <https://2016.angularattack.com/entries/506-armyofone>. Accedido: 2017-12-01.
- [17] *MQTT Essentials part 5*. <https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices>. Accedido: 2017-12-01.

- [18] *Mycroft Community Skill Repository*. <https://github.com/MycroftAI/mycroft-skills/blob/master/README.md#community-contributed-skill-list>. Accedido: 2017-12-01.
- [19] *Mycroft Website*. <https://mycroft.ai/>. Accedido: 2017-08-20.
- [20] Lavinia Năstase. «Security in the Internet of Things: A Survey on Application Layer Protocols». En: (2017).
- [21] *paho-mqtt Docs*. <https://pypi.python.org/pypi/paho-mqtt/1.1>. Accedido: 2017-12-01.
- [22] *Phillips Lighting Website*. <http://www.lighting.philips.com/main/systems/system-areas>. Accedido: 2017-12-01.
- [23] *PyMongo Docs*. <http://api.mongodb.com/python/current/>. Accedido: 2017-12-01.
- [24] *Raspberry Pi 2*. <https://www.raspberrypi.org/products/raspberry-pi-2-model-b/>. Accedido: 2017-12-01.
- [25] *Raspberry Pi 3*. <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>. Accedido: 2017-12-01.
- [26] *React Native Website*. <https://facebook.github.io/react-native/>. Accedido: 2017-12-01.
- [27] *ReactJS Website*. <https://reactjs.org/>. Accedido: 2017-12-01.
- [28] *reactstrap Website*. <https://reactstrap.github.io/>. Accedido: 2017-12-01.
- [29] *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.
- [30] *USB to Serial Adapter*. [http://product.hstatic.net/1000157790/product/nnes00015\\_converter\\_usb\\_to\\_serial\\_ttl\\_master.jpg](http://product.hstatic.net/1000157790/product/nnes00015_converter_usb_to_serial_ttl_master.jpg). Accedido: 2017-12-01.
- [31] *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](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.
- [32] *What is PlatformIO*. <http://docs.platformio.org/en/latest/what-is-platformio.html>. Accedido: 2017-08-20.

## BIBLIOGRAFÍA

---

- [33] *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.