Proyecto IoT para Informática Industrial.

Grupo 8

Franciso Luque del Castillo

Sergio Mesa Martín

Pablo Moreno Moreno

Adrián Sánchez Quintana

Ingeniería Electrónica, Robótica y Mecatrónica.

26 de enero de 2021.

UNIVERSIDAD DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

ÍNDICE

1. Introducción y objetivos.

El presente Proyecto consiste en diseñar e implementar un sistema de monitorización y control multi-ubicación compuesto por una red de cuatro placas de desarrollo basadas en el módulo WiFi ESP8266, cada una perteneciente a un integrante del grupo, programándolas con el IDE de Arduino.

Cada dispositivo será capaz de leer la información proporcionada por un sensor de temperatura y humedad DHT11 y, además, uno de ellos dispondrá también de un sensor analógico para leer el nivel del recipiente que almacena el agua que proviene del aire acondicionado. También se leerá el estado de dos actuadores: uno que se comportará como un interruptor (GPIO16) y otro que controlará una luminaria mediante PWM (GPIO2), así como algunos parámetros de la red WiFi a la que se encuentra conectado del dispositivo y su estado de conexión.

La información recogida por los dispositivos se enviará de forma inalámbrica por MQTT a una aplicación en Node-RED, donde se procesará y se almacenará en una base de datos.

Además, desde esta aplicación será posible enviar órdenes para controlar los dos actuadores y para modificar aspectos de configuración, así como una orden para que los dispositivos comprueben si existe una nueva actualización disponible para ellos.

La información recogida se mostrará mediante el paquete dashboard de Node-RED de forma individual para cada dispositivo, desde el que se permitirán también realizar acciones como la consulta de determinados datos de acuerdo a diferentes criterios y su representación gráfica, la descarga de los mismos en formato CSV y el envío de las órdenes mencionadas en el párrafo anterior. Todo ello será posible realizarlo tanto para un dispositivo determinado como para los cuatro simultáneamente.

Se dispondrá además de registros para almacenar los cambios de estado de los actuadores, el estado de conexión de los dispositivos y los errores que se produzcan al leer la información de los sensores.

La información también será accesible desde Telegram, desde donde se podrá entablar una conversación con un bot y enviar órdenes a los dispositivos mediante comandos. Esta aplicación también se usará como canal de notificación para los usuarios, en caso de que la temperatura o la humedad registradas se encuentren fuera de un rango determinado.

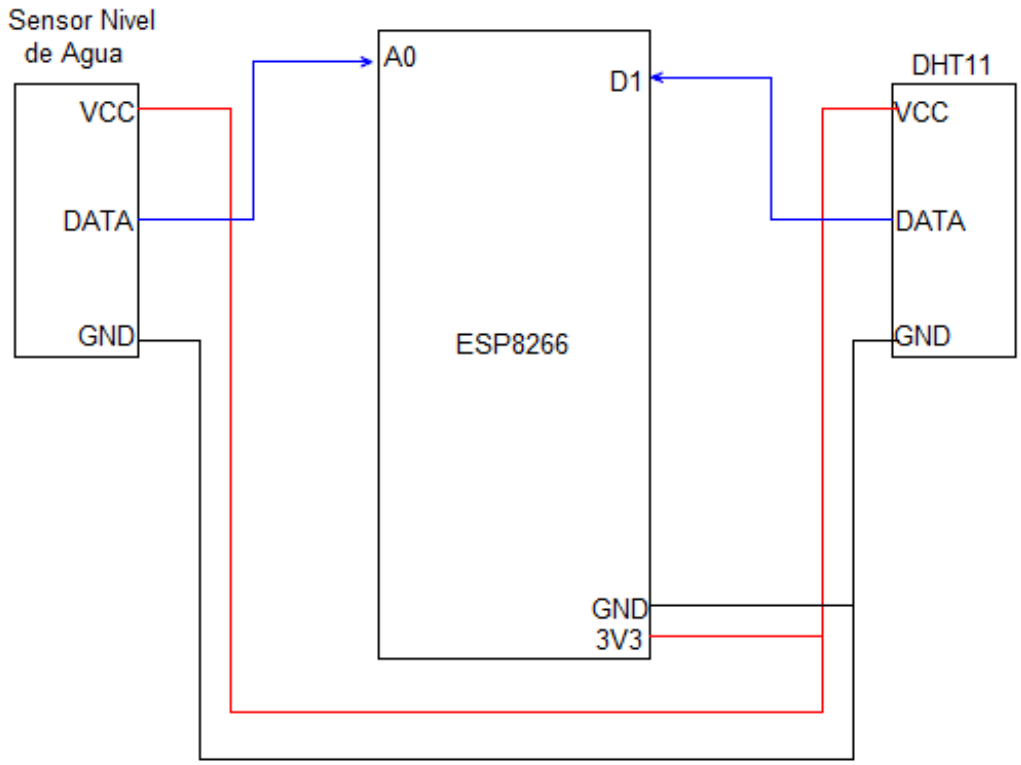
Por otro lado, también será posible actuar sobre el dispositivo pulsando el botón flash conectado al GPIO0 para realizar acciones como modificar el estado de los actuadores o enviar una orden de comprobación de actualizaciones.

2. Diseño hardware y esquema de conexionado.

Como hemos mencionado, cada dispositivo ESP8266 tiene conectado un sensor DHT11 y a uno a uno de ellos, adicionalmente, se le ha conectado un sensor analógico de nivel de agua.

Ambos sensores se alimentan mediante la salida de 3.3V de la placa de desarrollo. La señal de datos del sensor DHT11 se ha conectado al GPIO5 (D1) y la del sensor de nivel de agua al pin ADC0 (A0).

El esquema de conexionado es el siguiente:



3. Programa Arduino.

A continuación, se describe de forma general cómo ha sido implementado el programa "Grupo8_Proyecto_IoT.ino".

Las librerías utilizadas son:

- *ESP8266WiFi,* necesaria para la conexión WiFi del dispositivo.
- *PubSubClient,* para poder enviar y recibir mensajes por MQTT.
- *DHTesp*, necesaria para leer las medidas tomadas por el sensor DHT11.
- *ArduinoJson,* para la serialización y deserialización de mensajes en JSON.
- *ESP8266httpUpdate*, necesaria para realizar las actualizaciones de firmware de forma inalámbrica.
- *Button2,* para facilitar el tratamiento de la lectura del botón de la placa.

En cuanto a las funciones que se han empleado, por orden de aparición en el código, son las siguientes:

- ***setup_wifi***

Mediante esta función se realiza la conexión WiFi del dispositivo.

El programa se ha generalizado para que pueda ser usado por los cuatro integrantes del grupo sin tener que hacer ninguna modificación en el código. Para ello, se han introducido las ssid y contraseñas WiFi de cada uno de los integrantes de forma que se seleccionen automáticamente los valores correspondientes para cada usuario en función del chipID del dispositivo en el que se ejecute el programa.

- ***callback***

Esta función se ejecutará cada vez que recibamos un mensaje por MQTT por cualquiera de los topics a los que estamos suscritos.

A continuación, se enumeran los valores que se reciben por cada uno de los topics y el tratamiento que se hace a cada uno de ellos. El significado de cada uno de estos valores está descrito en el Manual de Usuario.

Como aclaración, el valor "X" que aparece en los topics siguientes, se sustituye automáticamente por el chipID del dispositivo en el que se ejecuta el programa. Esto se hace en la función *reconnect*.

- Topic infind/GRUPO8/ESPX/config
    - Frecuencia de envío de datos.

Este valor se envía desde Node-RED en segundos. Para el correcto funcionamiento del programa, se ha establecido que este valor tiene que ser mayor o igual a 2 segundos. Si es así, se almacena en la varible "Temp". En caso contrario, se muestra un error por consola indicando que el dato introducido no está dentro del rango válido.

    - Frecuencia de cambio del LED PWM.

Este valor se almacenará en la variable “frecled” en caso de que sea mayor o igual que 0. En caso contrario, mostraremos un error por la consola, ya que una frecuencia negativa puede provocar un comportamiento no deseado del programa.

    - Frecuencia de actualización.

Igualmente, comprobamos que sea mayor o igual que 0 y la almacenamos en la variable “frecActualizacion”.

    - Configuración de la lógica del Switch.

El valor que se recibe desde MQTT es booleano, por lo almacenamos en una variable también booleana: “LogicaDigital”. Cada vez que se recibe un valor, se llama a la función *ControlLogica* para actualizar la configuración de la lógica inmediatamente.

    - Configuración de la lógica del LED.

Al igual que la anterior, se almacena en una variable booleana: “LogicaPWM” y se llama a la función *ControlLogica.*

- Topic infind/GRUPO8/ESPX/FOTA
    - Orden de actualización.

El valor recibido se almacena en la variable booleana “actualizaMQTT”.

- Topic infind/GRUPO8/ESPX/led/cmd
    - Intensidad del LED.

Tras comprobar que está dentro del rango [0, 100], almacenamos este valor en la variable “level”. Además, indicamos que el origen de esta orden es MQTT actualizando la variable “origenPWM”. Por último, llamamos a la función *LED* para actualizar el valor PWM que se envía al dispositivo.

- Topic infind/GRUPO8/ESPX/switch/cmd
    - Control del LED Digital (Switch).

Almacenamos el valor booleano recibido en la variable “LedDigital”. Además, indicamos que el origen de la orden de cambiar el valor del switch ha sido MQTT modificando el valor de la variable “origenDigital” y llamamos a la función *switchControl* para modificar el valor que enviamos al GPIO16.

- ***reconnect***

Mediante esta función nos conectamos al broker MQTT. Esto se hace con un identificador de cliente único en el que aparece el chipID del dispositivo.

Al conectarnos, se envía el mensaje LWT compuesto por el campo “online”: false. Este mensaje se enviará automáticamente si se produce una desconexión inesperada del dispositivo. Además, al conectarnos también enviamos el mensaje retenido compuesto por el campo “online”: true, para indicar que el dispositivo se ha conectado. Ambos mensajes se envían por el topic infind/GRUPO8/ESPX/conexion.

Por último, nos suscribimos a los topics enumerados anteriormente. Con el objetivo, nuevamente, de generalizar el programa para que sea válido para los cuatro integrantes del grupo, los topics a los que nos suscribimos están en función del chipID del dispositivo. De esta forma, automáticamente, cada placa solo recibirá los mensajes MQTT que vayan dirigidos a ella.

- ***actualizar***

Esta función se encargará de comprobar si hay un programa nuevo para nuestro dispositivo. En caso de que lo haya, lo descarga y lo sustituye por el actual. Una vez reemplazado, se reinicia la placa y comienza a ejecutarse el nuevo programa.

Para ello, tenemos que indicar la dirección IP, puerto y path donde se encuentra el servidor de actualizaciones, el cual está implementado en Node-RED en el flujo “FOTA”.

- ***setup***

En esta función se realizan diversas operaciones de inicialización:

- Configuramos el GPIO16 (switch) como salida.
- Llamamos a la función *setup_wifi* para establecer la conexión WiFi.
- Indicamos que hemos conectado el sensor DHT11 al pin 5.
- Establecemos el servidor MQTT.
- Fijamos el tamaño del buffer para el manejo de mensajes MQTT a 512 para que no haya problemas con los mensajes largos.
- Llamamos a la función *actualizar* para comprobar si hay alguna actualización nueva para el dispositivo al arrancar la placa.
- Configuramos el LED PWM y el switch digital llamando a las funciones *LED* y *switchControl*, respectivamente.
- Ejecutamos los comandos de la librería *button2* necesarios para el tratamiento del pulsador flash.
- Configuramos el nombre de los topics por los que publicaremos en MQTT en función del chipID del dispositivo, al igual que hicimos con los topics a los que

nos suscribimos. Los topics por los que publicaremos son: infind/GRUPO8/ESPX/conexion, infind/GRUPO8/ESPX/led/status, infind/GRUPO8/ESPX/switch/status y infind/GRUPO8/ESPX/datos.

- ***ControlLogica***

Esta función configura la interpretación de las señales que enviamos tanto al LED PWM como al switch digital, es decir, configura si la lógica de estos actuadores es positiva o negativa.

- ***switchControl***

En esta función modificamos el valor digital que enviamos al GPIO16 en función del valor de la variable "SalidaSwitch" y publicamos cuál es su estado actual por el topic infind/GRUPO8/ESPX/switch/status. En el mensaje a publicar incluimos también un campo con el chipID del dispositivo y el origen de la orden de cambio (MQTT o botón).

- ***LED***

Esta función realiza lo mismo que la anterior, pero con el LED PWM, es decir, actualiza el valor que se envía por el GPIO2 en función de la variable "level" y publica el nivel actual del LED (0-100) por el topic infind/GRUPO8/ESPX/led/status. Al igual que antes, al mensaje publicado le añadimos un campo con el chipID del dispositivo y otro con el origen de la orden de actualización del nivel del LED (MQTT o botón).

- **Funciones *button2*.**

Estas funciones de la librería *button2* nos facilitan la lectura del botón Flash de la placa. Haciendo uso de ellas, se han implementado las siguientes funcionalidades:

- Al realizar una pulsación prolongada del botón (superior a 4 segundos), se llama a la función *actualizar* para comprobar si hay alguna actualización nueva del firmware para nuestro dispositivo.
- Al hacer una pulsación simple, el LED PWM se apaga o se enciende al nivel establecido previamente y el estado del switch cambia de valor. Además, actualizamos el valor de las variables "origenPWM" y "origenDigital" para indicar que las órdenes han sido efectuadas por el botón.
- Al hacer una pulsación doble, se enciende el LED al nivel máximo y se indica que la orden ha sido efectuada por el pulsador modificando el valor de la variable "origenPWM".

- ***Loop***

Cada vez que entramos en esta función, comprobamos si está activa la conexión MQTT e intentamos reconectarnos en caso de que no lo esté.

Por otro lado, publicamos los datos con una frecuencia indicada por la variable "Temp", así como inmediatamente al encender la placa (PrimeraVez=true). Para ello, realizamos las siguientes operaciones:

- Leemos los valores de temperatura y humedad del sensor DHT11.
- Obtenemos el RSSI de la conexión Wi-Fi y la IP del router al que estamos conectados.
- Leemos el nivel de agua que nos proporciona el sensor analógico y hacemos un escalado para mostrar este nivel en porcentaje.
- Formateamos los valores correspondientes al uptime, estado del switch, nivel del LED, nivel de agua, temperatura y humedad (DHT11), ssid, ip y RSSI (WiFi) así como el chipID del dispositivo, en la estructura JSON "datos" y los publicamos por el topic infind/GRUPO8/ESPX/datos. Debido a que hemos incluido un sensor analógico de nivel de agua, no es posible leer la tensión de alimentación del dispositivo (Vcc).

Además, si corresponde, actualizamos el nivel del LED para que cambie gradualmente según la frecuencia almacenada en "frecled".

Por último, en caso de que se haya efectuado una orden de comprobación de actualización del firmware, ya sea proveniente de MQTT, por haber hecho una pulsación prolongada del botón Flash o porque haya transcurrido el tiempo establecido en "frecActualizacion", llamamos a la función *actualizar* para realizar dicha comprobación.

- **Funciones OTA.**

Son un conjunto de funciones de la librería *ESP8266httpUpdate* necesarias para la actualización del firmware de forma inalámbrica.

4. Flujos Node-RED.

El diseño de la aplicación en Node-RED se ha distribuído en seis flujos: *Recepción de datos, Configuración y acciones, Consultas, Descargas, Telegram y Control IFTT.*

4.1. Recepción de datos.

En este flujo recibimos la información publicada por los cuatro dispositivos en los topics infind/GRUPO8/ESPX/conexion, infind/GRUPO8/ESPX/led/status e infind/GRUPO8/ESPX/switch/status y infind/GRUPO8/ESPX/datos, donde el valor de X corresponde al chipID de cada dispositivo.

En primer lugar, convertimos los mensajes recibidos de JSON a objetos JavaScript. Posteriormente se realizan dos operaciones:

A) Almacenamiento de los mensajes en la Base de Datos.

Antes de almacenar los registros, añadimos al payload de todos ellos el campo “date”, que contiene la fecha y hora actual, para poder recuperarlos luego según el momento en el que se registraron.

Todos los mensajes tienen el campo “CHIPID”, por lo que podemos almacenar registros que provengan de dispositivos distintos en una misma colección, pudiendo recuperarlos luego utilizando este criterio

Hemos organizado los datos en cinco colecciones:

- *conexion*, en la que almacenaremos el estado de conexión de los dispositivos, es decir, se trata de un registro de cada vez que se ha conectado o desconectado un dispositivo.
- *led,* en la que se almacenan los cambios en el nivel del LED, así como el origen de los mismos (MQTT o botón).
- *switch,* en la que se almacenan los cambios de estado de los switches junto con la entidad que los efectuó (MQTT o botón).
- *datos*, en la que se guardan los datos de los sensores, el estado de los actuadores, los parámetros de la conexión WiFi y el uptime cada vez que se envían desde el dispositivo (por defecto, cada 15 minutos).
- *errores*, en la que se registran los errores en la lectura de los sensores.

La identificación de si ha habido un error en la lectura de alguno de los parámetros (temperatura, humedad y nivel de agua), se realiza mediante la función “error?”.

Desconectando los sensores, se ha comprobado que si hay un error en la lectura de la temperatura o de la humedad, se guarda “null” en dichos campos, mientras que si se produce un error en la lectura del nivel de agua, se guarda un “0” en este campo, por lo que se ha usado este criterio para identificar los errores.

Si hay un error en cualquiera de los parámetros, se almacenará en la colección *errores* un registro con tres campos booleanos: “errorTemp”, “errorHum” y “errorNivelAgua”, de los cuales, valdrán true los correspondientes a los errores producidos y el resto false. Además, como todos los registros almacenados en la base de datos, también incluyen el campo “date” y “CHIPID”.

En caso de que no haya ningún error en los datos, no se almacena nada en esta colección, solo se guardan en la colección *datos.*

B) Representación en el Dashboard de los últimos datos recibidos.

Los datos se muestran en la pestaña “Datos actuales” del Dashboard y son: el estado de la conexión, el uptime, los datos de los sensores, el estado de los actuadores y los parámetros del WiFi a los que está conectado del dispositivo, así como la fecha y hora a la que corresponden esos datos.

Se ha incluido un selector de forma que podamos elegir fácilmente de qué dispositivo queremos visualizar los datos. Tras realizar la selección, la variable global “selectorID” toma el valor del chipID del dispositivo elegido.

La salida del selector se he conectado a las funciones previas a los nodos de salida del dashboard. De esta forma, al cambiar la selección de usuario, se ejecutan dichas funciones, enviando al dashboard los datos del dispositivo seleccionado. Los nodos MQTT de entrada de datos también están conectados a estas funciones, de forma que cada vez que se recibe un nuevo dato por cualquiera de los topics, si el dato proviene del dispositivo que esté seleccionado, se actualiza inmediatamente en el dashboard así como la fecha y hora a la que corresponde.

4.2. Configuración y acciones.

Este flujo permite la selección de distintos parámetros de configuración de los dispositivos, el envío de órdenes a los actuadores (LED y switch) y el envío de una orden para comprobar si existe alguna actualización del firmware de los dispositivos. Todas estas acciones se introducen en la pestaña “Configuración y acciones” del Dashboard.

Tanto la selección de los parámetros de configuración como el envío de acciones se puede realizar para un dispositivo determinado o bien para todos simultáneamente, para lo que se ha incluido un selector de usuario.

Los valores introducidos en el Dashboard se almacenan en variables globales y se envían por MQTT a los dispositivos formateados en JSON.

En la variable global “selector” se almacena un 0 si seleccionamos todos los dispositivos y un valor entre 1 y 4 si solo seleccionamos uno de los dispositivos. El valor

almacenado en esta variable determinará, en las funciones previas a los nodos MQTT de salida, a qué dispositivos enviamos los mensajes.

Además, en esta pestaña del Dashboard podemos seleccionar los umbrales de temperatura y humedad máximos y mínimos que determinan la activación de la alarma.

4.3. Consultas.

Se pueden realizar consultas a los datos almacenados en las cinco colecciones mencionadas anteriormente en función de dos criterios: la fecha en la que fueron almacenados y el dispositivo del que provienen. Para este último criterio, hemos incluido en el dashboard un selector de usuario que permite filtrar la posterior consulta en función del ID de la placa del usuario seleccionado.

Los datos almacenados que estarán disponibles para una consulta son los siguientes:

- **Datos recibidos por los sensores**. Entre estos tenemos la temperatura y la humedad, provenientes del sensor DHT11, y el nivel del agua medido con el sensor analógico. Están almacenados en la colección *datos.*
- **Historial de cambio de los actuadores**. Los actuadores disponibles son el LED PWM y el switch. Los cambios en ellos se han almacenado en las colecciones *led* y *switch*, respectivamente.
- **Historial del estado de la conexión de los dispositivos**, correspondiente a la colección *conexion.*
- **Registro de los errores procedentes de medidas de los sensores,** que corresponde a la colección *errores.*

Centrándonos más en los intervalos de tiempo a consultar, tenemos las siguientes opciones:

- **Consulta del último día**. Permite realizar una consulta de los datos de los sensores almacenados en el periodo de las últimas 24 horas respecto de la hora actual. La consulta se realiza por agregación, agrupando cada una de las medidas de los sensores según el intervalo mencionado. A estos datos agrupados se le calculan los siguientes estadísticos: valor máximo, valor mínimo, media y número de muestras. Dichos estadísticos son también representados gráficamente en función de la hora. A continuación, se muestra un ejemplo de la agregación por días para la temperatura (a la izquierda), y la agregación utilizada para obtener los datos para las gráficas (a la derecha). El resto de datos tiene la misma estructura, pero particularizada para ellos).

```
var ms_per_dia = 60000*60*24;

// Obtenemos el timestamp actual
var now_ms = new Date().getTime();

// Creamos un objeto con la hora de hace 24h
var ayer = new Date(now_ms-ms_per_dia);

var usuario = global.get("usuario");

if (usuario=="ESP0")
{
    msg.payload=
[
    { "$match": { "date": { "$gte": ayer } } },
    { "$project": {
        "temperatura": "$DHT11.Temperatura"
        }
    },
    { "$group": {
        "_id": 0,
        "Tmedia":{"$avg":"$temperatura"},
        "Tmax":  {"$max":"$temperatura"},
        "Tmin":  {"$min":"$temperatura"},
        "datos": {"$sum":1}
    }}
];
}
else
{
     msg.payload=
[
    { "$match": { "date": { "$gte": ayer } } },
    { "$match": { "CHIPID": usuario } },
    { "$project": {
        "temperatura": "$DHT11.Temperatura"
        }
    },
    { "$group": {
        "_id": 0,
        "Tmedia":{"$avg":"$temperatura"},
        "Tmax":  {"$max":"$temperatura"},
        "Tmin":  {"$min":"$temperatura"},
        "datos": {"$sum":1}
    }}
];
}

return msg;
```

```
var ms_per_dia = 60000*60*24;

// Obtenemos el timestamp actual
var now_ms = new Date().getTime();

// Creamos un objeto con la hora de hace 24h
var ayer = new Date(now_ms-ms_per_dia);

var usuario = global.get("usuario");

if (usuario=="ESP0")
{
msg.payload=
[
    { "$match": { "date": { "$gte": ayer } } },
    { "$project": {
        "date": 1,
        "temperatura": "$DHT11.Temperatura"
        }
    },
    { "$group": {
        "_id": {
        "$dateFromParts" : {
            "year": { "$year": "$date" },
            "month": { "$month": "$date" },
            "day": { "$dayOfMonth": "$date" },
            "hour": { "$hour": "$date" }

        }},

        "Tmedia":{"$avg":"$temperatura"},
        "Tmax":  {"$max":"$temperatura"},
        "Tmin":  {"$min":"$temperatura"},
        "datos": {"$sum":1},
    }},
    { "$sort": {"_id": 1} }
];
}
else
{
  msg.payload=
[
    { "$match": { "date": { "$gte": ayer } } },
    { "$match": { "CHIPID": usuario } },
    { "$project": {
        "date": 1,
        "temperatura": "$DHT11.Temperatura"
        }
    },
    { "$group": {
        "_id": {
        "$dateFromParts" : {
            "year": { "$year": "$date" },
            "month": { "$month": "$date" },
            "day": { "$dayOfMonth": "$date" },
            "hour": { "$hour": "$date" }

        }},

        "Tmedia":{"$avg":"$temperatura"},
        "Tmax":  {"$max":"$temperatura"},
        "Tmin":  {"$min":"$temperatura"},
        "datos": {"$sum":1},
    }},
    { "$sort": {"_id": 1} }
];
}
return msg;
```

- **Consulta de la última semana**. Esta consulta recupera los datos de los sensores almacenados en los últimos 7 días respecto del día actual de consulta. Sigue la misma estructura que la consulta para el último día, simplemente se amplía el periodo de tiempo (multiplicándolo por 7) y se elimina la línea 56 para la agrupación de las gráficas.

- **Consulta del último mes**. Esta opción permite consultar los datos para los últimos 30 días respecto de la fecha en la que se realiza la consulta, así como el cálculo de los estadísticos y gráficas disponibles para las otras opciones. Se multiplica por 30 el tiempo que se usaba para la consulta del último día y se agrupa de la misma forma que en la última semana (por días).

- **Consulta por rango de fechas**. Esta es la última opción de intervalo de tiempo para realizar una consulta. Permite escoger la fecha inicial y la fecha final mediante un selector, con el fin de agrupar los datos almacenados en ese rango. Se agregan de nuevo los datos de los sensores con el cálculo de los estadísticos y su representación gráfica. A esta opción se le suma también la posibilidad de consultar otras colecciones, como los actuadores, estado de conexión o errores registrados, con el objetivo de mostrar en el "dashboard" un listado con el registro de dichos datos.
  Para la agregación por rango de fechas se ha modificado la cabecera de las funciones de agregación con la siguiente estructura:

```
1  // fecha por defecto es hoy
2  var ahora = new Date();
3  // pillo los globales si los hay, si no la fecha de hoy
4  var desde = new Date(global.get("desde") || ahora);
5  var hasta = new Date(global.get("hasta") || ahora);
6  // el día de inicio desde las cero horas
7  var inicio = new Date(desde.getFullYear(),desde.getMonth(),desde.getDate(),1,0,0,0);
8  // el día final es hasta final del día
9  var final = new Date(hasta.getFullYear(),hasta.getMonth(),hasta.getDate(),24,59,59,99);
10
11 var usuario = global.get("usuario");
12
13 if (usuario=="ESP0")
14 {
15     msg.payload=
16 [
17     { "$match": { "date": { "$gte": inicio , "$lte": final } } },
```

Con respecto a las consultas por último día, última semana y último mes, no se han añadido consulta de actuadores, conexión ni errores puesto que no tienen valor estadístico, así que simplemente se podrán consultar en una lista por rango de fechas.

Las listas se han hecho en formato HTML con el siguiente con el siguiente código (ejemplo concreto para el estado de la conexión, pero el resto de colecciones sigue la misma estructura):

```
1  <h1>Registros conexion: </h1>
2  <ul>
3  {{#payload}}
4    <li>{{date}} : {{conexion}}, {{chipid}} </li>
5  {{/payload}}
6  </ul>
```

4.4. Descargas.

Usando un flujo de Node-RED, hemos implementado la posibilidad de descargar los datos consultados de una colección, pudiendo escoger los datos de un usuario concreto o de todos (al igual que las consultas), así como el rango de tiempo deseado mediante una fecha inicial y otra final.

Con respecto a la implementación de la descarga se ha añadido un botón (plantilla HTML) con el siguiente código:

```
1  <style>
2  .button {
3      text-align: center;
4      font: bold 17px Arial;
5      text-decoration: none;
6      background-color: #339966;
7      color: white;
8      padding: 8px 10px;
9      border: 2px solid #CCCCCC;
10   }
11
12   .button:hover
13   {
14    background-color: #26734d;
15   }
16
17 </style>
18 <a href="/registros_datos" class="button">Descarga registros de datos en CSV (EXCEL)</a>
19
```

Para adaptarlo al resto de colecciones solo debemos modificar la línea 18 para escoger el nombre del “trigger” que desencadenará la descarga (en caso del ejemplo se trata de “/registro_datos”) y el mensaje que aparecerá sobre el botón (en este caso “Descarga registros de datos en CSV”). El trigger mencionado será un nodo HTTP con la función GET y el nombre establecido.

Se realizará una agregación de los datos consultados en función del rango de fechas escogido y se seleccionarán qué campos queremos obtener con la descarga (con $project), ordenándolos por fecha. El siguiente ejemplo muestra el caso de la agregación para descargar los datos de los sensores:

```
// fecha por defecto es hoy
var ahora = new Date();
// pillo los globales si los hay, si no la fecha de hoy
var desde = new Date(global.get("desde") || ahora);
var hasta = new Date(global.get("hasta") || ahora);
// el día de inicio desde las cero horas
var inicio = new Date(desde.getFullYear(),desde.getMonth(),desde.getDate(),1,0,0,0);
// el día final es hasta final del día
var final = new Date(hasta.getFullYear(),hasta.getMonth(),hasta.getDate(),24,59,59,99);

var usuario = global.get("usuario");

if (usuario=="ESP0")
{
    msg.payload=
[
    { "$match": { "date": { "$gte": inicio , "$lte": final } } },
    { "$project": {
        "date": 1,
        "temperatura": "$DHT11.Temperatura",
        "humedad": "$DHT11.Humedad",
        "Nivel_agua": "$NivelAgua",
        "CHIPID" : "$CHIPID"
        }
    },
    { "$sort": {"date": 1} }
];
}
else
{
    msg.payload=
[
    { "$match": { "date": { "$gte": inicio , "$lte": final } } },
    { "$match": { "CHIPID": usuario } },
    { "$project": {
        "date": 1,
        "temperatura": "$DHT11.Temperatura",
        "humedad": "$DHT11.Humedad",
        "Nivel_agua": "$NivelAgua",
        "CHIPID" : "$CHIPID"
        }
    },
    { "$sort": {"date": 1} }
];

}

return msg;
```

Para el resto de colecciones simplemente se ha de cambiar el campo que se desea agrupar para que coincida con el nombre del campo almacenado.

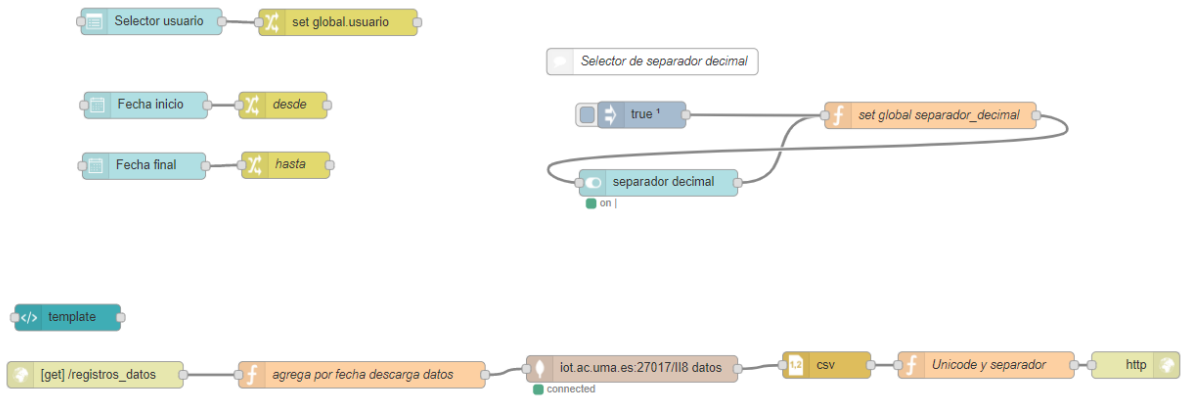
Los nombres establecidos serán importantes, puesto que de ello depende qué se descargará posteriormente.

Tras el nodo de MongoDB se coloca un nodo CSV para dar formato a las columnas que queremos descargar. Aquí introducimos los nombres de los campos que escogimos con $project en la agrupación.

Tras esto, se añade formato Unicode y el separador decimal, pudiendo elegir entre punto o coma, en función de lo que se escoja mediante un switch del dashboard.

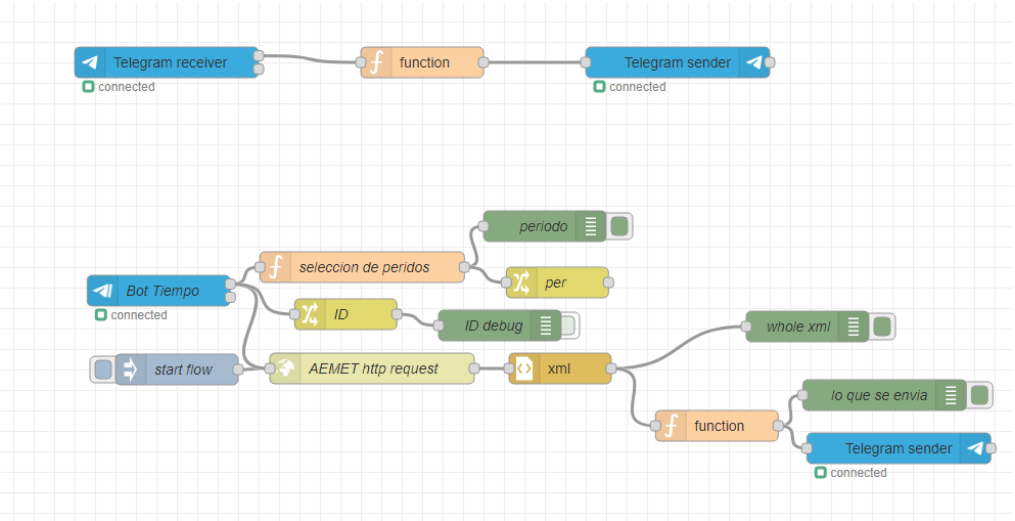
Por último, se coloca un nodo HTTP que establece el nombre del archivo a descargar.

El flujo descrito tendría el siguiente aspecto para los datos de los sensores:



La descarga del resto de colecciones sigue la misma estructura.

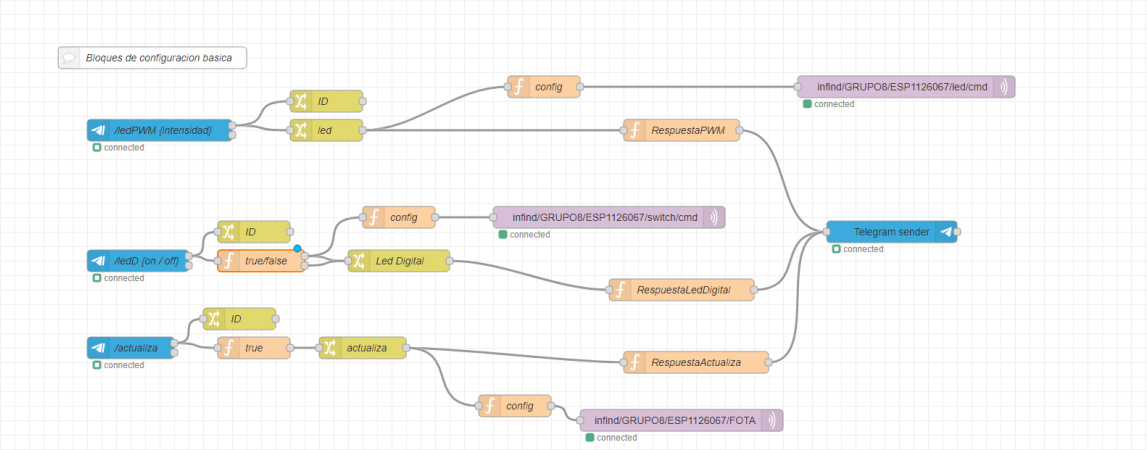
4.5. Telegram.



Se crea un flujo para interactuar con el bot en Telegram, para ello esta primera parte sirve para el comando /tiempo y para la ayuda, la ayuda lo forman los tres primeros nodos de arriba, un Telegram receiver, una función y un Telegram sender, la función muestra todos los comandos con sus parámetros si tienen, y una información breve del uso.

```
if(msg.payload.content.substring(0,1)!="/")
{
 msg.payload.content="Hola "+msg.originalMessage.from.first_name+" soy un bot ejecutando en NodeRED. Puedo responderte a estos comandos:\n";
 msg.payload.content+="/tiempo --> Para ver el tiempo de malaga\n";
  msg.payload.content+="/calefactor [on/off] --> Para activar o desactivar el controlador de temperatura automatico\n";
 msg.payload.content+="/temp [temperatura] --> Para establecer la temperatura del cuarto\n";
 msg.payload.content+="/ledPWM [intensidad] --> Establece la intensidad del led según el parametro\n";
 msg.payload.content+="/ledD [on/off] --> Para encender o apagar el led digital\n";
 msg.payload.content+="/actualiza --> Para actualizar la ESP\n";
 msg.payload.content+="/TempMax [temperatura] --> Para establecer un limite superior de temperatura\n";
 msg.payload.content+="/TempMin [temperatura] --> Para establecer un limite inferior de temperatura\n";
 return msg;
}
```

Para el comando /tiempo se ha copiado el ejercicio que se hizo en clase.

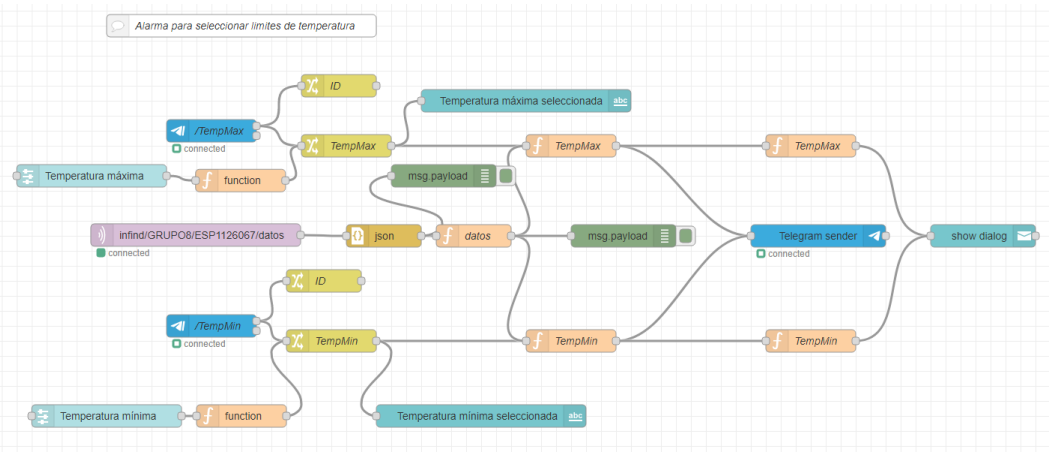


El comando /ledPWM {intensidad} se encarga de cambiar la intensidad del led de la placa, la función *config* pasa el parámetro a entero para enviarlo, ya que no funcionaría si enviamos un string que es lo que recibimos, para ello usamos la función *parseint(global.get("led"))* y generamos una respuesta con el valor del PWM.

En cuanto al comando */ledD {on/off},* controla el switch (Led Digital), para ello enviamos un true o un false según si nos llega encender o apagar, como se hizo en el flujo de IFTTT, enviamos el true o false por MQTT, y enviamos una respuesta por Telegram de confirmación.

Por último, el comando */actualiza* sirve para comprobar la actualización de la placa, para este nodo lo único que se necesita es una función que mande true siempre, y en la configuración montamos el mensaje con true y después ponemos a false la variable antes de mandar el mensaje por MQTT, además una respuesta por Telegram.

Pasamos al siguiente bloque, una alarma que muestra en el dashboard un mensaje y en Telegram cuando se supera una temperatura seleccionada o sobrepasa el límite inferior.



En primer lugar, hemos creado un slider en el dashboard y un nodo de comando de Telegram, /TempMax, /TempMin, luego guardamos ese número, y lo usamos para hacer un controlador que envíe mensajes cada vez que se cumpla la condición de que

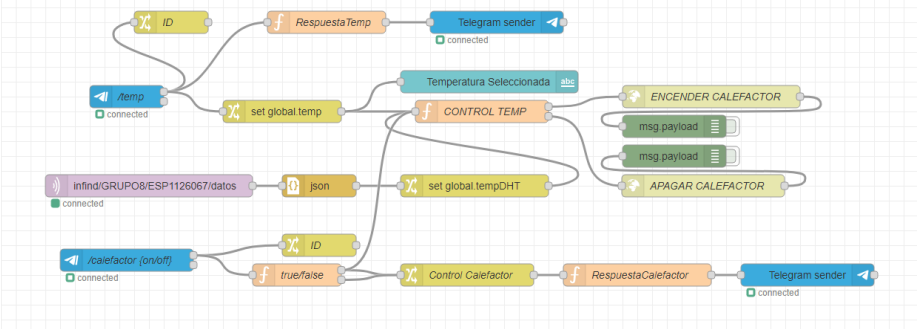
la temperatura sea mayor o menor a los límites. Para ello usamos una función TempMax o TempMin (en esta función se cambia la condición a MqttTemp<tempMin).

```
var tempMax=parseFloat(global.get("tempMax"));
var MqttTemp=msg.payload;

msg.payload={};

msg.payload.chatId=global.get("idtempMax");
msg.payload.type="message";

if(MqttTemp>=tempMax){

msg.payload.content="Se ha superado el limite de temperatura previsto de: "+tempMax+"ºC, la temperatura actual es de: "+MqttTemp+" ºC";

return msg;

}
```

Luego enviamos un aviso por Telegram y para poder sacarlo por notificación emergente necesitamos otra función, la cual añade un topic que es lo que muestra en la ventana emergente.

```
var tempMax=parseFloat(global.get("tempMax"));

msg.topic="Se ha superado el limite de temperatura previsto de: "+tempMax;

return msg;
```

4.6. Control IFTTT.



Se ha creado un módulo que controle la temperatura de un cuarto, para ello necesitamos únicamente un calefactor y un enchufe inteligente, que se controla a través de la app Smart life.

La temperatura del cuarto será controlada a través de Telegram. Para ello, se usa un nodo “command” de la librería de Telegram (*node-red-contrib-telegrambot*) y recibimos un comando /temp {temperatura}, el cual debe ser número. Primero enviamos un mensaje de respuesta a través del módulo RespuestaTemp, y comprobamos si el flujo está activado o no.

```
var tempDHT=global.get("tempDHT");
var temp =msg.payload.content;
var on = global.get("on");
msg.payload.chatId=global.get("id");
msg.payload.type="message";

if(on) msg.payload.content="La temperatura del cuarto es de "+tempDHT+"ºC y la que se quiere es de" +temp+"ºC, me pongo con ello.";
else if(!on) msg.payload.content="La temperatura del cuarto es de "+tempDHT+"ºC, pero el controlador esta desactivado, activalo usando /calefactor on";
return msg;
```

El encendido y apagado del calefactor ser hace también desde Telegram. Escribimos el comando */calefactor {on/off}* que guarda una variable, la cual es true cuando mandamos un on y false cuando mandamos un off.

```
var estado=msg.payload.content;
var men;
var error;
if(estado==" on" || estado==" ON"){
    men={payload:true};
    return [men,null];

}
else if(estado==" off" || estado==" OFF"){
    men={payload:false};
    return [men,null];
}
else{
    error={payload:"error"};
    return [null,error];
}
```

Continuamos mandando un mensaje de confirmación tanto para el comando */calefactor* como para el */temp*. Para ello, usamos dos nodos “Telegram sender”. En el caso del comando */calefactor* hay que montar el mensaje, para ello usamos *RespuestaCalefactor* que es una función.

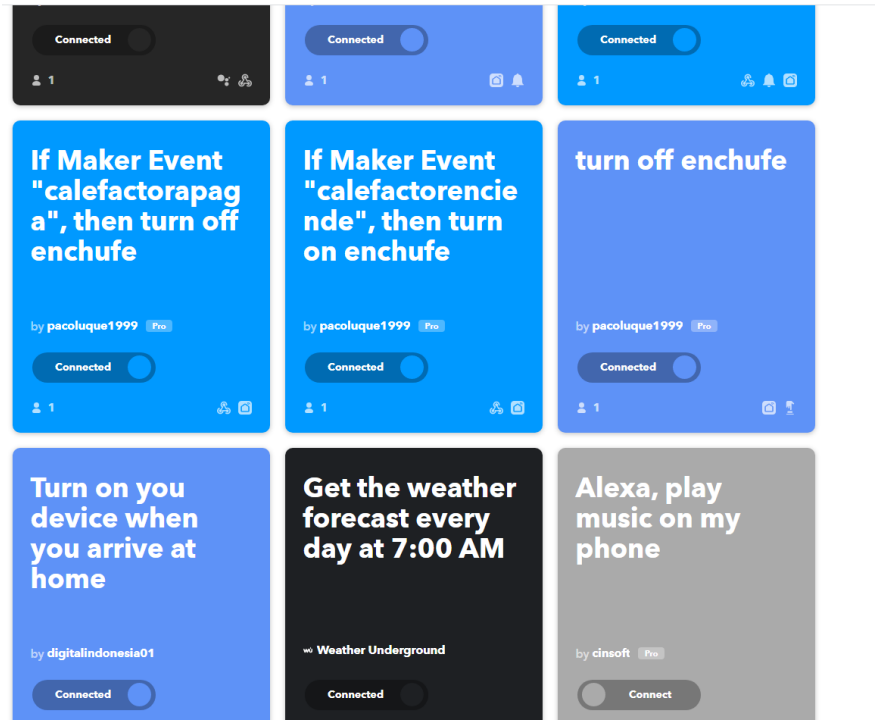
```
msg.payload={};
var on=global.get("on");
var estado;

if(on==true)estado="Activando el flujo del calefactor";
else if(on==false)estado="Desactivando el flujo de control del calefactor";
else if(on=="error")estado="Error al interpretar el comando"
msg.payload={};

msg.payload.chatId=global.get("id");
msg.payload.type="message";

msg.payload.content=estado+"\n";

return msg;
```

Después de recibir el parámetro de */temp* lo guardamos como variable global y pasamos al bloque principal, una función que se encarga de comprobar las temperaturas para encender o apagar el calefactor, con función de controlador.

```
 temp=global.get("temp");
 tempDHT=global.get("tempDHT");
 on=global.get("on");
 enciende;
 apaga;
(temp<=tempDHT || !on){
 apaga={payload: "apagando"};


 return [null,apaga];

e if(temp>tempDHT && on){
 enciende={payload: "enciende"};

 return [enciende,null];
```

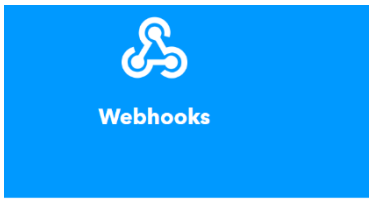
Si la temperatura recibida por MQTT es menor que la seleccionada, se enciende el calefactor. Sin embargo, cuando la temperatura recibida es mayor, este lo apaga y se generan dos salidas encargadas de mandar la información de encendido o apagado a un nodo de Webhook, el cual se encargará de enviar información al servicio IFTTT para encender/apagar el calefactor.

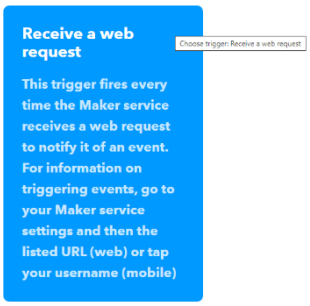
Para configurar el ultimo módulo solo tenemos que configurar el “evento” por IFTTT:

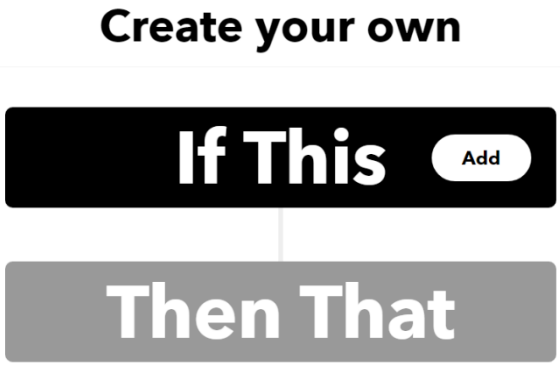


Los eventos se llaman “If Marker Event ‘calefactorapaga’ turn off enchufe” y “If Marker Event ‘calefactorenciende’ turn on enchufe”. Se pueden observar más eventos que afectan al enchufe, como por ejemplo encender el calefactor cuando entras en una ubicación cerca de casa.

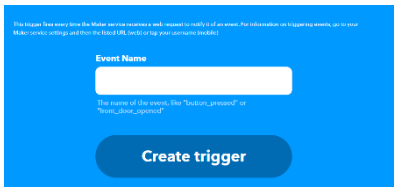
Se va a crear un nuevo evento desde cero para explicar cómo se ha creado. Primero accedemos al panel *create*:



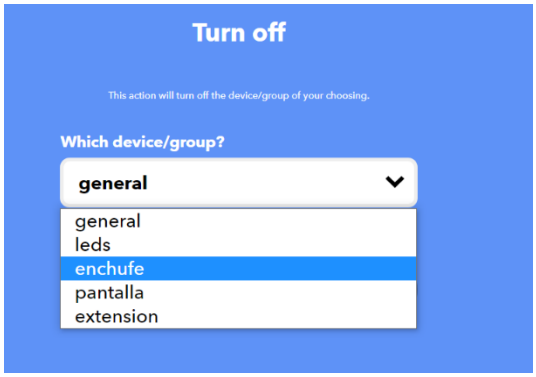




A continuación, buscamos el servicio webhook y le damos a recibe a web request, para darle un nombre a nuestro evento.

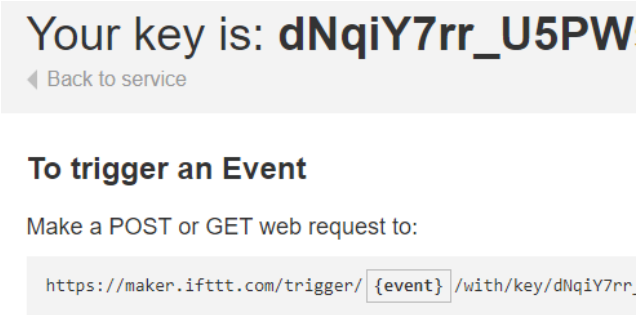


Se selecciona “Then That” y se busca el servicio “Smart life” y buscamos la opción de *turn off* o *turn on* dependiendo si queremos apagarlo o encenderlo.



Ya estaría creado el evento, ahora solo queda comprobar la documentación del Webhook, para enviarle solicitudes desde Node-RED.

Finalmente, obtenemos un enlace que se genera para la cuenta de IFTTT:



5. Resultados y conclusiones.

Hemos conseguido desarrollar una aplicación capaz de almacenar y consultar una gran cantidad de información que nos permite, además, actuar sobre la red de dispositivos de una forma muy cómoda. Se trata de un sistema robusto y tolerante a errores que permite al usuario utilizar la aplicación de una forma muy intuitiva, sin necesidad de que posea conocimientos de informática, ya que, además, se ha documentado debidamente con un manual de usuario.

Gran parte de ello ha sido posible gracias a Node-RED, que nos permite la implementación de una infinitud de funcionalidades de una forma muy sencilla y, sobre todo, gracias a la integración que presenta con otras herramientas.

Por un lado, el envío de mensajes por MQTT formateados en JSON hace que sea posible una conversión directa a objetos JavaScript, lo que permite un procesamiento muy cómodo de los mensajes, añadiendo los campos que necesitemos y extrayendo información de los mismos.

En cuanto a la base de datos MongoDB, nos permite almacenar una gran cantidad de información y acceder a ella según múltiples criterios, recuperando solo el contenido en el que estemos interesados, a la vez que nos permite agrupar los datos como queramos y calcular estadísticos de una forma muy sencilla.

Por otro lado, la integración directa con Telegram que incluye Node-RED, ha hecho posible la creación de un interfaz para nuestra aplicación accesible desde cualquier dispositivo, permitiéndonos consultar cualquier dato y actuar sobre los elementos de la red de forma remota.

Por último, cabe destacar las posibilidades que nos permiten otras herramientas utilizadas como IFTTT, que incluye un montón de funcionalidades muy prácticas que se pueden utilizar de forma muy sencilla y desde un alto nivel, sin necesidad de entrar en los aspectos técnicos más profundos que hacen que ello sea posible.

MANUAL DE USUARIO

1. Puesta en funcionamiento.

En primer lugar, deberemos establecer la conexión con nuestro WIFI. Para ello, es necesario cambiar la ssid y la password del código Arduino suministrado por las correspondientes de nuestro WIFI.

A continuación, habrá que incorporar el flujo Node-RED dado. Para poder establecer la conexión con el servidor MQTT, tendremos que introducir el nombre de usuario y la contraseña en su respectivo bloque dentro de Node-RED. Será necesario también introducir tanto el usuario como la clave para el uso de las bases de datos.

Una vez realizadas dichas configuraciones iniciales, el proyecto estará listo para realizar las funciones para las que ha sido programado.

2. Dashboard.

El dashboard de Node-RED será el interfaz que mayor número de opciones y funcionalidades nos dará para poder controlar nuestros sistemas.

Por un lado, podremos comprobar los datos actuales gracias a las últimas lecturas realizadas por los sensores, además de otras características que nos proporciona la placa. Por otro lado, se podrán controlar los actuadores estableciendo las consignas deseadas para su funcionamiento. El dashboard se encuentra dividido en varias páginas:

- **Datos actuales.** En esta página del dashboard se encontrarán todos los datos que se están tomando. Estos datos se irán actualizando de forma periódica cada 5 minutos por defecto, aunque es posible cambiar este tiempo en la parte dedicada a la configuración.

    - Conexión. En primer lugar, encontramos un selector que nos permite elegir el usuario del que se mostrarán los datos actuales en el dashboard. En el campo online se indica si se ha establecido la conexión de forma correcta mediante true o si no hay conexión mediante un false. Por último, el uptime muestra el tiempo en milisegundos que lleva un usuario conectado, es decir, con el campo online a true.

    - Última actualización. En este apartado encontramos la hora y la fecha en la que se ha realizado la última actualización de los datos.

    - Actuadores. En esta sección encontramos los datos de los actuadores. Por un lado, se indica de forma gráfica tanto la

intensidad del LED que funciona con PWM desde 0 hasta 100, como si el LED que funciona como interruptor (switch) se encuentra encendido o apagado. Por otro lado, podemos comprobar cuál ha sido el origen de la orden que ha modificado el estado de alguno de los dos LEDs: mediante el botón de la propia placa o mediante el dashboard a través de MQTT.

- Wifi. Podemos visualizar los datos relativos al Wifi nos proporciona la placa como son: el SSId, el indicador de fuerza de la señal recibida (RSSI) y la dirección IP.

- Sensores. El sensor DHT11 nos da las medidas de la temperatura y humedad actuales, representadas en forma gráfica mediante un gauge. Además, se dispone de otro sensor que mide el nivel de agua en tanto por ciento del recipiente en el que se encuentra.

- **Consultas.** Esta página del dashboard está dedicada a la realización de consultas de diferentes datos en el periodo de tiempo que elijamos.

    - Selector. De igual modo que en la página de datos actuales, lo primero que encontramos es un selector para indicar el usuario al que se le va a realizar la consulta de datos. En esta ocasión, hay una opción que nos permite consultar los datos de todos los usuarios al mismo tiempo ya que se trabaja con estadísticos y registros. En esta sección encontramos también dos calendarios, en los que se podrá elegir la fecha de inicio y la de fin respectivamente si se desea.

    - Consulta por rango de fechas. Si se ha establecido un rango de fechas correcto, es posible realizar una consulta en ese periodo de tiempo establecido. Según el usuario elegido, se mostrarán los estadísticos (media, máximo, mínimo, número de datos) de los datos de los sensores; es decir, de la temperatura, la humedad y el nivel de agua. Además, se representarán mediante una gráfica los estadísticos de cada dato por días.

    - Consultas rápidas. En este tipo de consultas no es necesario seleccionar un rango de fechas, pero sí un usuario. Tenemos las opciones de consultar los datos del último día, de la última semana o del último mes; pulsando su respectivo botón. De igual manera que en la opción de rango de fechas, se mostrarán los estadísticos (media, máximo, mínimo, número de datos) de los datos de los sensores; pero en el periodo de tiempo que se haya seleccionado. Además, se representarán mediante una gráfica los estadísticos de

cada dato por días si se ha elegido la consulta de la última semana o del último mes; o por horas si se ha elegido la consulta del último día.

- Consulta historial. Por último, en esta página se permite realizar una consulta del historial de cambios en ciertos datos en el rango de fechas y usuario seleccionados.
  Es posible obtener un registro de los errores de los sensores disponibles, de manera que si alguno de ellos falla se almacena la fecha y la hora a la que tuvo lugar el error junto con el sensor que ha tenido el problema. Primero se mostraría la fecha y la hora, luego los tres sensores seguidos de su estado de funcionamiento (true si no funciona correctamente o false si ha habido un error en su funcionamiento) y al final el ChipId del usuario que ha tenido el fallo.
  También se puede obtener un registro de los cambios que ha habido en el LED con funcionamiento de PWM y en el LED con funcionamiento de interruptor. Primero se mostraría la fecha y la hora, luego el ChipId del usuario que ha realizado algún cambio en el LED, después la intensidad a la que se ha cambiado el led (en el caso de PWM) o si el LED se encuentra encendido o apagado (en el caso de switch) y al final el origen desde el que se ha realizado la orden.
  Finalmente, se puede obtener un registro de los cambios en el estado de conexión. Primero se mostraría la fecha y la hora, luego el ChipId del usuario que ha realizado algún cambio en el led, después el estado de la conexión (true si se encuentra conectado o false si se encuentra desconectado) y al final el ChipId del usuario que ha tenido el cambio.

- **Descargas.** Esta página del dashboard está dedicada a la realización de descargas de diferentes datos en un fichero en formato CSV en el periodo de tiempo que elijamos.
  - En primer lugar, encontramos un selector para indicar el usuario del que se realizará la descarga de datos o si se realizará de todos los usuarios disponibles. También tenemos los dos calendarios para elegir el rango de fechas de la toma de los datos que se descargarán. Después de esto, encontramos la elección del separador decimal en forma de punto o de coma que se mostrará en el archivo CSV.

  - Tenemos la posibilidad de hacer cinco descargas diferentes: registros de conexión, registro de datos, registro de errores, registro de LED y registro de switch. Al pulsar el botón, se realizará la descarga seleccionada de los registros tal y como se mostraban en la página anterior.

- **Configuración y acciones.** En esta página del dashboard podemos indicar diferentes actuaciones para el control del dispositivo.

  Configuración:

  - Frecuencia envío datos. En este apartado es posible cambiar el tiempo en el que se actualizan periódicamente los datos. Para ello, solo es necesario escribir el nuevo tiempo de actualización de datos en segundos.

  - Frecuencia cambio LED. Cuando se produce un cambio en la intensidad del LED se realiza de forma gradual a una velocidad de ±1% cada 10ms hasta alcanzar la nueva intensidad establecida. En este apartado es posible introducir un nuevo tiempo de cambio en la intensidad del led en milisegundos.

  - Frecuencia actualización. Una de las maneras de programar una actualización del firmware es escribiendo en este campo el tiempo en minutos en el que se comprobarán de forma periódica las actualizaciones. Inicialmente no hay un tiempo definido, por lo que solo se comprobarán las actualizaciones periódicamente si se escribe el tiempo; y de la misma manera se dejarán de comprobar si escribimos un 0. El tiempo en minutos para la frecuencia de actualización deberá de ser entero mayor o igual que 0.

  - Lógica del LED/Swith. Por defecto los LEDs de la placa funcionan con lógica negativa. Sin embargo, si se conecta un nuevo LED al dispositivo que funcione con lógica positiva, es posible realizar el cambio y que funcione de forma adecuada. Para ello, es posible seleccionar el estado OFF que indica la lógica negativa y el estado ON que indica la lógica positiva.

  Órdenes actuadores:

  - Intensidad del LED. Para controlar la intensidad del LED se utiliza un slider, mediante el cual se puede variar desde 0 hasta 100 (siendo 0 apagado y 100 encendido a intensidad máxima).

  - Swicth OFF-ON. Para controlar el encendido y el apagado del LED que funciona como interruptor, solo hay que seleccionar el estado OFF si queremos apágalo o el estado ON si queremos encenderlo.

Actualización firmware:

- Actualizar. Al pulsar este botón se empezará a comprobar al instante si hay alguna actualización del firmware disponible para descargar.

3. ESP8266.

A través de la propia placa ESP8266 también podemos realizar ciertas funciones que nos permiten controlar algunos aspectos del dispositivo.

En ella, disponemos del GPIO2, en el que se controlará la iluminación mediante PWM, y del GPIO16, que se comportará como interruptor pudiendo solo encenderse y apagarse. Además, es posible interactuar con el dispositivo mediante el botón (flash) de la siguiente manera:

- Al pulsar el botón, el LED se apagará o encenderá al nivel establecido previamente.
- Si se realiza una pulsación doble, el LED se encenderá a nivel máximo.
- Si se pulsa el botón de forma prolongada, el dispositivo comprobará si hay una actualización disponible en ese instante.

Todas estas acciones seguirán siendo visibles en el dashboard, independientemente de que se utilice la propia placa para controlar el dispositivo.

4. Telegram.

A través de la aplicación de Telegram, se podrá entablar una conversación con el Bot de nuestro sistema para preguntar sobre los datos que queramos saber. También será posible controlar el sistema en el mismo Bot. Sin la necesidad de usar el dashboard de Node-RED, podremos interactuar con el sistema a través de la aplicación de Telegram en nuestro dispositivo móvil u ordenador. Para ello es necesario introducir tanto el nombre del bot como el token en los nodos de Node-RED dedicado a Telegram.

Al abrir la aplicación de Telegram y empezar una conversación con nuestro Bot podemos escribir los siguientes comandos:

- /ledPWM [intensidad] → Permite establecer la intensidad del LED que funciona como PWM según el parámetro que mandemos entre 0 y 100.
- ledD [on/off] → Permite encender o apagar el LED digital que funciona como interruptor.
- /actualiza → Permite actualizar el firmware.
- /TempMax [temperatura] → Permite establecer un límite superior de temperatura.

- /TempMin [temperatura] → Permite establecer un límite inferior de temperatura.

En estas dos últimas acciones, si nuestro sensor DHT11 mide una temperatura superior a la máxima o inferior a la mínima, recibiremos un mensaje por Telegram avisándonos de que hemos superado el límite máximo o nos encontramos por debajo del límite mínimo. Estos límites también serán configurables mediante un slider, que permitirá elegir una temperatura determinada y mostrarla en el apartado del dashboard dedicado a Telegram.

Otras de las acciones que se han implementado mediante comandos en Telegram es controlar un calefactor, haciendo uso también de la herramienta IFTTT y dispositivos Wifi tales como enchufes.

- /calefactor [on/off] → Permite activar o desactivar el controlador de temperatura automático.
- /temp [temperatura] → Permite establecer la temperatura de la habitación según el parámetro enviado.