industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

Universidad Politécnica de Cartagena

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Sistema de monitorización y análisis de consumos en viviendas

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: **Álvaro Díaz Molina.**
Director: Manuel Jiménez Buendía
Codirector: José Alfons Vera Repullo

Cartagena, ……

RESUMEN

Durante la realización de este trabajo de fin de grado se han aunado los conocimientos aprendidos en diversas asignaturas de la carrera con la finalidad de desarrollar un sistema inalámbrico, de bajo coste y bajo consumo para la monitorización de consumos de agua en viviendas. El sistema será capaz de detectar consumos anómalos y avisar de posibles averías en la instalación mediante los datos medidos.

ABSTRACT

During the completion of this final degree project, the knowledge learned in various subjects of the career has been combined with the proposal to develop a low cost and low consumption wireless system for monitoring water consumption in homes. The system will be able to detect anomalous consumption and warn of possible failures in the installation using the measured data.

Índice

Índice de figuras

1 Introducción

1.1 Marco de trabajo

Hoy en día es cada vez más común oír hablar sobre el IoT ("Internet of Things"), y las ventajas que derivarían de su inclusión en diversos dispositivos: mayor eficiencia, seguridad y control sobre los aparatos que utilizamos en el día a día.

Con este TFG se pretende aplicar el paradigma del IoT, tan presente hoy en día, para desarrollar un sistema inalámbrico que monitorice consumos de agua, electricidad y gas entre otros…

1.2 Descripción

El sistema deberá realizar una gestión de incidencias para avisar de comportamientos o consumos anómalos, así como tener un reducido consumo. Esto será posible mediante la toma de decisiones de forma autónoma por parte del sistema, aunque siempre en base a parámetros fijados de antemano.

Para construir el sistema de gestión se cuenta con:

- ESP8266-12E/ESP8266-12F, un microcontrolador de reducido tamaño con *stack* TCP/IP completo.
- Raspberry PI 3, un ordenador de placa única y bajo coste.
- Un contador de agua que medirá el flujo de agua.
- Una electroválvula para poder controlar el paso del agua.

Además, una importante parte del proyecto consiste en el desarrollo del software implementado dentro de la raspberry y el microcontrolador. Durante el proyecto se han utilizado las siguientes tecnologías:

- IoBroker, una plataforma ideal para la integración del Internet de las Cosas.
- Node-RED, una herramienta de programación para conectar dispositivos hardware APIs y servicios online.
- MQTT, un protocolo de conectividad M2M (*machine to machine*).
- MySQL, un sistema de gestión de bases de datos.
- ArduinoIDE, el entorno de programación elegido para programar el ESP8266.

1.3 Objetivos

El objetivo de este proyecto será el de diseñar un sistema que nos permita monitorizar consumos con la finalidad de:

1. Generar históricos y analizar los datos en intervalos de tiempo definidos.
2. Detectar anomalías en el consumo y/o posibles fugas y averías.
3. Generar mecanismos de respuesta que comuniquen las incidencias y protejan la vivienda frente a posibles fallos o consumos no autorizados.

Para este proyecto se ha intentado crear un sistema lo más agnóstico posible ante la variable de consumo a medir. No obstante, durante la realización se ha tomado como caso de estudio el análisis del consumo de agua.

2 Estado del arte: Antecedentes de integración de sistemas del Internet de las cosas (IoT).

2.1 Introducción

Se conoce como 'Internet de las cosas' (de ahora en adelante utilizando su acrónimo IoT, del inglés *Internet of Things*) a la unión de sistemas convencionales con canales de acceso a internet. Es decir, la integración de sistemas cotidianos anteriormente no comunicables con internet.

En 2009, en un artículo para el diario RFID, "Esa cosa del 'internet de las cosas'" Kevin Ashton hizo esta declaración:

> *Los ordenadores actuales —y, por tanto, internet— son prácticamente dependientes de los seres humanos para recabar información. Una mayoría de los casi 50 petabytes (un petabyte son 1024 terabytes) de datos disponibles en internet fueron inicialmente creados por humanos, a base de teclear, presionar un botón, tomar una imagen digital o escanear un código de barras. Los diagramas convencionales de internet, dejan fuera a los routers más importantes de todos: las personas. El problema es que las personas tienen un tiempo, una atención y una precisión limitados, y no se les da muy bien conseguir información sobre cosas en el mundo real. Y eso es un gran obstáculo. Somos cuerpos físicos, al igual que el medio que nos rodea. No podemos comer bits, ni quemarlos para resguardarnos del frío, ni meterlos en tanques de gas. Las ideas y la información son importantes, pero las cosas cotidianas tienen mucho más valor. Aunque, la tecnología de la información actual es tan dependiente de los datos escritos por personas que nuestros ordenadores saben más sobre ideas que sobre cosas. Si tuviéramos ordenadores que supieran todo lo que tuvieran que saber sobre las "cosas", mediante el uso de datos que ellos mismos pudieran recoger sin nuestra ayuda, nosotros podríamos monitorizar, contar y localizar todo a nuestro alrededor, de esta manera se reducirían increíblemente gastos, pérdidas y costes. Sabríamos cuándo reemplazar, reparar o recuperar lo que fuera, así como conocer si su funcionamiento estuviera siendo correcto. El internet de las cosas tiene el potencial para cambiar el mundo tal y como hizo la revolución digital hace unas décadas. Tal vez incluso hasta más.* (1)

De esta declaración inicial se desprenden algunas de las principales ventajas de los sistemas IoT; descargar a las personas de la pesada carga de la adquisición de datos y posibilitar la comunicación entre distintos dispositivos sin necesidad de la intervención humana.

2.2 Origen y primeros pasos del Internet de las Cosas.

¿Pero de donde viene la expresión IoT? Kevin Ashton fue la primera persona en acuñar el término.

En 1999 Kevin tenía 28 años y trabajaba en Procter & Gamble (P&G) tratando de resolver el siguiente problema: los productos más populares no se encontraban disponibles en tiendas. El motivo era que a mayor publicidad más rápido se agotaba el producto, pero la información de stock bajo tardaba mucho en llegar hasta el fabricante. En su opinión la solución era poner sensores conectados a la red en los productos de P&G para saber cuándo se quedaban sin stock en las tiendas. Recordemos que estamos hablando de los años 90 cuando las conexiones eran muy lentas y se hacían a través de la línea telefónica.

Para intentar convencer a los ejecutivos Kevin decidió incluir la palabra “internet” porque en aquel momento había un boom en todo lo relacionado con esto. Por otro lado, la palabra “cosa” se usaba porqué se empezaba a tener la idea de empotrar computadoras en otros objetos, además de que cada vez las computadoras eran más pequeñas y baratas.

Con esto en mente Kevin Ashton comenzó sus labores de investigación en el MIT y mientras allí se encontraba realizó muchas presentaciones de PowerPoint por todo el mundo donde se incluía el termino IoT.

Alrededor de 2009 el término se volvió muy popular, gracias en gran parte a Twitter y al hashtag #IoT.

2.3 El internet de las cosas hoy en día.

Uno de las principales puntas de lanza de la utilización del paradigma IoT sería su aplicación en las llamadas "*Smart Cities*". Algunos de los posibles campos de actuación del internet de las cosas en las ciudades actualmente son:

- Monitorización de la salud estructural de edificios históricos y monumentos.
- Una mejor gestión de basuras que permita reducir costes y optimizar procesos mediante el análisis de que ruta sería la más eficiente a recorrer en la recogida de basuras.
- Monitorizar la calidad del aire por zonas, ofreciendo estos datos a la ciudadanía se podrían organizar actividades al aire libre que se desarrollen en las zonas con mejor calidad del aire.
- Monitorización del ruido para, por ejemplo, evitar la contaminación acústica de zonas de ocio.
- Atascos y tráfico. Obteniendo información sobre el estado del tráfico las personas podrías elegir otras rutas o verificar que el transporte público es una mejor opción en muchos casos.
- Analizar el consumo energético en la ciudad, identificando las principales fuentes de consumo para priorizar la optimización, así como la instalación de paneles fotovoltaicos en la ciudad.
- Sistemas de parking inteligente, guiando a los conductores se obtendrían tiempos más cortos de búsqueda de estacionamiento y por ende menores emisiones dentro de la ciudad.
- Gestión de la iluminación. Permitiría el encendido progresivo de la iluminación en función de la luz natural existente en cada día.
- Automatización en edificios públicos. Mejorando el control de luces, calefacción/aire acondicionado, humedad, etc. Para así aumentar el grado de confort de la gente que utiliza estos espacios y reducir los costes.

(2)

En el plano industrial siempre han existido muchos recelos a la hora de integrar nuevas tecnologías que pudiesen comprometer la seguridad de los datos y una posible difusión de información crítica.

Sin embargo, en la actualidad existe una corriente en expansión hacia la integración de nuevos sistemas especialmente en fábricas altamente automatizadas, donde la inclusión de sistemas IIoT ("Industrial Internet of Things) aportar nuevas capas de información para tomar decisiones respecto a procesos de mejora continua o mantenimiento preventivo.

Otro de los campos donde se están logrando grandes mejoras es el del sector de Oil&Gas. Gracias a una monitorización continua se puede predecir el estado de las explotaciones y reducir de forma controlada las inspecciones innecesarias. Se espera que este tipo de tecnologías siga implantándose en un futuro a corto plazo.

El adoptar estas nuevas metodologías está modificando el uso de la información para pasar de sistemas preventivos a sistemas reactivos capaces de tomar decisiones inmediatas en función de la información existente en tiempo real, en lugar de llevar a cabo acciones en previsión de posibles fallos con un mayor grado de incertidumbre. Un estudio de Aberdeen Research estima que el coste medio por hora de parada en fábrica asciende a 260.000$. El análisis de la situación de una fábrica automatizada puede reducir costes, así como actualizar las relaciones logísticas con clientes y proveedores.

(3)

2.4 Estimación de pérdidas en viviendas.

Si bien es difícil conocer con exactitud las fugas que se producen en una vivienda en concreto, dado que la mayoría de estudios se focalizan en las pérdidas conjuntas de la red de abastecimiento y viviendas, sin desagregar las pérdidas de la vivienda de la red de abastecimiento se ha intentado modelar cuál podría ser el ahorro ofrecido por el dispositivo creado durante el desarrollo de este TFG.

En primer lugar, habría que diferenciar entre las que denominaremos fugas mayores y fugas menores. Al hablar de fuga mayor se hace referencia a cualquier pérdida de agua que es fácilmente por los habitantes de la vivienda, ya sea en el momento que se produce o a los pocos días, ya sea porque el caudal de agua perdido es tan grande como para generar humedades rápidamente, inundar una habitación o provocar una fuga que se puede ver a simple vista.

En cambio, las fugas menores no pueden ser detectadas con facilidad y pueden conllevar un consumo fantasma de bajo nivel que se continua en el tiempo siendo detectable únicamente si los habitantes de la vivienda se ausentan durante un período de tiempo medio/largo y evalúan así que el consumo no es cero durante el tiempo que nadie realizaba consumo alguno de agua en la vivienda.

En este segundo caso se han utilizado los datos de este estudio: **DETERMINACIÓN DEL VOLUMEN DE FUGAS EN INSTALACIONES INTERIORES DE SUMINISTRO DE AGUA** (4)

En este estudio se recogen los datos de dos viviendas con similar consumo. Partiendo de que en la vivienda A no existen pérdidas y que en la vivienda B se sospecha de pequeñas fugas se obtienen los siguientes datos:

	Vivienda A	**Vivienda B**
Caudal medio consumido (l/día)	451	422
Error medio ponderado (%)	4.69%	18.53%
Volumen no contabilizado (l/día)	21.15	78.20

Admitiendo que el contaje de ambas viviendas es similar, la diferencia entre ambas debe corresponder a fugas en la instalación. Si en la vivienda A se están dejando de contar 21,15 litros/día por errores de medida en el contador y en la vivienda B unos 78,20 litros/día, se puede considerar como estimación válida que hay en torno a 57,1 litros/día de pérdidas en la vivienda B que en un año equivalen a 20,54 m^3.

En la actualidad un hogar en España paga unos 1,84€ de media por metro cúbico por lo tanto en este caso las pérdidas anuales son de 37,80 € al año.

Esta cantidad puede no parecer muy alta a primera vista, pero hay que tener en cuenta que se pueden dar fugas de mayores proporciones y que superen esta cantidad en un período de tiempo menor.

Partamos del supuesto de que tenemos una acometida con un caudal máximo de 1 litro/segundo. En el caso de una rotura de la tubería podríamos hablar de unas pérdidas de 3,6 m^3/hora que equivaldrían a unos 6,48€/hora máximos. Hay que tener en cuenta que la rotura puede producirse en puntos de menor caudal.

También se dispone de los datos de consumo de una vivienda unifamiliar que durante varios años que ha sufrido sucesivas pérdidas por la rotura de las mangueras de riego. Estos son los datos de consumo de la factura en función del mes:

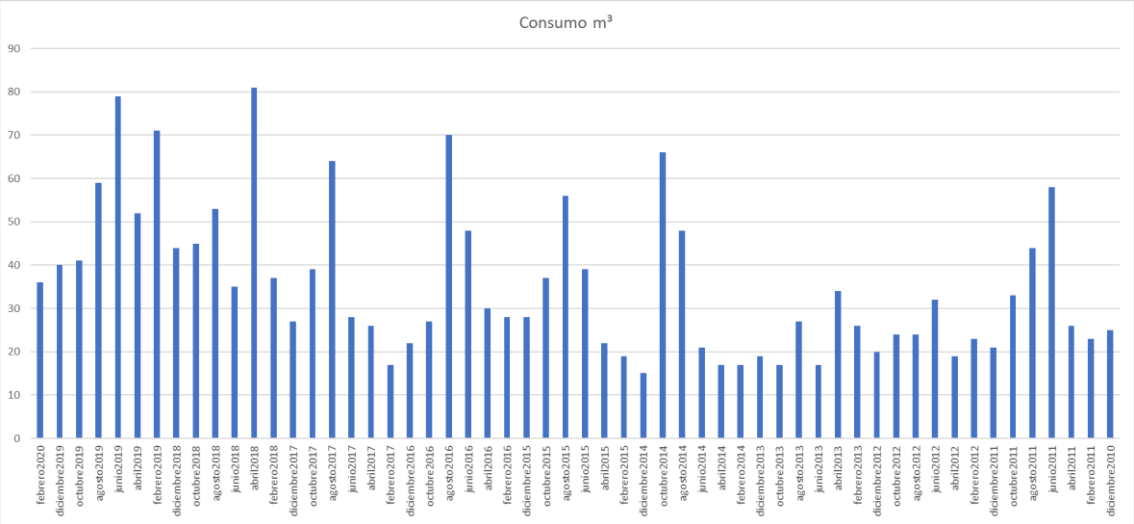

FIG. 1 - CONSUMO EN VIVIENDA UNIFAMILIAR

Para analizar las pérdidas se han cogido los valores más extremos y se ha comparado su valor con el consumo de años anteriores y posteriores durante el mismo mes de facturación.

Comparativa de junio de 2011 con junio de 2013 y 2012				
jun-13	jun-11	jun-12	Incremento en m3	Incremento en €
17	58	32	33.5	43.55

Comparativa de octubre de 2014 con octubre de 2015 y 2013				
oct-15	oct-14	oct-13	Incremento en m3	Incremento en €
37	66	17	39	50.7

Comparativa de agosto de 2016 con agosto de 2017 y 2015				
ago-17	ago-16	ago-15	Incremento en m3	Incremento en €
64	70	56	10	13

Comparativa de agosto de 2017 con agosto de 2018 y 2016				
ago-18	ago-17	ago-16	Incremento en m3	Incremento en €
35	64	70	11.5	14.95

Comparativa de abril de 2018 con abril de 2019 y 2017				
abr-19	abr-18	abr-17	Incremento en m3	Incremento en €
52	81	26	42	54.6

Comparativa de febrero de 2019 con febrero de 2020 y 2018				
feb-20	feb-19	feb-18	Incremento en m3	Incremento en €
36	71	37	34.5	44.85

Comparativa de junio de 2019 con junio de 2018 y 2017				
jun-18	jun-19	jun-17	Incremento en m3	Incremento en €
35	79	28	47.5	61.75

Algunos de los incrementos anuales que corresponden a los meses de verano son muy pequeños respecto al consumo de años colindantes durante el mismo periodo (agosto entre 2018 y 2015) por lo tanto estos consumos durante los meses más calurosos del verano se consideran normales.

Sin embargo, el cómputo del resto de consumo anómalos asciende a 255.45 € durante un período de 8 años. Algo que concuerda con las pérdidas detectadas de forma tardía.

Por tanto, con la solución propuesta en este TFG se pretenderá afrontar ambos tipos de problemas.

3 Diseño de la solución

Para este proyecto se ha construido una maqueta que pretende similar la entrada de agua de una vivienda unifamiliar estándar

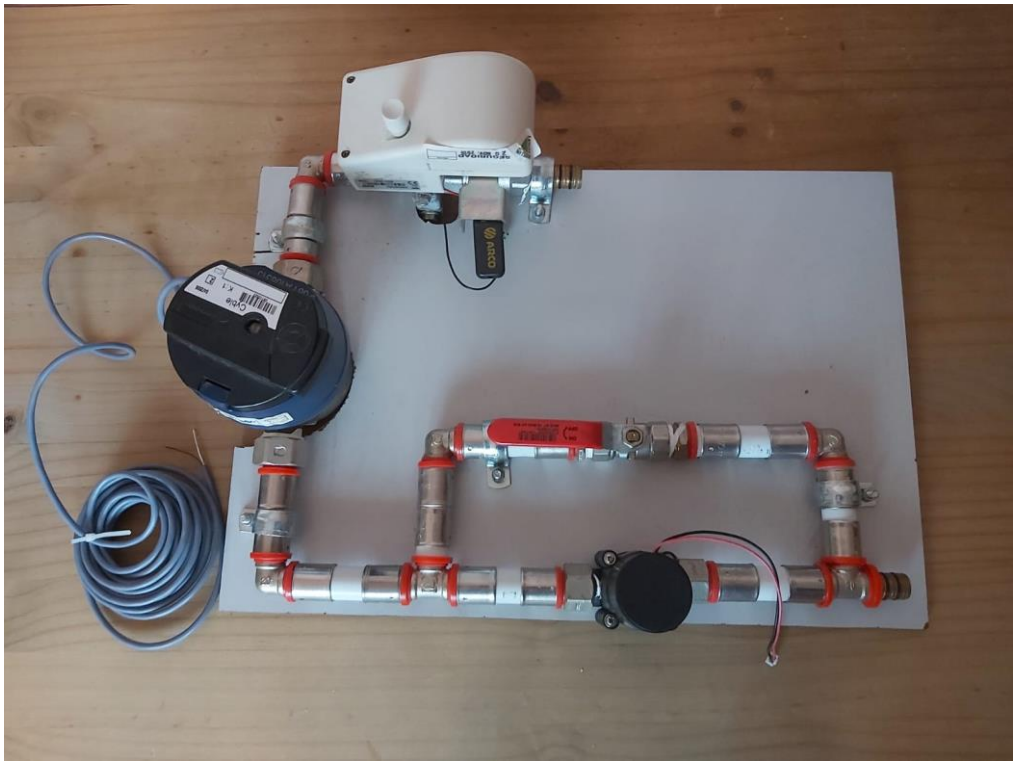

FIG. 2 - MAQUETA DEL PROYECTO.

La maqueta está compuesta de una llave de paso motorizada para abrir o cerrar el paso de agua totalmente. Un contador para contabilizar los litros de agua que pasan por la tubería y una turbina para aprovechar el paso de agua para generar energía. En paralelo a la turbina se ha instalado otra llave de paso para manejar que porcentaje del caudal pasa por la turbina y regular la posible pérdida de carga que se produciría en esta.

Cerca de esta maqueta se instalará la placa diseñada con el ESP8266 en una caja estanca para evitar que posibles fugas de agua puedan dañar los componentes electrónicos.

3.1 Diseño de la arquitectura

A la hora de hablar sobre el funcionamiento del sistema se puede hacer distinción entre dos partes bien diferenciadas.

Por un lado, tendremos un conjunto compuesto por un sensor (medirá la cantidad de agua que pasa por la tubería), un actuador (todo o nada, abre o cierra el paso por la tubería), y el ESP8266 que servirá de enlace con la raspberry. El microcontrolador enviará toda la información y ejecutará las órdenes de actuación sobre la electroválvula.

Por otro lado, la raspberry almacenará la información obtenida en una base de datos, ejecutará acciones en función de estos y de la configuración establecida y permitirá la configuración del sistema. Además, habilitará el acceso a los datos y su interconectividad con KNX.

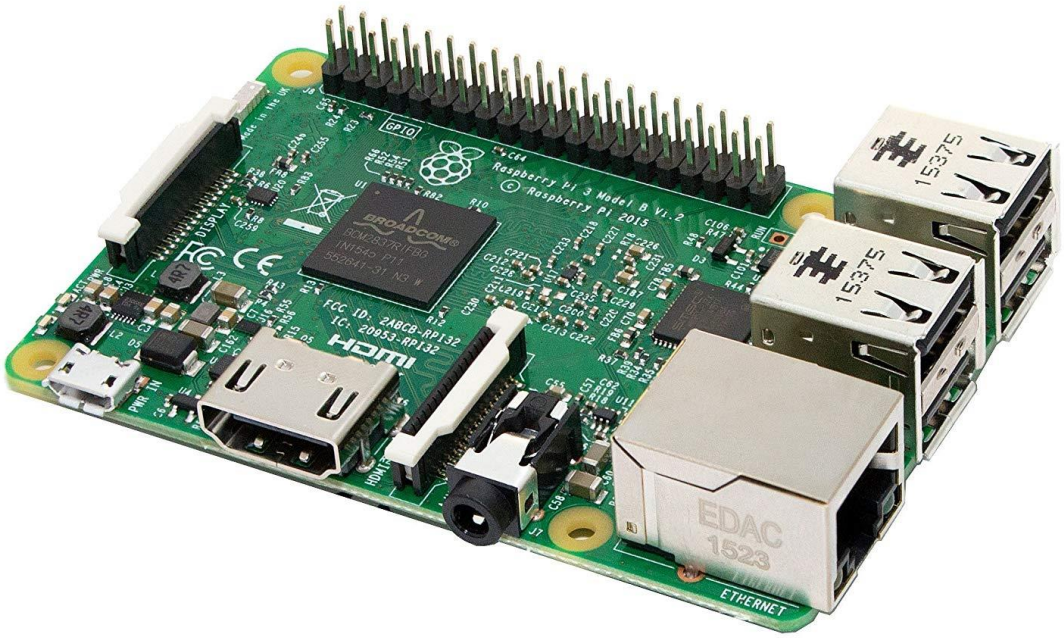

FIG. 3 – RASPBERRY PI 3 MODEL B V1.2

3.2 IoBroker

FIG. 4 - LOGO DE IOBROKER.

IoBroker es una plataforma de integración *Open Source* para la automatización de sistemas IoT; centrada especialmente en la domótica y desarrollada en JavaScript. IoBroker no es únicamente una aplicación sino más bien un esquema conceptual que de forma sencilla permite la interoperabilidad entre distintos sistemas.

IoBroker consta de una estructura modular. Cada instancia de un adaptador se ejecuta con un proceso propio, que se comunica con el controlador IoBroker. Esta modularidad permite que IoBroker se ejecute en varios hosts para dividir la carga de trabajo o conectar un host directamente al hardware.

En IoBroker las variables se intercambian entre adaptadores a través de los denominados puntos de datos. Utilizando scripts los puntos de datos pueden variar en función de los eventos. Los scripts pueden crearse tanto gráficamente como a través de JavaScript y es posible incluir sensores para los cuales no existe adaptador porque IoBroker posibilita la integración de módulos externos a Node.js.

La visualización de datos se realiza también mediante adaptadores, permitiendo crear una interfaz gráfica mediante métodos sencillos de arrastrar y soltar en el navegador, o utilizando herramientas más avanzadas si se dispone de conocimientos en HTML, CSS y JavaScript.

Hoy en día existen más de 200 adaptadores para IoBroker.(5)(6)

Resumiendo, las principales características de IoBroker son las siguientes:

- Está desarrollado íntegramente en JavaScript.
- Puede manejar el acceso a muchas bases de datos tales como: MS-SQL, MySQL, SQLite, InfluxDB, PostreSQL, y ficheros planos de texto.
- Arquitectura modular que posibilita su aplicación en cualquier dispositivo que soporte Node.js.
- Soporte rápido y sencillo para el protocolo MQTT.
- Una gran comunidad.
- El uso de adaptadores para la integración de las distintas aplicaciones.
- Integración nativa de NodeRed.
- La creación de un proceso Node.js por cada instanciación de un adaptador. Este hecho puede ser limitante si no se dispone de mucha memoria RAM.

En IObroker se han desplegado diversos adaptadores para facilitar la interconectividad entre los módulos de la solución. Los más importantes son:

- Node-red. Para desplegar el código que dirige el procesamiento de datos dentro de la raspberry.
- MQTT Broker/Client. Para las comunicaciones entre la raspberry y el ESP8266.
- SQL History. Para poder acceder desde el código desplegado en node-red a la información contenida dentro de la base de datos.

3.2.1 Configuración de los adaptadores

Algunos de los adaptadores utilizados deben configurarse de la siguiente manera:

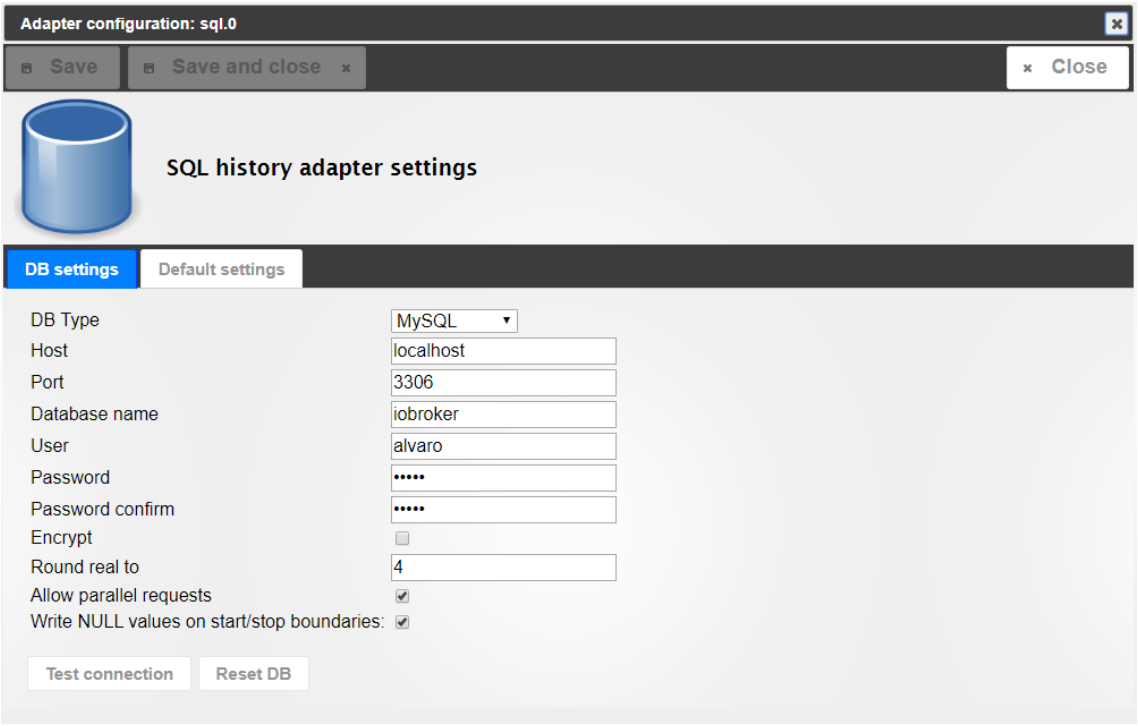

FIG. 5 - SQL HISTORY CONFIGURACIÓN DEL ADAPTADOR.

Importante indicar el tipo de base de datos, el nombre de la base de datos a utilizar, puesto que podríamos tener varias, así como el nombre de usuario y la contraseña indicados al instalar MySQL

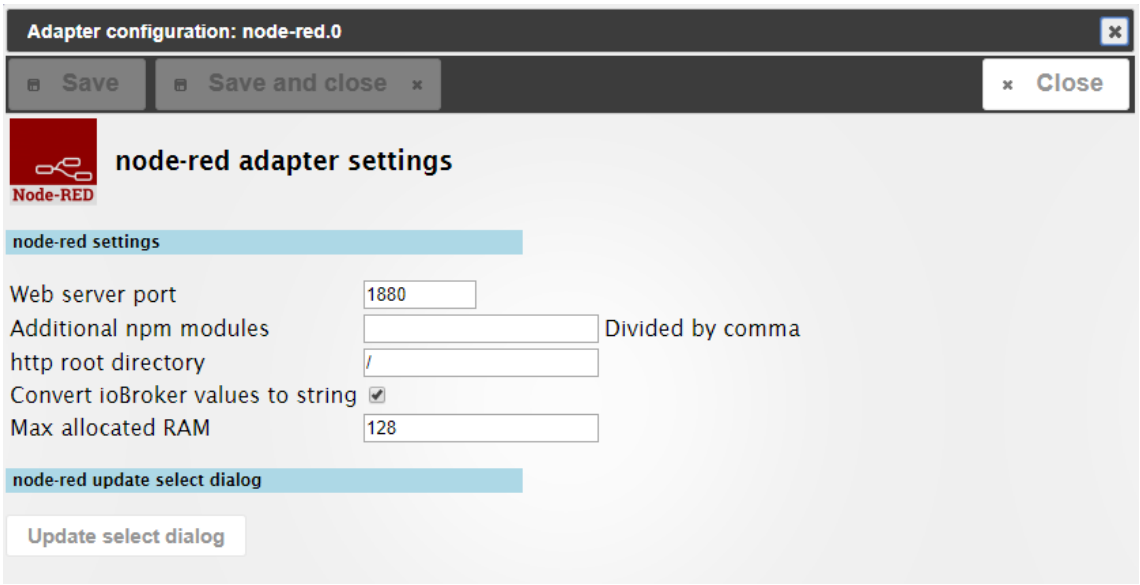

FIG. 6 - NODE-RED CONFIGURACIÓN DEL ADAPTADOR.

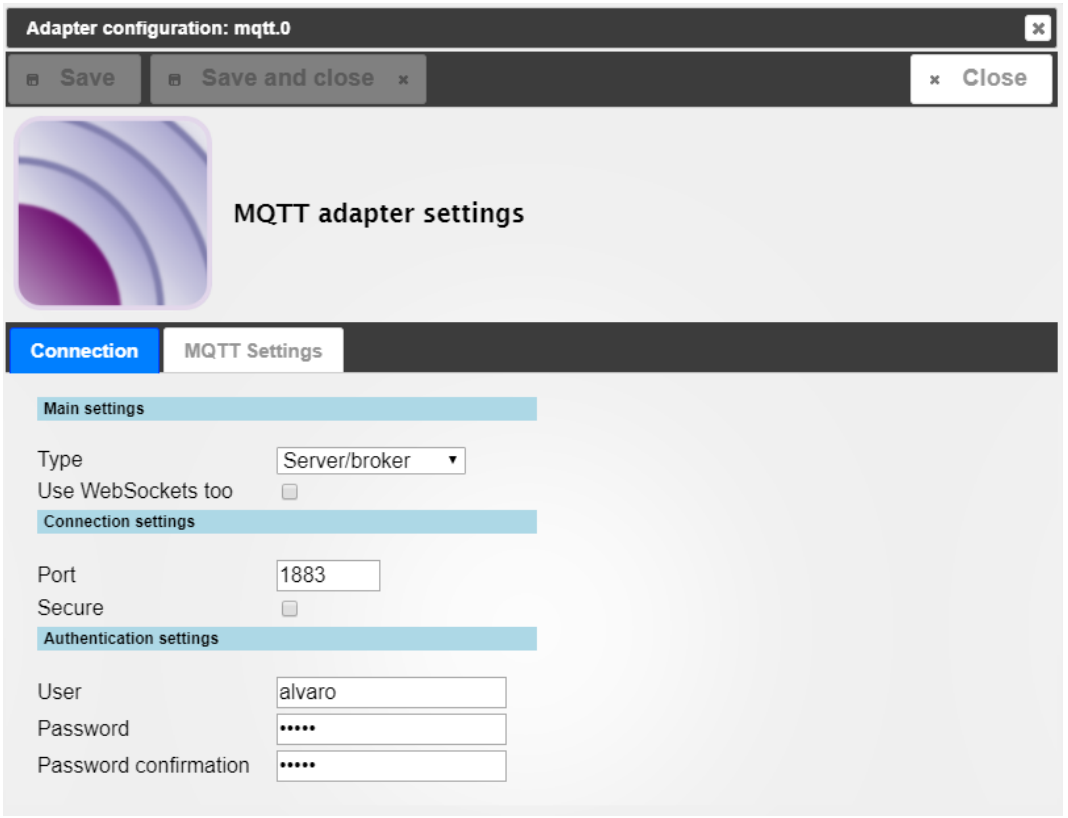

FIG. 7 - MQTT CONFIGURACIÓN DEL ADAPTADOR, CONEXIÓN.

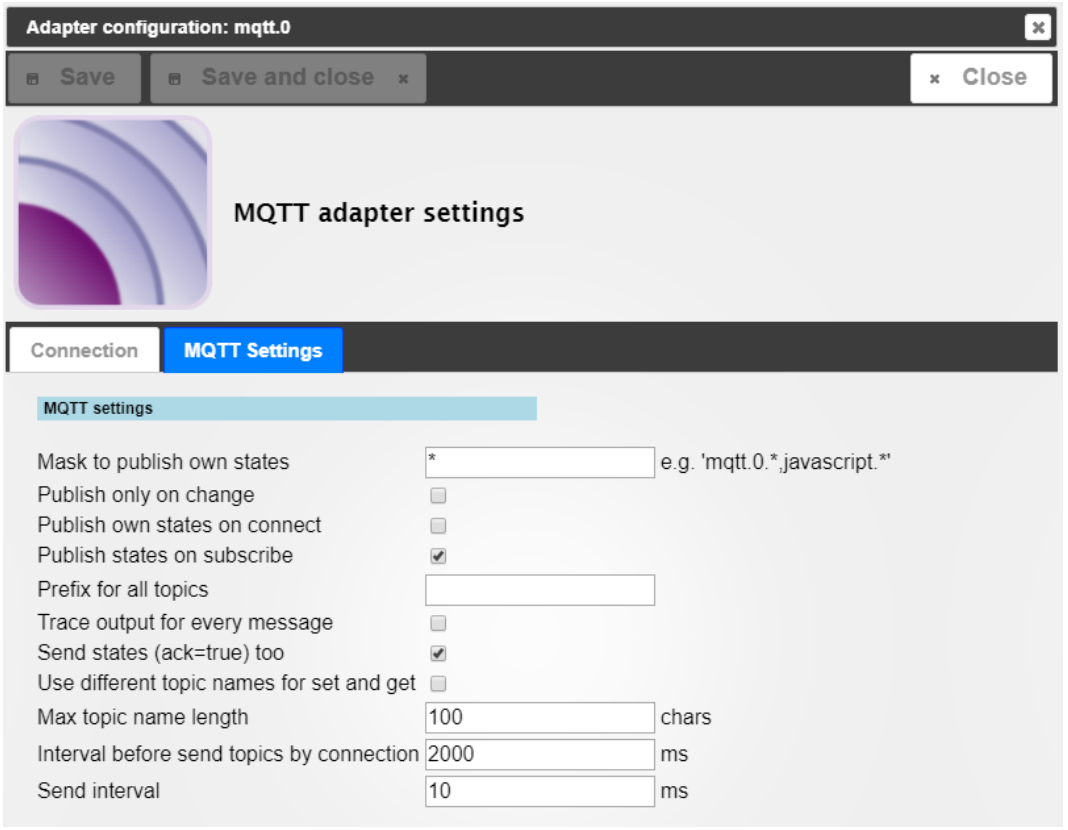

FIG. 8 - MQTT CONFIGURACIÓN DEL ADAPTADOR, OPCIONES MQTT.

3.3 Base de datos

Para almacenar la información medida se ha creado una base de datos dentro de la raspberry. Se ha elegido PHPMyAdmin por su facilidad de uso, gracias a su interfaz gráfica, y rapidez de instalación. Para poder utilizar PHPMyAdmin la raspberry debe contar con un servidor Web como Apache.

3.3.1 Instalación

En primer lugar, se recomienda actualizar el sistema mediante los comandos:

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

A continuación, se instala MySQL y se introduce una contraseña root:

```
sudo apt-get install mysql-server
```

Se instala Apache y PHP:

```
sudo apt-get install apache2 php5 libapache2-mod-php5
```

Instalamos PHPMyAdmin:

```
sudo apt-get install phpmyadmin
```

Editamos el siguiente archivo:

```
Sudo nano /etc/apache2/apache2.conf
```

Incluyendo al final la línea:

```
Include /etc/phpMyAdmin/apache.conf
```

Reiniciamos el servidor apache:

```
sudo /etc/init.d/apache2 restart
```

3.3.2 Utilización.

Conectando a la dirección de la raspberry/phpmyadmin/ debe aparecer esta pantalla de inicio:

FIG. 9 - PÁGINA DE INICIO DE PHPMYADMIN.

Tras acceder con la contraseña antes definida se accede a la interfaz gráfica. Para este proyecto se ha creado una base de datos denominada iobroker la cual cuenta con las siguientes tablas:

- Calendario. 25 columnas y 7 filas. Cada fila contiene el nombre de un día de la semana y 24 booleanos correspondientes a las 24 horas del día. Su uso se explica con más detalle en el apartado calendario.
- Config. Una única fila con 7 columnas. Aquí se almacenan los datos de configuración del proyecto.
- Contadores. Una única fila de 3 columnas. Registros con los litros que han pasado por el contador desde la instalación. Los dos últimos pueden resetearse desde la *dashboard*.

- Raw_data. Aquí se va almacenando la información conforme llegan mensajes desde el ESP8266. Cada fila se corresponde con un mensaje con la fecha del mensaje, los litros acumulados, el caudal y el *epoch* del mensaje. A continuación, se muestra un ejemplo de esta tabla:

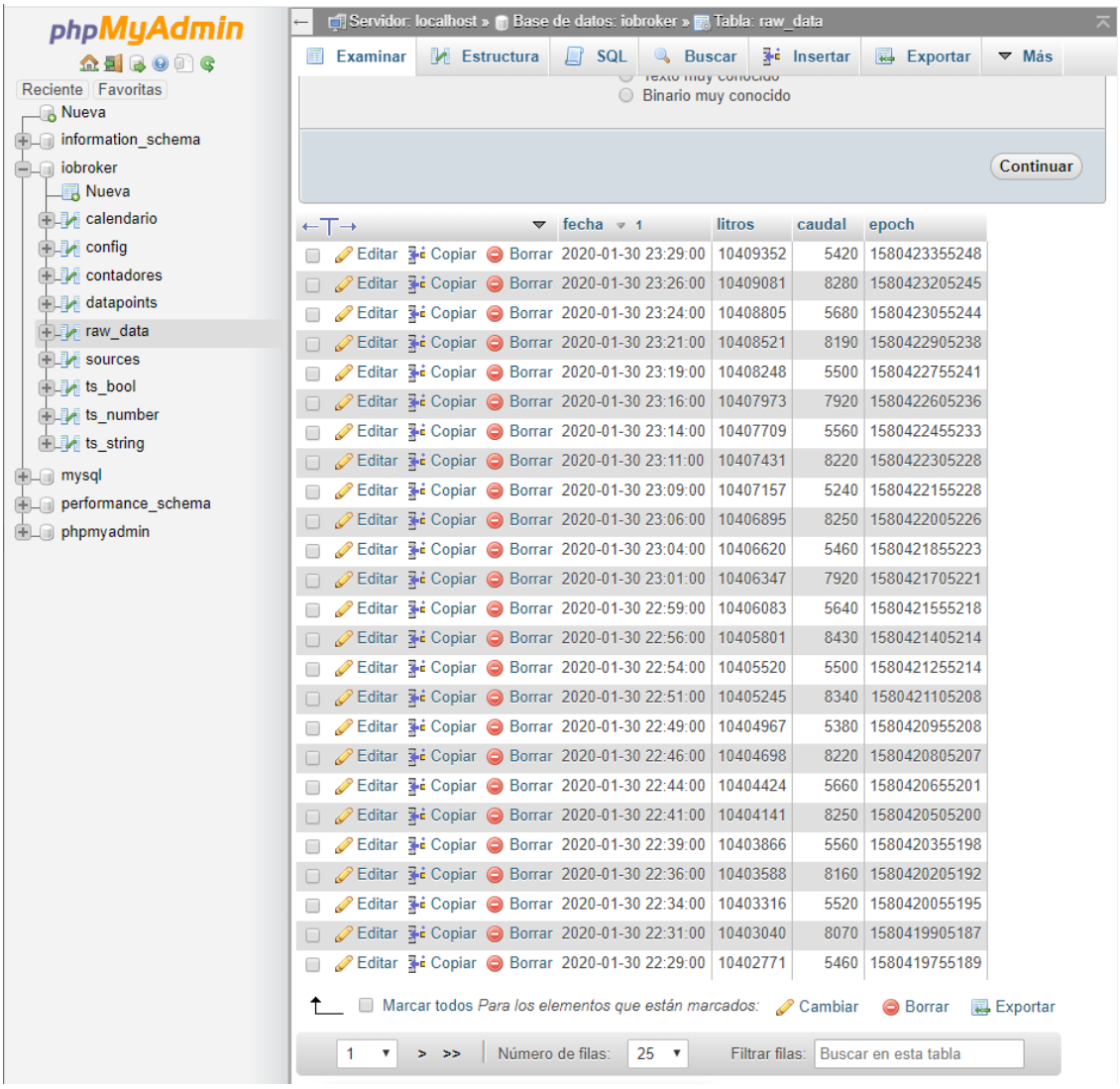

FIG. 10 - TABLA RAW_DATA.

PhpMyAdmin permite editar los valores de cualquier tabla, así como utilizar consultar escritas en SQL.

3.4 Comunicaciones MQTT

FIG. 11 - MQTT LOGO.

El intercambio de datos entre el ESP8266 y la raspberry se realiza mediante el protocolo MQTT. Para ellos se crea una lista de *topics*, estos conforman una estructura arborescente, lo que nos permite añadir más ramas y suscribirnos a varios *topics* al mismo tiempo simplificando la modularidad y el posible crecimiento de la solución si se añaden más topics o nuevos clientes que utilicen MQTT.

Para este proyecto los topics se organizan según la estructura de la imagen:

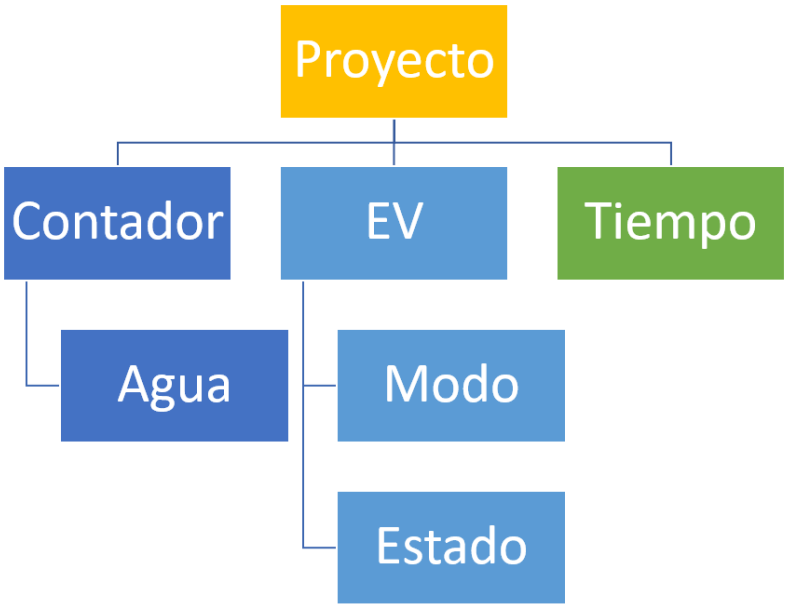

FIG. 12 - MQTT TOPICS

Tendremos por tanto 4 topics: proyecto/contador/agua, proyecto/ev/estado, proyecto/ev/modo y proyecto/tiempo.

Mediante el topic proyecto/contador/agua obtendremos la lectura acumulada del contador del esp8266, con proyecto/ev/modo se configura el modo de funcionamiento de la electroválvula, con proyecto/ev/estado actuaremos sobre la electroválvula que regula el paso del agua y con proyecto/tiempo gestionamos la fecha y hora entre la raspberry y el esp8266.

Mediante esta estructura podemos suscribirnos rápidamente a todos los topics mediante el uso de wildcards suscribiéndonos a proyecto/* (lo cual facilita al cliente la

suscrición a todos los topics dependientes de proyecto) y podemos agregar nuevos topics dentro de una de las ramas ya existentes o creando nuevas ramas.

A la hora de establecer las comunicaciones se ha optado por una comunicación muy sencilla donde el esp8266 publica información y las raspberry Pi la lee. Con excepción del tiempo que es enviado desde la raspberry hacia el esp8266

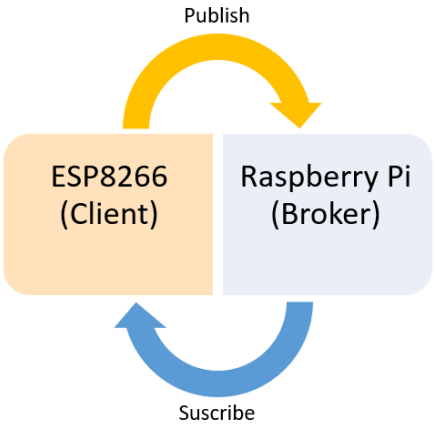

FIG. 13 - RED MQTT

En cuanto al nivel de QoS (Quality of service), la lectura de agua se ajusta a QoS 0, podríamos permitir perder algún dato dado que recibiríamos la información en el próximo mensaje, mientras que la actuación sobre la electroválvula y el envío del tiempo se mantienen en QoS 1, asegura que el mensaje llegará al menos una vez.

(7)

3.5 Contador de agua.

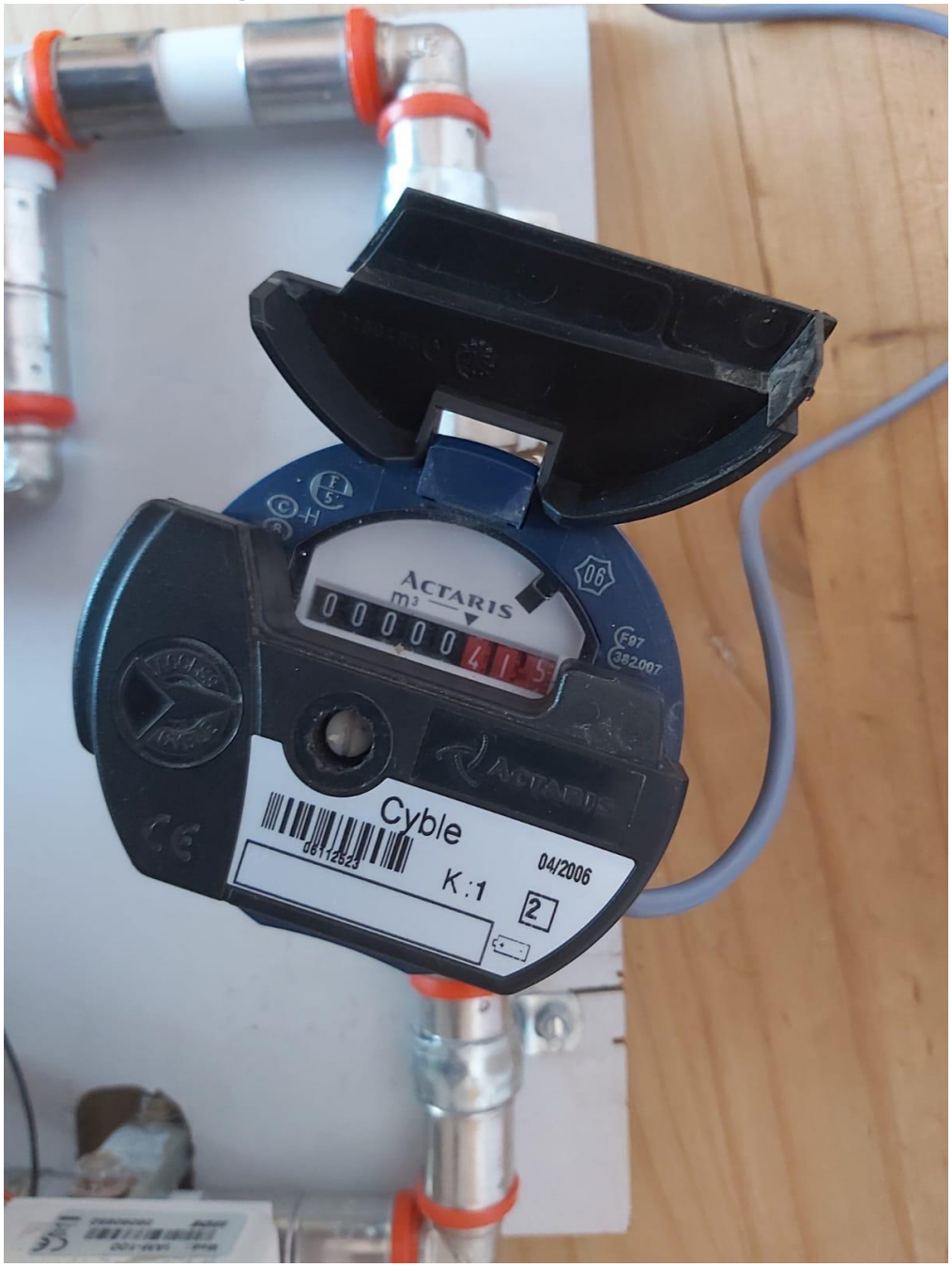

FIG. 14 - CONTADOR DE AGUA

Para registrar el paso de agua se dispone de un contador de agua Actaris. Se ha dispuesto una conexión de dos hilos mediante la cual no es necesario respetar la polaridad y cuya señal es equivalente a un contacto libre de potencial. Esta señal se registra mediante una interrupción del ESP8266. Otra cosa importante a tener en cuenta del contador es el factor K del módulo generador de pulsos que se define como el número de pulsos que equivalen a un litro. En el caso de estudio el contador tiene un factor K = 1.

3.6 Alimentación

De cara a mantener un bajo consumo y una alta autonomía del conjunto compuesto por el ESP8266 y la maqueta se ha utilizado una batería de litio y un módulo de carga de baterías junto con la turbina hidráulica para alimentar la placa y la válvula motorizada. Siendo la alimentación de esta última independiente, por lo que se podría utilizar una fuente distinta para accionar la llave de paso.

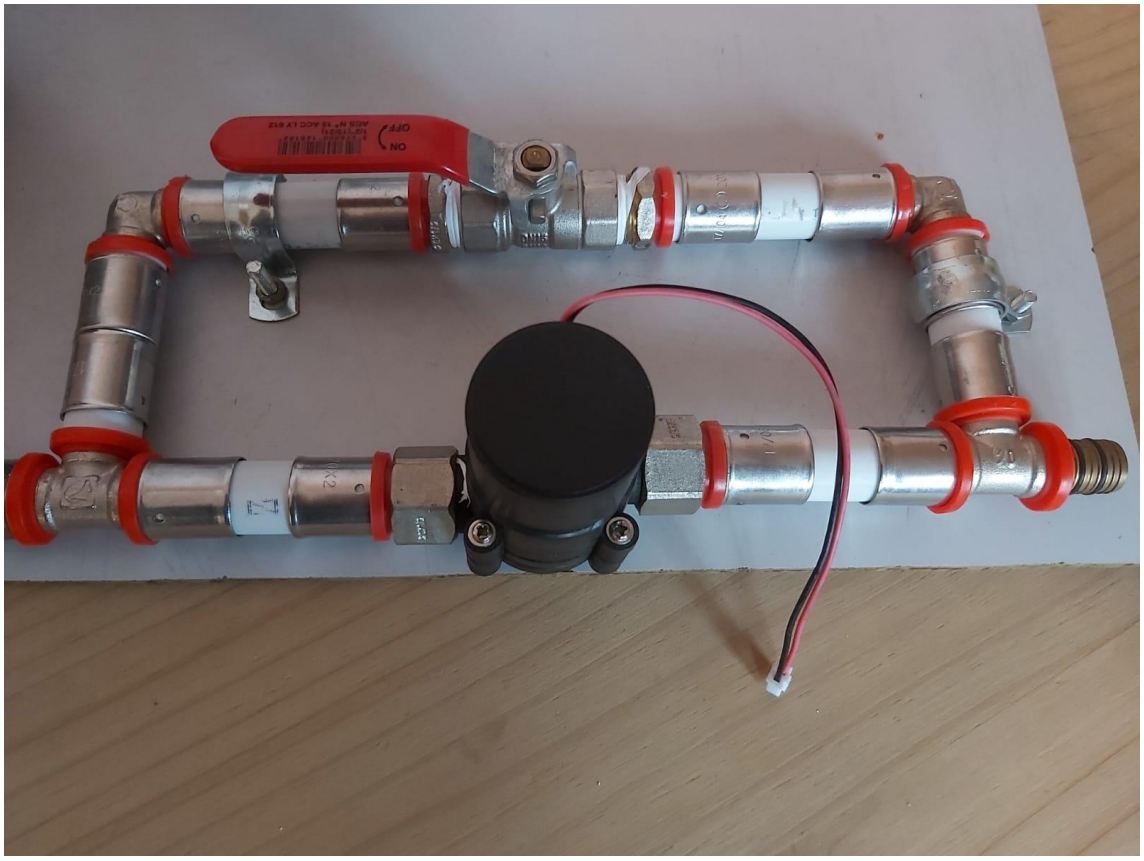

FIG. 15 – TURBINA

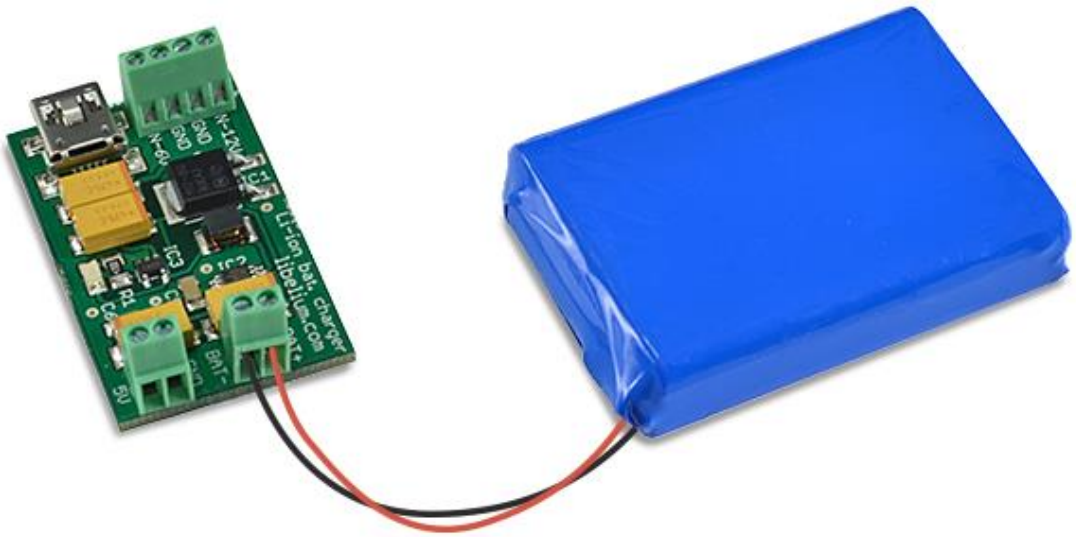

FIG. 16 - MÓDULO CARGADOR DE BATERÍAS Y BATERÍA.

El módulo de carga de baterías seleccionado dispone de una entrada para la turbina, un panel solar u otro elemento generador de energía que de un voltaje de hasta 12V. Un conector para la batería como se ve en la imagen y una salida a 5V de tensión continua. Además, también cuenta con un puerto USB.

A la hora de estimar la autonomía de la batería instalada se ha medido la intensidad nominal que consume la placa. Esta concuerda con las especificaciones del fabricante del ESP8266, alrededor de 140mA si se utiliza una conexión WiFi. Sin embargo, es posible reducir el consumo del microcontrolador si se utiliza alguno de los modos de bajo consumo. En el caso de utilizar el menos restrictivo de estos modos se puede llegar a unos 15mA de intensidad nominal mediante el modo de "*Modem-Sleep*". Con este modo se mantiene una conexión WiFi abierta sin transmisión de datos con un ciclo de 300ms de sueño y 3ms para recibir paquetes.

Si utilizamos una batería de 4400mAh con un consumo de 15mA por parte de la placa se lograría una autonomía aproximada de 293 horas, algo más de 12 días.

Para cargar la batería la turbina produce una corriente de entre 128 y 260 mA con un voltaje de salida de 12V DC y una pérdida de flujo de 0.12 m/s. La presión de trabajo del agua debe estar entre 80 y 550 kPa.

3.7 PCB

Para contener el esp8266 se diseña una PCB en la que se instalará tanto el microcontrolador como otros componentes necesarios para actuar sobre la electroválvula. El software elegido para el diseño fue DipTrace.

3.7.1 Esquemático

Para una mejor comprensión el esquemático ha sido dividido en varias partes según su función dentro de la placa las cuales serían: alimentación, programación, ESP8266, relés, configuración, salidas y tornillería.

3.7.1.1 Alimentación

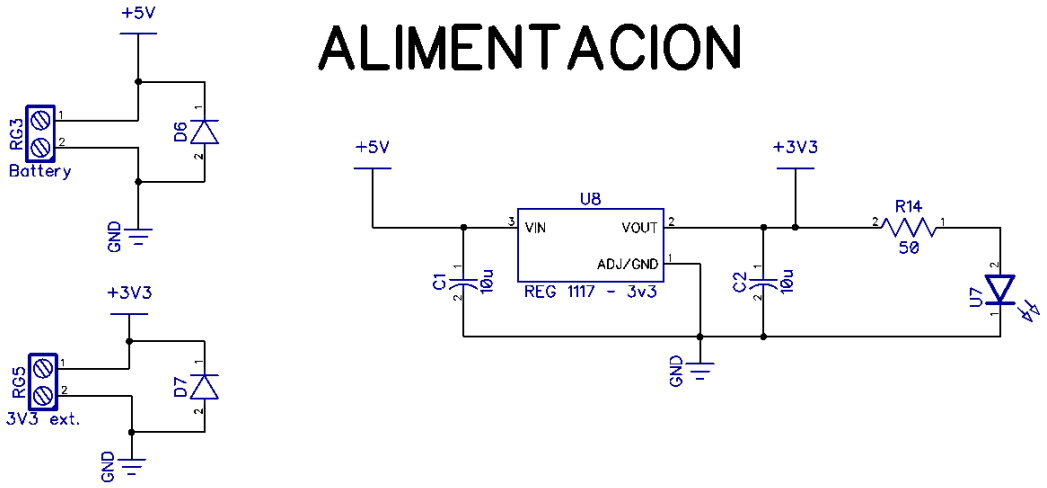

FIG. 17 - ESQUEMÁTICO-ALIMENTACIÓN.

La placa dispone de una entrada de 5V que viene del adaptador de carga de las baterías, así como de una entrada fija a 3.3V para pruebas. Un regulador lineal reduce los 5V a 3.3V dado que el esp8266 funciona a menor voltaje. Un led sirve de indicador para constatar que se la placa dispone de voltaje a través del regulador. Se han incluido diodos para proteger la placa de una conexión con polaridad inversa. Además de todo esto la placa puede alimentarse mediante el adaptador USB que puede verse en el siguiente apartado.

3.7.1.2 Programación

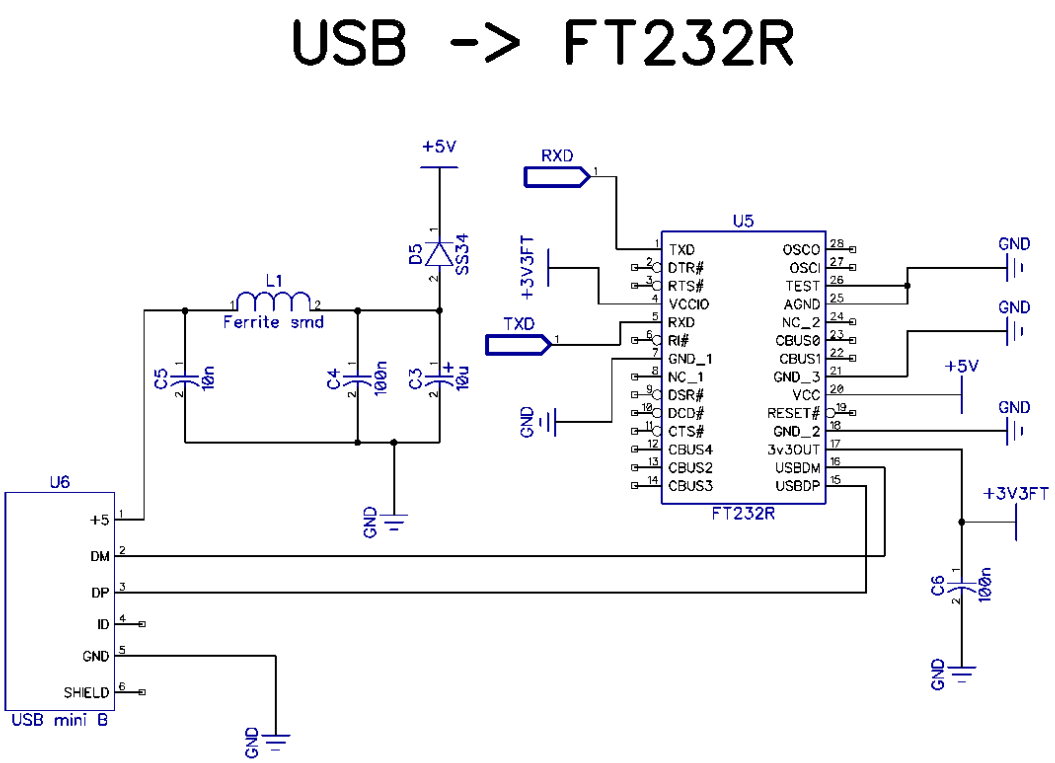

FIG. 18 - ESQUEMÁTICO-PROGRAMACIÓN.

Para programar el esp8266 se ha incluido un adaptador USB – serial UART, el FT232R. En la figura también se puede ver un USB mini B hembra que sirve para la programación, así como para alimentar la placa si fuese preciso. El resto de componentes sirven para filtrar el ruido que pueda ser producido por los componentes y llegar a través del cable USB hasta el host USB.

Las etiquetas RXD y TXD comunican con el esp8266.

3.7.1.3 ESP-8266

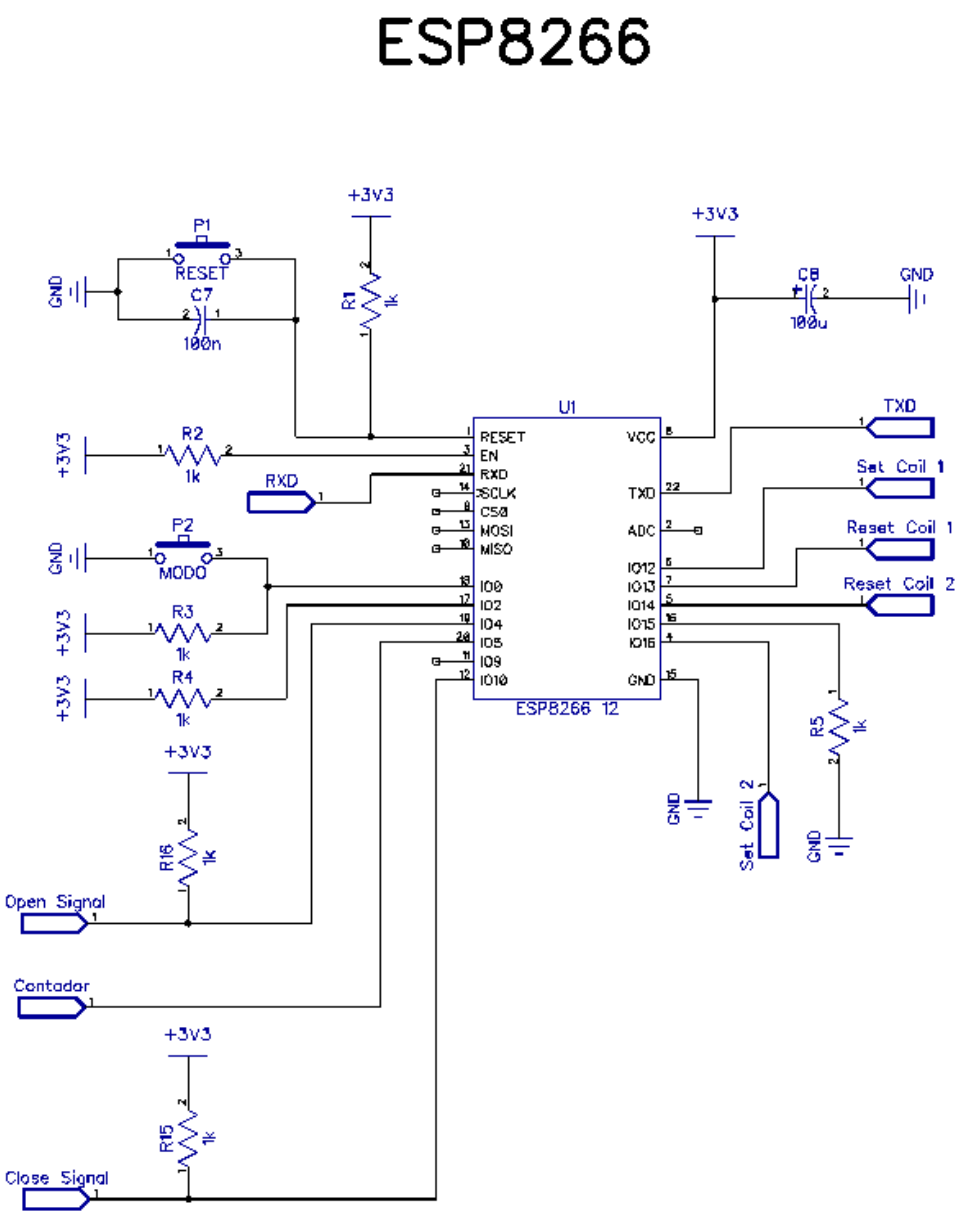

FIG. 19 - ESQUEMÁTICO-ESP8266.

El corazón de la placa. Para su normal funcionamiento es necesario poner algunas resistencias *pull-up* y *pull-down*. El pulsador P1 habilita el reseteo y el P2 pone en modo *flash* al integrado para su programación. Las etiquetas RXD y TXD llevan al FT232R. Las etiquetas *Open Signal* y *Close Signal* son entradas que provienen de la electroválvula en caso de que esta ofrezca esa posibilidad, la otra entrada proviene del contador. Las etiquetas restantes corresponden a las salidas para conmutar los relés descritos en el siguiente apartado.

3.7.1.4 Relés

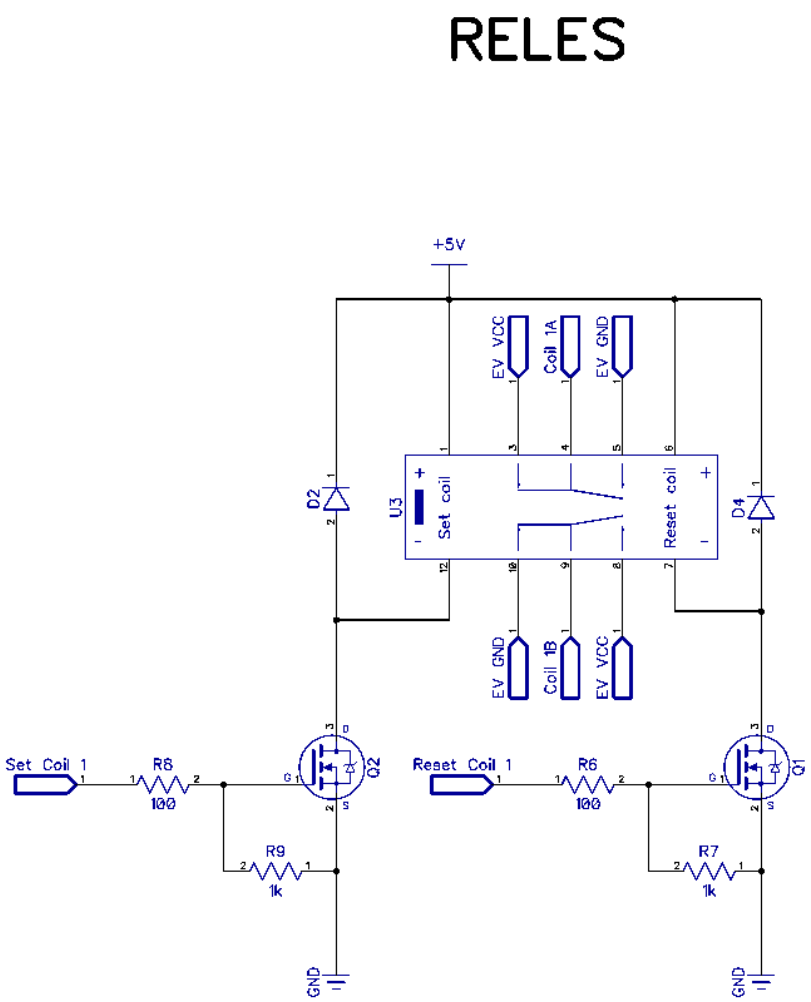

FIG. 20 - ESQUEMÁTICO-RELÉS1.

La placa consta de dos relés de tipo *latch*, una vez cambian de posición no es necesario mantener la corriente de excitación, las salidas del ESP8266 pasan por sendos transistores MOSFET para conmutar el relé. El primer relé se utiliza para algunas de las posibles configuraciones de electroválvula mientras que el segundo relé se utiliza para otras.

La electroválvula utiliza una entrada de voltaje distinta a las descritas en alimentación, esta es la marcada por las etiquetas EV VCC y EV GND.

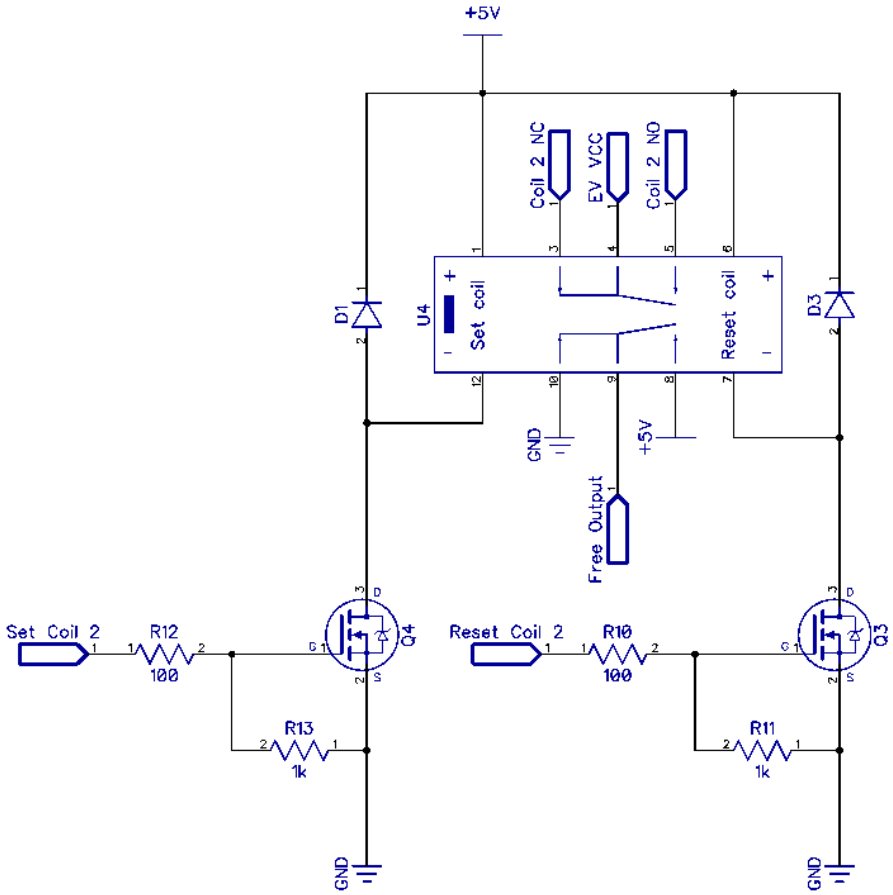

FIG. 21 - ESQUEMÁTICO-RELÉS2.

En el caso del segundo relé la disposición es casi idéntica al primero con la excepción de que al quedar una salida del relé libre se ha añadido una salida que podría ser utilizada con una finalidad distinta no contemplada en este proyecto (por ejemplo, activar una alarma sonora externa) en el caso de que se utilicé una configuración de electroválvula que no requiera el uso del segundo relé (por ejemplo, el modo1).

3.7.1.5 Configuración

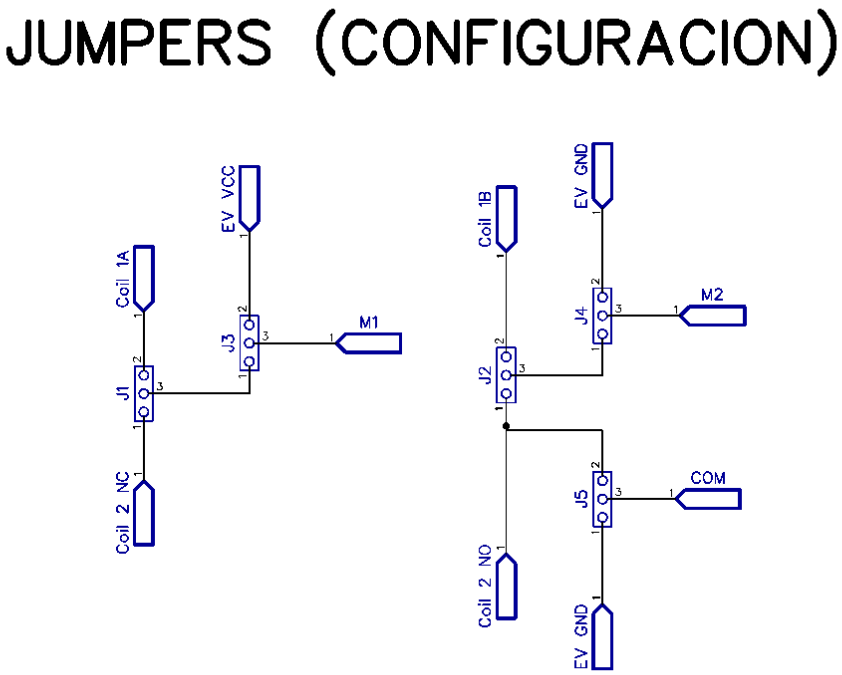

FIG. 22 - ESQUEMÁTICO-CONFIGURACIÓN.

La electroválvula es controlada mediante las salidas M1, M2 y COM. En función del modo de configuración hay 5 jumpers que deben colocarse para crear rutas desde las salidas de los relés Coil 1A, Coil 1B, Coil2 NC o E VCC y EV GND con M1, M2 y COM. Existe más información sobre esto en el apartado funcionamiento de la electroválvula.

3.7.1.6 Entradas/Salidas

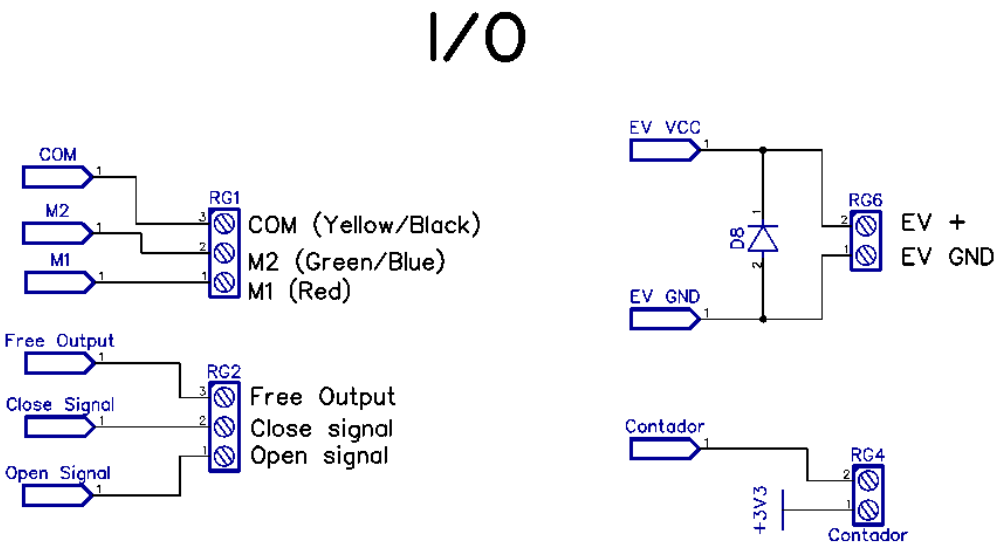

FIG. 23 - ESQUEMÁTICO-ENTRADAS/SALIDAS.

Como salidas la placa cuenta con M1, M2 y COM para controlar la electroválvula y una salida extra, *free output*. Como entradas está la entrada de pulsos del contador, la alimentación de la electroválvula, EV + y EV GND, y las señales de realimentación de estado de la electroválvula, *Close Signal y Open Signal*.

3.7.1.7 Tornillería

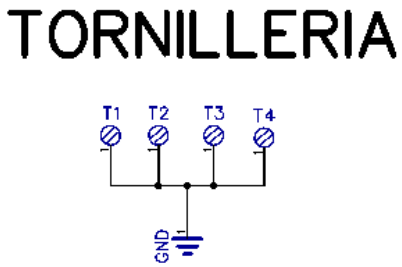

FIG. 24 - ESQUEMÁTICO-TORNILLERÍA.

Los cuatro tornillos de la placa se han unido a masa.

3.7.2 PCB Layout

Tras el diseño del esquemático llega la hora de situar los componentes sobre la placa. A la hora de abordar la disposición de los componentes se siguieron las siguientes directrices en orden de prioridad:

1. El USB debe situarse en el borde de la placa
2. Evitar que la antena WiFi del ESP8266 se encontrase encima de un plano de masa, por ello el microcontrolador se encuentra al borde de la placa.
3. Todos los conectores deben situarse en el borde de la placa.
4. Fácil acceso a los pulsadores P1 y P2.
5. Fácil acceso a los jumpers para reconfigurar el modo de la electroválvula.

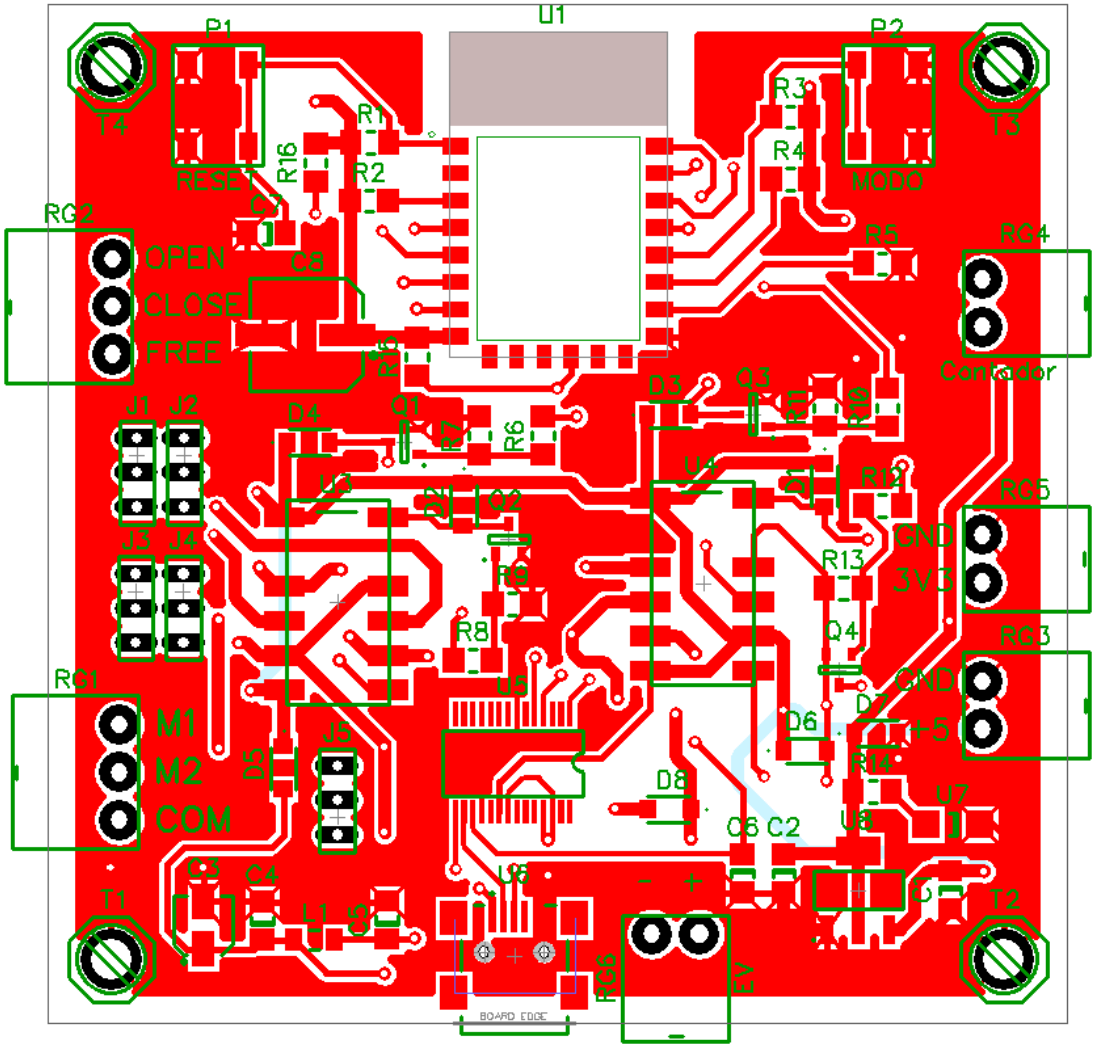

FIG. 25 - LAYOUT TOP.

Además, el proceso de ruteo se realizó varias veces intentando en cada iteración disminuir el tamaño del PCB. Se comenzó con una superficie de 100x100 mm y finalmente se ajustó hasta 75x75 mm.

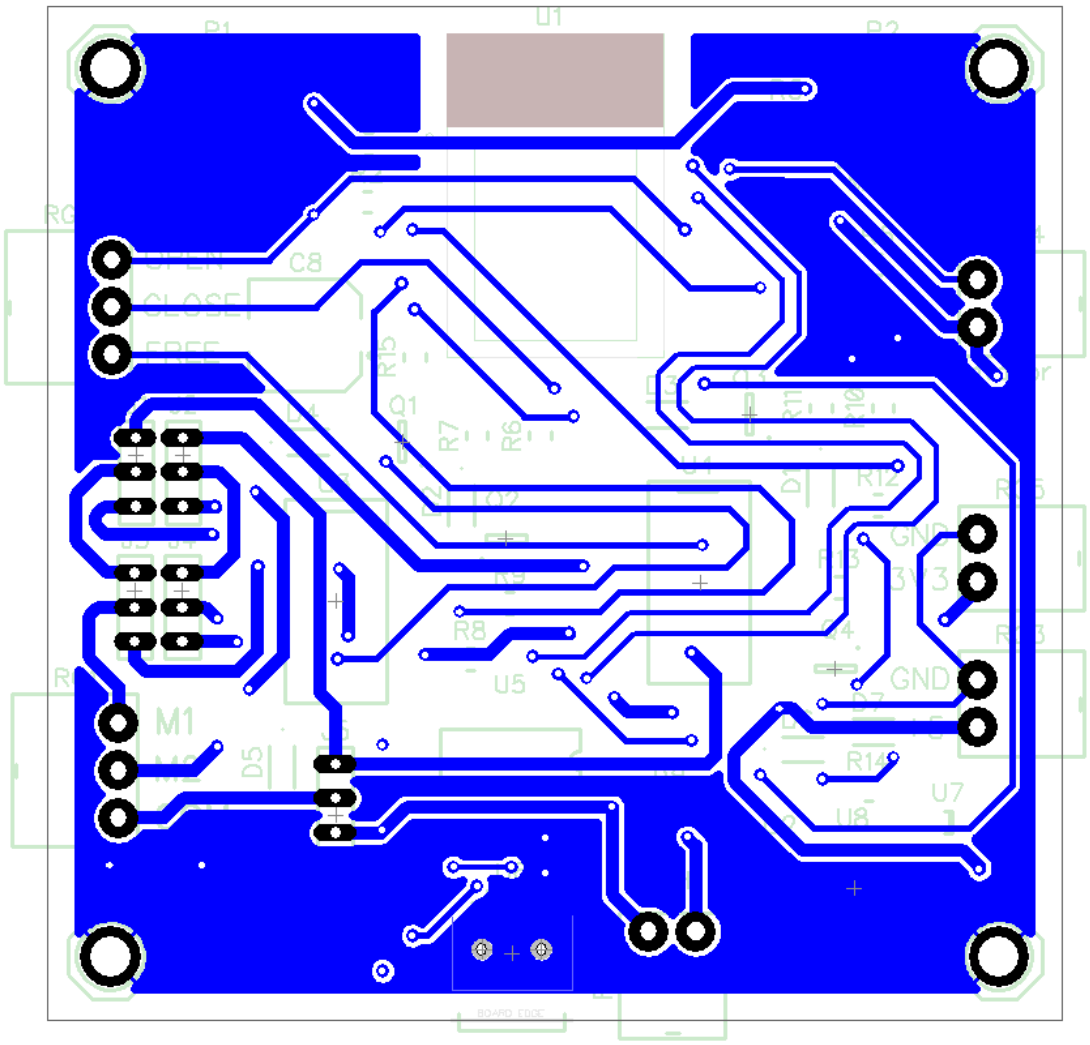

FIG. 26 - LAYOUT BOTTOM

El grosor de las pistas de 5V es sensiblemente mayor y existen planos de masa en ambas caras.

3.7.3 Modelo 3D

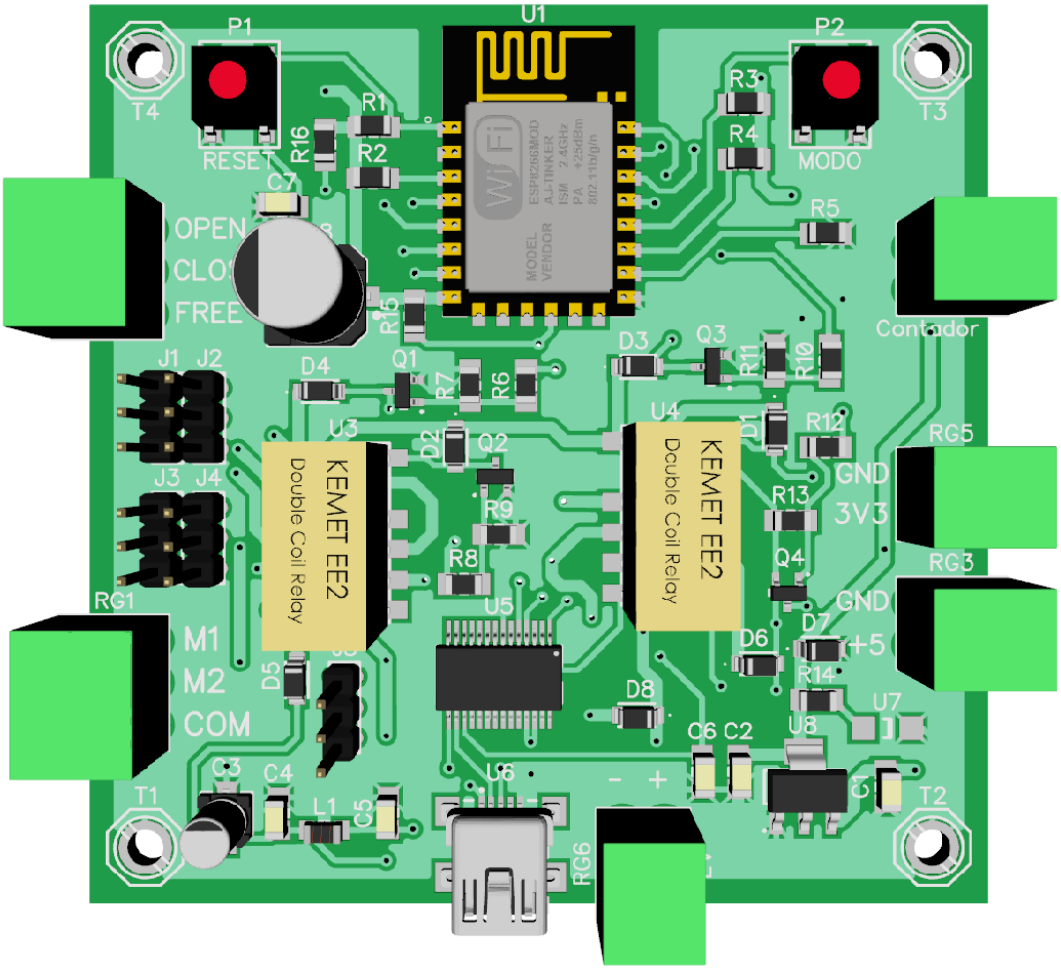

FIG. 27 - PCB 3D 1 FRONTAL.

Gracias a la herramienta de visualización 3D de DipTrace se puede comprobar el resultado final antes de mandar a producción las placas y comprar los componentes lo que puede ahorrar mucho dinero en caso de un error de diseño al no seleccionar el componente con el encapsulado correcto. También se puede ver si algún componente tapará la información serigrafiada importante como la que hace referencia a la polaridad de los conectores.

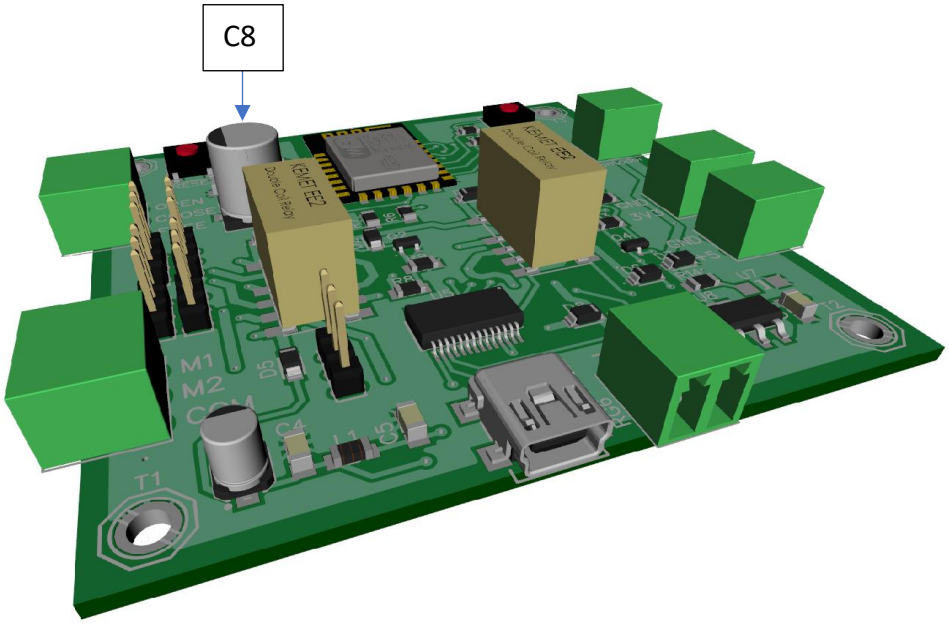

FIG. 28 - PCB 3D 2.

La mayor parte de los modelos han salido de los propios fabricantes, con la excepción de los conectores cuyo diseño se realizó con SolidWorks.

En nuestro caso los únicos componentes que muestran diferencias con los finales son los pulsadores, se eligió otro modelo de pulsador tras mandar la placa a producción y son algo más pequeños, y con el condensador C8, cuyo encapsulado es menor del elegido en fase de diseño. Aun así, no ha habido problemas para instalarlo en la placa.

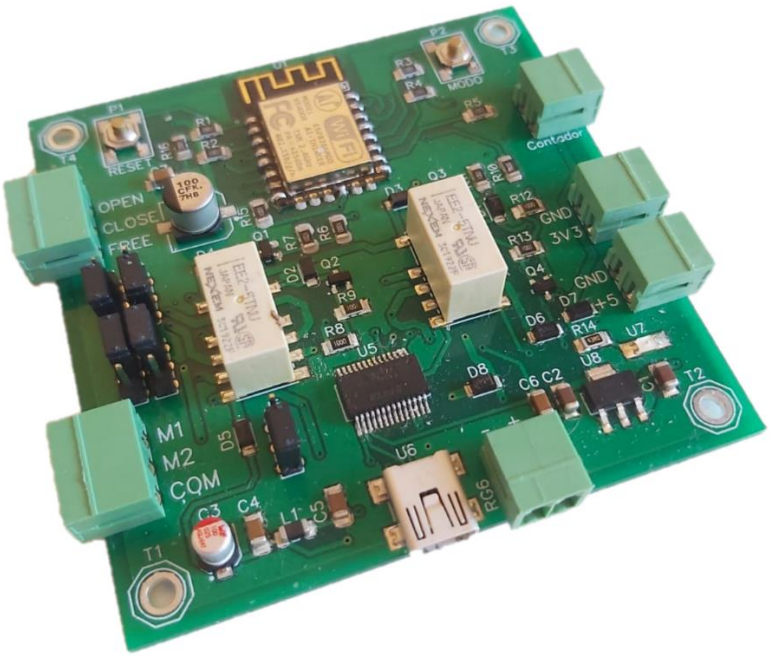

FIG. 29 - PCB FINAL.

3.7.4 Fabricación

A la hora de realizar el pedido de las placas se ha confiado en el fabricante PCBWay. Para la producción es necesario el envío de los *gerbers*, archivos de con la extensión .gbr exportados mediante DipTrace que contienen la información de cada una de las capas, así como un archivo .drl (*drill*) con la información de los taladros que deben realizarse.

A la hora de realizar el pedido es muy importante no olvidar pedir el *stencil*, una placa metálica que permite aplicar la pasta de soldadura de forma homogénea exactamente en el lugar requerido.

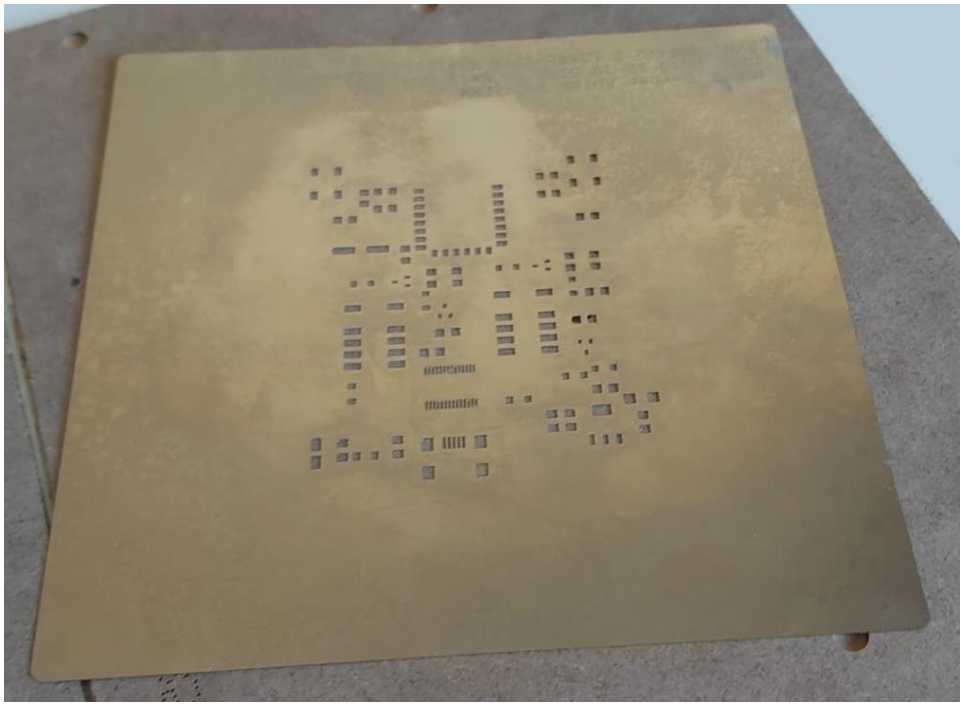

FIG. 30 - STENCIL.

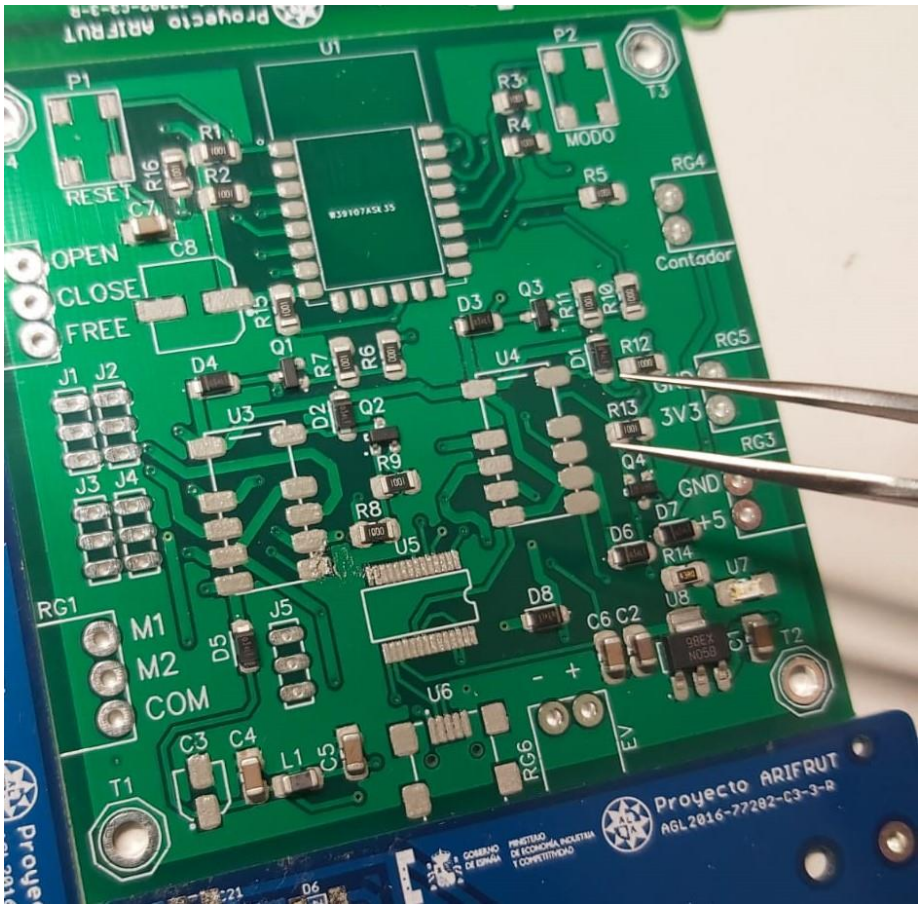

FIG. 31 - DETALLE DE MONTAJE SMD.

La mayor parte de los componentes son de montaje superficial y tras aplicar la pasta de soldadura estos deben ser colocados de forma minuciosa sobre la placa. Es importante seguir cierto orden para facilitar el trabajo y no mover un componente ya instalado al poner otro, en este caso se comenzó con los componentes más pequeños, transistores, resistencias y diodos, y se terminó colocando el ESP8266 y los relés.

Después de colocar todos los componentes se introduce la placa en un horno especial para soldadura SMD. Este horno calienta de forma progresiva la pasta térmica según un perfil de temperatura predeterminado. Al terminar el proceso es importante esperar hasta que la temperatura caiga de nuevo y no manipular la placa hasta que vuelva a temperatura ambiente.

Como algunos componentes son de agujero pasante, conectores y pines de configuración, se terminan de instalar estos mediante una estación de soldadura convencional.

Tras terminar con el montaje se debe realizar una exhaustiva comprobación del trabajo realizado. En primer lugar, se comprueba de forma visual si durante la soldadura se han unido pads entre sí, especialmente en el caso de encapsulados con muchas patillas muy cercanas entre sí como en nuestro caso el FT 232 RL, tras ello se comprueba la conductividad entre pads, no ya entre los pads que sí deben estar conectados entre sí sino sobre entre los que un cortocircuito podría provocar la rotura de algún componente, como por ejemplo entre las *nets* de 5V y GND.

Durante la fabricación del primer prototipo algunos pads del FT 232 RL se habían unido entre sí en el horno debido a un exceso de pasta de soldadura y en algunos pads del ESP8266 la soldadura había quedado fría y no había buena conductividad. Todos estos problemas se solventaron retirando parte del estaño y volviendo a aplicarlo con un soldador. Finalmente se determinó que no había más fallos y se alimentó la placa por primera vez tras lo cual se verificó que todo funcionaba correctamente.

3.8 Funcionamiento de la electroválvula

FIG. 32 - VÁLVULA MOTORIZADA.

Si bien en un principio se diseñó la solución para trabajar con una electroválvula con un funcionamiento concreto más adelante se decidió ampliar la funcionalidad de la placa para trabajar con electroválvulas con distintas filosofías de funcionamiento. Para ello se incluyeron unos pines que mediante jumpers permiten reconectar las salidas de la placa hacia la electroválvula con distintas salidas de los relés o entradas de la placa. Además, la programación dentro del ESP8266 varía según el modo de funcionamiento. A continuación, se explican los distintos modos de funcionamiento, así como la colocación de los jumpers para cada uno.

3.8.1 Modo 1. Dos cables y polaridad inversa.

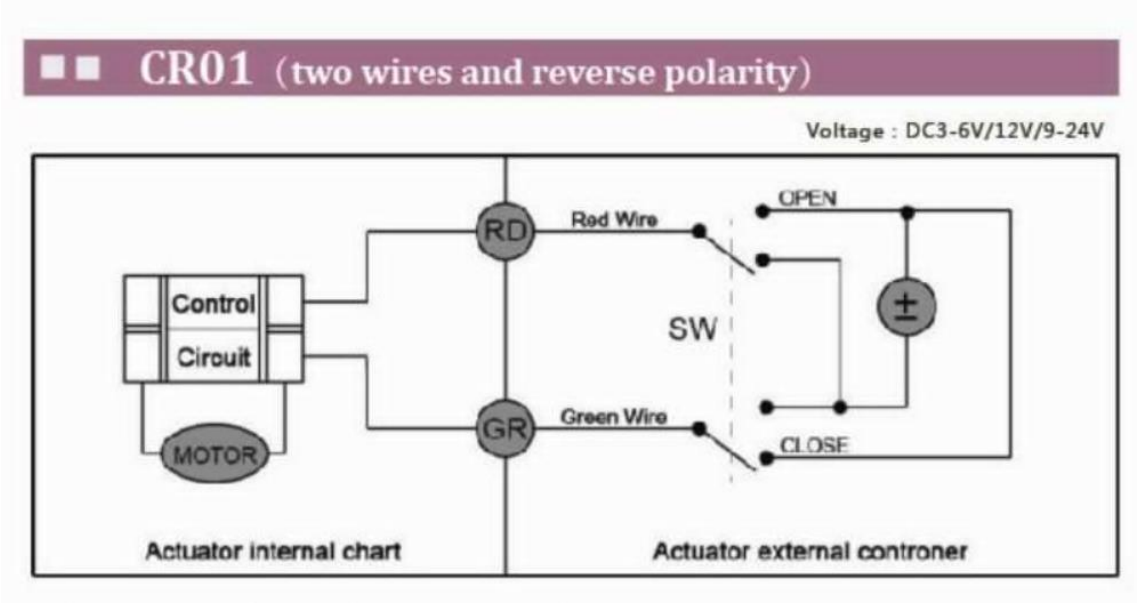

FIG. 33 - MODO EV 1

Conectando SW a OPEN la válvula se abre y conectando SW a CLOSE se cierra. La válvula mantiene su posición totalmente abierta o cerrada tras conmutar.

Lo que en el esquema se denomina cable rojo debe conectarse a la salida M1 de la placa mientras que el cable verde debe conectarse a M2.

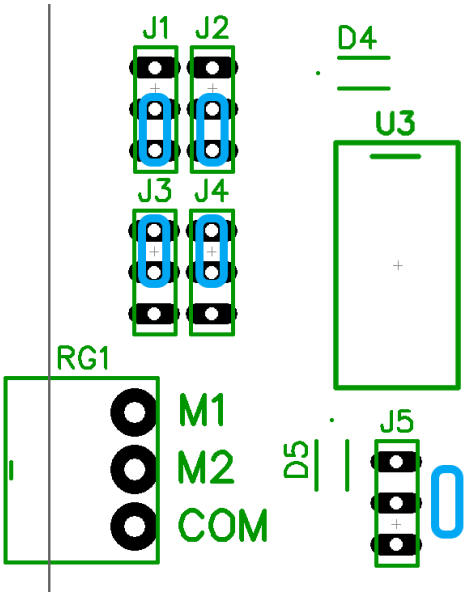

FIG. 34 - CONFIGURACIÓN EV 1

En la imagen se indica como han de colocarse los *jumpers* en J1, J2, J3 y J4 para el modo 1. En el caso de J5 es indiferente donde se coloque el jumper.

3.8.2 Modo 2. Tres cables y control a dos puntos.

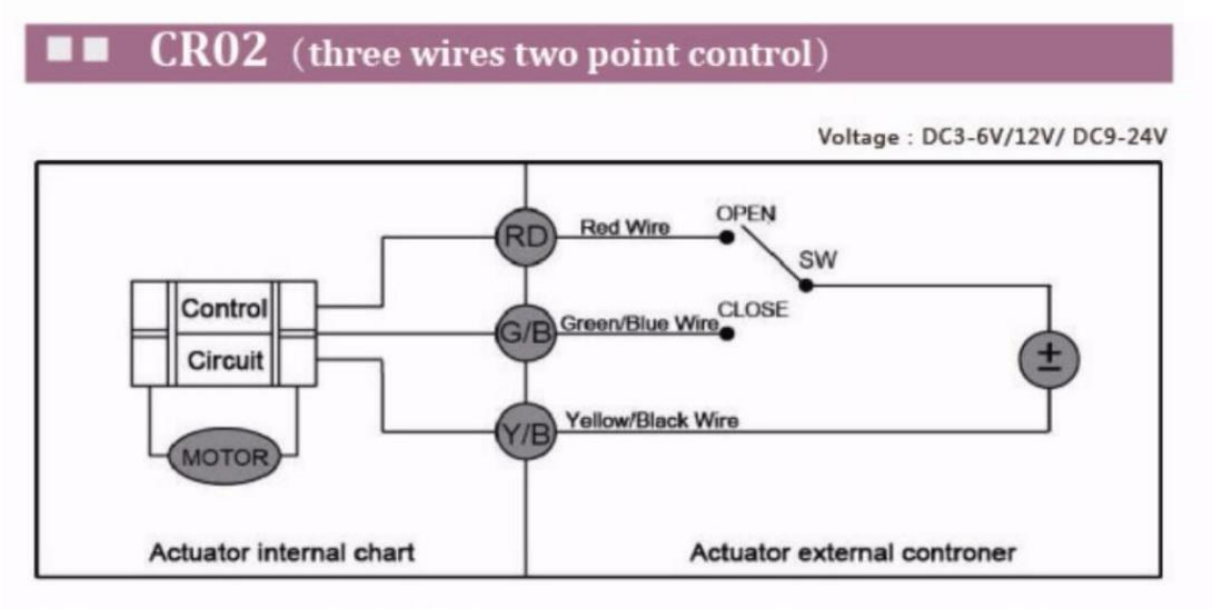

FIG. 35 - MODO EV 2

Conectando SW a OPEN la válvula se abre y conectando SW a CLOSE se cierra. La válvula mantiene su posición totalmente abierta o cerrada tras conmutar.

Lo que en el esquema se denomina cable rojo debe conectarse a la salida M1 de la placa, el cable verde/azul debe conectarse a M2 y el cable amarillo/negro a COM.

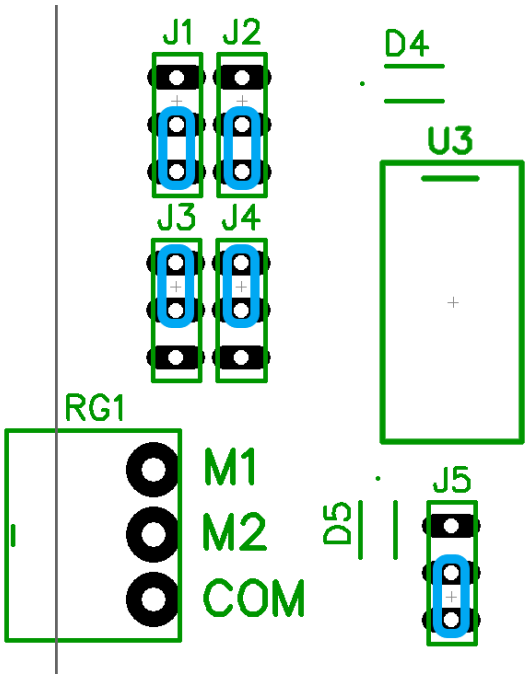

FIG. 36 - CONFIGURACIÓN EV 2

3.8.3 Modo 3. Tres cables y control a un punto.

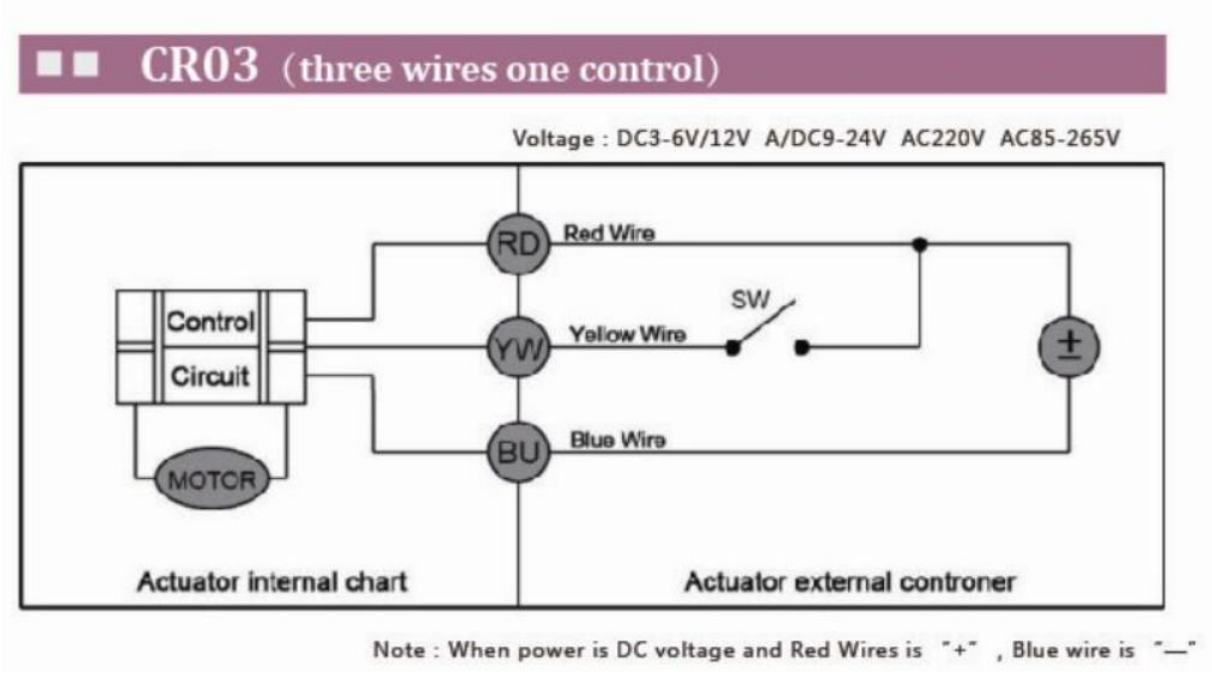

FIG. 37 - MODO EV 3

Conectando SW la válvula se abre y desconectando SW se cierra. La válvula mantiene su posición totalmente abierta o cerrada tras conmutar.

Lo que en el esquema se denomina cable rojo debe conectarse a la salida M1 de la placa, el cable azul debe conectarse a M2 y el cable amarillo a COM.

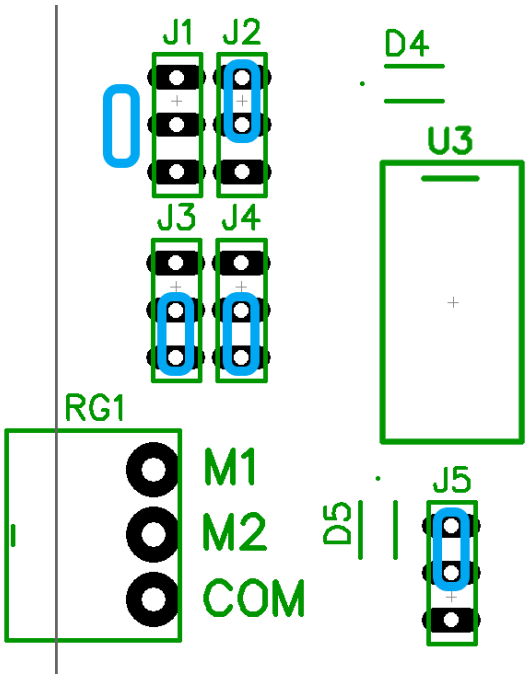

FIG. 38 - CONFIGURACIÓN EV 3

3.8.4 Modo 4. Dos cables y modo a prueba de fallos.

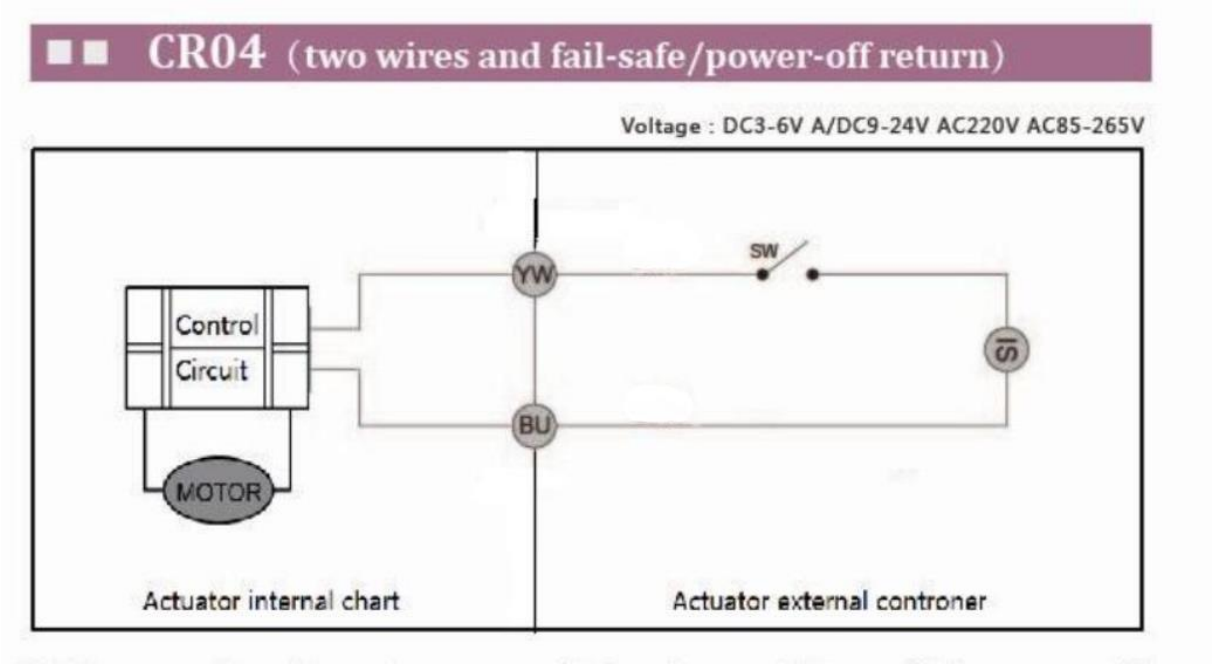

FIG. 39 - MODO EV 4

Conectando SW la válvula se abre y desconectando SW se cierra. La válvula mantiene su posición totalmente abierta o cerrada tras conmutar.

Lo que en el esquema se denomina cable rojo debe conectarse a la salida M1 de la placa, el cable azul debe conectarse a M2 y el cable amarillo a COM.

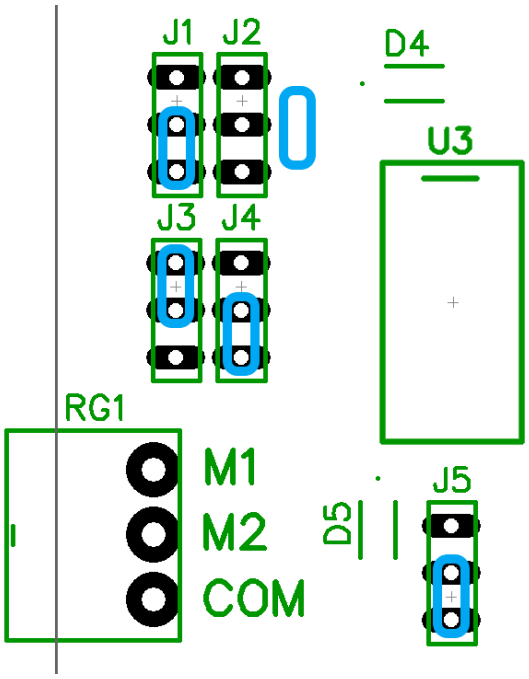

FIG. 40 - CONFIGURACIÓN EV 4

3.8.5 Modo 5. Cinco cables y señales de realimentación.

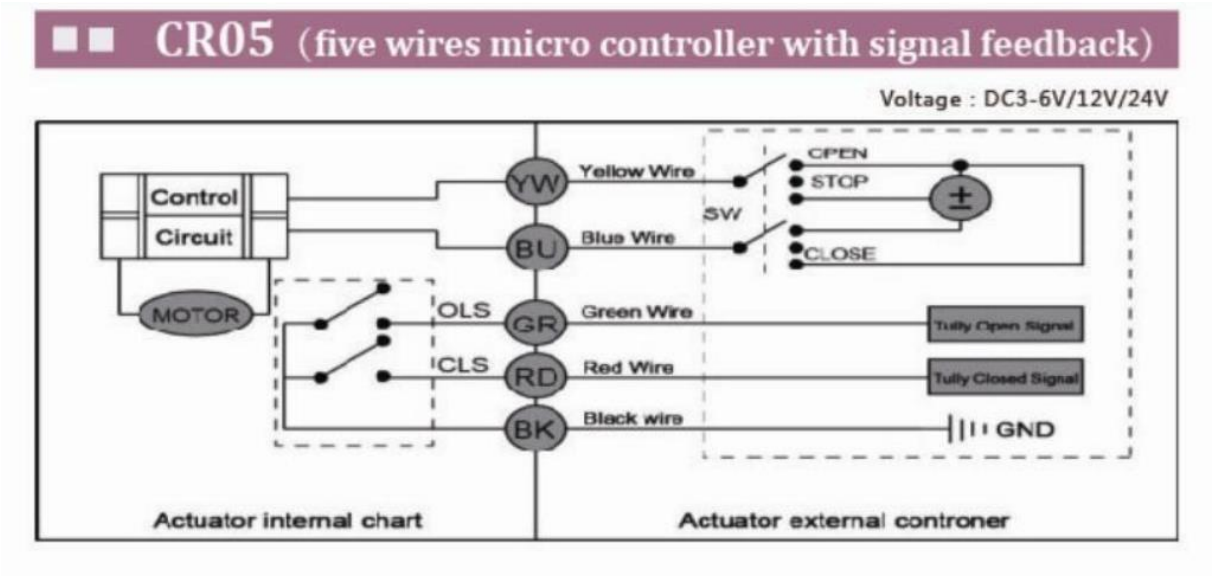

FIG. 41 - MODO EV 5

Conectando SW a OPEN la válvula se abre, tras completar el movimiento se activa la señal OLS pasando SW a STOP.
Conectando SW a CLOSE la válvula se cierra, tras completar el movimiento se activa la señal CLS pasando SW a STOP.

Lo que en el esquema se denomina cable amarillo debe conectarse a la salida M1 de la placa, el cable azul debe conectarse a M2, el cable negro a COM, el cable verde a la entrada OPEN y el rojo a la entrada CLOSE.

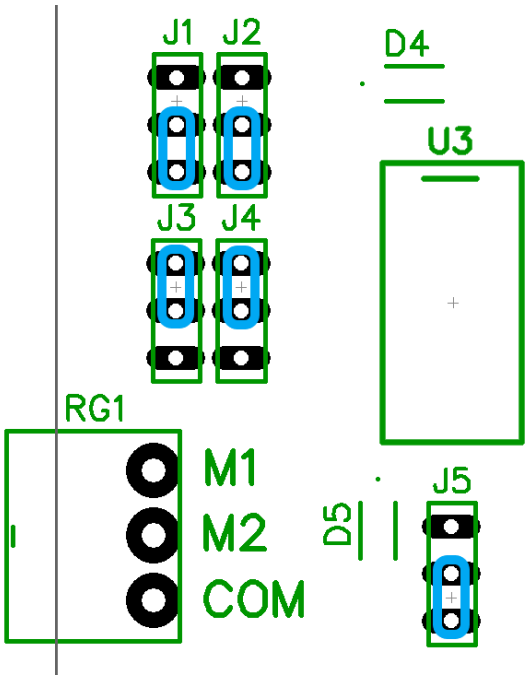

FIG. 42 - CONFIGURACIÓN EV 5

3.9 Programación ESP8266

El ESP8266 de este proyecto ha sido programado con el IDE de Arduino. Además de la versión 1.8.9 del entorno de programación de Arduino se utilizan las librerías:

- FS: Permite acceder a la memoria flash reservada para guardar archivos.
- ESP8266WiFi. Para establecer una conexión WiFi con el ESP8266.
- ArduinoJson: Para escribir y leer el archivo de configuración que se guarda en la memoria flash mediante SPIFFS.
- ESP8266WebServer: Habilita al esp8266 para utilizarlo como servidor web.
- DNSServer: Servidor DNS para redirigir todas las solicitudes al portal de configuración.
- WiFiManager: Permite la configuración de la red y su modificación sin tener que descargar de nuevo software al esp8266.
- PubSubClient. Crea un cliente MQTT para Arduino. Es compatible con el ESP8266.
- Ticker: Facilita la creación de parpadeo en leds.

La función del ESP8266 será la de conectarse a Internet y mandar a intervalos fijos la información del consumo de agua. A su vez ejecutará las ordenes de abrir o cerrar la electroválvula.

A continuación, se expone el código y la explicación de su funcionamiento.

3.9.1 Librerías

```
#define DEBUG
#include <FS.h>
#include <ESP8266WiFi.h>
#include <ArduinoJson.h>
#include <ESP8266WebServer.h>
#include <DNSServer.h>
#include <WiFiManager.h>
#include <PubSubClient.h>
#include <Ticker.h>

#define I_PULSO 5
#define I_OPEN_SIGNAL 4
#define I_CLOSE_SIGNAL 10
#define CONFIG_PIN 0

#define O_SET_COIL1 12
#define O_RESET_COIL1 13
#define O_RESET_COIL2 14
#define O_SET_COIL2 16
#define BUILT_IN_LED 2
```

Se incluyen las librerías. Se define un modo DEBUG para pruebas y constantes para las entradas y salidas.

- I_PULSO: Entrada de pulsos del contador de agua.
- I_OPEN_SIGNAL: Entrada que indica si la electroválvula está completamente abierta.
- I_CLOSE_SIGNAL: Entrada que indica si la electroválvula está completamente cerrada.
- CONFIG_PIN: Reutilizar el pulsador de modo flash para poner el esp en modo AP (Access point).
- O_SET_COIL1: Activa el primer relé de tipo *latch* para que pase a posición *set*.
- O_RESET_COIL1: Activa el primer relé de tipo *latch* para que pase a posición *reset.*
- O_RESET_COIL2: Activa el segundo relé de tipo *latch* para que pase a posición *reset.*
- O_SET_COIL2: Activa el segundo relé de tipo *latch* para que pase a posición *set.*
- BUILT_IN_LED: Utiliza el led integrado en el esp8266 para indicar si nos encontramos en modo configuración de red o en el modo normal de funcionamiento. Si parpadea nos encontramos en modo configuración, mientras que si está apagado funciona en modo normal.

3.9.2 Variables

```
//Variables
const char* CONFIG_FILE = "/config.json";

//Indica si el ESP tenia credenciales WiFi guardadas de la última vez.
bool initialConfig = false;

//Configuración por defecto:
//Se pueden dejar estos campos en blanco ("").
char clientid [12] = "ESP12E";
char user [12] = "alvaro";
char pass [12] = "55555";
char mqtt_server [17] = "192.168.1.254";

//Para enviar mensajes con el caudal acumulado a intervalos.
unsigned long now = 0;
unsigned long lastMsg = 0;
unsigned int sendInterval = 150000; //2,5 minutos.
unsigned long lastLitros = 0;

//Envío del tiempo y la cantidad en el contador.
char msg[250];
unsigned int contador = 0;
byte message_buffer[200];
unsigned int hora;
unsigned int minuto;

//EV
unsigned int evmodo = 1; //Del 1 al 5. Por defecto utiliza el modo 1.
boolean timerFlag = false;
unsigned long timerMillis = 0;
unsigned long timerInterval = 5000;
bool evOpenedState = 0;
bool evClosedState = 0;
```

Se crean las siguientes variables:

- Una constante, **CONFIG_FILE**, para apuntar al archivo de configuración que en formato .json se guardará dentro de la memoria *flash*.
- Un booleano, **initialConfig**, para marcar si el ESP8266 tenía variables guardadas en inicios anteriores.
- Variables para la conexión MQTT, clientid como nombre dado al ESP8266 como cliente MQTT, user y pass para autentificarse frente al servidor MQTT y mqtt_server como la dirección ip del servidor MQTT dentro de la red (es decir la dirección de la raspberry), estas variables pueden dejarse en blanco puesto que se modifican en el modo configuración.
- Las variables **now**, **lastMsg**, **sendInterval** y **lastLitros** se utilizan para almacenar *timestamps* mediante el comando *millis()* y posteriormente realizar comparaciones para ejecutar ciertos comandos a intervalos fijos de tiempo.
- La variable **msg** contiene el mensaje a mandar hacia el servidor MQTT.
- En **contador** el número de pulsos que pasan por el contador. Se resetea a 0 tras mandar el mensaje.

- Con **message_buffer** almacenamos en un array de bytes la información recibida de los topics a los que se suscribe el ESP8266.
- Las variables **hora** y **minuto** se utilizan en el modo DEBUG.
- Para la actuación sobre la electroválvula tenemos las siguientes variables: **evmodo** para actuar de forma distinta según el modo de la electroválvula, tres variables para controlar que la salida del ESP8266 que acciona los relés solo se activa durante un intervalo de tiempo determinado (**timerFlag**,**timerMillis**,**timerInterval**) y dos booleanos, **evOpenedState** y **evClosedState**, que almacenan el estado de las entradas OPEN y CLOSE.

3.9.3 Instancias

```
//MQTT->
WiFiClient espClient;
PubSubClient client(espClient);

//Instanciamos ticker para el parpadeo->
Ticker ticker;
```

Se crea un objeto de la clase **WiFiClient**, y se utiliza para que la librería **PubSuclient** inicialice parcialmente una instancia. Estos pasos serán necesarios para más adelante utilizar las funcionalidades que ofrece esta librería para trabajar con MQTT.

También instanciamos un objeto **Ticker** para controlar el parpadeo del led incluido en el ESP8266 durante el modo configuración.

3.9.4 Funciones

```
void parpadeoLed(){
  // Cambiar de estado el LED del esp.
  byte estado = digitalRead(BUILT_IN_LED);
  digitalWrite(BUILT_IN_LED, !estado);
}
```

La función **parpadeoLed()** invierte el estado del led incluido en el ESP8266 durante el modo configuración.

```
void cuentaLitros() {
   contador += 1;
}
```

La función **cuentaLitros()** va acumulando los pulsos en contador y los pasa a la raspberry, esta utilizar el factor K configurado para realizar la conversión hacia la cantidad de litros.

```
bool writeConfigFile() {
  Serial.println("Guardando archivo de configuración.");
  DynamicJsonDocument jsonDocument(1024);
  JsonObject json = jsonDocument.to<JsonObject>();

  //Convierte a json los parámteros locales de configuración.
  json["clientid"] = clientid;
  json["user"] = user;
  json["pass"] = pass;
  json["mqtt_server"] = mqtt_server;

  // Abre el archivo para escribir
  File f = SPIFFS.open(CONFIG_FILE, "w");
  if (!f) {
    Serial.println("Fallo al abrir el archivo.");
    return false;
  }

  serializeJson(json,Serial);
  //Escribe en el archivo y luego lo cierra.
  serializeJson(json,f);
  f.close();

  Serial.println("\nEl archivo de configuración se ha guardado
correctamente.");
  return true;
}
```

La función **writeConfigFile()** guarda los parámetros de configuración mediante SPIFFS (Serial Peripheral Interface Flash File System) en la memoria flash. Gracias a esto podemos modificar el programa principal y volver a volcarlo en el ESP8266 sin perder estos datos que son retenidos en una parte de la memoria flash reservada para guardar archivos.

```
bool readConfigFile() {
  //Abre el archivo de configuración en modo lectura.
  File f = SPIFFS.open(CONFIG_FILE, "r");

  if (!f) {
    Serial.println("Archivo de configuración no encontrado.");
    return false;
  } else {
    size_t size = f.size();
    //Prepara un buffer para los datos leídos.
    std::unique_ptr<char[]> buf(new char[size]);

    //Lee y guarda los datos en el buffer.
    f.readBytes(buf.get(), size);
    //Cierra el archivo.
    f.close();

    DynamicJsonDocument jsonDocument(1024);
    auto json = deserializeJson(jsonDocument,buf.get());
    //JsonObject& json = jsonDocument.parseObject(buf.get());
    //Si no tenemos exito.
    if (json) {
      Serial.print("JSON deserializeJson() error, code:");
      Serial.println(json.c_str());
      return false;
    }
    serializeJson(jsonDocument,Serial);
    //json.printTo(Serial);

    // Parse all config file parameters, override
    // Escribe las variables del archivo de configuración sobre las
variables locales para su posterior uso en el programa.
    if (jsonDocument.containsKey("clientid")) {
      strcpy(clientid, jsonDocument["clientid"]);
    }
    if (jsonDocument.containsKey("user")) {
      strcpy(user, jsonDocument["user"]);
    }
    if (jsonDocument.containsKey("pass")) {
      strcpy(pass, jsonDocument["pass"]);
    }
    if (jsonDocument.containsKey("mqtt_server")) {
      strcpy(mqtt_server, jsonDocument["mqtt_server"]);
    }
  }
  Serial.println("\nArchivo de configuración correctamente
volcado.");
  return true;
}
```

Al igual que existe una función para escribir el archivo de configuración existe otra para leerlo y guardar su contenido en variables locales para su posterior uso en el programa. Esta es la función **readConfigFile()**.

```
void reconnect() {
  //Provoca un loop hasta que se establezca conexión MQTT con el
servidor.
  while (!client.connected()) {
    Serial.print("Intentando establecer conxión MQTT...");
    if (client.connect(clientid, user, pass)) {
      Serial.println("Conexión exitosa.");
      //Se suscribe a los topics requeridos.
#ifdef DEBUG
      client.subscribe("toESP8266");
#endif
      /*Con esta sencilla instrucción se suscribe a todos los topics
       del tipo proyecto/ */
      client.subscribe("proyecto/#");
    }
    else {
      Serial.print("fallo al conenctar, rc=");
      Serial.print(client.state());
      Serial.println(" intentandolo de nuevo en 5 segundos.");
      delay(5000);
    }
  }
}
```

A la hora de establecer conexión mediante MQTT entre el ESP8266 y la raspberry se utiliza la función **reconnect()**, genera un bucle e intenta la reconexión cada 5 segundos hasta tener éxito.

```
void callback(char* topic, byte* payload, unsigned int length) {
  //Convertimos el topic en un string.
  String topicString = String(topic);
  //Si estamos en modo DEBUG se muestra el mensaje que ha llegado.
#ifdef DEBUG
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (unsigned int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();
#endif

   //Convertimos el contenido del mensaje en un string.
    unsigned int j = 0;
    for (j = 0; j < length; j++) {
      message_buffer[j] = payload[j];
    }
    message_buffer[j] = '\0';
    const      char*      p_payload      =      reinterpret_cast<const
char*>(message_buffer);
    snprintf (msg, 200, "%c", p_payload[0]);
    String payloadString = String(p_payload);


  if (topicString.equals("proyecto/tiempo")) {
    for (unsigned int j = 0; j < length; j++) {
      message_buffer[j] = payload[j];
    }
    message_buffer[j] = '\0';
    const      char*      p_payload      =      reinterpret_cast<const
char*>(message_buffer);
    snprintf     (msg,     200,     "%c%c%c%c-%c%c-%c%c-%c%c-%c%c-%u",
p_payload[0],      p_payload[1],      p_payload[2],      p_payload[3],
p_payload[5],      p_payload[6],      p_payload[8],      p_payload[9],
p_payload[11],    p_payload[12],    p_payload[14],    p_payload[15],
contador);
  }
  else if (topicString.equals("proyecto/ev/modo")) {
    evmodo = atoi(p_payload);
    #ifdef DEBUG
      Serial.println("-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*");
        Serial.print("Modo de la elevtroválvula: ");
        Serial.println(evmodo);
        Serial.println("-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.");
    #endif
  }
```

```
else if (topicString.equals("proyecto/ev/estado")){
    if (payloadString == "true"){
    if (evmodo == 1){
      digitalWrite (O_SET_COIL1, HIGH);
      timerFlag = true;
      timerMillis = millis();
    }
    else if(evmodo == 2){
      digitalWrite (O_SET_COIL1, HIGH);
      timerFlag = true;
      timerMillis = millis();
    }
    else if(evmodo == 3){
      digitalWrite (O_SET_COIL2, HIGH);
      timerFlag = true;
      timerMillis = millis();
    }
    else if(evmodo == 4){
      digitalWrite (O_SET_COIL2, HIGH);
      timerFlag = true;
      timerMillis = millis();
    }
    else if(evmodo == 5){
      digitalWrite (O_SET_COIL1, HIGH);
    }
  }

  else if (payloadString == "false"){
    if (evmodo == 1){
      digitalWrite (O_RESET_COIL1, HIGH);
      timerFlag = true;
      timerMillis = millis();
    }
    else if(evmodo == 2){
      digitalWrite (O_RESET_COIL1, HIGH);
      timerFlag = true;
      timerMillis = millis();
    }
    else if(evmodo == 3){
      digitalWrite (O_RESET_COIL2, HIGH);
      timerFlag = true;
      timerMillis = millis();
    }
    else if(evmodo == 4){
      digitalWrite (O_RESET_COIL2, HIGH);
      timerFlag = true;
      timerMillis = millis();
    }
    else if(evmodo == 5){
      digitalWrite (O_RESET_COIL1, HIGH);
    }
  }
  }

  else {
    #ifdef DEBUG
    Serial.println("-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+");
    Serial.println("Topic no contemplado.");
    Serial.println("-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+");
    #endif
  }
}
```

La función **callback()** recibe la información de los topics a los que está suscrito el cliente. En caso de encontrarnos en modo DEBUG cada vez que recibe un mensaje lo muestra. En función del topic del mensaje el ESP8266 realiza una acción u otra:

- Si el topic es proyecto/tiempo recompone la información y le agrega la cantidad guardada en contador al mensaje. Cuando durante la ejecución del loop() se requiera mandar la cantidad almacenada en el contador se mandará este mensaje (msg) que incluye además de la cantidad de agua del contador una marca temporal que servirá para guardar la información en la base de datos.
- En el caso del topic proyecto/ev/estado se actúa sobre la electroválvula abriéndola o cerrándola según el valor del mensaje (0 o 1). Para operar la electroválvula se tendrá en cuenta el modo de funcionamiento de la electroválvula.
- En el caso de recibir un mensaje de un topic al que estamos suscritos, pero no es ninguno de los anteriores se muestra el mensaje: Topic no contemplado.

3.9.5 Setup

```
void setup() {
  Serial.begin(115200);
  Serial.println("\n Inicializando:");
  //El delay se ha añadido para poder entrar en modo configuración
aún
  //cuando el ESP tiene credenciales guardadas con las que puede
conectarse.
  delay(3000);
  //Entradas.
  pinMode(I_PULSO, INPUT_PULLUP);
  attachInterrupt(I_PULSO, cuentaLitros, FALLING);
  pinMode(I_OPEN_SIGNAL, INPUT_PULLUP);
  pinMode(I_CLOSE_SIGNAL, INPUT_PULLUP);

 pinMode(CONFIG_PIN, INPUT_PULLUP);

  //Salidas.
  pinMode (O_SET_COIL1, OUTPUT);
  digitalWrite (O_SET_COIL1, LOW);
  pinMode (O_RESET_COIL1, OUTPUT);
  digitalWrite (O_RESET_COIL1, LOW);
  pinMode (O_SET_COIL2, OUTPUT);
  digitalWrite (O_SET_COIL2, LOW);
  pinMode (O_RESET_COIL2, OUTPUT);
  digitalWrite (O_RESET_COIL2, LOW);

  //Pin del esp.
  pinMode(BUILT_IN_LED, OUTPUT);



  //Inicia el sistema de archivos SPIFFS.
  bool result = SPIFFS.begin();
  Serial.println("SPIFFS abierto: " + result);

  if (!readConfigFile()) {
    Serial.println("Error al cargar el archivo de configuración, se
utilizarán los valores por defecto.");
  }

  //WiFi.printDiag(Serial); //Comentada para evitar el mostrar la
contraseña WiFi por el puerto serie.
```

En el **setup()** comenzamos inicializando el puerto serie tras lo cual se indica el funcionamientos de los GPIO. Además, como se puede observar se establece una interrupción que activará la función **cuentaLitros()** cada vez que el GPIO I_PULSO haga un flanco descendente (señal proporcionada por él contador).

A continuación, se inicializa SPIFFS y se comprueba el archivo de configuración.

```
 if (WiFi.SSID() == "") {
    Serial.println("Sin credenciales de punto de acceso, preparando
el modo configuración.");
    initialConfig = true;
  } else {
    digitalWrite(BUILT_IN_LED, HIGH); //Se apaga el led al finalizar
la configuración.
    WiFi.mode(WIFI_STA); //Forzamos el modo STA para evitar entrar
en modo AP al volver a encender el ESP8266.
    unsigned long startedAt = millis();
    Serial.print("Tras esperar ");
    int connRes = WiFi.waitForConnectResult();
    float waited = (millis()- startedAt);
    Serial.print(waited/1000);
    Serial.print(" segundos en setup() el resultado de la conexión
es: ");
    Serial.println(connRes);
    //Iniciamos MQTT.
    client.setServer(mqtt_server, 1883);
    client.setCallback(callback);
  }

  if (WiFi.status()!=WL_CONNECTED){
    Serial.println("Fallo al conectar, terminando setup.");
  } else{
    Serial.print("IP local: ");
    Serial.println(WiFi.localIP());
  }
}
```

Se continúa comprobando si existen credenciales para una conexión WiFi. En caso de tenerlas se apaga el parpadeo del led del ESP y se inicia MQTT. En caso de tener un problema se muestra la información por el puerto serie.

3.9.6 Loop

```
void loop() {
  //Inicio del portal de configuración requerido?
  if ((digitalRead(CONFIG_PIN) == LOW) || (initialConfig)) {
    Serial.println("Portal de configuración requerido.");
    //Temporizador del led.
    ticker.attach(0.8, parpadeoLed);

    //Configuración de parámetros extra.
    //Tras la conexión, parameter.getValue() da el valor configurado.
    //Formato: <ID> <Placeholder text> <default value> <length>
<custom HTML> <label placement>

    WiFiManagerParameter p_clientid("clientid", "MQTT client id",
clientid, 12);
    WiFiManagerParameter p_user("user", "MQTT user", user, 12);
    WiFiManagerParameter p_pass("pass", "MQTT pass", pass, 12);
    WiFiManagerParameter p_mqtt_server("mqtt_server", "MQTT server
ip", mqtt_server, 17);

    //Iniciando WiFIManager
    WiFiManager wifiManager;

    //Parámetros:

    wifiManager.addParameter(&p_hint);
    wifiManager.addParameter(&p_clientid);
    wifiManager.addParameter(&p_user);
    wifiManager.addParameter(&p_pass);
    wifiManager.addParameter(&p_mqtt_server);

    //Timeout en segundos tras el cual se apaga el portal de
configuración.
    //Si no se especifica el dispositivo se mantendrá en modo
configuración
    //hasta ser apagado o reiniciado.
    wifiManager.setConfigPortalTimeout(600); //10 minutos

    //Commienza un access point (AP)
    //y se mantiene en un bucle bloqueado esperando los valores de
configuración.
    //Cuando el usuario deja el portal de configuración se continua.
    if (!wifiManager.startConfigPortal("ESP8266-TFG", "123456789"))
{ //Nombre del AP y contraseña para ingresar.
      Serial.println("Sin conexión WiFi.");
    } else {
      Serial.println("Conectado!!!:)");
    }
```

```
    //Sobreescribiendo los valores de las variables globales con los
valores de los formularios del portal.
    strcpy(clientid, p_clientid.getValue());
    strcpy(user, p_user.getValue());
    strcpy(pass, p_pass.getValue());
    strcpy(mqtt_server, p_mqtt_server.getValue());
    //Escribiendo el archivo JSON de configuración manteniendo los
nuevos parámetros para futuros inicios.
    writeConfigFile();
    //
    ticker.detach(); //Quia el parpadeo.

    ESP.reset(); //Reseteo para poder iniciar el webserver de nuevo
en futuros inicios
    //reseteando el dispositivio posibilita volver al modo de
configuración tras un apagado y encendido.
    delay(5000);
  }

  //Este es el código que se ejecuta si hemos salido del portal de
configuración.
  if (!client.connected()) {  //Intenta volver a realizar la conexión
MQTT.
    reconnect();
  }

  now = millis(); //toma el tiempo actual.

  //Reseteo automático de salidas del esp8266 tras pasar el tiempo
de timerInterval
  //para evitar mantener alguna salida del ESP encendida demasiado
tiempo.
  if (timerFlag == true) {
    if (now - timerMillis >= timerInterval) {
      digitalWrite (O_SET_COIL1, LOW);
      digitalWrite (O_RESET_COIL1, LOW);
      digitalWrite (O_SET_COIL2, LOW);
      digitalWrite (O_RESET_COIL2, LOW);
      timerFlag = false;
    }
  }

  if (now - lastMsg > sendInterval) { //Tiempo en milisegundos entre
mensajes. 2,5 minutos.
    lastMsg = now;

#ifdef DEBUG
    Serial.print("Publish message: ");
    Serial.println(msg);
#endif

    client.publish("proyecto/contador/agua",    msg);    //Manda    la
lectura del contador de agua junto con la marca de tiempo.
    contador = 0;
  }
  client.loop();
}
```

El programa ejecutado es muy sencillo. El primer bucle *if* solo se utiliza en caso de encontrarnos en modo configuración, inicia el parpadeo del led y todos los parámetros configurables que serán accesibles desde el portal de configuración, tras esto inicia el ESP8266 en modo AP. Tras la conexión sobrescribe los valores de las variables globales, guarda la información en el archivo de configuración, anula el parpadeo y resetea el ESP8266.

Si ya contase con credenciales WiFi o no se requiere del modo de configuración porque no se ha pulsado el botón de modo se inicia la conexión MQTT y si no se consigue un bucle bloquea el programa hasta su establecimiento. El resto del código sirve para resetear las salidas dirigidas a los relés tras un período de tiempo definido en timerInterval y mandar mediante **msg** en el topic proyecto/contador/agua a intervalos constantes (definidos por sendInterval). Al terminar se refresca la conexión MQTT mediante **client.loop().**

3.10 Node-RED

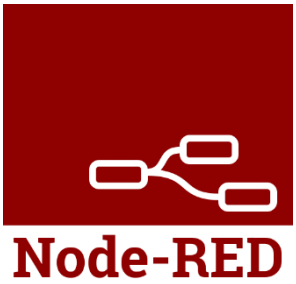

FIG. 43 – LOGO DE NODE-RED.

Node-RED es una herramienta de programación que permite interconectar dispositivos hardware, APIs y servicios online entre sí.

Además, proporciona un editor web que facilita en gran medida la conexión entre sistemas mediante el uso de "nodos". Node-RED incluye una amplia gama de estos nodos y además posibilita y anima a la creación de más nodos para extender sus funciones.

Se trata de un sistema construido sobre Node.js, un entorno de ejecución de JavaScript orientado a eventos asíncronos que evita el bloqueo del proceso a diferencia de los modelos de concurrencia basados en hilos.

(8)

3.10.1 Instalación

Puesto que vamos a utilizar node-red como un adaptador dentro de iobroker la instalación es muy sencilla ya que solo hay que instalar el adaptador específico dentro de iobroker.

Una vez hecho esto se accede al editor mediante: dirección ip de la raspberry:puerto del adaptador.

En nuestro caso 192.168.1.254:1880.

Para acceder a la dasboard: 192.168.1.254:1880/ui.

3.10.2 Estructura

El código del proyecto ha sido dividido en dos partes, lo que node-red denomina como *flows*. El primero tiene el nombre de Home Monitoring y funciona como el núcleo de la solución. El segundo se llama Dashboard y contiene todo el código que controla la visualización del *dashboard* mediante el teléfono móvil o el ordenador.

Node-red funciona mediante eventos asíncronos así que la división del código en uno o más *flows* sirve principalmente para facilitar la comprensión del programa y para poder desplegar y modificar partes del código sin tener que redesplegarlo todo de nuevo.

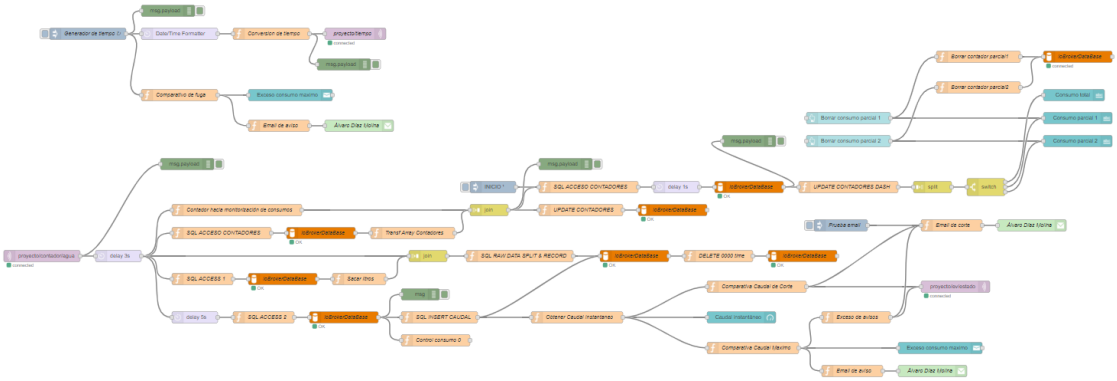

FIG. 44 - HOME MONITORING FLOW.

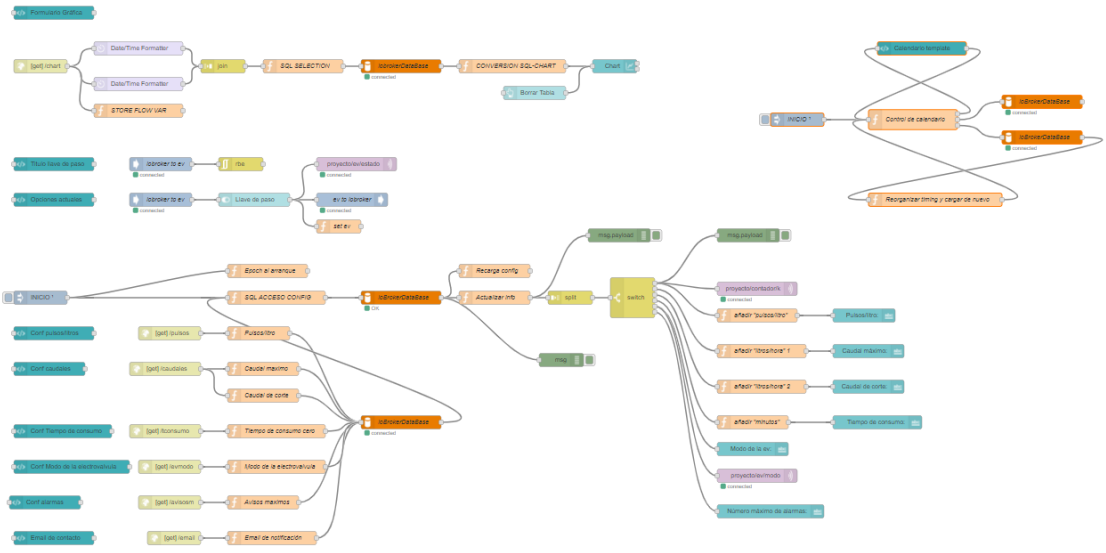

FIG. 45 - DASHBOARD FLOW.

3.10.3 Home Monitoring envío de tiempo

El *flow* Home Monitoring se puede dividir en dos partes principalmente:

La primera controla el envío de la fecha de la raspberry al ESP9266 y funciona de este modo:

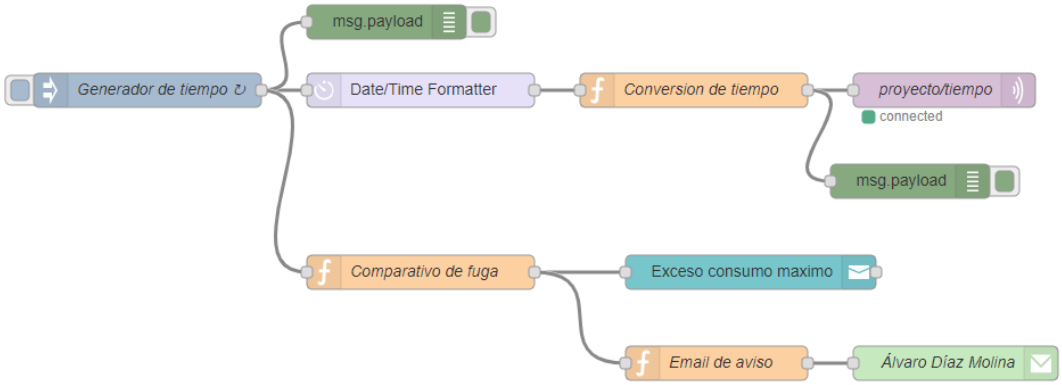

FIG. 46 - ENVÍO DE TIEMPO MEDIANTE NODE-RED.

El primer nodo se ejecuta automáticamente cada 10 segundos enviando como mensaje un *timestamp* con el tiempo actual. Por un lado, a este mensaje se le da un formato de fecha y hora que luego se transforma mediante el siguiente código:

```
var str = msg.payload;
msg.payload = 0;
var array = [];

array = str.split("");
var year = array[0] + array[1] + array[2] + array[3];
var month = array[5] + array[6];
var day = array[8] + array[9];
var hour = array[11] + array[12];
var minute = array[14] + array[15];
//var second = array[17] + array[18];

msg.payload = year + "-" + month + "-" + day + "-" + hour + "-" +
minute;//+ second;
return msg;
```

Una vez formateado de este modo se manda mediante el topic proyecto/tiempo al ESP8266.

En la parte inferior de la imagen se encuentra el código referente al tiempo de consumo máximo excedido.

Con el nodo “Comparativa de fuga”:

```
var epoch = global.get("epoch");
var now = msg.payload + 7200000; //Paso de UTC a GMT+1
var tconsumo = global.get("tconsumo");
if (now - epoch >= tconsumo)
{
    msg.payload = "Es posible que tenga una fuga. Ha tenido un consumo anómalo continuado entre la fecha " + new Date(epoch) + " y la fecha " + new Date(now);
    return msg;

}
else
{
    msg.payload = false;

}
```

Se realiza una comparación entre el *epoch* actual y el último *epoch* cuyo caudal fue igual a 0. Si esta diferencia de tiempo es mayor a la establecida en la variable global tconsumo se manda una alerta al *dashboard* y un email de aviso.

3.10.4 Home Monitoring flujo principal

Este código es el que controla las principales interacciones con la base de datos y como está información refresca parte de los datos del *dashboard*.

Para observar mejor el flujo del programa se incluido una versión aumentada en el Anexo B.

Toda la cadena se ejecuta cuando un mensaje llega al topic proyecto/contador/agua, tras un pequeño delay de 3 segundos se inician 2 caminos distintos.

Los mensajes enviados a través de este topic tienen el siguiente formato:

"2019-11-17-12-34-218"

Cuya información corresponde al año, mes, día, horas, minutos y pulsos captados por el contador.

3.10.4.1 Actualización y gestión de contadores

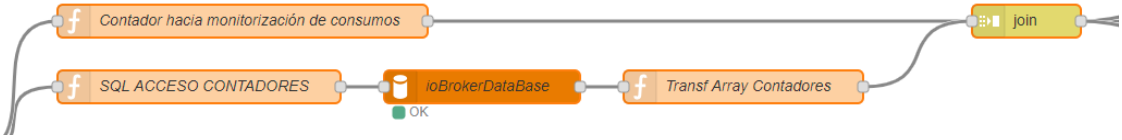

FIG. 47 - CONTADORES PARTE 1.

El camino superior utiliza el nodo "Contador hacia monitorización de consumos" para extraer la información referente a la cantidad de pulsos del contador que se divide entre el número de pulsos/litro contenido en la variable global 'pulsos'. Si la información no es un número manda un cero para evitar cometer un error.

```
var str=[];
str = msg.payload.split("-");
if(isNaN(str[5]/global.get('pulsos')) === true )
{
    msg.payload = 0;
}
else
{
    msg.payload = str[5]/global.get('pulsos');
}
return msg;
```

Al mismo tiempo el nodo "SQL ACCESO CONTADORES" se ejecuta lanzando la siguiente consulta a la base de datos:

```
msg.topic = "SELECT * FROM contadores";
return msg;
```

El resultado de la consulta se divide en "Transf. Array Contadores".

```
msg.payload      =      msg.payload[0].contador_general      +'-'+
msg.payload[0].contador_parcial1                             +'-'+
msg.payload[0].contador_parcial2;
return msg;
```

Y a continuación todo se une con un join.

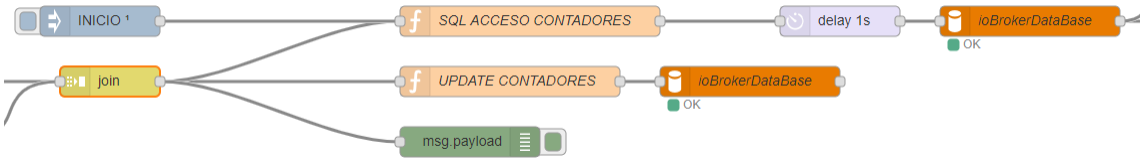

FIG. 48 - CONTADORES PARTE 2.

Por un lado, este *join* activa de nuevo una consulta a la base de datos mediante "SQL ACCESO CONTADORES" con un código análogo al anterior. Tras esto se produce un retardo de un segundo. La razón de retrasar la consulta es esperar hasta que los datos hayan sido actualizados en la base de datos mediante "UPDATE CONTADORES".

```
var contador_general;
var contador_parcial1;
var contador_parcial2;
var str=[];
str = msg.payload.split("-");
contador_general = Number(str[0])+Number(str[1]) ;
contador_parcial1 = Number(str[0])+Number(str[2]) ;
contador_parcial2 = Number(str[0])+Number(str[3]) ;
msg.topic    =    "UPDATE    contadores    SET    contador_general=
'"+contador_general+"', contador_parcial1= '"+contador_parcial1+"',
contador_parcial2= '"+contador_parcial2+"' ";
return msg;
```

Este nodo añade el valor de pulsos obtenido anteriormente a los tres contadores y luego genera la consulta para guardar los valores en la base de datos.

En la figura también se observa un nodo denominado INICIO[1]. Este ejecuta la consulta de acceso a contadores para refrescar la información por primera vez y evitar que no se muestren datos en el *dashboard*.

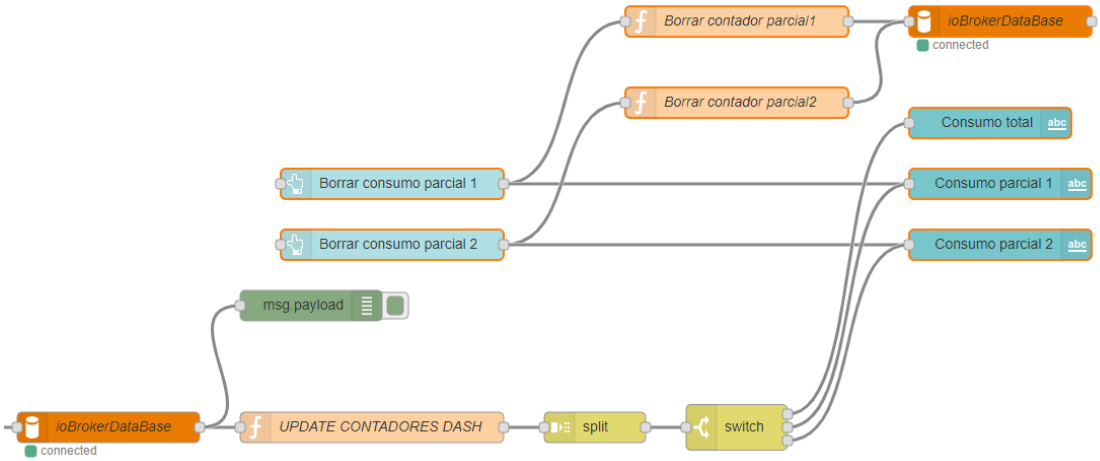

FIG. 49 - CONTADORES PARTE 3.

La información obtenida de la base de datos, ahora ya actualizada, se prepara para ser dividida mediante "UPDATE CONTADORES DASH"

```
var contador_general = msg.payload[0].contador_general;
var contador_parcial1 = msg.payload[0].contador_parcial1;
var contador_parcial2 = msg.payload[0].contador_parcial2;

msg.payload = contador_general + "/n" + contador_parcial1 +  "/n" +
contador_parcial2;
return msg;
```

El nodo *split* las trocea y el *switch* las manda a donde corresponde para mostrar los datos en el *dashboard*. En la siguiente figura se muestra cómo queda la interfaz gráfica.

Los nodos "consumo total", "consumo parcial1" y "consumo parcial2" son idénticos entre sí y sirven para generar los campos de visualización de los contadores.

Los nodos "Borrar consumo parcial 1" y "Borrar consumo parcial 2" crean los botones para borrar la información de la base de datos mediante una consulta de este tipo:

```
var contador_parcial1 = 0;
msg.topic    =    "UPDATE    contadores    SET    contador_parcial1=
'"+contador_parcial1+"'";
return msg;
```

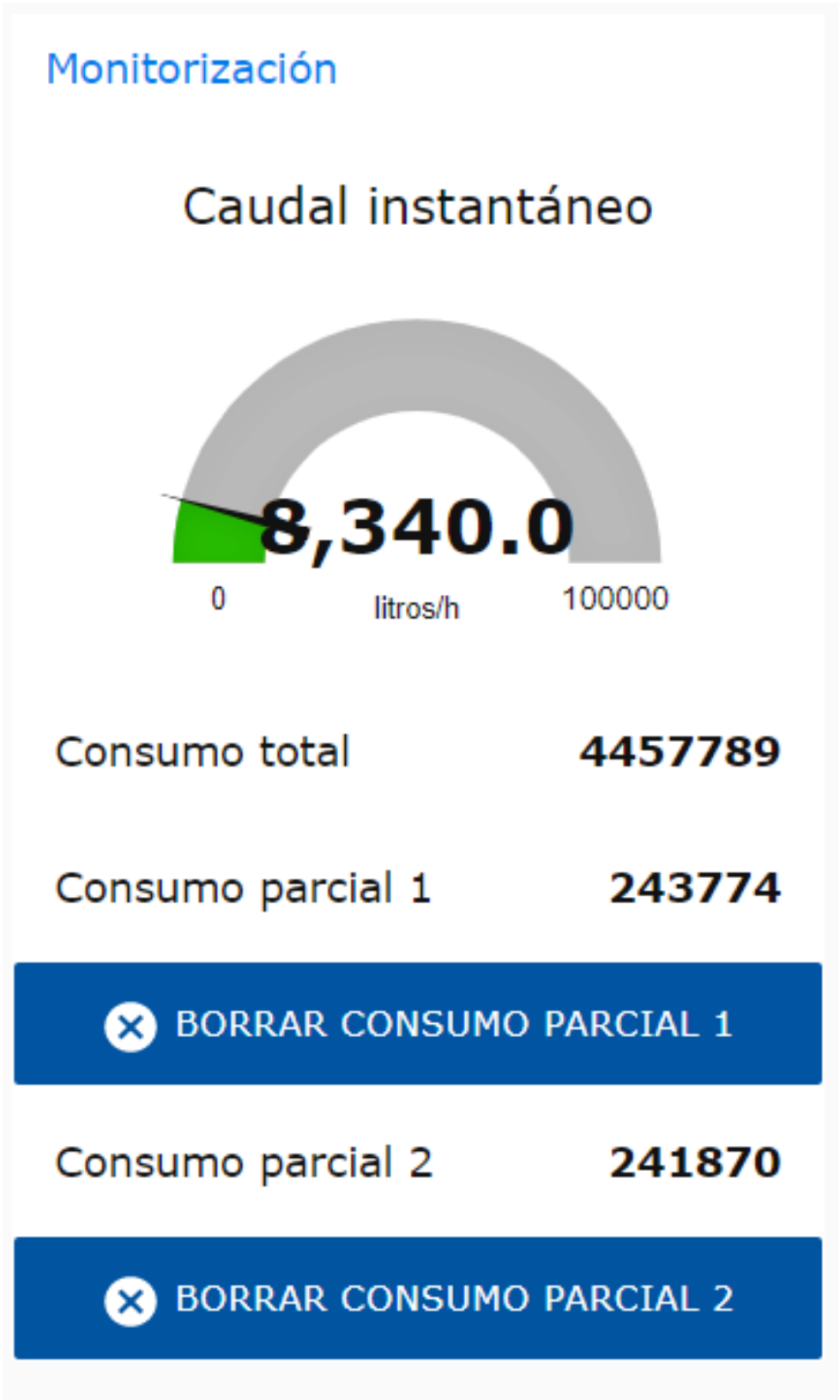

FIG. 50 - MONITORIZACIÓN.

3.10.4.2 Flujo principal

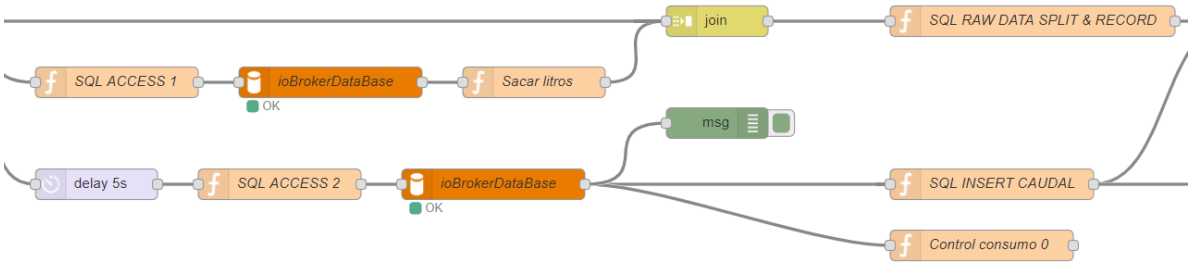

FIG. 51 - SECCIÓN PRINCIPAL PARTE 1.

El camino inferior realiza algunas operaciones en paralelo. Para comenzar mediante "SQL ACCESS 1" se realiza una consulta a la base de datos:

```
msg.topic = "SELECT * FROM raw_data ORDER BY fecha DESC limit 1";
return msg;
```

Así se obtiene el último valor almacenado en la tabla raw_data y a continuación se obtiene el valor del campo litros con "Sacar litros":

```
msg.payload = msg.payload[0].litros;
return msg;
```

A continuación, los datos se unen en un nodo *join* y se lanza una consulta, "SQL RAW DATA SPLIT & RECORD", para guardar este valor:

```
var str=[];
var a = 0;
var b = 0;
var litrosMsg;
var caudal = 0;
var d = new Date();
var epoch = d.getTime();

str = msg.payload.split("-");
//Por orden los valores dentro del array str ser□ a□mes, d hora,
minuto, segundo y por ltimo medida del contador en litros.
var fechaMsg = str[0]+"-"+str[1]+"-"+str[2]+" "+str[3]+":"+str[4];
a = Number(str[5]);
b = Number(str[6]);
a = a / (global.get('pulsos')|| 0);
litrosMsg = a + b;
msg.topic = "INSERT INTO raw_data VALUES
('"+fechaMsg+"','"+litrosMsg+"','"+caudal+"','"+epoch+"')";
msg.payload = a;

return msg;
```

Mientras se ejecutan estas operaciones un nodo de retardo espera 5 segundos para realizar una segunda consulta mediante “SQL ACCESS 2”:

```
msg.topic = "SELECT * FROM raw_data ORDER BY fecha DESC limit 2";
return msg;
```

Esta consulta obtiene las dos filas más recientes de la tabla raw_data. Una de ellas coincidirá con la fila obtenida mediante “SQL ACCESS 1” mientras que la segunda será igual a los valores recientemente incluidos tras la consulta “SQL RAW DATA SPLIT & RECORD”.

Con estos valores el nodo “Control consumo 0” compara los valores y de ser iguales determina que no ha habido paso de agua por el contador durante este intervalo de tiempo y guarda el valor de epoch del último mensaje de forma global. Este valor es el utilizado en el nodo “Comparativa de fuga” para determinar si hay una pequeña fuga que se produzca de manera continuada en el tiempo.

```
if(msg.payload[0].litros == msg.payload[1].litros)
{
global.set("epoch",msg.payload[1].epoch)
return msg;
}
```

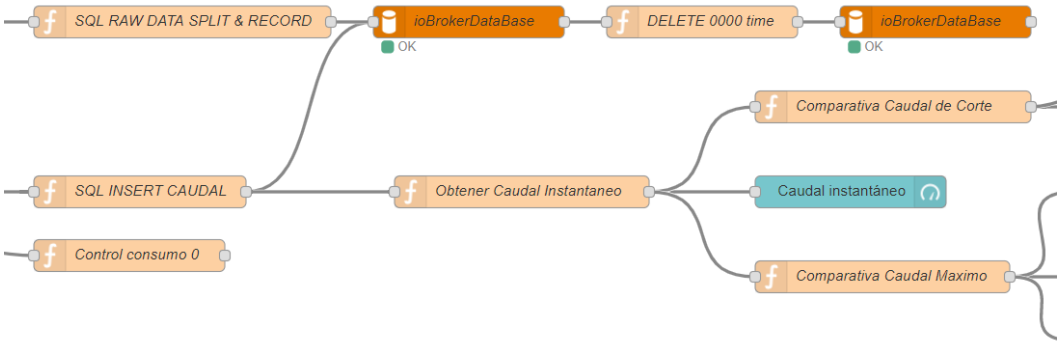

FIG. 52 - SECCIÓN PRINCIPAL PARTE 2.

Con estos dos valores el nodo "SQL INSERT CAUDAL" calcula el caudal y genera una consulta para guardar el valor en la base de datos:

```
var caudalMsg; //Almacena el caudal en litros por segundo.
var caudalTiempoMsg; //Devuelve el tiempo para colocar el caudal en
la fila correcta de la tabla.
//MANEJO DEL TIEMPO INICIO
var date = msg.payload[0].fecha;
var month = date.getMonth()+1;
var dt = date.getDate();
if (dt < 10) {
  dt = '0' + dt;
}
if (month < 10) {
  month = '0' + month;
}
caudalTiempoMsg       =       date.getFullYear()+'-'+month+'-'+dt+'
'+date.getHours()+':'+date.getMinutes()+':'+date.getSeconds();
//MANEJO DEL TIEMPO FIN
caudalMsg           =           (msg.payload[0].litros           -
msg.payload[1].litros)/((msg.payload[0].fecha                    -
msg.payload[1].fecha)/3600000);

if(caudalMsg == msg.payload[0].litros){
    return null;
}
else if(caudalMsg <= 0){
    return null;
}
else{
    //msg.payload = caudalMsg;
    msg.topic = "UPDATE raw_data SET caudal= '"+caudalMsg+"' WHERE
fecha = '"+caudalTiempoMsg+"'";
    return msg;
}
```

Tras guardar la información en la base de datos se manda una nueva consulta que borra de la tabla de datos las filas con fecha nula mediante el nodo "DELETE 0000 time".

```
msg.topic = "DELETE FROM raw_data WHERE fecha = '0000-00-00 00:00:00'
";
return msg;
```

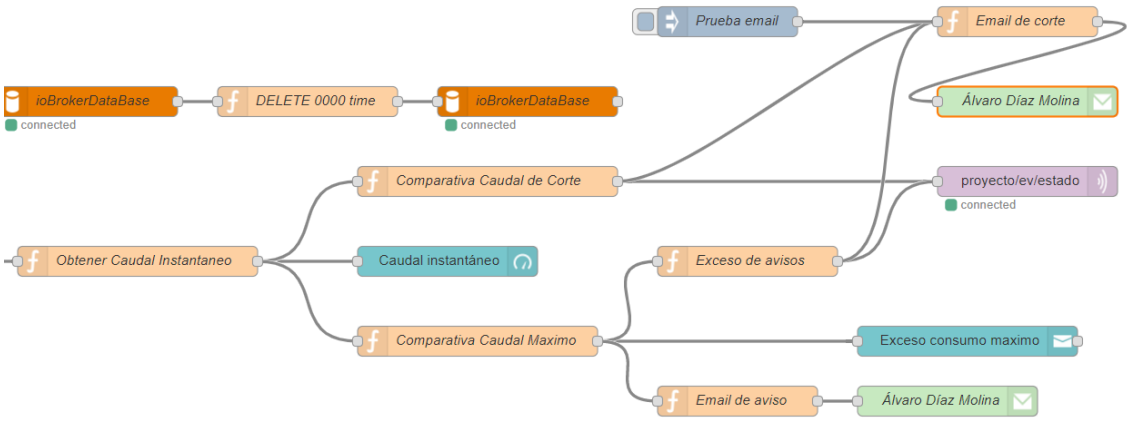

FIG. 53 - SECCIÓN PRINCIPAL PARTE 3.

Mientras se inserta el caudal calculado en la base de datos se activa el nodo "Obtener Caudal Instantáneo":

```
msg.payload = msg.payload[1].caudal;
return msg;
```

Este caudal se manda al dashboard mediante el nodo "Caudal instantáneo" para su visualización mediante un indicador (ver figura: Monitorización).

El valor también se utiliza para disparar el cierre de la electroválvula con el nodo "Comparativa Caudal de Corte":

```
if(msg.payload>=global.get("caudalc"))
{
    msg.payload = "false";
    return msg;
}
```

Si el caudal es mayor que el almacenado en la variable global caudalc se abre la electroválvula y se activa el nodo "Email de corte":

```
if (msg.payload != global.get("ev"))
{
msg.payload = "Se ha cortado su suministro de agua por razones de
seguridad.";
msg.topic = "Corte de suministro.";
msg.to = global.get("email");
return msg;
}
```

Esta información se pasa a un nodo que envía un email a la dirección especificada en la variable global email.

La configuración de todos los nodos de envío de email es la siguiente:

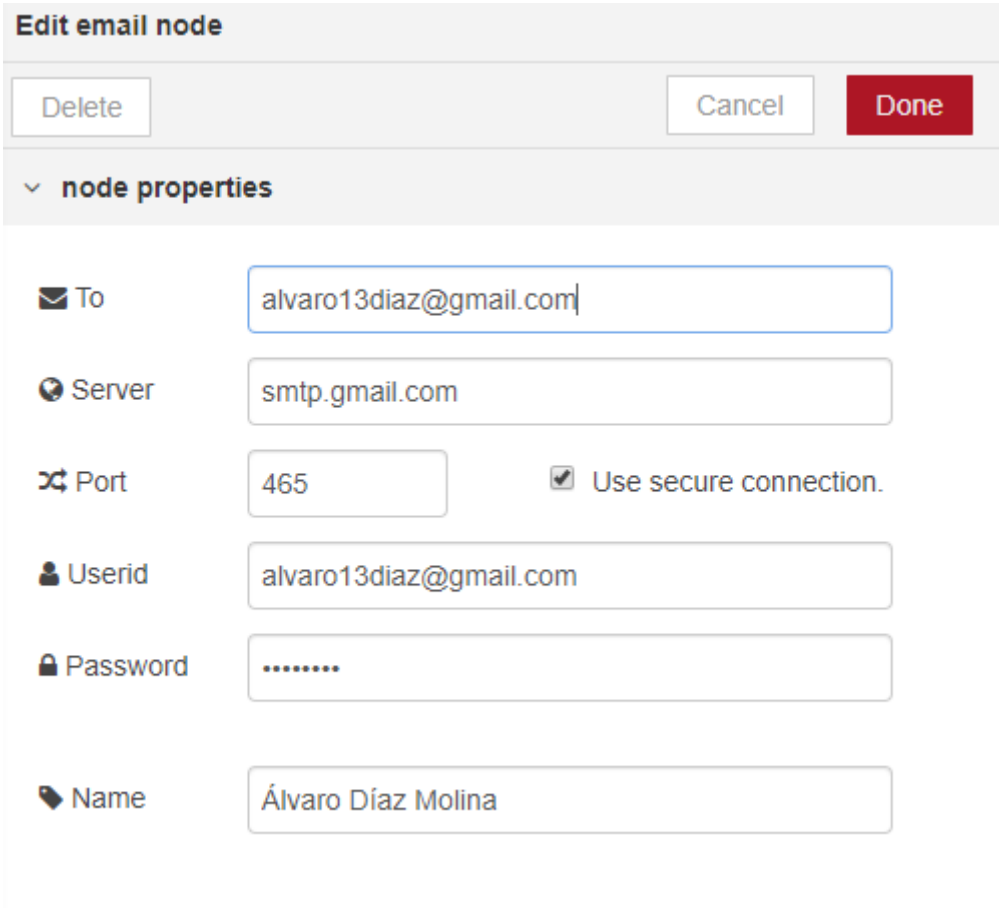

FIG. 54 - CONFIGURACIÓN DEL NODO EMAIL.

Esta será la cuenta utilizada para mandar los emails.

Por otro lado, hay otra forma de avisos por superación de caudal máximo, siguiendo el nodo “Comparativa Caudal Máximo”:

```
var counter;
if(msg.payload>=global.get("caudalm"))
{
    msg.payload = "Se supera el caudal máximo permitido";
    counter+=1;
    msg.payload.contador = counter;
    return msg;
}
else
{
    counter = 0;
}
```

Pero a diferencia de la comparación por exceso del caudal de corte este nodo va aumentado un contador. Si bien se manda un aviso al *dashboard* y un email cada vez que llega un mensaje con superación del caudal máximo solo se cierra la electroválvula en caso de superar el número máximo de avisos:

```
if(msg.payload.contador>=avisosm)
{
    msg.payload = "false";
    return msg;
}
```

3.10.5 Dashboard

Para visualizar la información recogida y poder configurar las distintas variables, así como actuar sobre la electroválvula, se ha creado una interfaz gráfica que aun todas estas funciones. Se puede acceder a ella mediante la dirección 192.168.1.254:1880/ui siempre que tu dispositivo este dentro de la red wifi que contiene la raspberry.

Node-red ofrece una forma nativa de estableces esta interfaz sencilla de utilizar y con mucha potencia. Para el proyecto la *dashboard* se ha dividido en dos pestañas, una principal y otra destinada a la configuración.

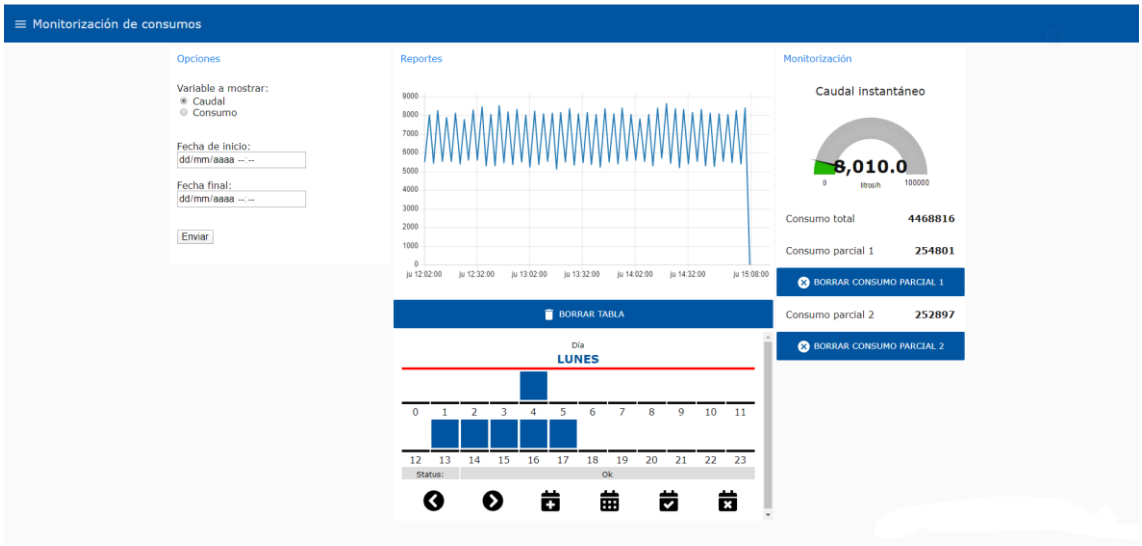

FIG. 55 - PANTALLA DE MONITORIZACIÓN DE CONSUMOS DEL DASHBOARD.

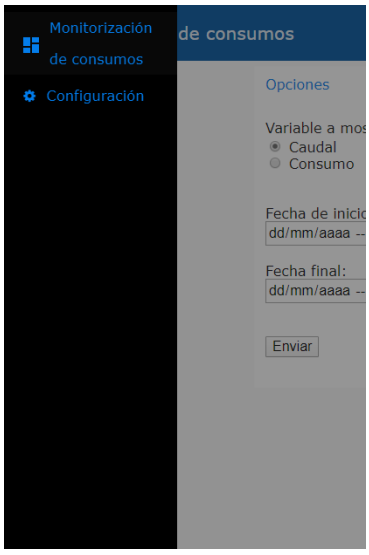

FIG. 56 - MENÚ DE NAVEGACIÓN DEL DASHBOARD.

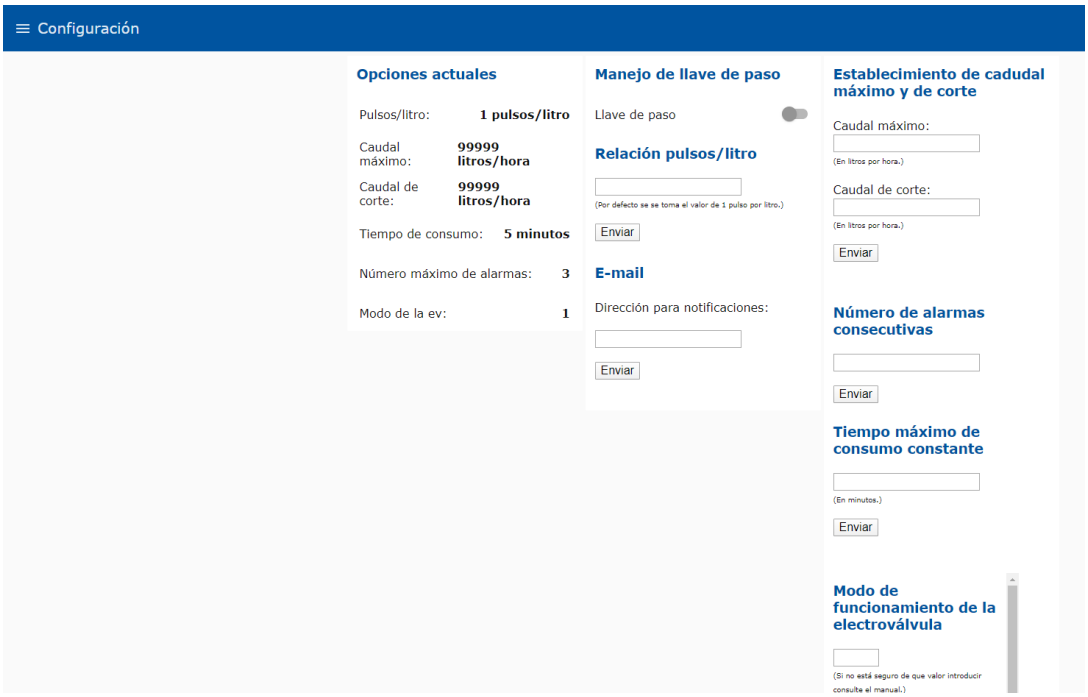

FIG. 57 - PANTALLA DE CONFIGURACIÓN DEL DASHBOARD.

Los colores, iconos y distribución de los formularios son configurables lo que da una gran flexibilidad. Además, es muy fácil de utilizar a través de distintos dispositivos puesto que cada widget se reordena para ofrecer la mejor visualización en función de la relación de aspecto de la pantalla.

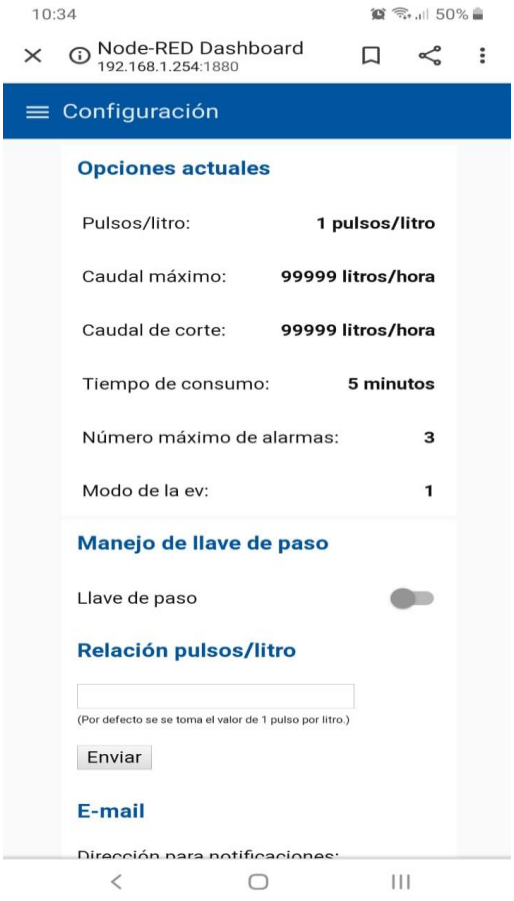

FIG. 58 - PANTALLA DE CONFIGURACIÓN DEL DASHBOARD EN DISPOSITIVO MÓVIL.

3.10.5.1 Reportes

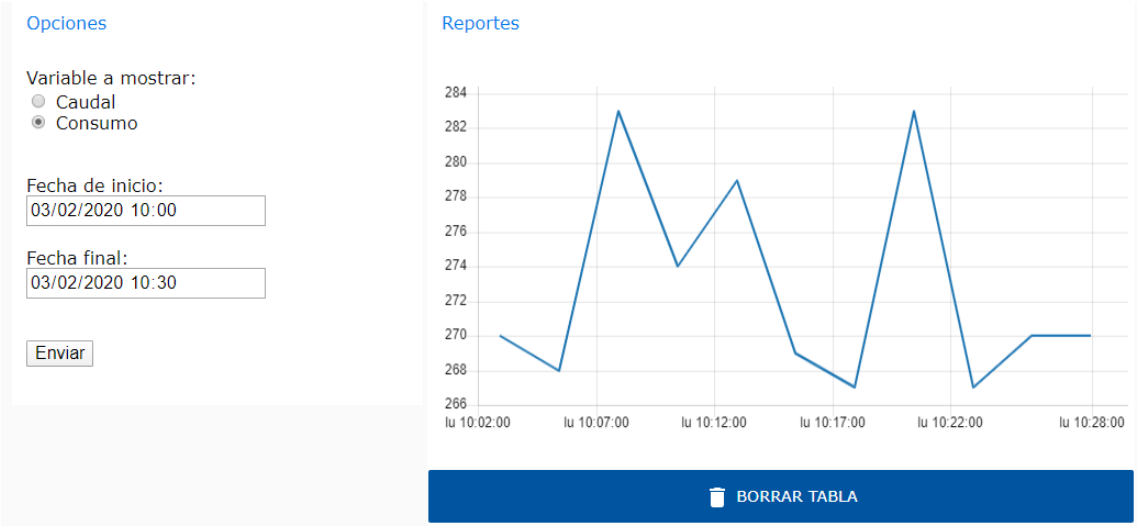

FIG. 59 - WIDGET DE OPCIONES Y WIDGET DE REPORTES DEL DASHBOARD.

Esta parte de la interfaz gráfica permite seleccionar el consumo o el caudal entre dos fechas determinadas. Para entender su funcionamiento hay que acudir a los nodos que dentro del flow Dashboard contienen el código.

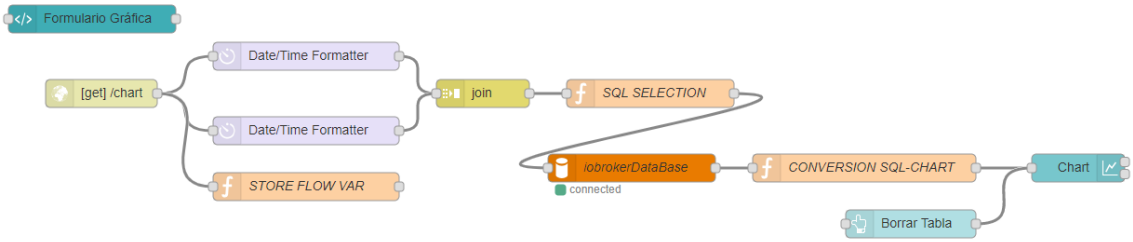

FIG. 60 - PROGRAMACIÓN DE OPCIONES Y REPORTES DEL DASHBOARD.

Para comenzar hay un nodo de tipo *template* llamado “Formulario Gráfica”. El código en HTML permite dar la forma deseada al formulario, añadiendo como en este caso campos de entrada circulares para la selección de la variable a mostrar o campos de entrada que facilitan la introducción de la fecha y la hora, fuente y tamaño del texto, así como la acción a realizar al pulsar el botón. En este caso la información activa el nodo [get]/chart que a su vez inicia la cadena de acciones necesarias para actualizar la información de la gráfica.

```
<iframe width="0" height="0" border="0" name="hiddenframe"
style="display:none;"></iframe>
<form action="/chart" target="hiddenframe">

  <!--
  <h3 style="color:RGB(0,84,160);font-type:verdana"
>Formulario</h3><br>
 -->
  <p>Variable a mostrar:</p>

  <input type="radio" name="variable" value="caudal" checked>
Caudal<br>
  <input type="radio" name="variable" value="litros"> Consumo<br>
  <br>

  <br>
  Fecha de inicio:<br>
  <input type="datetime-local" name="fecha1" required
autocomplete="on" title="Introduzca la fecha y hora a partir de la
cual se mostrarán los datos en la gráfica."><br>
  <br>
  Fecha final:<br>
  <input type="datetime-local" name="fecha2" required
autocomplete="on" title="Introduzca la fecha y hora hasta la cual se
mostrarán los datos en la gráfica."><br>
  <br>

  <br>
  <input type="submit" value="Enviar">
</form>
```

El mensaje obtenido por el nodo [get]/chart contiene 3 atributos: fecha1, fecha2 y variable. El campo dentro de variable se guarda de forma global mediante el nodo "STORE FLOW VAR":

```
var variable = flow.get('variable') || 0;
flow.set('variable',msg.payload.variable);
return null;
```

Fecha1 y fecha2 son formateadas se unen y luego se utilizan para una consulta a la base de datos en "SQL SELECTION":

```
var fecha1;
var fecha2;
var parts= [];
parts = msg.payload.split("-");
fecha1 = parts[0];
fecha2 = parts[1];

msg.topic="SELECT * FROM raw_data WHERE epoch >= "+fecha1+" AND epoch
<= "+fecha2+" ORDER BY epoch";
return msg;
```

En este caso el número de valores que puede devolver la base de datos es variable en función de las fechas utilizadas para la consulta. La información devuelta por la base de datos se da en forma de array. Este mensaje se transforma en el nodo "CONVERSION SQL_CHART":

```
var variable = flow.get('variable');
var msg2 = [];
var output = [];
if(variable == "caudal"){
    for (var i=0; i<msg.payload.length; i++) {
    output.push([msg.payload[i].epoch, msg.payload[i].caudal]);
    }
}else if(variable == "litros"){
        for (var i=1; i<msg.payload.length; i++) {
    output.push([msg.payload[i].epoch,    (msg.payload[i].litros    -
msg.payload[i-1].litros)]);
    }
}

msg2.push({ key: variable, values: output});
msg.topic = variable;
msg.payload = msg2;
return msg;
```

Tras lo cual se manda al nodo Chart para que se muestre en la interfaz.

Por otro lado, el nodo "Borrar Tabla" inyecta un mensaje con contenido {[]} que deja en blanco el gráfico.

3.10.5.2 Calendario

Dentro de la sección de reportes se encuentra un calendario que sirve para indicar los momentos del día donde se espera un consumo mantenido en el tiempo, como por ejemplo en el caso de tener programadas unas horas de riego o al llenar una piscina. Esto evitará que se active el aviso por exceder el valor en tconsumo.

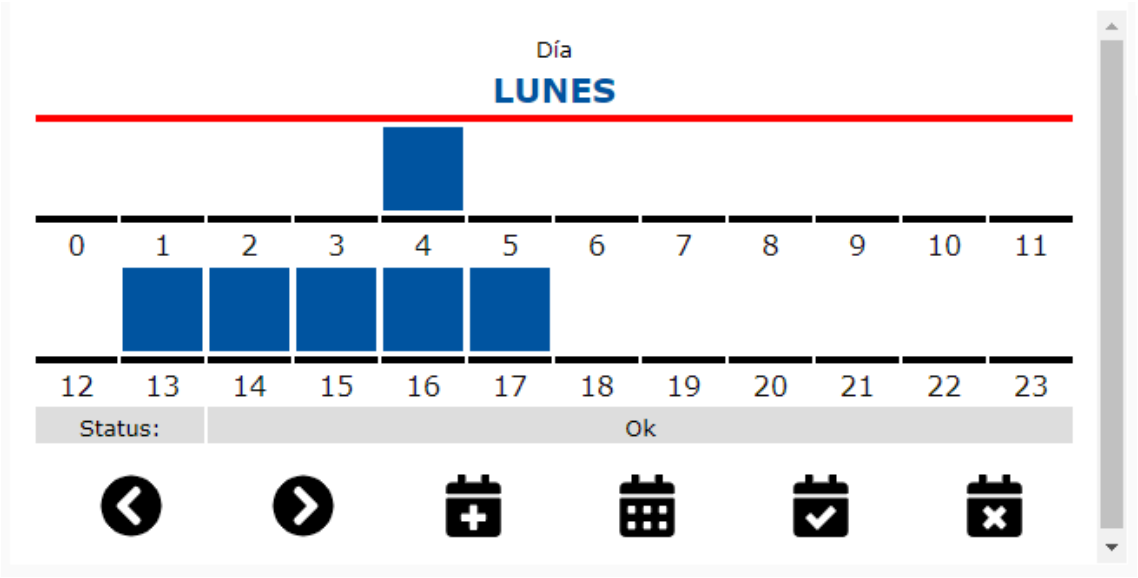

FIG. 61 - CALENDARIO DASHBOARD.

Cada uno de los rectángulos azules indica una hora del día donde no se activará el aviso por exceso de tiempo máximo de consumo constante.

Con los botones inferiores el usuario puede:

1. Retroceder al día anterior.
2. Avanzar al siguiente día.
3. Copiar la configuración del día actual al día siguiente, además avanza al día siguiente.
4. Copiar la configuración del día actual para todos los días de la semana.
5. Guardar los cambios en la base de datos.
6. Borrar los cambios. También actualiza la información a mostrar con la última configuración guardada en la base de datos.

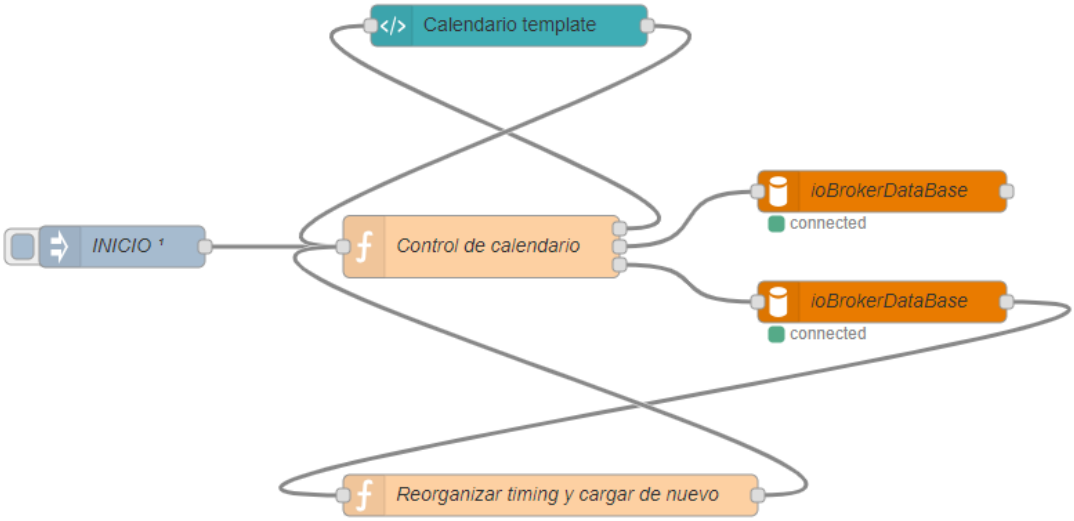

FIG. 62 - PROGRAMACIÓN DE CALENDARIO DASHBOARD.

El primer nodo “INICIO1” refresca la información al inicio de la aplicación para asegurarnos que siempre se muestra algo.

Dado que el código de los otros tres nodos, “Calendario template”, “Control de calendario” y “Reorganizar timing y cargar de nuevo”, es muy extenso en lugar de incluir aquí el código este se ha situado en un el Anexo B.

En líneas generales su función es controlar la visualización y la ejecución de las acciones de los botones anteriormente mencionados. Para ello las acciones sobre el dashboard activan el nodo “Control de calendario” que a su vez realiza consultas para guardar información u obtenerla de la base de datos.

El nodo “Reorganizar timing y cargar de nuevo” refresca la información.

3.10.5.3 Control de la electroválvula

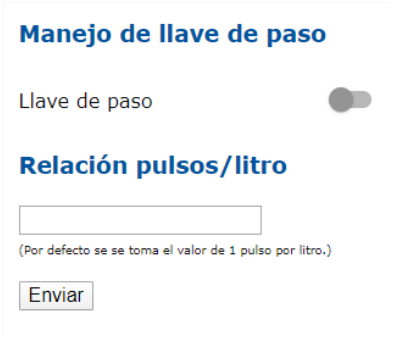

FIG. 63 - CONTROL DE LA ELECTROVÁLVULA DASHBOARD.

Esta sección de la pestaña de configuración permite abrir y cerrar la electroválvula, así como cambiar los valores de la relación pulsos/litro guardándolos en la base de datos.

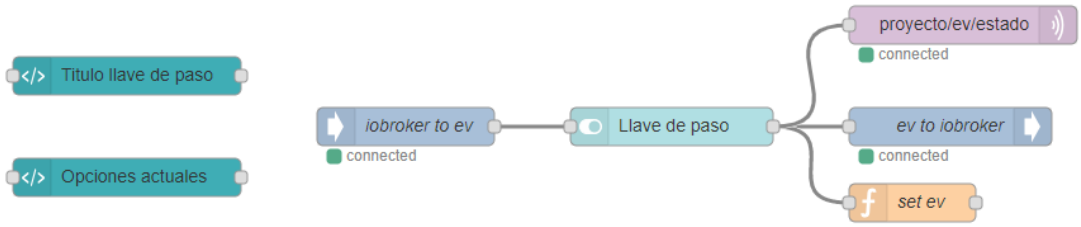

FIG. 64 - PROGRAMACIÓN DE CONTROL DE LA ELECTROVÁLVULA DASHBOARD.

Los dos primeros nodos simplemente sirven para establecer el título:

```
<h3 style="color:RGB(0,84,160);font-type:verdana" >Manejo de llave de paso</h3>
```

```
<h3 style="color:RGB(0,84,160);font-type:verdana" >Opciones actuales</h3>
```

El resto del código que se puede ver en la figura manda el cambio de la electroválvula que es ejecutado desde el dashboard por mqtt al ESP8266, lo guarda dentro de una variable de iobroker y lo guarda como variable global de node-red.

```
global.set("ev",msg.payload);
return msg;
```

3.10.5.4 Flujo principal del dashboard 1

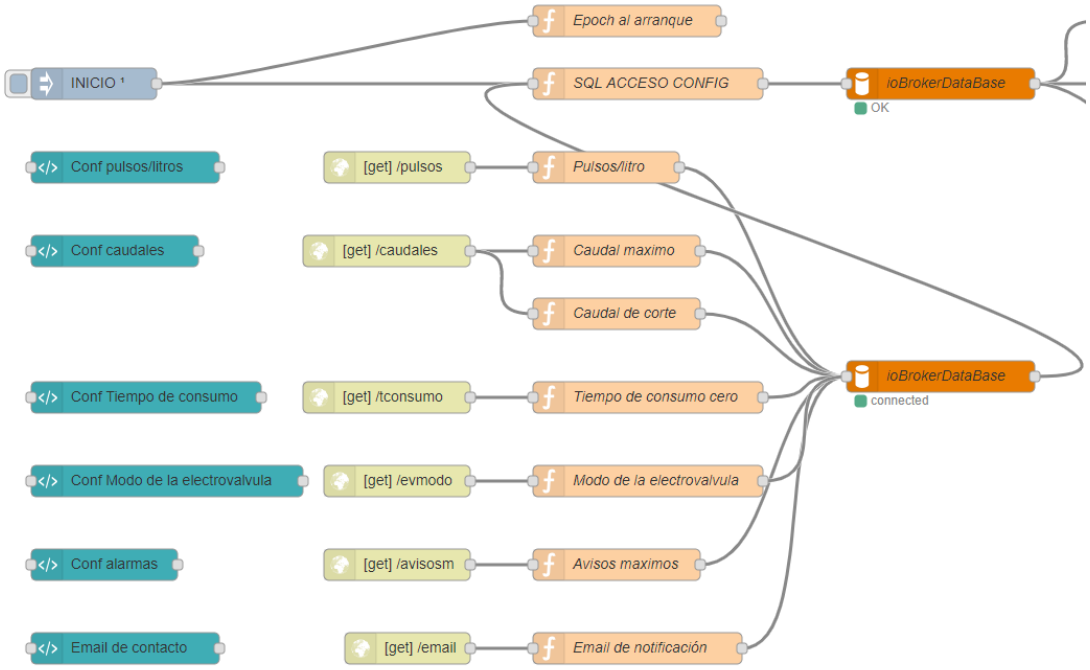

FIG. 65 - PROGRAMACIÓN DEL FLUJO PRINCIPAL 1 DASHBOARD.

Este código gestiona la configuración de parámetros del *dashboard*. Por un lado, nada más iniciar el programa el nodo "Epoch al arranque" guarda el valor del *epoch* para evitar no disponer de ningún valor:

```
msg.payload += 7200000; //Paso de UTC a GMT+1
global.set("epoch",msg.payload);
return msg;
```

Por otro lado se activa "SQL ACCESO CONFIG" para obtener la información de la tabla config de la base de datos:

```
msg.topic = "SELECT * from config"
return msg;
```

Los nodos de tipo *template* situados a la izquierda de la figura contiene la información relativa a cómo deben mostrarse los formularios:

“Conf pulsos/litros”:

```
<iframe  width="0"  height="0"  border="0"  name="hiddenframe3"
style="display:none;"></iframe>
<form action="/pulsos" target="hiddenframe3">

  <h3                style="color:RGB(0,84,160);font-type:verdana"
>Relaci□ulsos/litro</h3><br>
  <input type="number" step="0.1" name="pulsos" title="Escriba aqui
el nmero de pulsos por litro de su contador." required><br>
  <p1 style="font-size:60%">(Por defecto se se toma el valor de 1
pulso por litro.)</p1>
  <br><br>
  <input type="submit" value="Enviar" title="Enviar dato.">
</form>
```

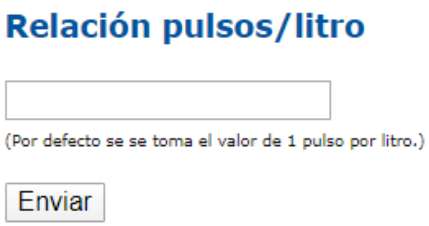

FIG. 66 - RELACIÓN PULSOS/LITRO DASHBOARD.

“Conf Tiempo de consumo”:

```
<iframe  width="0"  height="0"  border="0"  name="hiddenframe7"
style="display:none;"></iframe>
<form action="/tconsumo" target="hiddenframe7">

    <h3 style="color:RGB(0,84,160);font-type:verdana" >Tiempo mḥmo
de consumo constante</h3><br>
    <input  type="number"  step="1"  min="1"  name="tconsumo"
title="Caudal mḥmo admisible." required><br>
    <p1 style="font-size:60%">(En minutos.)</p1><br><br>
    <input type="submit" value="Enviar" title="Enviar caudales y
avisos.">
</form>
```

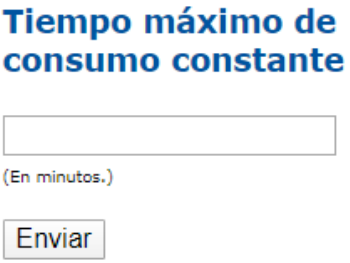

FIG. 67 - TIEMPO MÁXIMO DE CONSUMO CONSTANTE DASHBOARD.

“Conf caudales”:

```
<iframe   width="0"   height="0"   border="0"   name="hiddenframe4"
style="display:none;"></iframe>
<form action="/caudales" target="hiddenframe4">

    <h3               style="color:RGB(0,84,160);font-type:verdana"
>Establecimiento de cadudal mínimo y de corte</h3><br>
    <p>Caudal mínimo:</p>
    <input  type="number"  step="0.1"  name="caudales"  title="Caudal
mḥmo admisible." required><br>
    <p1 style="font-size:60%">(En litros por hora.)</p1>
    <br><br>
    <p>Caudal de corte:</p>
    <input type="number" step="0.1" name="caudales" title="Caudal de
corte." required><br>
    <p1 style="font-size:60%">(En litros por hora.)</p1>
    <br><br>
    <input type="submit" value="Enviar" title="Enviar caudales.">
</form>
```

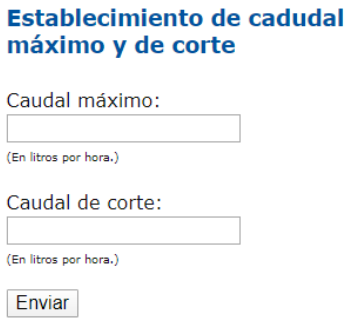

FIG. 68 - CAUDALES DASHBOARD.

“Conf Modo de la electroválvula”:

```
<iframe width="0" height="0" border="0" name="hiddenframe8" style="display:none;"></iframe>
<form action="/evmodo" target="hiddenframe8">

    <h3 style="color:RGB(0,84,160);font-type:verdana" >Modo de funcionamiento de la electrovávula</h3><br>
    <input type="number" step="1" min="1" max="4" name="evmodo" title="Modo de la electrov□ula." required><br>
    <p1 style="font-size:60%">(Si no est*eguro de que valor introducir consulte el manual.)</p1><br><br>
    <input type="submit" value="Enviar" title="Enviar caudales y avisos.">
</form>
```

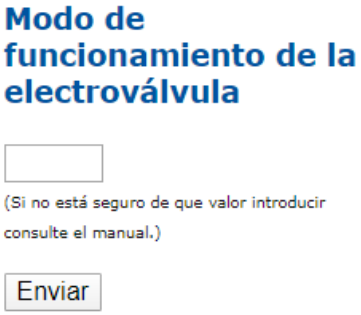

FIG. 69 - MODO DE LA ELECTROVÁLVULA DASHBOARD.

“Conf alarmas”:

```
<iframe width="0" height="0" border="0" name="hiddenframe5" style="display:none;"></iframe>
<form action="/avisosm" target="hiddenframe5">
    <h3 style="color:RGB(0,84,160);font-type:verdana" >Número de alarmas consecutivas</h3><br>
    <input type="number" step="1" min="1" name="avisos" title="Nmero de alarmas consecutivas." required><br>
    <br>
    <input type="submit" value="Enviar" title="Enviar caudales y avisos.">
</form>
```

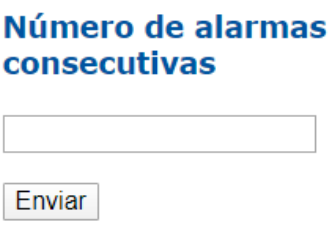

FIG. 70 - ALARMAS CONSECUTIVAS DASHBOARD.

“Email de contacto”:

```
<iframe   width="0"   height="0"   border="0"   name="hiddenframe6"
style="display:none;"></iframe>
<form action="/email" target="hiddenframe6">

    <h3      style="color:RGB(0,84,160);font-type:verdana"      >E-
mail</h3><br>
    <p>Direcci□ara notificaciones:</p> <br>
    <input                  type="text"                  name="email"
title="email_ejemplo@dominio.com" required><br><br>
    <input                type="submit"                value="Enviar"
title="email_ejemplo@dominio.com">
</form>
```

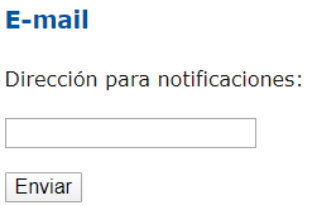

FIG. 71 - EMAIL DASHBOARD.

Los nodos de tipo [get] reciben la información de los formularios. Al presionar enviar en uno de los formularios el nodo en cuestión se activa y genera una consulta a la base de datos para guardar el nuevo valor en la tabla config.

Tras guardar la información se realiza la misma consulta que al arranque para actualizar la información en pantalla.

3.10.5.5 Flujo principal del dashboard 2

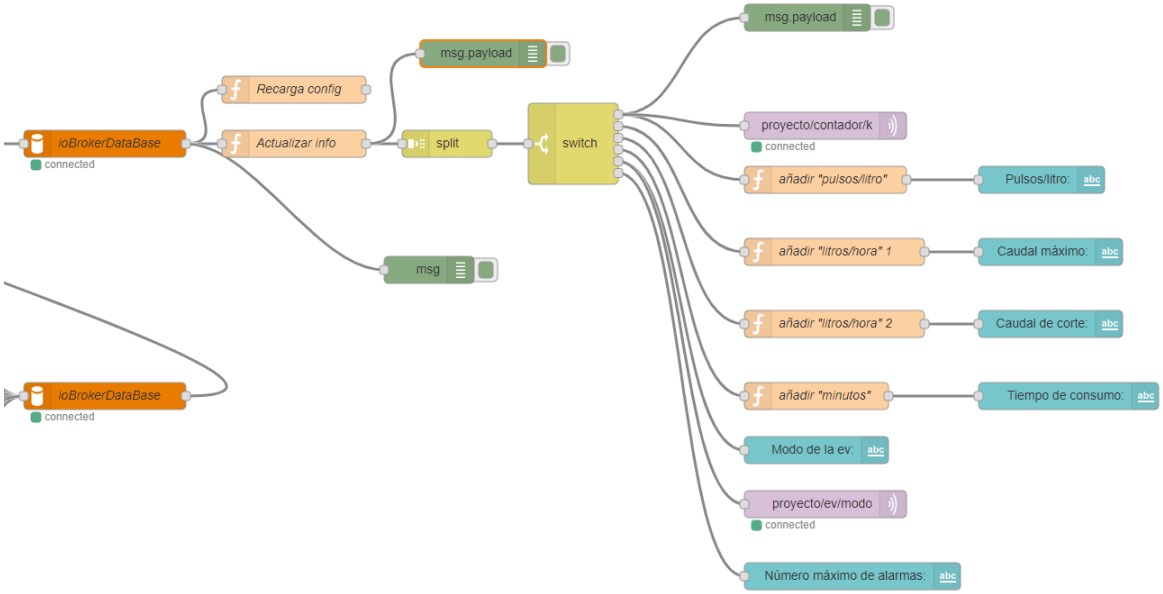

FIG. 72 - PROGRAMACIÓN DEL FLUJO PRINCIPAL 2 DASHBOARD.

De la consulta anterior se saca la información contenida en la tabla config. Esta es introducida en variables mediante el nodo "Recarga config":

```
global.set("pulsos",msg.payload[0].pulsos);
global.set("caudalm",msg.payload[0].caudalm);
global.set("caudalc",msg.payload[0].caudalc);
global.set("tconsumo",msg.payload[0].tconsumo);
global.set("avisosm",msg.payload[0].avisosm);
global.set("email",msg.payload[0].email);
global.set("evmodo",msg.payload[0].evmodo);
return msg;
```

Y se une con "Actualizar info":

```
msg.payload = msg.payload[0].pulsos + "-" + msg.payload[0].caudalm +
"-" + msg.payload[0].caudalc + "-" + msg.payload[0].tconsumo + "-" +
msg.payload[0].evmodo + "-" + msg.payload[0].avisosm;
return msg;
```

Para luego ser separada y mandada al dashboard. A algunos de estos datos se les añaden las unidades antes de salir al dashboard. Otros como el modo de la electroválvula o la constanteK del contador son enviados mediante MQTT al ESP8266.

Finalmente, la información se muestra del siguiente modo:

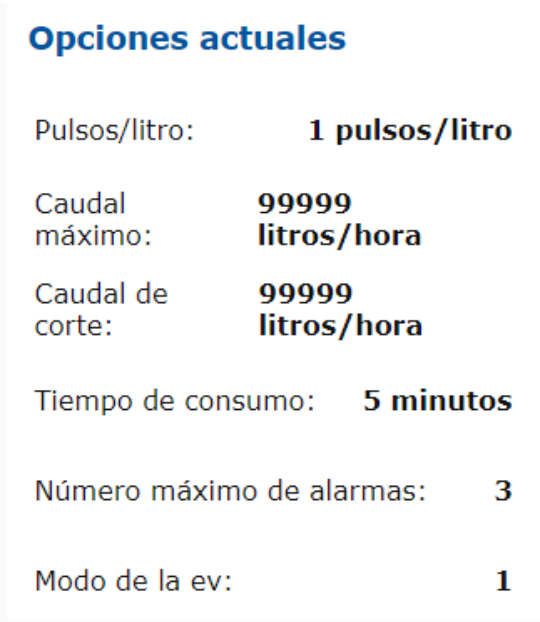

FIG. 73 - OPCIONES ACTUALES DASHBOARD.

3.10.5.6 Consideraciones adicionales sobre el dashboard

Para organizar la forma en las que los widgets aparecen en pantalla hay que acudir a la barra lateral derecha, sección dashboard.

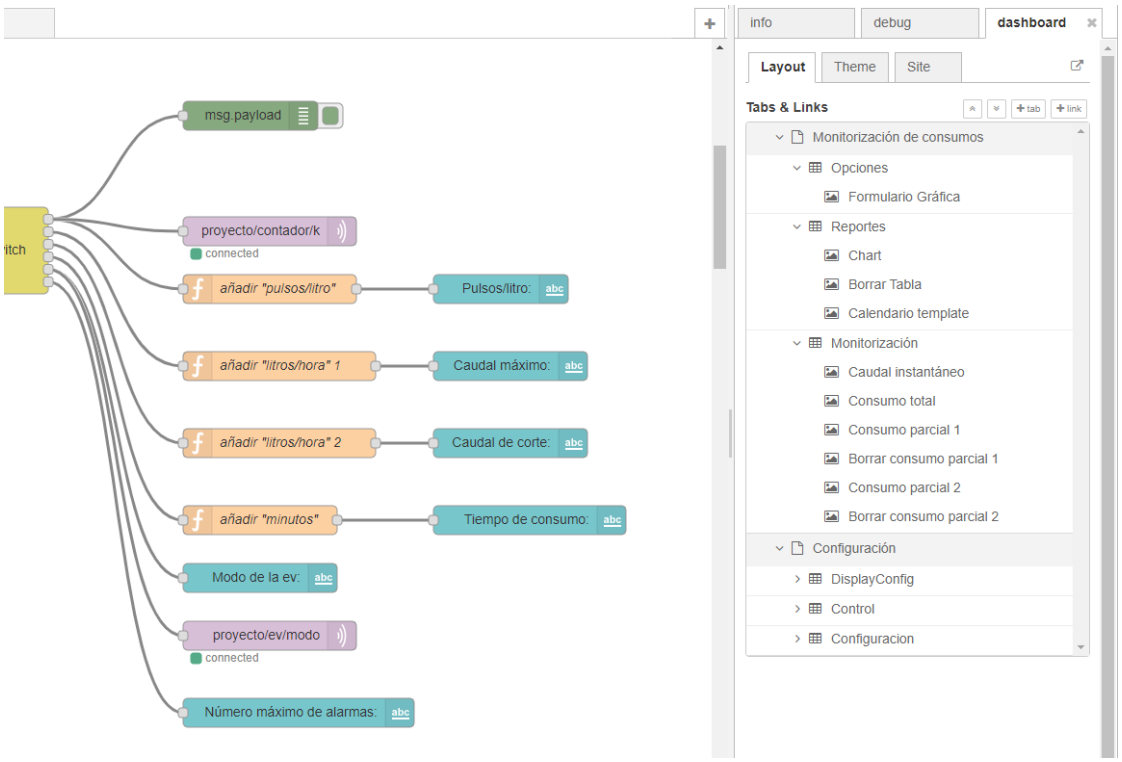

FIG. 74 - LAYOUT DASHBOARD.

Aquí los widgets son organizados en pestañas (Monitorización de consumos y Configuración) y en secciones que agrupan los distintos nodos (Opciones, reportes, monitorización, etc.).

4 Análisis económico

Para comprobar la viabilidad económica del proyecto se ha estimado el coste por unidad ante el supuesto de que se fabricasen al menos 1000 unidades, en la siguiente tabla se muestra el coste derivado de la fabricación de la PCB:

Artículo	Unidades por PCB	Precio
Regularo lineal	1	0,22 €
Led	1	0,08 €
Condensador 10uF	1	0,09 €
Conector USB	1	0,63 €
Cuentas de ferrita	1	0,10 €
Diodos	8	0,17 €
Resistencias 1K	11	0,30 €
Rresistencias 100	4	0,14 €
Resistencias 47	1	0,04 €
Cpndensadores 0.33 uF	1	0,05 €
Condensadores 100n	5	0,32 €
Condensadores 10n	1	0,10 €
Reles	2	2,68 €
Mosfets	4	0,38 €
Pulsadores	2	0,75 €
FT232RL	1	2,39 €
ESP8266-12E	1	1,35 €
PCB	1	0,71 €
Montaje 30%	-	3,14 €
Precio por PCB:		13,62 €

FIG. 75 - COSTE UNITARIO POR PCB

El coste del resto de equipos necesario:

Artículo	Precio
Raspberry Pi3 (con accesorios)	4,24 €
Cable Ethernet	1,22 €
Batería	3,04 €
Circuito de carga	1,04 €
Contador con salida de pulsos	7,73 €
Turbina	2,17 €
Llave de paso motorizada	16,60 €
Llave de paso	10,00 €
Total:	46,04 €

FIG. 76 - COSTE DE EQUIPOS

Se ha añadido a estos costes el beneficio esperado:

Calculo del beneficio:	59,66 €

*(+100% sin contar el gasto de instalación)

FIG. 77 - BENEFICIO

El costo total estimado asciende a:

Precio PCB:	13,62 €
Otros artículos:	46,04 €
Instalación:	100,00 €
Beneficio:	59,66 €
Precio total final:	219,32 €

FIG. 78 - PRECIO FINAL

Cruzando estos datos con el coste estimado de las pérdidas descritas en el apartado 2.4 Pérdidas en vivienda de este documento se obtiene que el tiempo necesario para amortizar la inversión es de: **3 años y 2 meses.**

5 Conclusiones y desarrollos futuros

Mediante la realización de este proyecto de fin de grado se han logrado los objetivos planteados de crear un sistema que controle el consumo de agua y pueda reaccionar ante consumos anómalos enviando mensajes o cerrando el paso del agua. Además, este sistema es de bajo coste y gran autonomía, con una instalación poco invasiva y sencilla.

De cara a desarrollos futuros queda por realizar pruebas de funcionamiento montando la maqueta en una vivienda. Principalmente para la comprobación de pérdidas de carga por la turbina, autonomía de la batería en condiciones reales de funcionamiento y comportamiento de la placa base tras ser utilizada por períodos prolongados de tiempo.

Queda también abierta la puerta a la adaptación a otros protocolos de comunicación tales como 5G o LoRa, la parametrización del dispositivo mediante OTA (Over the Air) y el uso de machine learning para que una IA gestione la información y tome decisiones sobre cuando cortar el flujo de agua.

Además, se deja la puerta abierta a reutilizar lo expuesto en este trabajo para expandir el desarrollo a otros sistemas de gestión de consumos (gas, electricidad, etc.) u otras plataformas que serían fácilmente integrables gracias a la existencia de adaptadores para IoBroker como KNX.

6 Bibliografía

1. K. Ashton. That «Internet of Things» Thing. RFID J [Internet]. 22 de junio de 9d. C.;1. Disponible en: https://www.rfidjournal.com/articles/view?4986

2. Zanella A, Bui N, Castellani A, Vangelista L, Zorzi M. Internet of Things for Smart Cities. IEEE Internet Things J [Internet]. febrero de 2014 [citado 20 de febrero de 2020];1(1):22-32. Disponible en: https://ieeexplore.ieee.org/document/6740844/

3. Joe Biron, Shawn Kelly, David Immerman, Jon Lang. The State of Industrial Internet of Things 2019: Spotlight on Operational Effectiveness. PTC [Internet]. 2019; Disponible en: https://www.ptc.com/-/media/Files/PDFs/IoT/State-of-IIoT-Report-2019.pdf

4. Pedro L. Iglesias Rey, Gonzalo López Patiño, Fco. Javier Martínez Solano, P. Amparo López Jiménez. DETERMINACIÓN DEL VOLUMEN DE FUGAS EN INSTALACIONES INTERIORES DE SUMINISTRO DE AGUA. VI SEREA - Semin Iberoam Sobre Sist Abast Urbano Água João Pessoa Bras 5 7 Junho 2006. 5 de junio de 2006;12.

5. ioBroker Smarthome [Internet]. [citado 10 de junio de 2019]. Disponible en: https://www.iobroker.net/

6. ioBroker | MySensors - Create your own Connected Home Experience [Internet]. [citado 10 de junio de 2019]. Disponible en: https://www.mysensors.org/controller/iobroker

7. Node-RED [Internet]. [citado 10 de junio de 2019]. Disponible en: https://nodered.org/

7 Anexo A: Código del calendario dashboard

Código del nodo “Reorganizar timing y cargar de nuevo”:

```
for(i=0; i<7; i++)
{
    var k = -2;
    for(var j in msg.payload[i])
    {
        if (msg.payload[i].hasOwnProperty(j)){
        k++;
        if((msg.payload[i])[j]>=0 || (msg.payload[i])[j]<=1)
        {

            if ((msg.payload[i])[j] === 1)
            {
                context.global.calendario[(i*24)+k] = true;
            }
            else
            {
                context.global.calendario[(i*24)+k] = false;
            }
        }
        else
        {

        }
    }}
}
msg.payload = '';
return msg;
```

Código del nodo “Calendario template”:

```
<head>
<style>
.thedays { vertical-align:bottom; height:48px; }
.linea2px  { background-color:black; height:2px; }
.theblocks {width:100%; height:0%;}
.greybuttons { background-color:#ffffff !important; width:48px; }
.smallheadings { color:black; font-size:80%; }
</style>
<script>
var
thedays=["LUNES","MARTES","MIERCOLES","JUEVES","VIERNES","SABADO","
DOMINGO"];
var last=1; //Nos aseguramos de tener una variable clickeada?

function bar(mm,val) //Funcion para marcar el nivel de relleno azul
o blanco al clickar en un cuadrado horario.
{
if (val==false) {$(mm).height("100%"); $(mm).css('background-color',
'#ffffff'); } //#ffffff blanco
if (val==true) { $(mm).height("100%"); $(mm).css('background-color',
'#0054a0'); } // azul upct (on)
}
function stat(text)
{
$("#info").text(text);
var         tm=setTimeout(function(){          $("#info").text("Ok");
clearTimeout(tm);}, 3000);
}
function   selec(val,sta)   //Para   colorear   de   rojo   cada   item
seleccionado. O devolverlo a negro cuando se selcciona otro item.
{
var w="#td"+val;
 if        (sta)        $(w).css('background-color','red');        else
$(w).css('background-color','black');
}

    (function(scope){
        scope.$watch('msg', function(msg) {
            selec(last,0);     last=msg.selector;     selec(last,1);
//Esta   linea   "apaga   la   barra   roja"   del   item   anteriormente
seleccionado y activa el color rojo en el nuevo item.
            for (var x=0; x<24; x++)
                {
                    var                                          w="#t"+x;
bar(w,msg.calendario[((msg.days)*24)+x]);  //Llama a la función para
poner azul o blanco el cuadrado en cuestión/span>
                }
            for    (var    x=0;    x<2;    x++)    {    var    w="#s"+x;
$(w).text(msg.calendario[168+x]); }
             $("#d0").text(thedays[msg.days]);

            if (msg.foryou!="") { stat(msg.foryou);  }
        });

    })(scope);


</script>
```

```
</head>
<body>
    <table width="100%">

    <tr>
        <td      colspan=12><center><span      class="smallheadings"
>D/span></span></center></td>
    </tr>

    <tr>
        <td             ng-click="send({payload:             '24'})"
colspan=12><center><span    id="d0"    style="color:#0054a0;    font-
size:120%; font-weight:bold">LUNES</span></center></td>
    </tr>

    <tr style="height:2px">
        <td       id="td24"       colspan=12       style="background-
color:black;height:2px;"></td>
    </tr>

    <tr>
        <td ng-click="send({payload: '0'})" class="thedays">    <div
id="t0" class="theblocks"> </div></td>
        <td ng-click="send({payload: '1'})" class="thedays">    <div
id="t1" class="theblocks"> </div></td>
        <td ng-click="send({payload: '2'})" class="thedays">    <div
id="t2" class="theblocks"> </div></td>
        <td ng-click="send({payload: '3'})" class="thedays">    <div
id="t3" class="theblocks"> </div></td>
        <td ng-click="send({payload: '4'})" class="thedays">    <div
id="t4" class="theblocks"> </div></td>
        <td ng-click="send({payload: '5'})" class="thedays">    <div
id="t5" class="theblocks"> </div></td>
        <td ng-click="send({payload: '6'})" class="thedays">    <div
id="t6" class="theblocks"> </div></td>
        <td ng-click="send({payload: '7'})" class="thedays">    <div
id="t7" class="theblocks"> </div></td>
        <td ng-click="send({payload: '8'})" class="thedays">    <div
id="t8" class="theblocks"> </div></td>
        <td ng-click="send({payload: '9'})" class="thedays">    <div
id="t9" class="theblocks"> </div></td>
        <td ng-click="send({payload: '10'})" class="thedays">   <div
id="t10" class="theblocks">    </div></td>
        <td ng-click="send({payload: '11'})" class="thedays">   <div
id="t11" class="theblocks">    </div></td>
    </tr>

    <tr style="height:2px">
        <td id="td0" class="linea2px"></td>
        <td id="td1" class="lineapx"></td>
        <td id="td2" class="linea2px"></td>
        <td id="td3" class="linea2px"></td>
        <td id="td4" class="linea2px"></td>
        <td id="td5" class="linea2px"></td>
        <td id="td6" class="linea2px"></td>
        <td id="td7" class="linea2px"></td>
        <td id="td8" class="linea2px"></td>
        <td id="td9" class="linea2px"></td>
        <td id="td10" class="linea2px"></td>
        <td id="td11" class="linea2px"></td>
    </tr>
```

```
    <tr>
        <td><center>0</center></td>
        <td><center>1</center></td>
        <td><center>2</center></td>
        <td><center>3</center></td>
        <td><center>4</center></td>
        <td><center>5</center></td>
        <td><center>6</center></td>
        <td><center>7</center></td>
        <td><center>8</center></td>
        <td><center>9</center></td>
        <td><center>10</center></td>
        <td><center>11</center></td>
    </tr>

    <tr>
        <td  ng-click="send({payload:  '12'})"  class="thedays"><div
id="t12" class="theblocks"></div></td>
        <td  ng-click="send({payload:  '13'})"  class="thedays"><div
id="t13" class="theblocks"></div></td>
        <td  ng-click="send({payload:  '14'})"  class="thedays"><div
id="t14" class="theblocks"></div></td>
        <td  ng-click="send({payload:  '15'})"  class="thedays"><div
id="t15" class="theblocks"></div></td>
        <td  ng-click="send({payload:  '16'})"  class="thedays"><div
id="t16" class="theblocks"></div></td>
        <td  ng-click="send({payload:  '17'})"  class="thedays"><div
id="t17" class="theblocks"></div></td>
        <td  ng-click="send({payload:  '18'})"  class="thedays"><div
id="t18" class="theblocks"></div></td>
        <td  ng-click="send({payload:  '19'})"  class="thedays"><div
id="t19" class="theblocks"></div></td>
        <td  ng-click="send({payload:  '20'})"  class="thedays"><div
id="t20" class="theblocks"></div></td>
        <td  ng-click="send({payload:  '21'})"  class="thedays"><div
id="t21" class="theblocks"></div></td>
        <td  ng-click="send({payload:  '22'})"  class="thedays"><div
id="t22" class="theblocks"></div></td>
        <td  ng-click="send({payload:  '23'})"  class="thedays"><div
id="t23" class="theblocks"></div></td>
    </tr>
```

```
    <tr style="height:2px">
        <td id="td12" class="linea2px"></td>
        <td id="td13" class="linea2px"></td>
        <td id="td14" class="linea2px"></td>
        <td id="td15" class="linea2px"></td>
        <td id="td16" class="linea2px"></td>
        <td id="td17" class="linea2px"></td>
        <td id="td18" class="linea2px"></td>
        <td id="td19" class="linea2px"></td>
        <td id="td20" class="linea2px"></td>
        <td id="td21" class="linea2px"></td>
        <td id="td22" class="linea2px"></td>
        <td id="td23" class="linea2px"></td>
    </tr>

    <tr>
        <td><center>12</center></td>
        <td><center>13</center></td>
        <td><center>14</center></td>
        <td><center>15</center></td>
        <td><center>16</center></td>
        <td><center>17</center></td>
        <td><center>18</center></td>
        <td><center>19</center></td>
        <td><center>20</center></td>
        <td><center>21</center></td>
        <td><center>22</center></td>
        <td><center>23</center></td>
    </tr>

    <tr height="20px">
        <td  colspan=2  bgcolor="#dddddd"><center><span  id="status"
class="smallheadings" >Status:</span></center></td>

        <td  colspan=10  bgcolor="#dddddd"><center><span  id="info"
class="smallheadings" >Ok</span></center></td>

    </tr>

    <tr height="10px">
        <td colspan="12"></td>
    </tr>
```

```
    <tr style="height:48px">
        <td colspan=2>
            <center>
            <md-button class="greybuttons" ng-click="send({payload:
'izquierda'})">
                <img style="outline : none;"
                                ng-src="{{(msg.payload)?'/chevron-
circle-left-solid.svg':'/chevron-circle-left-solid.svg'}}"
                                height="36px"
                />
            </md-button>
            </center>
        </td>

        <td colspan=2>
            <center>
            <md-button  class="greybuttons" ng-click="send({payload:
'derecha'})">
                <img style="outline : none;"
                                ng-src="{{(msg.payload)?'/chevron-
circle-right-solid.svg':'/chevron-circle-right-solid.svg'}}"
                                height="36px"
                />
            </md-button>
            </center>
        </td>

        <td colspan=2>
            <center>
            <md-button  class="greybuttons" ng-click="send({payload:
'copiardia'})">
                <img style="outline : none;"
                                ng-src="{{(msg.payload)?'/calendar-
plus-solid.svg':'/calendar-plus-solid.svg'}}"
                                height="36px"
                />
            </md-button>
            </center>
        </td>

        <td colspan=2>
            <center>
            <md-button  class="greybuttons" ng-click="send({payload:
'copiarsemana'})">
                <img style="outline : none;"
                                ng-src="{{(msg.payload)?'/calendar-
alt-solid.svg':'/calendar-alt-solid.svg'}}"
                                height="36px"
                />
            </md-button>
            </center>
        </td>
```

```
        <td colspan=2>
            <center>
            <md-button  class="greybuttons" ng-click="send({payload:
'guardar'})">
                <img style="outline : none;"
                                ng-src="{{(msg.payload)?'/calendar-
check-solid.svg':'/calendar-check-solid.svg'}}"
                                 height="36px"
                />
            </md-button>
            </center>
        </td>

        <td colspan=2>
            <center>
            <md-button  class="greybuttons" ng-click="send({payload:
'refrescar'})">
                <img style="outline : none;"
                                ng-src="{{(msg.payload)?'/calendar-
times-solid.svg':'/calendar-times-solid.svg'}}"
                                 height="36px"
                />
            </md-button>
            </center>
        </td>

    </tr>
    </table>
</body>
```

Código del nodo “Control de calendario”:

```
if ( typeof context.days == 'undefined' ) context.days=0;
if ( typeof context.selector == 'undefined' ) context.selector=24;
if ( typeof context.saving == 'undefined' ) context.saving=0;
if ( typeof context.global.calendario == 'undefined' )
    {
        context.global.calendario=[

false,false,false,false,false,false,false,false,false,false,false,f
alse,false,false,false,false,false,false,false,false,false,false,fa
lse,false,

false,false,false,false,false,false,false,false,false,false,false,f
alse,false,false,false,false,false,false,false,false,false,false,fa
lse,false,

false,false,false,false,false,false,false,false,false,false,false,f
alse,false,false,false,false,false,false,false,false,false,false,fa
lse,false,

false,false,false,false,false,false,false,false,false,false,false,f
alse,false,false,false,false,false,false,false,false,false,false,fa
lse,false,

false,false,false,false,false,false,false,false,false,false,false,f
alse,false,false,false,false,false,false,false,false,false,false,fa
lse,false,

false,false,false,false,false,false,false,false,false,false,false,f
alse,false,false,false,false,false,false,false,false,false,false,fa
lse,false,

false,false,false,false,false,false,false,false,false,false,false,f
alse,false,false,false,false,false,false,false,false,false,false,fa
lse,false,
                        ];
        guardado=0;
    }

var calendario=global.get("calendario");
var outputMsgs =[];
```

```
switch (msg.payload)
    {
    case 'izquierda' :   if (context.days>0) context.days--; else
context.days=6;
                msg.foryou="Danterior.";
    break;

    case 'derecha' :    if (context.days<6) context.days++; else
context.days=0;
                msg.foryou="Dsiguiente.";
    break;

    case 'copiardia' :

        if (context.days<6)
        {
            for (var i=0;i<24;i++)
            {

calendario[((context.days+1)*24)+i]=calendario[((context.days)*24)+
i];
            }
            context.days++;
        }
        else
        {
            for (var i=0;i<24;i++)
            {
                calendario[i]=calendario[((context.days)*24)+i];
            }
            context.days = 0;
        }
        msg.foryou="Configuraci□uardada al siguiente d";
    break;

    case 'copiarsemana' :

        for(var i = 0; i<7; i++)
        {
            for (var j=0;j<24;j++)
            {

calendario[(i*24)+j]=calendario[((context.days)*24)+j];
            }
        }
        context.days = 0;
        msg.foryou="Configuraci□uardada para toda la semana.";
    break;
```

```
    case 'guardar': msg.foryou="Configuraci□uardada.";
                var
thedays=["LUNES","MARTES","MIERCOLES","JUEVES","VIERNES","SABADO","
DOMINGO"];
                var cuentadias = -1;
                    for(var i = 0; i<145; i=i+24){
                        cuentadias++;
                    outputMsgs.push({
                        topic:  "UPDATE  `calendario`  SET  `h0`  =
"+context.global.calendario[i]+",               `h1`               =
"+context.global.calendario[i+1]+",              `h2`               =
"+context.global.calendario[i+2]+",              `h3`               =
"+context.global.calendario[i+3]+",              `h4`               =
"+context.global.calendario[i+4]+",              `h5`               =
"+context.global.calendario[i+5]+",              `h6`               =
"+context.global.calendario[i+6]+",              `h7`               =
"+context.global.calendario[i+7]+",              `h8`               =
"+context.global.calendario[i+8]+",              `h9`               =
"+context.global.calendario[i+9]+",             `h10`               =
"+context.global.calendario[i+10]+",             `h11`              =
"+context.global.calendario[i+11]+",             `h12`              =
"+context.global.calendario[i+12]+",             `h13`              =
"+context.global.calendario[i+13]+",             `h14`              =
"+context.global.calendario[i+14]+",             `h15`              =
"+context.global.calendario[i+15]+",             `h16`              =
"+context.global.calendario[i+16]+",             `h17`              =
"+context.global.calendario[i+17]+",             `h18`              =
"+context.global.calendario[i+18]+",             `h19`              =
"+context.global.calendario[i+19]+",             `h20`              =
"+context.global.calendario[i+20]+",             `h21`              =
"+context.global.calendario[i+21]+",             `h22`              =
"+context.global.calendario[i+22]+",             `h23`              =
"+context.global.calendario[i+23]+"       WHERE       `dia`       =
'"+thedays[cuentadias]+"' ",

                    });
                    }
                    var guardado = 1;
    break;
```

```
    case 'refrescar' :
        msg.topic = "SELECT * FROM calendario";
        msg.foryou="Cambios cancelados.";
        node.send([null,null,msg]);
    break;

    case '0':
    case '1':
    case '2':
    case '3':
    case '4':
    case '5':
    case '6':
    case '7':
    case '8':
    case '9':
    case '10':
    case '11':
    case '12':
    case '13':
    case '14':
    case '15':
    case '16':
    case '17':
    case '18':
    case '19':
    case '20':
    case '21':
    case '22':
    case '23':
    case '24': context.selector=parseInt(msg.payload);
               if        (msg.payload=='24')        msg.foryou="Dia
seleccionado.";
               else  {
                   msg.foryou="Hora " + (parseInt(msg.payload)) + "
seleccionada.";
                       if
(calendario[((context.days)*24)+context.selector] === false)
                       {

calendario[((context.days)*24)+context.selector] = true;
                       }
                       else                                        if
(calendario[((context.days)*24)+context.selector] === true)
                       {

calendario[((context.days)*24)+context.selector] = false;
                       }
                       }
    break;
    }
msg.calendario=calendario;
msg.days=context.days;
msg.selector=context.selector;
node.send([msg,null,null]);
if(guardado == 1){
    guardado = 0;
    node.send([null,outputMsgs]);
}
    msg.foryou=""
```

8 Anexo B: Detalle del flujo principal de programa del flow Home Monitoring

Para un mejor entendimiento y facilidad a la hora de seguir el código descrito en la sección: Home Monitoring flujo principal se incluyen unas imágenes más detallas con un formato de página con orientación horizontal.

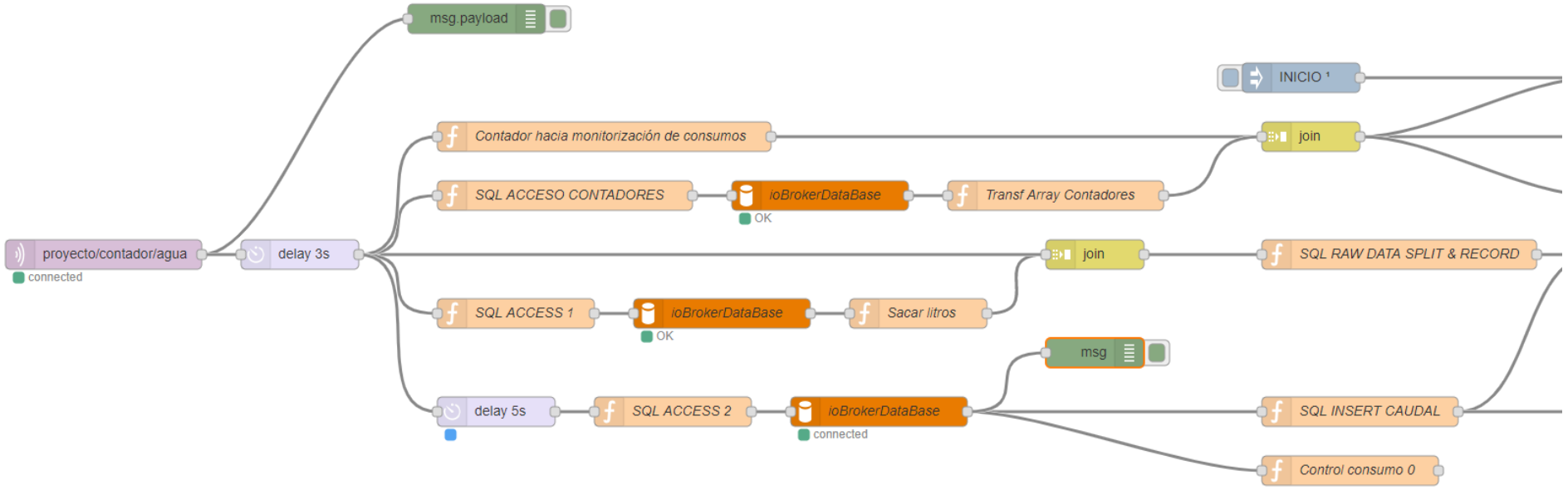

FIG. 79 - HOME MONITORING 1 NODE-RED.

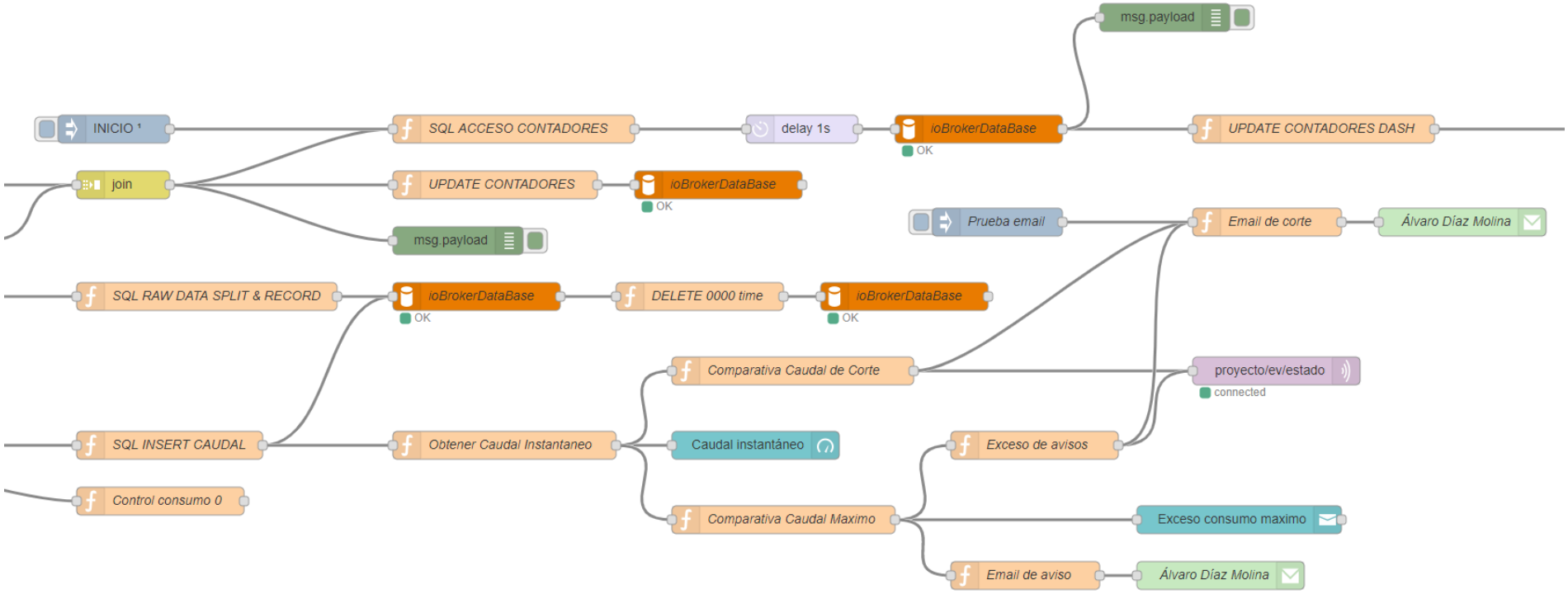

FIG. 80 - HOME MONITORING 2 NODE-RED.

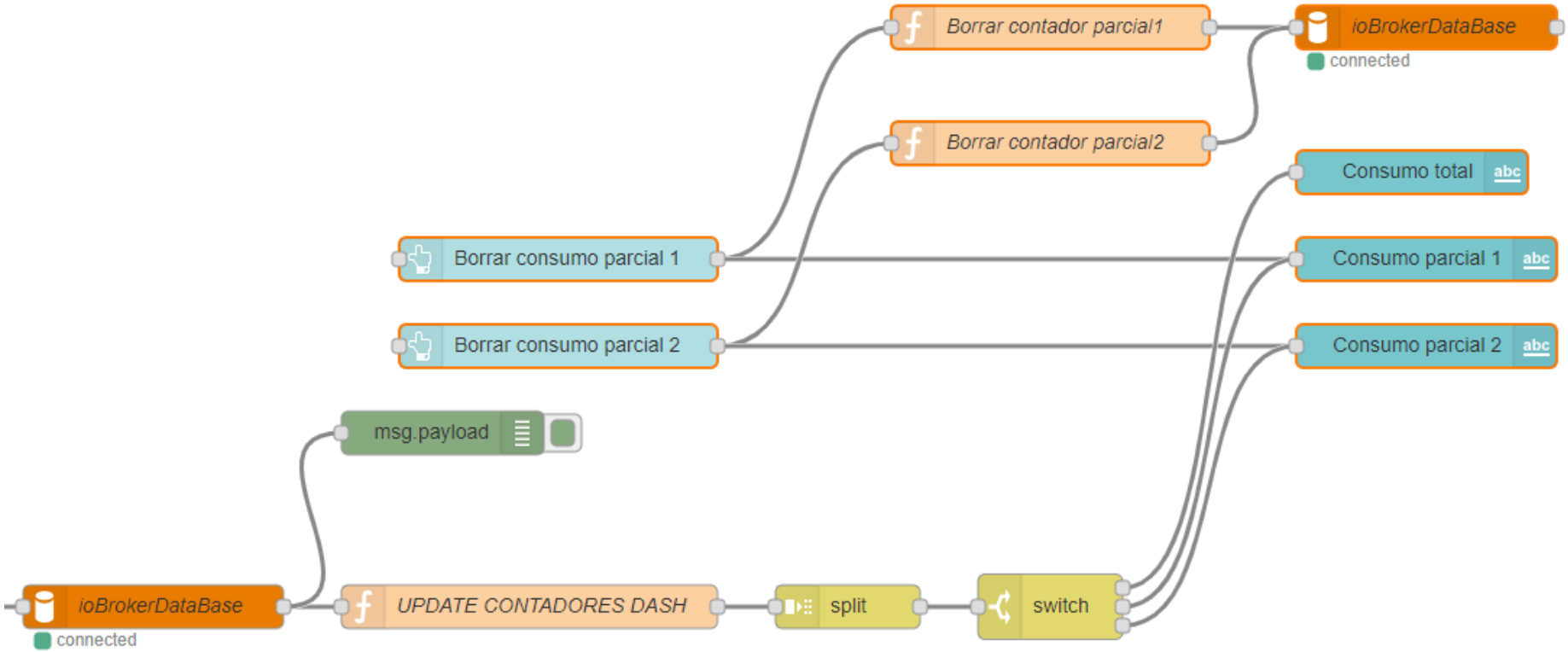

FIG. 81 - HOME MONITORING 3 NODE-RED