UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

Diseño de una plataforma en la nube con Node-RED para Internet de las cosas

Titulación: Grado en Ingeniería en Tecnologías de la Telecomunicación
Autor: D. Carlos Espacios López
Tutores: D. Miguel Ángel Quintana Suárez
D. David de la Cruz Sánchez Rodríguez

Fecha: Enero 2018

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

Diseño de una plataforma en la nube con Node-RED para Internet de las cosas
HOJA DE FIRMAS

Alumno

Fdo.: Carlos Espacios López

Tutor **Tutor**

Fdo.: D. Miguel Ángel Quintana Suárez Fdo.: D. David de la Cruz Sánchez Rodríguez

Fecha: Enero 2018

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

Diseño de una plataforma en la nube con Node-RED para Internet de las cosas

HOJA DE EVALUACIÓN

Calificación: _____

Presidente

Fdo.:

Vocal **Secretario/a**

Fdo.: Fdo.:

Fecha: Enero 2018

ÍNDICE DE CONTENIDOS

ÍNDICE DE ILUSTRACIONES

Parte I

Memoria descriptiva

Capítulo 1. Introducción

1.1 Antecedentes

Este proyecto surge tras la tendencia vista hoy en día de conectar todo lo que nos rodea de manera virtual: la educación, la administración, el ocio, la comunicación, el trabajo, nuestros propios hogares, etc. Para poder tenerlo en cualquier momento, y en cualquier lugar.

A lo largo de la historia, el ser humano se ha querido comunicar con otras personas. Primero con lenguajes de gestos, después con dialectos primitivos y así, sucesivamente, hasta lenguajes comunes. Pero esta comunicación no sólo se ha intentado en distancias cortas, sino también a distancias medias y largas. Casos como las señales de humo, mensajeros, palomas mensajeras etcétera son sólo algunos de los ejemplos que ha utilizado la humanidad para poder comunicarse atravesando ríos, montañas, fronteras…

Ya desde hace unos siglos hasta hoy en día, se han conseguido comunicaciones más precisas, seguras y eficaces por medio de la tecnología, teniendo como máximos exponentes el sistema Morse, la radio y el teléfono. Incluso hoy en día la televisión es un medio bidireccional de comunicación en algunos aspectos.

Pero hay otro medio posterior para transmitir información, mucho más universal que todos los demás: Internet. Es el medio por antonomasia que permite conectarse con cualquier parte del mundo. Este medio y la infraestructura que existe (como por ejemplo, la fibra óptica) ha permitido que a día de hoy se puedan realizar envíos/recepciones de grandes cantidades de datos de manera casi instantánea, siendo irrelevante la parte del mundo donde este el emisor/receptor.

Este afán de necesitar la información en periodos ínfimos de tiempo y el medio que se dispone con Internet son los principales pilares en los que se basa el concepto del ***Internet of the Things*** [1] (desde ahora, ***IoT***).

IoT es un concepto que se basa en la conexión de máquinas automáticamente a Internet para distintos fines, sobre todo la recopilación de datos de cualquier índole. La gran mayoría de estas máquinas son simples sensores que recopilan distintas informaciones para que alguien o algo hagan uso de ellas.

Ya en 1926 [2], en una entrevista a la revista *Colliers*, **Nikola Tesla** (uno de los mayores descubridores de la historia, entre otras cosas, de las comunicaciones inalámbricas) pronosticó esta tecnología:

"Cuando lo inalámbrico esté perfectamente desarrollado, el planeta entero se convertirá en un gran cerebro, que de hecho ya lo es, con todas las cosas siendo partículas de un todo real y rítmico… y los instrumentos que usaremos para ellos serán increíblemente sencillos comparados con nuestros teléfonos actuales. Un hombre podrá llevar uno en su bolsillo"

Inicialmente, Internet no existía como tal, sino que se denominaba ARPANET, que se usaba para uso académico y militar exclusivamente en EEUU en los años 60-70, donde se generaron los primeros protocolos de comunicación. No fue hasta los 90 donde se hizo una versión universal y comercial de Internet por medio del archiconocido protocolo TCP/IP, que creó un estándar general para las comunicaciones entre distintas redes.

Antes de esta versión, ya se había creado el primer objeto conectado a la red. En el evento **Interop**, en EEUU, **John Romkey**, con la ayuda de Simon Hackett, presentó una tostadora que se podría apagar y encender remotamente usando *TCP/IP* para la conectividad y *SNMP* para el control de estas funciones. El siguiente año, presentó el mismo mecanismo pero con una mano robótica integrada, también con funcionamiento remoto, que insertaba en la tostadora una rebanada de pan, haciendo aún más automático el proceso.

Más adelante, en el inicio del siglo XXI, la tecnología avanzó creando las redes inalámbricas, proporcionando teléfonos móviles o el protocolo ***WiFi***. Estas tecnologías permitieron la creación de muchos tipos de dispositivos que, al igual que hizo la tostadora en de John Romkey en 1990, funcionarían por medio de Internet mediante uso remoto y con la creación en años posteriores de la tecnología ***M2M*** (*Machine-to-Machine*), entre otras, se obtiene el ***IoT*** inicialmente explicado.

Con un punto de vista más técnico, se observa la oportunidad de facilitar la transmisión y recepción por medio de Internet de datos suministrados por sensores a cualquier distancia del usuario, dándole una mayor comodidad en tiempo y forma, ya que todos los datos se quedarán almacenados en una base de datos a la cual tendrán acceso diferentes usuarios de forma remota. Actualmente, hay plataformas que proporcionan este tipo de servicios, como son ***Amazon Web Services***[3], ***Microsoft Azure Iot Suite***[4], ***freeboard.io*** [5], ***IBM Bluemix***[6], entre otras.

La mayor parte de estos entornos de gestión de la información generados por sensores para Internet de las cosas se basan en plataformas web. Para el diseño de entornos web existe la posibilidad de elegir una gran variedad de lenguajes ó frameworks como ***PHP***,

Java, ***Python***, etc. Uno de los frameworks con más proyección son los basados en ***Node.js*** [8] por tener ventajas tales como que puedes usar el mismo lenguaje tanto en servidor como en cliente (***Javascript***), poder tener en el servidor toda la funcionalidad necesaria (acceso a ficheros, a bases de datos, conexiones con clientes...). Además, se disponen de entornos gráficos de desarrollo en node.js que aprovechan el modelo de diseño basado en flujos. El más ampliamente conocido es ***Node-RED*** [9]

Node-RED provee una interfaz basada en un navegador web, permitiendo crear flujos de eventos e interconectarlos todos ellos a través de un ligero entorno de trabajo y desarrollo. Está construido en *Node.js*, lo que le permite funcionar justo al borde de la red o en la nube, dotándole de flexibilidad.

Hay diversas formas de enviar información bajo redes en la nube. Se pueden encontrar desde protocolos de transferencia de archivos donde funcionan como un cliente-servidor, hasta comunicaciones entre equipos conocedores de sus direcciones para entablar conexiones de envío de datos en ambas direcciones usando, por ejemplo, sockets. Entre todas las posibles variedades de hacer “simplemente” eso, enviar información a través de la red, existe el protocolo ***MQTT*** (*Message Queue Telemetry Transport*) [10].

MQTT es un protocolo usado para la comunicación M2M en el denominado ***IoT***. Este protocolo está orientado a la comunicación de sensores, debido a que consume muy poco ancho de banda y puede ser utilizado en la mayoría de los dispositivos empotrados con pocos recursos (CPU, RAM, etc). Esta comunicación se puede cifrar entre otras opciones.

Este protocolo es el que se usará para la comunicación intermedia entre la plataforma y los sensores. Sin embargo, este protocolo necesita de un servidor “*broker*” o intermediario que enlace las fuentes y destinos de la información. Existen diferentes implementaciones del broker, dando algunas más soporte que otras.

Como se puede apreciar las plataformas *Amazon Web Services* y *Microsoft Azure Iot Suite* están basadas en *Node-RED*, utilizando *MQTT* en un servidor virtualizado en la nube. En este proyecto se pretende desarrollar una nube propia. Es decir, proporcionar un entorno similar a los descritos pero basado en sistemas locales. Para ello serán necesarios conocimientos de gestión y administración de sistemas operativos, programación en *JavaScript*, *Node.js* y *Node-RED*, gestión de bases de datos, *HTML5*, *MQTT*, entre otros.

1.2 Objetivos

El objetivo general de este proyecto es el desarrollo de una plataforma en la nube donde el usuario pueda tener acceso a datos proporcionados por sensores y elementos en la red.

Este objetivo general se divide en los siguientes objetivos específicos:

O1: Análisis de los diferentes componentes y protocolos que conformarán la plataforma.

O2: Adquirir conocimientos de *Node.js*, *Node-RED* y *JavaScrip*t.

O3: Trabajar en el uso del protocolo *MQTT* para la comunicación de datos recibidos por sensores.

O4: Gestionar de manera simultánea un *front-end* y un *back-end*.

1.3. Peticionario

Tras haber superado satisfactoriamente las asignaturas especificadas en el Grado en Ingeniería en Tecnologías de la Telecomunicación, con mención en Telemática, impartidas por la Escuela de Ingeniería de Telecomunicación y Electrónica de la Universidad de Las Palmas de Gran Canaria, se solicita, como requisito indispensable para la obtención del título de Graduado en Ingeniería en Tecnologías de la Telecomunicación, el desarrollo, la redacción, la exposición y la defensa de un Trabajo Fin de Grado.

1.4 Organización del trabajo

En este TFG se ha seguido una cierta organización por fases con el fin de continuar una metodología de trabajo:

- Fase 1. Análisis del mercado: El primer paso ha sido analizar las diferentes tecnologías que se podían usar para llevar a cabo este proyecto. En el caso de ya tener una tecnología escogida para trabajar, como fue el caso de *Node-RED*, se analizaron otras tecnologías también para comprobar lo que ofrecían y para tener un modelo de lo que se quería realizar.
- Fase 2. Instalación, configuración y pruebas. Una vez teniendo las tecnologías con las que se quería trabajar, se completó su instalación y configuración. Una vez hecho esto, se realizaron distintas pruebas con la finalidad de conocer la tecnología de primera mano.

- Fase 3. Diseño de los componentes. Ya con una familiarización gracias a las pruebas realizadas con las tecnologías escogidas, se determinó el diseño que iba a tomar cada componente de la plataforma por separado.
- Fase 4. Cohesión de los componentes. Última parte del TFG, donde se unieron todos los componentes formando la plataforma, previo solvento de los problemas que surgieron al unir las diferentes partes.

El presente documento está diferenciado en tres partes: la memoria descriptiva, presupuesto y anexos. La memoria está compuesta por 5 capítulos, que documentan las fases anteriormente explicadas. Obviando el primer capítulo, que es el actual, el segundo capítulo describe las diferentes tecnologías más relevantes observadas en el mercado para cada parte de la plataforma, con una comparación entre cada una de ellas y la elección de una de ellas. El tercer capítulo ofrece una versión más detallada cada tecnología seleccionada en el capítulo anterior con ejemplos.

Ya el cuarto capítulo proyecta el diseño que tiene cada una de las partes de la plataforma y su cohesión con las demás para poder conformar la plataforma completa y funcional. Por último, en el quinto capítulo se presentan las conclusiones a las que se ha llegado y posibles líneas futuras que se podrían seguir.

Como anexos, se tienen dos anexos: El anexo I tiene las diferentes instalaciones que se han tenido que hacer para poder tener los componentes instalados, configurados y listos para ser usados. El anexo II es un manual de usuario para el usuario final que use esta plataforma.

Capítulo 2. Introducción de tecnologías actuales

En este segundo capítulo se explicarán las diferentes tecnologías que se utilizan para el desarrollo de una plataforma de *IoT*. Entre ellas, se analizarán las siguientes:

- El protocolo *MQTT* y sus componentes para la conexión entre los sensores externos y el resto de la plataforma.
- Una base de datos para almacenar los datos provenientes de los sensores.
- Herramienta que proporcione un entorno para realizar la programación necesaria con la que se las tecnologías anteriores, y que proporcione un front-end y un back-end para la gestión de datos.

Se analizarán las opciones que ofrece el mercado actual para los distintos componentes (ilustración 1), los cuales constituirán la plataforma que se quiere construir en este proyecto. Para ello, se ha estudiado el mercado actual con el fin de encontrar las mejores opciones posibles. Teniendo como criterios: el futuro de la tecnología, el gasto, la documentación de esta y la facilidad que tenga a la hora de trabajar con ella.

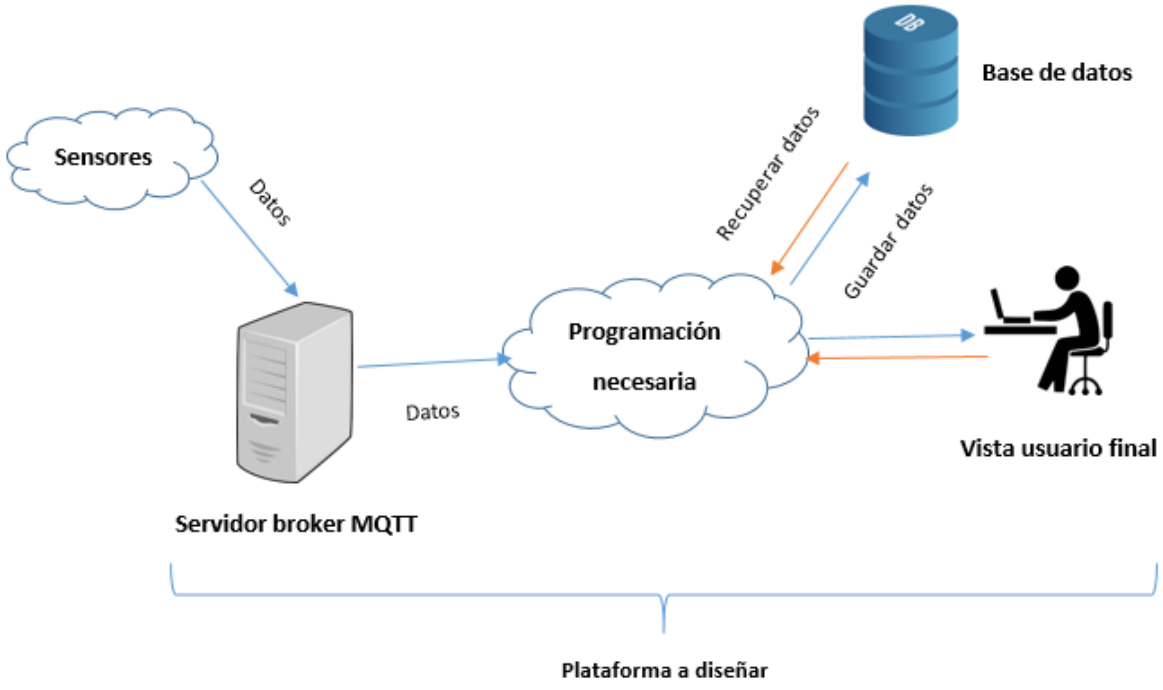

Ilustración 1 Esquema de la plataforma a diseñar

2.1 MQTT

MQTT es un protocolo desarrollado para la transmisión de datos entre máquinas. Está diseñado para comunicaciones donde los dispositivos en cuestión tengan recursos limitados y sus mensajes no necesiten grandes anchos de banda.

Este protocolo de comunicaciones funciona con un modelo de publicación/subscripción donde, los subscriptores "escuchan" los mensajes que publican las direcciones a las que se han suscrito. En el caso de este proyecto, los sensores harán el papel de publicadores de datos, mientras que dentro de la plataforma se tendrán los subscriptores para recopilar estos mensajes. MQTT es totalmente necesario para el funcionamiento de la plataforma, ya que ejerce el papel de intermediario entre los sensores y los demás componentes de la plataforma, permitiendo que estos puedan usar las muestras provistas por los sensores.

Para funcionar se debe de hacer uso de un servidor acorde a sus necesidades para el intercambio de mensajes, debido a que sin la infraestructura adecuada el protocolo por sí sólo no puede funcionar. En este caso hablamos de un servidor de tipo **broker.**

Los servidores broker se usan en entornos asíncronos donde los componentes son independientes, pero entre todos realizan una tarea conjunta. Para ello, se intercambian información por medio de mensajes. Esto genera un nivel de coordinación bastante útil en este tipo de entornos donde funciona de forma óptima *MQTT*.

Se tienen multitud de opciones en el mercado de servidores *broker*, tanto *open-source* como *premium* etc. La mayoría poseen las mismas funcionalidades, sin embargo, se distinguen en la capacidad de estas, ya sea por su sencillez, por ser dar entornos más gráficos o ser más escalables. A continuación, se describen las implementaciones más conocidas: ***EMQ***[12]**,** ***HiveMQ***[13] **y** ***Mosquitto***[14]**.**

2.1.1 EMQ - *Erlang MQTT Broker*

EMQ (ilustración 2) es de código abierto e implementa las especificaciones del protocolo MQTT V3.1 y V3.1.1, además de soportar *MQTT-SN*, *CoAP*, *WebSocket*, *STOMP* y *SockJS* simultáneamente.

Se pueden conectar dispositivos móviles, sensores, dispositivos M2M (entre otros, como *EMQ brokers,* trabajando con mensajes asíncronos del tipo publicación/suscripción mediante *MQTT*.

Ilustración 2 Logotipo EMQ

2.1.2 HiveMQ

HiveMQ (ilustración 2) es un *broker* de código abierto que implementa las *V3.1* y *V.3.1.1* del protocolo *MQTT*. Este *broker* está diseñado específicamente para *MQTT*, por lo tanto tiene diversas optimizaciones para este protocolo. Permite tener seguridad en los mensajes por medio de autenticación y autorización por medio de estándares como *SSL/TLS*.

Ilustración 3 Logotipo HiveMQ

2.1.3 Mosquitto

Mosquitto (ilustración 4) es un *broker* de código abierto que implementa las *V3.1* y *V3.1.1* del protocolo *MQTT*. En comparación con los anteriores, ofrecen unas limitaciones que los demás no tienen respecto a la escalabilidad, ya que no llega a tal nivel, pero *Mosquitto* trabaja de una manera más ligera y es mucho más popular (mejor a la hora de recopilar información). Además, aun teniendo desventajas respecto a la escalabilidad, cumple todas las funciones para *MQTT*, según su propio repositorio de *Github* [15], permitiendo tener

autenticación, cumplir el estándar SSL/TLS, funcionalidad con *websockets*, entre otras, siendo algunas de estas no cumplidas por otros *brokers* de mayor escalabilidad.

Como características se tiene que:

- El ejecutable del servidor tiene un tamaño de 120KB consumiendo alrededor de 3MB de RAM para 1000 clientes.
- Se ha llegado a tener pruebas exitosas con 100000 clientes en entornos de mensajes ligeros.

Ilustración 4 Logotipo Mosquitto

2.1.5 Elección del broker

Por su sencillez a la hora de trabajar y la cantidad de documentación que hay respecto a él, se ha optado por escoger *Mosquitto* como el servidor *broker* para soportar el protocolo *MQTT*.

2.2 Base de datos

La base de datos es uno de los componentes más importantes de la plataforma, ya que se ocupará del almacenamiento de los datos ofrecidos por las muestras de los sensores y de la identificación de estos y de sus autores (propietarios).

2.2.1 Bases de datos relacionales

Las bases de datos relacionales son las primeras que se crearon y, como indica su nombre, se basan en relaciones. Cada una de las relaciones equivaldría en la base de datos a una *table* (tabla), que estará compuesta por registros (cada fila de la tabla) y campos (cada columna).

Las tablas guardan información de distintos elementos, llegando a que las tablas se complementan entre sí. El nombre de las tablas no puede ser repetido ni tener el mismo registro. Los datos que se encuentran ubicados en las tablas pueden necesitar una clave.

SQL (o lenguaje de consulta estructurada) es la interfaz principal para trabajar con estas bases de datos. Gracias a él, se pueden hacer funciones como agregar, actualizar o eliminar filas de datos, recuperar datos…

Estas bases de datos se basan en que sus datos tengan integridad: se tiene la totalidad de ellos de manera precisa y con coherencia gracias al uso de unas reglas. Estas normas derivan en restricciones propias de estas bases de datos, como el uso de claves primarias, foráneas… Además, también se puede añadir un código personalizado para mejorar esta integridad.

Estas bases de datos, para llevar a cabo transacciones de elementos, deben cumplir los principios *ACID* (*Atomicity*, *Consistency*, *Isolation* y *Durability*).

Ejemplos de estas BD son las siguientes: *MySQL* [16], *Microsoft SWL Screen*…

Ventajas de una base de datos *SQL*

- ✓ A nivel de buscar compatibilidades, su uso está estandarizado.
- ✓ Son fáciles de instalar y configurar.
- ✓ Baja probabilidad de corromper datos, incluso si los errores no se producen en el propio gestor, sino en el sistema en el que está.
- ✓ Conectividad y seguridad.
- ✓ Los principios *ACID* implican que una operación no se puede quedar a medias, por lo que se realiza de manera completa o no se hace (uso de *rollback*).

Desventajas de una base de datos *SQL*

- × Algunas de las utilidades de estas bases de datos no están documentadas.
- × No es intuitivo.

- × Muchas de las operaciones en estas bases de datos tienen que realizarse de forma atómica, lo que puede suponer un gran problema a nivel de rendimiento.
- × Su escalabilidad suele ser menor en comparación a las bases de datos no relacionales.

2.2.2 Bases de datos *NoSQL* (*Not only SQL*)

Las bases de datos no relacionales almacenan la información de una manera distinta a las vistas anteriormente, debido a que no cumplen con esas relaciones. En lugar del uso de tablas para estructurar los datos, se usan sistemas como grafos, mapeo de columnas o mecanismos clave-valor. Debido a esto, no se pueden garantizar los principios ACID, pero en la gran mayoría de aplicaciones y sistemas no resulta imprescindible, ya que se valora muy por encima el rendimiento y la respuesta en tiempo real que la coherencia suministrada por *ACID* (la cual consume una cantidad ingente de recursos).

Estas bases de datos resuelven los problemas de escalabilidad y rendimiento de las anteriores cuando hay muchos usuarios concurrentes haciendo millones de consultas, como en aplicaciones del tipo *Facebook*, *Google*…

Las principales diferencias con los sistemas *SQL* son:

- En estas bases de datos *SQL* no se utiliza o se hace como apoyo.
- Los datos son estructurados sin usar tablas.
- Las operaciones *JOIN* no suelen ser usadas.
- Arquitectura distribuida.
- No se garantizan los principios *ACID.*

Ventajas de una base de datos *NoSQL*

- ✓ Son escalables y no están centralizadas, por lo que se pueden estructurar de forma distribuida.
- ✓ Las máquinas donde se ejecuten pueden ser de pocos recursos.
- ✓ Para evitar tener que usar grandes máquinas, se puede realizar la escalabilidad en número de máquinas.
- ✓ Están optimizadas para grandes consultas de datos de manera simultánea.
- ✓ Es posible cambiar las bases de datos sin necesidad de pararlas.
- ✓ Son mucho más flexibles a la hora de adaptarse a nuevas necesidades en los proyectos en comparación con las relacionales.

Desventajas de una base de datos *NoSQL*

- × Soportan la denominada consistencia eventual, pero no todas cumplen los principios *ACID* de atomicidad e integridad de datos.
- × No son totalmente compatibles con *SQL*, lo que provoca problemas con sus instrucciones de consulta.
- × Aún no existe una estandarización para las bases de datos *NoSQL,* como sí existe en las *SQL*.
- × Es bastante mejorable el soporte multiplataforma para que funcionen sin problemas en S.O. diferentes a *Linux*.

Uso de cada una de ellas

- En el caso de que los datos y las operaciones deban de ser consistentes sin ningún tipo de fallo (principios *ACID*): *SQL*.
- Si el presupuesto es bajo y no se pueden tener máquinas de alto rendimiento sino de bajo: *NoSQL*.
- En el caso de que sean variables las estructuras de datos: *NoSQL*.
- Si existe la necesidad de realizar consultas de grandes cantidades de datos en modo lectura: *NoSQL*.
- Si se están capturando y procesando eventos: *NoSQL*

2.2.3 *MySQL* VS *MongoDB*

A continuación, se comparará dos de las bases de datos más usadas hoy en día, *MySQL* y *MongoDB* [17]. Se podría decir que son las máximas representantes de las bases de datos relacionales, por el lado de *MySQL*, y no relacionales, por el lado de *MongoDB*.

2.2.3.1 *MySQL*

MySQL (ilustración 5) es una de las bases de datos relaciones *open-source* más conocida y usada, pudiéndola encontrar en la compañía Oracle. Al igual que otras, tiene la capacidad de almacenar los datos en tablas haciendo uso de *SQL* para su acceso.

Al estructurar una base de datos se tiene que tener en cuenta la predefinición de un esquema. Este esquema dependerá de:

- Requerimientos impuestos por el desarrollador.

- Reglas que participan entre el vínculo de los datos con las tablas y respecto a otras tablas. Se tendrá una duplicación de datos en estas tablas ínfima, se debe a su capacidad de almacenamiento en tablas separadas; que podrán ser asociadas por *joins*. Esto se puede conseguir pese a que la información esté relacionada.

Ilustración 5 Logotipo MySQL

2.2.3.2 MongoDB

Por contraposición, *MongoDB* (ilustración 6) es una de las bases de datos no relacionales (*NoSQL*) más distinguidas. Se puede encontrar en el mercado y fue hecha por *MongoDB Inc*. Una de sus principales ventajas es la capacidad de trabajar con distintos tipos de datos a la vez. Esto se puede lograr gracias a que *MongoDB* es capaz de almacenar los datos en archivos del tipo *JSON*.

Con el fin de la creación de archivos partiendo de los datos guardados, sin necesidad de predefinir una estructura (como en *MySQL*), en esta plataforma se hará uso de esquemas dinámicos. Una de las características más singulares es la posibilidad del cambio de estructura de estos archivos; con la adición de nuevos ficheros o la eliminación de alguno de ellos ya existente. Gracias a todo esto se podrán representar relaciones complejas, como por ejemplo jerárquicas, de una forma menos compleja. Para un acceso más rápido, la información relacionada será almacenada conjuntamente

Ilustración 6 Logotipo MongoDB

Existen analogías entre una y otra infraestructura, que se nombrarán en la tabla 1, y, también, se verán las diferencias que existen en la manera de gestionar los datos de una y otra.

MySQL	*MongoDB*
Tabla	*Colección*
Fila	*Documento*
Columna	*Campo*
Joins	*Documentos incrustados, enlaces*

Tabla 1 Analogías MySQL-MongoDB (extraída de la referencia)

2.2.3.3 Comparativa entre *MySQL* y *MongoDB*

Si se comparan las propias prestaciones de una y otra (tabla 2), para la mayoría de los casos, por su sencillez, adaptabilidad y escalabilidad, se decantará por las bases de datos *MongoDB*. Algunos ejemplos serían para usos móviles, análisis en tiempo real, pantallas sencillas, catálogos y el *IoT*. Sin embargo, esto no sería posible si se tuviera que gestionar transacciones complejas, pero ese no es el caso de este proyecto e implicará que se decante por el uso de *MongoDB* en él:

Característica	*MySQL*	*MongoDB*
Modelo rico en datos	*No*	*Sí*
Esquemas dinámicos	*No*	*Sí*
Typed Data	*Sí*	*Sí*
Localización de datos	*No*	*Sí*
Field Updates	*Sí*	*Sí*
Fácil para los programadores	*No*	*Sí*
Transacciones complejas	*Sí*	*No*
Auditing	*Sí*	*Sí*
Auto-Sharding	*No*	*Sí*

Tabla 2 Comparativa MySQL vs MongoDB (extraída de la referencia)

Nota: Esto no quiere decir que uno sea incompatible con otro, ya que se pueden usar de forma conjunta en sistemas híbridos.

2.2.4 Elección base de datos

Debido a que la plataforma que se desarrollará va a necesitar ser muy escalable, tener mucha afluencia de datos y no necesitar de transacciones complejas, se optará por el uso de la base de datos *NoSQL* ***MongoDB***.

2.3 Entornos de gestión de datos

Una vez se tienen los componentes necesarios para la recopilación de datos y su almacenamiento, se necesita una infraestructura que conecte unos componentes con otros. Para esta infraestructura se tienen diversas plataformas de las que hacer uso, ya generadas y terminadas como ***AWS IoT Core*** (*Amazon Web Services IoT Core*) y otras que dan el soporte básico para que el desarrollador pueda realizar el diseño propio por su cuenta como ***Node-RED***.

2.3.1 AWS IoT Core

AWS IoT Core es una plataforma creada por Amazon ubicada en la nube. Ofrece la capacidad para interconectar distintos tipos de dispositivos con hardware o software directamente e interactuar con ellos. Dado que es una plataforma de pago, está totalmente terminada y configurada para que el usuario pueda tener tantos dispositivos como desee y con un flujo de mensajes casi ilimitado con fiabilidad y seguridad. Plataformas similares a esta son *Microsoft Azure Iot Suite*, *IBM Bluemix*, entre otras nombradas en el capítulo anterior.

2.3.2 Node-RED

Node-RED es un entorno gráfico *open source* que está basado en *Node.js.* Su uso ofrece una gran variedad de módulos interconectables con funciones preestablecidas de diversos tipos, cómo trabajar con mensajes que provengan desde servidores que estén usando el protocolo *MQTT*. Esto hace que *Node-RED* se posicione como una solución a la hora de plantear cual plataforma usar en este proyecto.

A la hora de tener que programar en *Node-RED*, más allá de la funcionalidad que ofrezcan sus módulos, se debe de hacer uso de JavaScript para la realización de funciones más específicas que se quieran implementar en la plataforma, ya que *Node-RED* se basa en *Node.js*, que a su vez se basa en *JavaScript*.

2.3.3 Elección del entorno de gestión de datos

En este Trabajo se busca trabajar con software libre y el aprendizaje de programación en distintos lenguajes, por lo tanto, se optará por la vía del entorno *Node-RED*.

2.4 Metodologías para generación de vistas de usuario.

A la hora de realizar vistas de usuario, hay metodologías de trabajo con las que se realiza software de una manera más sencilla y reutilizable, ahorrando bastante trabajo al programador. Para ello, existen metodologías donde el código depende del *framework* con el que se trabaje, y hay otras que funcionan de manera genérica. Se debe elegir con qué punto de vista se quiere trabajar y plasmar las ideas previas en los flujos de *Node-RED* para crear las vistas de usuario. Esto no quiere decir que no se pueda generar código mixto, puesto a que no llegan al punto de ser incompatibles, pero son puntos de vista distintos respecto a cómo trabajar. Actualmente, las dos tecnologías que sobresalen en este aspecto son las siguientes: trabajar con ***Components específicos de cada framework*** y trabajar con ***Web Components*** [18].

2.4.1 *Components específicos*

Los C*omponents* son módulos de código reutilizable, siendo el diseño y el desarrollo del software de tipo modular. Esto implica la facilidad de mantenerlo y reutilizarlo, donde existe una menor probabilidad de error, ya que el código, al estar en módulos, se aísla, se simplifica y se puede comprobar de una manera más sencilla donde están los fallos. Cada *framework* (ya sea ***Angular 2*** [19], ***React*** , ***JQuery*** …) tiene especificado como deben de ser diseñados los *Components* para que funcionen en ellos, por lo que trabajan de una forma óptima con ellos pero no son reutilizables por otros *frameworks* de manera sencilla.

2.4.2 *Web Component*

Se lleva el concepto anterior al siguiente nivel, haciendo que los propios navegadores los usen de forma nativa. Al hacerlo así, los *Web Components* generan un grado de reutilización que los *frameworks*, los cuales usan *Components,* no pueden llegar. Esto se logra al poder encapsular módulos enteros de funcionalidad en elementos ***DOM*** (*Document Object Model*), que son la unidad más básica de un *HTML*.

A día de hoy, casi todos los navegadores (por no decir todos) ya tienen implantados los *Web Components* como nativos y, en el caso de que no se tengan, se puede añadir ciertas líneas para que sean funcionales en estos navegadores (los cuales estarán ciertamente obsoletos).

Para la realización de una comparativa, se escogerán uno de los *frameworks* más potentes que usa *Components* y una de las librerías más utilizadas para la creación de Web *Components*: ***Angular 2*** y ***Polymer*** [20].

2.4.3 *Angular 2* vs *Polymer*

2.4.3.1 *Angular 2*

Angular 2 (ilustración 7), como su predecesor *Angular*, es uno de los *frameworks* de *JavaScript* más conocidos e influyentes que hay en el mercado actual. Ambos están diseñados bajo el concepto de *Components*, aunque *Angular 2* también puede ser configurado para trabajar con *Web Components,* pero no está diseñado para eso.

Aparte de poder realizar el diseño de los *Components*, *Angular 2* tiene la capacidad de ofrecer soluciones para realizar el enrutamiento como el manejo de estados en nuestro código.

Ilustración 7 Logotipo Angular 2

2.4.3.2 *Polymer*

Esta biblioteca optimiza todo el potencial de los *Web Components,* debido a que está destinada para ello. Diferenciándose de *Angular 2*, *Polymer* (ilustración 8) está diseñado para utilizar las funciones de la propia plataforma web y, bajo ella, crear los *Components* y no al revés. Gracias a esto, los *Web Components* se usan de una forma más sencilla y liviana que otros *frameworks*, debido a que están basados en los propios navegadores.

Ilustración 8 Logotipo Polymer

2.4.3.3 Comparativa entre *Polymer* y *Angular 2*

- **Tamaño**: *Polymer* es mucho más ligero que *Angular 2*. *Polymer* ocupa 127KB si el navegador es actual y tiene los *Web Components* como nativos, en caso contrario, ocupará 168KB al añadir el paquete *webcomponents-lite.js*. En el caso de *Angular 2* se tienen como mínimo 566KB en su versión simplificada hasta los 766KB con la librería *Rxjs*. Además, los desarrolladores de *Polymer* trabajan bajo el objetivo de que cada vez sea más ligero, con forme los navegadores web sean más potentes.

- **Reusabilidad**: los *Web Components* generados con *Polymer* se pueden utilizar en todo tipo de aplicaciones. Mientras que los generados bajo *Angular 2,* solo funcionarán en aplicaciones hechas en *Angular 2*.

- **Longevidad de código**: los *Web Components* son el siguiente paso de los *Components*, por lo tanto, *Polymer* va un paso por delante respecto a *Angular 2*. Este último se basa en *Components* y, aunque puede funcionar con los *Web Components*, no está diseñado para ellos y deben de hacerse diversos pasos previos para su adaptación.

2.4.3.4 Elección entre Angular y Polymer

En el punto anterior se han visto las principales diferencias que se pueden encontrar entre uno y otro, debido a que en lo demás son, a grandes rasgos, similares. Pero las diferencias influyen en la elección de la librería *Polymer* y el concepto de los *Web Components* para su uso en este proyecto, en lugar de usar *Angular 2* y sus *Components* específicos.

Capítulo 3. Análisis de tecnologías usadas

En este tercer capítulo se analizarán las tecnologías usadas en este proyecto en profundidad. Inicialmente se abordará la explicación del protocolo TCP/IP MQTT, ya que será usado a lo largo de este Trabajo, así como el broker MQTT Mosquitto que servirá como infraestructura de este protocolo. A continuación, se detallará la base de datos *MongoDB*, el entorno gráfico de programación Node-RED, y los Web Components.

3.1 MQTT

MQTT es un protocolo abierto que se usa para las comunicaciones *M2M* (*machine-to-machine*) en el *IoT*. Sus creadores fueron Andy Stanford-Clark de *IBM* y Arlen Nipper de *Arcom* (actual *Eurotech*) en 1999. La versión estable más actual es *MQTT V3.1.1*.

Su poco consumo de ancho de banda y poder funcionar en dispositivos que recursos limitados (*RAM*, *CPU*…) hace que sea una opción ideal para la comunicación entre sensores.

Características:

- Protocolo de mensajería de publicación / suscripción ligero implementándose en servidores *broker*
- Diseñado para ser abierto, sencillo, ligero y fácil de implementar
- Ideal para uso en entornos restringidos, pero no por ello sólo útil en estos entornos
- Buen uso en dispositivos con procesador limitado o memoria
- Buen uso cuando la red es costosa, tiene poco ancho de banda o no es fiable
- Uso de *TCP/IP* para proveer una básica conectividad de red
- Tres tipos de calidad de servicio.

3.1.2 Funcionamiento MQTT

El funcionamiento de *MQTT* es el que se ha nombrado en el capítulo anterior: Se basa en un modelo de publicación/subscripción, en donde los subscriptores esperan de determinada dirección mensajes que tengan determinado *topic*. El topic es el identificativo para diferenciar mensajes a una misma dirección. Por ejemplo, un sensor puede tener dos lecturas, pero una de ellas no activarse. Entonces, el topic será distinto respecto a que las dos estuvieran activas. Se da el siguiente ejemplo (ilustración 9):

Por un lado, se tienen determinados sensores que publican sus datos a la dirección donde está ubicado el *broker MQTT.* Estos sensores envían los datos que tengan que enviar bajo un determinado *topic* que los identificará. Para este ejemplo se usan dos sensores, uno de temperatura y otro de tensión, donde el *topic* será el tipo de dato que están enviando.

Por el otro lado, se tienen los subscriptores, que pueden ir desde ordenadores hasta terminales móviles. Para realizar la subscripción, el equipo debe de entablar conexión con el servidor *broker* y comunicar qué *topic* desea recibir. Una vez hecho esto, si la conexión no expira o se cierra, el equipo recibirá todos los datos que lleguen con ese campo *topic* determinado al servidor *broker* (siempre que se le haya dado permiso desde el *broker* para recibirlos en el caso de que se necesiten determinadas claves) hasta el fin de la conexión.

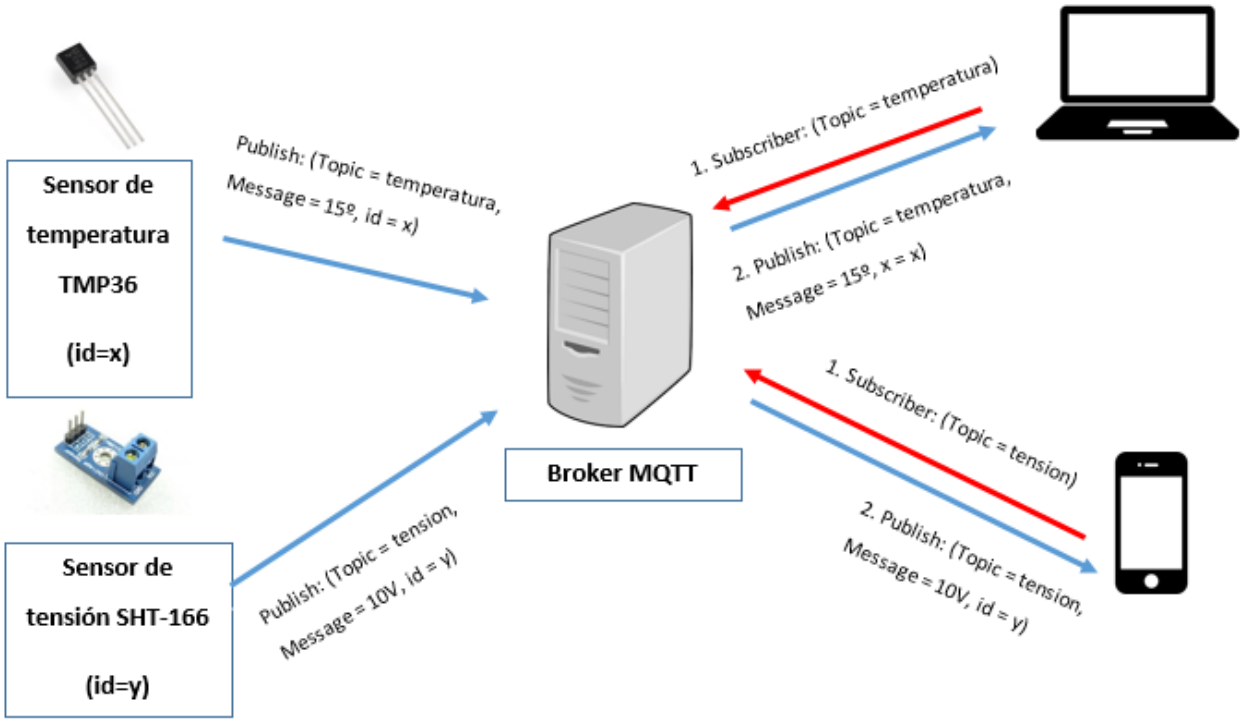

Ilustración 9 Ejemplo de funcionamiento de un servidor broker MQTT

El formato general para el intercambio de mensajes entre un cliente y un broker *MQTT*, es el visto en la ilustración 10. (Figuras sacadas de [21]).

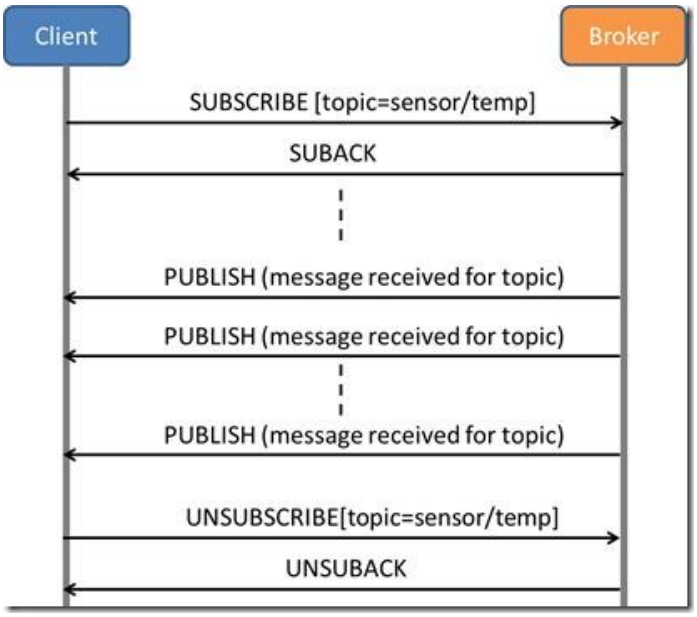

Ilustración 10 Intercambio general de mensajes entre cliente y broker MQTT

Una vez dicho esto, dependiendo de la *QoS* que se tenga en cada sistema, será de un tipo u otro el intercambio de mensajes. Si el nivel de *QoS* es 0, el intercambio de mensaje es, simplemente, el publicador inicial publica su mensaje en el *broker*, y este lo reenvía a los subscriptores (ilustración 11). El publicador inicial no guarda en ningún momento su mensaje inicial.

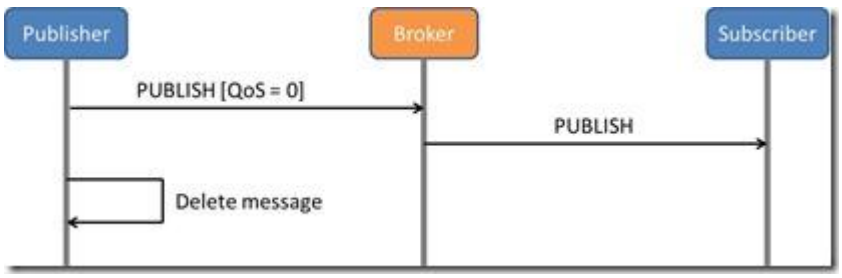

Ilustración 11 Intercambio de mensajes entre publisher-broker-subscriber MQTT con QoS 0

El siguiente nivel de *QoS* es el nivel 1, donde el publicador inicial guarda su mensaje hasta que el broker le confirma que el mensaje ha sido reenviado a los subscriptores, evitando pérdida de mensajes (ilustración 12).

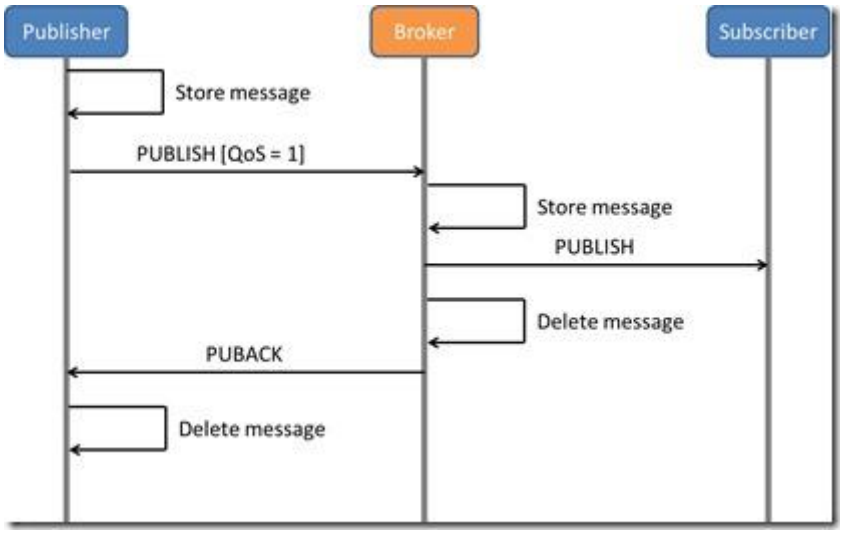

Ilustración 12 Intercambio de mensajes entre publisher-broker-subscriber MQTT con QoS 1

Como último nivel de QoS (ilustración 13), el nivel 2 implica que el mensaje llegue exactamente una vez, evitando duplicación de mensajes por medio un tráfico extra.

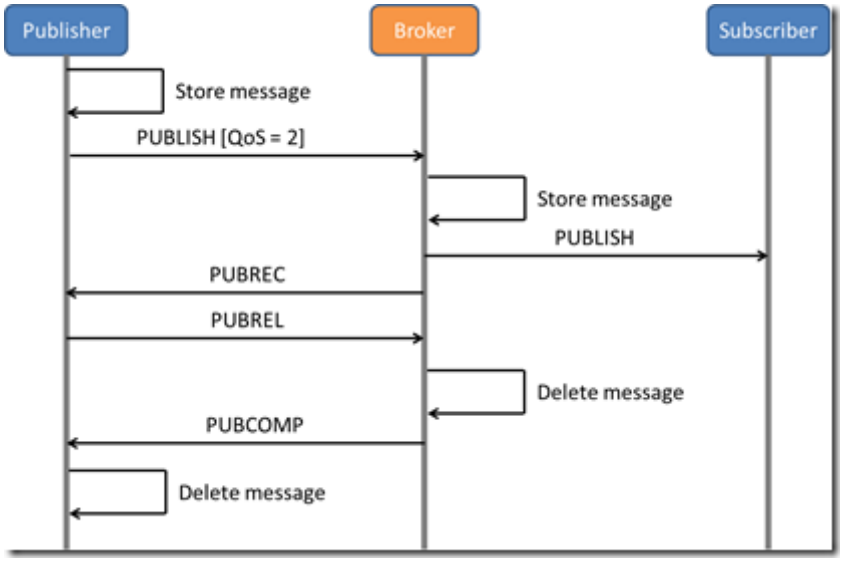

Ilustración 13 Intercambio de mensajes entre publisher-broker-subscriber MQTT con QoS 2

Si una conexión existe, pero sin intercambio de mensajes, se realizan envíos constantes de PINGREQ, desde el cliente, con respuesta PINGRESP, del broker, para que la conexión no se pierda (ilustración 14).

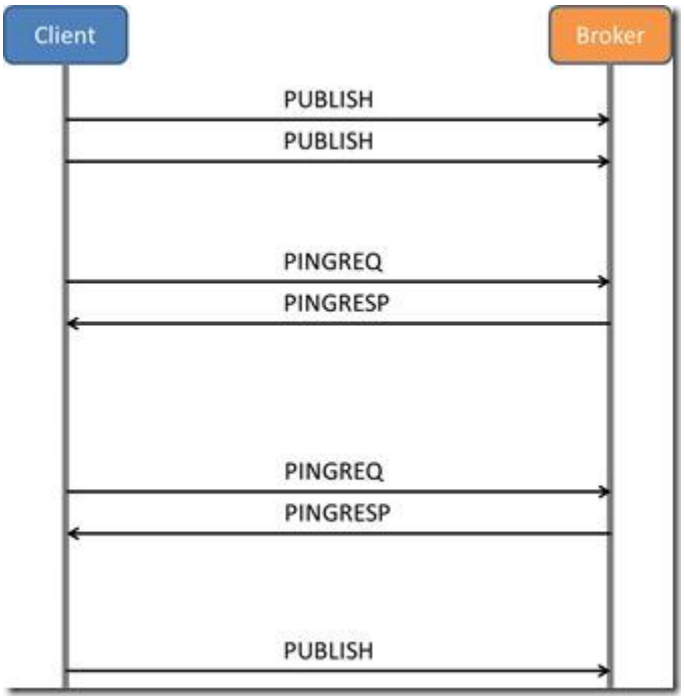

Ilustración 14 Intercambio de mensajes para evitar fin de conexión MQTT

3.1.1 Mosquitto

Servidor *broker* de código abierto que soporta las versiones más actuales de *MQTT*. *Mosquitto* ofrece una manera ligera y sencilla de poder usar el protocolo *MQTT* para el desarrollo de sus funciones entre él y los subscritores de sus publicaciones. Gracias a este servidor, se podrán suscribir a las *topics* ofrecidos desde un dispositivo x (normalmente los sensores que suministran los datos) los demás dispositivos con la mediación central del servidor.

Mosquitto también ofrece la posibilidad de conectarse con otros servidores *broker* que estén usando *MQTT*, creando un puente entre ambas entidades consiguiendo grandes redes de estos servidores pudiendo pasar datos de unos a otros.

Los mensajes publicados por medio de *Mosquitto* usando *MQTT* pueden complicarse muchísimo dependiendo de la cantidad de seguridad que se le quiera dar, sin embargo se va a optar a la versión más básica donde sólo hay que tener en cuenta los campos:

- -h: la dirección a la que se tiene que escuchar o en la que se tiene que publicar.
- -p: el puerto al que se tiene que escuchar o en el que se tiene que publicar.
- -t: el *topic* que identificará este mensaje.
- -m: el mensaje en sí que se enviará (en el que publica).

- -v: identifica que se tiene que escuchar a la espera de publicaciones (en el subscriptor).

Para realizar una prueba (ilustración 15), se puede abrir un segundo terminal creando un subscriptor que esté escuchando en esa dirección a la espera de que un mensaje tenga el *topic* descrito. Una vez se tenga al subscriptor escuchando, en el otro terminal se enviarán los distintos mensajes que se quieran publicar. El subscriptor recibirá todos los mensajes publicados con este topic hasta que se cierre la conexión.

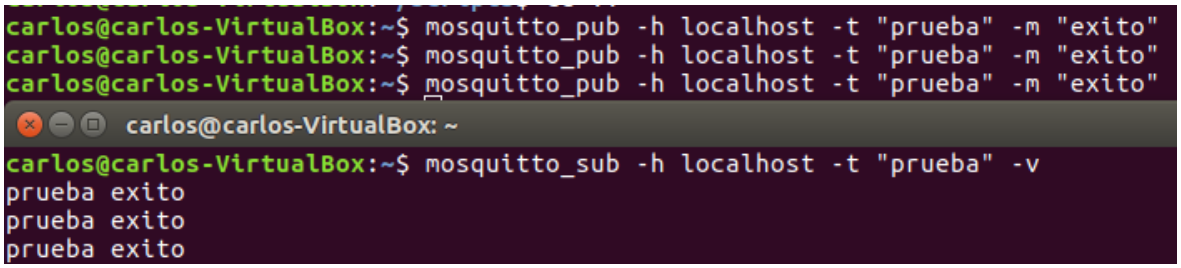

Ilustración 15 Ejemplo de publicación/suscripción en Mosquitto

3.2 MongoDB

MongoDB es una base de datos NoSQL orientada hacia documentos, no a relaciones. La primera ventaja de esto implica poder dar a las aplicaciones escalabilidad de una manera mucho más sencilla. Como se ha comentado en el capítulo anterior, se cambia el concepto "fila" de las bases de datos relacionales por el concepto "documento". Esto supone una gran diferencia para los desarrolladores, ya que así tienen una manera más natural para poder trabajar con sus datos.

En esta base de datos no se tienen esquemas predefinidos respecto a las claves y valores, con lo que no está sujetas a determinados tipos o tamaños.

3.2.1 Uso de una base de datos MongoDB.

En primer lugar, con el fin de poder usar una base de datos cualquiera, no sólo *MongoDB*, se debe tener claro qué jerarquía se va a tener en ella. El ejemplo siguiente es el que se tendrá también como versión final en este Trabajo. La jerarquía a seguir en este caso será empezar creando un perfil de usuario para asociar los sensores a su cargo y, posteriormente, las muestras de estos.

Para empezar a trabajar con una base de datos *MongoDB,* lo primero que hay que hacer es crearla (ilustración 16). Se debe usar el comando “use” y luego escribir el nombre con el que se quiere denominar a la base de datos. En el caso de que ya exista una base de datos con ese nombre, no se creará y se pondrá esa base de datos en primer plano para trabajar con ella.

```
> use sensoresDB
switched to db sensoresDB
```

Ilustración 16 Creación de una base de datos en MongoDB (o tenerla en primer plano)

Una vez en el interior de la base de datos, se debe de crear un usuario con determinados permisos para poderla usar desde fuera de ella más adelante (en este caso, desde *Node-RED*). El perfil creado es el mostrado en la (ilustración 17):

```
> db.createUser({user:"carlos",pwd:"start.1234", roles:[{role:"root",db:"admin"}
]})
Successfully added user: {
        "user" : "carlos",
        "roles" : [
                {
                        "role" : "root",
                        "db" : "admin"
                }
        ]
}
```

Ilustración 17 Creación de un usuario en la base de datos.

Una vez creada la base de datos, se pueden crear las colecciones que la compondrán. Primero se crea la colección “autor” para identificarlos (ilustración 18):

```
> db.createCollection("autor", {AutoIndexID:true})
{ "ok" : 1 }
```

Ilustración 18 Creación de una colección en MongoDB

Nota: cuando se crean colecciones en *MongoDB*, se pueden configurar bajo ciertas premisas como darle un tamaño máximo o un tamaño máximo de registros. En este caso, la opción “AutoIndexID” a “true” implica que cada vez que se genere un “autor” en esta colección, se generará para este registro un “_id” automáticamente. Esta opción se usará en todas las colecciones.

Teniendo la colección “autor” creada, inicialmente estará vacía. A continuación se crea un primer registro con los campos predefinidos en el diseño de la base de datos (“nombre”, “email” y “password”).

db.autor.insert({“nombre”: “carlos espacios”, “email” : “carlos.espacios101@alu.ulpgc.es”, “password”: “start.1234”})

En *MongoDB* hay diversas formas de búsqueda, siendo la más sencilla la función “find()”, que mostrará todos los registros de una colección. Además, con la función “pretty()” (ilustración 19), acompañando seguidamente a la anterior escrita, se logran unos resultados más visuales y fáciles de interpretar.

Ilustración 19 Búsqueda sencilla en MongoDB.

Nota: se ha añadido un segundo perfil de idéntica manera a la anterior para comprobar que el campo “_id” se genera automáticamente.

Una vez creado un perfil de autor, ya se le pueden asociar distintos sensores. Para ello, se debe de crear antes de nada la colección “sensor”, la cual ha sido creada de manera idéntica a la de “autor”.

A continuación, se le “añadirá” a este primer “autor” un sensor del modelo *DHT11*, el cual está disponible en la ULPGC en el caso de hacer para este proyecto experimentos con él. Este sensor recopila en sus muestras la temperatura y la humedad a la que se encuentra su entorno.

El proceso que se sigue no es el de añadir realmente a un “autor” un “sensor”, sino identificar al “sensor” como que pertenece al “autor”. Este modelo de relaciones entre colecciones se denomina “de referencia”, donde ambos documentos permanecerán separados pero donde el “sensor” tendrá un campo que identificará a su “autor”.

db.sensor.insert({“modelo”: “DHT11”, “localizacion” : “28.0717492, -15.4537252”, “autor_id”: “ObjectId(“5a045528c3b048442258031e”})

De una forma análoga, se añadirá a la base de datos una “muestra” (con la previa creación de su colección) e identificando al sensor de donde viene dicha muestra. En este caso, sólo a modo de ejemplo, y siendo los datos del sensor *DHT11*, se enviarán ambos en diferentes tipos, uno en *String* y otro en *int* (ilustración 20).

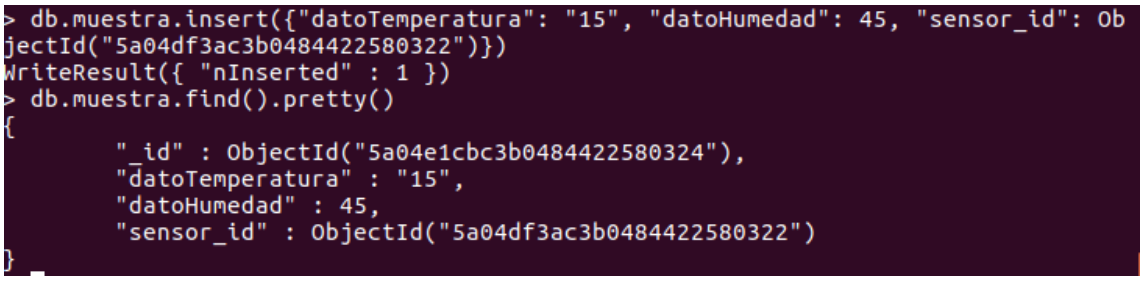

Ilustración 20 Inserción de una “muestra” identificando a su “sensor” y su búsqueda.

3.3 Node-RED

Node-RED es una herramienta gráfica y visual para desarrolladores con el fin de facilitar el diseño de su software. Se desarrolló a finales de 2013 por *IBM* como una forma de facilitar conexiones entre hardware, software y dispositivos para sus trabajadores, pero, al ser bastante aceptado por la comunidad, se tiene como una muy buena herramienta, sobre todo para el diseño en aplicaciones de *IoT*. Esto se debe al uso de “nodos”.

Los nodos se definen como bloques de código predefinidos que se conectan entre sí formando ‘’flujos’’ (*flows*), teniendo en cuenta tanto los nodos que funcionan como inputs, los que funcionan como outputs y los que se encuentran en medio; añaden la funcionalidad.

Los flujos son paquetes generados desde un nodo input e irán pasando por todos los nodos intermedios hasta llegar al nodo final del flujo. El paquete en *Node-RED* se denomina *msg* y tiene muchos campos para sincronismo para los diferentes módulos dentro de este entorno. Sin embargo, el campo más usado es el campo *payload;* campo donde está ubicada la información propia del mensaje. Todo esto posee la función de tener un entorno mucho más gráfico y visual para la elaboración de este Trabajo.

3.3.1 Node.js

Node.js es un entorno de ejecución para *JavaScript* que usa el modelo de ejecución E/S orientado a eventos y sin bloqueos. Se vale de ***npm***[22], una gran agrupación de librerías de código abierto propias de *node.js*.

Node.js fue creado para desarrollar aplicaciones en la red escalables. La mayoría de los sistemas de concurrencia usan hilos. Esto implica que para realizar un determinado proceso, se puede llegar a necesitar de una operación ubicada en otro hilo, el cual hasta determinado momento no nos dará su resultado. Esto puede llegar a implicar un bloqueo en el sistema, más peligroso aún si el bloqueo se produce en el hilo principal del sistema, lo que dará lugar a un fallo general. *Node.js* resulta más estable ya que casi ninguna función en *Node.js* realiza I/O directamente, así que el proceso nunca se bloquea. Debido a que no hay bloqueo es muy razonable desarrollar sistemas escalables en *Node.js*.

Este entorno se usa en videojuegos multijugador, software en la red, sistemas de tiempo real y aplicaciones con miles de usuarios concurrentes. Empresas como *eBay* y *LinkedIn* lo usan en sus aplicaciones.

Este nivel de concurrencia es debido a que Node.js usa el motor *V8* de *Google* para trabajar con *JavaScript*. Este motor, al estar diseñado para la web y sus protocolos como *HTTP*, *DNS* y *TCP*, hace que tenga una gran velocidad de ejecución.

Node.js tiene diversas ventajas tales como usar el mismo lenguaje, tanto en servidor como en cliente (*JavaScript*), tener en el servidor toda la funcionalidad necesaria (acceso a ficheros, a bases de datos, conexiones con clientes...). Además, se dispone de entornos gráficos de desarrollo en node.js que aprovechan el modelo de diseño basado en flujos. El más ampliamente conocido es *Node-RED*.

3.3.2 Diversos ejemplos de *Node-RED*

Para mostrar la potencia que *Node-RED* ofrece a sus usuarios (ilustración 21), se mostrarán algunos módulos interesantes y ciertas combinaciones de estos que sirvan como ejemplo de algunas de sus funciones que vienen en el paquete inicial, debido a que la comunidad siempre está creando módulos nuevos que aumentan la flexibilidad de este entorno.

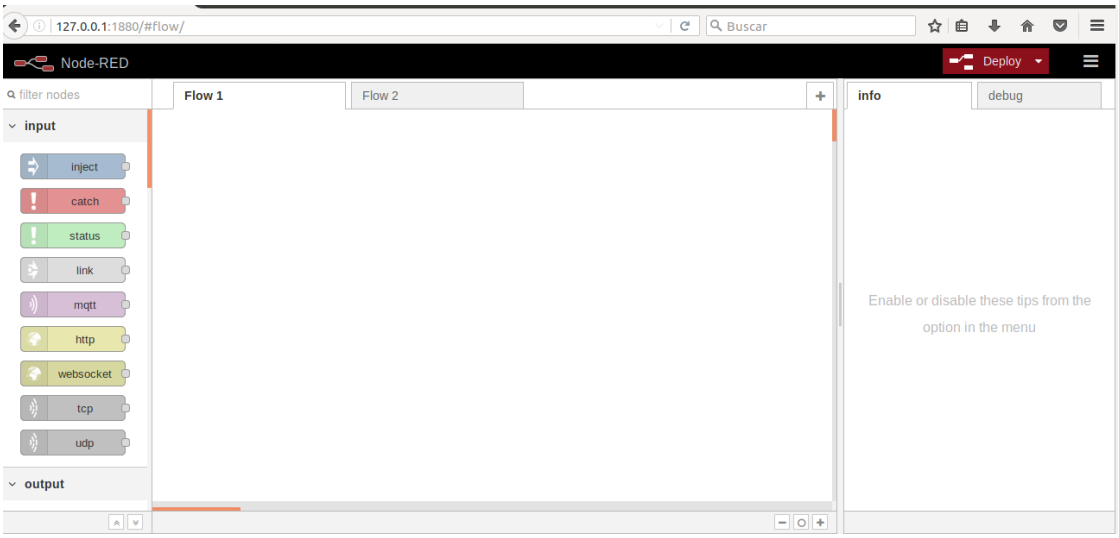

Ilustración 21 Ventana inicial de Node-RED

3.3.2.1 Uso del procotolo MQTT en Node-RED

Es bastante intuitivo generar un flujo en *Node-RED* con el protocolo *MQTT*, debido a que se tienen módulos predefinidos que harán esta función. Se debe escoger el módulo *input mqtt* visto en la figura x.x. y configurarlo con el fin de que escuche como subscriptor a un determinado *topic* enviado desde una dirección determinada (ilustración 22).

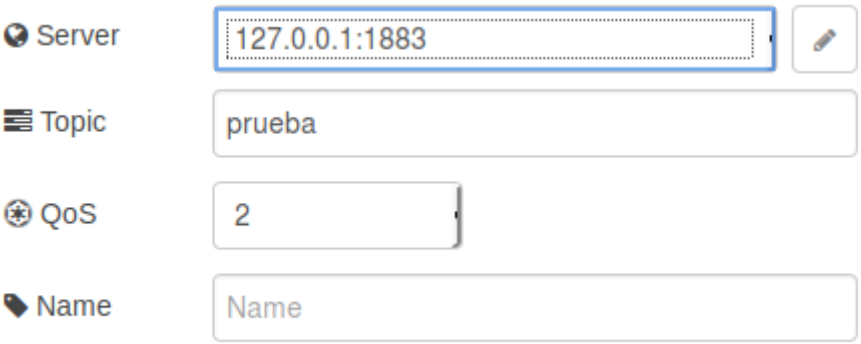

Ilustración 22 Configuración de un módulo input mqtt en Node-RED

Teniendo ya configurado el módulo input mqtt, se puede trabajar con él. Si se le añade un módulo debug a la salida, se visualizará el mensaje que llegué a la dirección configurada con el topic descrito. En este caso, a modo de ejemplo, la dirección es localhost, y el topic ha sido nombrado como ‘’prueba’’. Teniendo esto, si se publica un mensaje con el topic “prueba” a la dirección anteriormente especificada, será visualizado en la columna derecha

de Node-RED gracias al debug. El mensaje es el siguiente y como resultado se tiene la ilustración 23.

mosquitto_pub –h 127.0.0.1 –p 1883 –d –t “prueba” –m “exito”

Ilustración 23 Recepción en Node-RED de un mensaje MQTT.

El siguiente ejemplo es sencillo (ilustración 24), pero bastante importante para la realización de inserciones de datos en la base de datos: la diferencia entre recibir un *String* y un *JSON*. Se recibe un objeto *JavaScript* cuando se obtiene un mensaje desde *MQTT*, pero en el *payload*, se recibirá una cadena de *Strings* o valores numéricos.

Interesa recibir un archivo de tipo *JSON*, debido a que es bastante más flexible para manejar diferentes tipos de datos. Para esto se puede cambiar el objeto *JavaScript* a un objeto *JSON* con el módulo correspondiente en *Node-RED*:

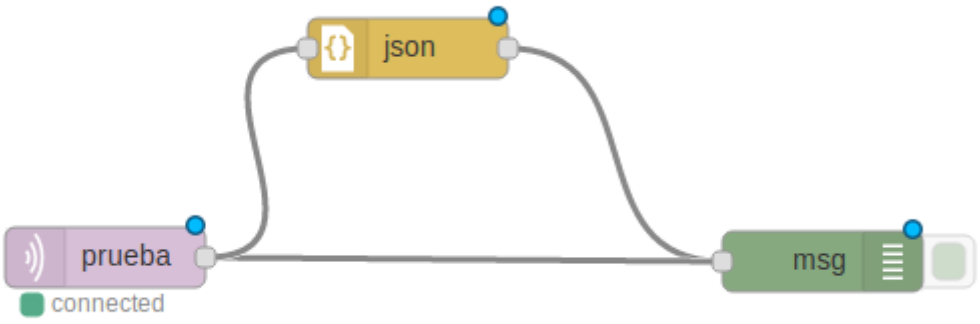

Ilustración 24 Esquema inicial con MQTT

Para este ejemplo, se conecta la salida del módulo de *MQTT* también directamente al *debug,* y no sólo al módulo que convierte el dato en *JSON* con el fin de observar la diferencia entre las dos salidas que se verán a continuación.

El mensaje publicado desde *mosquitto* es el mensaje que se ha utilizado otras veces como ejemplo, salvo que en la parte propia del mensaje se ha enviado un texto, el cual se pueda identificar como *JSON* y no solo como *String*.

Enviando el siguiente mensaje:

```
mosquitto_pub –h localhost –t “prueba” –m “{\”analyze\”:false, \”value\”:10}”
```

Las salidas en el *debug* son las ilustraciones 25 y 26:

Ilustración 25 Salida como JavaScript Object

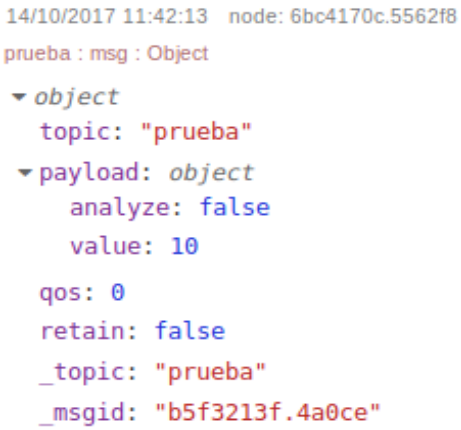

Ilustración 26 Salida como JSON Object

Se tienen estas dos salidas debido a que el debug está conectado directamente al módulo *mqtt* y también al módulo JSON. Como se puede apreciar, en la primera salida (ilustración 25) el *payload* es simplemente un *String*, por lo que con esos datos no se podría trabajar al no tenerlos diferenciados, al menos no de una manera simple. Por otra, se puede observar como en la segunda salida (ilustración 26), que es la que tiene como intermediario el módulo JSON, los datos están diferenciados, donde, en lugar de tener un String, se tiene dos variables (“analize” y “value”), estando separadas una de otra.

3.2.2.2 Módulos switch y change

Otras funciones bastantes útiles que provee *Node-RED* es el poder redireccionar los flujos de datos, dependiendo de ciertos valores que lo compongan (módulo *switch*), o cambiar, eliminar, o crear partes del mensaje según convenga (módulo *change*). En el siguiente ejemplo (ilustración 27) se trabajará con valores numéricos.

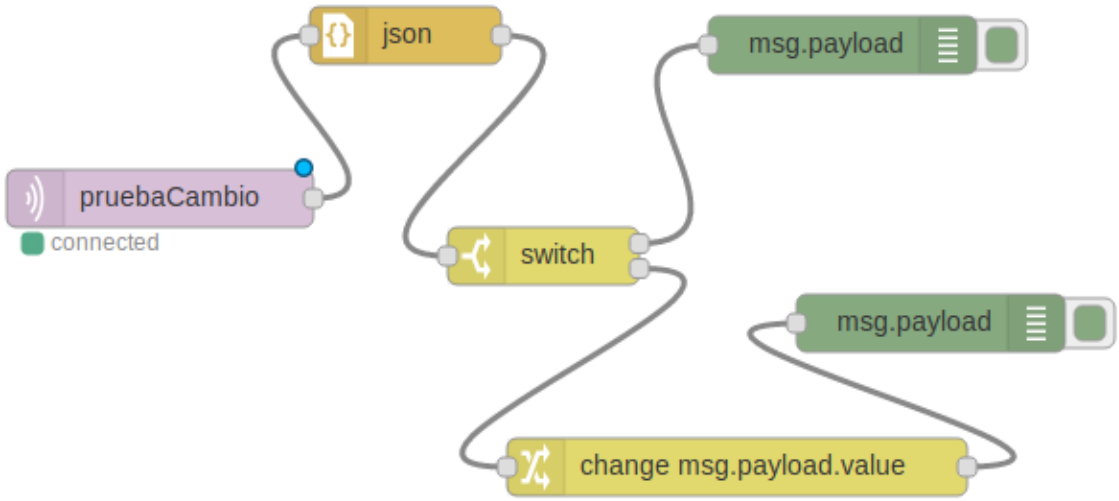

Ilustración 27 Ejemplo de redireccionamiento y cambio de variables

Para la prueba se configurará de tal manera al *switch* que se envíe el flujo al primer o al segundo *debug*, emulando dos salidas diferentes. En cuanto a este módulo, se le pueden añadir más salidas (no tienen que ser únicamente 2), y el número del camino x será el *output* x que se escoja de este módulo.

En este caso, sólo se tienen dos outputs para este ejemplo (ilustración 28), donde el flujo de datos se redireccionará al primer camino si el valor numérico es menor de 5 y por el segundo camino si es mayor (no se contemplará el 5 para realizar la prueba de que el flujo no avanzará, ya que no cumple ninguno de los requisitos).

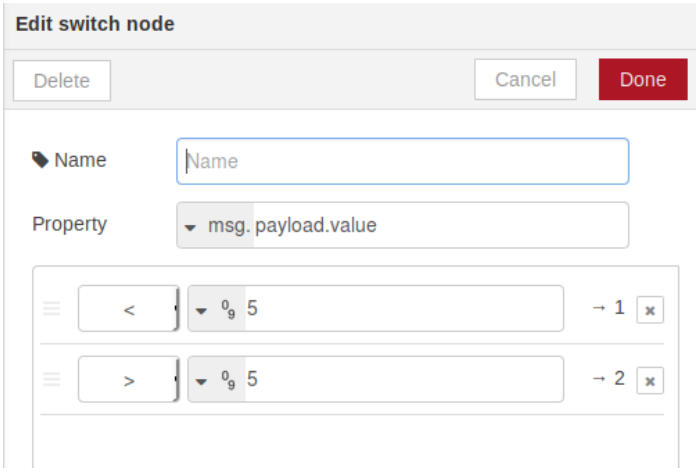

Ilustración 28 Configuración del módulo switch para valores numéricos.

Nota: la propiedad a cambiar es *msg.payload.value,* debido a que dentro del *payload* la variable donde se introduce el valor numérico se ha nombrado como *value*.

Para verificar esto, se hará uso de otra función nombrada anteriormente, el módulo *change*. En este caso, si el valor numérico publicado por *mosquitto* es un 7, el módulo nombrado, que estará colocado entre el *debug* y el segundo output, cambiará este valor por 8 (ilustración 29).

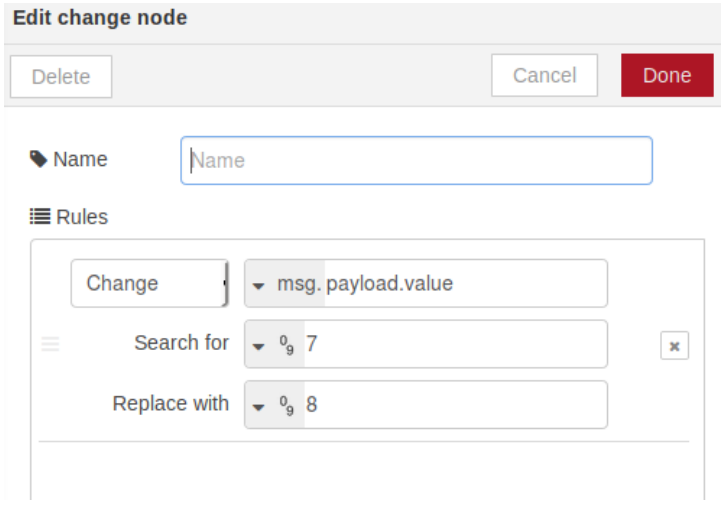

Ilustración 29 Configuración del módulo change.

Teniendo todo esto, se ejecuta la publicación por *mosquitto* de los valores 4, 5, 6 y 7 añadiendo un *String* identificando el camino por donde debería ir (ilustración 30).

```
carlos@carlos-VirtualBox:/etc/apt/sources.list.d$ mosquitto_pub -h localhost -t
"pruebaCambio" -m "{\"value\":4, \"nota\":\"primer camino\"}"
carlos@carlos-VirtualBox:/etc/apt/sources.list.d$ mosquitto_pub -h localhost -t
"pruebaCambio" -m "{\"value\":5, \"nota\":\"sin camino\"}"
carlos@carlos-VirtualBox:/etc/apt/sources.list.d$ mosquitto_pub -h localhost -t
"pruebaCambio" -m "{\"value\":6, \"nota\":\"segundo camino\"}"
carlos@carlos-VirtualBox:/etc/apt/sources.list.d$ mosquitto_pub -h localhost -t
"pruebaCambio" -m "{\"value\":7, \"nota\":\"segundo camino\"}"
```

Ilustración 30 Envío de mensajes

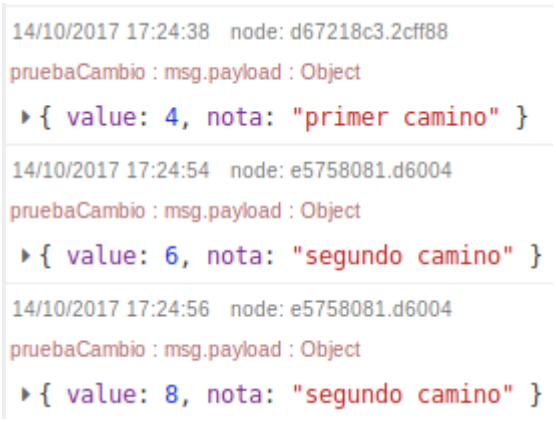

Ilustración 31 Mensajes recibidos por los distintos nodos.

Se puede ver como “value”=7 ha sido cambiado por “value”=8 y que el paquete con “value”=5 ha sido desechado (ilustración 31). Además, los identificadores de los *debug* (node=x) son iguales en los paquetes que tuvieron que ir por el segundo camino y diferentes con respecto al que se le había designado el primer camino.

Nota: se ha realizado la prueba de enviar de un 7 con un valor acompañado (un 17) para verificar si se realizaba la función *change* con cualquier 7 en el mensaje, pero no ha sido así.

Para evitar tener que añadir notas desde *MQTT* y que sea más eficaz, el módulo *change* también ofrece la función de *set* con la que se pueden insertar datos y campos en el *msg* (ilustración 32).

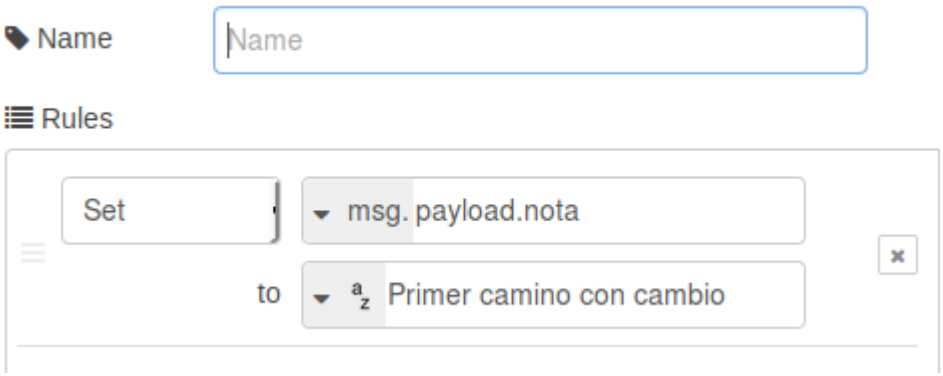

Ilustración 32 Módulo change con función set.

Este módulo está actualizando el campo ‘*’nota”* (si no estuviera creado se crearía la variable y el valor) en el mensaje con el *string* ‘’Primer camino’’. Colocando este módulo entre el primer *output* y publicando el mismo mensaje anterior, la salida quedaría como en la ilustración 33:

Ilustración 33 Resultado del módulo set.

El resultado obtenido es que se sobrescribe el campo ‘*’nota”* con el mensaje escrito en el módulo *change*.

En el caso de querer que los flujos de datos funcionen con *websockets*, *Node-RED* proporciona módulos tanto de input como output con esta tecnología (ilustración 34).

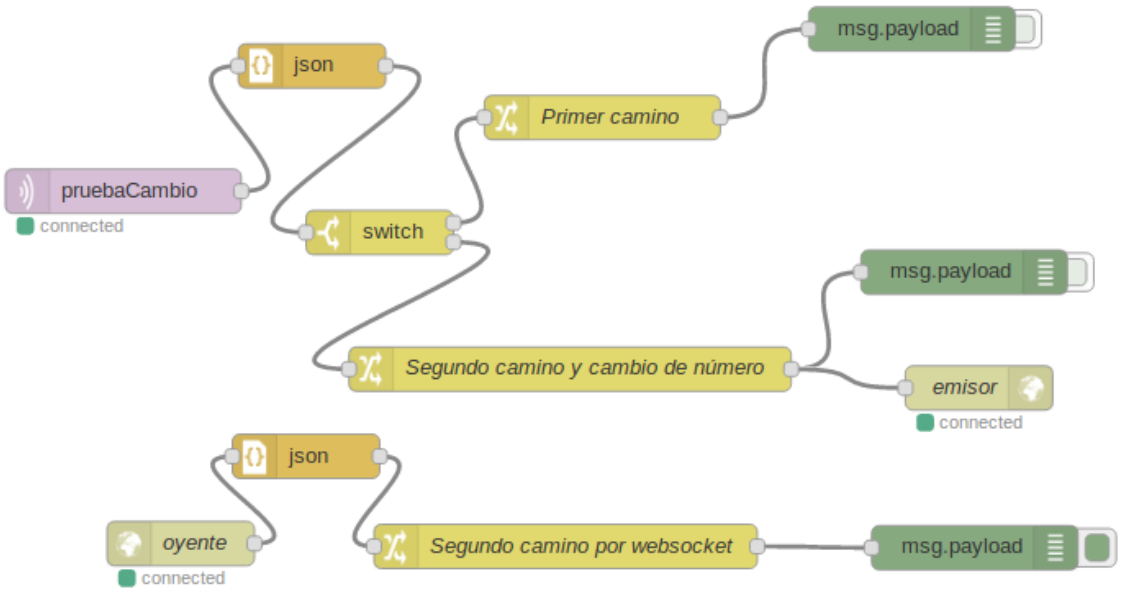

Ilustración 34 Esquema anterior trabajando con websockets.

Los *websockets* son bastantes útiles en el caso de que se requiera la existencia de una comunicación bidireccional y *fullduplex*. Trabajando sobre un único *socket TCP*, se pueden dar estas comunicaciones entre servidores web y navegadores, o para aplicaciones cliente/servidor como caso genérico.

Enviando el siguiente mensaje:

```
mosquitto_pub –h localhost –t “pruebaCambio” –m “{\”value\”:7, \”nora\”:\”sin nota\”}”
```

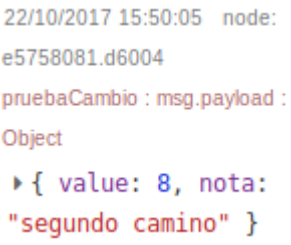

Ilustración 35 Salida por segundo camino.

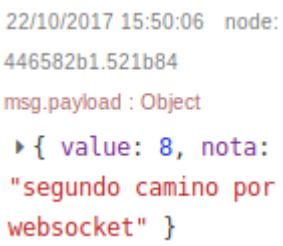

Ilustración 36 Salida por segundo camino mediante websockets camino.

Se puede comprobar como el mensaje ha sido cambiado (ilustración 35) y también transmitido por hacia otra salida usando *websockets* (ilustración 36).

3.3.2.2 Módulos split y join

El módulo presentado a continuación tiene una función muy interesante: dividir el mensaje. El módulo *split* pide el identificador oportuno para separar un dato de otro. En este caso (ilustración 37), el mensaje usado separa un valor de otro con una coma (*{"value":7, "nota":"sin nota"}*), por lo que será el identificador que *split* usará para separar un dato de otro.

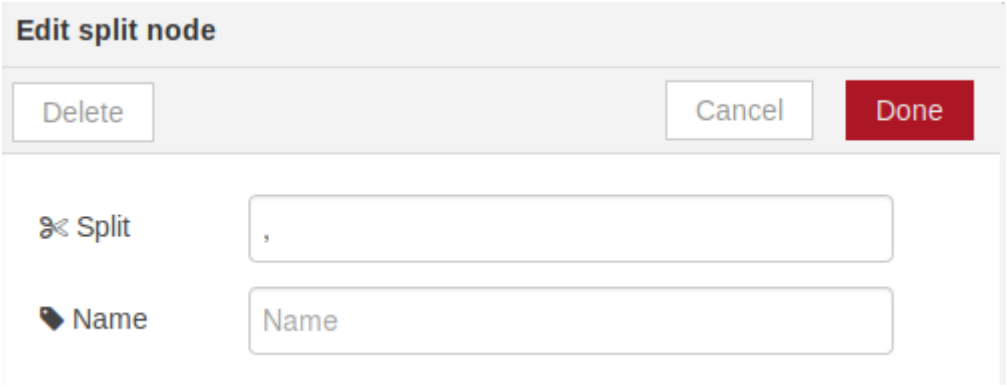

Ilustración 37 Configuración del módulo split.

El primer ejemplo (ilustración 18) de este módulo es uno sencillo, sólo con los módulos de *MQTT*, el módulo *JSON* que pasará su salida a un objeto *JSON*, y un *debug* para mostrar el resultado.

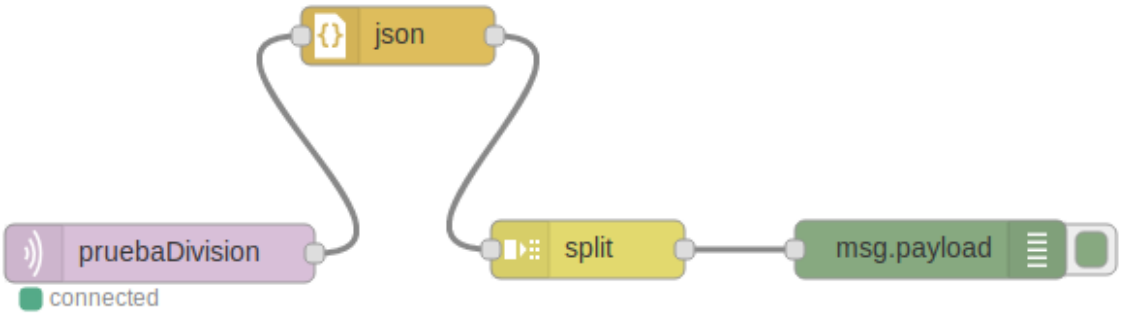

Ilustración 38 Esquema inicial con módulo split.

Publicando el siguiente mensaje:

```
mosquitto_pub –h localhost –t "pruebaDivision" –m "{\"value\":7, \"nota\":\"sin nota\"}"
```

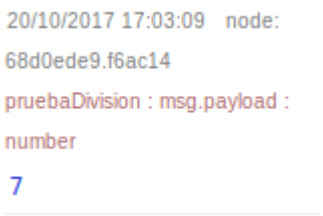

Ilustración 40 Salida numérica resultante del módulo split

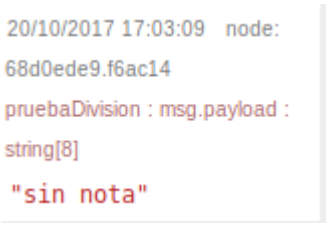

Ilustración 39 Salida de texto resultante del módulo split.

El mensaje ha sido separado en dos (ilustraciones 39 y 40), teniendo los valores que componían el mensaje inicial en dos mensajes independientes el uno del otro. Pero queda una duda: si los mensajes están separados y muestra el valor de la variable, ¿se guarda también la variable en sí o sólo se pasa el valor final?

Al igual que existe el módulo *split,* también existe el módulo *join*, cuya función es recuperar y agrupar los datos divididos por un *split* anterior. El ejemplo es el siguiente:

Teniendo el mismo caso anterior, se le añade después del *split* un módulo *join* con otra salida aparte (ilustración 41): con eso se podrá observar cómo se pueden recibir los mensajes juntos y separados dependiendo del *debug* escogido. El mensaje enviado será el mismo que en el supuesto especificado con anterioridad.

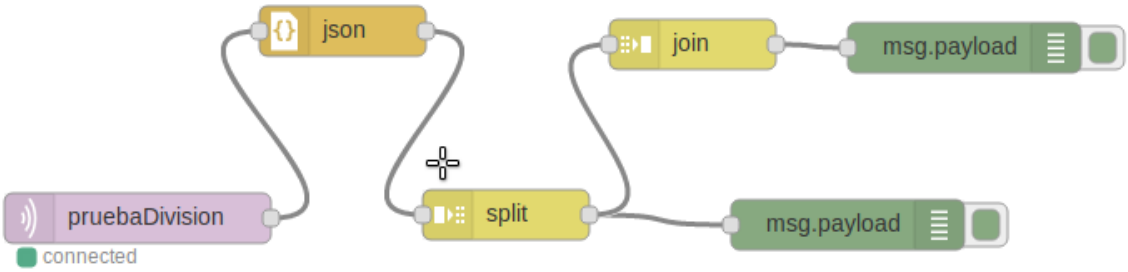

Ilustración 41 Esquema con módulos split y join.

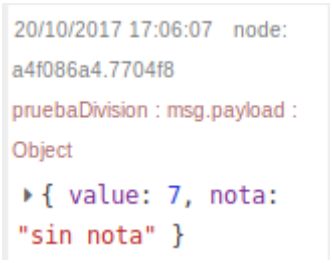

Ilustración 42 Salida con el módulo join

Las salidas representan cómo han llegado, por un lado, los mensajes separados en un *debug* (mismas ilustraciones que las 39 y 40), y en el otro juntos, gracias a la función del módulo *join* (ilustración 42). En este ejemplo no se visualiza el efecto real de dividirlos y luego agruparlos, pero responde a la pregunta anterior, ya que siguen existiendo las variables ‘*’value”* y ‘*’nota”,* y no sólo sus valores.

En el último ejemplo (ilustración 44) se presentará un módulo *split* que separará los datos y se crearán dos flujos:

- El primero de ellos cambiará los datos, tanto el numérico como el *string*, con un módulo *change* (ilustración 43), el cual tendrá dos salidas: una directa a un *debug* para visualizar los datos separados, y otra que tendrá entre el *change* y el *debug* un módulo *join,* con el fin de visualizar los datos conjuntos y constatar que la estructura del mensaje sigue correcta.

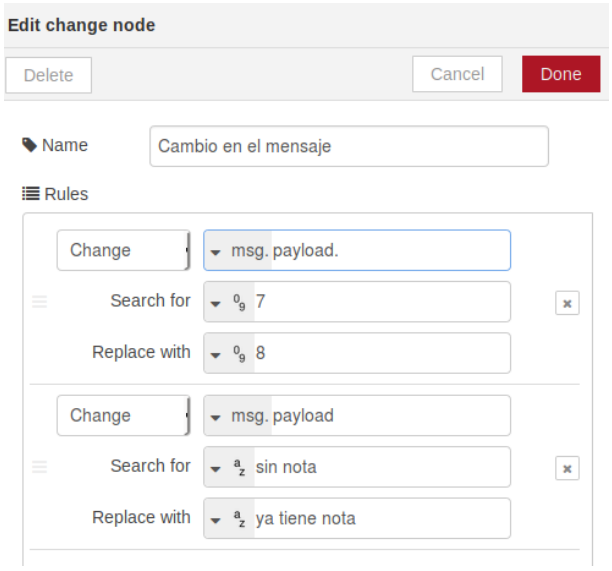

Ilustración 43 Configuración del módulo change en el ejemplo con los módulos split y join

- El segundo camino simplemente estará conectado a un *debug* con un módulo *join* en medio para que se pueda comparar la salida con el *join* del primer camino.

Con esto se quiere comprobar que, en el caso de estar el mensaje separado, se pueda cambiar el valor de las variables y que, aun así, no se pierda la variable en sí. El mensaje a enviar seguirá siendo el mismo.

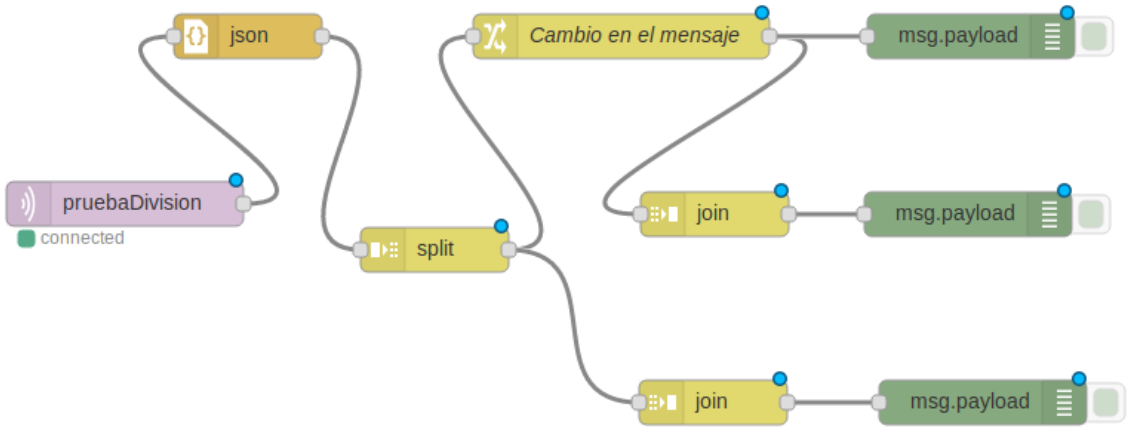

Ilustración 44 Esquema con los módulos split, join y change.

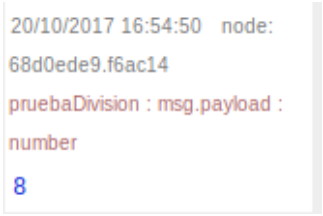

Ilustración 46 Salida numérica con cambios sin join.

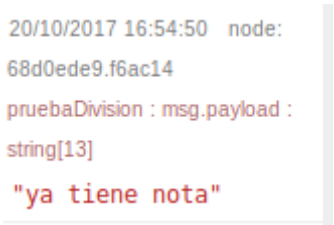

Ilustración 45 Salida de texto con cambios sin join

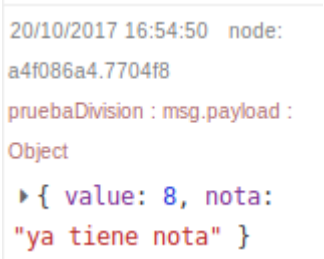

Ilustración 47 Salida con cambios y split-join.

La salida de la ilustración 47 demuestra que, aunque las variables sufran cambios durante los flujos, la variable sigue existiendo aunque su valor cambie, ya que los valores que se publicaron son los mismos que en la ilustración 42. Las ilustraciones 45 y 46 quedan como demostración de que el mensaje antes de unirse fue separado y modificado

3.3.2.3 Módulo range

Teniendo en cuenta que este proyecto se basa en que se puede llegar a tener muchísimos datos de sensores con sus determinadas escalas, resulta necesario disponer de un módulo que equivale las medidas a una estándar en el caso de necesitarlo. Este módulo es el *range*.

Este módulo funciona de la siguiente manera: se tienen que introducir 4 cifras numéricas, dos que irán del mínimo al máximo de la entrada y otras dos del mínimo al máximo de la salida. El módulo se encargará de ajustar la medida que le llegue, para que, en su salida, esta medida esté en la escala que se pretenda.

El problema que presenta este módulo es que no acepta ninguna entrada que no sea sólo un número, por lo que se tiene que pasar el dato que se quiera convertir de manera aislada. Si por ejemplo, se pretendiera realizar conversión de temperatura en *Kelvin* a grados centígrados, el esquema en *Node-RED* sería el expuesto en la ilustración 48:

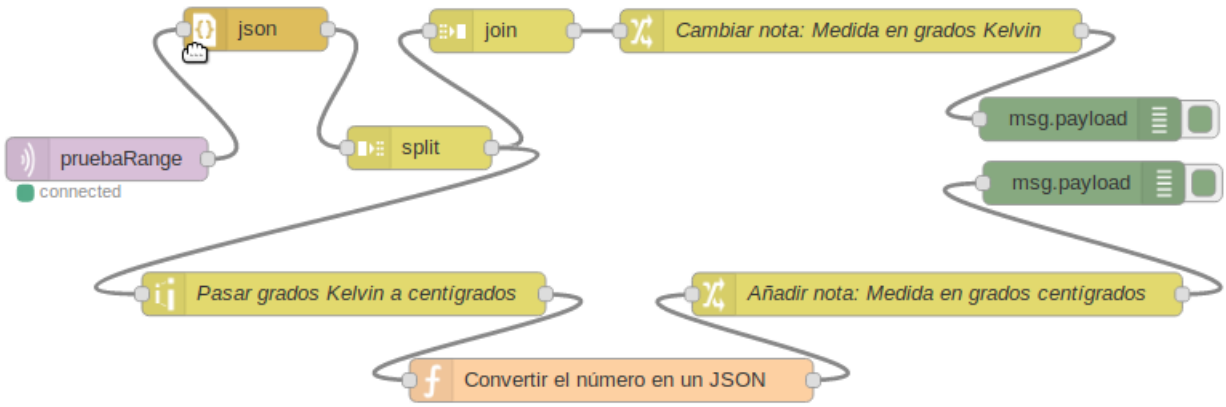

Ilustración 48 Diagrama para el ejemplo de uso del módulo range.

Para este ejemplo, la escala con la que se trabajará será desde los 253 hasta los 313 Kelvin y desde los -20 hasta los 40 grados centígrados (ilustración 49).

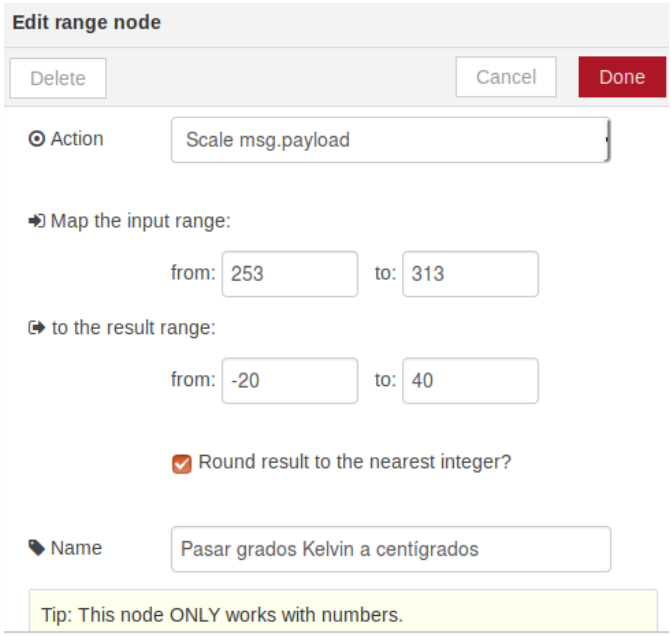

Ilustración 49 Configuración del módulo range para las medidas anteriormente especificadas

El dato que se enviará como ejemplo será "*value*"=280. A la hora de dividir el mensaje, el módulo *range* descartará el mensaje que tenga la variable "*nota*" y sólo trabajará con el valor numérico, por lo que no se podrá recomponer el mensaje. Por esto, se añade el módulo de *function* (ilustración 50*)*, que introducirá la salida del módulo *range* (que es únicamente numérica, sin estar en ninguna variable) en una nueva variable y se creará un objeto *JSON*.

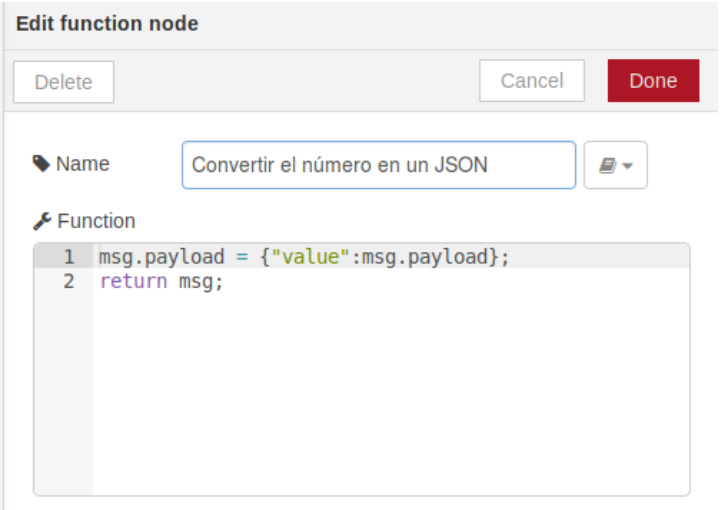

Ilustración 50 Módulo function para convertir el valor numérico en un JSON.

Enviando el siguiente mensaje:

```
mosquitto_pub –h localhost –t “pruebaRange” –m “{\”value\”:280, \”nota\”:\”sin nota\”}”
```

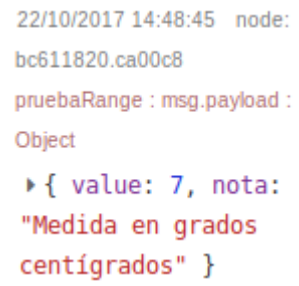

Ilustración 52 Temperatura cambiada a grados centígrados.

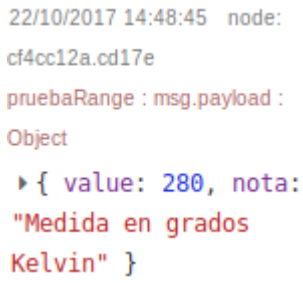

Ilustración 51 Temperatura en Kelvin inicial.

Como se puede comprobar, debido al módulo *range*, se ha hecho el cambio de *Kelvin* (ilustración 52) a grados centígrados (ilustración 51). Este módulo sólo funciona de manera lineal, por lo que no se pueden hacer más pruebas con él teniendo una diferencia significativa que los justifique.

3.4 Web Components

Los *Web Components* son un conjunto de *APIs* de plataformas web que permiten crear, reutilizar y encapsular etiquetas HTML para su uso en aplicaciones y páginas web. Los *Custom Elements* que se creen estarán basados en los estándares de *Web Components*, que a su vez están basados en estándares web existentes. Con ello funcionarán en casi todos los navegadores web modernos, y funcionarán en cualquier entorno de *JavaScript* con *HTML*.

3.4.1 Especificaciones

Los *Web Components* se basan en cuatro especificaciones:

- La especificación del **Custom Element**: da base para el diseño y uso de nuevos tipos de elementos *DOM*. Gracias a esta especificación, se da un formato a los elementos *DOM* creados por sus diseñadores, lo que a la larga se ha visto muy útil dado que resuelve problemas de construcción en los navegadores y como se deben de comportar los elementos que componen estas clases ante los cambios.

Los *Custom Elements* funcionan independientemente del resto de la aplicación y contienen todo lo necesario para su funcionamiento.

- La especificación **shadow DOM**: se define como se debe de usar el encapsulado y el marcado en los *Web Components*. En ella además se describe el método para combinar varios DOM para jerarquizarlos y como deben de interactuar entre sí en el documento donde se encuentren. Gracias a esto se obtiene una mejor composición del DOM.

- La especificación **HTML imports:** define como incluir y reusar documentos HTML en otros. Estas importaciones se crean al vincular los documentos HTML que se quieran reusar como recursos externos de los nuevos HTML. El documento que se vincula se denomina import referrer y estos documentos pueden, a la vez, derivarse de otras vinculaciones.

- La especificación **HTML template element:** define como se deben de declarar los fragmentos y el marcado que no se use al cargar la página pero si durante el tiempo de ejecución si surgiera los condicionantes necesarios para ello.

Ilustración 53 Compatibilidades con los principales navegadores web a 9/12/2017

Para arreglar determinados problemas con respecto a los navegadores (ilustración 53), se usan los ***polyfills***, que son códigos que simulan las distintas compatibilidades, que a día de hoy no están aún estables, que se tienen que dar entre los *Web Components* y el navegador y evitar fallos que se puedan producir. Obviamente, no resuelve todas y se pueden dar fallos como por ejemplo ciertas limitaciones en los *CSS* o falta de implementación en ciertas funciones.

3.4.2 La librería Polymer

Una de las librerías más importantes de los *Web Components* es ***Polymer***, a la que se le dará uso en este proyecto. Fue creada por Google, y todos los navegadores actuales la pueden usar de manera nativa. En la siguiente versión será nativa propiamente en el navegador.

En este caso, en lugar de crear *Custom Elements* se crearían ***Polymer Elements***, los cuales son similares pero se trabaja con ellos de una manera mucho más rápida gracias al uso de la librería.

El objetivo de esta librería es, simplemente, hacer la vida más fácil a los creadores de software, proveyendo más de 200 *Polymer Elements* (ilustración 54) diseñados por sus desarrolladores (en www.webcomponents.org se pueden encontrar más de 600 *Custom Elements* y en *Github* más de 5000 creados por la comunidad).

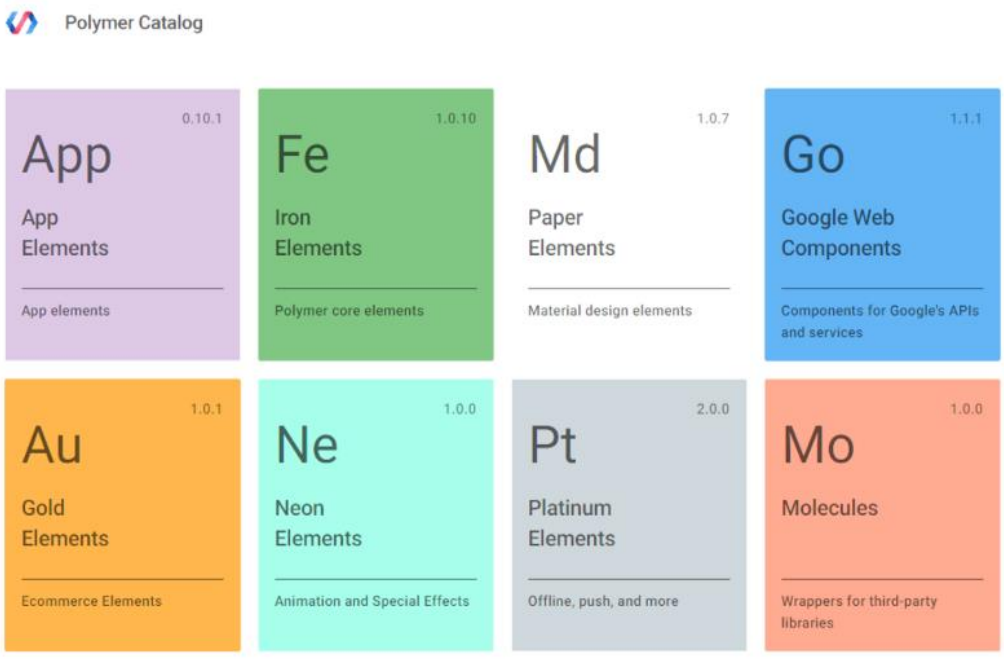

Ilustración 54 Catálogo de Polymer Elements.

3.4.2.1 Ejemplo Polymer

Un ejemplo útil de lo que ofrece la librería Polymer es generar tablas. Para este ejemplo se debe de generar, como mínimo, las figuras expuestas en este apartado para crear la tabla (hará falta más programación, pero la aquí explicada es la parte que concierne a Polymer).

Primero, se debe de crear un archivo que haga de módulo de Polymer, el cual será usado más adelante en el HTML. Para ello, se debe de crear con los styles predeterminados para el archivo que vamos a usar de la librería, en este caso *polymer-element* (ilustración 55). Con esto se consigue tener el código en formato modular lo que implica su reusabilidad y un mejor mantenimiento.

```
<link rel="import" href="../bower_components/polymer/polymer-element.html">
<link rel="import" href="../bower_components/polymer/lib/elements/dom-repeat.html">

<dom-module id="carlos-table">
  <template strip-whitespace="">
    <style>
      :host {
        display: block;
      }
      th {
        padding: 5px;
        background-color: #ccc;
      }
      th, td {
        text-align: right;
      }
      .colSensor {
        width: 80px;
      }
      .colDate {
        width: 100px;
      }
      .colTime {
        width: 100px;
      }
      .col-temperature {
        width: 120px;
      }
      .col-humidity {
        width: 120px;
      }


    </style>
```

Ilustración 55 Dependencias y styles necesarios para la configuración del módulo.

Una vez hecho esto, se configurará las etiquetas que debe de tener la tabla que se va a crear (ilustración 56).

```
<table>
  <thead>
    <tr>
      <th class="colSensor">Sensor</th>
      <th class="colDate">Date</th>
      <th class="colTime">Time</th>
      <template is="dom-repeat" items="[[cols]]">
        <th class$="col-[[item.property]]">[[item.title]]</th>
      </template>
    </tr>
  </thead>
  <tbody>
    <template is="dom-repeat" items="[[data]]" as="dataItem">
      <tr>
        <td>[[dataItem.sensor]]</td>
        <td>[[dataItem.date]]</td>
        <td>[[dataItem.time]]</td>
        <template is="dom-repeat" items="[[cols]]" as="col">
          <td>[[_getSensorData(dataItem, col)]]</td>
        </template>
      </tr>
    </template>
  </tbody>
</table>
</template>
```

Ilustración 56 Especificación de la tabla en el módulo Polymer.

Ya para finalizar este módulo, se debe crear la clase que extiende de la librería para su uso, y será la que generará la tabla como tal (ilustración 57). Las columnas estarán especificadas por *arrays* debido a que así serán los datos que se reciban.

```
<script>

  class CarlosTable extends Polymer.Element {

    static get is() { return 'carlos-table'; }

    static get properties() {
      return {
        cols: {
          type: Array,
          value() {return []}
        },
        data: {
          type: Array,
          value() {return []}
        }
      }
    }

    _getSensorData(item, col) {
      return item[col.property];
    }

  }

  window.customElements.define(CarlosTable.is, CarlosTable);
</script>
</dom-module>
```

Ilustración 57 Clase que retornará la table "carlos-table".

Una vez que se tiene el módulo terminado, en el archivo HTML, donde se usará este módulo, se deben de añadir las dependencias necesarias para que pueda funcionar. Además, el Web Component que se esté creando, en este caso un objeto Polymer para hacer una tabla, debe de nombrarse como esta nombrado el módulo anteriormente visto (en este caso, "carlos-table"). Ya dentro de la etiqueta google-chart (Polymer está creado

por Google), se escriben los diferentes nombres que con las que se quiere nombrar a la tabla, columnas etc (ilustración 58).

```
<!DOCTYPE html><html lang="en"><head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, minimum-scale=1.0, initial-scale=1, user-scalable=yes">

  <title>MUESTRAS</title>

  <script src="bower_components/webcomponentsjs/webcomponents-lite.js"></script>
  <link rel="import" href="./components/carlos-table.html">
  <link rel="import" href="bower_components/google-chart/google-chart.html">
  <link rel="import" href="bower_components/paper-slider/paper-slider.html">

  <style>
    google-chart {
      height: 300px;
      width: 600px;
    }
    paper-slider {
      width: 600px;
    }
  </style>
</head>

<body>
  <carlos-table id="carlosTable"></carlos-table>
  <google-chart id="line-chart" type="line" options="{&amp;quot;title&amp;quot;: &amp;quot;Temperature in Arinaga&amp;quot;}"
                cols="[{&amp;quot;label&amp;quot;: &amp;quot;Date&amp;quot;, &amp;quot;type&amp;quot;: &amp;quot;string&amp;quot;},{&amp;quot;label&amp;quot;:
                &amp;quot;Temperature&amp;quot;, &amp;quot;type&amp;quot;: &amp;quot;number&amp;quot;},{&amp;quot;label&amp;quot;: &amp;quot;Humidity&amp;quot;,
                &amp;quot;type&amp;quot;: &amp;quot;number&amp;quot;}]">
  </google-chart>
  <paper-slider></paper-slider>
```

Ilustración 58 Dependencias y styles de Polymer necesarios en el HTML.

Para finalizar el ejemplo, se debe de “apuntar” los diferentes componentes de la tabla para poder trabajar con ellos dentro de los scripts mediante el constructor (ilustración 59).

```
constructor() {
  this.data = [];
  this.dataLength = 10;
  this.table = document.querySelector('carlos-table');
  this.table.cols = [
    {title: 'Temperature', property: 'temperature'},
    {title: 'Humidity', property: 'humidity'}
  ];
  this.chart = document.querySelector("google-chart");
  this.slider = document.querySelector("paper-slider");
  this.slider.addEventListener("immediate-value-change", evt => {
    this._setOutputData(evt.target.immediateValue);
  });
  this.slider.value = 0;
  this._generateTestData();
  this.slider.max = this.data.length - this.dataLength;
  this._setOutputData(0);
}
```

Ilustración 59 Constructor en el archivo HTML.

Esto es la parte que concierne a Polymer respecto a la tabla que se va a crear, ahora quedaría rellenarla de la forma que se quiera según el caso. El resultado, una vez que se ha programado todos los demás componentes para la recopilación de datos y su incrustación en la tabla, quedaría como en la ilustración 60.

Sensor	Date	Time	Temperature	Humidity
5a32b106dbe65609cd63af23	2018-01-08	16:48:34	45	90
5a32b106dbe65609cd63af23	2018-01-08	16:48:35	45	90
5a32b106dbe65609cd63af23	2018-01-08	16:48:36	45	90
5a32b106dbe65609cd63af23	2018-01-08	16:48:36	45	90
5a32b106dbe65609cd63af23	2018-01-08	16:48:37	45	90

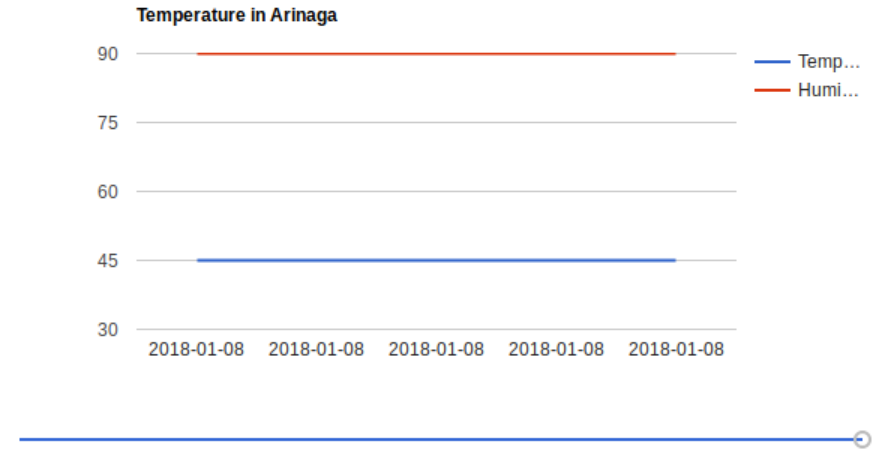

Ilustración 60 Tabla ejemplo Polymer terminada.

Capítulo 4. Desarrollo e implementación de la plataforma

En este cuarto capítulo se abordará la propia plataforma finalizada, la cual tendrá como componentes: el servidor *MQTT Mosquitto*; atendiendo a la publicación de datos de los sensores asociados a este servidor, la base de datos; donde se guardarán estos datos aparte de los perfiles de los usuarios y sus sensores, y la configuración de distintos flujos en Node-RED, que cohesionarán todos los componentes y darán una vista de usuario.

El caso de uso propio que se tendrá en este proyecto será, en primer lugar, una interfaz para permitir al usuario, o bien iniciar sesión si ya está registrado, o bien darse de alta en el caso de que aún no tenga cuenta.

En segundo lugar, se tendrá una segunda página donde el usuario en cuestión dispondrá de todos los sensores asociados a su cuenta y de un botón que le brindará la posibilidad de añadir sensores nuevos a su perfil.

La tercera y última página se abrirá si el usuario selecciona algún sensor de la página anterior. En esta, se le mostrarán todas las muestras que hay en la base de datos del sensor seleccionado y se le brindará una gráfica de estos datos a lo largo del tiempo.

Se proveerá también de una página de administrador de usuario por medio del *dahsboard* de *Node-RED* con el fin de poder realizar funciones sobre los perfiles de usuario.

Aparte de todo esto, en este capítulo también se explicará la estructura y jerarquía con la que está diseñada la base de datos de esta plataforma.

4.1 Diseño de la base de datos.

A continuación, se mostrará cómo está diseñada la base de datos para este Trabajo, dónde se podrá observar su organización y cómo estarán conformadas las colecciones para complementarse unas a otras.

Todas las colecciones tendrán, aparte de sus propios campos, el campo “_id” que identificará a cada uno de los registros que haya en la base de datos inequívocamente, dando igual en cuál de las colecciones esté ubicado, debido a que para todos será distinto.

Se tendrán cuatro colecciones: “autor”, “sensor”, “muestra” y “sesion” jerarquizadas entre ellas.

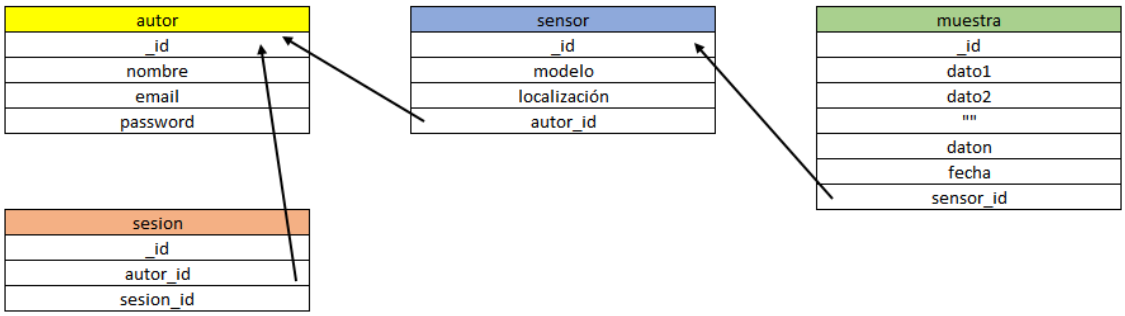

Ilustración 61 Diseño de la base de datos.

4.1.1 Colección autor

La colección autor será la utilizada para guardar los perfiles de los usuarios que suban datos a la plataforma. Esta colección tendrá como campos:

- "nombre": nombre del usuario en cuestión. Será con el campo con el que se identificará el usuario. Tipo: *String*.
- "email": dirección de correo electrónico del usuario. Tipo: *String*.
- "password": contraseña de la cuenta de usuario. Tipo: *String*.

Los emails se podrán repetir en la colección, al igual que las "passwords". Sin embargo, más adelante, por medio de scripts en *Node-RED*, se impedirá que se pueda repetir el campo del nombre.

autor
_id
nombre
email
password

Ilustración 62 Colección "autor".

4.1.2 Colección sensor

La colección “sensor” será utilizada para guardar los diferentes modelos de sensores que estarán asociados a un perfil de ‘’autor’’. Esta colección tendrá como campos:

- “modelo”: modelo identificativo del sensor. Tipo: *String*.
- “localizacion”: coordenadas geográficas de donde se encuentra el dispositivo. Tipo: *String*.
- “autor_id”: “_id” del “autor” al que está asociado el sensor. Tipo: *ObjectId*.

Teniendo en cuenta que es suficiente la identificación de los sensores por el “_id”, debido a que no existe un tipo de suplantación del “_id”, el resto de campos podrá ser repetido.

sensor
_id
modelo
localización
autor_id

Ilustración 63 Colección “sensor”.

Aunque se esté trabajando en bases de datos no relacionales, esto no quiere decir que no existan relaciones entre unas colecciones y otras. La relación que existe entre las colecciones de autor y sensor es la descrita en la tabla x.x., ya que todos los sensores deben estar asociados a un autor específico.

4.1.3 Colección muestra

La colección “muestra” registrará los diferentes datos que transmitirán los sensores en sus muestras. Esta colección tendrá como campos:

- “dato1”, “dato2”,…“daton”: este campo almacenará los diferentes datos que sean enviados por los sensores en sus muestras. Dependiendo del número de datos diferentes que sean enviados, se crearán más o menos campos como datos, teniendo que estar identificados cada uno por el valor que almacenan (por ejemplo, datoTemperatura, datoPresion, datoPotencia…). Tipo: dependiente de si se envía el valor o el valor con la unidad.
- “fecha”: fecha en la que se envió la muestra. Tipo: *String*.
- “hora”: hora a la que se recibió la muestra. Tipo: *String*.

- “sensor_id”: “_id” del sensor al que está asociada la muestra. Tipo: *ObjectId*.

muestra
_id
dato1
dato2
"""
daton
fecha
hora
sensor_id

Ilustración 64 Colección “muestra”.

Al igual que entre las colecciones “sensor” y “autor”, entre “muestra” y “sensor” debe existir una relación para que cada muestra tenga identificado el sensor del que viene.

4.1.4 Colección sesión

Esta colección es más singular que las tres anteriores. En ella se crearán “sesiones” que identificarán cuando un usuario tenga una sesión abierta en la página. Esto se dará cuando el usuario esté dentro de la página. Una vez el usuario haya cerrado sesión, el objeto “sesion” creado al haber iniciado sesión será borrado. Esta colección tendrá los siguientes campos:

- autor_id: campo que hará referencia al “_id” del autor con el que el usuario ha iniciado sesión.
- sesion_id: campo que tendrá un número aleatorio autogenerado en el flujo *Node-RED* para identificar una sesión (se ha intentado usar el “_id” de la propia “sesion” pero con diversos fallos).

sesion
_id
autor_id
sesion_id

Ilustración 65 Colección “sesion”.

La relación entre “sesion” y “autor” es similar a las anteriores.

4.2 Estructura de los flujos Node-RED por pantallas

Una vez explicados cada uno de los componentes previos, se implementarán todos ellos en *Node-RED* para su uso conjunto. Primero, se explicará la función que tiene cada pantalla y luego cómo trabajan los flujos de *Node-RED* con el fin de llegar a ese funcionamiento.

4.2.1 Pantalla login/alta de usuario

Esta es la página inicial que visualizará el usuario. Se basa, en primer lugar, en un HTML que usará el usuario para darse de alta o iniciar sesión (ilustraciones 66 y 67).

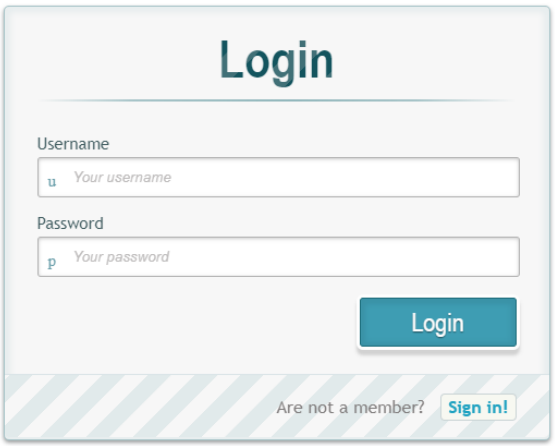

Ilustración 66 Login de usuario.

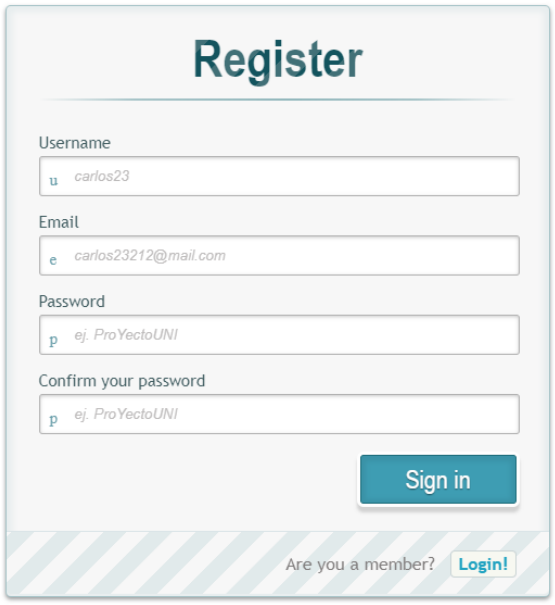

Ilustración 67 Registro de usuario.

El flujo de *Node-RED* que permite estas vistas iniciales es el expuesto en la ilustración 68:

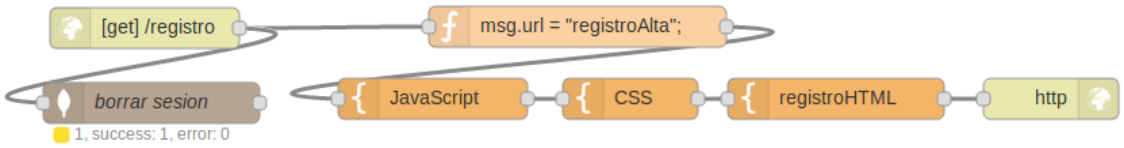

Ilustración 68 Flujo Node-RED para el registro y login de usuario (inicial).

Los módulos que se usan son los siguientes:

- Un módulo *input HTTP* (con función *GET*) para realizar las peticiones *HTTP* oportunas.
- Un módulo *output HTTP* que complemente al módulo anterior y que reciba las respuestas a sus peticiones.
- Un módulo *function* con el objetivo de cambiar la url entrante en el *input HTTP* con el fin de que, a la salida, la respuesta del usuario vaya a un módulo *input HTTP* diferente.
- Un módulo *template* con el nombre de *CSS* donde irán los diferentes *styles* que se necesitan o se quieren para el *HTML*.
- Un módulo *template* con el nombre de *Javascript* donde se incluye el código para usar *Ajax* y que, gracias a él, se hagan las comprobaciones en el *HTML* para verificar que todos los campos estén cumplimentados.
- Un módulo *template* con el nombre de "registroHTML" donde irá el código *HTML*, por el cual el usuario podrá hacer uso de la página web. Se basa en dos formularios: uno para el *login* y otro para el registro, los cuales no se ven simultáneamente, sino que por medio de *CSS* se ''tapa'' a uno con el otro empezando inicialmente con el *login* a la vista. Además, tiene un campo oculto para respuestas negativas en el caso de que no se encuentre un perfil que ya esté creado o que los campos de contraseña y su verificación no sean el mismo, en el cual se escribirán mensajes que veremos posteriormente.
- Un módulo *mongodb2* que borrará la sesión del usuario de la base de datos cuando se haya cerrado sesión desde otra página.

Nota: un módulo *input HTTP* debe estar acompañado de su *output* correspondiente, y el mensaje que debe ir desde un extremo a otro no se puede borrar, debido a que tiene variables de sincronismo que, de no estar al final, el flujo no funcionará.

Una vez mostrado esto, el usuario deberá darse de alta o iniciar sesión y, para ello, se tiene el siguiente flujo (ilustración 69):

Según se venga de una opción u otra, el flujo de datos se dirigirá por distinto camino. Para ello, una vez que llega el mensaje al módulo input *HTTP* (en este caso no se realiza la función *GET*, sino la función *POST*), el siguiente módulo en el camino es un *switch*. Este *switch* busca en el *payload* el campo *username*, el cual está en el formulario del *login* (en el otro formulario el usuario a introducir tiene la etiqueta *usernamesingup*).

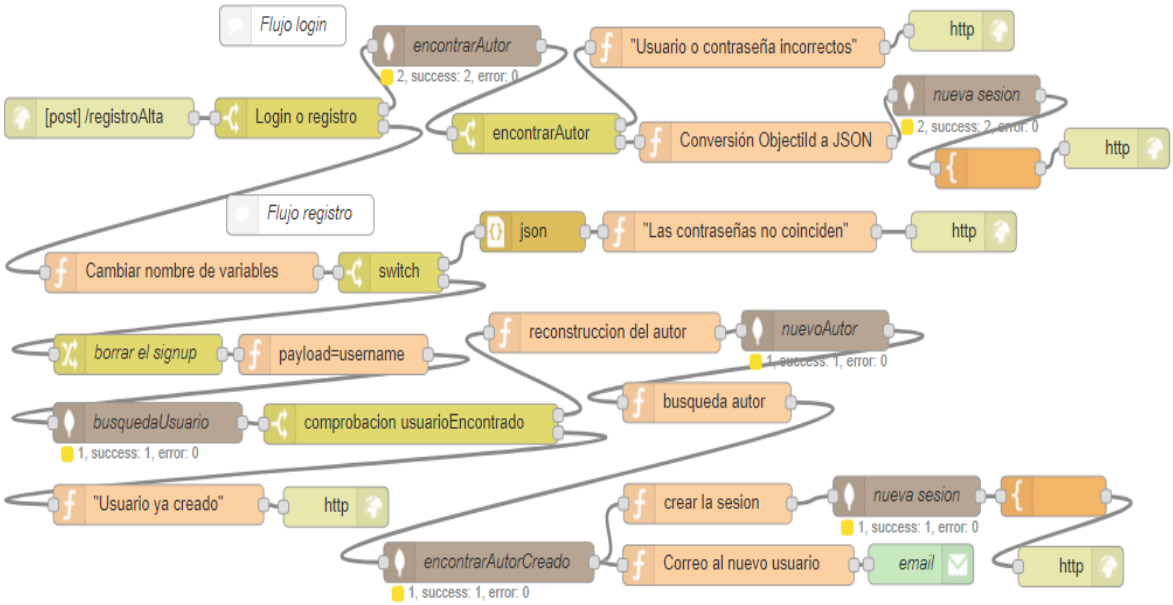

Ilustración 69 Flujo completo POST login/registro

4.2.1.1 Flujo login

En el caso de que el campo "*username"* exista, el paquete poseerá los campos "*username"* y "*password"*. Estos campos serán usados en el siguiente módulo, el cual es un módulo *mongodb2* con la función *findOne()*, lo que devolverá un objeto *JSON* con los campos anteriores más el "*_id"* asociado a ese perfil, que fue creado automáticamente por la base de datos cuando se creó el perfil en el caso de que se encuentre este "*username"* junto a esta "*password"* en la colección autor de la base de datos. En caso negativo, se devuelve un *null* en el *payload*.

Según la respuesta obtenida de la base de datos, en el siguiente módulo (otro *switch*) se diferenciarán dos caminos para la salida si es *null* o no:

En el caso de que sí devuelva null:

1. Se encontrará un módulo *function* que escribirá en el *payload* del *msg* “Usuario o contraseña incorrectos”.
2. Esto tendrá como siguiente módulo un *output HTTP*, el cual escribirá en el campo anteriormente mencionado de respuesta en el *HTML* el mensaje anterior por medio de la función *POST* de *HTML* y del código *Ajax*.

En el caso de que la base de datos no devuelva *null*, se tendrá que usar el campo “*_id”* para, en posteriores flujos, buscar sus sensores asociados. Para ello:

1. Se usa un módulo *function* que añadirá al payload el “*_id*” recuperado del “*autor*” renombrándolo como “***autor_id”***. Además de ello, este módulo también crea un número aleatorio que será utilizado como el identificador de la “*sesion*” creada para el usuario hasta que sea cerrada y, por lo tanto, borrada.
2. Un módulo *mongodb2* que creará una nueva “*sesion*” con los campos anteriores de “*autor_id*”.

A este mensaje resultante le espera un módulo *template*, el cual posee el siguiente *script*:

```
window.location = “./urs2?sesion_id= {{req.body.sesion_id}}”;
```

Gracias a este código, se mostrará la página *HTML* que está en esta url al usuario, la que se comentará posteriormente. Junto a esta url también se envía el “sesion_id” con el fin de poder hacer uso de él en flujos *futuros*. Este código se volverá a usar en otros casos.

Nota: Los campos del *msg.payload* que se quieran mostrar en un *HTML* deben ir entre doble llave y sin *msg: {{payload.x}}*.

4.2.1.2 Flujo registro

Si el mensaje no ha llegado con el campo *username*, el flujo seguirá el segundo camino del primer *switch*.

En el caso de que el flujo llegara hasta el final:

1. Primeramente, se encuentra con un módulo *function* que cambiará el nombre de las variables y añadirá el campo “confirmacion” a *TRUE* si los campos *passwordsingup* y *passwordsingup_confirm* son iguales. Sino, lo devuelve tal cual en la entrada (esto hará que, más adelante, sea rechazado). El siguiente módulo es otro *switch* que comprobará si el campo “comprobacion” está a *TRUE* o no existe.
2. Seguidamente, se busca en la base de datos si el “username” ya existe. Si no es así, se creará un nuevo “*autor*” con los campos “username”, “email” y “password”.
3. Por último, se busca este autor de nuevo para recuperar su “_id” y se renombra como “autor_id” para su uso en el flujo. En este punto, se crean dos flujos donde uno hará la creación de una nueva “sesión” y se abrirá la segunda pantalla al usuario de manera similar a la vista en el flujo *login*. Paralelamente, se enviará al email suministrado por el usuario, un correo electrónico con el “username” y “password” escogidos.

Nota: cuando se hace uso de algunos módulos, como los módulos *mongodb2*, el *msg.payload* resultante es una respuesta a la función entrante, como puede ser una confirmación de que un documento se ha insertado, el objeto que se está buscando o un *null* en caso de que la función no se haya podido realizar. Esto implica que el *payload* anterior a ese módulo sea sustituido por este nuevo. Si se quieren preservar datos a posteriori de estos módulos, en lugar de guardar los datos en el *payload*, se puede usar el campo *msg.req.query.x*, que preservará y permitirá recuperar los datos después de estos módulos.

El flujo se detendrá en caso de que:

- Las contraseñas entrantes no coincidan, para lo que se hará uso de la función *POST* y se le escribirá al usuario el mensaje “Las contraseñas no coinciden”, de manera análoga al caso del flujo *Login*.
- El “username” ya esté creado en la base de datos, se responderá con el mensaje de “El usuario ya existe”.

4.2.2 Pantalla sensores

Una vez que el usuario se haya dado de alta o haya iniciado sesión, se le mostrará una segunda pantalla con todos los sensores que están bajo su cuenta agrupados en una tabla. De estos sensores se mostrará el modelo del sensor, la localización donde se encuentra (ambos suministrados por el usuario) y su identificador propio (provisto por la base de datos). También, en esta pantalla, se podrán añadir nuevos sensores a la base de datos bajo el identificador del usuario y cerrar sesión (ilustración 70).

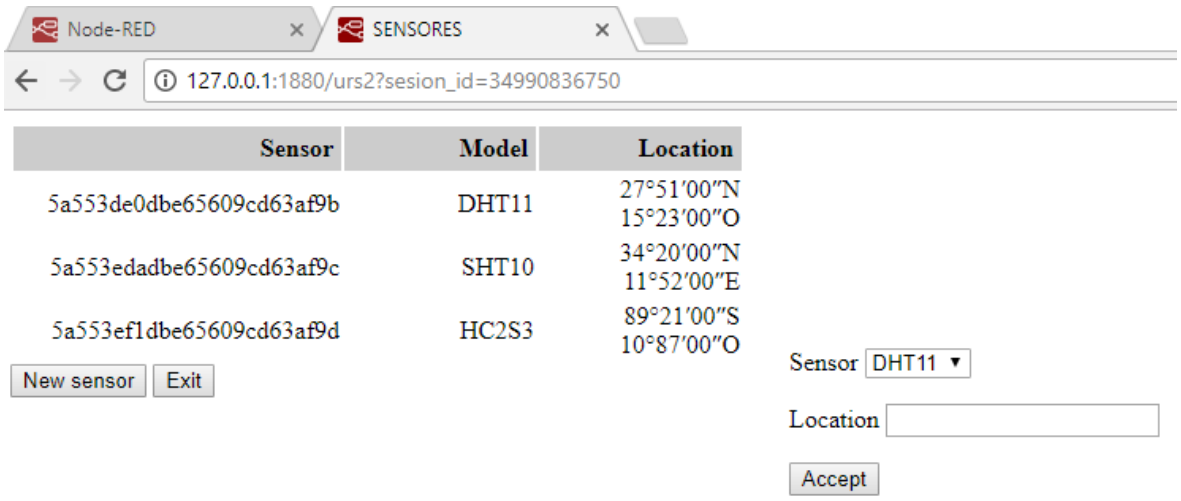

Ilustración 70 Pantalla sensores

El flujo en *Node-RED* que permite todas estas funciones es el mostrado en la ilustración 71:

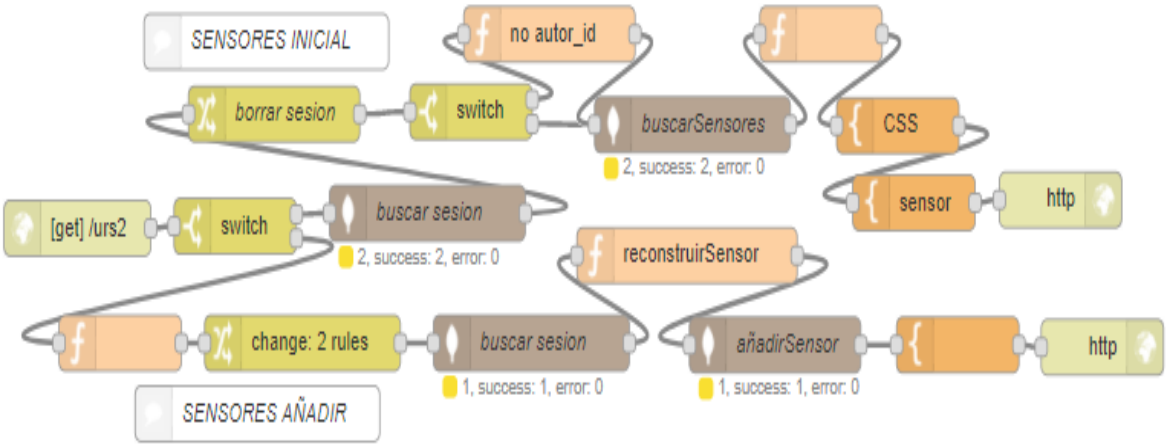

Ilustración 71 Flujo Node-RED para el front-end de sensores.

Este flujo se compondrá de dos caminos en función si el usuario viene de la página de *login*/registro o si ha añadido un nuevo sensor. Esto se diferenciará en el único módulo *switch*, el cual comprobará que el atributo ‘’modulo’’ exista. En caso afirmativo, el paquete estará recibiéndose después de que el usuario haya añadido un nuevo sensor (unos de los valores que debe cumplimentar).

Ambos flujos comparten un módulo *input HTTP* con función *GET*, donde la url es */urs2* y un módulo *output HTTP* que lo complementa (aparte del módulo *switch* que los diferencia).

4.2.2.1 Flujo de sensores inicial

El flujo inicial de los sensores es un flujo compuesto por:

- Un módulo *mongodb2* que, teniendo el campo “sesion_id”, la busca en la colección “sesion” con el fin de poder recuperar el “autor_id” asociado a ella. Los campos referidos a la “sesion” serán borrados del *payload* en el siguiente módulo, ya que, de no ser así, entorpecerían la búsqueda del “autor” posteriormente.

- Un módulo *switch* que difiere si está creado el módulo “autor_id”. Esto es porque, si el usuario cierra sesión, la “sesion” asociada es borrada y, con ella, se pierde el “autor_id” que se consigue con ella. Con esto se consigue que un usuario después de cerrar sesión no pueda añadir nuevos sensores a su perfil.

- Un módulo *mongodb2* que realizará la búsqueda de todos los sensores que tengan el campo ‘’autor_id’’ provisto por la “sesion” como campo. Devolverá un *Array* de objetos *JSON* con todos estos sensores. En el caso que el autor aún no haya añadido sensores, este *Array* estará vacío.

- Un módulo *function* que ejecute la función *JSON.stringify* sobre el paquete (el módulo que realiza las conversiones *JSON* que provee *Node-RED* ha dado fallos para este caso).

- Un módulo *template* con función de *CSS*.

- Un módulo *template* con identificador *sensor* con función *HTML* que se tiene que explicar en profundidad.

4.2.2.1.1 Módulo sensor de la pantalla sensores

El módulo sensor que muestra los sensores asociados a un autor tiene diferentes partes que hay que explicar:

En primer lugar, este módulo es un módulo *template* que tiene código *HTML* y *JavaScript*. Tiene dependencias incluidas de *Web Components* para poder trabajar con la librería *Polymer*.

En segundo lugar, en el *body* del *HTML* se hace referencia a sensor-selector, que es una tabla que se ha creado externamente usando también *Polymer*. Esta tabla dispondrá de *listenners* en los identificadores de los sensores, los cuales estarán "escuchando" en el caso de que se pulse en ellos.

Esta tabla se rellena desde un *script* dentro del módulo. De esto se ocupa el método *_generateTestData*, desde donde se recopilan los datos del *msg.payload* y se escriben en la tabla cada uno de los sensores asociados a la cuenta.

En este *script* también se añade un *listenner* al identificador del sensor, con el fin de que, al pulsar sobre uno, se le muestre al usuario la siguiente página enviando a esta el *sensor_id* para usarlo posteriormente (de forma análoga a la primera pantalla con *autor_id*).

Además de todo esto, el *HTML* también posee un *button* "new" que muestra un *div*, el cual está inicialmente oculto. Este *div* muestra los campos a introducir de "modelo'' y "localizacion" para que el usuario los rellene y se añadan a la base de datos.

Si el usuario rellena los campos y pulsa en el botón "accept" dará comienzo el segundo camino de este flujo.

Por última función, se tiene el botón "exitBtn", que está identificando al botón en la pantalla "Cerrar sesión". Este botón abrirá la pantalla inicial con el campo "sesion_id" adjuntado. Debido a esto, entrará este campo en el módulo *mongodb2* ubicado al comienzo de este flujo y borrará la sesión con el "sesion_id" enviado.

4.2.2.2 Flujo de sensores con sensor añadido

Si el usuario ha rellenado los campos del *div*, la función *window.location* llevará al usuario a esta misma página, pero con los campos "modelo" y "localizacion" en el mensaje. Debido a esto, el módulo *switch* abrirá el segundo camino al flujo de datos, con lo que se encontrará con un módulo *mongodb2,* que insertará estos campos y el "autor_id" a un nuevo sensor en la base de datos.

Antes de esto, se debe volver a buscar la “sesion”, debido a que sin ella no se podrá hacer uso del *autor_id.* Para ello, los campos “modelo” y “localizacion” son ubicados en *msg.req.body.modelo* y *msg.req.body.localizacion,* respectivamente. Ya guardados en ese campo, se borran del *payload,* para poder buscar con “sesion_id” (ubicado en la url) el autor asociado a esta “sesion”. Se recuperan los datos anteriores desde *msg.req.query* hacia el *payload* y se crea un nuevo sensor con ellos junto al “autor_id”.

Una vez hecho esto, se usa una vez más un *template* con la función *window.location,* con el finde recargar la página ya sin los campos del sensor, para que, una vez se recargue, aparezca también el nuevo sensor introducido.

4.2.3 Pantalla muestras

Última pantalla de usuario de la plataforma (ilustración 72). Esta se iniciará desde la selección por el usuario de un sensor en la pantalla sensores, llegando con el *sensor_id* (el “_id” del sensor), además de la *sesion_id*. Aunque en esta pantalla no se necesite el *autor_id*, se recibe la *sesion_id* para poder cerrar sesión también desde esta pantalla de una forma análoga a la de la pantalla anterior.

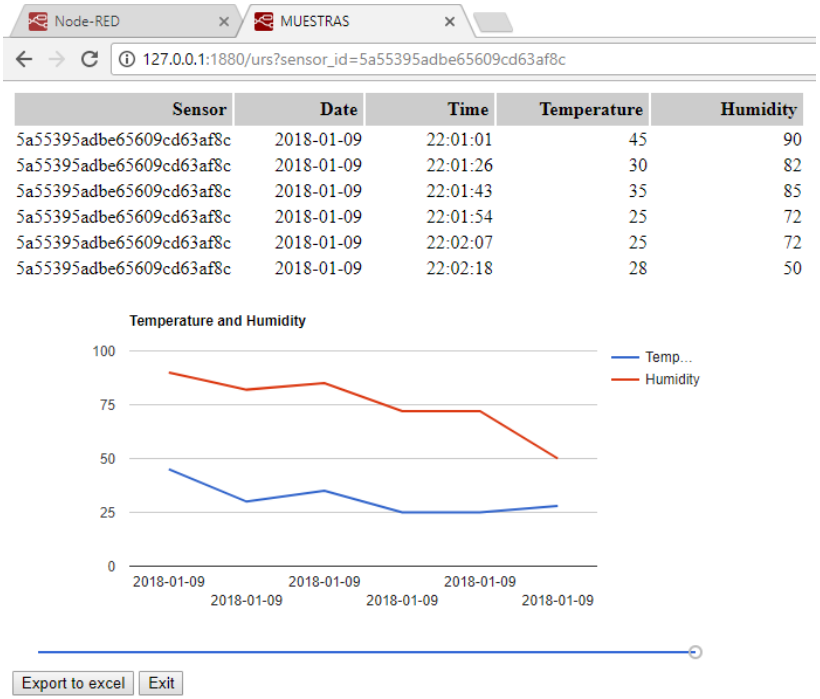

Ilustración 72 Pantalla muestras

En esta pantalla se tiene una tabla similar a la de la pantalla anterior, que mostrará el "_id" de la muestra, la hora y la fecha a la que llegó la muestra y los datos que lleguen. En este caso, a modo de caso de uso, se tendrán como datos posibles de temperatura y humedad.

Además de la tabla, se utilizará un *Web Component* de *Google* que proveerá a la página de un *char* en forma de gráfica, con el fin de que se puedan ver los cambios de los valores de los sensores a lo largo del tiempo. Para esta gráfica se utilizará como complemento otro *Web Component* (esta vez de *Polymer*), en forma de *paper-slider* para poder ver datos en la gráfica anteriores (si no hay suficientes muestras como para usar el *slider*, no funcionará correctamente).

Estos *Web Components* deben cumplimentarse con referencias a sus librerías y con sus propios *styles*.

Por último, el usuario también dispondrá de un *button* que le descargará la tabla con sus valores como un archivo *Excel*.

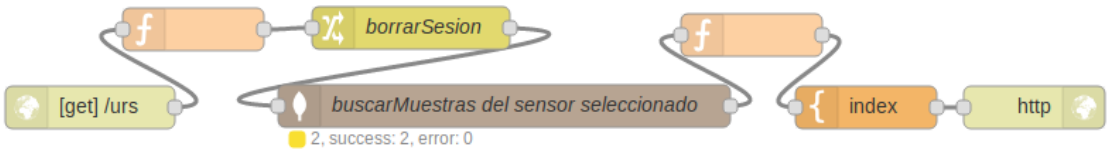

Ilustración 73 Flujo Node-RED para la pantalla muestras.

Para no ser redundante, se explicará directamente el *template index*.

4.2.3.1 Módulo index de la pantalla muestras

Este módulo es similar al *template sensor,* pero con algunas modificaciones. Se tiene, también, una referencia hacia una *table* creada aparte como estática llamada "carlos-table", donde tendrá sólo los valores del identificador de la muestra, la fecha y la hora, ya que los otros se crearán según lleguen en el *template* de forma dinámica (en este caso no, pero se deja preparado para el futuro). También, se encuentra aquí el código para transformar la tabla en un archivo Excel.

Después de esta referencia, se tiene la etiqueta del *chart*, con el fin de generar la gráfica (*line-chart*), donde las *label* serán las variables mostradas en la tabla.

Se tiene más adelante, ya en *script*, el código para rellenar la tabla con los valores que vienen desde el flujo y el código para añadir las dos columnas con el nombre de los datos (se debe llamar igual que las *label* del *char*, sino no habrá comunicación entre ambas entidades).

En este código script también se añade el único *listenner* que hay en esta pantalla, ubicado en el *slider,* para que se ejecute la función propia de este cuando el usuario mueva el círculo sobre la barra de un extremo a otro. A la vez que la gráfica cambia, los valores en la tabla también irán retrocediendo o avanzando en el tiempo.

Para que el *slider* no muestre más valores que el máximo (10) que se muestran en la tabla, se escribe la línea: *this.slider.max = this.data.length – this.data.Length*.

4.2.3.2 Flujo para añadir muestras

Toda la pantalla anterior no se puede llevar a cabo sin tener muestras de sensores. Los demás datos que se añaden en la base de datos, como son los sensores y los autores, los genera el usuario en la página, sin embargo, las muestras vienen de sensores. Para esto se tiene el siguiente flujo:

Ilustración 74 Flujo Node-RED para añadir nuevas muestras.

En este flujo se reciben las muestras por medio de *MQTT*, se añaden los campos de fecha y hora del momento que se reciben las muestras y se añaden a la base de datos. El usuario debe mandar el "sensor_id" como campo en cada muestra. Si no es así, las muestras no se añadirán a su base de datos.

4.2.4 Pantalla de administrador de usuario

La pantalla para el administrador de usuarios se ha hecho de una forma diferente al resto. En este caso, se ha hecho uso del ***dashboard*** (entorno gráfico que ofrecen algunos softwares como los videojuegos), que ofrece *Node-RED* con el fin de crear interfaces gráficas. Este *dashboard* no se ha usado en las demás pantallas, debido a que, aunque permita multitud de usuarios, no permite multitud de sesiones y si un usuario cambia un dato, se cambiará para todos los usuarios.

Sin embargo, para el caso del administrador es de gran utilidad, ya que no habrán diversos usuarios que accedan a esta funcionalidad.

El *dashboard* de *Node-RED* facilita de tal manera el trabajo, que no hace falta programar código. Únicamente con el uso de sus módulos se puede crear una interfaz gráfica con funcionalidades básicas. Uniendo esto a los módulos de *mongodb2*, se obtienen las funcionalidades necesarias para todo administrador de usuarios: buscar, modificar y borrar.

Todos los flujos para buscar un autor, independientemente de la función que se haga, tienen la misma estructura:

- Un formulario donde introducir los datos.
- Un módulo *mongodb2* para buscar el “autor”.
- Un módulo *text* que devuelva lo buscado.

En el flujo de la ilustración 75 se trabaja con un módulo *switch* para, en el caso de que el usuario no exista en la colección “autor”, se muestre una notificación “Usuario no encontrado”. Si existe, se devuelven todos los campos asociados a ese usuario (ilustraciones 76 y 77).

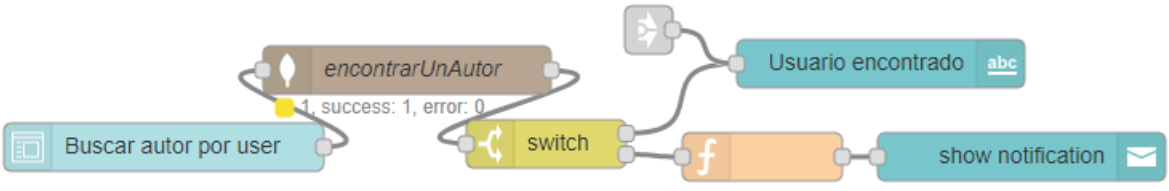

Ilustración 75 Flujo dashboard para buscar un autor.

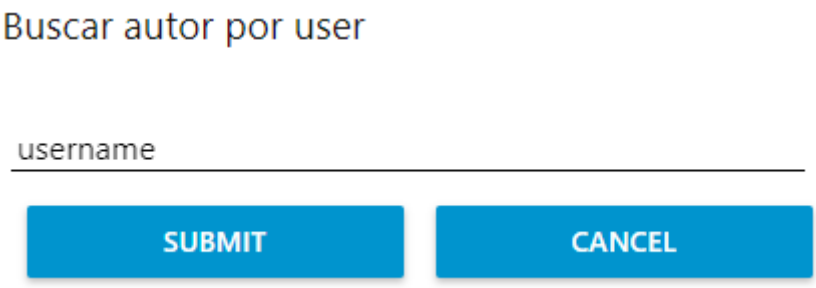

Ilustración 76 Búsqueda de usuarios en el dashboard de Node-RED

Usuario encontrado {"_id":"5a31730fdbe65609cd63af10","username":"carlos","password":"carlos","email":"2@gmail.com"}

Ilustración 77 Usuario encontrado.

Los flujos de modificar y borrar tienen un módulo *function* entre el formulario y la base de datos, con el fin de separar el filtro (el "username" a buscar) del *update* (lo que se vaya a modificar) (ilustración 76). El resultado de estos módulos mongodb2 es que, si la operación se ha hecho, devuelven el documento que antes estaba, aunque haya sido borrado o modificado. Sino, devuelve que no se ha encontrado.

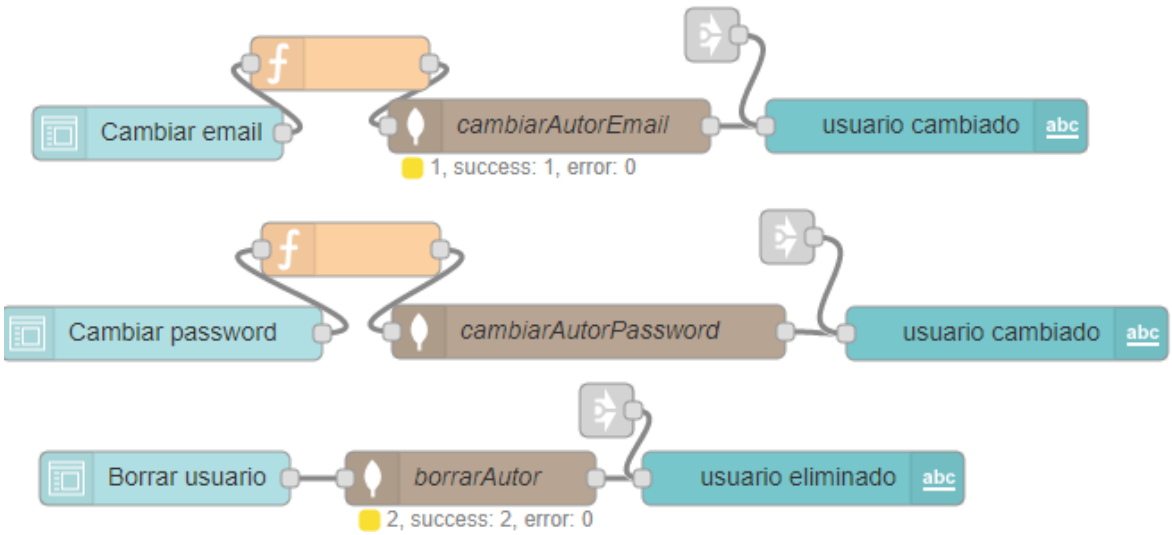

Ilustración 78 Flujos dashboard para modificar y borrar autores.

La última función que podrá usar el administrador será poder recopilar todos los documentos "autor" que estén registrados en la base de datos.

Ilustración 79 Flujo dashboard para devolver todos los autores.

Nota: todos los flujos de *dashboard* han sido conectados por medios de módulos links al módulo *clear* del flujo devolver todos los autores, para tener un único botón *clear* que limpie toda la pantalla de datos anteriores. También hay que comentar que, inicialmente y a pesar de que se pulse clear, en las *label* donde se muestran los resultados está escrito un identificador del *dashboard* que lo pone él por defecto y que da problemas a la hora de intentar quitarlo. Desaparece una vez se muestra un dato pero vuelve a aparecer cuando no hay ninguno.

Capítulo 5. Conclusiones

En este capítulo se describirán los objetivos alcanzados y los resultados logrados en este TFG, así como las líneas futuras de trabajo que se puedan realizar.

5.1 Resultados y revisión de objetivos.

Una vez creada la plataforma para el *IoT*, con todo lo que conlleva, se puede afirmar que se han llegado a lograr los objetivos buscados al inicio de este proyecto.

Primero, se han analizado diferentes tipos de tecnologías en diferentes ámbitos, los cuales han conllevado la elección de un protocolo específico para la comunicación entre los sensores (*MQTT*), la elección de un broker que sirva como infraestructura (Mosquitto), una base de datos para almacenar los datos provistos por los sensores (MongoDB), un entorno gráfico que conecte todas las demás partes (Node-RED) y una interfaz con la que el usuario pueda visualizar los datos (Web *Components*). Esto ha llevado al cumplimento total del primer objetivo.

Una vez hecho esto, en segundo lugar se realizaron pruebas para saber cómo funcionaba el protocolo MQTT sobre el servidor Mosquitto y conocer como enviar los datos y como recibirlos para ser usados. Esto ha llevado al cumplimento total del tercer objetivo.

Para realizar los distintos flujos que componen la parte de la plataforma en Node-RED, se ha debido estudiar los distintos módulos que provee este entorno, tanto inicialmente como los descargados a posteriori. También se ha debido realizar un estudio del lenguaje JavaScript para trabajar con Node-RED, ya sea en módulos de funciones genéricos, como en módulos que se encargaban de vistas de usuario.

Del párrafo anterior se obtiene también la generación de un front-end, realizado con los nombrados Web Components, y de un back-end, realizado con el dashboard de Node-RED. Tener que usar diversos planteamientos implica un mayor conocimiento de diferentes tecnologías y, por lo tanto un mayor aprendizaje. Estos dos últimos párrafos conllevan el cumplimento de los objetivos 2 y 4 de este proyecto.

El resultado final con este proyecto es que se ha obtenido lo que se buscaba inicialmente, dado que se ha podido crear la plataforma con las tecnologías escogidas en un primer momento dando un resultado adecuado y funcional. En determinados momentos, la falta de una buena bibliografía respecto a *Node-RED* ha implicado diversos retrasos, debido a que han surgido problemas entre los mensajes y los módulos en diferentes momentos. Sin

embargo, a base de probar diferentes opciones y observar los campos de los mensajes salientes de cada módulo, se logró una mejor funcionalidad del entorno y comprobar la potencia real de *Node-RED*.

Analizando tecnología por tecnología se obtienen los diferentes resultados:

- ***MQTT***: Su funcionamiento verifica las ideas iniciales que se tenían de este protocolo, ya que no necesita unos grandes requerimientos de sistema para poder funcionar. No se ha estudiado en profundidad las posibilidades de seguridad y cifrado que tiene, pero la funcionalidad que plantea se realiza de una manera ligera y eficaz.

- ***Mosquitto***: se ha comprobado que es un broker de fácil instalación, y que no ha dado fallos más allá de interpretación de caracteres especiales con arreglo sencillo.

- ***MongoDB***: Con una extendida bibliografía, se pudo instalar de manera sencilla y sin problemas en Ubuntu. Viniendo de haber usado siempre bases de datos ***SQL*** (***MySQL***), existía el temor de que fuera un problema usar una tecnología que diferente a estar. El resultado ha sido el contrario, ya que se trabaja de una forma muy sencilla y personalizable en esta base de datos. Sólo se echa en falta que la gran cantidad de funciones que tiene *MongoDB* estén implementadas en el módulo existente en *Node-RED* para esta base de datos y que, además, haya una mejor documentación para poder usar determinadas funciones en los flujos.

- ***Node-RED***: Teniendo claro los pasos a seguir para su instalación, esta resulta bastante sencilla. El uso de módulos evita una gran cantidad de programación, lo que genera una gran satisfacción a la hora de trabajar con este entorno gráfico. El único inconveniente que tiene este entorno es que es un entorno asíncrono, lo que implica que solo se puede trabajar con un flujo a la vez, por lo que no se pueden hacer comprobaciones separando el mensaje principal en dos flujos y que estos converjan, donde el mensaje principal espera a la comprobación. Esto implica que se tenga que idear otras formas con el fin de realizar estas comprobaciones. A pesar de esto, Node-RED queda como una muy buena opción para la creación de este tipo de plataformas.

- ***Web Components***: No se debe de instalar nada para su uso, pero si seguir ciertas premisas para su funcionamiento además de tener en local algunos archivos para poder funcionar. Más allá de esto, se trabaja de una forma sencilla con ellos ya que, mucha de la programación es reutilizada de *Web Components* ya hechos y, en el caso de no estar realizados, existe una muy buena documentación para su uso. Además, no existen problemas de incompatibilidades con código que no sea parte de ellos. Algunos de los archivos que se han generado, como los modelos de las tablas, están fuera de *Node-RED* ya que se debían de tener a nivel local, sin poder tenerlos en el propio flujo.

5.2 Líneas futuras

Una vez implementada la plataforma, se puede pensar sobre qué aspectos implementar o mejorar para cumplir mejor con la satisfacción del usuario, la seguridad o la escalabilidad de la plataforma.

- Para el usuario: crear un menú de configuración donde el usuario pueda tener opciones de personalización, (por ejemplo, diferentes colores para la página web). También se podría implementar la opción de añadir contactos para poder observar las muestras de los sensores de otros usuarios, y también la opción de que ciertos sensores, aun teniéndose agregados los usuarios, se deba de dar una clave para poder acceder a ciertos sensores que se quieran tener más privados.

- Para la seguridad: se podría estudiar de manera más exhaustiva el cifrado de MQTT y de la página web para incrementar la seguridad durante el proceso de envío/recepción de muestras, y evitar problemas de robo de datos durante las transmisiones. Otra línea de interés sería controlar la caché del browser de manera que se pudiese eliminar los datos asociados a un usuario, aunque el usuario realice la función back.

- Para la escalabilidad: tener una base de datos mucho más grande respecto a sensores con el fin de que el usuario tenga más opciones respecto a ellos.

6. Bibliografía

[1] Postscapes. (2016). What exactly is the "INTERNET of THINGS" ?. 19/11/2017, de Postscapes

https://s3.amazonaws.com/postscapes/IoT-Harbor-Postscapes-Infographic.pdf

[2] Postscapes. (2016). Internet of Things (IoT) History. 19/11/2017, de Postscapes

https://www.postscapes.com/internet-of-things-history/

[3] Amazon. (2015). AWS Core IoT Features. 22/03/2017, de Amazon

https://aws.amazon.com/es/iot-core/features/

[4] Microsoft. (2015). Azure IoT Suite: conectando dispositivos y datos | Microsoft. 22/03/2017, de Microsoft

https://www.microsoft.com/es-es/cloud-platform/internet-of-things-azure-iot-suite

[5] Bug Labs. Inc. (2016). freeboard - Dashboards For The Internet Of Things. 21/03/2017, de Bug Labs. Inc

Sitio web: http://freeboard.io/

[6] IBM. (2015). IBM Bluemix -Infraestructura de cloud, servicios de plataforma, Watson y más soluciones para PaaS. 22/03/2017, de IBM

https://www.ibm.com/cloud-computing/bluemix/es

[8] George Ornbo. (2013). Sams Teach Yourself Node.js in 24 Hours. 800 East 96th Street, Indianapolis, Indiana, 46240 USA: Sams.

[9] Rodger Lea. (2016). Node-RED: Lecture 1 – A brief introduction to Node-RED. 15/04/2017, de noderedguide.com

http://noderedguide.com/nr-lecture-1/

[10] OASIS. (2014). MQTT Version 3.1.1. 5/03/2017, de OASIS

http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf

[12] EMQ Enterprise. (2017). EMQ - Erlang MQTT Broker V2.0. 7/03/2017, de EMQ Enterprise

http://emqtt.io/docs/v2/index.html

[13] Introducing HiveMQ, the enterprise MQTT broker. 7/03/2017, de dc-square GmbH

https://www.hivemq.com/hivemq/

[14] Mosquitto. (2017 (última versión)). Mosquitto An Open Source MQTT v3.1/v3.1.1 Broker. 5/03/2017, de Mosquitto

https://mosquitto.org/

[15] MQTT community. (2017 (última versión)). Server support. 12/11/2017, de MQTT community

https://github.com/mqtt/mqtt.github.io/wiki/Server%20support

[16] Paul DuBois. (2013 (Quinta edición)). MySQL. Massachusetts: Addison-Wesley.

[17] MongoDB, Inc.. (2017). MongoDB and MySQL Compared. 5/05/2017, de MongoDB, Inc.

https://www.mongodb.com/compare/mongodb-mysql?jmp=docs

[18] webcomponents.org. (2017). Introduction Web Components. 26/11/2017, de webcomponents.org

https://www.webcomponents.org/introduction

[19] Google. (2017). Architecture Overview. 24/11/2017, de Google Sitio web:

https://angular.io/guide/architecture

[20] Polymer Authors. (2017). Polymer library. 26/11/2017, de Polymer

https://www.polymer-project.org/2.0/docs/devguide/feature-overview

[21] ppatierno. (2013). Message Flow. 11/04/2017, de Embedded101

http://www.embedded101.com/Develop-M2M-IoT-Devices-Ebook/DevelopM2MIoTDevicesContent/ArticleId/224/3-7-Message-Flow

[22] npm, Inc.. (2017). What is npm?. 7/04/2017, de npm, Inc.

https://docs.npmjs.com/getting-started/what-is-npm

[23] JS Foundation. (2017). User Guide Configuration. 3/12/2017, de JS Foundation

https://nodered.org/docs/configuration

Parte II

Presupuesto

Presupuesto

En este capítulo se recoge el presupuesto total del presente TFG. Para ello se han seguido las recomendaciones del Colegio Oficial de Ingenieros Técnicos de Telecomunicación (a partir de ahora, COITT) sobre los baremos orientativos para trabajos profesionales desde 2009. Este presupuesto se ha organizado en varios apartados para poder diferencias los distintos costes derivados del desarrollo de este TFG:

1. Recursos materiales.
2. Recursos humanos.
3. Gastos en material fungible.
4. Derechos de visado del COITT.
5. Aplicación de impuestos.

P.1 Recursos materiales

Debido a que en este proyecto se ha intentado siempre trabajar con un software open-source, sólo se tendrán que tener en cuenta el equipo hardware utilizado, su sistema operativo y los paquetes software que se han usado para la redacción de esta memoria.

El coste de amortización viene estipulado en un plazo de tres años. Para realizar este cálculo, se hará uso de un sistema de amortización lineal o constante, donde el bien material que se tenga, va perdiendo valor a lo largo de vida útil de una forma constante, quedando como valor residual el valor que se le da a un material una vez que ha sobrepasado su vida útil. Según la ecuación P.1:

$$Cuota\ anual = \frac{Valor\ de\ adquisición - Valor\ residual}{Años\ vida\ útil} \qquad \text{(Ec. P.1)}$$

P.2 Recursos hardware

Se han utilizado en el proceso de este proyecto las siguientes herramientas hardware (tabla 3):

Ordenador portátil Lenovo de 15.6”, con procesador Intel Core i7, memoria RAM de 8 GB y con 1 TB de capacidad de disco duro.

COSTES DE LAS HERRAMIENTAS HARDWARE				
Descripción	Valor de adquisición	Valor residual	Vida útil	Valor amortización (4 meses)
Ordenador portátil Lenovo de 15.6”	699 €	200€	3 años	33.33€

Tabla 3 Costes de las herramientas hardware.

P.3 Recursos software

Las herramientas software que se hayan usado en este proyecto, aparte de las nombradas en el propio proyecto, son:

Microsoft Office 2010 para Windows. Como este software esta suministrado por la ULPGC, no hay gastos en recursos software.

P.4 Recursos humanos

Para realizar este proyecto se han invertido alrededor de 300 horas en diseño, desarrollo y elaboración de documentación. Según el COITT, para calcular el valor del trabajo empleado se ha de calcular según la ecuación P.2:

$$H = C_t * 74.88 * H_n + C_t * 96.72 * H_e \qquad \text{(Ec. P.2)}$$

Teniendo:

- H: Honorarios totales recibidos por el proyecto.
- C_t: factor de corrección dependiente del número de horas trabajadas.
- H_n: Horas trabajas en horario laboral.
- H_e: Horas trabajas fuera de horario laboral (en este proyecto no han existido así que su valor es 0)

Teniendo también en cuenta los criterios del COITT, el factor de corrección es el siguiente según las horas trabajadas (tabla 4):

HORAS EMPLEADAS	FACTOR DE CORRECION C_t
X < 36	1
36 < x < 72	0.90
72 < x < 108	0.80
108 < x < 144	0.70
144 < x < 180	0.65
180 < x < 360	0.60
360 < x < 540	0.55

Tabla 4 Valores de la variable Ct en función de las horas trabajadas.

Según esta tabla, al estar este proyecto finalizado en 300 horas, se deberá de escoger el valor del factor de corrección dado para el intervalo de las 180-360 horas: $C_t = 0.60$. Dado esto, la fórmula anterior quedaría:

$$H = 0.6 * 74.88 * 300 + 0.6 * 96.72 * 0 = 13478.40\ €$$

Los honorarios derivados del tiempo dedicado al proyecto libres de impuestos ascienden a unos **TRECE MIL CUATROCIENTOS SETENTA Y OCHO EUROS CON CUARENTA CÉNTIMOS (13478.40 €).**

También se debe de tener en cuenta los honorarios derivados de la redacción del TFG. Su cálculo se rige según la ecuación P.3:

$$R = 0.07 * P * C_n \qquad \text{(Ec. P.3)}$$

Teniendo:

- P: presupuesto del trabajo.
- C_n: Coeficiente de ponderación dependiente del presupuesto del trabajo.

El presupuesto del trabajo es la suma de lo anteriormente calculado por ahora (tabla 5):

Descripción	Costes
Recursos hardware	176.10€
Trabajo tarifado por tiempo empleado	13478.40 €
Total	13654.50 €

Tabla 5 Presupuesto de los costes materiales y tarifado por tiempo empleado.

Se obtiene que el presupuesto total hasta el momento es de 13654.50 €. De aquí a terminar el presupuesto no van a existir grandes desembolsos, y dado que el coeficiente de ponderación definido por el COITT para presupuestos menores de 30050€ es de 1, tenemos que:

$$R = 0.07 * 13654.50 \ * \ 1 = 955.81€$$

Por tanto, el coste libre de impuestos derivado de la redacción del TFG es de **NOVECIENTOS CINCUENTA Y CINCO EUROS CON OCHENTA Y UN CÉNTIMOS (955.81€)**

P.5 Gastos en material fungible

Aparte de los gastos derivados del material hardware (y software si hubiera), todos los Trabajos también han de usar otros materiales como son folios y tinta para impresiones, denominados material fungible (tabla 6):

Descripción	Costes
Folios	10€
Tóner de impresora	50€
Total	60€

Tabla 6 Costes del material fungible.

P.6 Derechos del visado del COITT

Los gastos del visado del COITT según la ecuación P.4 son:

$$V = 0.006 * P * \ C_v \qquad \text{(Ec. P.4)}$$

Teniendo

- P: Presupuesto del trabajo
- C_v: Coeficiente reductor en función del presupuesto del trabajo.

P es el presupuesto hasta este momento, dado por la suma de los costes de ejecución material, redacción y material fungible:

$$P = 13654.50 + \ 955.81 + 60 = 14670.31 \ €$$

Como el coeficiente de ponderación definido por el COITT para presupuestos menores de 30050 € es de 1, tenemos que:

$$V = 0.006 * 14670.31 \ * \ 1 = 88.02 \text{ €}$$

Por tanto, el coste de los derechos de visado del COIT para este Trabajo ascienden a **OCHENTA Y OCHO EUROS CON DOS CÉNTIMOS (88.02 €).**

Como último gasto fijo, tenemos fijados los gastos de tramitación y envío en 6.01 €

P.7 Aplicación de impuestos

El coste total de este TFG asciende a 14670.31 € libres de impuestos. Se le ha de sumar el 7% de IGIC, con lo que el coste definitivo de este TFG asciende a (tabla 7):

COSTES TOTALES DEL TRABAJO FIN DE GRADO	
Descripción	**Coste total (€)**
Recursos materiales	176.1
Coste de ingeniería	13654.50 €
Coste de redacción	955.81
Material fungible	60
Derechos de visado	88.02
Tramitación y envío	6.01
Aplicación de impuestos (7% IGIC)	1033.50
Total	15797.84

Tabla 7 Presupuesto total del Trabajo Fin de Grado.

Carlos Espacios López, autor del presente Trabajo de Fin de Grado, declara que:

El Trabajo Fin de Grado con título "**DISEÑO DE UNA PLATAFORMA EN LA NUBE CON NODE-RED PARA INTERNET DE LAS COSAS**", desarrollado en la Escuela de Ingeniería de Telecomunicaciones y Electrónica, de la Universidad de las Palmas de Gran Canaria, tiene un coste de desarrollo total de **QUINCE MIL SETECIENTOS NOVIENTA Y SIETE EUROS CON OCHENTA Y CUATRO CÉNTIMOS (15797.84 €),** correspondiente a la suma de las cantidades consignadas a los apartados considerados previamente.

Las Palmas de Gran Canaria, a 9 de enero de 2018

Firma: Carlos Espacios López

Parte III

Anexos

Anexo I

Instalaciones

Anexo I. Instalaciones necesarias para realizar el proyecto

Para realizar este proyecto, se han debido de realizar distintas instalaciones y configuraciones para poder usar los programas nombrados en otros capítulos como Mosquitto, Node-RED y la base de datos MongoDB. En este anexo se explica el proceso de instalación necesario en Ubuntu 16.04 y la configuración necesaria para poder usar estos componentes.

Nota: todas las instalaciones de este proyecto son para el S.O. Ubuntu 16.04. Además, cada vez que se haga un apt-get, el siguiente comando debe de ser sudo apt-get update con el fin de tener la versión más reciente.

A1.1 Instalación Mosquitto.

Los comandos que se han de seguir para instalar *Mosquitto* son los siguientes:

1. *sudo apt-add-repository ppa:mosquitto-dev/mosquitto-ppa*
2. *sudo apt-get install mosquitto*
3. *sudo apt-get install mosquitto-clients*

A1.2 Instalación Node-RED

Para instalar *Node-RED* también se tienen que tener en cuenta las dependencias de Node.js. Antes de nada, se debe de instalar el software *curl*, ya que su librería *libcurl* es bastante útil para automatizar transferencias de archivos o secuencias de operaciones no supervisadas.

sudo apt-get install curl

A continuación, se instala las dependencias de Node.js y seguidamente las de Node-RED.

1. *curl –silent –location https://deb.nodesource.com/setup_4.x | sudo bash*
2. *sudo apt-get install node.js*
3. *sudo npm install –g node-red*

Llegado este punto ya se tiene todo lo necesario para poder trabajar en Node-RED.

A1.3 Configuración de dependencias de Web Components en Node-RED

Para hacer uso de los Web Components, hacen falta distintos paquetes que se han de tener en local para poder satisfacer sus dependencias en el código. Entre ellos se encuentran archivos propios de los Web Components o archivos estáticos creados por el diseñador.

Se debe de realizar un simple proceso para tener un directorio local desde el cual poder usar contenido web estático en Node-RED [22]:

En primer lugar, se debe de buscar donde está ubicada la carpeta .node-red (normalmente, en la carpeta user).

Una vez se está en esa carpeta, se debe de abrir el archivo settings.js para eliminar la habilitación de comentarios de la siguiente línea (ilustración 78):

```
httpStatic: '/home/carlos/nol/node-red-static/',
```

Ilustración 80 Habilitación de contenido web estático en Node-RED..

Una vez está habilitada, los archivos que estén dentro de la última carpeta serán servidos ante peticiones *HTTP*. Aquí se colocaran las dependencias de los *Web Components* y los archivos que se decidan crear para uso en estático.

A1.4 Instalación de MongoDB en Ubuntu

Para empezar la instalación de *MongoDB* en *Ubuntu*, se debe importar la clave pública utilizada por el sistema de gestión de paquetes.

Nota: Todos los comandos para esta instalación se deben de ejecutar en el directorio /etc/apt/sources.list.d

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
0C49F3730359A14518585931BC711F9BA15703C6
```

Se debe de crear el archivo mongodb-org-3.4.list usando el comando dependiente de la versión de Ubuntu que se esté usando. En este caso la versión usada es 16.04, así que la orden quedaría así:

```
echo “deb [ arch=amd64,arm64 ] http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.4 multiverse” | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list
```

Se debe de hacer uso del comando update en este punto. Para finalizar, se instala la última versión estable

```
sudo apt-get install –y mongodb-org
```

A1.5 Módulos a instalar en Node-RED

Para trabajar con MongoDB en Node-RED existen módulos creados por la comunidad para hacer uso de estas bases de datos dentro de la propia plataforma. Se deberá acceder al menú principal de Node-RED, pulsar sobre Manage Palette, y se accederá a dos columnas: una son todos los módulos instalados en el Node-RED actual, yla otra permite instalar módulos nuevos desde Internet. El módulo usado en este proyecto para la función de MongoDB es el mostrado en la ilustración 79:

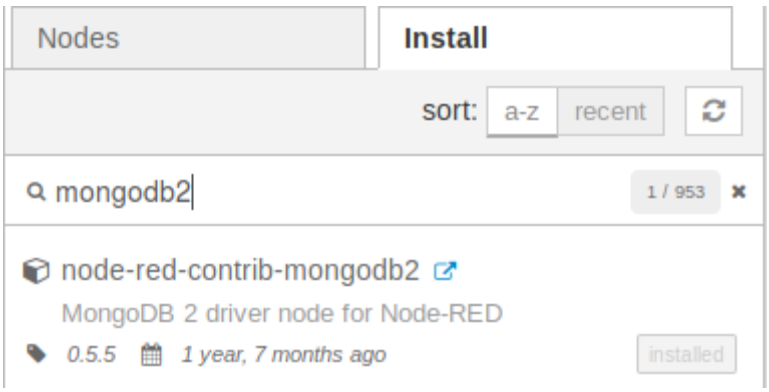

Ilustración 81 Módulo usado de MongoDB para Node-RED.

Para instalar los módulos que hacen de *dashboard* en *Node-RED* se sigue el mismo procedimiento instalando el módulo mostrado en la ilustración 80:

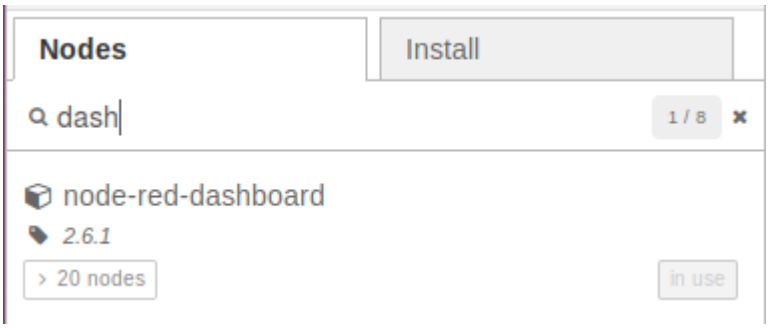

Ilustración 82 Módulo usado de para el dashboard de Node-RED.

Anexo II

Manual de usuario

A2. Manual de usuario

Este es un documento donde se explicará cómo usar la plataforma, poder registrar sensores, y guardar las muestras que provean cada uno de ellos.

A2.1 Inicio sesión

En el caso de que ya se tenga un perfil de usuario creado, simplemente se deben de completar el formulario inicial con su username y su password para poder entrar en su perfil (ilustración 66).

En el caso de que aún no se tenga un perfil de usuario creado, se debe de seleccionar el botón “Sign in!” donde encontrará un formulario (ilustración 67). Una vez se haya rellenado los campos pedidos, se debe de volver seleccionar el botón “Sign in”. Se enviará un correo electrónico al email cumplimentado para confirmar el alta.

Para cerrar sesión habrá un botón en las pantallas posteriores para este uso (se aconseja siempre cerrar sesión antes de abandonar la página)

A2.2 Añadir nuevos sensores

Una vez se esté en la pantalla de sensores (ilustración 70), para añadir nuevos sensores, se debe de pulsar “New sensor” para añadirlos uno a uno. Aparecerán un menú donde se escogerá un sensor de los que en ese momento la página soporte, y la ubicación geográfica donde se quiera ubicar. Una vez hecho, se debe de pulsar en “Accept” para confirmar el nuevo sensor. La página recargará y el nuevo sensor se añadirá a la tabla con los campos seleccionados y el identificador provisto por la página.

A2.3 Muestras de sensores

Para observar las diferentes muestras de cada sensor, se debe de pulsar sobre el identificador. Se mostrará una nueva pantalla (ilustración 72). Aquí se podrán ver las diferentes muestras que el sensor ha enviado a la plataforma desde que se dio de alta. El slidebar ubicado debajo de la gráfica sólo funcionará si hay un número adecuado de muestras para realizar su función (más de 10 muestras). Se podrá además descargar como archivo Excel esta tabla de muestras.

A2.4 Envío de muestras

A la hora de enviar las muestras de los sensores para ser guardadas en la base de datos de la plataforma, se deben de enviar con determinado formato para su correcto procesamiento. En esta primera versión de la plataforma, esta sólo acepta sensores que envíen datos de temperatura y humedad. Los datos deben de ser enviados siempre con la misma escala y sin letras.

El formato del mensaje es el siguiente:

*mosquitto_pub –h **localhost** –t "añadirMuestra" –m "{\"datoTemperatura\":\"x\",\"datoHumedad\":\"y\",\"sensor_id":\"z\"}"*

Donde los datos son los suministrados por el sensor, y el sensor_id es el identificador con el que se nombra al sensor una vez se crea en la segunda pantalla de la página web.

Nota: la dirección a la que enviar las muestras dependerá de la que se quiera reservar para recibir las muestras en el momento de arrancar la plataforma, además de también reservar determinado puerto para ello también. Por ejemplo, *–h 10.10.20.13 -p 1024*.