ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA INFORMÁTICA

UN MARCO DE DESARROLLO PARA LA PROGRAMACIÓN DE APLICACIONES DE INTERNET DE LAS COSAS EN EL BORDE DE LA RED

AN EDGE COMPUTING FRAMEWORK FOR THE DEVELOPMENT OF INTERNET OF THINGS APPLICATIONS

Realizado por

Manuel Granados Molina

Tutorizado por

Bartolomé Rubio Muñoz,
Cristian Martín Fernández

Departamento

Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

Málaga, Junio de 2019

Fecha de defensa:
El Secretario del Tribunal

Resumen

Palabras claves: TFG, IoT, Lambda, CoAP, MQTT, Node-RED, Kafka, SmartGateway, ZigBee.

En este Trabajo Fin de Grado se ha analizado y programado, mediante Node-RED, un marco de desarrollo visual que permite la programación de aplicaciones IoT sobre Edge Computing.

Este marco permitirá la monitorización de los dispositivos IoT del sistema, la realización de determinados algoritmos con los datos obtenidos y la actuación sobre tales dispositivos. El sistema es extensible con el fin de que pueden incluirse nuevas funcionalidades en un futuro sin necesidad de modificar las existentes.

Además se ha desarrollado un servidor web asíncrono que permite la interconexión asíncrona de IoT con el marco de desarrollo.

Este sistema provee una serie de componentes y herramientas que permiten la programación de aplicaciones abstrayendo a los programadores de las subscripciones a los dispositivos IoT.

Finalmente se han realizado pruebas con datos reales en el laboratorio para comprobar el correcto funcionamiento del sistema.

Abstract

Keywords: TFG, IoT, Lambda, CoAP, MQTT, Node-RED, Kafka, SmartGateway, ZigBee.

In this project, a visual development framework that allows the programming of IoT applications on Edge Computing has been analyzed and programmed through Node-RED.

This framework will allow the monitoring of the IoT devices of the system, the realization of certain algorithms with the data obtained and the interaction with such devices. The system is extensible so that new functionalities can be included in the future without the need to modify the existing ones.

In addition, a web server has been developed that allows the asynchronous interconnection of the IoT with the development framework.

This system provides a series of components and tools that allow the programming of applications by abstracting the programmers of the subscriptions to the IoT devices.

Finally, tests have been performed with real data in the laboratory to evaluate the correct functioning of the system.

Índice general

Índice de figuras

Índice de cuadros

Acrónimos

UMA	Universidad de Málaga
ETSII	Escuela Técnica Superior de Ingeniería Informática
TFG	Trabajo Fin de Grado
IoT	Internet of Things
CoAP	Constrained Application Protocol
ERTIS	Embedded Real-Time Systems
HTTP	HyperText Transfer Protocol
REST	REpresentational State Transfer
UART	Universal Asynchronous Receiver-Transmitter
I2C	Inter-Integrated Circuit
CC	Cloud Computing
EC	Edge Computing

Capítulo 1

Introducción

1.1. Contexto y motivación

El término Internet de las Cosas (IoT – Internet of Things) hace referencia a la interconexión digital de objetos cotidianos o "cosas" con internet. Estos objetos pueden realizar medidas sobre su entorno en el mundo real y transferir esa información a través de internet para poder ser monitorizados y controlados. La idea principal de IoT es obtener información sobre el entorno y poder actuar sobre el mismo. Este concepto ha ido desarrollándose y evolucionando en los últimos años debido a los continuos avances en el campo.

IoT es una tecnología que está en pleno auge, ya que cada vez hay más dispositivos con sensores, etiquetas RFID y actuadores, capaces de obtener información de diversa índole (temperatura, humedad, contaminación, vibraciones, etc.), compartirla, manipularla y producir actuaciones sobre el entorno, que pueden ser controlados desde diferentes plataformas, como por ejemplo un smartphone. IoT puede aplicarse en una gran variedad de aspectos en nuestro entorno, como puede ser a casas inteligentes (smart homes), ciudades inteligentes (smart cities) o monitorización de infraestructuras críticas.

En los últimos años, el IoT ha sido integrado con tecnologías de Computación en la Nube (CC – Cloud Computing)[1] con el fin de paliar las limitaciones de sus dispositivos empotrados. Esta integración ha permitido complementar el IoT con grandes capacidades

de almacenamiento y procesamiento. Sin embargo, esta integración ha traído consigo una gran dependencia en otros factores, como las redes de comunicaciones.

La gran evolución que plantea el IoT supone grandes desafíos para estas infraestructuras en términos de ancho de banda y la necesidad de una baja latencia para actuaciones en tiempo real. Recientemente, un paradigma conocido como Computación en el Borde (EC – Edge Computing) plantea situarse entre el IoT y el Cloud para abordar estos desafíos.

El grupo de investigación ERTIS de la Universidad de Málaga ha estado investigando sobre la integración de IoT y CC en los últimos años. Como resultado, ha desarrollado la arquitectura Lambda-CoAP mostrada en la figura 1.1.

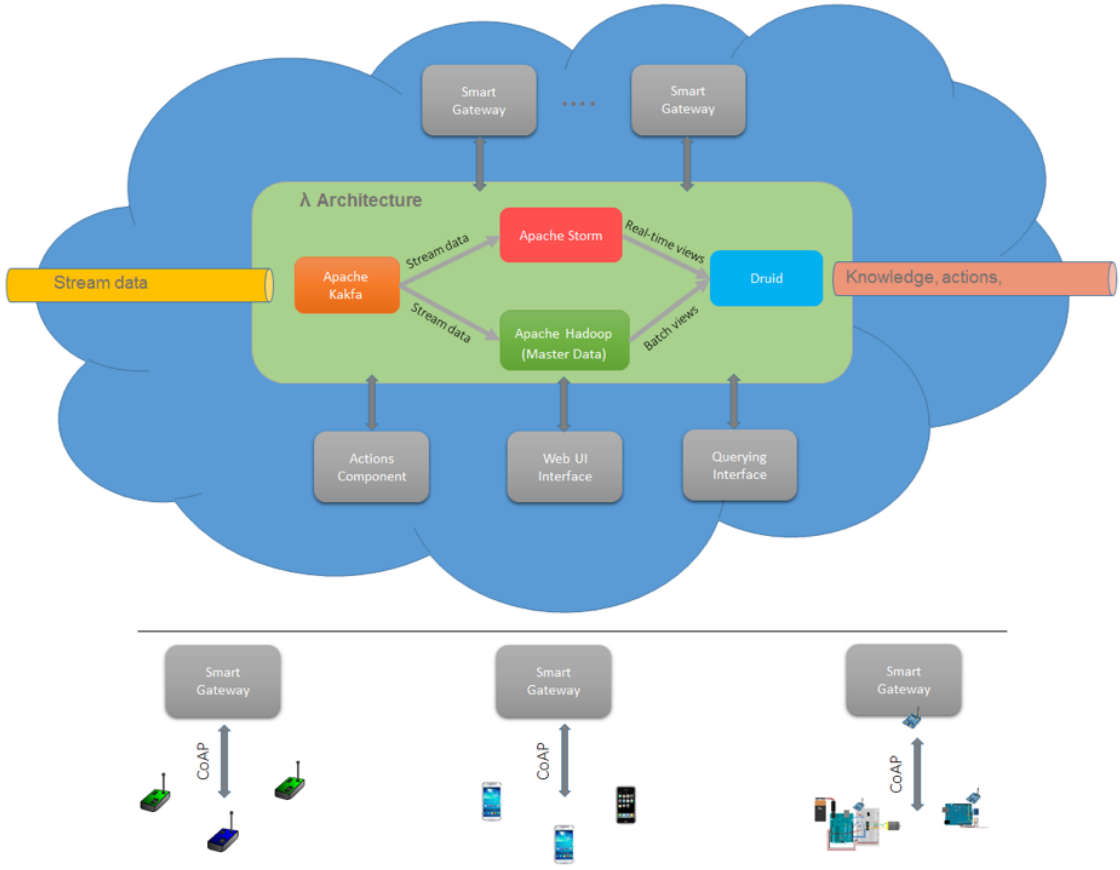


Figura 1.1: Arquitectura Lambda-CoAP, integración de IoT y CC.

Esta arquitectura permite integrar dispositivos empotrados, como placas de desarrollo Arduino, con CC. La arquitectura hace uso del protocolo para servicios web en recursos empotrados CoAP [2], y una arquitectura de CC que permite procesar grandes cantidades de datos en tiempo real, conocida como la arquitectura Lambda.

Estos componentes son a su vez conectados por una pasarela inteligente (Smart Gateway, en inglés), que ofrece protocolos de más alto nivel para acceder a los dispositivos IoT como HTTP y envía los datos suscritos a la arquitectura Lambda. Aunque esta pasarela permite integrar los dispositivos de IoT con el cloud, la arquitectura resultante es una arquitectura centralizada, representando un desafío para las redes de comunicaciones como hemos visto.

Este proyecto pretende lidiar con el anterior problema de la arquitectura Lambda-CoAP y plantea proporcionar a las pasarelas inteligentes la capacidad de realizar procesamiento con el fin de aligerar el envío de datos al cloud. Este paradigma, como hemos introducido, es conocido como Edge Computing. La arquitectura Lambda-CoAP resultante tras la realización de este proyecto viene reflejada en la figura 1.2.

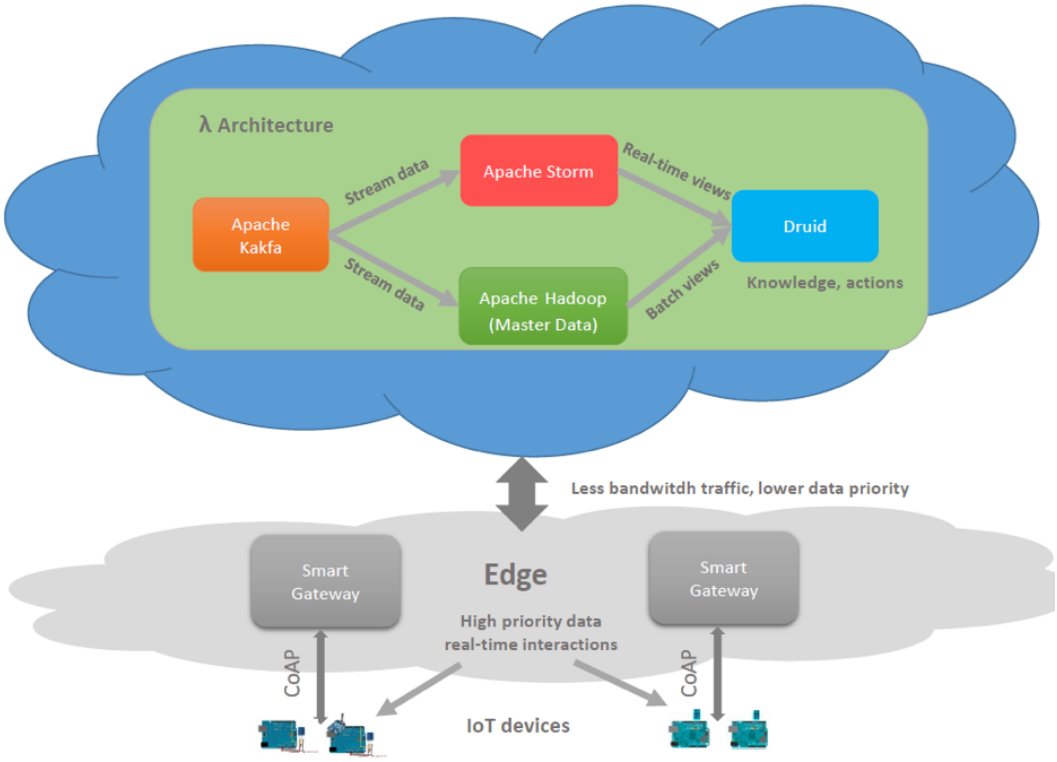


Figura 1.2: Arquitectura Lambda-CoAP con Edge Computing.

Esta nueva arquitectura incluirá capacidades de procesamiento en el borde de la red, permitiendo aligerar las redes de comunicaciones de la gran cantidad de datos que tiene previsto producir el IoT.

Las pasarelas inteligentes están compuestas por una serie de componentes para permitir las mencionadas funcionalidades. Este proyecto pretende proveer las capacidades de Edge Computing a la pasarela inteligente con la inclusión de un marco de desarrollo para la programación de aplicaciones y la habilitación de un mecanismo de comunicación asíncrona y bidireccional para que puedan suscribirse a los eventos de los dispositivos de IoT y puedan actuar sobre ellos.

En la figura 1.3 se muestra la pasarela inteligente resultante con la inclusión de estos nuevos componentes. La subscripción a los dispositivos de IoT debe de realizarse de forma abstracta a los programadores de aplicaciones, es decir, el marco de desarrollo se subscribirá automáticamente a las actualizaciones de dispositivos y datos dependiendo de la lógica definida en el marco de desarrollo.

Esta abstracción permitirá a los desarrolladores de aplicaciones centrarse únicamente en la lógica de las aplicaciones, mientras que el marco de desarrollo controlará las subscripciones a datos y dispositivos en el sistema. Además, el marco de desarrollo deberá de ser visual, con el fin de reducir su curva de aprendizaje y mejorar los resultados obtenidos de la plataforma en un amplio abanico de usuarios.

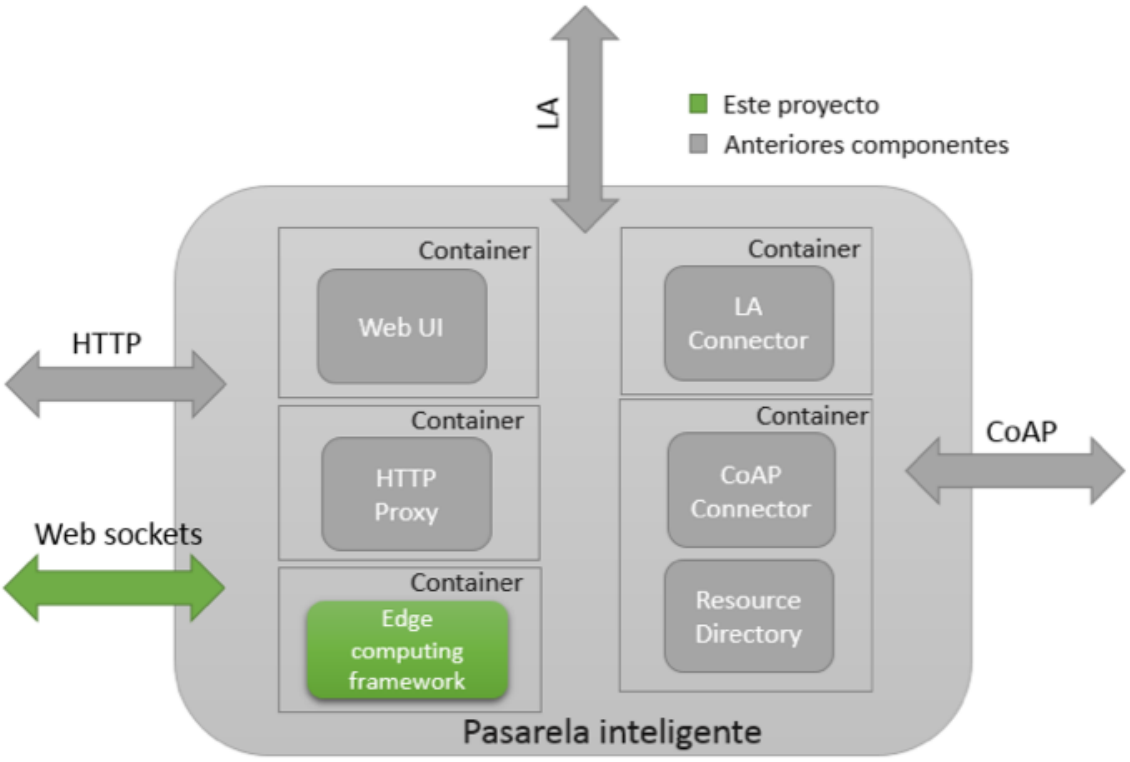


Figura 1.3: Ampliaciones de la pasarela inteligente en este proyecto.

El marco de desarrollo permitirá la monitorización de los dispositivos IoT del sistema, la programación de algoritmos con los datos obtenidos y la actuación sobre tales dispositivos. También, el sistema deberá ser extensible con el fin de que puedan incluirse nuevas funcionalidades en un futuro sin la necesidad de modificar las funcionalidades existentes.

Este marco se realizará mediante Node Red[3], una herramienta gráfica de flujo de datos de desarrollo para IoT creada por IBM y basada en la plataforma Node JS.

Las principales ventajas que se tiene al utilizar Node Red son:

1. Libre de costo y protegida por la licencia Apache.
2. Integración por defecto con el PaaS de IBM.
3. Baja curva de aprendizaje.
4. Tipo de programación similar a JavaScript.
5. Muy ligero de instalar y utiliza recursos de computo mínimo.

El integrarse con la nube representa una gran ventaja para realizar aplicaciones con sensores y actuadores y tener la capacidad de monitorizar remotamente a través de internet.

En Node Red todos los flujos se basan en el siguiente precepto básico:

- Una entrada.
- Un procesamiento.
- Una salida.

1.2. Objetivos

El objetivo de este proyecto es proveer un marco de desarrollo visual que permita la programación de aplicaciones IoT sobre Edge Computing. Para ello se definen los siguientes sub-objetivos:

1. El sistema deberá proveer una serie de componentes y herramientas que permitan la programación de aplicaciones abstrayendo a los programadores de las subscripciones a los dispositivos IoT.

2. Realización de un servidor web asíncrono que permita la interconexión asíncrona de IoT con el marco de desarrollo.

3. El sistema deberá poder ser instalado en dispositivos de sistema en chip como Raspberry Pi.

4. Interconexión del marco de desarrollo con la arquitectura Lambda.

5. Estudio de algoritmos para la decisión de envío de datos al cloud.

6. Despliegue real en los laboratorios que el grupo ERTIS tiene en el edificio de I+D Ada Byron.

1.3. Estructura de la memoria

Este trabajo fin de Grado está organizado en 10 capítulos. A continuación se describirán los aspectos principales tratados en cada uno:

- *Capítulo 2: Metodología*

  En este capítulo se expone la metodología utilizada para el desarrollo del proyecto.

- *Capítulo 3: Tecnologías usadas*

  En este capítulo se explica brevemente las tecnologías y protocolos utilizados en este proyecto.

- *Capítulo 4: Visión General*

  En este capítulo se realiza una visión general de la infraestructura empleada para este proyecto.

- *Capítulo 5: Análisis*

  En este capítulo se desarrolla un análisis del proyecto, estableciendo los diferentes elementos del sistema, características y requisitos de cada uno de ellos.

- *Capítulo 6: Especificación*

  En este capítulo se elabora una especificación del proyecto a través de los requisitos extraídos en el análisis del proyecto. Se realiza el modelo de dominio y los respectivos casos de uso.

- *Capítulo 7: Diseño*

  En este capítulo se define el diseño llevado a cabo para el sistema, tanto la arquitectura como el formato de los datos.

- *Capítulo 8: Implementación*

  En este capítulo se muestra la implementación aplicada, a través del análisis, la especificación y el diseño especificados en el sistema.

- *Capítulo 9: Evaluación*

  En este capítulo se expone las pruebas de rendimiento realizadas y una breve aplicación real que puede tener este proyecto en la vida cotidiana.

- *Capítulo 10: Conclusiones y líneas futuras*

  En este capítulo se indicarán las conclusiones obtenidas tras finalizar el proyecto y las líneas futuras para una posible continuación del mismo.

Capítulo 2

Metodología

En este capítulo se va detallar la metodología utilizada para el desarrollo de este TFG.

Se conoce por metodología de un proyecto al proceso seguido para gestionar las actividades que se realizarán siguiendo unos requisitos y pasos con el fin de conseguir un camino de trabajo optimizado.

Un proyecto es una unidad única de trabajo, en la que se realiza una gestión de recursos disponibles para alcanzar el objetivo especifico del proyecto. Para ello se establece un periodo de tiempo directo en la planificación del mismo. Los pasos generales para la elaboración de un proyecto se pueden definir como [4]:

- **Definir el objetivo y la necesidad** que nos lleva a realizar el proyecto. No se debe perder de vista ese objetivo fijado en todo el procedimiento.
- **Identificar la información y los recursos con los que se dispone**. En los primeros pasos es fundamental descubrir todas las posibles dudas lo mas rápido posible a través de las preguntas que se formulen los distintos colaboradores del proyecto.
- **Fase de investigación y planificación**. En esta fase aumentan los detalles y requisitos de nuestro proyecto pudiéndose asignar recursos específicos disponibles a las actividades planificadas.
- **Planificación colaborativa**. El equipo de trabajo comparte los requisitos planifi-

cados y se definen los objetivos concretos de cada actividad.

- **Revisión y control continuo**. Gracias al tablero de actividades planificado en su momento se podrán ver las tareas pendientes, las que están retrasadas, hechas y validadas. Ante la duda de variar la fecha de alguna de esas actividades, el tablero ayuda a estudiar el impacto que podrá tener ese cambio en el proyecto.

En este proyecto se ha intentado seguir una metodología ágil tomando como producto la idea de Cristian y considerarlo a él como "cliente", con el propósito de ofrecer pequeñas soluciones regularmente y ajustarse a los nuevos cambios emergidos entre tanto durante el desarrollo del proyecto.

2.1. Metodología ágil

El método ágil es un proceso que permite adaptar la forma de trabajo a las condiciones del proyecto, suministrando respuestas rápidas a las pequeñas tareas que componen el proyecto. Cada una de esas tareas se enmarcan en iteraciones o sprint, que son pequeños periodos de trabajo de 1 a 3 semanas por lo general.

Figura 2.1: Ciclo de vida de un proyecto ágil.

Al comienzo del proyecto, nos reunimos Cristian y yo para estudiar los requisitos iniciales que necesitaba el sistema y se detallaron las correspondientes tareas a realizar para satisfacer esos primeros requisitos.

Al principio de cada sprint, llamado "Sprint Planning", se examinaron los requisitos por si hubiera que cambiarlos y se desarrollan los casos de uso correspondientes. Al acabar el sprint se reúne de nuevo y se realiza el "Sprint Review" donde se ha podido ver el alcance obtenido tras las semanas de trabajo y se ofrece un pequeña solución con los requisitos implementados como parte del proyecto. Cristian observa en ese momento si la solución se corresponde con lo que el quería inicialmente y si los requisitos que propusieron eran útiles. De esta manera se entregan soluciones rápidas y de fácil mantenimiento en cada interacción, que es el concepto esencial de las metodologías ágiles.

Capítulo 3

Tecnologías utilizadas

En este capítulo se describen las tecnologías utilizadas para el desarrollo del proyecto. Se mencionarán los aspectos fundamentales de cada tecnología, así como las decisiones tomadas para su utilización en el proyecto.

3.1. Tecnologías inalámbricas

Podemos definir la tecnología inalámbrica como aquella que establece la comunicación sin la necesidad de utilizar una conexión física, es decir, sin cables mediante ondas electromagnéticas.

Este concepto puede tener origen en el famoso físico alemán Heinrich Rudolf Hertz, quien descubrió la propagación de las ondas electromagnéticas así como la forma de producirlas y detenerlas.

Años posteriores Guillermo Marconi impulsado por la idea de Hertz, realizó una serie de experimentos con éxito a través de su invento más famoso, la radio, considerándose así el primer caso de comunicación inalámbrica.

Existen una gran cantidad de tecnologías que han sido desarrolladas a lo largo de estos años, las cuales se clasifican básicamente en 3 tipos (Figura 3.1) según el área de cobertura de cada una [5]:

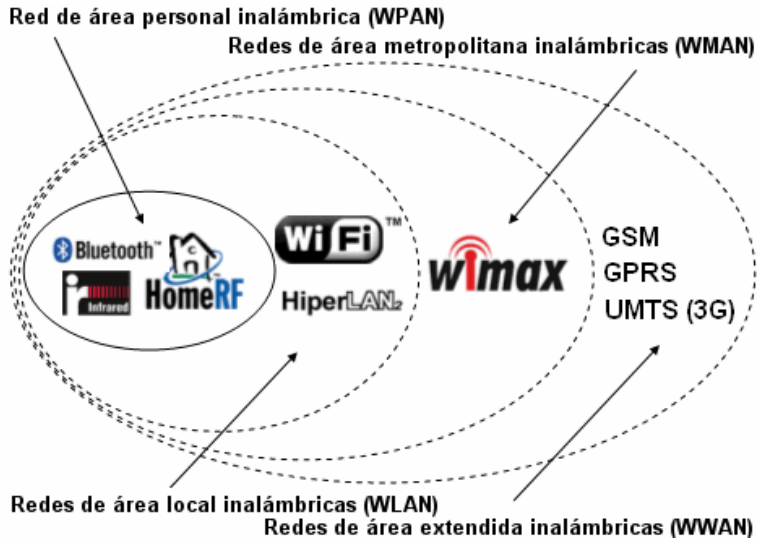


Figura 3.1: Clasificación de las tecnologías inalámbricas.

- **WPAN: Wireless Personal Area Network**. Redes inalámbricas de área personal
- **WLAN: Wireless Wide Area Network**. Redes inalámbricas de área local
- **WMAN: Wireless Metropolitan Area Network**. Redes inalámbricas de área metropolitana

A continuación se detallará la tecnología usada en este proyecto utilizada principalmente en viviendas y entornos industriales.

3.1.1. Zigbee

ZigBee es una especificación basada en el estándar IEEE 802.15.4 para redes de área personal inalámbricas (WPAN) [6]. ZigBee opera en las bandas de radio ISM, y define una red de malla de propósito general, de bajo costo, que puede ser utilizada para control industrial, recolección de datos médicos, advertencia de humo e intrusos, automatización de edificios y domótica, etc.

Hay tres tipos diferentes de dispositivos en una red ZigBee:

- **Dispositivo de funciones completas - Coordinador ZigBee (Maestro):** Solo existe un coordinador en cada red ZigBee. Su función es almacenar información

sobre la red y determinar la ruta de transmisión óptima entre cualquiera de los dos nodos de la red.

- **Dispositivo de función completa - ZigBee Router (esclavo):** El enrutador actúa como un nodo intermedio que siempre está en el modo activo. Su función principal es transferir y pasar los datos entre los dispositivos.
- **Dispositivo de función reducida - Dispositivo final ZigBee (esclavo):** Este dispositivo contiene una cantidad mínima de funcionalidad para permitirle hablar con su nodo principal (ya sea el coordinador o un enrutador); normalmente está en modo de suspensión y no puede transmitir datos directamente desde otros dispositivos.

Hay tres topologías definidas en el estándar ZigBee: Malla, Estrella y Árbol de clústeres (Figura 3.2).

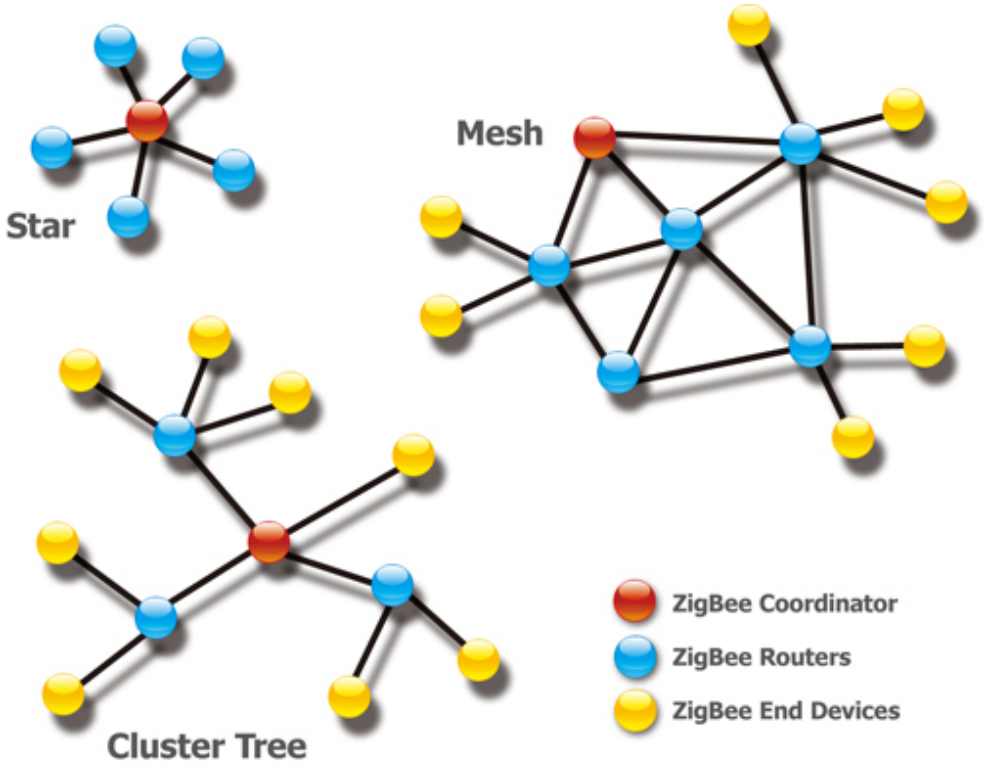


Figura 3.2: Topologías Zigbee.

La red ZigBee utiliza una configuración básica maestro-esclavo que se adapta a las redes de malla dinámica de muchos dispositivos de uso poco frecuente que hablan a través de pequeños paquetes de datos. Se permiten hasta 256 nodos.

3.2. Protocolos de aplicación

A continuación se describen dos de los protocolos más prometedores de la IoT para pequeños dispositivos. Estos son MQTT y CoAP.

Ambos comparten varias características como son:

- Son estándares abiertos
- Se adaptan mejor a entornos restringidos que HTTP
- Proporcionan mecanismos de comunicación asíncrona.
- Se ejecutan en IP
- Disponen de una gama de implementaciones.

MQTT proporciona flexibilidad en los patrones de comunicación y actúa puramente como una tubería para datos binarios. CoAP está diseñado para la interoperabilidad con la web [7].

3.2.1. MQTT

MQTT es un protocolo de mensajería de publicación/suscripción diseñado para comunicaciones M2M ligeras. Fue desarrollado originalmente por IBM y ahora es un estándar abierto.

Tiene un modelo cliente/servidor, donde cada sensor es un cliente y se conecta a un servidor, conocido como intermediario, a través de TCP.

Está orientado a mensajes. Cada mensaje es un fragmento de datos discreto, opaco para el intermediario. Este se publica en una dirección, conocida como un tema. Los clientes pueden suscribirse a múltiples temas. Cada cliente suscrito a un tema recibe todos los mensajes publicados sobre el tema.

Por ejemplo, imaginemos una red simple con tres clientes y un intermediario central. Los tres clientes abren conexiones TCP con el broker. Los clientes B y C se suscriben al

tópico ”temperature”(Figura 3.3).

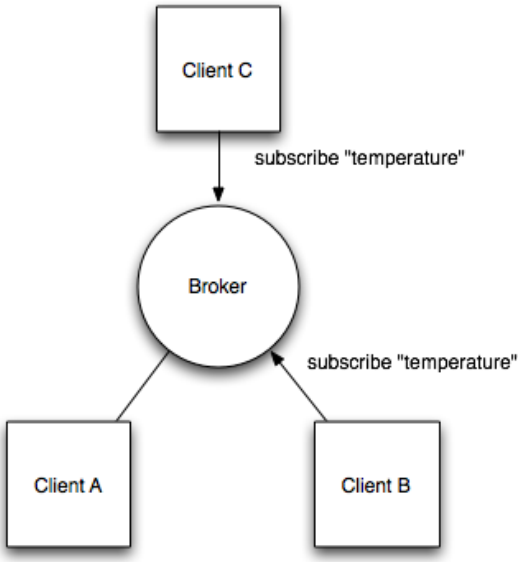


Figura 3.3: Suscripción MQTT A.

Posteriormente, el Cliente A publica un valor de 22.5 del tópico ”temperature”. El agente reenvía el mensaje a todos los clientes suscritos (Figura 3.3).

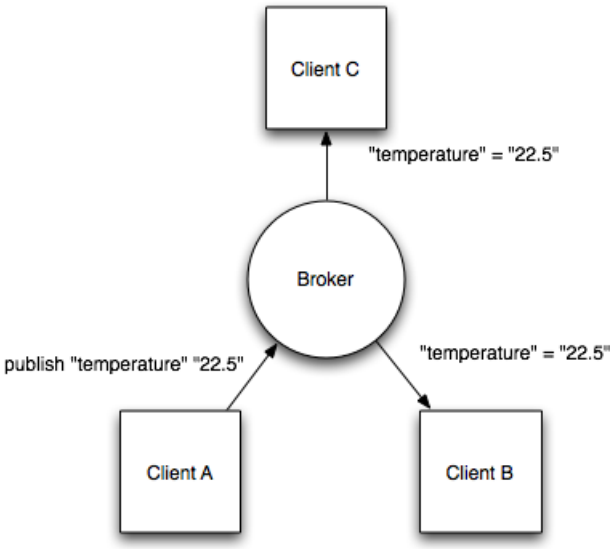


Figura 3.4: Suscripción MQTT B.

Este modelo permite que los clientes MQTT se comuniquen uno a uno, uno a muchos y muchos a uno.

3.2.2. CoAP

CoAP es el protocolo de aplicación para dispositivos empotrados del grupo IETF CoRE (Entornos de recursos restringidos).

Al igual que HTTP, CoAP es un protocolo de transferencia de documentos. A diferencia de HTTP, CoAP está diseñado para las necesidades de los dispositivos restringidos.

Los paquetes CoAP son mucho más pequeños que los flujos HTTP-TCP. Los campos de bits y las asignaciones de cadenas a enteros se utilizan ampliamente para ahorrar espacio. Los paquetes son fáciles de generar y se pueden analizar en su lugar sin consumir RAM extra en dispositivos restringidos.

CoAP se ejecuta sobre UDP, y por tanto, los clientes y servidores se comunican a través de datagramas sin conexión. Los reintentos y el reordenamiento se implementan en la pila de aplicaciones. Eliminar la necesidad de TCP puede permitir redes IP completas en microcontroladores pequeños. CoAP permite utilizar la difusión UDP y la multidifusión para el direccionamiento.

CoAP sigue un modelo cliente/servidor. Los clientes hacen solicitudes a los servidores y los servidores envían respuestas. Los clientes pueden hacer peticiones GET, PUT, POST y DELETE a los recursos. Está diseñado para interactuar con HTTP y la red RESTful a través de simples proxies. Debido a que CoAP está basado en datagramas, se puede usar sobre SMS y otros protocolos de comunicaciones basados en paquetes.

3.2.3. Comparación entre ambos protocolos

MQTT y CoAP son útiles como protocolos de IoT, pero tienen diferencias fundamentales.

MQTT es un protocolo de comunicación de muchos a muchos para pasar mensajes entre múltiples clientes a través de un intermediario central. Desacopla al productor y al consumidor al permitir que los clientes publiquen y que el intermediario decida dónde enrutar y copiar los mensajes. Si bien MQTT tiene cierto soporte para la persistencia, funciona mejor como un bus de comunicaciones para datos en vivo.

CoAP es, principalmente, un protocolo uno a uno para transferir información de estado entre el cliente y el servidor. Si bien es compatible con los recursos de observación, CoAP es más adecuado para un modelo de transferencia de estado, no puramente basado en eventos.

Los clientes MQTT realizan una conexión TCP saliente de larga duración a un intermediario. Esto generalmente no presenta ningún problema para los dispositivos detrás de NAT. Los clientes y servidores CoAP envían y reciben paquetes UDP. En los entornos NAT, se puede usar la tunelización o el reenvío de puertos para permitir el CoAP, o los dispositivos pueden iniciar primero una conexión a la cabecera.

MQTT no proporciona soporte para etiquetar mensajes con tipos u otros metadatos para ayudar a los clientes a entenderlo. Los mensajes MQTT se pueden usar para cualquier propósito, pero todos los clientes deben conocer los formatos de los mensajes por adelantado para permitir la comunicación. CoAP, a la inversa, proporciona soporte integrado para la negociación y el descubrimiento de contenido, lo que permite a los dispositivos sondearse entre sí para encontrar formas de intercambiar datos.

Ambos protocolos tienen ventajas y desventajas, elegir el correcto depende de su aplicación. En nuestro caso, ha sido elegido CoAP ya que permite una mejor integración con la web, admitiendo un mayor numero de dispositivos con conexiones diferentes.

3.3. Sistema de mensajes distribuidos (Kafka)

Apache Kafka es un sistema de intermediación de mensajes basado en el modelo productor/consumidor. El sistema tiene como objetivo proporcionar una plataforma unificada, de alto rendimiento y de baja latencia para la manipulación en tiempo real de fuentes de datos. [8]. La figura 3.5 muestra el logotipo del lenguaje.

Figura 3.5: Logotipo de Apache Kafka.

Apache Kafka ha sido utilizado en el proyecto para permitir a los usuarios conectar sus aplicaciones desarrolladas con el cloud.

Apache Kafka es una plataforma de transmisión que tiene tres capacidades clave [9]:

- Se publica y se suscribe a flujos de registros, similar a una cola de mensajes o un sistema de mensajería empresarial.
- Se almacenan flujos de registros de forma duradera y tolerante a fallos.
- Se procesan flujos de registros a medida que se producen.

Kafka se usa generalmente para dos amplias clases de aplicaciones:

- Construir flujos de datos en tiempo real para obtener datos de manera confiable entre sistemas o aplicaciones.
- Crear de aplicaciones de transmisión en tiempo real que transforman o reaccionan a los flujos de datos.

Para entender cómo Kafka hace estas cosas, analicemos y exploremos las capacidades de Kafka desde abajo hacia arriba comentando primero algunos conceptos:

- Kafka se ejecuta como un clúster en uno o más servidores que pueden abarcar varios centros de datos.
- El clúster Kafka almacena flujos de registros en categorías llamadas temas.
- Cada registro consta de una clave, un valor y una marca de tiempo.

Kafka tiene cuatro funciones principales como podemos ver en la figura 3.6:

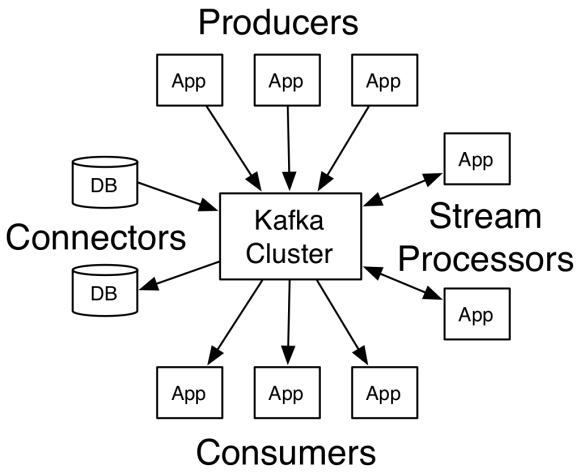


Figura 3.6: APIs Kafka.

- La función del productor permite que una aplicación publique una secuencia de registros a uno o más temas de Kafka.
- La función del consumidor permite que una aplicación se suscriba a uno o más temas y procese la secuencia de registros que se les ha producido.
- La función de flujos permite que una aplicación actúe como un procesador de flujo, consumiendo un flujo de entrada de uno o más temas y produciendo un flujo de salida a uno o más temas de salida, transformando efectivamente los flujos de entrada en flujos de salida.
- La función del conector permite crear y ejecutar productores o consumidores reutilizables que conectan los temas de Kafka a las aplicaciones o sistemas de datos existentes. Por ejemplo, un conector a una base de datos relacional podría capturar cada cambio en una tabla.

3.4. Python

El lenguaje de programación Python es un lenguaje de programación multiparadigma creado en el año 1991 por Guido van Rossum. Está influido por varios lenguajes de programación entre los que destacan C, Haskell y Java [8].

Al ser multiparadigma soporta orientación a objetos, programación imperativa y programación funcional. Su principal característica es que tiene una sintaxis que favorece un código legible. Es un lenguaje interpretado y multiplataforma. Todo el código en Python se organiza en clases con extensión «.py». La figura 3.7 muestra el logotipo del lenguaje.

Figura 3.7: Logotipo de Python.

Python ha sido utilizado en el proyecto para el desarrollo del servidor websockets. Se seleccionó este lenguaje porque parte del servidor que tenia realizado Cristian estaba hecho con este lenguaje.

3.5. Sublime Text

Sublime Text es un editor de texto y código fuente desarrollado en C++ y Python para los plugins. Fue desarrollado por Jon Skinner en 2008 como una extensión de Vim, pero con el tiempo fue cogiendo identidad propia. La figura 3.8 muestra el logotipo de la aplicación.

Figura 3.8: Logotipo de Sublime Text.

Sublime Text ha sido utilizado en el proyecto para el desarrollo de los módulos de Node-RED.

3.6. PyCharm CE

PyCharm CE es un entorno de desarrollo integrado (IDE) utilizado en la programación, especialmente para el lenguaje Python. Proporciona análisis de código, un depurador gráfico, un probador de unidades integrado, integración con sistemas de control de versiones y es compatible con el desarrollo web con Django [8]. La figura 3.9 muestra el logotipo de la aplicación.

Figura 3.9: Logotipo de PyCharm CE.

Fue creado por la compañía checa JetBrains en 2010. PyCharm CE es multiplataforma y existen dos ediciones del programa, The Community Edition, publicada bajo la licencia de Apache, y Professional Edition, lanzada bajo una licencia propietaria con características adicionales.

PyCharm CE ha sido utilizado en el proyecto para el desarrollo del servidor websocket.

3.7. Node-RED

Node-RED es una herramienta de programación visual. Muestra visualmente las relaciones y funciones y permite al usuario programar sin tener que escribir prácticamente nada de código. Es un editor de flujo basado en el navegador donde se puede añadir o eliminar nodos y conectarlos entre sí con el fin de hacer que se comuniquen entre ellos [10]. La figura 3.10 muestra el logotipo de la herramienta.

Figura 3.10: Logotipo de Node-RED.

Node-RED ha sido utilizado en el proyecto para realizar el marco de desarrollo que permite la monitorización de los dispositivos IoT del sistema.

3.8. Draw.io

Draw.io es un software gratuito de diagramas en línea para hacer diagramas de flujo, diagramas de proceso, organigramas, UML, ER y diagramas de red. La figura 3.11 muestra el logotipo de la herramienta.

Figura 3.11: Logotipo de Draw.io.

Draw.io ha sido utilizado en el proyecto para realizar todos el diagrama del modelo de dominio, de los casos de uso y los diagramas de clases.

3.9. Fritzing

Fritzing es una herramienta de software libre para el modelado de prototipos, diseño y fabricación de pcbs profesionales. La figura 3.12 muestra el logotipo de la aplicación.

Figura 3.12: Logotipo de Fritzing.

Fritzing ha sido utilizado en el proyecto para realizar el esquema del cableado del prototipo del nodo utilizado.

3.10. Numbers

Numbers es una aplicación de hojas de cálculo creada por Apple Inc.. Es utilizada en tareas financieras y contables, con fórmulas, gráficos y un lenguaje de programación. La figura 3.13 muestra el logotipo de la aplicación.

Figura 3.13: Logotipo de Numbers.

Numbers ha sido utilizado en el proyecto para realizar las gráficas con los datos de las pruebas de rendimiento.

3.11. iMovie

iMovie es una aplicación de software de edición de vídeo creada por Apple Inc., por el cual se permite a los usuarios editar sus propios vídeos de forma profesional desde casa. La figura 3.14 muestra el logotipo de la aplicación.

Figura 3.14: Logotipo de iMovie.

iMovie ha sido utilizado en el proyecto para la edición del montaje del video DEMO realizado.

Capítulo 4

Visión General

En este capítulo se realiza una visión general de la infraestructura empleada para este proyecto detallando los dos componentes principales: Nodo y SmartGateway.

4.1. Dispositivo IoT

El sistema aplica una red IoT cuya función principal es compartir recursos a los distintos elementos de la red por medio de tecnologías inalámbricas, permitiendo la modificación de los periféricos de los nodos mediante el protocolo CoAP sin tener que reconfigurar el dispositivo.

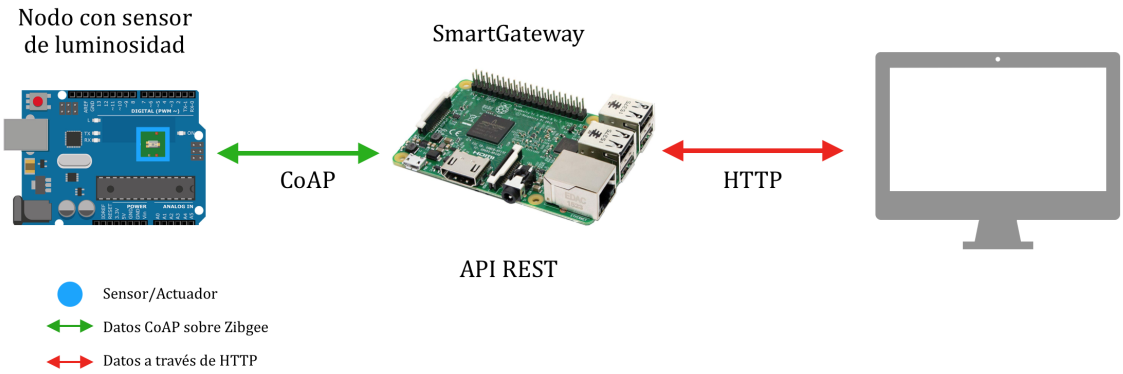


Figura 4.1: Visión general de la arquitectura del proyecto.

En la figura 4.1 observamos una visión general de los elementos que constituyen la arquitectura aplicada a este proyecto, así como las comunicaciones, tecnología y protocolo

implicados en entre cada una. Se distinguen dos componentes, Nodo y SmartGateway, donde la conexión entre ellos se realiza mediante ZigBee utilizando CoAP.

4.2. Nodo

El nodo, siendo el elemento más importante de la arquitectura, es el que menor capacidad de procesamiento y consumo tiene. Consiste en un servidor CoAP que obtiene datos en tiempo real a través de sus sensores.

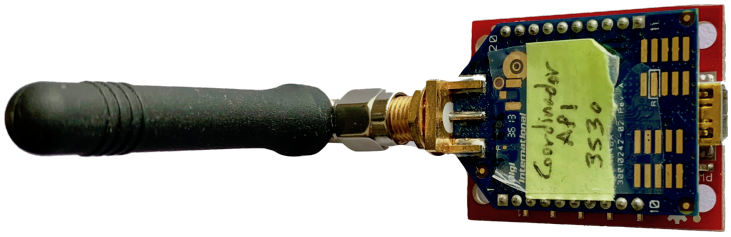


Figura 4.2: Coordinador Zigbee incorporado en la SmartGateway.

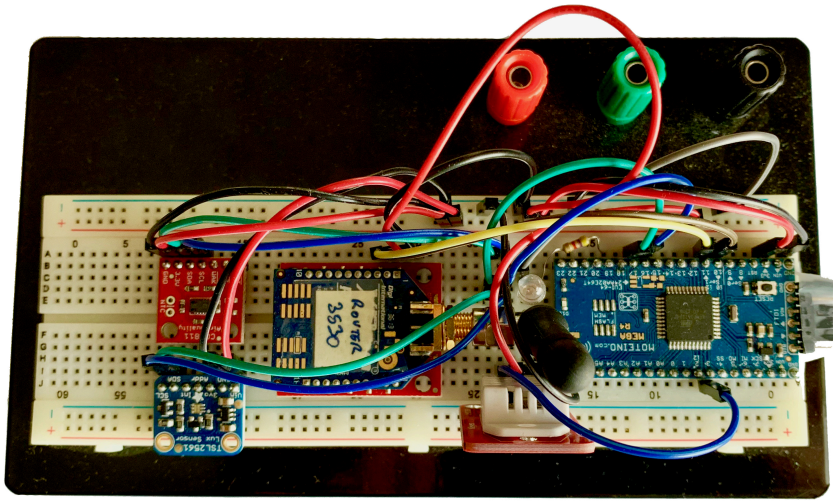


Figura 4.3: Prototipo integrado.

En las figuras 4.2 y 4.3 podemos apreciar los prototipos utilizados para este proyecto. Están constituidos por un microcontrolador Moteino Mega, varios sensores (luminosidad, temperatura, humedad relativa y calidad del aire) responsables de recibir información del entorno y dos módulos de comunicación ZigBee Xbee Pro S2B. En la figura 4.4 vemos el cableado de la placa protoboard del prototipo.

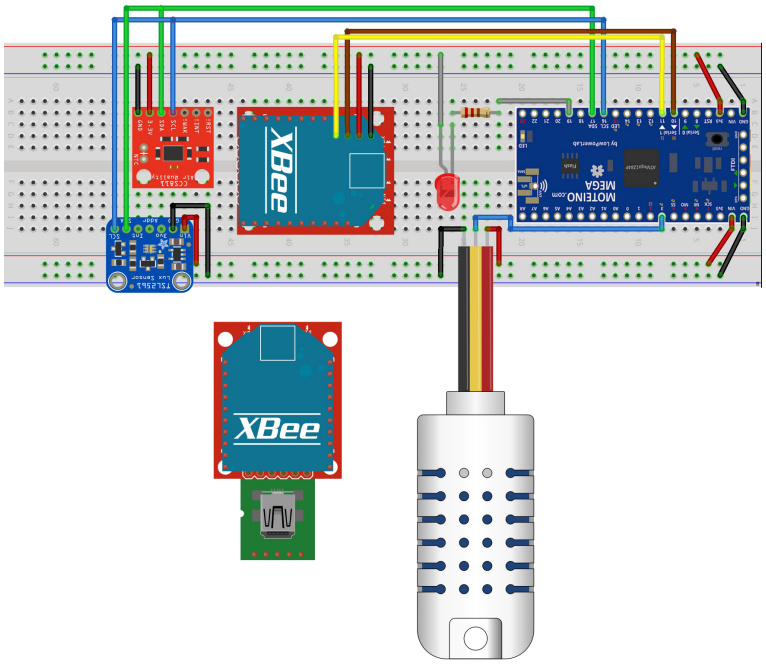


Figura 4.4: Cableado prototipo del nodo utilizado.

A continuación se describe brevemente cada elemento que conforma el prototipo.

Moteino MEGA

Moteino MEGA [11] es un dispositivo inalámbrico (Figura 4.5) compatible con Arduino basado en el microcontrolador Atmel ATMega1284P. Destaca su mínimo consumo en comparación con los demás modelos. Sus principales características son:

- 128 KB de memoria flash
- 16 KB de memoria RAM
- Salida de 3.3V
- 2 puertos UART (Universal Asynchronous Receiver-Transmitter)
- 1 puerto SPI (Serial Peripheral Interface)
- 1 bus de comunicaciones I2C
- Conexión FTDI (Future Technology Devices International) con la cual nos conectamos al ordenador para su programación

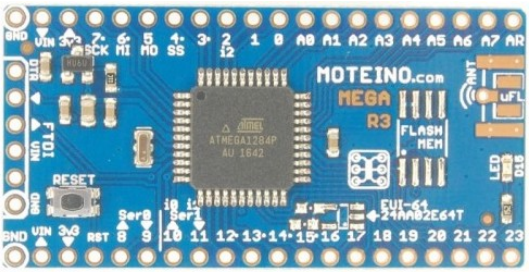


Figura 4.5: Moteino Mega.

Xbee Pro S2B

El módulo Zigbee Xbee Pro S2B [12] (Figura 4.6) está fabricado por DIGI. Se trata de un módulo programable (FDD) que permite crear redes de malla complejas basadas en el firmware de malla XBee ZB ZigBee. Sus principales características son:

- Compatible con otros módulos preconfigurados de la compañía
- Voltaje entre 2.7 y 3.6 V
- Sensibilidad de recepción de -101 dBm
- Rango de comunicaciones de 90 metros aproximadamente
- 1 puerto SPI
- 1 puerto UART, el cual utilizaremos para los módulos de comunicación inalámbrica

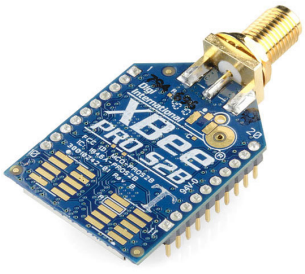


Figura 4.6: Módulo XBee Pro S2B.

CCS811 - Sensor de calidad del aire

Se trata de un sensor de gas digital [13] que detecta una amplia gama de Compuestos Orgánicos Volátiles Totales (TVOC)[1], y nos suministra su análogo de dióxido de carbono.

Este sensor (Figura 4.7) está destinado a la monitorización de la calidad del aire interior en dispositivos personales como relojes y teléfonos, pero se ha colocado en una placa protoboard para que pueda usarse como un dispositivo I2C normal.

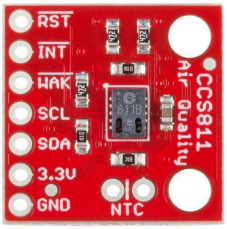


Figura 4.7: Sensor de calidad de aire CCS811.

TSL2561 Sensor digital de luminosidad

Se trata de un sensor de luminosidad [14] digital de gran precisión que permite detectar rangos de luz de 0.1 a 40,000 Lux[2]. Asimismo puede medir por separado la luz infrarroja, el espectro completo o la luz visible por el humano.

El sensor (Figura 4.8) tiene una interfaz digital (I2C), la cual utilizaremos para conectarlo al nuestro sistema. Se necesita una alimentación entre 2.7 y 3.6V y el consumo es extremadamente bajo, aproximadamente 0,5 mA cuando se detecta activo, y menos de 15 uA cuando está apagado.

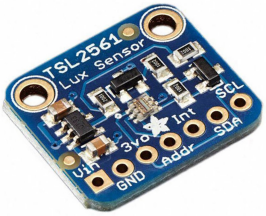


Figura 4.8: Sensor de luminosidad TSL2561.

[1] Sustancias químicas orgánicas que incluyen carbono y se transforman en vapor o gas.
[2] Unidad del Sistema Internacional de Unidades para la iluminación.

AM2302/DHT22 Sensor digital de temperatura y humedad relativa

Se trata de un sensor de temperatura y humedad [15] digital. Incluye un elemento de sensor de humedad capacitivo y dispositivos de medición de temperatura de alta precisión, con un microcontrolador de alto rendimiento de 8 bits conectado.

El sensor (Figura 4.9) tiene una distancia de transmisión de señal hasta 20 metros. Se necesita una alimentación de 3.3V y el rango que detecta es de -40 a 80^{o}C en temperatura y de 20-90 % RH (Humedad relativa)

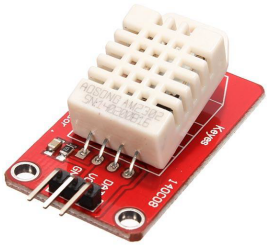


Figura 4.9: Sensor de humedad relativa y temperatura DHT22.

4.3. SmartGateway

El SmartGateway es el dispositivo más potente y con mayor capacidad de procesamiento de la arquitectura hasta la capa de edge. Su función es permitir a los dispositivos que utilizan una arquitectura REST, la interacción con los nodos a través de una implementación de una API REST.

Dicho elemento se podrá desplegar en cualquier dispositivo con las capacidades necesarias para correr sus componentes, por ejemplo una Raspberry PI, dado que es un dispositivo embebido y de bajo coste que puede albergar dichos componentes, además de ser un dispositivo representativo para el edge, que es donde está enmarcado este proyecto.

En la figura 4.10 vemos la visión de la infraestructura de la SmartGateway.

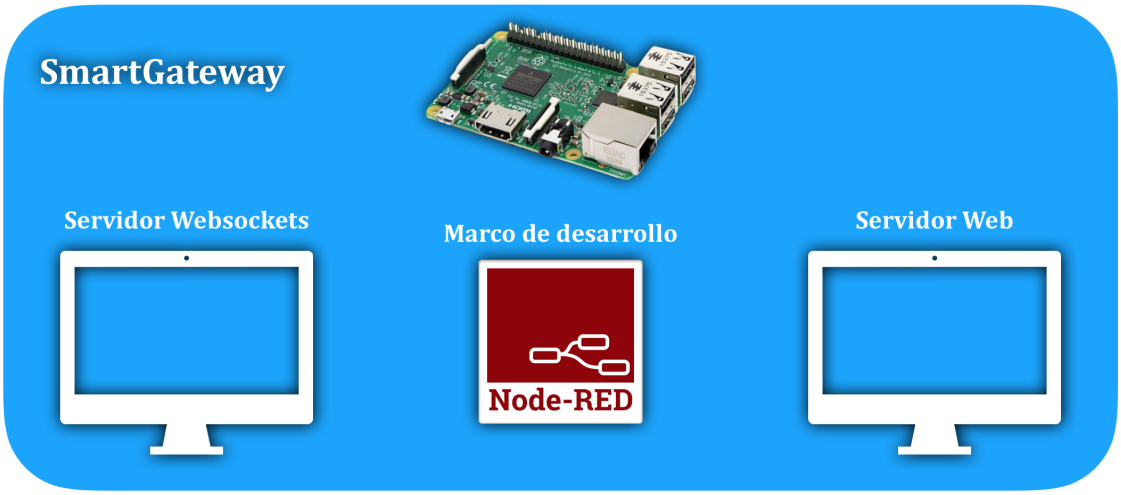


Figura 4.10: SmartGateway.

Capítulo 5

Análisis del sistema

El análisis de un proyecto es una fase esencial para una adecuada elaboración del mismo. En esta etapa se establecen las bases del proyecto, las características fundamentales y se identifican los posibles usuarios que utilizarán la aplicación desarrollada. Un buen análisis ayuda a desarrollar correctamente el proyecto y delimitar los posibles riesgos que puedan surgir durante ese desarrollo.

A través del análisis realizado se ha procurado encontrar el conjunto de requisitos y funcionalidades que constituyen el sistema. Para ello, al empezar el proyecto se reunieron las partes afectadas en el proyecto y se acordaron el conjunto de funcionalidades requeridas.

En este capítulo se exponen los requisitos funcionales y no funcionales que forman el sistema y una descripción global de este.

5.1. Descripción del sistema

El sistema se constituye de una serie de componentes y herramientas cuyo objetivo principal es proveer al usuario un marco de desarrollo visual que permita la programación de aplicaciones IoT sobre Edge Computing.

Para dar apoyo a todas las funcionalidades, el sistema cuenta con varios componentes que se detallan a continuación:

- ***Marco de desarrollo visual (Node-RED)***. Es el componente principal de este proyecto a través del cual el usuario puede programar aplicaciones abstrayéndose de las subscripciones a los dispositivos IoT.
- ***Servidor web asíncrono***. Este servidor web será el encargado de permitir la interconexión asíncrona de IoT con el marco de desarrollo.
- ***Aplicativo web de gestión de recursos***. El aplicativo web dará soporte a los usuarios mediante un mecanismo con el que el sistema podrá consultar y gestionar el listado de recursos que lo componen.

5.2. Requisitos funcionales

Los requisitos funcionales exponen los servicios que facilitará el sistema, es decir, la forma en que reaccionará a determinadas acciones.

A través de ellos se consigue el comportamiento del sistema, así como la identificación de los diferentes entes que interactúan con este.

A continuación se detallan los requisitos funcionales del sistema:

- ***RF.1***. Los usuarios podrán desarrollar la lógica de la aplicación edge a través de una interfaz web.
- ***RF.2***. Los usuarios podrán conectar sus aplicaciones desarrolladas con el cloud a través de Kafka.
- ***RF.3***. Los usuarios podrán condicionar la información recibida de los recursos de los servidores CoAP de la infraestructura subyacente del sistema en la aplicación edge a través de una interfaz web.
- ***RF.4***. Los usuarios podrán filtrar la información asociada a los recursos de los servidores CoAP de la infraestructura subyacente del sistema en la aplicación edge a través de una interfaz web.
- ***RF.5***. Los usuarios podrán obtener todos los recursos de los servidores CoAP recibidos en el sistema.

- ***RF.6***. Los usuarios podrán observar los recursos de los servidores CoAP recibidos en el sistema y recibir una actualización asíncrona de sus estado.
- ***RF.7***. Los usuarios podrán actuar sobre los recursos de los servidores CoAP recibidos en el sistema.

5.3. Requisitos no funcionales

Los requisitos no funcionales no se refieren directamente a las funciones especificas suministradas por el sistema, sino a una propiedad incluida de una funcionalidad del sistema.

Estos requisitos manifiestan aspectos como rendimiento, seguridad, estabilidad, disponibilidad o aspectos de la interfaz.

A continuación se detallan los requisitos no funcionales del sistema:

- ***RNF.1***. El servidor websockets permanecerá siempre activo esperando a nuevos clientes.
- ***RNF.2***. El servidor Node-RED permanecerá siempre activo esperando a nuevas comunicaciones.
- ***RNF.3***. El entorno de programación deberá ser accesible y amigable para los usuarios finales.
- ***RNF.4***. La aplicación web resultante será fácilmente ampliable para la incorporación de nuevos componentes.
- ***RNF.5***. El entorno de programación podrá ser desplegado en un dispositivo de capacidades limitadas como una Raspberry Pi.

Capítulo 6

Especificación

Para especificar correctamente un proyecto se debe de realizar una descripción completa de las características técnicas que debe verificar el sistema. Dentro de la especificación se detalla el camino a realizar para que se cumpla la funcionalidad.

A través del modelo de dominio se especifica el conjunto de datos y relaciones que forman el sistema ilustrándolos gráficamente. Además, con los casos de uso se describen las actividades que se deben realizar para llevar a cabo algún proceso y la forma de interactuar que tienen los usuarios para interactuar con las diferentes funcionalidades.

Este capítulo incluye un modelo de dominio, diagramas y descripción de los casos de uso con los que se expone la especificación del sistema a partir de los requisitos mencionados en el Capítulo 5.

6.1. Modelo de dominio

Desde los requisitos citados en el Capítulo 5, se elabora el modelo de dominio de la figura 6.1, estableciendo los datos y la relación entre ellos.

A continuación, se detalla la finalidad de cada elemento que constituye el modelo de dominio.

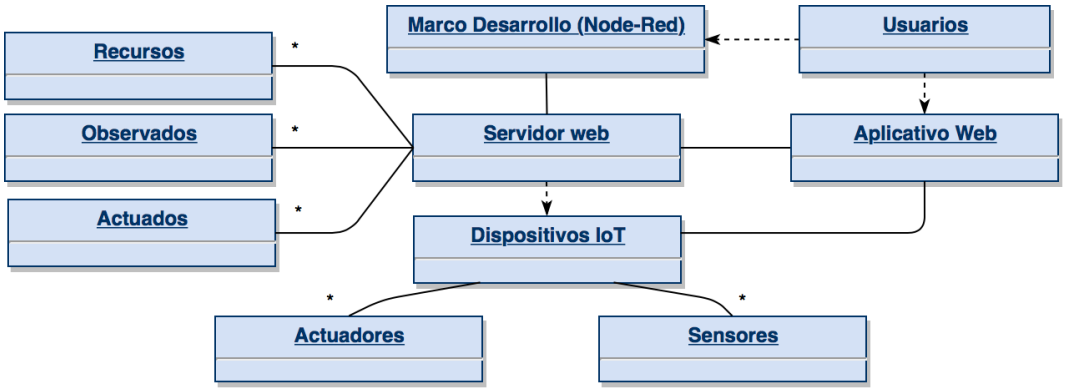


Figura 6.1: Modelo de dominio.

- ***Marco desarrollo (Node-RED)***. Representa la herramienta por la cual los usuarios pueden desarrollar sus aplicaciones.
- ***Usuarios***. Representa el grupo de usuarios que pueden usar el marco de desarrollo y acceder al aplicativo web.
- ***Servidor web***. Representa el servidor websocket asíncrono con el que se realizará la interconexión de IoT con el marco de desarrollo.
- ***Aplicativo web***. Representa la aplicación web que da soporte a los usuarios para poder consultar y gestionar el listado de recursos que componen el dispositivo IoT.
- ***Recursos***. Representa el conjunto de los distintos recursos disponibles en el dispositivo IoT.
- ***Observados***. Representa el conjunto de recursos a los que se está subscrito para obtener periódicamente datos.
- ***Actuados***. Representa el conjunto de recursos a los que se está subscrito para poder actuar sobre ellos.
- ***Dispositivos IoT***. Representa el dispositivo en el que están conectados los distintos componentes IoT.
- ***Actuadores***. Representa el conjunto de actuadores, como un LED o un altavoz, disponibles en el dispositivo IoT.
- ***Sensores***. Representa el conjunto de sensores, como un sensor de temperatura o humedad, disponibles en el dispositivo IoT.

6.2. Diagrama de casos de uso

En este apartado se expone a través de la figura 6.2, el diagrama de casos de uso que puede tener un usuario en el sistema.

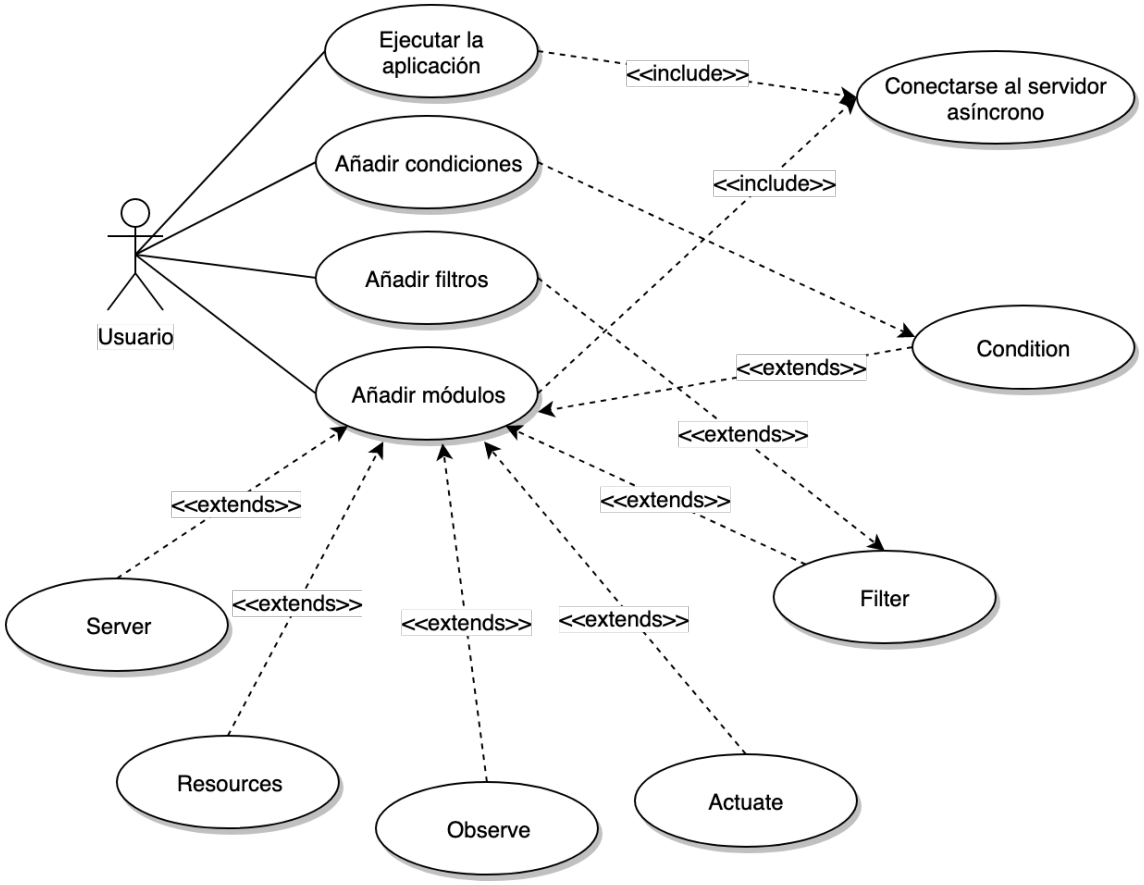


Figura 6.2: Caso de uso P.1.

6.3. Descripción de los casos de uso

En este apartado se detallan los casos de uso más significantes del sistema. Se especifica la interacción que hay entre los actores y el sistema y el comportamiento que se debe llevar a cabo en cada caso

CU.1	**Los usuarios podrán desarrollar la lógica de la aplicación edge a través de una interfaz web**
Contexto de uso	Cuando un usuario lo necesite, puede desarrollar la lógica de la aplicación edge a través de una interfaz web.
Ámbito	Marco de desarrollo visual.
Nivel	Usuario.
Actor principal	Cualquier usuario.

Participantes y objetivos

- Programador. Acceder al marco de desarrollo para programar la lógica de la aplicación edge a través de una interfaz web.
- Usuario. Acceder al marco de desarrollo para ejecutar la aplicación edge desarrollada.

Precondiciones	Estar arrancado el servidor Node-RED.
Garantías de éxito	El usuario accede al marco de desarrollo para poder empezar a desarrollar la lógica de la aplicación edge a través de una interfaz web.
Disparo	El usuario desarrolla la lógica de la aplicación edge que desee.

Escenario de éxito principal

1. El usuario accede al marco de desarrollo.
2. El usuario desarrolla la aplicación edge que desee.
3. El usuario ejecuta la aplicación desarrollada.
4. La aplicación se ejecuta correctamente.

Cuadro 6.1: Descripción Caso de Uso 1.

CU.2	**Los usuarios pueden conectar sus aplicaciones desarrolladas con el cloud a través de Kafka en la aplicación edge a través de una interfaz web**
Contexto de uso	Cuando un usuario lo necesite, puede conectar su aplicación desarrollada con el cloud a través de Kafka.
Ámbito	Marco de desarrollo visual.
Nivel	Usuario.
Actor principal	Cualquier usuario.

Participantes y objetivos

- Programador. Acceder al marco de desarrollo para conectar su aplicación desarrollada con el cloud a través de Kafka.

Precondiciones

1. Estar arrancado el servidor Node-RED.
2. Estar arrancado y configurado el servicio Apache Kafka.

Garantías de éxito	El usuario accede al marco de desarrollo para conectar su aplicación desarrollada con el cloud a través de Kafka.
Disparo	El usuario conecta su aplicación desarrollada con el cloud a través de Kafka.

Escenario de éxito principal

1. El usuario accede al marco de desarrollo.
2. El usuario conecta la aplicación desarrollada con el cloud a través de Kafka.
3. El usuario ejecuta la aplicación desarrollada.
4. Los datos son enviados al cloud.

Cuadro 6.2: Descripción Caso de Uso 2.

CU.3	**Los usuarios pueden añadir condiciones en la aplicación edge a través de una interfaz web**
Contexto de uso	Cuando un usuario lo necesite, puede añadir condiciones en la aplicación edge a través de una interfaz web.
Ámbito	Marco de desarrollo visual.
Nivel	Usuario.
Actor principal	Cualquier usuario.

Participantes y objetivos

- Programador. Acceder al marco de desarrollo para añadir una condición que afecte a los datos en la aplicación edge.

Precondiciones	Estar arrancado el servidor Node-RED.
Garantías de éxito	El usuario accede al marco de desarrollo y añade condiciones a la aplicación.
Disparo	El usuario añade las condiciones que desee a la aplicación.

Escenario de éxito principal

1. El usuario accede al marco de desarrollo.
2. El usuario añade la condición que desee a la aplicación.

Cuadro 6.3: Descripción Caso de Uso 3.

CU.4	**Los usuarios pueden añadir filtros en la aplicación edge a través de una interfaz web**
Contexto de uso	Cuando un usuario lo necesite, puede añadir filtros en la aplicación edge a través de una interfaz web.
Ámbito	Marco de desarrollo visual.
Nivel	Usuario.
Actor principal	Cualquier usuario.

Participantes y objetivos

- Programador. Acceder al marco de desarrollo para añadir un filtro que afecte a los datos en la aplicación edge.

Precondiciones	Estar arrancado el servidor Node-RED.
Garantías de éxito	El usuario accede al marco de desarrollo y añade filtros a la aplicación.
Disparo	El usuario añade los filtros que desee a la aplicación.

Escenario de éxito principal

1. El usuario accede al marco de desarrollo.
2. El usuario añade el filtro que desee a la aplicación.

Cuadro 6.4: Descripción Caso de Uso 4.

CU.5	**Los usuarios pueden añadir módulos para obtener todos los recursos de los servidores CoAP recibidos en el sistema en la aplicación edge a través de una interfaz web**
Contexto de uso	Cuando un usuario lo necesite, puede añadir módulos para obtener todos los recursos en la aplicación edge.
Ámbito	Marco de desarrollo visual.
Nivel	Usuario.
Actor principal	Cualquier usuario.

Participantes y objetivos

- Programador. Acceder al marco de desarrollo para añadir módulos para obtener todos los recursos en la aplicación edge.

Precondiciones	Estar arrancado el servidor Node-RED.
Garantías de éxito	El usuario accede al marco de desarrollo para añadir módulos para obtener todos los recursos.
Disparo	El usuario añade los módulos que desee para obtener todos los recursos.

Escenario de éxito principal

1. El usuario accede al marco de desarrollo.
2. El usuario añade el módulo que desee para obtener todos los recursos a la aplicación.

Cuadro 6.5: Descripción Caso de Uso 5.

CU.6	**Los usuarios pueden añadir módulos para observar los recursos de los servidores CoAP recibidos en el sistema y recibir una actualización asíncrona de sus estado en la aplicación edge a través de una interfaz web**
Contexto de uso	Cuando un usuario lo necesite, puede añadir módulos para observar los recursos de los servidores CoAP en la aplicación edge.
Ámbito	Marco de desarrollo visual.
Nivel	Usuario.
Actor principal	Cualquier usuario.

Participantes y objetivos

- Programador. Acceder al marco de desarrollo para para observar los recursos de los servidores CoAP en la aplicación edge.

Precondiciones	Estar arrancado el servidor Node-RED.
Garantías de éxito	El usuario accede al marco de desarrollo para observar los recursos de los servidores CoAP.
Disparo	El usuario añade los módulos que desee para observar los recursos de los servidores CoAP.

Escenario de éxito principal

1. El usuario accede al marco de desarrollo.
2. El usuario añade el módulo que desee para observar los recursos de los servidores CoAP y recibe una actualización asíncrona de sus estados.

Cuadro 6.6: Descripción Caso de Uso 6.

CU.7	**Los usuarios pueden añadir módulos para actuar sobre los recursos de los servidores CoAP recibidos en el sistema en la aplicación edge a través de una interfaz web**
Contexto de uso	Cuando un usuario lo necesite, puede añadir módulos para actuar sobre los recursos de los servidores CoAP en la aplicación edge.
Ámbito	Marco de desarrollo visual.
Nivel	Usuario.
Actor principal	Cualquier usuario.

Participantes y objetivos

- Programador. Acceder al marco de desarrollo para para actuar sobre los recursos de los servidores CoAP en la aplicación edge.

Precondiciones	Estar arrancado el servidor Node-RED.
Garantías de éxito	El usuario accede al marco de desarrollo para actuar sobre los recursos de los servidores CoAP.
Disparo	El usuario añade los módulos que desee para actuar sobre los recursos de los servidores CoAP.

Escenario de éxito principal

1. El usuario accede al marco de desarrollo.
2. El usuario añade el módulo que desee para actuar sobre los recursos de los servidores CoAP.

Cuadro 6.7: Descripción Caso de Uso 7.

Capítulo 7

Diseño

El diseño es uno de los componentes más importantes en la elaboración del TFG. En el diseño se determina la arquitectura del sistema, se organizan y diseñan los componentes y se constituye la forma de intercomunicación entre estos

En este capítulo se hablará de la arquitectura de diseño tanto del servidor Websocket como del servidor Node-RED. Además se mostrará el diagrama de clases del servidor Websocket y el formato de los datos para la comunicación entre los diferentes componentes que forman el sistema.

7.1. Arquitectura Node-RED

En esta sección se detallarán los distintos módulos que componen el servidor web, indicando la función que tiene cada uno, así como los datos que reciben y envían dentro del flujo de datos.

7.1.1. Módulo Websocket In

La figura 7.1 muestra el módulo WebSocket In. Este componente de Node-RED es el encargado de recibir los datos en formato JSON del servidor Websocket y enviarlos al resto del flujo.

Figura 7.1: Módulo Websocket In.

7.1.2. Módulo Websocket Out

La figura 7.2 muestra el módulo WebSocket Out. Este componente de Node-RED tiene la función de enviar de nuevo los datos en formato JSON al servidor Websocket después de haber pasado por todo el flujo.

Si el mensaje que llega a este nodo comenzó en un nodo de WebSocket In, el mensaje se enviará nuevamente al cliente que desencadenó el flujo. De lo contrario, el mensaje se transmitirá a todos los clientes conectados.

Figura 7.2: Módulo Websocket Out.

7.1.3. Módulo Resources

La figura 7.3 muestra el módulo Resources. Este componente de Node-RED recibe el flujo de datos del módulo Websocket In. Manda un mensaje al modulo Websocket Out con el contenido "Consola 1", indicando que quiere obtener una lista con todos los recursos disponibles.

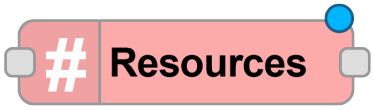


Figura 7.3: Módulo Resources.

```
{ 'op': 'Resources' }
```

Consola 1: Resources.

7.1.4. Módulo Observe

La figura 7.4 muestra el módulo Observe. Este componente de Node-RED recibe el flujo de datos del módulo Resources o del módulo Filter. Procesa los datos y por cada recurso recibido manda un mensaje al modulo Websocket Out con el contenido "Consola 2", cuyos campos son:

- resource: nombre del recurso.
- end_point: nombre del end_point.
- max_age: intervalo de tiempo en segundos de cada actualización de datos del recurso.

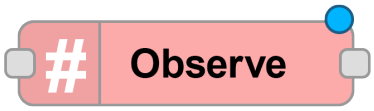


Figura 7.4: Módulo Observe.

```
{ 'op': "Observe", 'resource': resource_name,
'end_point': end_point_name, 'max_age': max_age }
```

Consola 2: JSON Observe Data.

7.1.5. Módulo Actuate

La figura 7.5 muestra el módulo Actuate. Este componente de Node-RED recibe el flujo de datos del módulo Resources o Filter y del módulo Condition. Procesa los datos y por cada recurso recibido manda un mensaje al módulo Websocket Out, siempre y cuando se cumplan las condiciones de los valores recibidos en el módulo Condition, con el contenido "Consola 3", cuyos campos son:

- resource: nombre del recurso.
- end_point: nombre del end_point.
- value: valor para actuar sobre el recurso.

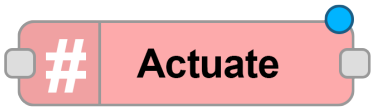


Figura 7.5: Módulo Actuate.

```
{ 'op': "Actuate", 'resource': resource_name,
'end_point': end_point_name, 'value': value }
```

Consola 3: JSON Actuate Data.

7.1.6. Módulo Filter

La figura 7.6 muestra el módulo Filter. Este componente de Node-RED tiene la función de filtrar por el nombre que se le indique los recursos que le llegan. Procesa los datos y manda un mensaje con el recurso filtrado con el contenido "Consola 4". Se puede añadir más de un filtro separado por comas.

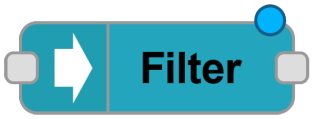


Figura 7.6: Módulo Filter.

```
{ 'op': "resources", 'resource': resource_name }
```

Consola 4: JSON Filter Data.

7.1.7. Módulo Condition

La figura 7.7 muestra el módulo Condition. Este componente de Node-RED tiene la función de establecer una condición para que en caso de que los valores recibidos de los recursos las cumplan, sigan el flujo de datos. En caso de cumplir dicha condición se manda un mensaje con el contenido "Consola 5".

Figura 7.7: Módulo Condition.

```
{ 'op': "data", 'resource': resource_name, "end_point": end_point_name,
'value': value}
```

Consola 5: JSON Filter Data.

7.1.8. Módulo Producer Kafka

La figura 7.8 muestra el módulo Producer Kafka. Este componente de Node-RED tiene la función de ejercer como productor de mensajes Kafka. Este módulo no ha sido creado para este proyecto sino que ha sido instalado del repositorio de Node-RED ("node-red-contrib-kafka-node") [16].

Figura 7.8: Módulo Producer Kafka.

7.1.9. Módulo Consumer Kafka

La figura 7.9 muestra el módulo Consumer Kafka. Este componente de Node-RED tiene la función de ejercer como consumidor de mensajes Kafka. Este módulo no ha sido creado para este proyecto sino que ha sido instalado del repositorio de Node-RED ("node-red-contrib-kafka-node").

Figura 7.9: Módulo Consumer Kafka.

7.2. Arquitectura Websocket

En esta sección se detallarán las distintas clases programadas en lenguaje Python que componen el asíncrono Websockets, indicando la función que tiene cada una.

La figura 7.10 muestra el diagrama de clases del servidor Websockets en el que podemos apreciar 5 clases.

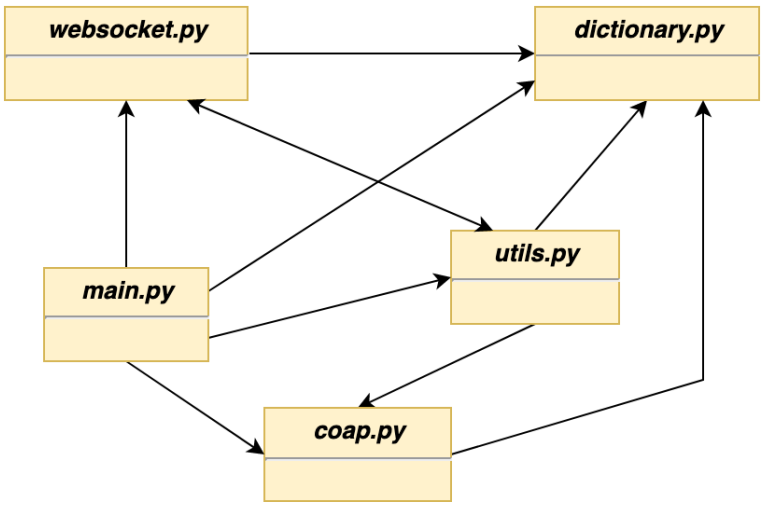


Figura 7.10: Diagrama de clases del servidor Websocket.

A continuación, se describen los componentes incluidos en el diagrama de clases.

7.2.1. Main

Es la clase principal del proyecto. Está diseñada por Cristian Martín Fernández y a la, que para este TFG, se le ha añadido la instrucción para iniciar el servidor Websocket. Además inicializa el servidor HTTP, incluyendo sus operaciones, y la comunicación de CoAP con Zigbee.

7.2.2. Dictionary

Esta clase gestiona el diccionario donde se almacenarán el identificador de los servidores CoAP junto con las direcciones y recursos de los servidores. Es utilizada por todas las demás clases.

7.2.3. Coap

Se trata de una clase que provee una interfaz con CoAP, gestiona la recepción y el envío de paquetes CoAP.

7.2.4. Utils

Sirve de enlace entre la clase Main y Websocket. A partir de ella se pueden crear observaciones, recibir datos de las observaciones e interactuar con recursos en la clase Websocket.

7.2.5. Websocket

Es la clase principal de este TFG, en ella se encuentran las operaciones principales del servidor websockets como son obtener recursos, realizar una observación de un recurso y actuar sobre un recurso.

Capítulo 8

Implementación

La implementación de un proyecto consta de la realización y modelado del sistema a partir del análisis, la especificación y el diseño descritos en los capítulos anteriores.

En la implementación se ha mencionado cada clase que compone el sistema con un nombre representativo que describa su función u objetivo. La programación se ha desarrollado de forma modular, fraccionando los algoritmos en módulos para posibilitar la reutilización de código y hacer el trabajo mas adaptable. El código se ha documentando para que se pueda entender el funcionamiento de cada función.

En este capítulo se explica la implementación llevada a cabo en el sistema. También se describen la implementación de cada componente que forma el sistema.

8.1. Servidor asíncrono Websocket

A continuación, se describen los componentes incluidos en el diagrama de clases.

8.1.1. Dictionary

En la figura 8.1 vemos un diagrama UML de la clase Dictionary en la que hay una variable “dictionary” donde almacenaremos el diccionario.

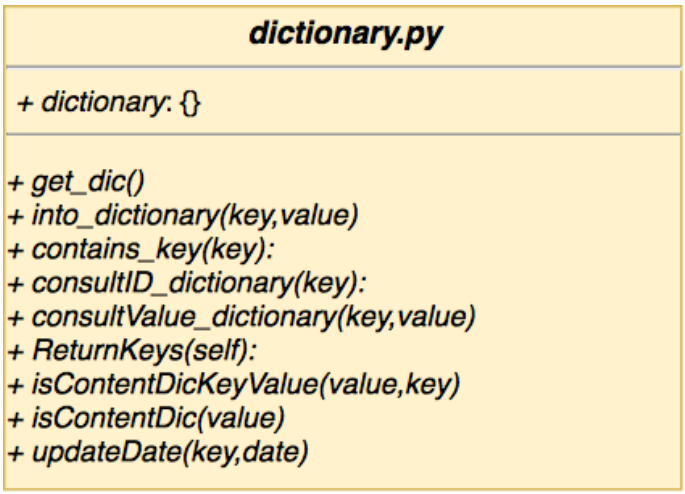


Figura 8.1: Clase Dictionary.

Los métodos de esta clase son:

- ***get_dic()***. Devuelve el diccionario.
- ***into_dictionary(key,value)***. Introduce en el diccionario la tupla (key, value).
- ***contains_key(key)***. Comprueba si una clave existe en el diccionario.
- ***consultID_dictionary(key)***. Devuelve el valor almacenado en el diccionario asociado a key.
- ***consultValue_dictionary(key,value)***. Devuelve el valor almacenado en el diccionario asociado a las claves key, value.
- ***ReturnKeys()***. Devuelve las claves de primer nivel del diccionario.
- ***isContentDicKeyValue(value,key)***. Comprueba si key esta contendido en el diccionario de segundo nivel.
- ***isContentDic(value)***. Comprueba si value esta contenido en el diccionario.
- ***updateDate(key,date)***. Actualiza el tiempo asociado a un identificador almacenado en el diccionario.

8.1.2. Utils

En la figura 8.2 vemos un diagrama UML de la clase Utils en la que hay una serie de variables utilizadas para ayudarnos en la programación:

- ***dictionary***. Instancia de la clase Dictionary.
- ***mutex***. Semáforo utilizado para esperar las respuestas de los servidores CoAP en el proxy HTTP.
- ***coap***. Instancia de la clase CoAP para enviar y recibir paquetes CoAP.
- ***ws***. Instancia de la clase Websocket.

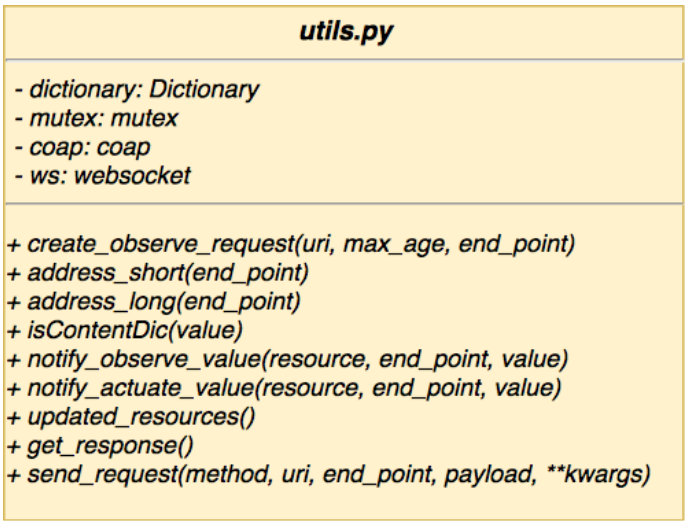


Figura 8.2: Clase Utils.

Los métodos de esta clase son:

- ***create_observe_request(uri, max_age, end_point)***. Crea una una petición observe.
- ***address_short(end_point)***. Obtiene la dirección corta física del dispositivo a partir de su identificador.
- ***address_long(end_point)***. Obtiene la dirección larga física del dispositivo a partir de su identificador.

- ***isContentDic(value)***. Comprueba si el value esta contenido en el diccionario.
- ***notify_observe_value(resource, end_point, value)***. Recibe una llamada cuando obtiene una nueva observación.
- ***notify_actuate_value(resource, end_point, value)***. Recibe una llamada cuando obtiene una nueva actuación.
- ***updated_resources()***. Recibe una llamada cuando se actualizan los recursos.
- ***get_response()***. Método para obtener una respuesta de una petición GET de CoAP.
- ***send_request(method, uri, end_point, payload, **kwargs)***. Envía un paquete CoAP.

8.1.3. Main

En la figura 8.3 vemos un diagrama UML de la clase Main en la que hay una serie de variables utilizadas para ayudarnos en la programación:

- ***libraries***. Diccionario utilizado para guardar los tipos de sensores y actuadores.
- ***dictionary***. Diccionario que guarda la información de los servidores CoAP y sus recursos.
- ***mutex***. Semáforo utilizado para esperar las respuestas de los servidores CoAP en el proxy HTTP.
- ***coap***. Instancia de la clase CoAP para enviar y recibir paquetes CoAP.
- ***ws***. Instancia de la clase Websocket.
- ***utils***. Instancia de la clase Utils.

Figura 8.3: Clase Main.

También tenemos varios métodos con los que se realizan tareas específicas:

- ***general(request, method)***. Construye un paquete CoAP y lo envía al Servidor CoAP recibido en la URL.
- ***coap_update(request)***. Petición PUT para actualizar un recurso.
- ***coap_get(request)***. Realiza una petición GET al Servidor CoAP.
- ***coap_post(request)***. Realiza una petición POST al Servidor CoAP.
- ***coap_delete(request)***. Realiza una petición DELETE al Servidor CoAP.
- ***coap_put(request)***. Realiza una petición PUT al Servidor CoAP.
- ***resource_middleware(request)***. Redirigimos al tipo de petición que corresponde.

- ***coap_nodes(request)***. Obtiene el listado de servidores CoAP al sistema y lo devuelve en una respuesta HTTP.
- ***disable_observe(request)***. Petición que contiene una petición para deshabilitar la observación de un recurso.
- ***observe_request(request)***. Petición que contiene una petición de observe a un recurso.
- ***libraries_request(request)***. Petición para obtener las librerías disponibles de un servidor.
- ***coap_resource(request)***. Obtiene los recursos de un dispositivo y los devuelve en una respuesta HTTP.
- ***observe_middleware(request)***. Controla todas las peticiones observe a un recurso.
- ***get_coap_id(request, observe)***. Obtiene el identificador del Middleware CoAP a partir de la URL.
- ***get_coap_uri(request,observe)***. Obtiene el nombre del recurso al que se quiere realizar una petición a partir de la URL.
- ***GetUriGet(request,observe)***. Obtiene el nombre del recurso sobre el que se realiza la peticion GET.
- ***GetUriPost(request)***. Obtiene el nombre del recurso sobre el que se realiza la petición POST.
- ***Payload(request)***. Obtiene el valor introducido por el Usuario en la petición POST.
- ***loop_in_thread(loop,ws)***. Función auxiliar para poder llamar a una función asíncrona desde una síncrona.

8.1.4. Coap

En la figura 8.4 vemos un diagrama UML de la clase Coap donde existen numerosos métodos en la que seguidamente explicaremos la función de cada uno de ellos.

Figura 8.4: Clase Coap.

Los métodos utilizados de esta clase son:

- ***set_zigbee(xbee)***. Establece el controlador ZigBee para las comunicaciones.
- ***set_utils(utils)***. Establece el la clase Utils.
- ***is_resource_observable(end_point, resource)***. Comprueba si el recurso tiene una observación para obtener el valor.

- ***get_value_observe(end_point, resource)***. Obtiene el estado de el ultimo observe enviado para ser utilizado en las peticiones GET.
- ***get_id_observe(uri, addr, addr_long)***. Obtiene el identificador de la relación observe.
- ***remove_id_observe(key)***. Elimina un observador previamente creado.
- ***add_observe(uri, addr, addr_long,end_point)***. Crea la nueva relación de observe establecida.
- ***getbuffer_data()***. Devuelve los datos del paquete CoAP recibido.
- ***new_token()***. Devuelve el valor de token para un nuevo paquete.
- ***__resource_size(index, payload, len)***. Devuelve la posición del ultimo carácter de un recurso.
- ***__contentDic(data, coap_id)***. Obtiene los nombres y tipos de lo recursos, y lo almacena en el diccionario cuando se recibe una respuesta de recursos.
- ***create_resource(uri_resource, parameters, type, method, id)***. Crea el recurso del CoM indicado.
- ***update_resource(uri_resource, new_uri, parameters, id)***. Actualiza el recurso del CoM indicado.
- ***remove_resource(uri_resource, id)***. Elimina el recurso del CoM indicado.
- ***getResourceToken()***. Obtiene el token de respuesta.
- ***getObserveToken()***. Obtiene el token de una respuesta observe.
- ***save_ep_id(ep_id, addr, addr_long, lt)***. Almacena los datos proporcionados en el diccionario.
- ***updated_ep_id(ep)***. Función que actualiza el tiempo de actualización de un ep.
- ***request_resources(coap_id, addr, addr_long)***. Realiza una petición al Servidor CoAP para obtener todos los recursos de un dispositivo.
- ***confirm_broadcast(addr, addr_long)***. Realiza la confirmación de broadcast inicial.
- ***check_resouces()***. Comprueba si se ha pasado el tiempo limite el estado de los CoM del sistema.

- ***getResource_response()***. Devuelve el valor de la variable que comprueba si se ha recibido respuesta del Servidor CoAP.
- ***getCode()***. Devuelve el valor del código de la respuesta.
- ***get_final_time()***. Devuelve el tiempo de la petición CoAP.
- ***setResource_response(value)***. Actualiza la variable que comprueba si se ha recibido respuesta Servidor CoAP.
- ***__sendToCoAP(addr_long, addr, bytes)***. Función que envía un paquete a zigbee con las direcciones proporcionadas.
- ***sendToCoAP(addr_long, addr, bytes)***. Función que crea una hebra para enviar un paquete a Zigbee.
- ***get_resource(id,resource_name)***. Devuelve el recurso de un nodo.
- ***close()***. Termina la conexión con kafka al terminar.
- ***__sendToServer(bytes, topic)***. Función que envía datos a Kafka.
- ***check_observe(resource, end_point)***. Comprueba que el recurso del end_point esta siendo observado.
- ***receiver(packet)***. Método principal de la clase CoAP, encargado de recibir paquetes de los Servidores CoAP.

8.1.5. Websocket

En la figura 8.5 vemos un diagrama UML de la clase Websocket en la que hay una serie de variables utilizadas para ayudarnos en la programación:

- ***i***. Variable numérica para contar los clientes conectados.
- ***clients***. Lista para almacenar los clientes conectados.
- ***dictionary***. Instancia de la clase Dictionary.

Figura 8.5: Clase Websocket.

Los métodos de esta clase son:

- ***addClient(websocket)***. Añade un cliente a la lista de clientes conectados.
- ***removeClient(id)***. Borra el cliente de la lista.
- ***set_utils(utils)***. Guarda una referencia de la clase Utils.
- ***getResources()***. Devuelve una lista con todos los recursos disponibles.
- ***updateResources()***. Notifica a los clientes que estén interesados en los recursos.

- ***observe(id, resource, end_point, max)***. Crea un observer llamando a la clase Utils y se guarda que el cliente "id" observa el recurso.
- ***newObserveData(resource, end_point, value)***. Obtiene el dato del recurso observado.
- ***send_message***. Método auxiliar para hacer una llamada asíncrona de un método desde una síncrona.
- ***actuate(resource, end_point, data_to_sent)***. Se envía el valor al recurso para que se actúe sobre el.
- ***actuateResponse(resource, boolean)***. Respuesta del actuate.
- ***runServer(self, websocket, path)***. Método principal, arranca el servidor.

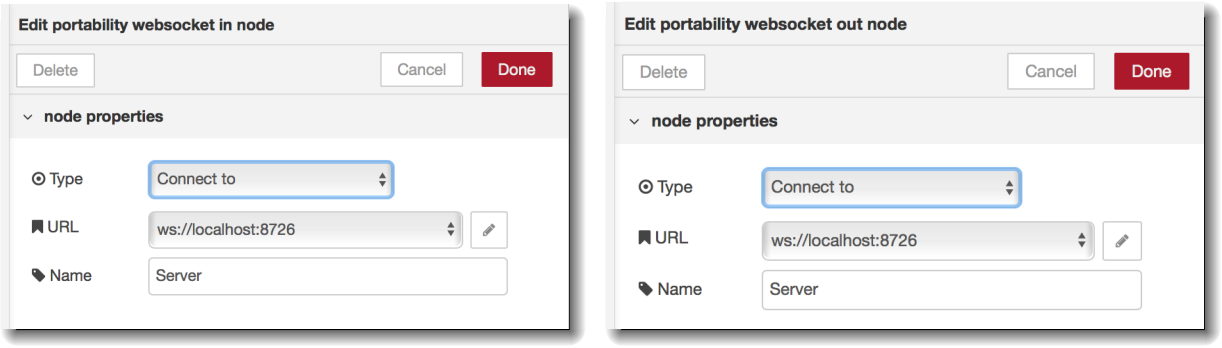
8.2. Node-RED

En esta sección se detallará la implementación de los los distintos módulos que componen el servidor web. Para la creación de un nodo en Node-RED hacen falta dos archivos; un archivo JavaScript que define lo que hace el nodo, y un archivo html que define las propiedades del nodo, el diálogo de edición y el texto de ayuda.

8.2.1. Módulo Websocket In/Out

Aunque existan dos módulos diferentes, Websocket In y Websocket Out, en realidad comparten el mismo archivo Html y Javascript. Vamos a describir brevemente su implementación.

Html



(a) Websocket In (b) Websocket Out

Figura 8.6: Html Websocket.

El archivo html, como podemos ver en las figuras 8.6a y 8.6b, está implementado para configurar tres partes:

- ***Type***. Identifica el tipo de conexión que se realiza. "Connect to" para realizar una conexión como cliente y "Listen on" para realizar una conexión como servidor. En nuestro caso utilizaremos el tipo "Connect to" para establecer una conexión cliente con el servidor asíncrono Websocket.

- ***URL***. Establece la URL de conexión con el servidor asíncrono Websocket.
- ***Name***. Establece un nombre específico que le queramos dar al componente.

Javascript

Dentro del archivo Javascript tenemos las funciones necesarias para gestionar la conexión Websocket con los datos introducidos a través de las propiedades del modulo en Node-RED. Las características básicas son:

En el módulo Websocket In, de forma predeterminada, los datos recibidos de WebSocket estarán en msg.payload. El socket puede configurarse para esperar una cadena JSON correctamente formada, en cuyo caso analizará el JSON y enviará el objeto resultante como el mensaje completo.

En el módulo Websocket Out, de forma predeterminada, msg.payload se enviará a través de WebSocket. El socket puede configurarse para codificar todo el objeto msg como una cadena JSON y enviarlo a través de WebSocket.

Si el mensaje que llega a este nodo comenzó en un nodo de WebSocket In, el mensaje se enviará de vuelta al cliente que desencadenó el flujo. De lo contrario, el mensaje se transmitirá a todos los clientes conectados.

Si se desea transmitir un mensaje que comenzó en un nodo de WebSocket In, se debe eliminar la propiedad msg._session dentro del flujo.

8.2.2. Módulo Resources

Html

El archivo html, como podemos ver en la figuras 8.7 está implementado para configurar sólo el nombre específico que le queramos dar al componente.

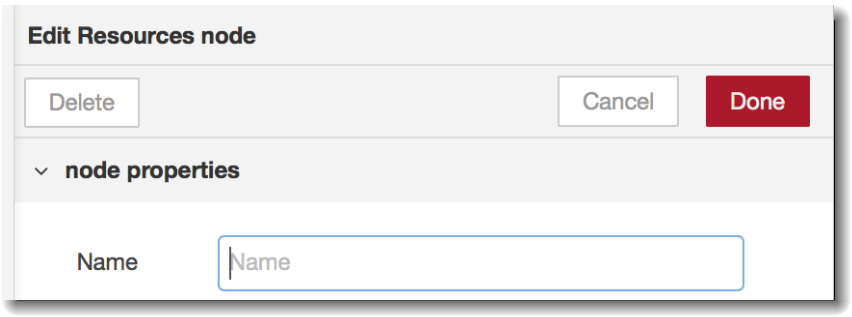


Figura 8.7: Html Resources.

Javascript

Dentro del archivo Javascript, la única función que existe como vemos en el siguiente código es recibir el flujo de datos del módulo "Websocket In" y si existe un cliente, manda un mensaje con la operación "Resources", indicando que quiere obtener una lista con todos los recursos disponibles.

```
function Resources(config) {
    RED.nodes.createNode(this, config);
    var node = this;
    this.on('input', function (msg) {
        if (msg.client != null) {
            msg.payload = { "op": "Resources" };
        }
        node.send(msg);
    });
}
```

Consola 6: resources.js

8.2.3. Módulo Observe

Html

El archivo html, como podemos ver en la figura 8.8, está implementado para configurar dos partes:

- ***Name***. Establece un nombre específico que le queramos dar al componente.

- ***Max_age***. Establece el intervalo de tiempo en segundos de cada actualización de datos del recurso.

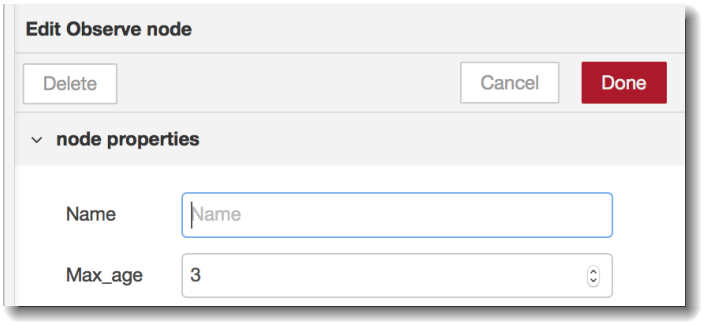


Figura 8.8: Html Observe.

Javascript

La única función existente en este archivo se encarga de mandar, por cada end-point que haya en el mensaje recibido, un mensaje con la operación "Observe", el recurso que se quiere observar, el end-point correspondiente al recurso y el intervalo de tiempo que que tendrá cada actualización de datos del recurso observado.

```
function Observe(config) {
    ...
    this.on('input', function (msg) {
        obj = msg.payload;
        if (obj['resources'] != undefined) {
            endPoints = obj['resources'];
            for (var x in endPoints) {
                endPoints[x].forEach(function (element) {
                    dataSend.payload = { 'op': "Observe",
                    'resource': element.resource, 'end_point': x,
                    'max_age': '' + max_age + '' }
                    node.send(dataSend);
                });
            }
        } else {
            node.send(msg);
        }
    });
}
```

Consola 7: observe.js

8.2.4. Módulo Actuate

Html

El archivo html, como podemos ver en la figura 8.9, está implementado para configurar dos partes:

- ***Name***. Establece un nombre específico que le queramos dar al componente.
- ***Value***. Establece el valor con el que se actuará en el recurso.

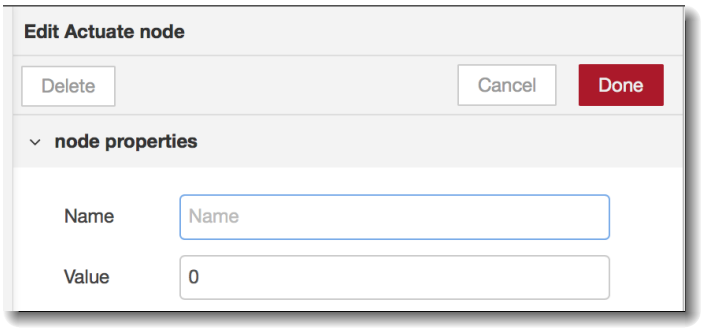


Figura 8.9: Html Actuate.

Javascript

En este archivo nos encontramos con 4 funciones. La función principal se encarga de mandar, por cada end-point que haya en el mensaje recibido, un mensaje con la operación "Actuate", el recurso que se quiere observar, el end-point correspondiente al recurso y el valor con el que se actuará en el recurso especifico.

Para que se llegue a mandar el mensaje, se debe cumplir una serie de condiciones: que el valor a enviar al recurso sea distinto al estado actual del recurso y que se cumplan las condiciones establecidas en el módulo Condition del cual llega el flujo de datos hasta este módulo Actuate. Para ello existen las siguientes funciones:

- ***getDataFiltered***. Esta función guarda en un array los recursos cuyos valores están siendo condicionados.
- ***getDataArrived***. Esta función guarda en un array los recursos que han cumplido las condiciones y por tanto han llegado al módulo Actuate.
- ***compareData***. Esta función comprueba que ambos arrays tengan los mismos elementos.

```
function Actuate(config) {
    ...
    this.on('input', function (msg) {
        obj = msg.payload;
        if (obj['resources'] != undefined) {
            endPoints = obj['resources'];
            for (var x in endPoints) {
                endPoints[x].forEach(function (element) {
                    dataSend.payload = { "op": "Actuate",
                    "resource": element.resource,
                    "end_point": x, "value": '' + value + '' }
                });
            }
        }
        getDataFiltered(flowContext, dataFiltered);
        if (obj['value'] != undefined) {
            getDataArrived(obj['value'], dataArrived);
            if (flowContext.get('state') != value &&
            compareData(dataFiltered, dataArrived)) {
                flowContext.set('state', value);
                dataFiltered = []; dataArrived = []; node.send(dataSend);
            }
        }
        ...
function getDataArrived(obj, dataArrived) {
    var aux = obj.split(/[ ,]+/);
    aux.forEach(function (element) {
        var aux2 = element.split(':');
        if (aux2[0] == 'T') {
            if (!dataArrived.includes('tempCondition')) {
                dataArrived.push('tempCondition');
            }
        } else if (aux2[0] == 'H') {
            if (!dataArrived.includes('humCondition')) {
                dataArrived.push('humCondition');
            }
        } else if (aux2[0] == 'CO2') {
            if (!dataArrived.includes('airCO2Condition')) {
                dataArrived.push('airCO2Condition');
            }
        ...
    });
}
```

Consola 8: actuate.js 1/2

```
function getDataFiltered(flowContext, dataFiltered) {
    if (flowContext.get('tempCondition') != undefined &&
    !dataFiltered.includes('tempCondition')) {
        dataFiltered.push('tempCondition');
    }
    if (flowContext.get('humCondition') != undefined &&
    !dataFiltered.includes('humCondition')) {
        dataFiltered.push('humCondition');
    }
    if (flowContext.get('airCO2Condition') != undefined &&
    !dataFiltered.includes('airCO2Condition')) {
        dataFiltered.push('airCO2Condition');
    }
    if (flowContext.get('airVOCCondition') != undefined &&
    !dataFiltered.includes('airVOCCondition')) {
        dataFiltered.push('airVOCCondition');
    }
    if (flowContext.get('luxCondition') != undefined &&
    !dataFiltered.includes('luxCondition')) {
        dataFiltered.push('luxCondition');
    }
}
function compareData(dataFiltered, dataArrived) {
    var res = false;
    dataFiltered.forEach(function (element) {
        if (dataArrived.includes(element)) {
            res = true;
        } else {
            res = false;
        }
    });
    return res;
}
```

Consola 9: actuate.js 2/2

8.2.5. Módulo Filter

Html

El archivo html, como podemos ver en la figura 8.10, está implementado para configurar dos partes:

- ***Name***. Establece un nombre específico que le queramos dar al componente.
- ***Filter Names***. Establece los filtros, separados por comas, de los recursos que se quieren filtrar.

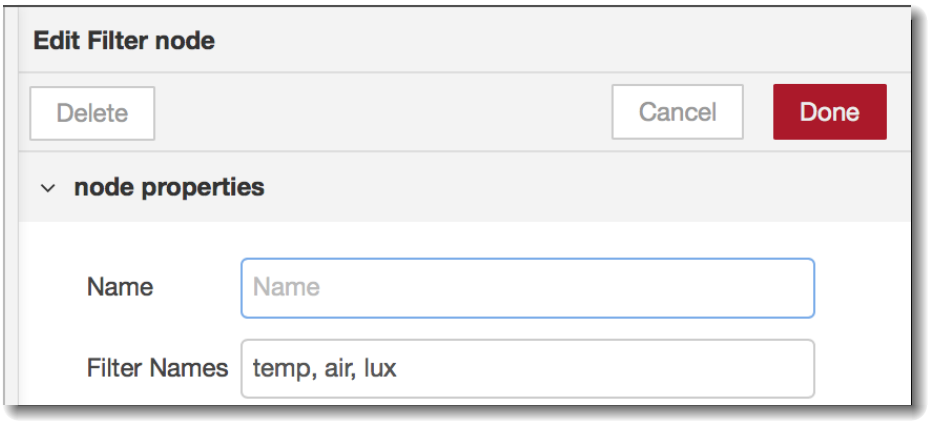


Figura 8.10: Html Filter.

Javascript

La única función existente en este archivo se encarga filtrar, comparando todos los recursos que le llegan con los recursos que se han indicado en las propiedades del modulo.

```
function Filter(config) {
    ...
    this.on('input', function (msg) {
        ...
        if (obj['resources'] != undefined) {
            endPoints = obj['resources'];
            for (var x in endPoints) {
                endPoints[x].forEach(function (element) {
                    if (filters.includes(element.resource)) {
                        endPointsFiltered.push(element);
                    }
                });
                endPoints[x]=endPointsFiltered;
            }
            msg.payload = obj;
            node.send(msg);
            return;
        }
    });
}
```

Consola 10: filter.js

8.2.6. Módulo Condition

Html

El archivo html, como vemos en la figura 8.11, está implementado para configurar tres partes:

- ***Name***. Establece un nombre específico que le queramos dar al componente.
- ***Recurso***. Establece el tipo de recurso que se quiere condicionar.
- ***Value condition***. Establece el valor de la condición.
- ***Condition***. Establece el tipo de condición.

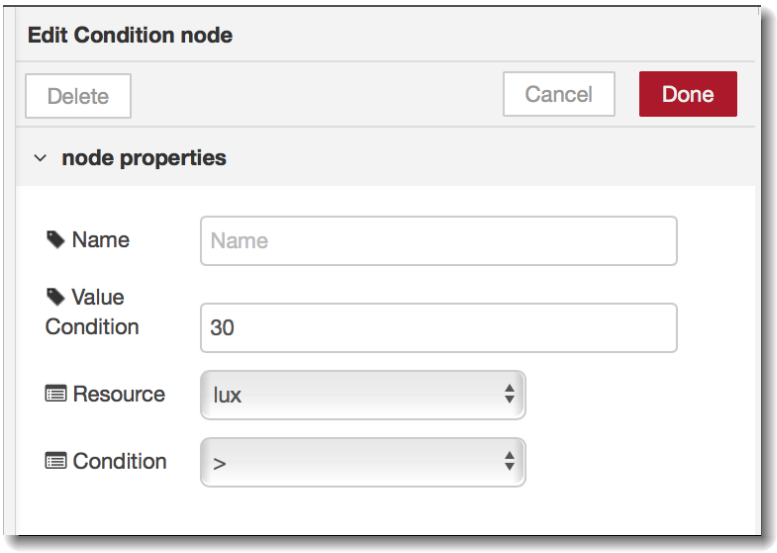


Figura 8.11: Html Condition.

Javascript

En este archivo nos encontramos con 2 funciones. La función principal se encarga de mandar el valor del recurso condicionado. Para ello nos ayudamos de otra función "compare", en la que comparamos la condición, el valor indicado en el componente a comparar y el valor recibido. Si se cumple la condición establecida mandamos el mensaje con el valor del recurso al resto del flujo.

```
function Condition(config) {
    ...
    this.on('input', function (msg) {
        if (typeof msg.payload != 'string') { obj = msg.payload; }
        else {
            if (msg.payload != 'connected') {
                obj = JSON.parse(msg.payload);
                if (obj['value'] != undefined) {
                    var valueAux = obj['value'].split(/[ ,]+/);
                    var resourceAux = resource.split(':');
                    valueAux.forEach(function (element) {
                        var aux = element.split(':');
                        if (aux[0] == 'T' && resourceAux[1] == 'T') {
                            value = Number(aux[1]);
                            resourceToSend = resourceAux[0];
                            flowContext.set('tempCondition', true);
                        }
                        if (aux[0] == 'H' && resourceAux[1] == 'H') {
                            value = Number(aux[1]);
                            resourceToSend = resourceAux[0];
                            flowContext.set('humCondition', true);
                        }
                        if (aux[0] == 'CO2' && resourceAux[1] == 'CO2') {
                            value = Number(aux[1]);
                            resourceToSend = resourceAux[0];
                            flowContext.set('airCO2Condition', true);
                        }
                        if (aux[0] == 'VOC' && resourceAux[1] == 'VOC') {
                            value = Number(aux[1]);
                            resourceToSend = resourceAux[0];
                            flowContext.set('airVOCCondition', true);
                        }
                        if (resourceAux[0] == 'lux') {
                            value = Number(aux);
                            resourceToSend = resourceAux[0];
                            flowContext.set('luxCondition', true);
                        }
                    });
                }
                ...
```

Consola 11: condition.js 1/2

```
                ...
                if (compare(condition, valueCondition, value)
                && resourceToSend == obj['resource']) {
                    msg.payload = obj;
                    node.send(msg);
                }
            }
        }
    });
}
function compare(condition, valueCondition, value) {
        var res = false;
        switch (condition) {
                case "equal":
                        if (valueCondition === value) { res = true; }
                        break;
                case "notEqual":
                        if (valueCondition !== value) { res = true; }
                        break;
                case "major":
                        if (valueCondition < value) { res = true; }
                        break;
                case "minor":
                        if (valueCondition > value) { res = true; }
                        break;
                case "majorOrEqual":
                        if (valueCondition <= value) { res = true; }
                        break;
                default:
                        if (valueCondition >= value) { res = true; }
                        break;
        }
        return res;
}
```

Consola 12: condition.js 2/2

8.2.7. Módulo Producer Kafka

Html

El archivo html, como vemos en la figura 8.12, está implementado para configurar tres partes:

- ***ZK Quorum***. Establece la dirección y el puerto del cluster de kafka.
- ***Topic(s)***. Establece el valor de los topics a producir.
- ***Debug***. Marcado, imprime información adicional en la consola.

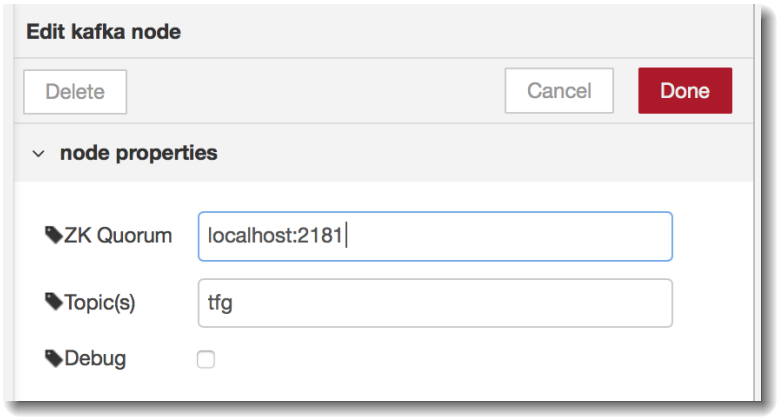


Figura 8.12: Html Producer Kafka.

8.2.8. Módulo Consumer Kafka

Html

El archivo html, como vemos en la figura 8.13, está implementado para configurar cinco partes:

- ***ZK Quorum***. Establece la dirección y el puerto del cluster de kafka.
- ***Topic(s)***. Establece el valor de los topics a producir.
- ***GroupId***. Establece el valor del identificador de grupo de los consumers.

- ***Auto Commit***. Marcado, imprime información adicional en la consola.
- ***Debug***. Marcado, hace que los mensajes se confirmen automáticamente..

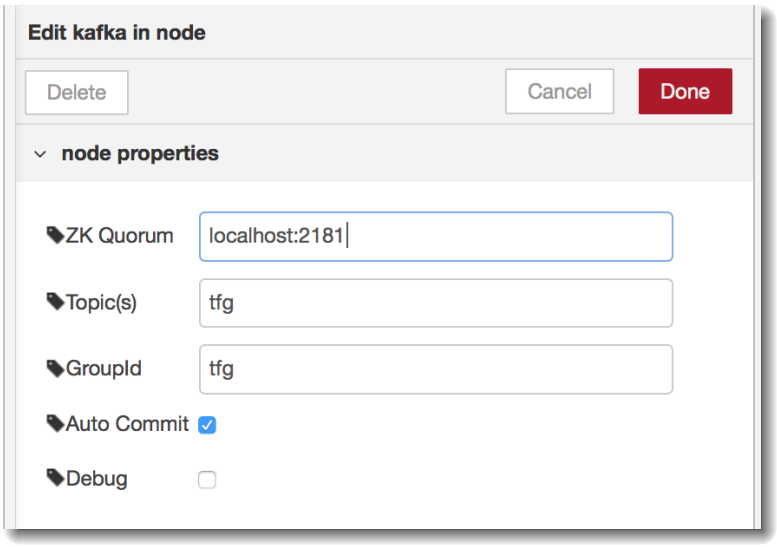


Figura 8.13: Html Consumer Kafka.

8.3. Sistema de mensajes distribuidos (Kafka)

En esta sección explicaremos como se ha procedido a configurar el sistema de mensajes distribuidos.

Primeramente debemos descargarnos los archivos necesarios para poder ejecutar el sistema Kafka a través de su pagina web [1]. Una vez hecho esto, descomprimimos dichos archivos en nuestro PC y abrir un terminal para poder ejecutar los siguientes comandos necesarios.

El siguiente paso es iniciar el Zookeper con el siguiente comando:

```
$ bin/zookeeper-server-start.sh config/zookeeper properties
```

[1] https://kafka.apache.org/downloads

Debemos de modificar el archivo “kafka-server-start.sh” añadiendo los parámetros del JMX para poder monitorizar los datos recibidos en Kafka posteriormente:

- ***export KAFKA_JMX_OPTS***
- ***export JMX_PORT=9999***

Seguido iniciamos el broker con el siguiente comando:

```
$ bin/kafka-server-start.sh config/server.properties
```

Creamos un tópico “tfg” con el siguiente comando:

```
$ bin/kafka-topics.sh --create --zookeeper localhost:2181
--replication-factor 1 --partitions 1 --topic tfg
```

Debemos crear un productor con el siguiente comando:

```
$ bin/kafka-console-producer.sh --broker-list localhost:9092
--topic tfg
```

Y un consumidor con el siguiente comando:

```
$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092
--topic tfg --from-beginning
```

Finalizado todo este proceso ya tenemos en marcha nuestra plataforma de transmisión de datos a la que llamamos “nube” en este proyecto.

Podemos monitorizar distintos datos proporcionados por Kafka. Para ello tenemos que ejecutar distintos comandos según el tipo de datos que queramos monitorizar. Por ejemplo, aquí tenemos dos ejemplos por los cuales monitorizamos el número de mensajes recibidos por segundo o el número de bytes por segundo recibidos.

```
$ ./bin/kafka-run-class.sh kafka.tools.JmxTool
 --object-name kafka.server:type=BrokerTopicMetrics,name=MessagesInPerSec
```

```
$ ./bin/kafka-run-class.sh kafka.tools.JmxTool
 --object-name kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec
```

Capítulo 9

Evaluación

En este capítulo se detallarán el resultado y análisis de las pruebas de rendimiento realizadas así como un ejemplo de caso real de uso de este proyecto.

9.1. Pruebas de rendimiento

Las pruebas realizadas tratan de estudiar el comportamiento que tiene el servidor websocket ante dos factores importantes para la IoT como son la memoria y la latencia. Para ello se ha llevado a cabo una serie de pruebas con 10, 50 y 100 clientes que se conectan al servidor websocket y piden obtener los recursos disponibles a este servidor.

La prueba consiste en el envío por parte del cliente de un mensaje con el siguiente contenido para obtener la lista de recursos disponibles:

```
{ 'op': 'Resources' }
```

El servidor responde con el siguiente mensaje, donde se muestran los recursos disponibles, en este caso tenemos 4 recursos:

```
{
  'op': 'resources',
  'resources': {
    'ertis': [{
      'resource': 'led', 'type': '2', 'method': 'core.a',
      'parameters': ['19'], 'type_name': 'LED actuator',
      'connectors': [{ 'name': 'Digital output' }], 'observe': false
    },
    {
      'resource': 'temp', 'type': '3', 'method': 'core.s',
      'parameters': ['3'],
      'type_name': 'DHTT22 temperature and humidity sensor',
      'connectors': [{ 'name': 'Digital input' }], 'observe': true
    }, {
      'resource': 'lux', 'type': '5', 'method': 'core.s',
      'parameters': [''], 'connectors': [], 'observe': true,
      'type_name': 'TLS2561 Light sensor'
    }, {
      'resource': 'air', 'type': '4', 'method': 'core.s',
      'parameters': [''], 'connectors': [], 'observe': true,
      'type_name': 'CCS811 CO2 and VOC sensor'
    }]
  }
}
```

9.1.1. Latencia

La latencia es un parámetro que define el tiempo que ocurre entre que envías una petición hasta que recibes su respuesta. En nuestro caso, podemos ver en la figura 9.1 la media de la latencia en función del numero de clientes conectados. En concreto, la media con 10 clientes ha sido 0,033043 segundos, con 50 clientes 0,120250 segundos y por último con 100 clientes 0,196336 segundos. El incremento de la media con 50 clientes con respecto a 10 ha sido de un 3,64, y de 100 clientes con respecto a 50 ha sido un 1,63. Los incrementos de latencia no siguen ningún patrón comparado con el número de clientes aumentado. Esta latencia resultante se considera una latencia baja y adecuada para el edge.

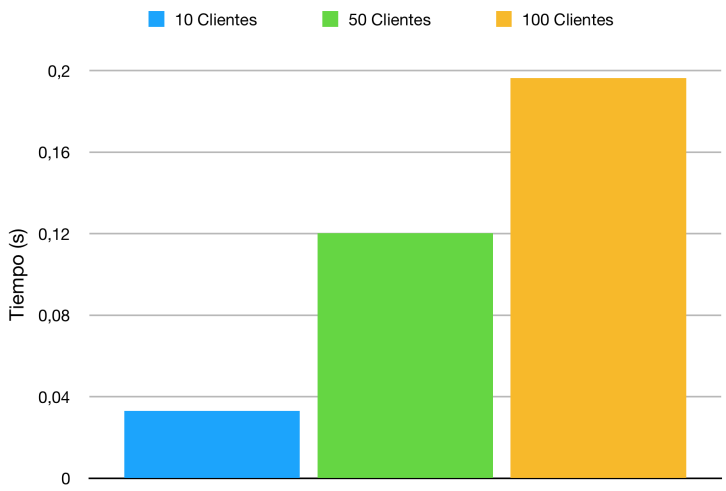


Figura 9.1: Media de latencia del servidor websocket.

9.1.2. Memoria

La gran mayoría de los dispositivos utilizados en IoT cuentan con una memoria limitada ya que son dispositivos embebidos, por lo que ajustar el consumo de memoria de nuestro sistema se convierte en tarea fundamental para obtener un rendimiento óptimo. En la figura 9.2 podemos ver la media de memoria usada en función del numero de clientes conectados. En concreto, la media de la memoria usada con 10 clientes ha sido 18.856.755 bytes, con 50 clientes 20.165.345 bytes y por último con 100 clientes 21.833.073 bytes. A pesar de que en este caso el incremento de la media de los bytes usados sigue una relación casi de 1:1, viendo las gráficas podemos concluir que el numero de clientes conectados influye en la memoria usada.

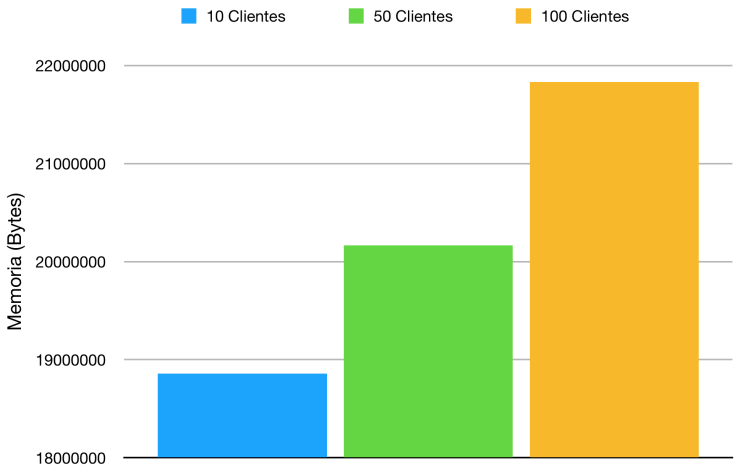


Figura 9.2: Media del consumo de memoria del servidor websocket.

9.2. Caso real de uso

A continuación se va a proceder a plasmar un ejemplo en el que nuestro proyecto tendría una aplicación real.

Nuestro sistema estaría situado en la pared de nuestra casa. Tendríamos un sensor de temperatura, un sensor de calidad de aire y un sensor de luz. Por otra parte tendríamos un actuador que en este caso sería un LED, pero también podría ser una alarma.

La situación real sería que se formara fuego por alguna circunstancia en nuestra vivienda por lo que el sensor de temperatura recogería datos de altos niveles de calor, en nuestro caso más de 30^{o}C , junto al sensor de luz que recaudaría altos niveles de iluminación, más de 200 lux, a causa del fuego. En consecuencia se activaría el LED, el cual hace que se avise a los bomberos.

El LED permanecería encendido hasta que el sensor de luz detectara que la temperatura y la iluminación de la habitación ha bajado, refiriéndose a que se ha detectado que los bomberos están apagando el fuego.

En la figura 9.3 se muestra la lógica de nuestro sistema. Para este caso se han usado otros módulos facilitados por Node-RED, como son el módulo "function", a través del cual se ha programado una función que convierte los datos recibidos como JSON a número entero para ser usados en el otro módulo utilizado, "gauge", que sirve para plasmar datos de forma gráfica en Node-RED como se puede ver en la figura 9.4.

Además se ha realizado un vídeo adjuntado con el prototipo funcionando y mostrándose las trazas llegadas al servidor websocket y los datos enviados a la nube.

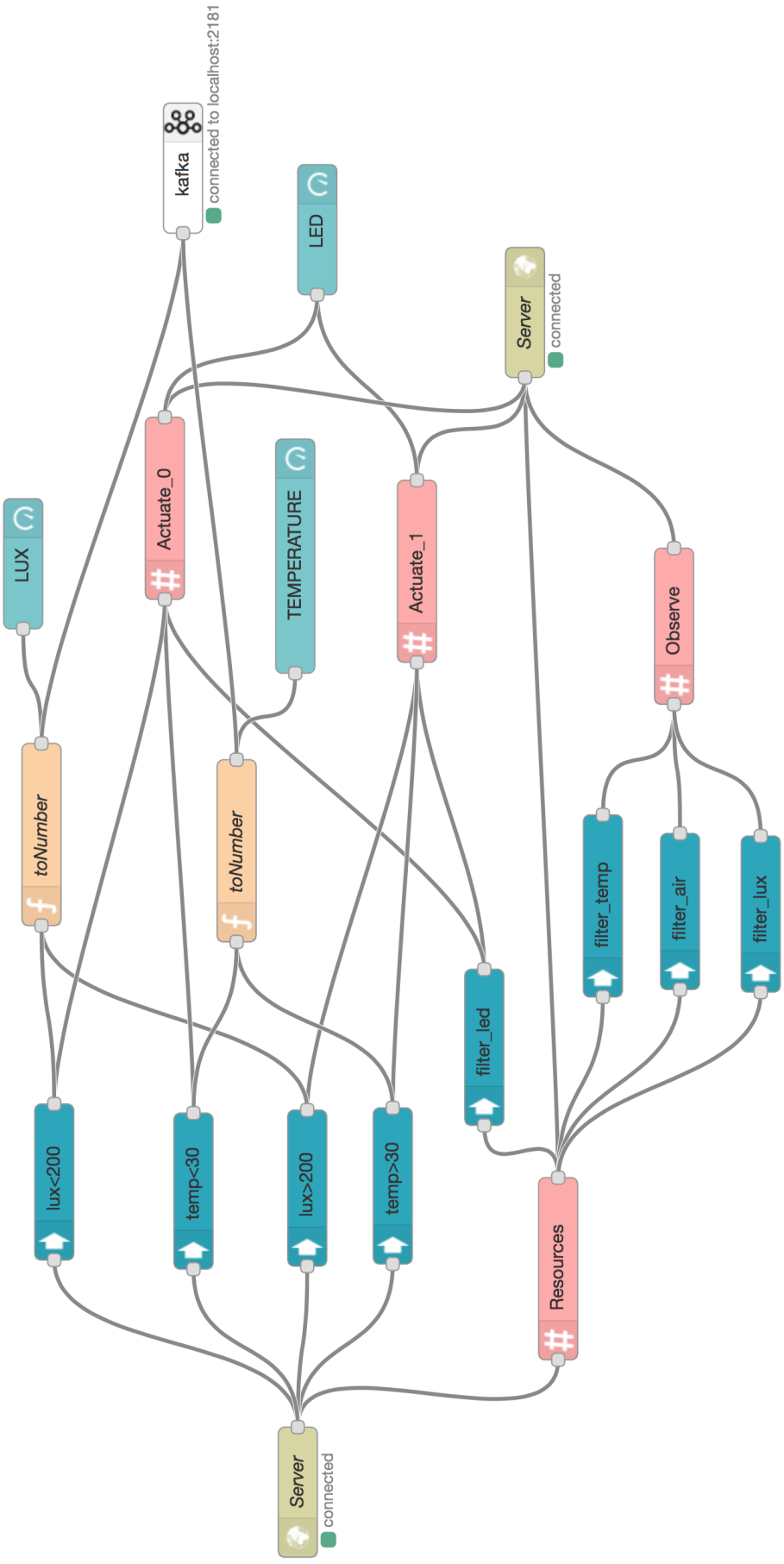


Figura 9.3: Lógica caso real.

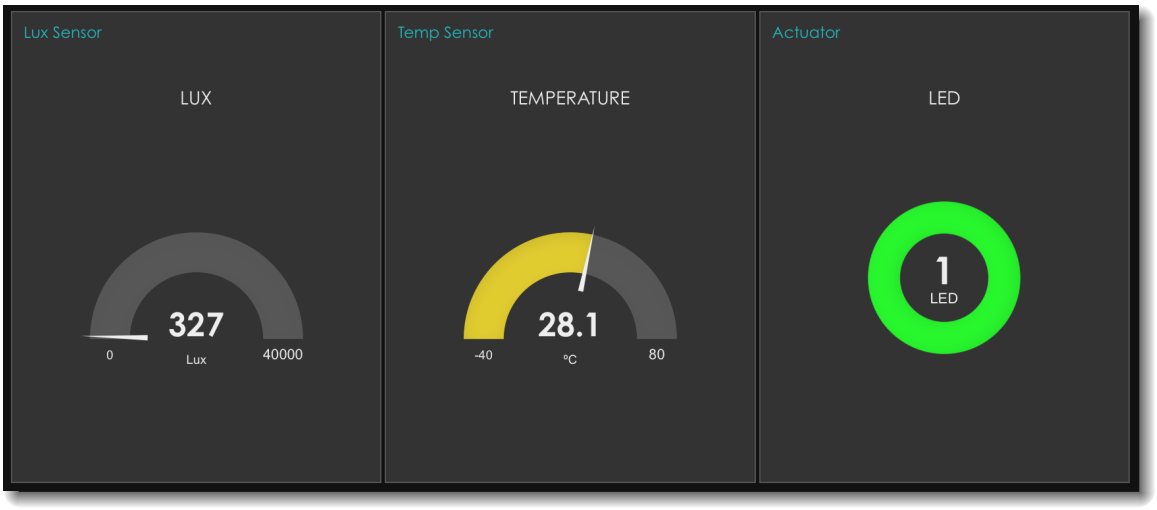


Figura 9.4: Representación gráfica de datos.

Capítulo 10

Conclusiones y líneas futuras

El resultado final de este trabajo fin de grado ha tenido dos puntos finales claros como son el desarrollo de un marco de desarrollo visual que permite la programación de aplicaciones IoT sobre Edge Computing y el desarrollo de un servidor web asíncrono que permite la interconexión asíncrona de dispositivos IoT con el marco de desarrollo mencionado.

El marco de desarrollo se compone de una serie de componentes con los que se puede interactuar con dispositivos IoT, como puede ser observar los datos de los sensores o actuar sobre un actuador IoT.

Se ha usado un prototipo con distintos sensores y actuadores para obtener una visión más cercana a la realidad de lo que sería trabajar con dispositivos IoT y el volumen de datos que se puede llegar a generar en tiempo real con varios sensores funcionando al mismo tiempo.

Esta funcionalidad es extensible en un futuro, ya que cualquier programador puede desarrollar sus propios componentes y añadirlos a dicho marco sin la necesidad de modificar las funcionalidades existentes.

Otro punto a desarrollar en un futuro sería la posibilidad de integrar otro tipo de arquitectura como puede ser “Kappa”, la cual propone solucionar los puntos débiles de la arquitectura Lambda usada en este proyecto.

Apéndice A

Manual de usuario del framework con Node-RED

A continuación, se va a detallar el funcionamiento del framework con Node-RED mediante el cual, de una manera sencilla, podemos utilizar tecnologías complejas sin tener que profundizar en ellas. Nos alejamos de todo lo que no es práctico y nos centramos en lo importante en una capa inicial donde la sencillez y rapidez de Node-RED es el punto fuerte [17].

La estructura mínima son los nodos. Estos nodos se conectan entre ellos y se organizan en flujos o flows. A través de una interfaz gráfica y sin apenas programar, podemos arrastrar esos nodos y realizar una tarea concreta como puede ser recibir una llamada HTTP, un dato de un sensor o la activación de un LED.

A.1. Iniciando la aplicación web

Una vez arrancada la aplicación web a través de la linea de comandos de nuestro equipo, observamos en la figura A.1 como el terminal nos indica que podemos acceder a ella mediante la siguiente dirección: `http://127.0.0.1:1880/`

```
iMacnuel-MBP:master manuelgranadosmolina$ node-red
8 May 20:16:01 - [info]

Welcome to Node-RED
===================

8 May 20:16:01 - [info] Node-RED version: v0.18.7
8 May 20:16:01 - [info] Node.js  version: v8.9.4
8 May 20:16:01 - [info] Darwin 17.7.0 x64 LE
8 May 20:16:01 - [info] Loading palette nodes
8 May 20:16:02 - [info] Dashboard version 2.9.6 started at /ui
8 May 20:16:02 - [warn] ---------------------------------------------------
8 May 20:16:02 - [warn] [node-red/rpi-gpio] Info : Ignoring Raspberry Pi specific node
8 May 20:16:02 - [warn] ---------------------------------------------------
8 May 20:16:02 - [info] Settings file  : /Users/manuelgranadosmolina/.node-red/settings.js
8 May 20:16:02 - [info] User directory : /Users/manuelgranadosmolina/.node-red
8 May 20:16:02 - [warn] Projects disabled : set editorTheme.projects.enabled=true to enable
8 May 20:16:02 - [info] Flows file     : /Users/manuelgranadosmolina/.node-red/flows_iMacnuel
-MBP.local.json
8 May 20:16:02 - [warn]


8 May 20:16:02 - [info] Starting flows
8 May 20:16:02 - [info] Started flows
8 May 20:16:02 - [info] Server now running at http://127.0.0.1:1880/
```

Figura A.1: Información consola Node-RED.

Al acceder con un navegador a la dirección indicada, podemos ver en la figura A.2 como se muestra la interfaz de la aplicación con la que programaremos en Node-RED.

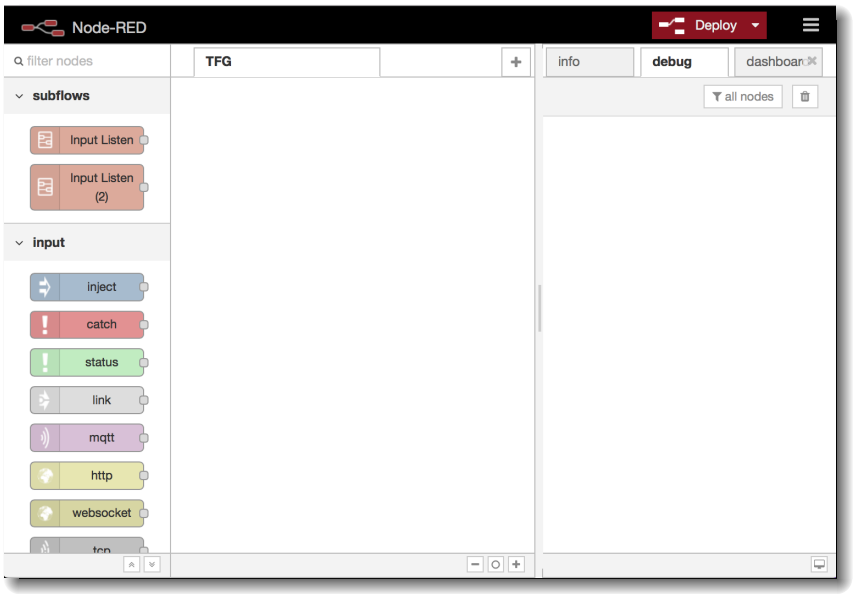


Figura A.2: Nuevo Flow Node-RED.

A.2. Entorno de desarrollo de Node-RED

En la figura A.3 podemos ver como Node-RED se distribuye en las siguientes secciones principales.

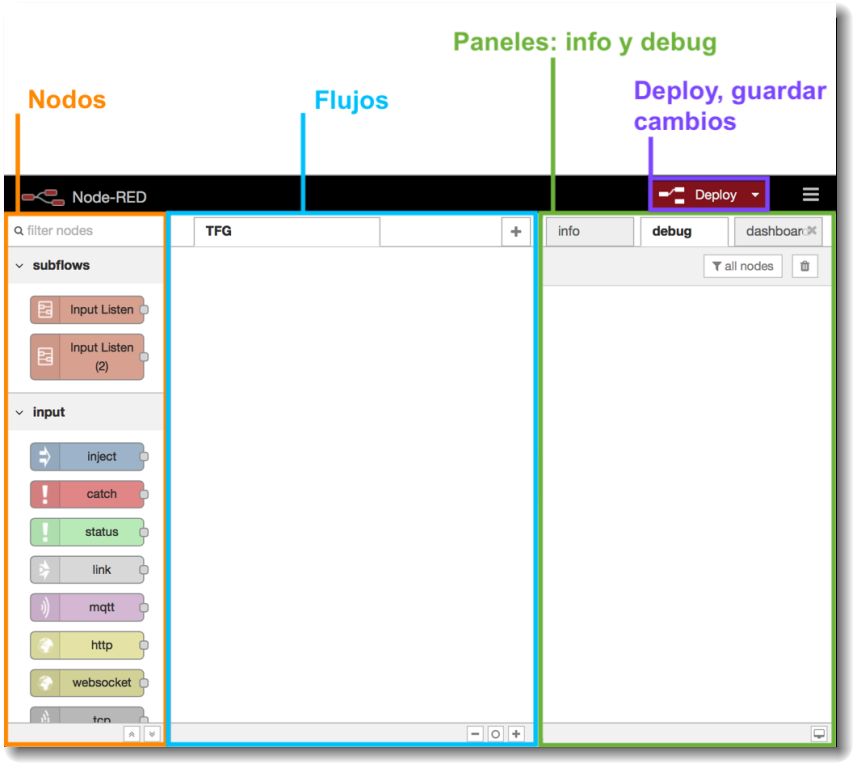


Figura A.3: Secciones Node-RED.

En la sección de la parte izquierda podemos ver la lista de nodos que vienen instalados por defecto y organizados en categorías según su funcionalidad.

Se pueden clasificar en tres tipos de nodos:

- Nodos con entradas: sólo admiten datos de entrada para ser enviados a algún sitio como pueda ser una base de datos o un panel de control.
- Nodos con salidas: son los nodos que sólo ofrecen datos tras recibirlos a través de diferentes métodos como por ejemplo un mensaje MQTT u otro nodo.
- Nodos con entradas y salidas: estos nodos nos permiten la entrada de datos y luego ofrecen una o varias salidas. Por ejemplo, podemos leer una temperatura, transformarla en grados Celsius y enviarla a otro nodo.

La otra sección es el flujo o flow, el lugar donde arrastraremos los nodos para darle lógica a nuestro diseño.

Por último, el panel de debug sirve para mostrar mensajes de lo que está ocurriendo dentro de cada flujo o tratamiento de datos. Tiene un comportamiento similar a la consola de nuestro PC [17].

A.3. Primer flujo con Node-RED

Una vez conocidas las partes principales de la interfaz procederemos a probar un flujo sencillo donde mostraremos un texto en el panel de debug.

Arrastramos un nodo inject de la categoría input y un nodo debug de la categoría output al flujo. Seguidamente los conectamos entre ellos arrastrando el punto de la parte derecha del nodo inject sobre el punto de la parte izquierda del nodo debug, quedándose como se muestra en la figura A.4.

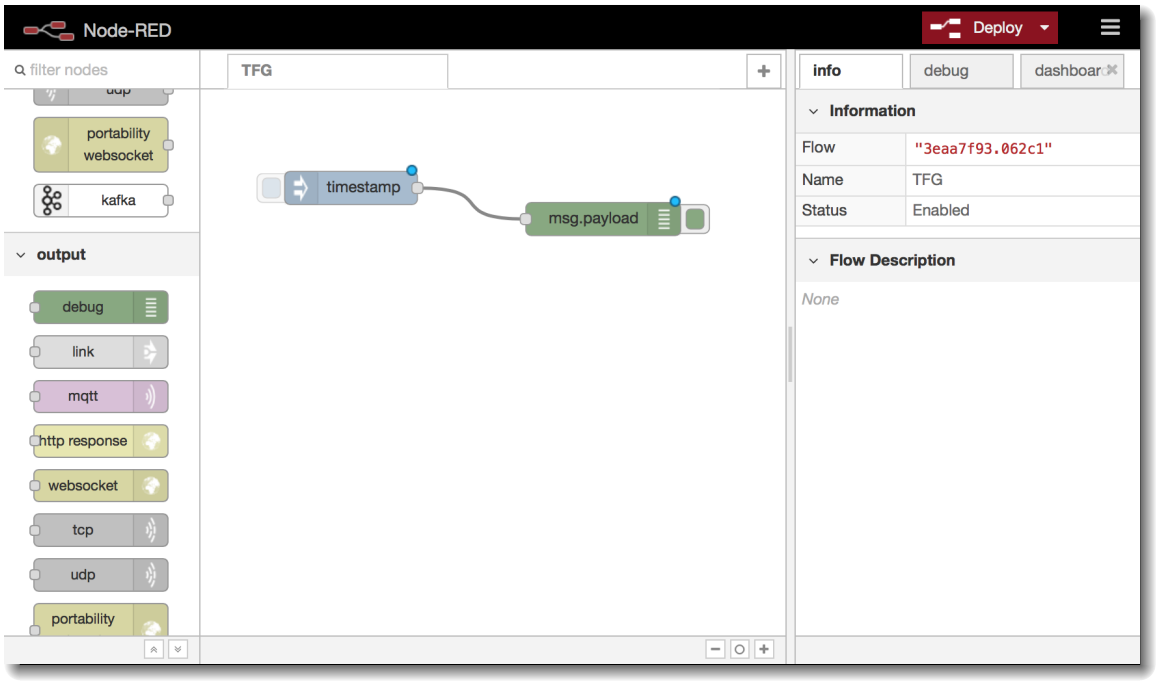


Figura A.4: Flujo sencillo.

A continuación vamos a editar el nodo inject. Hacemos doble click sobre el nodo y se abrirá un panel donde nos muestra diferentes parámetros. Seleccionamos del menú

desplegable donde pone Payload, la opción String y en el campo de texto escribimos "Hola Mundo". Por último hacemos click en Done.

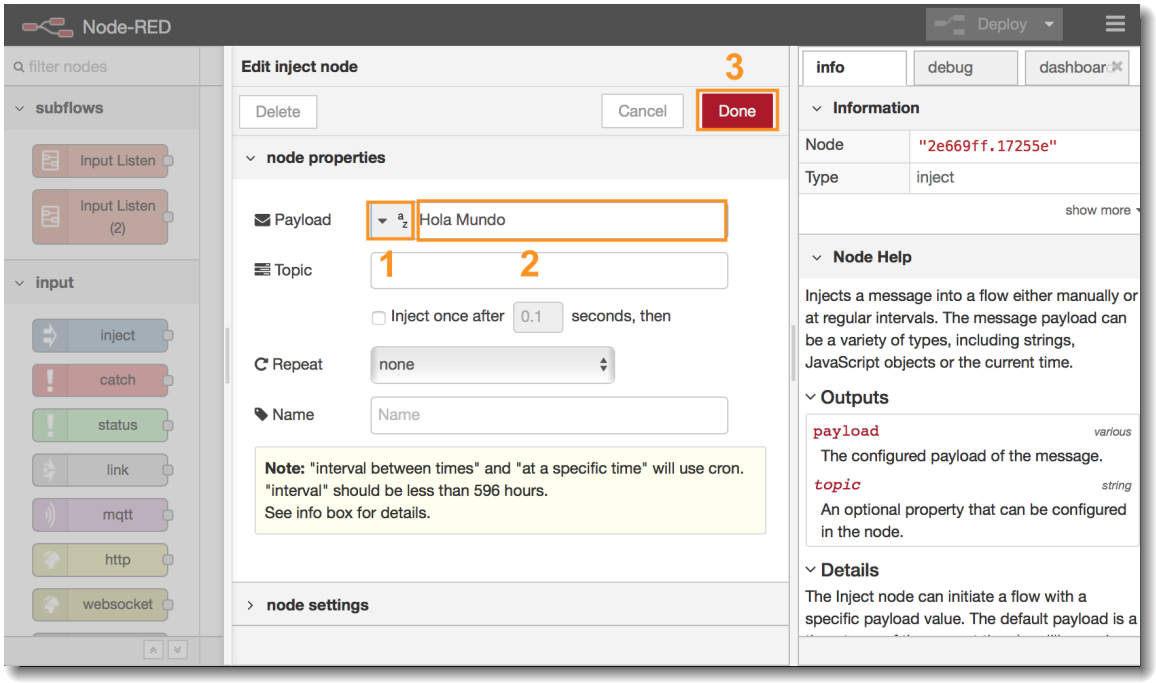


Figura A.5: Nuevo Flow Node-RED.

Para guardar la aplicación tendremos que hacer click en el botón Deploy y nos saldrá un mensaje de confirmación como en la figura A.7, significando que se han guardado todos los cambios del flujo de Node-RED.

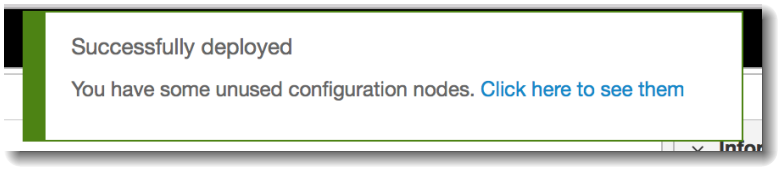


Figura A.6: Mensaje compilación Node-RED.

A.4. Probando el primer flujo con Node-RED

Para probar este primer flujo con Node-RED abriremos el panel de debug que está situado en la parte derecha y haremos click sobre la parte izquierda del nodo inject.

Por cada click que hagamos en el nodo, en el panel debug aparecerá "Hola Mundo" como podemos ver en el figura A.7.

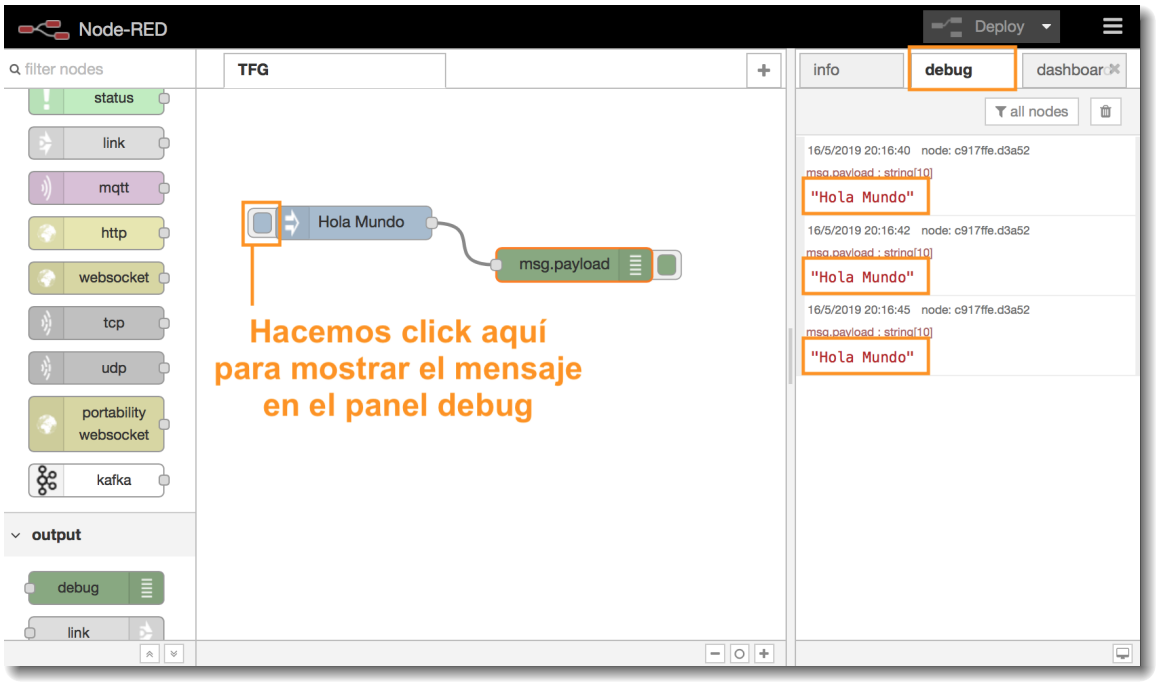


Figura A.7: Salida mensajes nodo.

A.5. Añadir nuevos nodos a Node-RED

Node-RED viene con un conjunto de nodos útiles, pero hay un gran número de nodos adicionales disponibles para instalar tanto desde el proyecto Node-RED como desde la comunidad en general.

A.5.1. Usando el editor

Podemos instalar nodos directamente usando el editor. Para hacerlo, seleccionamos "Manage Palette" en el menú (arriba a la derecha) como vemos en la figura A.8 y luego seleccionamos la pestaña "install" en la paleta como vemos en la figura A.9, y buscamos los nuevos nodos que queramos instalar.

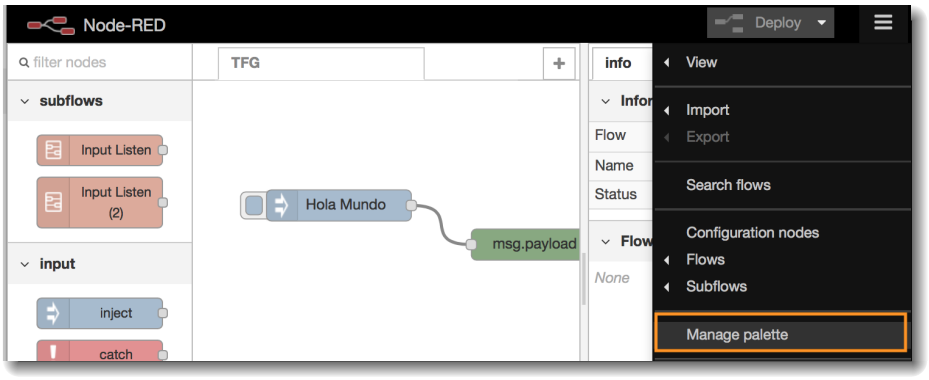


Figura A.8: Añadir nodo.

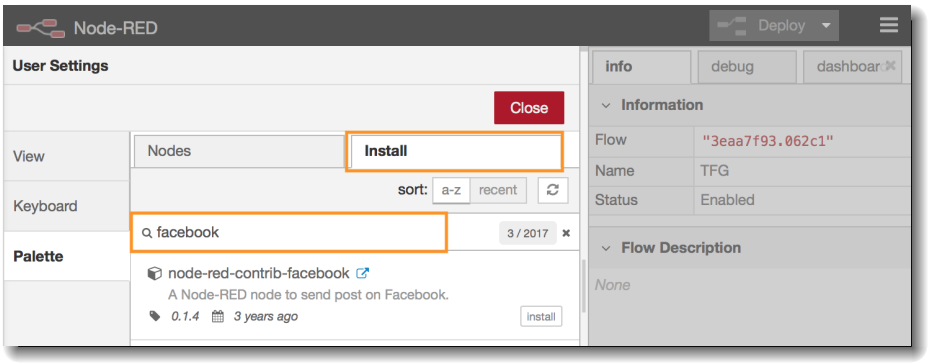


Figura A.9: Añadir nodo.

A.5.2. Instalación de nodos empaquetados npm

Para instalar un nodo empaquetado con npm, podemos instalarlo, como vemos en la consola 13, localmente dentro de nuestro directorio de datos de usuario:

```
cd $HOME/.Node-RED
npm install <npm-package-name>
```

Consola 13: Añadir nuevo nodo.

Para tener el nodo instalado disponible debemos detener y reiniciar Node-RED para que capte los nuevos nodos.

A.6. Creación de nuevos nodos

Además de poder añadir nodos existentes, también podemos crear nuestros propios nodos como se ha hecho en este TFG.

Para ello debemos crear dos archivos nuevos; un archivo JavaScript que define lo que hace el nodo, y un archivo html que define sus propiedades, el diálogo de edición y el texto de ayuda. También tenemos que modificar el fichero de configuración del nuevo paquete,"package.json", para añadir nuestro nuevo nodo.

Vamos a describir como creamos por ejemplo el nodo utilizado en este proyecto, "Resources". Tenemos que crear un directorio, "Node-RED-websockets", donde desarrollamos el código. Dentro de este directorio, creamos los siguientes archivos:

- package.json
- resources.js
- resources.html

A.6.1. package.json

Este es un archivo estándar utilizado por los módulos Node.js para describir sus contenidos. En él le indicamos una serie de parámetros para su correcto funcionamiento como se indica en la consola 14:

```
{
    "name": "Node-RED-websockets",
    "version": "1.0.0",
    "description": "Nodos Websockets",
    "dependencies": {},
    "keywords": ["Node-RED"],
    "Node-RED": {
        "nodes": { "resources": "resources.js" }
    }
}
```

Consola 14: package.json

A.6.2. resources.js

En este archivo definiremos lo que hará el nodo. El nodo en sí está definido por una función, "Resources" que se llama cada vez que se crea una nueva instancia del nodo. Se pasa un objeto que contiene las propiedades específicas del nodo establecidas en el editor de flujo.

```
module.exports = function (RED) {
    function Resources(config) {
        RED.nodes.createNode(this, config);
        var node = this;
        var flowContext = this.context().flow;
        this.on('input', function (msg) {
            if (msg.client != null) {
                msg.payload = { "op": "Resources" };
            }
            node.send(msg);
        });
    }
    RED.nodes.registerType("Resources", Resources);
};
```

Consola 15: resources.js

La función llama a la función "RED.nodes.createNode" para inicializar las funciones compartidas por todos los nodos. Después de eso, el código específico del nodo vive.

En este caso, el nodo registra una escucha del evento "input" al que se llama cada vez que llega un mensaje al nodo. Dentro de este oyente, se introduce el mensaje en formato JSON que será enviado y luego llama a la función "send" para enviar el mensaje al flujo.

Por último, la función "Resources" se ha registrado en el tiempo de ejecución usando el nombre para el nodo, "Resources".

A.6.3. resources.html

```
<script type="text/javascript">
    RED.nodes.registerType('Resources',{
        category: 'Resources',
        color: '#ffaaaa',
        defaults: {
            name: {value:""}
        },
        inputs:1,
        outputs:1,
        icon: "hash.png",
        label: function() {
            return this.name||"Resources";
        }
    });
</script>

<script type="text/x-red" data-template-name="Resources">
    <div class="form-row">
        <label for="node-input-name"><i class="icon-tag"></i>
        Name</label>
        <input type="text" id="node-input-name" placeholder="Name">
    </div>
</script>

<script type="text/x-red" data-help-name="Resources">
    <p>A node Resources<br/>
    </p>
</script>
```

Consola 16: resources.html

El archivo HTML de un nodo proporciona lo siguiente:

- La definición del nodo principal que se registra con el editor.
- La plantilla de edición.
- El texto de ayuda.

En este nodo solo se tiene una única propiedad editable, “name”.

A.7. Instalar el nuevo nodo creado

Una vez creado nuestro nuevo nodo como se describe anteriormente, podemos instalarlo en Node-RED con los siguientes comandos:

```
cd C:\Users\manuel\.node_red

npm install C:\Users\manuel\Documents\Node-RED-websocket
```

Consola 17: Instalar nuevo nodo.

Para tener el nodo instalado disponible debemos detener y reiniciar Node-RED para que capte los nuevos nodos.

Apéndice B

Manual de usuario de la aplicación web

A continuación se va a detallar el funcionamiento de una aplicación web que ha sido desarrollada por Cristian Martín Fernández y Pablo Martín Fernández para que cualquier usuario registrado en ella pueda interactuar con los sensores o actuadores del sistema conectados a los dispositivos Arduino de una manera más cómoda.

B.1. Iniciando la aplicación web

Una vez arrancada la aplicación web a través de la linea de comandos de nuestro equipo, observamos en la figura B.1 como el terminal nos indica que podemos acceder a ella mediante la siguiente dirección: `http://127.0.0.1:8000/`

Al acceder con un navegador a la dirección indicada, podemos ver en la figura B.2 como se muestra un panel de login en el que debemos introducir los datos de un usuario registrado para acceder al sistema.

```
iMacnuel-MBP:Lambda-UI manuelgranadosmolina$ ./main.sh
Performing system checks...

System check identified no issues (0 silenced).
January 11, 2019 - 17:20:39
Django version 1.9.4, using settings 'TFG.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Figura B.1: Información consola aplicación web.

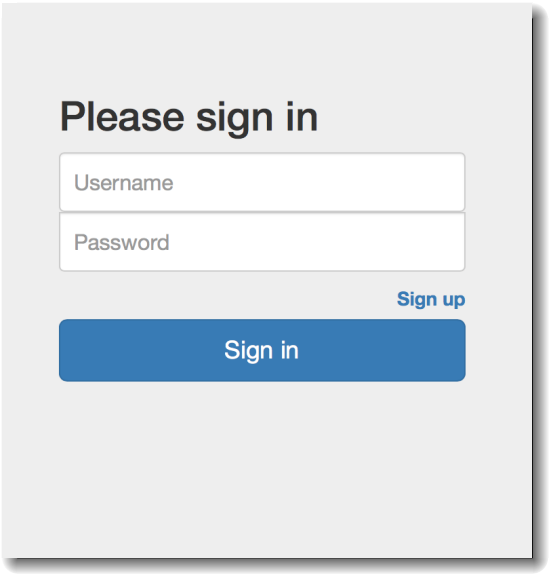


Figura B.2: Inicio de sesión de la aplicación web.

B.2. Conexión con un End Point

La conexión con un End Point se realiza automáticamente al conectar uno de los nodos a una fuente de alimentación, como la conexión mediante USB al PC. Realizado esto y después de haber accedido a través del Login a nuestra aplicación web, en la figura B.3 vemos como se muestra una lista con los End Points disponibles indicando el nombre, la ultima actualización, el tiempo de refresco en segundos y un botón para acceder a la lista de los recursos de cada uno de ellos.

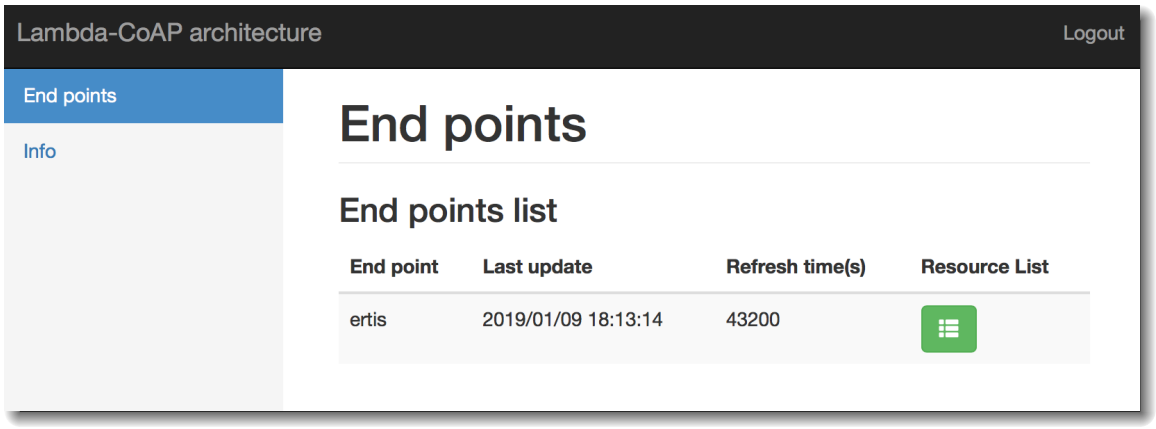


Figura B.3: Lista de End-point.

En el contenido "Consola 18" observamos la petición que se realiza en la aplicación web para obtener la lista de End Points.

```
[09/Jan/2019 00:00:00] "GET /nodes HTTP/1.1" 200 79
```

Consola 18: Obtener lista End-Point.

En el contenido "Consola 19" observamos la petición que se realiza en el servidor para obtener la lista de End Points.

```
127.0.0.1 - [09/Jan/2019 00:00:00] "GET /nodes HTTP/1.1" 200 79
```

Consola 19: Obtener lista End-Point.

Si el equipo no tiene ningún End Point conectado, en la figura B.4 se muestra el mensaje de error que salta indicando que el servidor no está disponible.

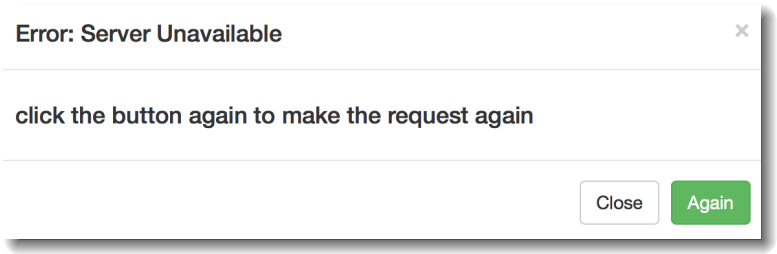


Figura B.4: Error al listar End point.

B.3. Añadir recursos

Después de haber conectado el nodo, pulsamos sobre el botón para ver los recursos disponibles y como indica la figura B.5 la lista aparece vacía en un principio a la espera de añadir nuevos recursos.

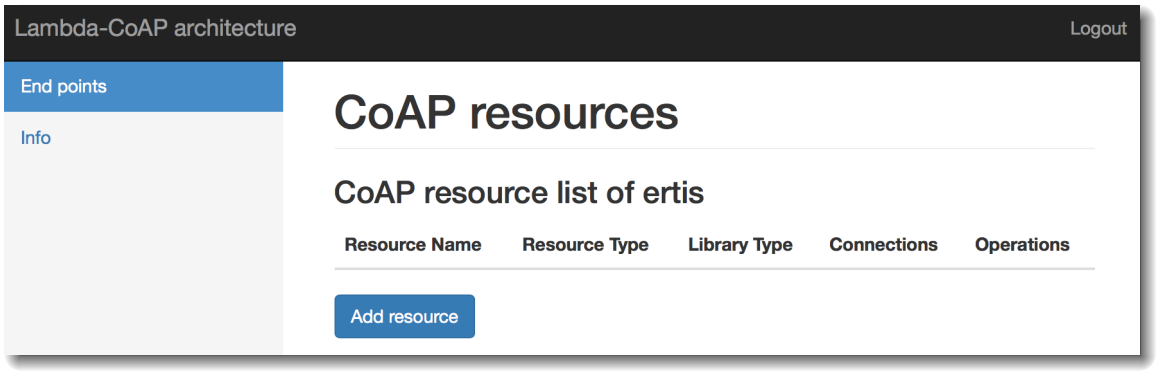


Figura B.5: Recursos Coap.

En el contenido "Consola 20" observamos la petición que se realiza en el servidor para obtener la lista de recursos.

```
127.0.0.1 - [09/Jan/2019 00:00:00]
"GET /ertis/.well-known/core HTTP/1.1" 200 2
```

Consola 20: Obtener lista recursos.

En el contenido "Consola 21" observamos la petición que se realiza en la aplicación web cada vez que vamos a añadir un recurso y se obtienen las librerías disponibles en el nodo.

```
[09/Jan/2019 00:00:00] "GET /ertis/lib HTTP/1.1" 200 325
```

Consola 21: Añadir recurso.

En el contenido "Consola 22" observamos la petición que se realiza en el servidor cada vez que vamos a añadir un recurso y se quieren obtener las librerías disponibles.

```
b'R\x01\x00\x02\x93lib!'
DATA ENVIADA
2019/01/09 00:00:00 Packet:{'id': 'rx',
'source_addr_long':
b'\x00\x13\xa2\x00@\xb1\xd2\xa6',
'source_addr': b'g\x03',
'options': b'\x01',
'rf_data': b'RE\x00\x02\x10\xa1\x14</lib>;a="2 3 5 4"'}

Data received
127.0.0.1 - [09/Jan/2019 00:00:00] "GET /ertis/lib HTTP/1.1" 200 325
```

Consola 22: Añadir recurso.

Procedemos a añadir los distintos recursos que tenemos disponibles en el dispositivo, empezando por el sensor de temperatura y humedad relativa, donde en la figura B.6 vemos como tenemos que indicar el puerto donde esta conectado al sistema, en este caso el '3".

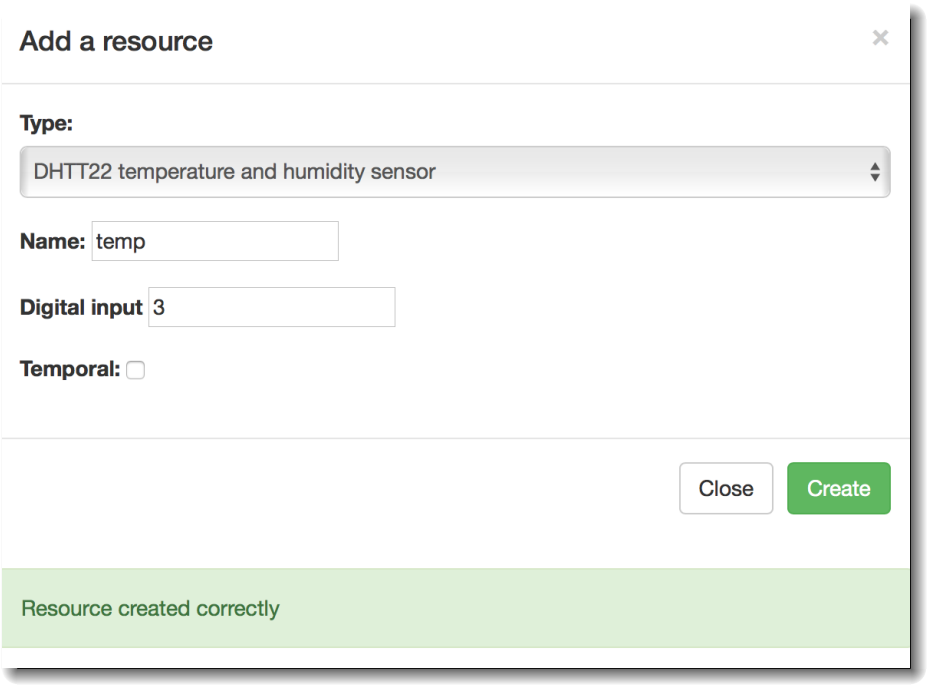


Figura B.6: Añadir recurso - Temperatura y humedad.

En el contenido "Consola 23" observamos la petición que se realiza en la aplicación web para añadir el recurso de temperatura y humedad relativa.

```
temp/3
URLhttp://localhost:8080/ertis/temp/3?p=[3]&t=0
[09/Jan/2019 00:00:00]
"POST /ertis/temp/3?p=[3]&t=0 HTTP/1.1" 200 29
```

Consola 23: Añadir recurso - Temperatura y humedad.

En el contenido "Consola 24" observamos la petición que se realiza en el servidor para añadir el recurso de temperatura y humedad relativa.

```
b'R\x02\x00\x02\x94temp!\x14</temp>;rt="3";p="3";t="0'
DATA ENVIADA
2019/01/09 00:00:00 Packet:{'id': 'rx',
'source_addr_long':
b'\x00\x13\xa2\x00@\xb1\xd2\xa6',
'source_addr': b'g\x03',
'options': b'\x01',
'rf_data': b'QA\x00\x02\xb1\x14'}
127.0.0.1 - [09/Jan/2019 00:00:00]
"POST /ertis/temp/3?p=[3]&t=0 HTTP/1.1" 200 29
```

Consola 24: Añadir recurso - Temperatura y humedad.

Seguidamente la figura B.7 indica como añadimos el sensor de luz, en este caso no tenemos que indicar el puerto ya que está conectado a través de I2C.

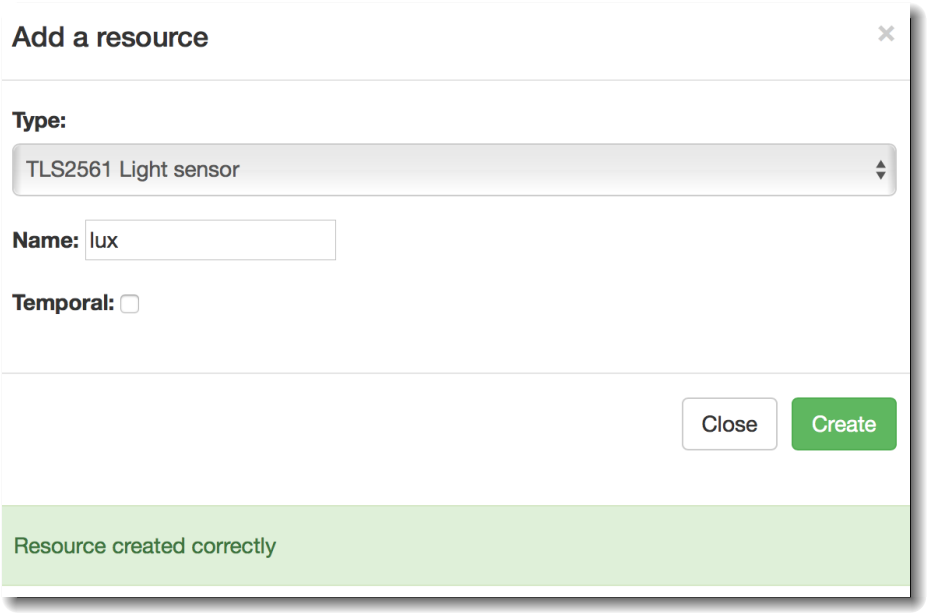


Figura B.7: Añadir recurso - Luz.

En el contenido “Consola 25” observamos la petición que se realiza en la aplicación web para añadir el recurso de luz.

```
lux/5
URLhttp://localhost:8080/ertis/lux/5?p=[]&t=0
[09/Jan/2019 00:00:00] "POST /ertis/lux/5?p=[]&t=0 HTTP/1.1" 200 31
```

Consola 25: Añadir recurso - Luz.

En el contenido “Consola 26” observamos la petición que se realiza en el servidor para añadir el recurso de luz.

```
b'R\x02\x00\x02\x93lux!\x14</lux>;rt="5";p="";t="0'
DATA ENVIADA
2019/01/09 00:00:00 Packet:{'id': 'rx',
'source_addr_long':
b'\x00\x13\xa2\x00@\xb1\xd2\xa6',
'source_addr': b'g\x03',
'options': b'\x01',
'rf_data': b'QA\x00\x02\xb1\x14'}

127.0.0.1 - [09/Jan/2019 00:00:00]
"POST /ertis/lux/5?p=[]&t=0 HTTP/1.1" 200 31
```

Consola 26: Añadir recurso - Luz.

Seguidamente, en la figura B.8 vemos como añadimos el sensor de calidad del aire.

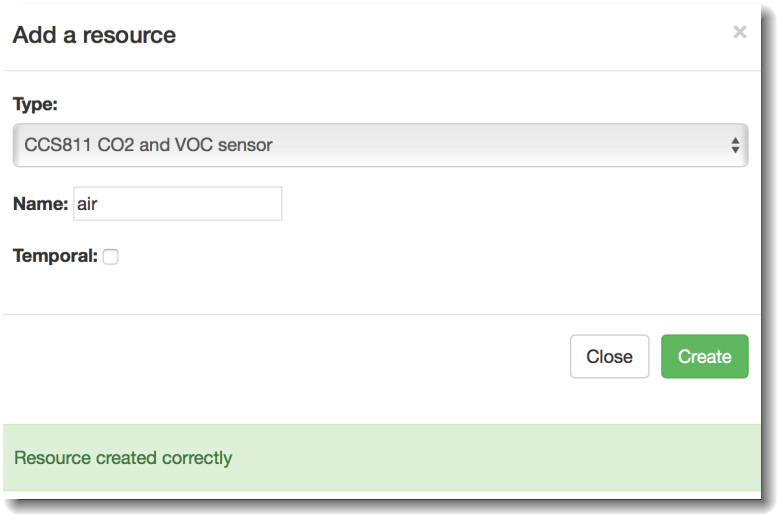


Figura B.8: Añadir recurso - Calidad de aire.

En el contenido “Consola 27” observamos la petición que se realiza en la aplicación web para añadir el recurso de calidad del aire.

```
air/4
URLhttp://localhost:8080/ertis/air/4?p=[]&t=0
[09/Jan/2019 00:00:00] "POST /ertis/air/4?p=[]&t=0 HTTP/1.1" 200 31
```

Consola 27: Añadir recurso - Calidad de aire.

En el contenido “Consola 28” observamos la petición que se realiza en el servidor para añadir el recurso de calidad del aire.

```
b'R\x02\x00\x02\x93air!\x14</air>;rt="4";p="";t="0'
DATA ENVIADA
2019/01/09 00:00:00 Packet:{'id': 'rx',
'source_addr_long':
b'\x00\x13\xa2\x00@\xb1\xd2\xa6',
'source_addr': b'g\x03',
'options': b'\x01',
'rf_data': b'QA\x00\x02\xb1\x14'}

127.0.0.1 - [09/Jan/2019 00:00:00]
"POST /ertis/air/4?p=[]&t=0 HTTP/1.1" 200 31
```

Consola 28: Añadir recurso - Calidad de aire.

En nuestro sistema, también tenemos un actuador Led, que para añadirlo tenemos que indicarte el puerto en el que se encuentra conectado como se ve en la figura B.9.

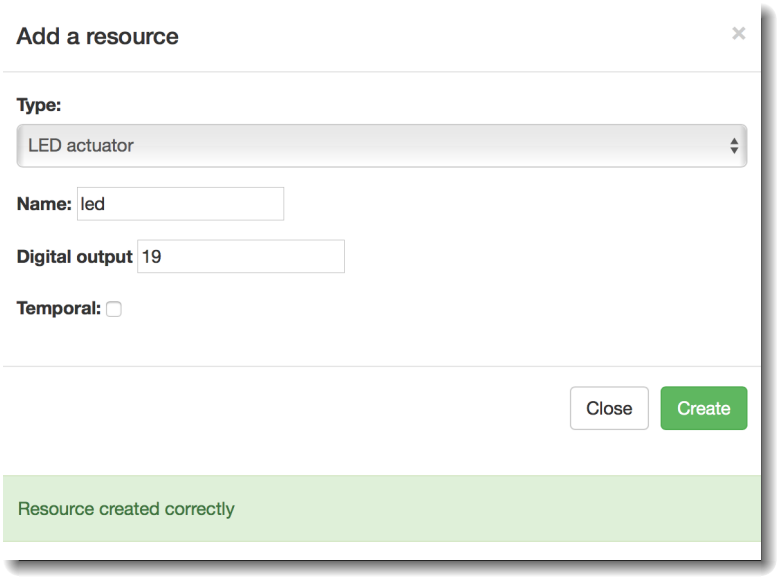


Figura B.9: Añadir recurso - Actuador Led.

En el contenido "Consola 29" observamos la petición que se realiza en la aplicación web para añadir el actuador Led.

```
led/2
URLhttp://localhost:8080/ertis/led/2?p=[19]&t=0
[09/Jan/2019 00:00:00] "POST /ertis/led/2?p=[19]&t=0 HTTP/1.1" 200 31
```

Consola 29: Añadir recurso - Actuador Led.

En el contenido "Consola 30" observamos la petición que se realiza en el servidor para añadir el actuador Led.

```
b'R\x02\x00\x02\x93led!\x14</led>;rt="2";p="19";t="0'
DATA ENVIADA
2019/01/09 00:00:00 Packet:{'id': 'rx',
'source_addr_long':
b'\x00\x13\xa2\x00@\xb1\xd2\xa6',
'source_addr': b'g\x03',
'options': b'\x01',
'rf_data': b'QA\x00\x02\xb1\x14'}

127.0.0.1 - [09/Jan/2019 00:00:00]
"POST /ertis/led/2?p=[19]&t=0 HTTP/1.1" 200 31
```

Consola 30: Añadir recurso - Actuador Led.

Llegados a este punto hemos añadido todos los sensores y actuadores utilizados en este proyecto. Seguidamente nos dirigimos al listado de los recursos que están disponibles en nuestro nodo. En la figura B.10 se muestra una lista con el nombre, tipo de recurso, tipo de librería y puerto en el que está conectado cada uno de esos recursos.

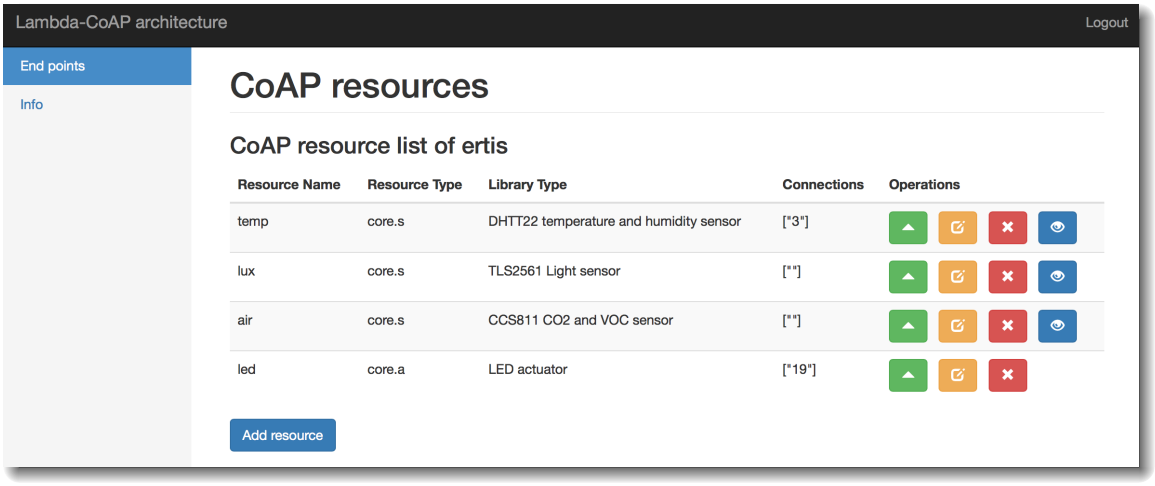


Figura B.10: Lista recursos Coap.

B.4. Editar recursos

Si por cualquier razón nos hubiésemos equivocado en algún puerto o nombre que hemos asignado a un recurso en concreto, cabe la posibilidad de editar dicho recurso pulsando en el botón amarillo que hay situado a la derecha y se nos abrirá una ventana, como vemos en la figura B.11, en la que modificar los parámetros como el nombre o el puerto.

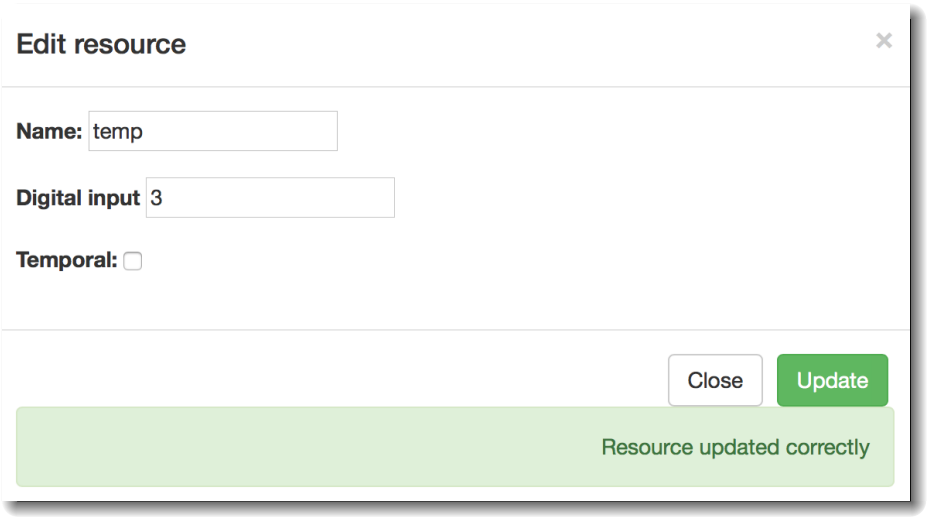


Figura B.11: Editar recurso.

En el contenido “Consola 31” observamos la petición que se realiza en la aplicación web para editar el sensor de temperatura.

```
URLhttp://localhost:8080/ertis/temp/temp?p=[3]&t=0
[09/Jan/2019 00:00:00] "PUT /ertis/temp/temp?p=[3]&t=0 HTTP/1.1" 200 43
```

Consola 31: Editar recurso.

En el contenido “Consola 32” observamos la petición que se realiza en el servidor para editar el sensor de temperatura.

```
b'R\x03\x00\x02\x94temp!\x14</temp>;p="3";t="0'
DATA ENVIADA
2019/01/09 00:00:00 Packet:{'id': 'rx', 'source_addr_long':
b'\x00\x13\xa2\x00@\xb1\xd2\xa6',
'source_addr': b'g\x03',
'options': b'\x01', 'rf_data': b'QD\x00\x02\xb1\x14'}

Resource changed
Recibido actualizacion de recursos
127.0.0.1 - - [09/Jan/2019 00:00:00]
"PUT /ertis/temp/temp?p=[3]&t=0 HTTP/1.1" 200 43
```

Consola 32: Editar recurso.

B.5. Eliminar recursos

Para eliminar cualquier recurso añadido en un principio debemos pulsar en el botón rojo situado a la derecha y aparecerá una alerta de confirmación como en la figura B.12. Si no se marca la opción “Temporal”, cuando el nodo se reinicia se recuperan todos los cambios guardados en los recursos.

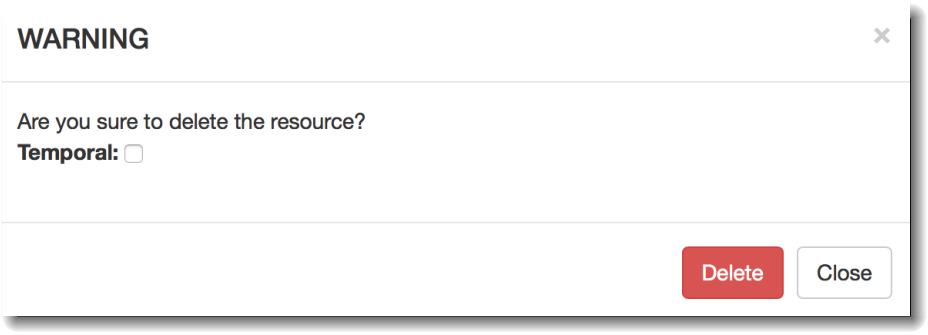


Figura B.12: Eliminar recurso.

En el contenido "Consola 33" observamos la petición que se realiza en la aplicación web para eliminar el sensor de temperatura.

```
[09/Jan/2019 00:00:00] "DELETE /ertis/temp?t=0 HTTP/1.1" 200 30
```

Consola 33: Eliminar recurso.

En el contenido "Consola 34" observamos la petición que se realiza en el servidor para eliminar el sensor de temperatura.

```
b'R\x04\x00\x02\x94temp!\x14</temp>;t="0'
DATA ENVIADA
2019/01/09 00:00:00 Packet:{'id': 'rx', 'source_addr_long':
b'\x00\x13\xa2\x00@\xb1\xd2\xa6',
'source_addr': b'g\x03',
'options': b'\x01', 'rf_data': b'QB\x00\x02\xb1\x14'}

Recibido actualizacion de recursos
127.0.0.1 - - [09/Jan/2019 00:00:00]
"DELETE /ertis/temp?t=0 HTTP/1.1" 200 30
```

Consola 34: Eliminar recurso.

B.6. Obtener datos de un recurso

Si queremos obtener los datos de un recurso debemos pulsar el botón verde situado a la derecha y se mostrará una ventana con la información correspondiente según el recurso elegido.

En este primer caso podemos ver en la figura B.13 como obtenemos la humedad relativa y la temperatura.

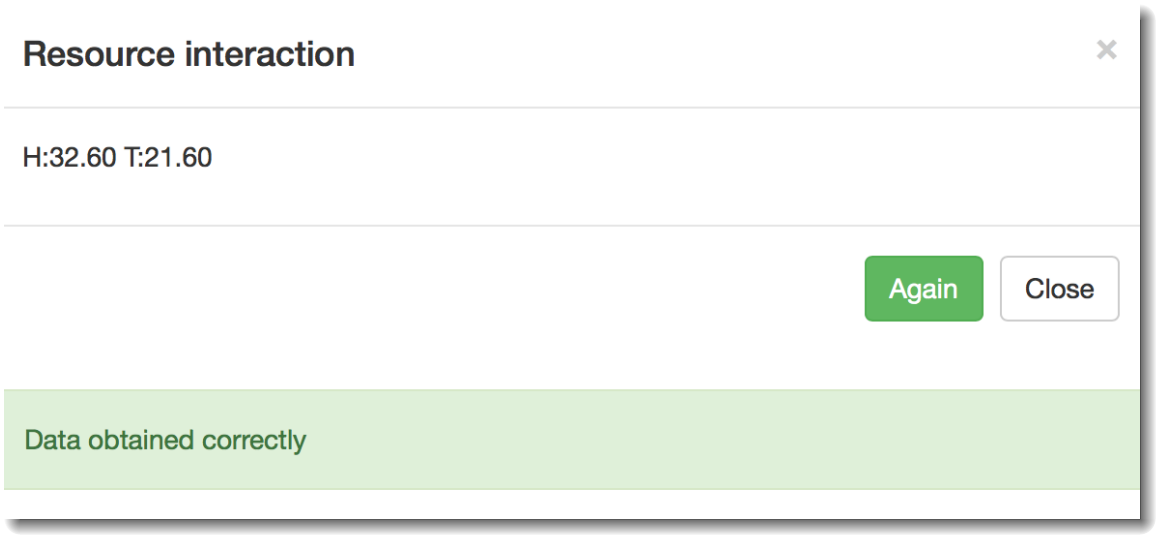


Figura B.13: Datos - Temperatura y Humedad.

En el contenido "Consola 35" observamos la petición que se realiza en la aplicación web para obtener los datos del recurso de temperatura.

```
[09/Jan/2019 00:00:00] "GET /ertis/temp HTTP/1.1" 200 58
```

Consola 35: Datos - Temperatura y Humedad.

En el contenido "Consola 36" observamos la petición que se realiza en el servidor para añadir el obtener los datos del recurso de temperatura.

```
b'R\x01\x00\x02\x94temp!'
DATA ENVIADA
2019/01/09 00:00:00 Packet:{'id': 'rx',
'source_addr_long':
b'\x00\x13\xa2\x00@\xb1\xd2\xa6',
'source_addr': b'g\x03',
'options': b'\x01',
'rf_data': b'RE\x00\x02\x11\x03\xa1\x14H:33.20 T:22.00'}

Data received
127.0.0.1 - [09/Jan/2019 00:00:00] "GET /ertis/temp HTTP/1.1" 200 58
```

Consola 36: Datos - Temperatura y Humedad.

B.7. Observar un recurso

Para que el nodo nos comunique de manera periódica información sobre un recurso seleccionado utilizaremos la función de observar. Para activar esta función debemos pulsar sobre el botón azul situado a la derecha y nos aparece una ventana como en la figura B.14 en la cual indicamos el número de segundos que indican la periodicidad de envío de datos.

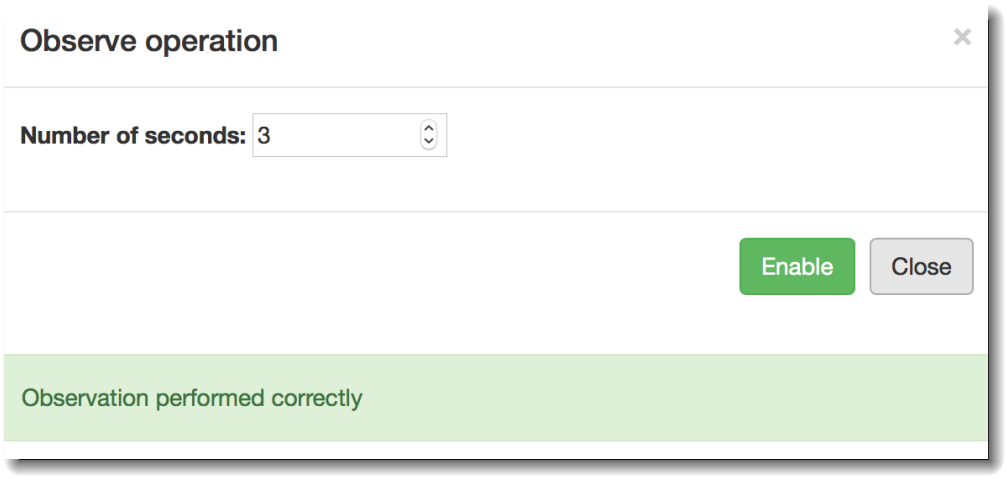


Figura B.14: Observar recurso.

En el contenido "Consola 37" observamos la petición que se realiza en la aplicación web para crear una observación de un recurso.

```
[09/Jan/2019 00:00:00] "GET /observe/ertis/temp?ma=3 HTTP/1.1" 200 0
```

Consola 37: Observar recurso.

En el contenido "Consola 38" observamos la petición que se realiza en el servidor para crear una observación de un recurso.

```
Data received
2019/01/09 00:00:00 OBSERVE:
El valor es: H:37.00 T:23.00
Recibido en main el resources temp del end point ertis
con el valor H:37.00 T:23.00
En 2019/01/09 00:00:00 se ha recibido el valor del recurso temp
del ID ertis y el valor H:37.00 T:23.00
2019/01/09 00:00:00 Packet:{'id': 'rx', 'source_addr_long':
b'\x00\x13\xa2\x00@\xb1\xd2\xa6', 'source_addr': b'g\x03', 'options':
b'\x01', 'rf_data': b'REA\xe7\xa1B\x11\x01H:37.30 T:23.00'}

Data received
2019/01/09 00:00:00 OBSERVE:
El valor es: H:37.30 T:23.00
Recibido en main el resources temp del end point ertis
con el valor H:37.30 T:23.00
En 2019/01/09 00:00:00 se ha recibido el valor del recurso temp
del ID ertis y el valor H:37.30 T:23.00
```

Consola 38: Observar recurso.

Para dejar de observar un recurso debemos pulsar de nuevo sobre el botón azul y aparecerá una ventana de confirmación como se ve en la figura B.15.

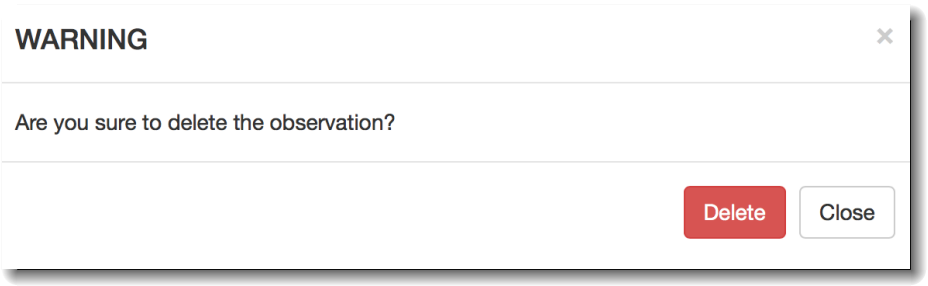


Figura B.15: Eliminar observación.

En el contenido "Consola 39" observamos la petición que se realiza en la aplicación web para dejar de observar un recurso.

```
[09/Jan/2019 00:00:00] "DELETE /observe/ertis/temp HTTP/1.1" 200 37
```

Consola 39: Eliminar observación.

En el contenido “Consola 40” observamos la petición que se realiza en el servidor para dejar de observar un recurso.

```
b'r\x00\x00\x02\x94temp!'
DATA ENVIADA
2019/01/09 00:00:00 Packet:{'id': 'rx', 'source_addr_long':
b'\x00\x13\xa2\x00@\xb1\xd2\xa6', 'source_addr': b'g\x03', 'options':
b'\x01', 'rf_data': b'a\x00\x00\x00\xb1\x01'}

Token enviado 1 y token recibido 1
127.0.0.1 - [09/Jan/2019 00:00:00]
"DELETE /observe/ertis/temp HTTP/1.1" 200 37
```

Consola 40: Eliminar observación.

B.8. Actuar sobre un recurso

Para poder mandar un valor a un recurso seleccionado utilizaremos la función de actuar. Para activar esta función debemos pulsar sobre el botón verde situado a la derecha y nos aparece una ventana como en la figura B.16 en la cual indicamos el valor que queremos mandar al recuro. En este caso como solo tenemos un actuador (LED), le mandaremos un “1” para que se encienda el LED.

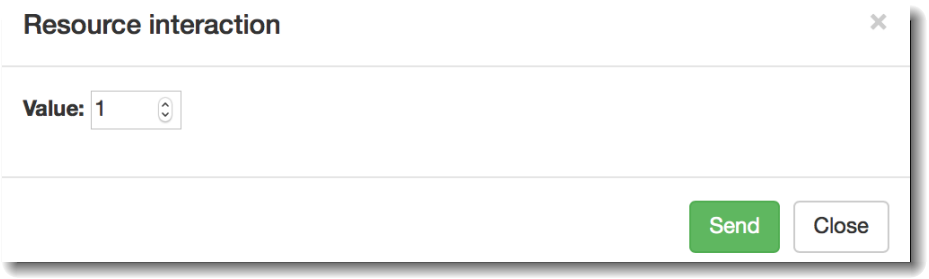


Figura B.16: Actuar sobre un recurso.

En el contenido “Consola 41” observamos la petición que se realiza en la aplicación web para actuar con un recurso.

```
[09/Jan/2019 00:00:00] "PUT /ertis/led/1 HTTP/1.1" 200 47
```

Consola 41: Actuar sobre un recurso.

En el contenido “Consola 42” observamos la petición que se realiza en el servidor para actuar con un recurso.

```
b'R\x03\x00\x02\x93led!\x14'
DATA ENVIADA
2019/01/09 00:00:00 Packet:{'id': 'rx', 'source_addr_long':
b'\x00\x13\xa2\x00@\xb1\xd2\xa6', 'source_addr': b'g\x03', 'options':
b'\x01', 'rf_data': b'RD\x00\x02\x11\x02\xa1\x14264'}

Resource changed
127.0.0.1 - - [09/Jan/2019 00:00:00] "PUT /ertis/led/1 HTTP/1.1" 200 47
```

Consola 42: Actuar sobre un recurso.

Bibliografía

[1] Manuel Díaz, Cristian Martín, and Bartolomé Rubio. State-of-the-art, challenges, and open issues in the integration of internet of things and cloud computing. *Journal of Network and Computer Applications*, 67:99–117, 2016.

[2] Cristian Martín, Manuel Díaz, and Bartolomé Rubio. Run-time deployment and management of coap resources for the internet of things. *International Journal of Distributed Sensor Networks*, 13(3):1550147717698969, 2017.

[3] Node-RED. `https://nodered.org/`, Junio 2019.

[4] Metodología. `https://www.sinnaps.com/blog-gestion-proyectos/metodologia-de-un-proyecto`, Junio 2019.

[5] Tipos de Redes. `https://sites.google.com/site/redesinalambricas3/tipos-de-redes-inalambricas`, Junio 2019.

[6] Zigbee. `http://www.icpdas.com/root/product/solutions/industrial_wireless_communication/wireless_solutions/zigbee_introduction.html`, Junio 2019.

[7] MQTT y CoAP. `https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php`, Junio 2019.

[8] Wikipedia. `http://es.wikipedia.org/`, Junio 2019.

[9] Kafka. `https://kafka.apache.org/`, Junio 2019.

[10] Node Red. `https://www.toptal.com/nodejs/programacion-visual-con-node-red-conectando-el-internet-de-las-cosas-con-facilidad/es`, Junio 2019.

[11] Moteino Mega. `https://lowpowerlab.com/shop/product/119`, Junio 2019.

[12] XBee Pro SB2. https://www.sparkfun.com/products/retired/10419, Junio 2019.

[13] Air Quality. https://www.sparkfun.com/products/14193, Junio 2019.

[14] Lux Sensor. https://www.adafruit.com/product/439, Junio 2019.

[15] Humidity Sensor. https://konnected.io/products/temperature-humidity-sensor-module-am2302-dht22, Junio 2019.

[16] Módulo Kafka Node-RED. https://flows.nodered.org/node/node-red-contrib-kafka-node, Junio 2019.

[17] Framework Tutorial. https://programarfacil.com/blog/raspberry-pi/introduccion-node-red-raspberry-pi/, Junio 2019.