Smart Home usando IoT y Chatbots

JORGE JARNE BRUN

MÁSTER EN INTERNET DE LAS COSAS. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin Máster en Internet de las Cosas

Convocatoria: Septiembre de 2018

Calificación obtenida: 8.5

Director:

Alberto Díaz Esteban

Autorización de difusión

Jorge Jarne Brun

Septiembre 2018

El/la abajo firmante, matriculado/a en el Máster en Internet de las Cosas de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Smart Home usando IoT y Chatbots", realizado durante el curso académico 2017-2018 bajo la dirección de Alberto Díaz Esteban en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Resumen en castellano

La tecnología IoT es cada día más usada en sectores muy diversos debido a su abaratamiento y flexibilidad de uso. Entre estos sectores destaca por su rápido crecimiento el de las casas inteligentes o Smart Homes, ya que suponen una mejora notable en la calidad de vida de sus huéspedes. Sin embargo, todavía hay grandes retos que suponen un problema para el despliegue de proyectos de este tipo, como la interoperabilidad entre los dispositivos IoT, el grado de dificultad de manejo para el cliente final y la administración de una gran cantidad de dispositivos con una arquitectura que sea escalable.

El objetivo de este trabajo es dar una solución unificada a los problemas anteriores. Para ello se hará uso de una plataforma que permita trabajar con todos los dispositivos conjuntamente independientemente del fabricante. Se propondrá también una arquitectura genérica de implementación de distintas funcionalidades para una Smart Home que sea reutilizable, y además, se facilitará al usuario el control esta Smart Home mediante el uso de lenguaje natural desde cualquier dispositivo móvil. Para que esto sea posible se utilizará la tecnología de desarrollo de chatbots.

Por último, se ha implementado el prototipo de una Smart Home capaz de llevar a cabo tareas como el control de las luces de una casa, la monitorización de las temperaturas o incluso el control de avisos de un sistema inteligente de alarmas. En todas estas tareas el usuario puede interactuar con la casa usando su propio móvil gracias al uso de una aplicación de mensajería instantánea.

Palabras clave

Internet de las Cosas, Aprendizaje Automático, Inteligencia Artificial, casa inteligente, chatbots, SoC, MQTT

Abstract

Nowadays, IoT technology is used in many sectors, the main causes is the low price of this technology and her facility to be implemented in any environment. In the last years, Smart Home business has growth quickly, because it suppose improve houses owners life. However, we must be able to overcome some challenges, as a the interoperability between all IoT devices, the different difficulties that users have to manage these devices and the capacity to handle a huge amount of IoT devices with a scalable architecture and low operating expense

The main objective of this project is to create a solution where previous problems were fixed in a unified way. First, we are going to use a platform that can work with all devices, no matter the device manufacturer. Furthermore, it is proposed a generic reusable architecture that implement some Smart Home functionalities, in addition, the house owners will control their Smart Home by using natural language in any smartphone. In order to achieve this, it will be used a technology able to develop a chatbot.

Finally, it has been made to Smart Home prototype which can do some tasks, for example: control of rooms lights, monitor house temperature or even perform as a security alarm. Users can interact with the Smart Home in all of these tasks by using instant messaging applications.

Keywords

Internet of Things, Machine Learning, Artificial Intelligence, Smart Home, chatbots, SoC, MQTT

Índice general

Índice de figuras

Índice de cuadros

Agradecimientos

Quiero agradecer a mi director Alberto todo el apoyo que ha puesto en mí y en el proyecto, la paciencia que ha tenido conmigo y también toda su dedicación en su labor docente.

Muchas Gracias.

Capítulo 1

Introducción

1.1. Motivación

El IoT (Internet of Things) es una tecnología que trata de interconectar cualquier dispositivo con todo aquello que le rodea, proporcionándole así inteligencia e independencia en la toma de decisiones. El crecimiento de esta tecnología ha sido muy rápido debido principalmente al abaratamiento de los dispositivos electrónicos y al gran avance en las redes de comunicaciones, las cuales permiten el intercambio de datos entre estos dispositivos. Es por ello que se espera que en el año 2020 se tengan cerca de 50 billones de dispositivos IoT desplegados [4]. Actualmente, esta tecnología se puede ver aplicada en coches, aviones, tiendas, fábricas, hogares, edificios y en aplicaciones relacionadas con el deporte, el medio ambiente o incluso con ciudades inteligentes.

Entre estos sectores, destaca por su rápido crecimiento el de las casas inteligentes o Smart Homes. Algunas de las aplicaciones más comunes en este tipo de casas son la regulación automática de la temperatura, la monitorización del gasto eléctrico y del consumo del agua, o incluso el control del encendido y apagado de las luces. A esto se añade los nuevos dispositivos que van saliendo al mercado como lavadoras o frigoríficos inteligentes que pueden ser controlados desde apps para móviles o tablets. A pesar de las grandes ventajas de tener esta tecnología IoT, su implantación desde cero en un entorno no es algo trivial. Hay que enfrentarse a diversas limitaciones como la falta de standards, que hace que cada uno de los fabricantes de estos dispositivos o nodos apuesten en muchas ocasiones por protocolos propios para interconectar únicamente sus dispositivos e incluso sus propias plataformas. Esto hace que en un entorno con muchos dispositivos desplegados se tenga una plataforma diferente para el manejo de los sensores de cada fabricante.

Además, la interacción con estos nodos y con sus plataformas en la mayoría de las ocasiones requiere de ciertos conocimientos técnicos por parte del cliente, lo que hace que se trate de una tarea complicada para un usuario normal. Algunos fabricantes crean su propia app para manejar su dispositivo usando una interfaz sencilla basada en componentes gráficos como botones, lo que permite al usuario controlar fácilmente los dispositivos de manera táctil haciendo un simple clic en la pantalla. Sin embargo, esto no soluciona el

problema comentado de tener que manejar diferentes dispositivos en un mismo entorno, ya que hay que ir navegando entre las apps de los diferentes fabricantes.

Actualmente en el mercado IoT ya existen compañías como Google o Apple [11] que ofrecen soluciones a estos problemas y que permiten hacer cualquier casa inteligente mediante la adición de dispositivos IoT. Lo que estas empresas ofrecen es una gama de productos de diferentes fabricantes de tipo plug and play como por ejemplo: termostatos, enchufes, cámaras de vigilancia, bombillas, cerraduras, etc. Estas empresas desarrollan también frameworks para móviles y tablets que permiten llevar a cabo la configuración y el control de estos dispositivos de manera centralizada desde el propio framework o incluso interactuando con los asistentes de voz instalados en los móviles, como Siri en IoS o Google Assistant en Android, de esta manera se consigue que el usuario pueda interactuar con los dispositivos usando lenguaje natural.

El objetivo de este trabajo no es crear un producto que pueda llegar hacer competencia a las grandes empresas sino proponer una solución de características parecidas que sea desarrollable con un bajo presupuesto y que sea de tipo Do It Yourself (DIY). De tal manera que permita también crear una Smart Home y donde el usuario tenga completa flexibilidad para añadir sus propios dispositivos o crear sus programas para hacer más inteligente la casa. Existen ya algunos proyectos de este tipo.

Por un lado, en el trabajo [9] se realiza el control de un sensor de humedad que está conectado a una Raspberry Pi usando el programa Node-RED. En el experimento que se realiza en dicho trabajo se crea un programa que se ejecuta en el Node-RED del cloud de IBM y que recibe a través de websockets los valores que envía la Raspberry Pi. Esta placa a su vez tiene una instancia de Node-RED ejecutándose localmente que se encarga de leer los puertos GPIOs en donde se encuentra conectado el sensor y enviarlos al cloud a través de una URL.

Respecto al proyecto [6], se lleva a cabo el control de una serie de dispositivos conectados de nuevo a una Raspbery Pi. El control se puede realizar a través de una interfaz gráfica creada con WebIOPi o mediante la interacción con un chatbot. Ambos se ejecutan en la placa y son accesibles desde cualquier dispositivo conectado a la misma red local que la Raspberry Pi. Tanto los programas encargados de manejar los dispositivos conectados a la placa como el desarrollo del chatbot se realiza en Python.

La solución que se propone tiene la ventaja de que funciona en cloud lo que significa que el manejo de los dispositivos se puede hacer desde cualquier parte del mundo, siempre y cuando se tenga acceso a internet. Además, la comunicación entre el cloud y la placa que maneja todos los dispositivos del hogar se hace a través de protocolos de bajo consumo lo que permite que todo el hardware que se despliega por la casa pueda ser alimentado con baterías y no tenga que estar conectado a la red eléctrica. Por otro lado, al igual que en [9], se utiliza un chatbot para que la interacción sea más intuitiva, sin embargo, en el proyecto comentado se emplea la librería NLTK de Python. El uso de esta librería en muchas ocasiones requiere de ciertos conocimientos en el tratamiento del lenguaje natural. Sin embargo, el chatbot de este proyecto se desarrolla en la plataforma Watson de IBM, que utiliza redes neuronales ya entrenadas de forma transparente para el desarrollador, lo que permite crear chatbots más robustos y tolerantes a fallos gramaticales de una forma mucho más sencilla. La plataforma

cuenta también con algoritmos que analizan la gramática y la semántica del texto introducido por el usuario y que son capaces de detectar sinónimos, hiperónimos, diferenciar palabras singulares y plurales, hacer análisis de sentimientos e incluso ir aprendiendo características nuevas conforme realiza la conversación con el usuario, lo cual era algo impensable hace unos años. Esto hace que el chatbot sea cada vez más inteligente y por tanto que mejore la interacción con el usuario. Además, en la solución que se propone en este proyecto, el usuario interactúa con el chatbot usando la aplicación de mensajería instantánea Telegram, por lo que no ha sido necesario el desarrollo de una app de este tipo.

Por último, en ambos trabajos [6] [9] la conexión entre la placa y los sensores es a través de los puertos de la placa, por lo que la escalabilidad está limitada al número de puertos GPIO y obliga a tener los sensores cerca de la placa y cableados. La board o placa que se ha decidido usar para el desarrollo de este trabajo cuenta con tecnologías de comunicación como Bluetooth Low Energy (BLE) o estándares como 6LoWPAN que permite conectar nuevos dispositivos a la Smart Home de manera inalámbrica con un bajo consumo de energía.

A lo largo de este trabajo se va a ver como se han ido configurado individualmente cada una de estas partes que componen el sistema propuesto y como se han interconectado todas para crear una arquitectura genérica y reutilizable.

1.2. Objetivos

El objetivo principal de este trabajo es crear el prototipo de una Smart Home en donde el usuario pueda controlar los dispositivos conectados de la casa enviando mensajes de texto a través de la aplicación de mensajería Telegram. Para ello, se propondrá una arquitectura genérica que permita lograr las siguientes metas:

- Búsqueda de una plataforma o herramienta online de alto nivel que permita el control de los dispositivos IoT y la conexión con otras plataformas como la del chatbot.
- Desarrollo de un chatbot orientado a la temática de Smart Home en la plataforma Watson de IBM.
- Comunicación hardware de los dispositivos IoT con la placa o nodo.
- Conectar todas las partes que componen el proyecto en una única plataforma

1.3. Alcance

Este proyecto parte de las ideas que aparecen en las investigaciones [6] y [9]. De la primera [6] se extrae la idea del uso de un chatbot para el control de un entorno IoT. En la segunda [9] se ve como se usa la herramienta Node-RED de IBM para el desarrollo de una aplicación completa de IoT. Este proyecto unifica las dos ideas y además hace uso de plataformas online tanto para el desarrollo del chatbot como para la unión de todos los recursos que componen el proyecto.

El trabajo comienza con el estudio del funcionamiento interno de la plataforma Watson y sus chatbots. Se crea un chatbot desde cero en esta plataforma y tras esto, se prepara el hardware que simulará la casa inteligente, lo que conlleva el desarrollo de programas en C y el uso de las librerías propias de los fabricantes de los sensores para que se pueda comunicar la placa o board con ellos. Como último paso, se configura la plataforma Node-RED para que sea capaz de comunicarse con las tres partes diferentes, con la placa de los sensores, con el chatbot desarrollado en Watson y con el usuario final a través de Telegram.

Por último, se hacen experimentos reales con el prototipo de la Smart Home y se analizan sus resultados.

1.4. Contenido

La memoria que sigue se encuentra estructurada de la siguiente manera:

- Capítulo 1 y 2: Se hace una introducción a este proyecto en español e inglés, respectivamente. Aquí se tratan temas como la motivación, los objetivos, el alcance y la estructura del proyecto.
- Capítulo 3: A lo largo de este capítulo se hace un repaso general de las tecnologías que engloban todo el proyecto. Se comienza describiendo los algoritmos que se usan actualmente para hacer funcionar a los chatbots y de las herramientas que permiten crearlos. Se estudian también las características de algunas de las plataformas usadas para desplegar un proyecto IoT. Para terminar, se tratan temas más técnicos relacionados con los protocolos usados para el intercambio de datos en dispositivos de bajo consumo y se explica los detalles del hardware usado en este proyecto.
- Capítulo 4: Hay una explicación detallada de como se ha implementado tanto la parte hardware como la de software del proyecto y de cuáles son los problemas que se han encontrado durante dicha implementación.
- Capítulo 5: Se analizan cuatro funcionalidades diferentes donde se ven situaciones concretas propias de un entorno Smart Home.
- Capítulo 6: Aquí se muestran las conclusiones del proyecto y se proponen algunas líneas de trabajo futuras para la mejora de los resultados obtenidos en este trabajo.

Capítulo 2

Introduction

2.1. Motivation

The IoT (Internet of things) is the technology that tries to interconnect any device with its surroundings in order to provide it with intelligence and autonomy when it comes to decision making. The growth of this architecture has been strong, mainly due to the reduction in the price of the electronic devices and to the great advancement of the communication networks, which enable the exchange of data among these devices. Consequently, it is expected that by 2020, around 50 billion of IoT devices will be deployed [4]. At present, this type of technology can be already seen applied to cars, planes, shops, factories, homes, buildings and in apps related to sports, the environment or even with smart cities.

The Smart Homes' sector outstands out of all these sectors due to its rapid growing. Some of the most common applications of this type of house are: the automatic regulation of temperature, the monitoring of the electric expenditures or consumption of water, or even the control of switching on or off the lights. Add to this the launching of new devices such as smart washing machines or refrigerators, which can be controlled by apps or tablets. In spite of the great advantages offered by the IoT technology, the issues found out of the installation of such technology from scratch in a new environment, are not trivial. It is mandatory to face some limitations such as the lack of standards; very often, each manufacturer follows his own protocols enabling only his own devices to be interconnected, and even using their own platforms. This requires the use of different platforms to control various devices in the same spot, because of the dissimilar handling of the sensors of each manufacturer.

In addition, the interaction with these nodes and their platforms in most cases requires certain technical knowledge on the part of the client, which makes it a complicated task for a normal user. Some manufacturers create their own app to manage their device using a simple interface based on graphic components such as buttons, which allows the user to easily control the devices in a tactile way by simply clicking on the screen. However, this does not solve the commented problem of having to manage different devices in the same environment, since you have to navigate between the apps of different manufacturers.

Currently in the IoT market there are already companies like Google or Apple [11] that

offer solutions to these problems and that allow any smart home to be made by adding IoT devices. What these companies offer is a range of products from different manufacturers of plug and play type such as: thermostats, plugs, surveillance cameras, light bulbs, locks, etc. These companies also develop frameworks for mobile phones and tablets that enable the configuration and control of these devices in a centralized way from the framework itself or even interacting with the voice assistants installed in mobile devices, such as Siri in IoS or Google Assistant in Android, in this way you get the user to interact with the devices using natural language.

The objective of this work is to create a product that can deal with large companies and propose a solution with similar characteristics, which can be developed with a low budget and type Do It Yourself (DIY). In the same way that you will allow creating a Smart Home and where the user has complete flexibility to add your own devices or create your programs to make the house more intelligent. There are already some projects of this type.

On the one hand, at work [9] the control of a humidity sensor that is connected to a Raspberry Pi using the Node-RED program was carried out. At the time the work was done, a program was created that was executed in the Node-RED node of the IBM cloud and that receives the values it sends to the Raspberry Pi through websockets. This board has an instance of Node-RED running locally that is responsible for reading the GPIO ports where the sensor is connected and sends the cloud through a URL.

Regarding the project [6], the control of a series of devices connected back to a Raspbery Pi is carried out. The control is done through a graphical interface created with WebIOPi and also through the use of a chatbot. Both are accessible to any device connected to the same local network as the Raspberry Pi. Both the programs responsible for managing the devices are connected to the platform and the development of the chatbot, they are done in Python.

The solution proposed with the advantage that it works in the cloud can mean that the management of the devices can be done anywhere in the world, as long as you have access to the internet. In addition, the communication between the cloud and the board that handles all the devices of the home is done through protocols of low consumption that allows the hardware that is deployed around the house can be powered with batteries and does not have to be connected to the electrical network. On the other hand, as in [9] a chatbot is used to make the interaction more intuitive, however, in the commented project the Python NLTK library is used. The use of this library in many occasions requires certain knowledge in the treatment of natural language. The chat of this project can be developed in the IBM Watson platform that uses neural networks and trained in a transparent way for the developer, which allows creating more robust chatbots to grammatical failures in a much simpler way. The platform also has algorithms that analyze the grammar and economics of the text introduced by the user and that are able to detect synonyms, differentiate singular and plural words, analyze feelings and even learn new features according to the user's conversation. what was unthinkable a few years ago. This makes the chatbot more and more intelligent and therefore improves the interaction with the user. In addition, in the solution proposed in this project, the user interacts with the chat using the Telegram instant messaging application, so it has not been necessary to develop an application of this type.

Finally, in both works [6] [9] the connection between the board and the sensors is through the ports of the board, so the scalability is limited to the number of ports on the board and requires having the sensors wired and close to the plate. The board that has been decided to use for the development of this work has communication technologies such as Bluetooth Low Energy (BLE) or standards such as 6LoWPAN that allows to connect new devices to the Smart Home with low power consumption.

Throughout this work we will see how each of these parts that make up the proposed system have been individually configured and how they have all been interconnected to create a generic and reusable architecture.

2.2. Aims

The main aim of this work is to create a prototype of a Smart Home in which the user can control the devices connected to the house, by just sending a text message through Telegram. To do this, it will be proposed a generic architecture that will achieve the next goals:

- To look for a high-level platform or tool that allows the control over the IoT devices and the connection to other platforms as the one of the chatbot.
- To develop a chatbot focused on the Smart Home topic, in the Watson platform (IBM).
- To enable the communication of the IoT devices with the board or node.
- To connect all the parts that compound the project in only one platform.

2.3. Scope

This project is based on the ideas reflected in the researches [6] and [9]. From the first one [6], the extracted idea is the utilization of a chatbot to control an IoT environment. In the second one [9], it is explained how to use the Node-RED tool of IBM for the development of a whole IoT application. This project unifies both ideas and uses online platforms both for the development of the chatbot and for the merging of all the resources that make up the project.

The paper begins with a study of the internal way of operation of the Watson platform and its chatbots. A new chatbot is created from scratch in this platform, afterwards a hardware is prepared. This process involves the development of C programs and the use of libraries specific to the manufacturers of the sensors to make possible the communication of the board with them. As last step, the node-RED is prepared to make possible the communication with three different parts, with the sensor board, with the chatbot developed in Watson and with the ultimate user through Telegram.

Lastly, some real experiments are carried out with the Smart Home's prototype, and the results are analysed.

2.4. Content

The report that follows obeys this structure:

- Chapter 1 and 2: It is done a introduction about this project in Spanish and English, respectively. Here there are topics as a the motivation, the aims, the scope and the project structure.
- Chapter 3: Along this chapter, there will be a general review about the technologies that encompass the whole project. It begins describing the algorithms that are currently in use for the functioning of chatbots and for the tools that are used to create them. It also includes a study of the characteristics of some of the platforms that are utilized to deploy an IoT project. Lastly, more technical topics will be treated, all in relation with the protocols of the exchange of data in low power consumption devices. Also, there is a detailed description of the hardware that has been used in this project.
- Chapter 4: Detailed explanation of how both hardware and software have been implemented and the problems found during this implementation.
- Chapter 5: Analysis of 4 different practical cases, where there can be seen specific situations of a Smart Home environment.
- Chapter 6: Conclusions of the project and proposal of some future work lines to improve the results obtained in this work.

Capítulo 3

Contexto del Proyecto

En este apartado se van a explicar las diferentes partes que componen el proyecto de la Smart Home. Se comienza hablando sobre el software utilizado y de como se ha interconectado internamente en la plataforma. El capítulo finaliza identificando cada uno de los componentes hardware.

3.1. Software

A continuación, se analizan las características de las diferentes herramientas software que son necesarias para el desarrollo de cada una de las partes que componen el prototipo de la Smart Home: una plataforma de desarrollo de chatbots, una plataforma IoT, una aplicación de mensajería instantánea y unos protocolos de comunicación que hagan posible la conexión entre los dispositivos y la plataforma IoT.

3.1.1. Chatbots

Los chatbots son cada día más nombrados en los medios de comunicación y son muy fáciles de ver implementados en cualquier tipo de página web o aplicación móvil para ayudar al usuario en cualquier tarea tal y como lo haría un servicio técnico. También los podemos ver instalados por defecto en nuestros smartphones o sistemas operativos como puede ser el caso de los asistentes Siri (Apple), Alexa (Amazon) o Google Assistant (Google) [5]. Sin embargo, un chatbot no es más que un programa informático que intenta mantener una conversación con una persona. Lo normal es que estos chatbots estén pensados para ejecutar un servicio concreto, como ayudar a planificar un viaje o realizar consultas médicas. Se puede ver como un sistema de pregunta-respuesta, en donde un especialista, del tema del que se quiere hacer el chatbot, le transfiere todo su conocimiento para que luego pueda simular ser dicho especialista durante una conversación. Actualmente el mayor reto al que se enfrenta el desarrollador de estos chatbots es que el programa sea capaz de entender las preguntas y respuestas de los humanos y de tener en cuenta el contexto de la conversación.

Aunque parece un tema muy novedoso, realmente es un concepto que ya se creó hace unos cincuenta años. En 1966, Joseph Weintraub (investigador del MIT) creo el primer chatbot llamado ELIZA [7], el cual intentaba imitar el comportamiento del psicólogo Carl Rogers. Su funcionamiento estaba basado en el uso de patrones o reglas que, aunque era un sistema básico y no conseguía mantener una conversación normal, en esa época tuvo mucho éxito ya que no era común hablar con una máquina. Otro chatbot muy famoso fue PARRY, desarrollado en 1972 por el psiquiatra Kenneth Colby en la Universidad de Standford. En este chatbot se intentaba imitar el comportamiento de un paciente con esquizofrenia. En una prueba real donde psiquiatras se ponían a hablar con PARRY, solo lograban descubrir que era una maquina el 48 % de los participantes. Hasta la fecha de hoy han ido apareciendo muchos más chatbots con características nuevas. Lo más habitual para construir cualquier tipo de chatbot hasta hace pocos años era usar técnicas como la de emparejamiento de patrones o *Pattern Matching* [2], que aunque se trata de una técnica bastante sencilla se puede llegar a obtener buenos resultados. Su funcionamiento se basa en clasificar el texto que introduce el usuario y en establecer una respuesta determinada para cada una de estas entradas. Las entradas deben coincidir exactamente con las preestablecidas en el programa. Una manera sencilla de implementar esta técnica es utilizar el lenguaje AIML o *Artificial Intelligence Modelling Language* [10] que está basado en el uso de etiquetas XML. Cuenta con caracteres especiales como * para hacer referencia a una o varias palabras. AIML también tiene etiquetas propias que permiten por ejemplo hacer llamadas recursivas dentro del programa y así reducir el número de patrones. Estos tags hacen posible crear programas fácilmente legibles, manejables y escalables.

Sin embargo, lo que ha hecho posible la creación de potentes chatbots en estos últimos años ha sido el uso de técnicas de machine learning, donde mediante el uso de redes neuronales y largos conjuntos de texto es posible detectar la intención del usuario en un contexto determinado, incluso se puede llegar a conocer sus sentimientos y emociones.

El desarrollo de estos chatbots se puede llevar a cabo mediante diversas herramientas online como Octane.ai, wit.ai, Chatfuel o Watson Conversation (desarollada por IBM). Esta última plataforma es la que se ha usado, ya que permite realizar el procesamiento de lenguaje natural sin necesidad de escribir código.

3.1.1.1. Watson Conversation

Con esta herramienta se puede desarrollar fácilmente un chatbot basado en redes neuronales, esto significa que basta con entrenar estas redes con palabras o frases de ejemplo para que vaya aprendiendo a detectar las diferentes intenciones del usuario. Además, permite diseñar la estructura del diálogo de la conversación, lo que lo hace una herramienta muy potente. Esta plataforma utiliza por debajo el software DeepQA y el framework Apache UIMA (Unstructured Information Management Architecture). Cuenta con el Botkit framework para hacer fácil la publicación de los chatbots desarrollados en Slack, Facebook, Messenger, Twilio o incluso en app propias. Cada uno de los chatbots de la plataforma [3] está compuesto por: *intents, entities* y *dialogs*.

- Intents: tratan de captar las peticiones de los usuarios. Hay que tener en cuenta las

diferentes maneras gramaticales con las que se pueden hacer referencia a las mismas situaciones. Se podría ver un *intent* como una etiqueta que representa un conjunto de propósitos. El conjunto de *intents* definirá la finalidad del chatbot.

- Entities: son objetos o términos que sirven para aportar más información dentro de un determinado *intent*. Las *entities* van a ser claves para el diseño del flujo del diálogo y para que sean detectadas deben coincidir exactamente con las definidas previamente, es por ello que es posible definir una serie de sinónimos para cada una de las *entities*. La plataforma cuenta con algunas *entities* ya creadas que facilitan por ejemplo la detección de números, de ciudades o incluso de nombres propios en una conversación. También es posible importar fácilmente gran cantidad de *entities* procedentes de bases de datos a través de archivos CSV.

- Dialog: se trata de crear el flujo de diálogo mediante la adición de nodos. Cada nodo tiene una condición basada en la detección de *intents* o valores concretos de *entities*. En caso de cumplirse la condición se respondería al usuario con el texto especificado en el nodo. Cada uno de estos nodos puede tener nodos hijos, lo que permite crear complejas estructuras de diálogo. Es recomendable tener claro desde el principio a que temática va a estar dedicado el chatbot y diseñar consecuentemente el *dialog*.

Hay varios detalles importantes a tener en cuenta respecto al *dialog*. Cuando llega un mensaje al chatbot se va comprobando una a una las condiciones que tiene cada nodo y se responde con la respuesta del nodo con el que ha habido mayor nivel de emparejamiento o coincidencia. El problema está, en que cuando los mensajes que se envían al bot son simples gramaticalmente, se activan los nodos genéricos como el de bienvenida o despedida. La solución a este problema está en usar el atributo numérico *confidence* de cada nodo para comprobar el nivel de match con el texto introducido por el usuario (el máximo es 1). De esta manera en cada nodo se puede poner como condición que el valor de *confidence* sea mayor que un límite. Otra característica importante es el uso de las variables de contexto, que permite guardar información relevante durante la conversación con el usuario, para ello es necesario usar el editor JSON de la plataforma.

Para ver más claro que son los *intents* y las *entities* se va a poner un ejemplo. Supóngase que se está desarrollando un chatbot de una web para poder resolver las dudas de los usuarios acerca de la temperatura que va a hacer en los próximos días en determinadas ciudades del país. Primero se crearía un *intent* para detectar cuando durante la conversación con el usuario, este introduce alguna frase preguntando acerca de la climatología. Este *intent* se podría llamar *#cuestiones_temperatura* y contendría frases lo más parecidas posibles a las que el usuario podría introducir: ¿Qué temperatura hace ahora en Jaca? ¿Va a hacer frio mañana en Madrid? Con este *intent* el chatbot ya sería capaz de reconocer si el usuario está preguntando por algo relacionado con la climatología. Sin embargo, el chatbot no es capaz todavía de diferenciar si se le está preguntando sobre la temperatura que hace en una ciudad u otra, o si se quiere conocer la temperatura que hace hoy o la que hará la semana que viene. Esto se debe a que el chatbot no está preparado todavía para conocer esos conceptos de tiempo y de lugar, por eso se usan las *entities*. Se crea por ejemplo para este caso las

entities llamadas *@ciudades* y *@dias*, la primera contendría los nombres de las ciudades de las que se tiene información acerca de su temperatura y la segunda tendría palabras como: hoy, mañana, ayer, pasado mañana, la semana que viene, etc. Luego, si durante una conversación el usuario de nuevo introduce la pregunta de antes: ¿Qué temperatura hace ahora en Jaca?, se activará por un lado el *intent* anterior, *#cuestiones_temperatura* y además la *entity* definida como *@ciudad* tendrá el valor *Jaca* y *@dias* será *hoy*. Jugando con las condiciones de si una *entity* tiene un valor u otro y de si un *intent* se ha activado o no, ya se podría ir creando el *dialog* para ser capaces de reconocer cada una de las posibles entradas de texto del usuario y así darle una respuesta.

3.1.2. Plataformas de desarollo IoT

Por otro lado, respecto a las plataformas online de IoT, existen diferentes aplicaciones creadas por empresas como Amazon, IBM o Microsoft y que están dedicadas a hacer de puente entre todos los dispositivos que componen un proyecto IoT. Sin embargo, antes de escoger una de estas plataformas hay que tener en cuenta ciertas características [18]:

- Confianza: es la incorporación de medidas de seguridad para que la plataforma sea tolerante a fallos y de esta manera se eviten errores como la pérdida de datos o el bloqueo de la propia plataforma.
- Customización: muchas plataformas ofrecen unos servicios ya definidos y cerrados para el cliente. Otras, sin embargo, dan flexibilidad al cliente ofreciéndole librerías, APIs o permitiéndole ampliar las funcionalidades de la plataforma usando plugins o desarrollando código.
- Escalabilidad: la capacidad de la plataforma para trabajar con un aumento de carga de datos en el caso de que crezca el número de dispositivos conectados posteriormente.
- Precio: el coste es una de las características más importantes de cualquier proyecto. En este caso, los gastos que supone el uso de las plataformas se dividen en dos partes, por un lado, el coste del host donde está la solución IoT y por otro, el coste que supone cada uno de los dispositivos conectados a este host. En este último caso lo más habitual es pagar un precio fijo por cada dispositivo conectado o pagar por cada uno de los mensajes que envía.
- Protocolos: son los distintos métodos que existen para comunicar los dispositivos con la plataforma.
- Seguridad: este es un tema muy importante ya que la información que envían los dispositivos muchas veces es sensible y por ello hay que garantizar que nadie ajeno sea capaz de obtener esta información. Su aplicación va desde garantizar seguridad en la capa de transporte (TLS) hasta la encriptación de los datos antes de almacenarlos [17].

- Support: es un servicio extra que tienen algunas compañías que permite contratar a un grupo de técnicos de la empresa para que se encarguen de solucionar problemas de seguridad y de mantener al día las actualizaciones del software del proyecto.

Actualmente, la mayoría de las plataformas tienen características muy parecidas y los detalles que las diferencian son pequeños. El precio podría ser el punto clave para decidir cual de ellas usar, sin embargo, para un proyecto tan pequeño como este, el coste del servicio es prácticamente idéntico. El criterio que se ha seguido para seleccionar la plataforma es la simplicidad para llevar a cabo la interconexión entre todos los componentes que forman parte del proyecto. Por ello se ha seleccionado la plataforma de IBM llamada Bluemix, que cuenta con la aplicación Node-RED. Esta herramienta permite llevar a cabo una programación visual mediante la creación de flujos con una interfaz de navegador web. Está construido con Node.js lo que hace que sea ligero y funcione mediante la llegada de eventos. Estas dos características hacen que se pueda ejecutar en tiempo real y con gran cantidad de dispositivos IoT simultáneamente. Su funcionamiento se basa en el uso de nodos o bloques, los cuales están interconectados entre ellos. Node-RED cuenta con miles de nodos preprogramados que permiten la comunicación entre aplicaciones internas de Bluemix, APIs, dispositivos IoT y servicios online, de manera sencilla.

3.1.3. Protocolos de comunicación

Otro tema importante por tratar es el de los protocolos empleados para el intercambio de datos entre los dispositivos IoT y los servidores que procesan la información. A continuación, se va a ver cuáles son los más usados en el mundo IoT [16] y cuáles son sus principales características.

- MQTT: protocolo para el envío de datos entre dispositivos. Funciona sobre TCP/IP con una seguridad TLS/SSL. Su arquitectura está formada por *publishers, brokers* y *subscribers*. Cada mensaje que se envía pertenece a un *topic* y los *subscribers* reciben todos los mensajes pertenecientes al *topic* al que se han subscrito.

- XMPP: es un protocolo desarrollado para mensajería instantánea y está basado en el envío de mensajes de texto. La estructura del mensaje contiene etiquetas *XML* y durante la comunicación se realiza un intercambio asíncrono de mensajes entre dos o más dispositivos utilizando una capa de seguridad *TLS/SASL*. Esta comunicación en algunas ocasiones llega a ser lenta.

- CoAP: es un protocolo parecido a *HTTP*, está basado en la arquitectura petición/respuesta donde los dispositivos pueden actuar como servidor o cliente. Los clientes pueden usar los métodos *GET, PUT, POST* y *DELETE*. Es un protocolo que funciona sobre *UDP* y cuenta con una capa de seguridad llamada *Datagram Transport Layer security* (DTLS).

- Websocket: es un protocolo pensado para la web y no para IoT. Permite una comunicación continua, en tiempo real, *full-duplex* y con baja latencia. Está basado en el

uso de *TCP* y el establecimiento de la conexión entre el cliente y el servidor se realiza mediante un *handshake*. Existe un subprotocolo de websocket llamado *Websocket Application Messaging Protocol* (WAMP), que es un sistema de mensajería basado en la publicación/suscripción.

De todos estos protocolos el que se ha usado es *MQTT* debido a que es muy ligero y usa *TCP*, lo que lo hace muy interesante para la comunicación con dispositivos de bajo coste. Para extender más la explicación realizada arriba, decir que este protocolo tiene una topología de estrella con el nodo de *broker* situado en el centro. Este nodo es el encargado de la gestión de la red y de la transmisión de los mensajes. El *broker* además es el encargado de mantener el canal de comunicación con cada uno de los clientes, por ello, estos envían periódicamente un paquete PINGREQ [1] y el *broker* les devuelve el paquete PINGRESP.

Otro concepto importante es el uso de *topics*, a través de estos es por donde se realiza la comunicación y el emisor del mensaje es el encargado de darle un nombre. La comunicación puede ser entre uno o varios emisores y uno o varios receptores. Los *topics* siguen una arquitectura jerárquica estableciendo relaciones padre-hijo.

En este protocolo también es posible activar diferentes niveles de encriptación en la comunicación. Por un lado, se puede usar la seguridad en la capa de transporte o TLS que es una evolución del certificado SSL y permite el intercambio de datos de manera privada entre dos nodos MQTT. También es posible usar credenciales como un usuario y contraseña para que tanto los subscribers como los publishers tengan que identificarse previamente a la suscripción o publicación en un *topic*. Se logra así dar seguridad a la capa de la aplicación[1] encriptando el payload del mensaje que se envía. Esta encriptación puede ser punto a punto, es decir de publisher a subcriber o solo del publisher al *broker*. Se evitan de esta manera ataques como el *eavesdropping*.

Además, el emisor del mensaje puede definir la calidad de la transmisión de su mensaje, hay tres niveles posibles y están ordenados de menor a mayor por el ancho de banda que consumen. Esto es muy importante en un entorno IoT ya que hay que minimizar el consumo eléctrico de los dispositivos porque en la mayoría de las veces se encuentran alimentados por baterías que deben de durar años cargadas:

- Nivel 0: el mensaje se entrega como mucho una vez al *broker*, de esta manera el emisor no conoce si el mensaje ha llegado correctamente o no.
- Nivel 1: el mensaje llega al menos una vez al *broker*. El emisor enviará el mensaje continuamente hasta que el *broker* le avise que lo ha recibido.
- Nivel 2: el mensaje llega solo una vez. Esto evita la duplicación de mensajes.

3.1.4. Aplicación de mensajería

Por otro lado, en este proyecto también se utiliza una aplicación de mensajería. Estas apps son muy usadas en nuestro día a día y se pueden ver instaladas en cualquier smartp-

[1]MQTT payload encryption https://www.hivemq.com/blog/mqtt-security-fundamentals-payload-encryption.

hone. Entre las más utilizadas se encuentra Telegram, que es una aplicación de mensajería instantánea que está centrada en la velocidad y en la seguridad de las comunicaciones. Hay dos configuraciones posibles de seguridad para los clientes de Telegram. Por un lado, la configuración por defecto establece un cifrado servidor-cliente basado en el protocolo de transmisión *MTProto* y que actualmente se encuentra en la versión 2.0. Este protocolo combina el cifrado simétrico AES de 256 bits, el cifrado RSA 2048 y el intercambio de claves Diffie-Hellman. La segunda configuración de seguridad se utiliza en lo que se conoce en Telegram como chats secretos y donde la encriptación es end-to-end, lo que significa que ni los servidores de Telegram son capaces de ver el contenido del mensaje, sin importar su contenido (fotos, videos, mensajes o archivos). Además, cuenta con la función de autodestruir el contenido de un mensaje en un determinado tiempo, tanto en la aplicación del emisor como en la del receptor del mensaje

Telegram es una app gratuita y abierta, por lo que pone una API a disposición de todo el mundo. Esta API permite además crear bots a sus usuarios con una interfaz HTTP. Los bots de Telegram ofrecen diferentes aplicaciones como la creación de noticias y notificaciones personalizadas, la gestión de pagos, el desarrollo de juegos o de herramientas personalizadas como traductores o pronósticos del tiempo y la conexión del bot con servicios externos. Hay dos características importantes de los bots que los diferencian de los usuarios normales en la app, por un lado, los bots tienen límite de almacenamiento en memoria por lo que los mensajes viejos pueden ser eliminados del servidor, y por otro lado, los bots no pueden iniciar una conversación, tiene que hacerlo siempre el usuario. El comienzo de esta conversación desde el cliente se puede llevar a cabo de dos maneras:

- Enviar comandos y mensajes al bot abriendo el chat con él o añadiéndolo a chats de grupos.
- Enviar peticiones desde una entrada de texto desde cualquier chat usando *@nombre_bot* junto a la consulta que se quiera realizar. De esta manera se envía contenido directamente a cualquier grupo o canal.

3.2. Hardware

Los dispositivos electrónicos que se pueden encontrar en una solución IoT de tipo DIY son muy variados, desde un smartphone hasta un sensor de temperatura, todos cumplen que directa o indirectamente están conectados a internet. Se podría clasificar estos dispositivos en dos categorías diferentes: sistemas embebidos y dispositivos *wereables* o *gadgets*. Las combinaciones entre estos dispositivos son infinitas, sin embargo los *gadgets* a diferencia de los sistemas embebidos son una solución hardware ya cerrada, por lo que para un desarrollador solo es posible realizar modificaciones a nivel de software creando sus propios programas.

Entre las placas más utilizadas [8] se encuentran Arduino, Raspberry Pi, Pinoccio, ESP8266, Particle Photon, Clodbit o Beaglebone Black. Para elegir entre una u otra placa hay que tener en cuenta tres cosas, las especificaciones de la placa, la API y el open hardware.

Las especificaciones de la placa ejercen un papel importante en el entorno IoT, por ello antes de empezar un proyecto se deben tener en cuenta las características o las especificaciones que son necesarias para hacer funcionar correctamente la solución IoT. Estas características son por ejemplo el procesador, la frecuencia de reloj, la cantidad de puertos para conectar dispositivos externos, los conversores ADC/DAC, las diferentes conectividades (Wi-Fi, Bluetooth o Ethernet), las comunicaciones posibles (I2C, UART, SPI) o el precio de la placa. Una API abierta es también muy importante para el desarrollo del proyecto. El desarrollador cuenta con amplias librerías, comunidades o foros para resolver dudas técnicas o estándares que permiten crear eficazmente las aplicaciones con un mejor uso de los recursos. Por último, el open hardware ayuda a crear un prototipo inicial y es clave a la hora de transformar un proyecto en un producto.

Por otro lado, es necesario seleccionar la plataforma software que se va a usar y a que nivel de la aplicación se va a trabajar, por ejemplo, elegir un lenguaje de programación como C, C++, Python, Java, NodeJS y .NET para las aplicaciones middleware o HTML5, CSS3, Java, android SDKs y Javascript para desarrollar los front-ends. También hay que conocer las diferentes arquitecturas que existen como Representational State Transfer (REST), Constrained Application Protocol (CoAP) o JavaScript Object Notation for Linked Data (JSON-LD).

A pesar del gran avance tanto en el software como en el hardware IoT, hay algunas restricciones que hay que tener en cuenta. Por un lado, con el rápido crecimiento de los dispositivos conectados a internet, el rango de direcciones IPv4 disponibles cada vez es más limitada. El uso del Internet Protocol IPv6 solventa este problema ya que tiene direcciones de 128 bits en comparación con los 32 bits de IPv4. Estos protocolos no fueron diseñados en principio para ser operables entre ellos por lo que la implantación de IPv6 es lenta. Sin embargo, es un cambio que se llevará a cabo y muchos de los dispositivos IoT que se utilizan en la actualidad no trabajan con IPv6. Al desplegar un nuevo proyecto hay que tener en cuenta esto y usar placas que trabajen por ejemplo con 6LoWPAN.

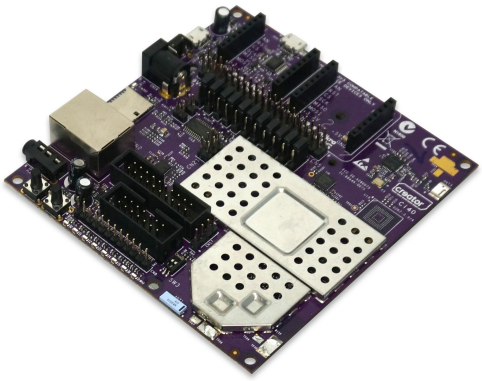
Otra característica importante es el consumo de estas placas, como se ha comentado anteriormente, en el mundo IoT la mayoría de los dispositivos están alimentados por baterías y su autonomía debe ser de meses o incluso de años ya que muchas veces las placas o sensores están situados en sitios remotos poco accesibles. Por eso será necesario comprobar las características eléctricas de los dispositivos para ver por ejemplo cuanta energía consumen cuando se encuentran en modo reposo. Existen ya interesantes proyectos con propuestas para sustituir la alimentación de estos dispositivos con energías renovables, lo que se conoce también como *Energy-Harvesting*. Aunque son proyectos en fase de desarrollo, posiblemente se vean implementados en el mercado IoT dentro de poco tiempo.

Por último, los datos que se recopilan en estos dispositivos pueden ser sensibles y hay que establecer mecanismos que aseguren la privacidad de los datos. Algunas placas o boards tienen integrados un *Trusted Platform Module* (TPM). Se trata de un chip que tiene diferentes mecanismos físicos de seguridad para que no haya manipulaciones en el hardware. Algunas de las características de este chip son la generación y almacenamiento de llaves criptográficas y el uso de una clave única RSA para procesos de autentificación.

El material que se ha utilizado en este proyecto ha sido una board de la compañía

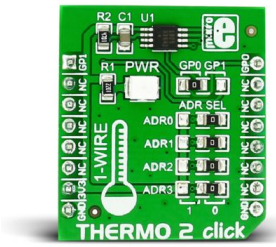
Imagination llamada Ci40 [19] que es la que aparece en la Figura 3.1(a), el relé de MikroElektronika [14] de la Figura 3.1(b), un sensor de temperatura [15] como el de la Figura 3.1(c) y el sensor de movimiento [13] de la Figura 3.1(d). La placa tiene el SOC cXT200 y cuenta con WiFi, bluetooth, 6LoWPAN y conexión Ethernet, además tiene dos conectores micro-USB, un chip TPM y dos interfaces mikroBUS [12]. Este último tipo de puerto es muy útil para conectar sensores que tengan el mismo tipo de conexión, ya que su instalación es muy fácil y permiten el uso de los protocolos *I2C, UART y SPI*. Otra característica importante es que el bluetooth que tiene la placa es *low energy*, lo que resulta muy interesante para conectar remotamente a la placa más sensores que tengan esta tecnología.

Esta placa, aunque cuenta con un sistema operativo, carece de entorno gráfico que haga fácil su manejo. Para la creación de los programas hay que establecer una conexión con la placa desde otro ordenador y mediante el uso de terminales, crear, editar y compilar los programas. Esto, según la complejidad del programa, puede hacerse una tarea pesada, sin embargo, existe un software para las placas de la marca *Imagination* llamada *Creator Dev* que permite desarrollar las aplicaciones desde cualquier ordenador y una vez ya compiladas, transferirlas a la placa.

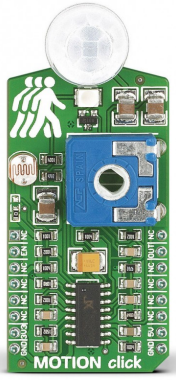


(a) Board Ci40

(b) Sensor Relay Click



(c) Sensor Thermo 2 Click



(d) Sensor Motion Click

Figura 3.1: Hardware utilizado en el proyecto

Capítulo 4

Diseño e Implementación de una Smart Home

En la primera parte del capítulo se analiza, sin entrar en explicaciones técnicas, el esquema general en el que se basa el proyecto presentado en esta memoria. En la segunda parte se describen cada uno de los módulos que aparecen en el esquema y que componen el proyecto.

4.1. Arquitectura Genérica

La misión de este apartado es tener una idea general de qué sucede internamente entre los módulos que componen el proyecto desde el momento en que el usuario escribe algo en la aplicación de Telegram hasta que se lleva a cabo una acción física en alguno de los sensores o actuadores de la placa que se encuentra en la casa.

Los módulos que hacen posible esto son los ya comentados, una plataforma donde se encuentra un chatbot a la espera de peticiones, una placa SoC que tiene una serie de sensores conectados, una aplicación de mensajería instantánea que será la puerta de entrada y salida para toda la información que pasa por el usuario y una plataforma IoT que se va a encargar de mediar entre los módulos anteriores.

Para ver como trabajan conjuntamente estos módulos se va a ver un ejemplo paso a paso de como sería el intercambio de mensajes entre ellos para llevar a cabo el control del encendido y apagado de una bombilla conectada a la placa. Por ejemplo, para el caso de que el usuario escribiera en Telegram "Apaga la luz de la cocina" e instantáneamente el relé conectado a la placa se abriera para apagar la luz.

El ejemplo paso a paso se encuentra a continuación, Figura 4.1:

- Paso 1: el usuario abre su aplicación de Telegram en el móvil y escribe en el cuadro de texto del chat alguna frase como la comentada "Apaga la luz de la cocina". La idea es que esta cadena de texto llegue al chatbot para que lo procese y así traducir lo que quiere decir el usuario a un lenguaje entendible por la máquina, sin embargo, el texto

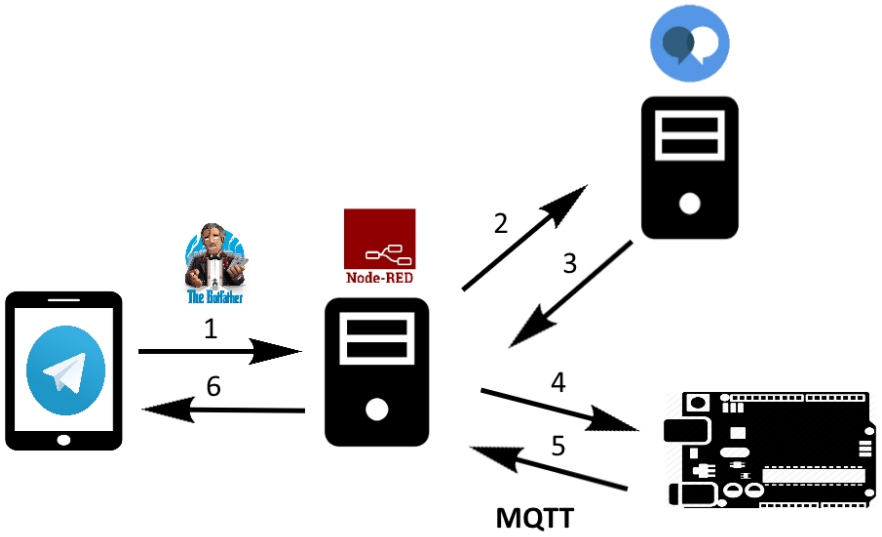


Figura 4.1: Esquema general del proyecto

no puede llegar directamente al chatbot, por lo que el texto escrito en Telegram llega antes al módulo mediador que es la plataforma IoT.

- Paso 2: como la plataforma Node-RED no es capaz de procesar el texto procedente de Telegram, lo reenvía directamente a la plataforma donde se encuentra el chatbot y este genera una respuesta.
- Paso 3: la respuesta es enviada de vuelta a la plataforma IoT, junto a la respuesta también se envía información acerca de que nodos del *dialog* del chatbot se han activado para generar dicha respuesta. En la plataforma IoT se analizan estos nodos y asi según se haya activado un nodo u otro se puede saber si el usuario quiere encender o apagar la luz, si quiere hacerlo en la bombilla del salón o en la del baño, etc.
- Paso 4: tras conocer el tipo de acción que desea realizar el usuario, se genera un mensaje con las ordenes correspondientes de apagado o encendido y se envía a la placa SoC. La placa tras recibir el mensaje solo debe abrir o cerrar el relé según el contenido del mensaje que haya recibido.
- Paso 5: como en toda comunicación, puede darse el caso en el que el mensaje que se envia por MQTT en el paso anterior falle, por eso es necesario que la placa le informe a la plataforma IoT si ha llegado correctamente el mensaje y si además la operación solicitada ha podido realizarse.
- Paso 6: si todo ha ido correctamente, la plataforma IoT envía al chat de Telegram la respuesta generada por el chatbot en el Paso 3, algo como por ejemplo *la luz se ha encendido correctamente*.

Todas estas conexiones entre los módulos se pueden ver traducidas a la creación de un flow como el siguiente en la plataforma IoT Node-RED, Figura 4.2.

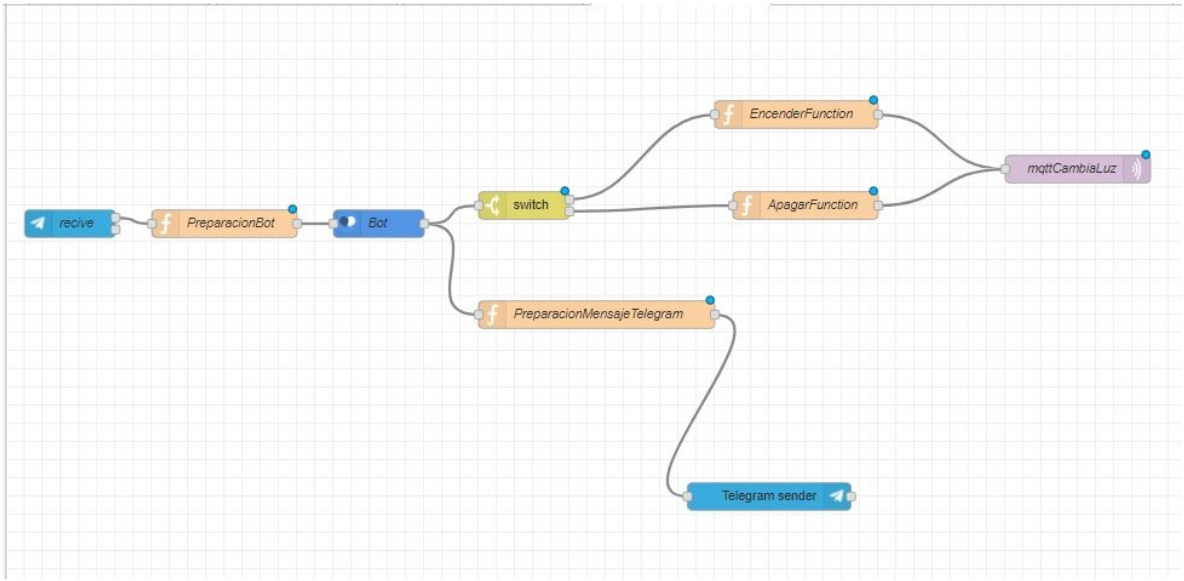


Figura 4.2: Flow en la plataforma Node-RED

4.2. Módulos

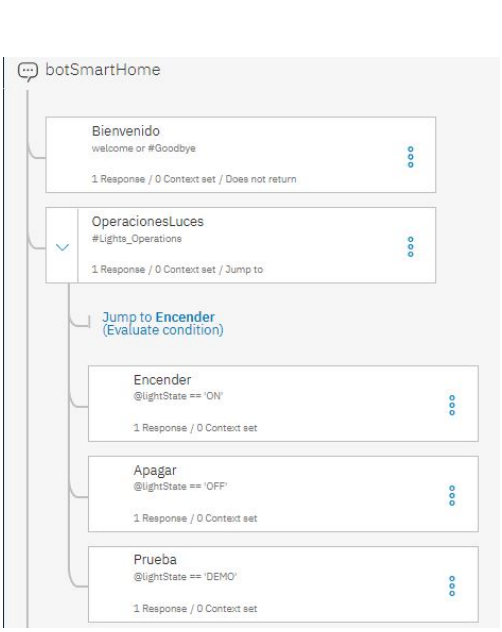
A continuación se va a describir cada una de las piezas que componen este proyecto y que están interconectadas dentro de la aplicación de Node-RED. Es importante analizar en profundidad como trabajan internamente estas herramientas para entender su funcionamiento.

4.2.1. Gestión del diálogo

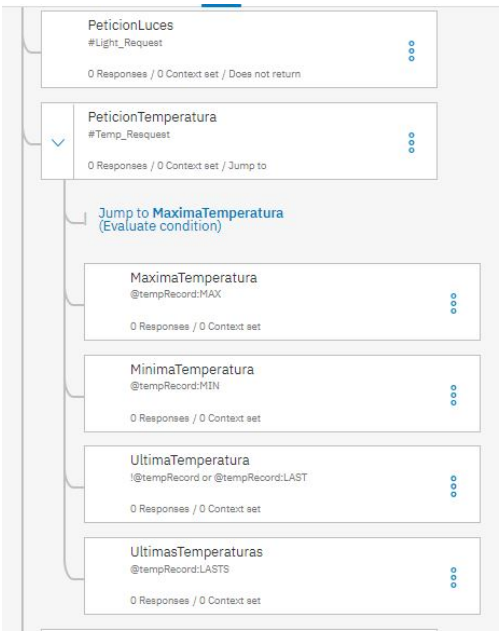
Lo primero que hay que hacer es crear un nuevo workspace en el Watson Assistant con un nombre, una descripción y el idioma con el que interactuará el chatbot. Tras esto, hay que implementar cada una de las partes que componen el chatbot: *intents*, *entities* y el *dialog*.

Los *intents* principales que se han implementado tratan de detectar cuando el usuario quiere hacer alguna operación sobre los sensores o realizar alguna consulta sobre la información que han recogido. Se han creado un total de cuatro *intents*:

- Light_Operations: utilizado para detectar cuando se quiere realizar algún tipo de acción sobre las luces.
- Light_Request: se usa cuando el usuario quiere conocer el estado de las luces.



(a) Estructura del diálogo I



(b) Estructura del diálogo II

Figura 4.3: Desarrollo del diálogo en la plataforma Watson Conversation

- Alarm_Request: sirve para consultar acerca del estado de la alarma y saber cuál ha sido el aviso más reciente.
- Temp_Request: usado para solicitar información acerca de la temperatura de una habitación.

Además de los anteriores, se han añadido también dos *intents* más para saludar y ayudar al usuario a la hora de interactuar con el chatbot. Por otro lado, las *entities* se van a usar para detectar que tipo de operación o información quiere el usuario:

- lightState: detecta cuando se quiere apagar o encender las luces.
- tempRecord: tiene cuatro valores y son usados para detectar cuando el usuario quiere conocer la última temperatura recogida, la temperatura mínima, la máxima o un listado con las últimas temperaturas medidas.

Con estos *entities* y *intents* se estructura el *dialog* como se ve en la Figura 4.3. Los nodos llamados *PeticionTemperatura* y *OperacionesLuces* son los únicos que cuentan con nodos hijos, y como su nombre indica, están hechos para tratar las peticiones acerca de la temperatura y para realizar operaciones sobre las luces, respectivamente. Cada nodo hijo tiene asignado un valor de la *entity* correspondiente y de esta manera, se realiza una operación u otra según requiera el usuario.

Dos parámetros muy importantes del entorno de Watson Conversation son el *workspaceID* que tiene cada uno de los chatbots (se consulta en Watson seleccionando la opción *view*

details estando en la pestaña *workspaces*) y el usuario/contraseña del servicio contratado (se puede consultar en la pestaña *deploy* que hay dentro de cada *workspace*).

La integración del chatbot en Node-RED es sencilla debido a que esta plataforma tiene ya implementado nodos para poder añadir cualquier aplicación de Bluemix. Para el Watson Assistant tiene dos nodos, uno llamado *conversation* y otro *conversation workspace manager*. Este último está pensado para poder crear y editar chatbots desde el Node-RED, lo cual no es necesario en este proyecto. El nodo *conversation* es el que va a permitir interactuar con el chatbot que se haya creado en el workspace de Watson y su configuración, Figura 4.4, requiere únicamente los parámetros detallados en el párrafo anterior.

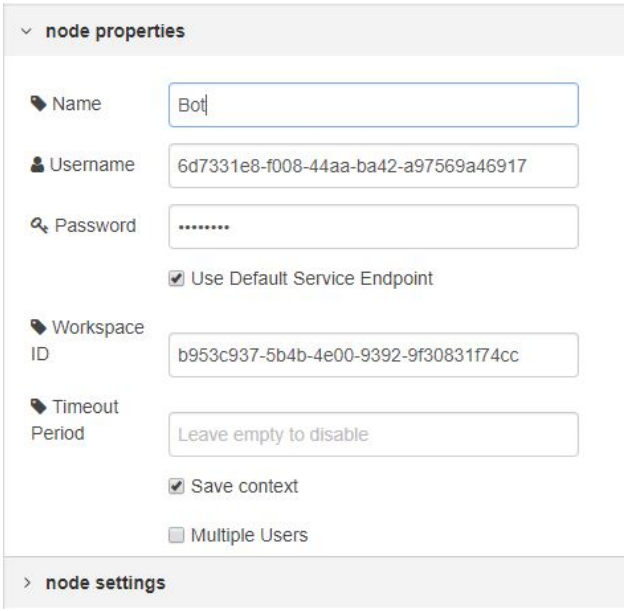


Figura 4.4: configuración del nodo conversation en Node-RED

Los campos que tienen cada uno de los mensajes que entran al nodo son muy variados y aunque la mayoría son opcionales es importante conocer para que sirve cada uno de ellos.

- msg.payload: contiene el mensaje que se quiere enviar al chatbot. En formato String.
- msg.users: (opcional): es un identificador único para cada usuario. Esto permite tener un contexto de conversación para cada uno de los diferentes usuarios. En formato String.
- msg.params.workspace_id: identificador único para diferenciar los workspaces. Este parámetro puede ser configurado también directamente en el nodo. Formato String.
- msg.params.tiemout (opcional): define el intervalo de tiempo que trascurre entre las peticiones que se hacen a la plataforma Watson (medido en milisegundos). Esta opción se puede configurar también directamente en el nodo. Al dejar el campo en blanco o con 0 se desactiva.

- msg.params.context (opcional): contiene información acerca del estado de la conversación. Esto sobrescribe cualquier contexto guardado en el nodo. Está en formato JSON.
- msg.params.alternate_intents (opcional): se activa si se quiere recibir más de un *intent*. Por defecto está en *false*. Al ponerlo a *true* se devolverán todos los *intents* con los que se ha hecho match. Es muy útil para los casos en que el parámetro *confidence* no tenga valores altos y se quieran analizar todos los *intents*.
- msg.params.output_learning: al ponerlo a *true* se inhabilita el registro de solicitudes.
- msg.additional_context (opcional): es información adicional que se añade al objeto del contexto anterior. En formato objeto.
- msg.params.username: al rellenarlo se usa como el nombre de usuario para autentificarse en los servicios de conversación. Este parámetro y los siguientes se pueden rellenar directamente en el nodo.
- msg.params.password: si se completa se usará como la contraseña para la autentificación junto al nombre de usuario anterior.
- msg.params.endpoint: se usa como la URL donde está localizado el servicio de conversacion. Si no se rellena, se usa la URL de Watson por defecto.

Todos estos campos se pueden ver más en profundidad en la API de watson [1].

Por otro lado, el mensaje de salida del nodo es más simple y está en formato JSON dentro del campo *msg.payload*. A continuación, se analiza uno de estos mensajes y sus campos:

- Intents: tiene la información del nombre del intent y el nivel de *confidence* con el que ha hecho match. En caso de haber activado el campo *msg.params.alternate_intents*, se recibirá un array de intents.
- Entities: contiene en un array las entities que se han detectado y para cada una, se detalla su nombre, el valor y de nuevo el nivel de *confidence*.
- Input: es el texto de entrada que ha recibido Watson.
- Output: la salida tiene la respuesta procedente del chatbot y también tiene el nombre de los nodos del diálogo que se han activado. Además, en la salida hay un campo dedicado a los mensajes de log.
- Context: tiene información sobre el contexto de la conversación. Está compuesto por el campo *conversation_id* y un conjunto de información sobre el *dialog*, como por ejemplo, *_node_output_map*, donde aparece el nombre de cada uno de los nodos que componen el *dialog*.

En la Figura 4.5 se puede ver un ejemplo de estos campos dentro del mensaje.

[1] API Watson Conversation https://www.ibm.com/watson/developercloud/conversation/api/v1/curl.html?curl.

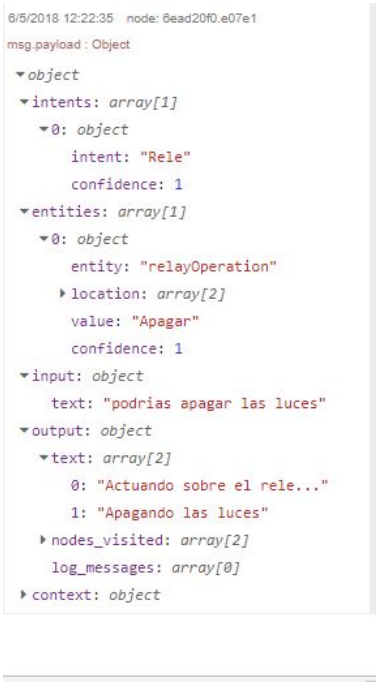


Figura 4.5: Estructura de mensaje JSON

4.2.2. Aplicación de mensajería para smartphones

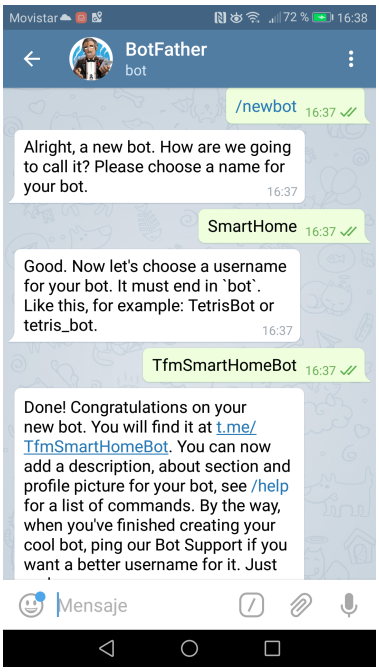
En esta sección se va a crear el bot de Telegram que hace de puente entre la app y la plataforma Node-RED. Su creación se hace desde la misma app de Telegram interactuando con un bot de la compañía llamado *BotFather*. Al enviarle el comando */newbot* y tras darle nombre al bot que se va a crear, el *BotFather* devuelve un token único que se utilizará posteriormente en Node-RED para poder referenciar al bot. Este proceso lo podemos ver en la Figura 4.6.

En Node-RED es necesario la instalación de un paquete llamado *node-red-contrib-telegrambot* que permite la integración de Telegram en la plataforma. Tras instalarlo se puede ver un conjunto nuevo de nodos como aparece en la Figura 4.7(a).

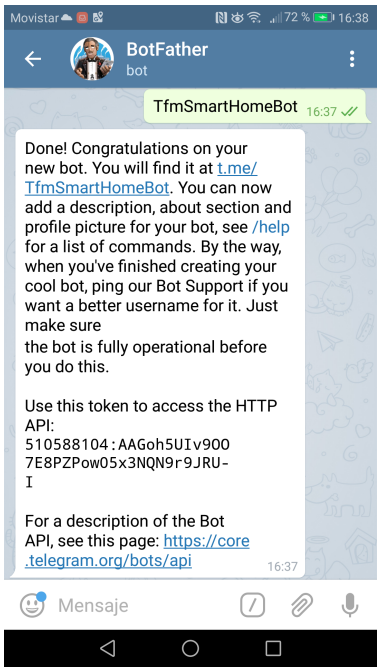
La configuración para cualquiera de estos nodos es muy parecida, Figura 4.7(b), hay que poner el nombre que se le ha puesto con anterioridad al bot y el token que se ha obtenido. El resto de los campos para rellenar, *Users*, *ChatIds* y *Server URL*, son opcionales.

Los nodos que se han usado en este proyecto son el *receiver* y el *sender*. El primero de ellos es de entrada y genera mensajes con los siguientes campos:

- msg.payload.content: tiene el contenido del mensaje.
- msg.payload.type: tipo de mensaje.
- msg.payload.messageId: el id del mensaje recibido.
- msg.payload.chatId: el id del chat del cual proviene el mensaje.

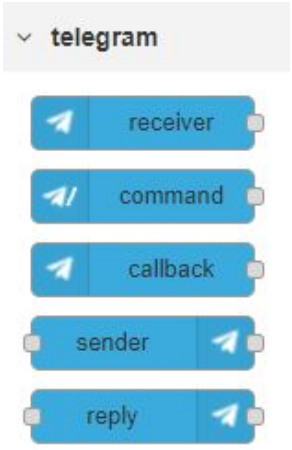


(a) Asignación de nombre al bot

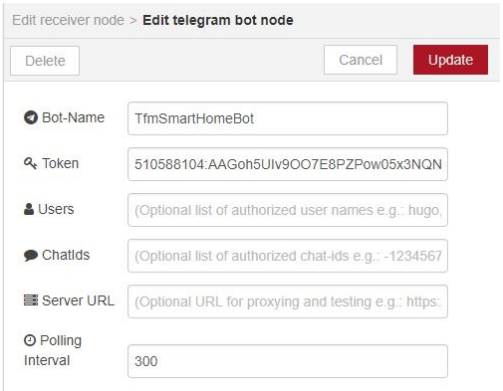


(b) Obtención de Token

Figura 4.6: Proceso de creación del bot en la aplicación de Telegram



(a) Nodos disponibles tras instalar el paquete de Telegram



(b) Campos de texto del nodo sender

Figura 4.7: Telegram en Node-RED

El nodo tiene dos salidas, el mensaje saldrá por una o por otra según su procedencia, si viene de un usuario autorizado saldrá por la primera salida, en caso contrario, por la segunda. En este caso como no se ha rellenado el campo de *Users* ni de *ChatIds*, el mensaje aparecerá siempre por la primera salida.

El nodo *sender* para que funcione correctamente debe recibir mensajes con al menos los campos *content, type y chatId* dentro del *payload*. Para ello es necesario añadir un nodo de función previamente al *sender* y así poder rellenar los campos que se requieren. El código de esta función se puede ver en la Figura 4.8.

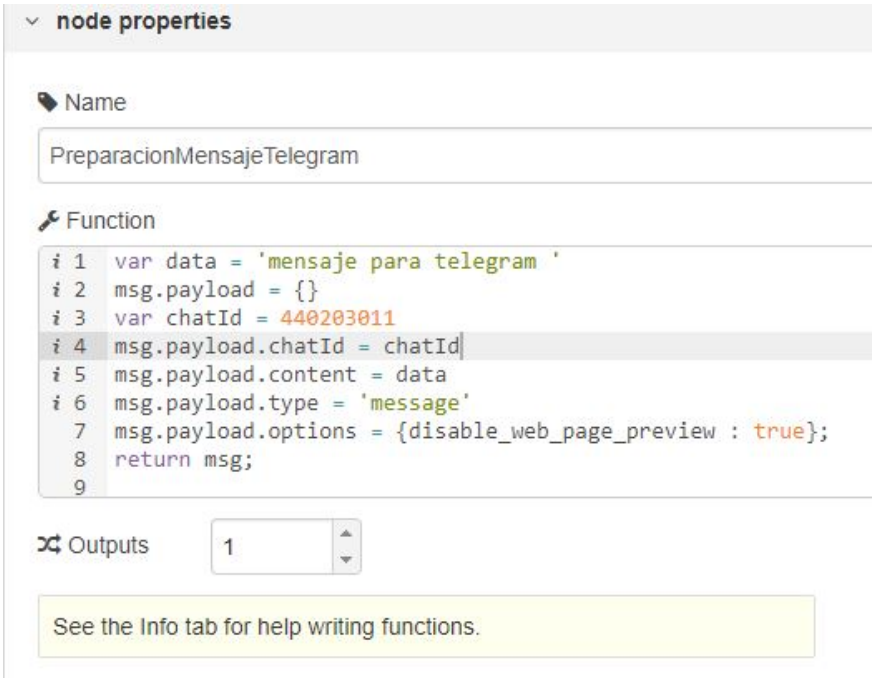
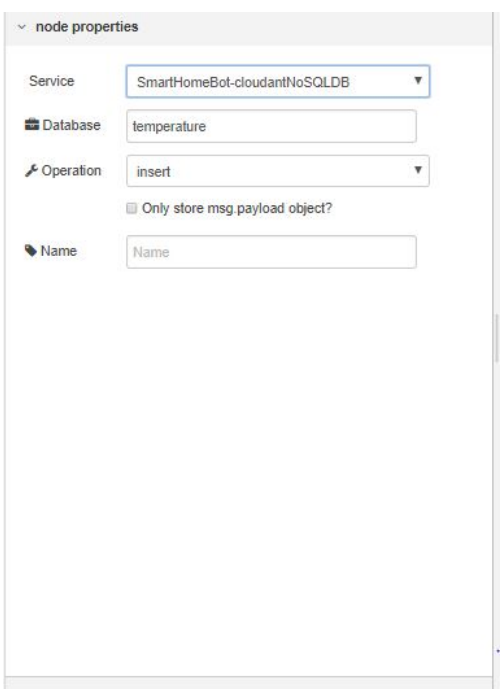


Figura 4.8: Función para crear el cuerpo del mensaje para el nodo Sender de Telegram



(a) Nodo para insertar datos en una tabla de Cloudant

(b) Función para adaptar el mensaje a formato JSON

Figura 4.9: Cloudant en Node-RED

4.2.3. Administración de la información

En este proyecto se ha usado también una base de datos NoSQL basada en servicios llamada Cloudant y que pertenece a la plataforma de IBM. Se va a usar para guardar toda la información procedente de los sensores. Para crear una base de datos nueva en Cloudant solo hay que seleccionar la opción *Create a Database* y asignarle un nombre. Se pueden añadir nuevos valores a la base de datos manualmente o también mediante llamadas externas a la API. En este caso no ha sido necesario usar ninguna de las anteriores ya que Node-RED cuenta con un nodo que es capaz de añadir o eliminar directamente datos almacenados en Cloudant. Para que este nodo almacene el valor que se le pasa en el flow es necesario escribir el nombre de la base de datos, Figura 4.9(a) y además hay que crear una función previa de preparación que adapte el valor a un formato JSON, Figura 4.9(b).

Se ha normalizado la estructura de los datos que se almacenan tanto de los sensores como la de los actuadores usando los siguientes campos:

- Id: un identificador diferente para cada uno de los valores que se almacenan.
- Valor/Estado: es el dato que realmente se obtiene del sensor o que representa el estado de un actuador en un momento determinado.

- Time-Stamp: se guarda la hora y fecha en la que se ha guardado el valor en la base de datos.

Otra función importante es hacer consultas a la base de datos. Para poder hacer consultas desde fuera de la plataforma de Cloudant es necesario crear un *Search Index function* como se ve en la Figura 4.10, donde es necesario definir dentro de la función que parámetro del JSON se va a usar para indexar. Esta función está escrita en JavaScript y se ejecuta para cada uno de los documentos almacenados.

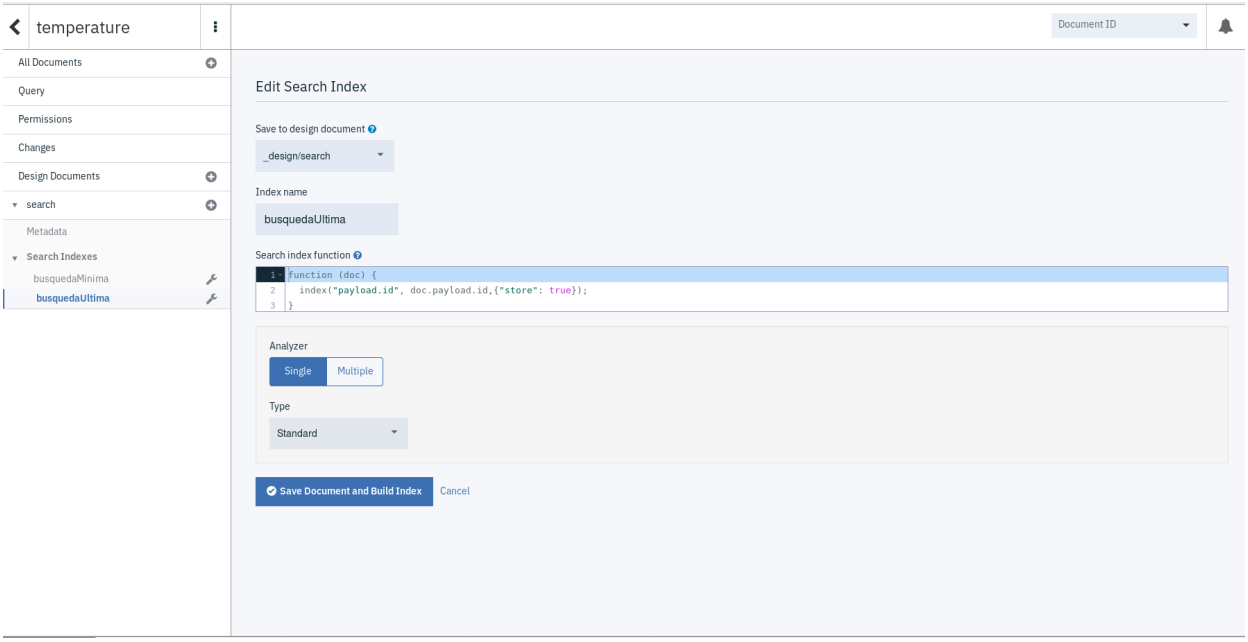


Figura 4.10: Función de consulta Javascript en la plataforma Cloudant

Para poder realizar consultas desde Node-RED es necesario referenciar, además del nombre de la base de datos, el nombre del *Search Index* que se ha creado. Este nodo recibe una consulta y devuelve un mensaje JSON con los documentos que cumplen la consulta. De nuevo, hay que añadir un nodo previo a este que se encargue de crear un mensaje JSON de consulta. El mensaje está compuesto por tres parámetros:

- Query: la consulta que se hace sobre la *Search Index function*. La consulta por defecto *:* devuelve todos los documentos almacenados.
- Limit: el número máximo de valores a devolver.
- Sort: la palabra clave del JSON que se va a usar para ordenar la búsqueda.

4.2.4. Gestión de la comunicación de los dispositivos

Para comenzar a trabajar con la placa es necesario realizar su configuración básica, esta se puede hacer usando las terminales SSH y UART, o de manera gráfica accediendo directamente a su IP desde el navegador web de cualquier PC de la red local. Esta opción es la más cómoda y para llevarla a cabo es necesario conectar un cable ethernet procedente de un router (con DHCP) a la placa y así se le asignará automáticamente una IP. La IP que le ha asignado se puede conocer entrando a la URL del router [2]. Tras conocer su dirección IP se accede a la interfaz de la placa escribiendo la dirección obtenida en el navegador web, de esta manera se obtiene una pantalla como la de la Figura 4.11. En la pantalla de configuración *Wireless* hay que escanear las redes WiFi e introducir las credenciales de la red correspondiente. Una vez hecho este paso ya se puede desconectar el cable ethernet de la placa.

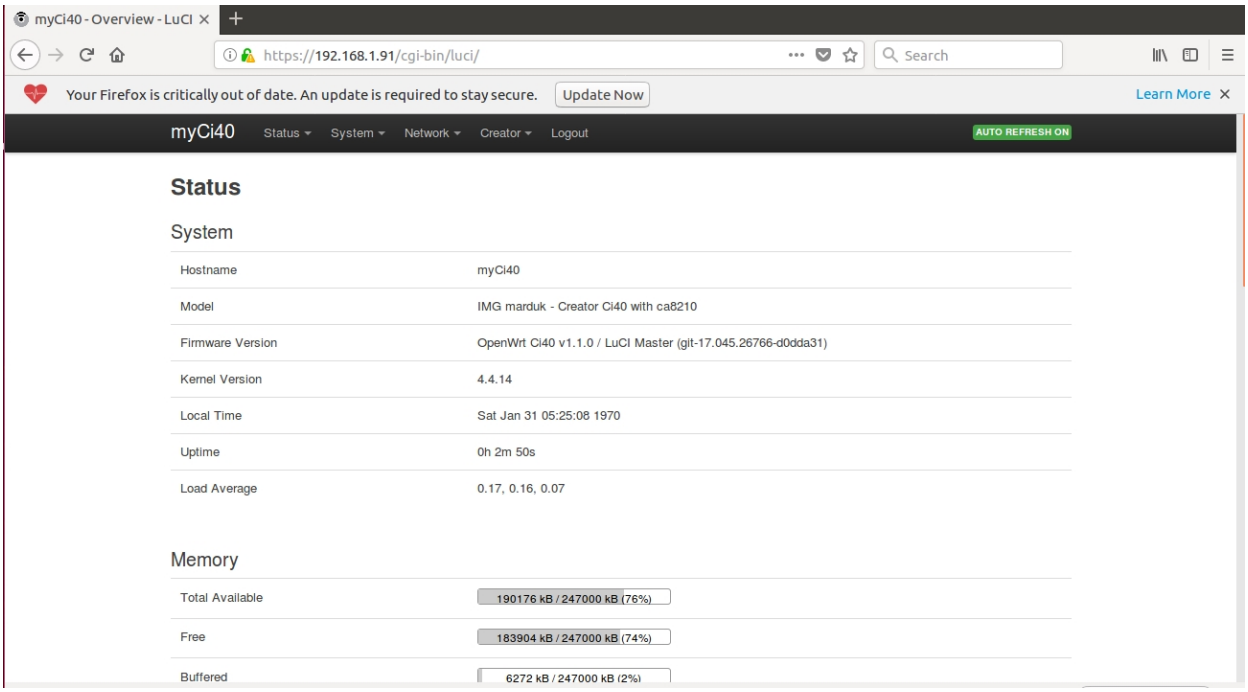


Figura 4.11: Configuración de la board Ci40 a través de la interfaz gráfica

Para poder llevar a cabo otro tipo de configuraciones, instalaciones, ejecuciones de programa o cualquier otro tipo de operación es necesario usar SSH y así poder acceder a la terminal de la placa tal y como se ve en la Figura 4.12.

La comunicación en este caso entre la plataforma Node-RED y la placa se hace con el protocolo MQTT. La board Ci40 hace de cliente, por ello es necesario instalar en ella el programa *mosquitto* que permite usar este protocolo y que está compuesto por las librerías: *mosquitto, mosquitto-client y libmosquitto.*

[2]Router web GUI `http://192.168.1.1/`.

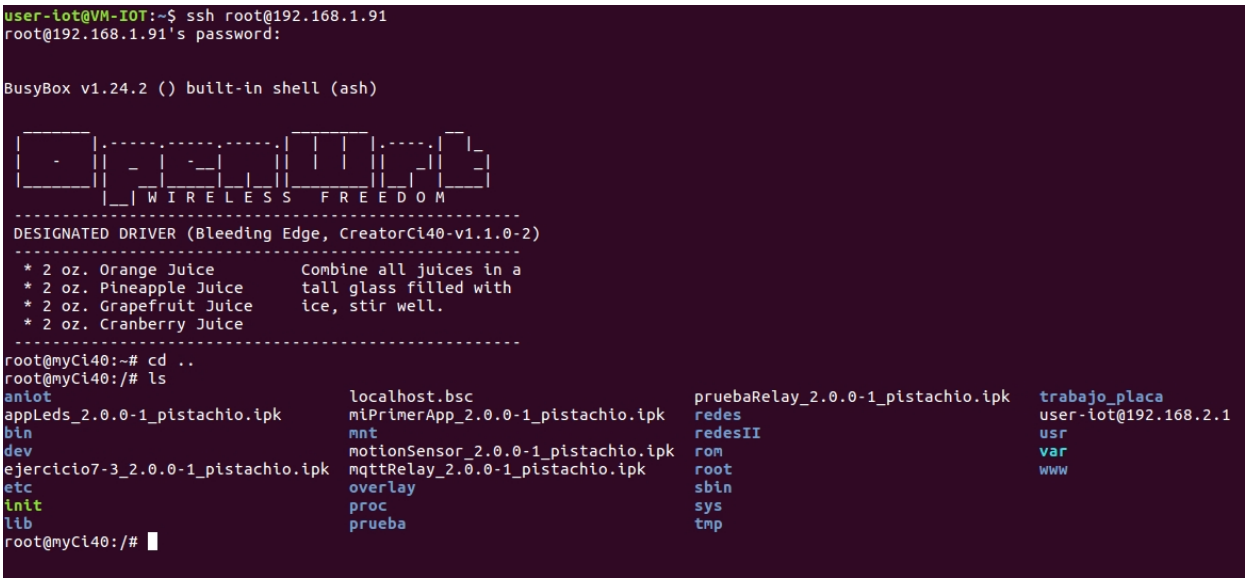


Figura 4.12: Establecimiento de la conexión SSH con la board Ci40

El lanzamiento del programa que permite subscribirse a un *topic* se hace con el siguiente comando: *mosquitto_ sub -h <ip>-p 1883 -t <nombre_ topic>*. El puerto 1883 no sería necesario especificarlo ya que es por donde por defecto escuchan los *brokers*. La <ip>es la dirección del *broker*. Es posible tener un *broker* funcionando en un servidor propio instalando previamente los paquetes correspondientes de *mosquitto* o una solución aún más sencilla es usar el *broker* propio del proyecto *mosquitto* [3].

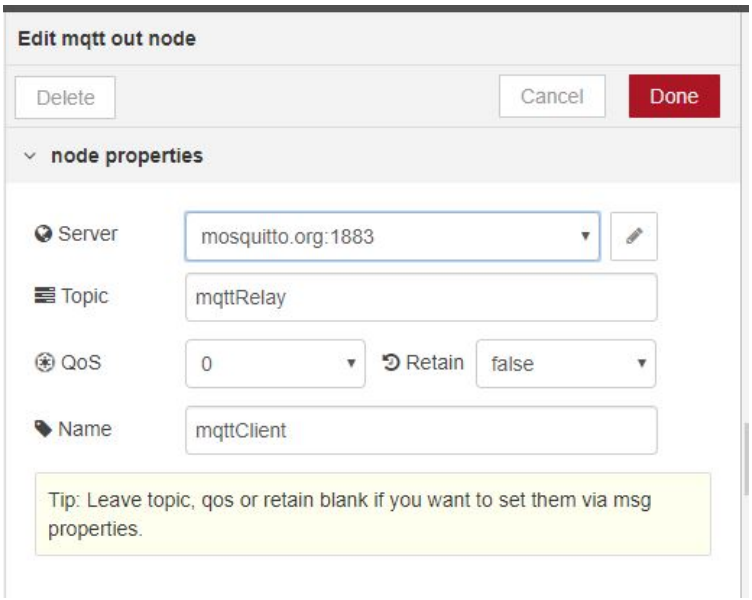
La comunicación con Node-RED es sencilla gracias a su nodo mqtt, tanto para enviar (publisher) como para recibir datos (subscriber). Por ejemplo, para el nodo de envío se tienen los campos de la Figura 4.13(a) y para la configuración del *broker* los campos que aparecen en la Figura 4.13(b).

El funcionamiento de este nodo es simple, se publica en el *topic* del Server la información que se encuentra guardada en el *msg.payload* del mensaje que llega al nodo.

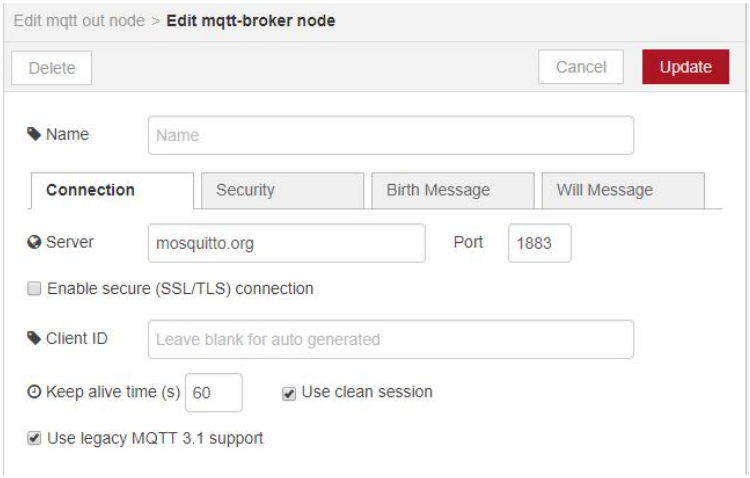
Con esta configuración, la comunicación Node-RED y board Ci40 ya es posible. Sin embargo, es necesario desarrollar una aplicación para la board que se encargue de interactuar con los sensores que tiene conectados en sus puertos mikroBUS en función del contenido de los mensajes que le lleguen de Node-RED. La solución más cómoda para realizar la aplicación es usar una programación cruzada de tal manera que desde el PC se crea un programa para la arquitectura MIPS que tiene la placa. El entorno de trabajo usado en el PC se llama *Creator Dev* y permite crear paquetes *.ipk* que posteriormente se envían por SSH a la placa y se instalan. Además, este entorno cuenta con cientos de librerías ya desarrolladas que hacen fácil la creación de aplicaciones en C.

El programa que se ejecuta en la placa necesita por un lado, ser capaz de comunicarse

[3]Broker web site http:\mosquitto.org.



(a) Ajustes del nodo de salida MQTT



(b) Configuración del servidor broker

Figura 4.13: Ajustes de la comunicación MQTT en en Node-Red

con el sensor que tiene conectado, y por otro lado, estar continuamente escuchando al *broker* para comprobar si llegan mensajes MQTT. Lo primero se soluciona usando las librerías que ya tiene implementadas el entorno *Creator* para comunicarse con el puerto mikroBUS. Lo segundo, debido a que no existe una librería oficial de *mosquitto*, requiere usar la función *system()*, propia de C, que permite ejecutar ordenes en la consola de comandos del sistema.

Capítulo 5

Implementación de Funcionalidades Concretas

En este apartado se desarrollan diferentes casos de uso aplicables a una Smart Home. Se va a explicar como se ha llevado a cabo la implementación técnica de cada caso y como se han interconectado los diferentes módulos explicados en el capítulo 4.

5.1. Consulta del estado actual de un sensor

En este primer modo de funcionamiento se hace una consulta del estado a una bombilla que es controlada por el relé de la Figura 2.1(b). Este caso es muy útil por ejemplo para las situaciones en que una persona quiere saber si se ha dejado una luz encendida en su casa.

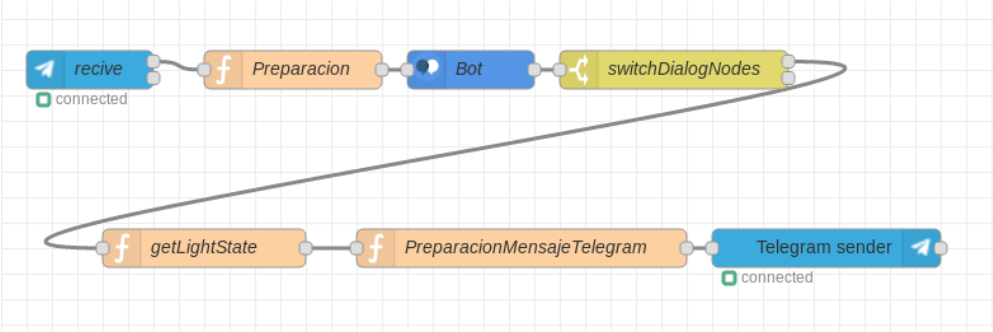


Figura 5.1: Flow en la plataforma Node-RED

El esquema del flow de Node-RED se puede ver en la Figura 5.1, comienza con el nodo de Telegram que recibe el mensaje escrito por el usuario en la app de Telegram, el mensaje se pasa al chatbot de Watson y luego se hace un filtro con el nodo *switchDialogNode* para hacer una acción u otra según el nodo del diálogo que se haya ejecutado, de esta manera se puede ver que intención tiene el usuario. En este caso, el nodo es el llamado *PeticionLuces* y aparece en el flujo del diálogo de la Figura 4.3(b). A continuación se lee una variable que

se llama *light_state* que almacena 0 o 1 según si la luz se encuentra encendida o apagada, respectivamente y se crea un mensaje que informa del estado al usuario. El mensaje debe pasar por un nodo de adaptación para que se pueda enviar finalmente por Telegram.

Figura 5.2: Consulta del estado desde Telegram

En la captura de pantalla de la aplicación de Telegram de la Figura 5.2 se ve un ejemplo de respuesta a la consulta del estado de la luz.

En este apartado se han usado las variables de almacenamiento con las que cuenta la plataforma Node-RED para guardar el estado de la luz (0 o 1) en lugar de usar una base de datos externa a la plataforma. Estas variables permiten guardar internamente cualquier tipo de información o dato en formato clave-valor. Hay una gran diferencia de tiempo según el tipo de tecnología que se use para guardar el estado de la luz, para demostrarlo se ha realizado el mismo flow sustituyendo la función que consulta el valor de la variable por otra que hace una consulta a una base de datos almacenada en la plataforma Cloudant. Este nuevo flow es el de la Figura 5.3. Se ha usado un nodo de Node-RED llamado *Interval length* que permite medir el tiempo que tarda cada uno de los métodos en obtener el valor del estado de la luz. Para ver mejor la comparativa de tiempos se ha hecho el Cuadro 5.1 donde se muestran los datos recogidos de los dos métodos en 5 consultas.

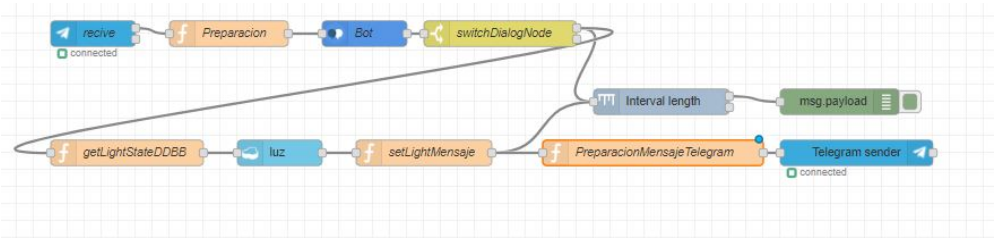


Figura 5.3: Flow en Node-RED para la medida de tiempos

Consulta	Node-RED (ms)	Cloudant (ms)
1	0,441268	31,340404
2	0,430309	43,871547
3	0,310021	31,893356
4	0,415047	29,129801
5	0,443161	28,243612

Cuadro 5.1: Comparativa de tiempos de consulta

5.2. Consulta de historial de valores

En esta implementación se hace recogida y consulta de datos de temperatura que se han obtenido usando el sensor Thermo 2 Click. La monitorización de temperaturas puede resultar muy interesante para detectar perdidas energéticas que se producen por ejemplo a través de malos aislamientos de las ventanas y así poder tomar medidas al respecto que supongan un ahorro económico.

En el flow de esta tarea se reciben los datos de temperatura que envía la placa cada cinco segundos por MQTT. Para ello se usa el flow superior de la Figura 5.4, donde el primer nodo se encarga de recibir los datos usando MQTT. A este nodo se le ha asignado un QoS 0 ya que los valores de temperatura se envían cada poco tiempo y el hecho de que haya algún tipo de fallo durante el envío de uno de estos datos no es algo relevante, además de esta manera se consigue no sobrecargar el ancho de banda de la red. El *topic* para el envío de datos es llamado *casa/salón/temperatura*, esta jerarquía facilita la posibilidad de escalar el proyecto en un futuro. El dato que llega se adapta a formato JSON para que pueda ser almacenado en la base de datos usando el nodo de Cloudant.

Por otro lado, se tiene el flow inferior de la Figura 5.4 para realizar las consultas sobre las temperaturas almacenadas. El flow comienza igual que en el apartado anterior, se recibe el mensaje del usuario y se redirige al chatbot, en este caso los nodos del diálogo que se activan cuando el usuario habla sobre temperaturas son *PeticionTemperatura* y dentro de este, se activa un subnodo según cuál sean las intenciones del *tempRecord* que se detecte. En este caso se ha hecho el ejemplo para detectar cuando el usuario quiere saber el historial de temperaturas, pero también podría haberse hecho para detectar la temperatura mínima o máxima. Después se hace una consulta a la base de datos donde se piden los diez últimos

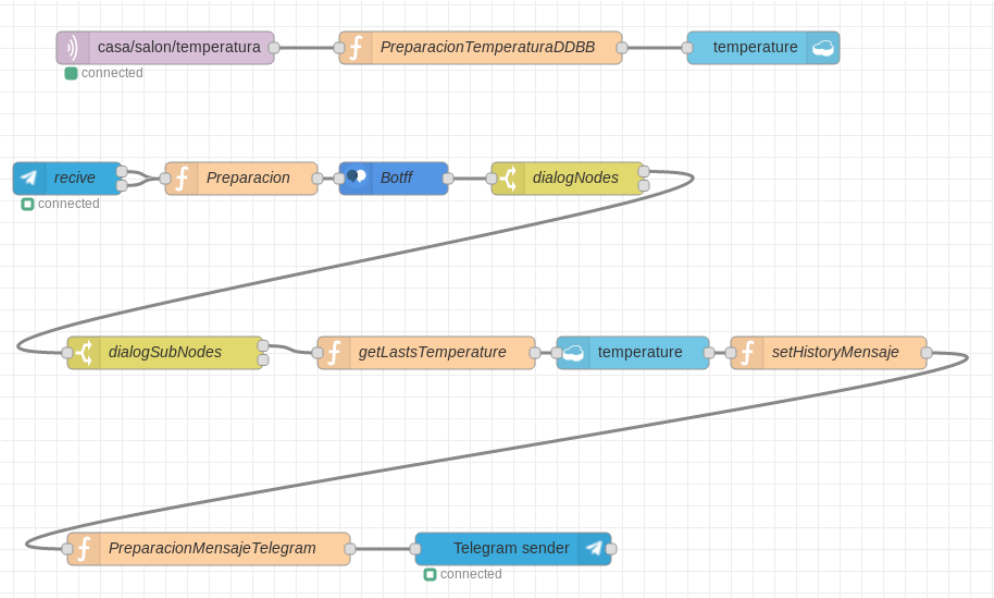


Figura 5.4: Flow en la plataforma Node-RED

valores de temperatura almacenados. Estos se encapsulan en formato JSON y se envían de vuelta al usuario por Telegram.

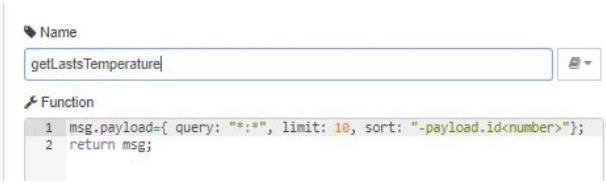


Figura 5.5: Función para hacer consulta a la base de datos

La función *getLastsTemperature* del flow de Node-RED es la que solicita el historial de temperaturas a la plataforma Cloudant. Esta función se puede visualizar en la Figura 5.5 y su código se encarga de hacer una consulta ordenando todos los datos almacenados en la tabla usando el parámetro id y limitando la consulta a solo diez valores. De esta manera se recuperan las diez últimas temperaturas almacenadas.

Por último, en la Figura 5.6 aparece como desde Telegram se realiza la petición del historial de temperaturas.

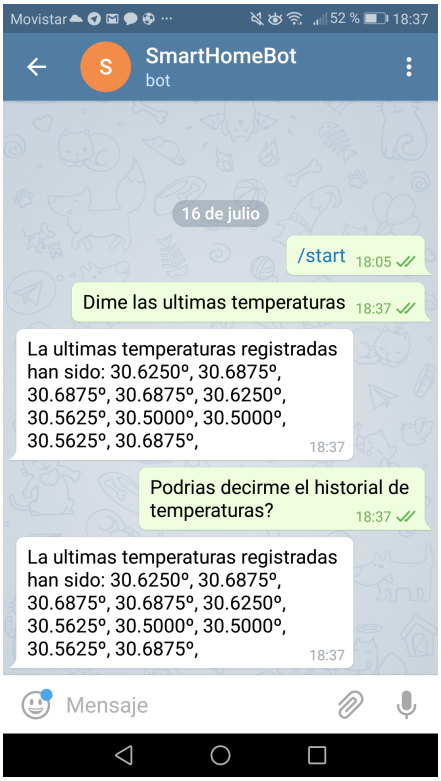


Figura 5.6: Consulta del historial de temperaturas desde Telegram

5.3. Envío de avisos asíncronos

En esta prueba se demuestra como se realiza el envío de información asíncrona a la plataforma. Es muy útil en aquellas situaciones en las que se quiere informar de que ha ocurrido algún tipo de evento poco habitual. De esta manera se consigue un mayor rendimiento que si se realizara un envío continuo de información. En la demostración se ha usado el sensor de movimiento Motion Click para detectar cuando hay intrusos en la casa y así enviar instantáneamente un mensaje al usuario alertándole de esta situación. En esta ocasión no se usa el chatbot ya que el envío de información se hace desde la plataforma sin que sea necesario que el usuario haya realizado previamente una consulta.

La placa o SoC tiene un programa en ejecución que cuando detecta movimiento con el sensor envía un mensaje por MQTT con la información de la hora en la que se ha detectado la presencia. El *topic* utilizado se llama *casa/salón/alarma* y se usa un QoS 2 para asegurar que el mensaje que se envía desde Node-RED llega al *broker*. Se ha usado también la propiedad del protocolo MQTT llamada *retain message*[1] que hace que el mensaje se guarde en el *broker*, de esta manera se asegura que la placa recibe el mensaje en el caso de pérdida de conexión con el *broker*.

[1] Retained messages https://www.hivemq.com/blog/mqtt-essentials-part-8-retained-messages.

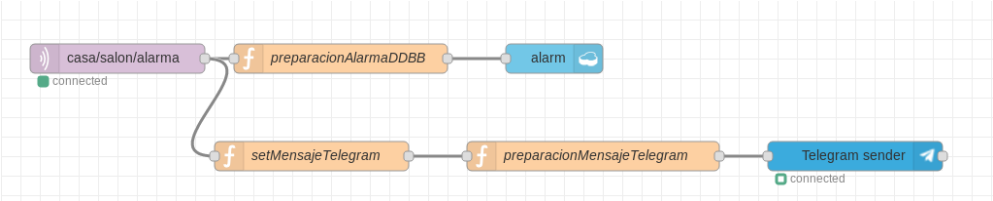


Figura 5.7: Flow en la plataforma Node-RED

El mensaje de alarma se almacena en la base de datos con la estructura que aparece en la Figura 5.8, así es posible tener el historial de todos los avisos. Además, también se hace llegar el aviso de alarma al usuario usando el nodo de Telegram del flow de la Figura 5.7.

Figura 5.8: Estructura de los datos en Cloudant

El aviso asíncrono que se le hace llegar al usuario a su aplicación móvil de Telegram se puede visualizar en la Figura 5.9.

Figura 5.9: Aviso de alarma en Telegram

5.4. Actuación sobre sensor

Otro caso de uso es la posibilidad de cambiar el estado de un sensor o actuador conectado a la placa. En las Smart Homes es muy habitual actuar sobre un relé para controlar el

encendido y apagado de algún aparato eléctrico, como una bombilla. En esta ocasión si que ha sido necesario el uso del chatbot para saber cuándo el usuario quiere hacer alguna acción sobre la bombilla. El control de la luz se activa cuando al flow de la Figura 5.10 llega un mensaje a través del nodo de Telegram que posteriormente es enviado al chatbot usando el nodo de Watson.

Dentro del chatbot, se activa en el *dialog* el nodo *OperacionesLuces* y según se active el nodo hijo *Encender* o *Apagar* se hará una acción u otra. En ambos casos se consulta previamente el estado actual de la luz, para ello se usan las funciones del flow *apagarFunction* y *encenderFunction*, Figura 5.11(a) y Figura 5.11(b) respectivamente.

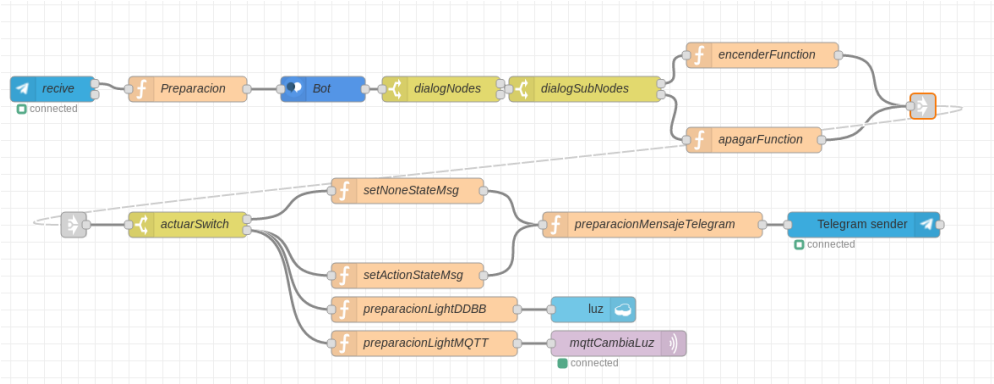
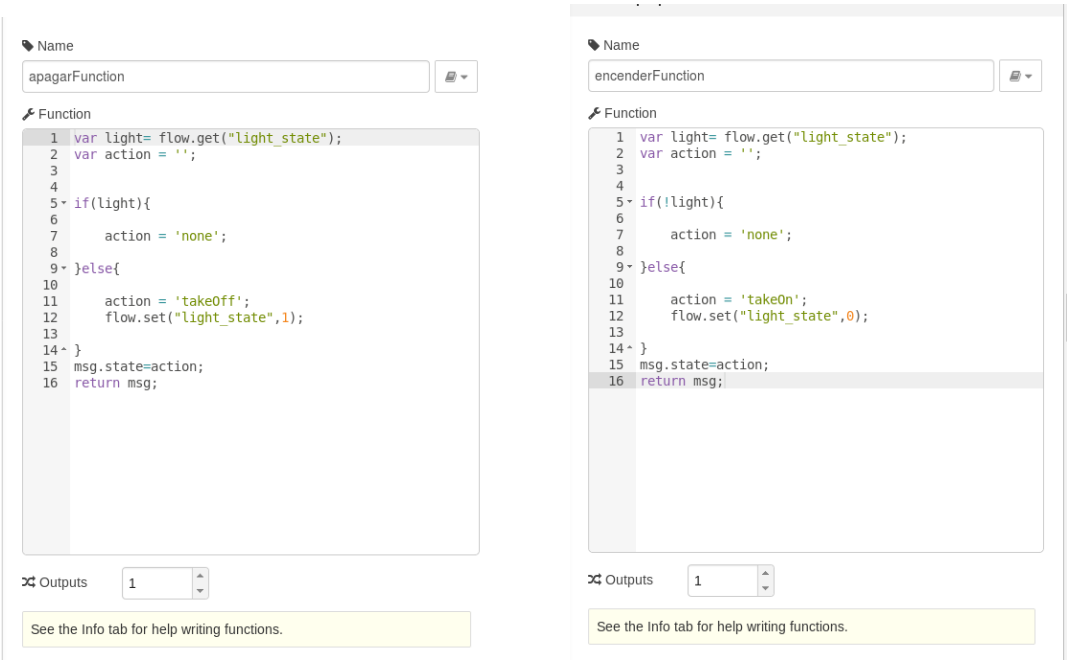


Figura 5.10: Flow en la plataforma Node-RED

Si el estado de la luz actual coincide con el estado que ha solicitado el usuario no se realiza ninguna acción, en caso contrario, se procede a cambiar el estado de la luz. En este caso se actualiza el estado de la luz en la base de datos y se envía un mensaje por MQTT a la placa con 1 o 0 para que actúe sobre el relé. El *topic* usado se llama *casa/salón/luz* y al igual que en el caso anterior se usa un QoS 2.

En la Figura 5.12 aparece el diálogo en Telegram entre el usuario y el chatbot. Aquí se puede ver como al principio el usuario solicita encender la luz y esta acción es llevada a cabo sin problemas, sin embargo, cuando el usuario de nuevo sugiere encender la luz, se le advierte que la luz ya estaba encendida. Lo mismo sucede en el proceso de apagar la luz. En el diálogo se pueden apreciar también las diferentes formas gramaticales que el usuario usa para referirse a las mismas acciones y que el chatbot sabe procesar igualmente.

El programa que se está ejecutando en la placa Ci40 es el que aparece en la Figura 5.13 y es el encargado de estar a la escucha de nuevos mensajes MQTT por el *topic* denominado *casa/salón/luz*. Como se ha comentado, estos mensajes tendrán el valor 0 o 1 y tras recibirlo se llama a una función que interactúa con el relé que está conectado al puerto MikroBUS.



(a) Código del nodo apagarFunction (b) Código del nodo encenderFunction

Figura 5.11: Nodos en Node-RED para consulta del estado actual

Figura 5.12: Control de encendido/apagado en Telegram

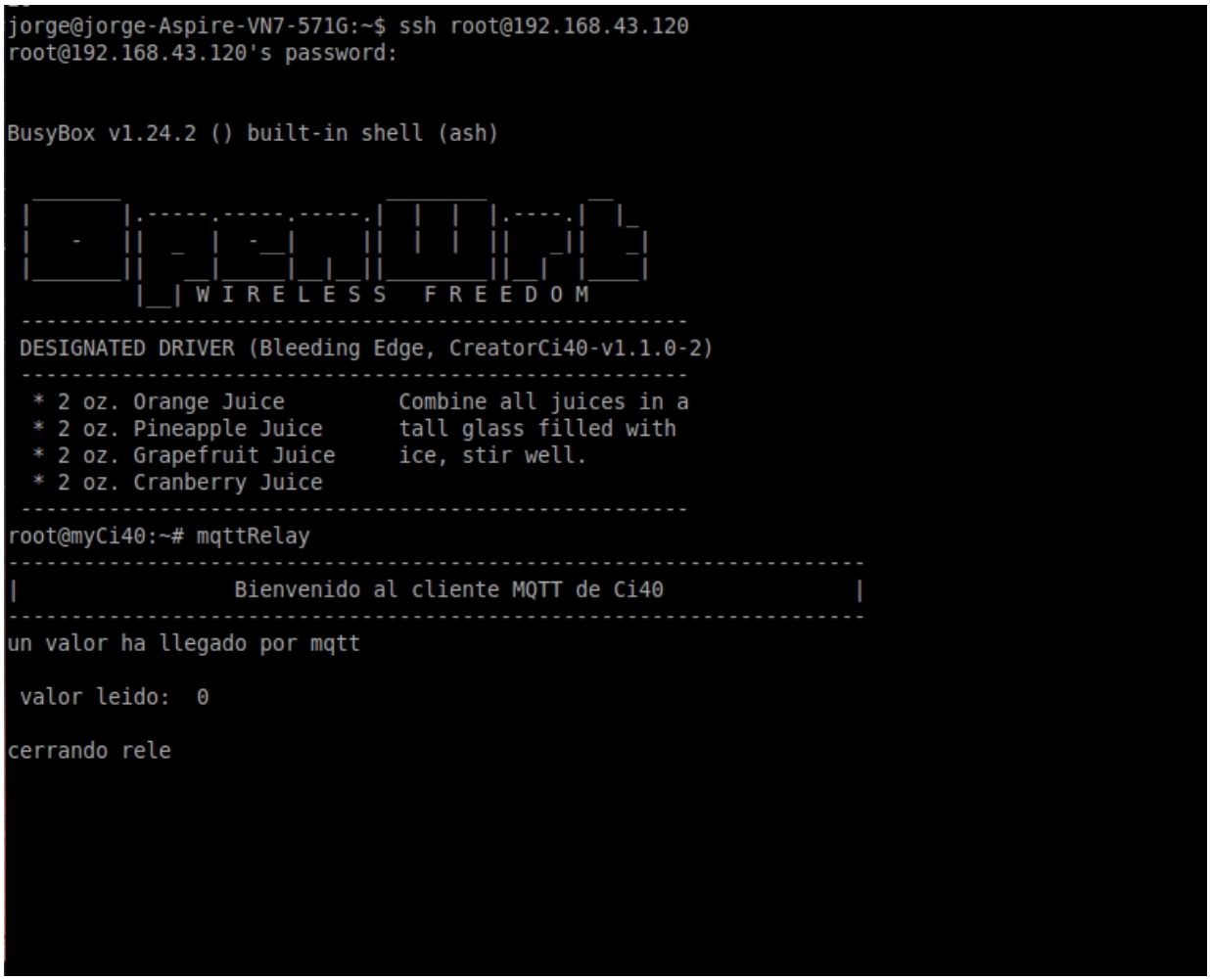


Figura 5.13: Programa ejecutándose en la terminal de la Ci40

Capítulo 6

Conclusiones y trabajos futuros

6.1. Conclusiones

En este trabajo se ha realizado el prototipo de una Smart Home usando tecnologías propias del paradigma IoT y añadiendo otras nuevas que facilitan la interacción de los usuarios con los dispositivos utilizando lenguaje natural desde sus smartphones. Se puede comprobar como se han cumplido los objetivos que se definieron al principio del proyecto (véase el apartado 1.2), para ello ha sido necesario usar una plataforma web de la compañía IBM capaz de conectarse tanto con dispositivos hardware como con otras plataformas o aplicaciones web, además se ha desarrollo también un chatbot para que la interacción del usuario sea más fácil e intuitiva. Respecto a la parte de hardware, se han desarrollado los programas que han hecho posible conectar la placa o SoC con los sensores y la placa con la plataforma web de IoT. Durante la realización de este proyecto han ido apareciendo nuevas necesidades, como la búsqueda de un protocolo de comunicación que haga posible el intercambio de información entre la placa y la plataforma, o la elección de una de las aplicaciones de mensajería instantánea ya existentes que contara con una infraestructura que permitiera conectar dicha app con la plataforma donde se ha desarrollado la solución IoT. Para ver con más detalle los resultados obtenidos, se va a analizar lo conseguido en cada uno de los casos prácticos explicados en el capítulo 5.

- Caso 1: en este ejemplo se han medido una serie de tiempos que hacen ver la diferencia de velocidad que hay entre los posibles métodos con los que cuenta la plataforma para guardar valores, usar variables propias en la plataforma o usar una base de datos externa. Esta diferencia se debe a que al usar una base de datos hay que comunicarse con un servidor externo y además se tiene que ejecutar la consulta que se formula. Sin embargo, hay que tener en cuenta que aunque la diferencia es grande, se están manejando valores de una magnitud de milisegundos y por tanto se puede seguir hablando de una aplicación en tiempo real. Además, el uso de variables de la plataforma solo tiene sentido en aquellos casos en el que se quiere guardar uno o dos valores concretos, para los casos en que se quiere almacenar continuamente valores que llegan por ejemplo de un sensor o cuando es necesario realizar diferentes consultas sobre un

historial de valores, la solución óptima es el uso de una base de datos. Por otro lado, en este caso no ha sido necesario comunicarse con la placa Ci40 para conocer el estado de la bombilla, ya que se parte de la base de que la bombilla no tiene ningún interruptor externo y únicamente es controlada a través de la plataforma (Caso 4), por lo que cada vez que se envía una orden de encendido/apagado se modifica la variable interna de la plataforma, que es la que se consulta en este caso. De esta manera de nuevo se consigue un ahorro de tiempo en la respuesta.

- Caso 2: en este entorno se usa MQTT para recibir todos aquellos valores del sensor que está conectado a la placa. Cabe aclarar la facilidad para escalar el proyecto usando más sensores de temperatura simplemente conectándolos a la placa y publicando sus valores en un nuevo *topic*, aprovechando así la jerarquía de este protocolo. Por ejemplo, si el sensor está en una habitación se podría usar la denominación *casa/habitacion1/temperatura*, en el flow de la plataforma solo habría que añadir un nuevo nodo MQTT que estuviera a la escucha de este *topic*. En este caso como se están recogiendo valores de temperatura periódicamente se usa la base de datos Cloudant para almacenar dichos datos. Si se quisiera almacenar valores de más sensores de temperatura no sería necesario añadir más tablas, bastaría con añadir un identificador o id a cada valor de temperatura que se almacene según pertenezca a un sensor u a otro.

- Caso 3: para el caso del envío de avisos asíncronos ha sido muy importante el uso del atributo *retain messages* a la hora de enviar el mensaje por MQTT desde la placa. De esta manera el *broker* retiene este mensaje hasta asegurarse de que lo ha recibido cada uno de los subscriptores, que en este caso sería únicamente el nodo MQTT de la plataforma. Si cuando se publica el mensaje, la plataforma no estuviera escuchando el *topic* o hubiera tenido algún problema de red, al conectarse de nuevo al *broker* le aparecería el mensaje. Además al usar el protocolo MQTT con un QoS 2 se asegura que cada aviso de alarma llega una única vez a la plataforma y así se evita el envío de avisos duplicados al usuario.

- Caso 4: en este ejemplo se hace notable la gran utilidad que tiene el chatbot para facilitar la tarea del manejo de los dispositivos para el usuario ya que se consigue detectar con mucha eficacia cuando este tiene la intención de apagar o encender las luces y siendo capaz además de diferenciarlo de las otras funciones que también tiene implementadas el chatbot, como las ya comentadas de consulta de temperaturas o del estado de la luz. Por otro lado, para el envío de la orden apagar/encender a la placa por MQTT de nuevo se ha usado un QoS de 2. El atributo *retain message* no tiene sentido aplicarlo en este contexto ya que las peticiones que manda el cliente son para ejecutarlas en ese momento, no pasado un tiempo cuando la placa recupere la conexión con el *broker*.

Para resumir, en todos los casos se ha conseguido crear un entorno que interacciona con el cliente en tiempo real y que además es tolerante a fallos, lo que lo hace muy robusto y aplicable a un entorno real como lo es una vivienda.

6.2. Conclusions

In this work it has been developed a prototype of a Smart Home. It has been done using the specific technology of the paradigm IoT, and a newer technology to facilitate the interaction between the user and the devices, by utilizing their smartphones and making use of natural language. It can be seen how the objectives that were defined at the beginning of the project (look at the section 2.2), have been reached. To fulfill this, it has been necessary the use of a web platform belonging to the company IBM, which is able to connect itself with hardware devices, and with other platforms or web applications. In addition, a chatbot has been developed to make the interactions with the user much easier and more intuitive. Regarding the hardware, the development of programs has made possible to connect the board or SoC to the sensors and the board to the IoT web platform. During the conduction of this project, new necessities have kept arising, such as the search for a communication protocol that enables the exchange of information between the board and the platform, or the choice of one of the instant messaging apps that already existed and which had an infrastructure that permitted the connection of that app to the platform where the IoT solution had been developed. To have a more detailed insight of the results, it will be analysed one by one each practical case which was explained in chapter 5.

- Case 1: this example includes a time series that show the difference between using the platform methods to the store of values, using inherent values in the platform and using an external data base. This difference is due to the need to communicate with an external server when using a data base, also, it is necessary to run the consultations that have been formulated. However, account is to be taken that even though the difference is great, the values that are being handled are of a magnitude of milliseconds and consequently, it can be said it to be an app operating in real time. Besides, the use of variables in the platform only makes sense in those cases when only one or two values have to be stored. When it is desired to store values continuously, that comes for example from a sensor; or when it is necessary to make different consultations of a history of values, the optimum solution is to use a data base. Conversely, in this case it was not necessary to communicate with the board Ci40 to know the bulb condition, since it has been assumed that the bulb lacks of any external switch and that it can only be controlled through the platform (case 4). So, each time that an order is sent of switching it on/off, the internal variable of the platform is being modified, which is the one being consulted in this case. In this way some response time is saved.

- Case 2: in this environment, MQTT is used to receive all values are measured by the sensor connected to the board. It should be pointed out the ease to make bigger the project by using more temperature sensors, adding them to the board and publishing their values in a new *topic*, taking advantage of the hierarchy of this protocol. For instance, if the sensor is located in a room, it could be named as *casa/habitacion1/temperatura*, in the flow of the platform it would be necessary to add only a new MQTT node that were listening to this *topic*. In this case, as the temperature values are being collected periodically, it will be the Cloudant date base the one in use in order to store such

data. If there is a will to store some values collected from other temperature sensors, it would not be necessary to add more database tables, it would be enough to add a new identifier or id to each of the temperature's values that are being stored, according to its belonging to one sensor or another.

- Case 3: in the case of sending asynchronous notifications, it is really important the use of the term *retain messages*, specifically when it is necessary to send a message through the MQTT protocol from the board. In this way, the *broker* retains this message until it is sure that it has been received by every subscriber, which in this case it will be only the MQTT node from the platform. If it is the case that when the message is published, the platform does not listen to the *topics*, or if there is network related problems, when the platform connects again to the *broker*, the message will be still there. Also, using the MQTT protocol together with a QoS 2, makes sure that each warning arrives only once to the platform, avoiding the resending of the same message to the user.

- Case 4: with this example, it is very noticeable the great importance of the chatbot when it comes to ease the handling of the devices for the customer, since it is possible to detect when he has the intention to turn on or off the lights, differentiating it from the other functions that the chatbot has implemented, such as the ones already mentioned of checking the temperature or the light state. On the other hand, to send the command to switch the lights on or off to the board by the MQTT, it is used again a QoS of 2. The attribute *retain message* makes no sense in this context since the petitions that are sent by the client are only expected to be run in that moment, not after a time when the board reconnects with the *broker*.

To sum up, in every case it has been possible to create an environment that interacts with the client in real time and that is fault tolerant, which makes it really robust and suitable for a real setting such as a Smart Home.

6.3. Trabajo futuro

Durante la realización de este proyecto han surgido nuevas ideas y características que podrían suponer una mejora del proyecto pero que quedaban fuera del alcance de este.

Por un lado, el *broker* usado en este trabajo para realizar la comunicación MQTT es público, gratuito y es utilizado en muchas ocasiones para realizar proyectos de prueba. Este servidor habitualmente sufre cortes o sobrecargas en la red y apenas tiene opciones de configuración de la comunicación, por lo que para un entorno real sería necesario o bien contratar alguna empresa que se encargue del manejo de un *broker* robusto o montar localmente uno propio.

Otra característica del protocolo MQTT que sería interesante usar es el Last Will and Testament (LWT)[1] que permite hacer saber al resto de los usuarios cuando uno de ellos se ha desconectado repentinamente. Por ejemplo se podría saber cuando la placa se ha desconectado de la red por algún motivo y ya no esta funcional, de esta manera se podrían llevar a cabo acciones correctivas correspondientes a esta situación y avisar así por ejemplo al usuario a través de Telegram.

Dejando el tema MQTT a un lado, en cada uno de los casos prácticos que se han realizado únicamente se usaban uno o dos sensores y/o actuadores conectados directamente a la board usando el puerto MikroBUS. En un entorno real no sería factible debido a que el número de puertos MikroBUS que tiene la placa es limitado y que la cantidad de sensores que se usan en una aplicación real es mucho más grande, además la conexión física de los sensores en la placa no es la mejor opción debido a que la idea de un Smart Home es tener sensores repartidos por toda la casa. Luego la mejora que se propone es usar una comunicación inalámbrica entre los sensores y el board. Una de las tecnologías más usadas actualmente para estos entornos es el Bluetooth Low Energy (BLE) que es igual que el Bluetooth clásico pero con un consumo menor. La arquitectura de la implementación tendría la board Ci40, que cuenta con esta tecnología BLE, como Gateway, es decir, de mediador entre la plataforma IoT y los sensores inalámbricos. Sería por tanto necesario usar nodos que cuenten con diferentes sensores ya integrados y que además tengan BLE para que así se puedan comunicar con la Ci40. Se podría usar por ejemplo el SoC de bajo coste llamado Sensor Tag [2].

También aclarar que aunque a lo largo del proyecto se ha hablado de interacción escrita con el chatbot, la solución propuesta está abierta también al uso de la voz ya que los teclados que tienen instalados los smartphones de Android e iOS cuentan con una función capaz de convertir la voz a texto, por lo que no es necesario escribir para poder dar órdenes a la Smart Home. Sin embargo, sería interesante llegar a implementar un asistente inteligente para el hogar como Google Home o Amazon Alexa y poder interactuar directamente con él usando la voz sin ser necesario tener el smartphone en frente.

Por último, hay que tener en cuenta la gran cantidad de datos que se pueden llegar a recoger de todos los sensores desplegados en una Smart Home a lo largo de un día. Es mucha la información relevante que se puede obtener de estos datos. Con las técnicas de tratamiento

[1] Last will and testament https://www.hivemq.com/blog/mqtt-essentials-part-9-last-will-and-testament.
[2] Sensortag web page http://www.ti.com/ww/en/wireless_connectivity/.

de datos que hay en la actualidad es posible trabajar con cualquier tipo de dato en tiempo real y en aplicaciones que se ejecutan en cloud. Una propuesta aplicable a este proyecto sería conectar la plataforma Node-RED con alguna herramienta de análisis de datos como Apache Spark o mongoDB. Con estas herramientas se tiene mucha flexibilidad a la hora de incorporar los datos procedentes de nuevos sensores independientemente de la estructura de estos datos, algo impensable en las bases de datos relacionales. De esta manera se podrían extraer hábitos o costumbres del usuario en el uso de los dispositivos de la Smart Home. Esto permitiría a la plataforma la toma de decisiones autónomamente sin la interacción del usuario. Por ejemplo, se podría variar la intensidad de la luz a una hora determinada del día si así lo suele hacer el usuario.

Además de extraer información de los hábitos del usuario, también es posible que la plataforma tome decisiones de forma inteligente con la información procedente de los dispositivos que lleva consigo el usuario como los smartphones o las smartbands y de esta manera obtener información muy valiosa como las coordenadas gps. Por ejemplo se podría encender la calefacción cuando se detecte que el usuario esté cerca de casa y así ahorrar energía, apagar automáticamente las luces cuando el usuario salga del hogar o incluso poner en marcha un robot de limpieza como Roomba para que limpie solo cuando el cliente esté fuera.

Bibliografía

[1] Kazi Masudul Alam and Ashrafi Akram. A Survey on MQTT Protocol for the Internet of Things. *Khulna University, Dept. of Computer Science and Engineering (CSE)*, 2016.

[2] Julie A. Ask, Michael Facemire, and Andrew Hogan. The State Of Chatbots. Technical report, Forrester, October 2016.

[3] Ahmed Azraq. *Building Cognitive Applications with IBM Watson* , volume 2. IBM Redbooks, 2017.

[4] Dave Evans. How the Next Evolution of the Internet Is Changing Everything. Technical report, Cisco Internet Business Solutions Group (IBSG), 2011.

[5] Benjamin Gautier. The rise of intelligent voice assistants. Technical report, Wavestone, 2016.

[6] Cyril Joe Baby, Faizan Ayyub Khan, and Swathi J N. Home Automation using IoT and a Chatbot using Natural Language Processing. Technical report, VIT University, School of Computer Science and Engineering, 2017. https://ieeexplore.ieee.org/document/8245185/.

[7] Dunja Mladenić Jožef, Stefan Institute, Luka Bradeško, and Dunja Mladenić. A Survey of Chatbot Systems through a Loebner Prize Competition. *Researchgate*, 2, 2012. https://www.researchgate.net/publication/235664166.

[8] By Kiran, Jot Singh, and Divneet Singh Kapoor. Create Your Own Internet of Things. *Ieee Consumer Electronics Magazine*, April 2017.

[9] Milica Lekić and Gordana Gardašević. IoT sensor integration to Node-RED platform. *International Symposium INFOTEH-JAHORINA*, 2018. https://ieeexplore.ieee.org/document/8345544/.

[10] Bruno Marietto, Maria das Graças Aguiar, and Rafael Varagode Aguias. Artificial Intelligence Markup Language: A Brief Tutorial. https://arxiv.org/ftp/arxiv/papers/1307/1307.3091.pdf. 2013.

[11] Martin Lärka. Smart homes with smartphones. Master's thesis, Umea University, Department of Applied Physics and Electronics, 2015.

[12] MikroEletronika. mikroBUS Standard specifications. https://download.mikroe.com/documents/standards/mikrobus/mikrobus-standard-specification-v200.pdf.

[13] MikroEletronika. Motion Click datasheet. https://download.mikroe.com/documents/add-on-boards/click/motion/motion-click-schematic-v101.pdf.

[14] MikroEletronika. Relay 2 Click datasheet. https://download.mikroe.com/documents/add-on-boards/click/relay-2/relay-2-click-schematic-v100.pdf.

[15] MikroEletronika. Thermo 2 Click Temperature Sensor datasheet. https://docs-emea.rs-online.com/webdocs/159a/0900766b8159ab22.pdf.

[16] Nitin Naik. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. *IEEE International Symposium on Systems Engineering*, 2017.

[17] Deepak Panth. A Survey on Security Mechanisms of Leading Cloud Service Providers. *International Journal of Computer Applications*, 98. http://research.ijcaonline.org/volume98/number1/pxc3897184.pdf. 2014.

[18] Dimitrios Sikeridis, Ioannis Papapanagiotou, Bhaskar Prasad Rimal, and Michael Devetsikiotis. A Comparative Taxonomy and Survey of Public Cloud Infrastructure Vendors. http://arxiv.org/abs/1710.01476. 2017.

[19] Imagination Technologies. Ci40 Hardware User Guide. https://docs-emea.rs-online.com/webdocs/1551/0900766b815516a7.pdf.