Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

# Trabajo de Fin de Grado

*Grado en Ingeniería Informática*

## Conectando tu hogar con XBee

***Connecting your home with XBee***

**Autor**

*Juan Carlos González Pascolo*

La Laguna, 10 de Junio de 2019

D. **Alberto F. Hamilton Castro**, con N.I.F. 43773884-P, profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

**C E R T I F I C A**

Que la presente memoria titulada:

*"Conectando tu hogar con XBee"*

ha sido realizada bajo su dirección por D. **Juan Carlos González Pascolo**,

con N.I.F. 78649897-V.

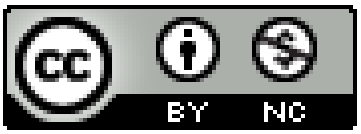
Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 10 de junio de 2018

# Agradecimientos

Tras tres meses dedicados este Trabajo de Fin de Grado quisiera dar las gracias a:

Mi tutor Alberto F. Hamilton Castro por la ayuda ofrecida tanto para la realización del trabajo como para la elaboración de la presente memoria.

# Licencia



# Resumen

Actualmente las redes de sensores inalámbricas han ganado mucha popularidad debido a su gran abanico de posibilidades, su facilidad de configuración y cada vez menor coste.

Orientado a este tema está enfocado el presente documento, a lo largo del cual explicaré las características principales de los módulos XBee y la creación de una interfaz web donde se pueda manejar de manera automática un sistema domótico basado en una red de sensores con este tipo de dispositivos.

Se comenzará con un capítulo introductorio para sumergirnos en el área de las redes de sensores y definir claramente los objetivos. Después se continuará con un capítulo sobre conocimientos previos en el que se mencionan las tecnologías utilizadas para llevar a cabo el trabajo y las especificaciones técnicas de los módulos XBee.

Luego se pasará a desarrollar el trabajo implementado, manifestando, las fases de desarrollo y los problemas detectados.

Por último, se expondrán las conclusiones a las que se ha llegado y cuáles podrían ser las mejoras ha implementar en el proyecto. También se dispondrá de una tabla referente al presupuesto necesario para poder llevar a cabo el Trabajo de Fin de Grado.

**Palabras clave:** XBee, ZigBee, JavaScript, NodeJS, VueJS

# Abstract

Currently the wireless sensor networks have risen its popularity due to its large range of possibilities, its ease of configuration and its low cost.

This document is oriented in that topic. Through this file I am going to explain the main features of XBee module and the creation of the web interface where the users can manage in an automatically way a domotic system based in a sensor network with this kind of devices.

The document begin with an introductory chapter where the technical specifications of XBee modules will be detailed. After that, a chapter about previous knowledge where the technologies that were used for the assignment development will be mentioned.

The next chapter is about the steps that were implemented in order to achieve the project. In this chapter the aims, the phases and the problems detected are clearly exposed.

Finally, we are going to show the conclusion and some improvements of the assignment. In addition, a chart with the budget is provided.

**Keywords:** XBee, ZigBee, JavaScript, NodeJS, VueJS

# Abreviaturas utilizadas

**IEEE:** Institute of Electrical and Electronics Engineers
**LR-WPAN:** Low-Rate Wireless Personal Area Network
**TFG:** Trabajo de Fin de Grado
**PB:** Protoboard
**JS:** JavaScript
**npm:** Node Package Manager
**PAN:** Personal Area Network
**E/S:** Entrada o Salida

# Índice general

# Capítulo 1 Introducción

## 1.1 Qué es Xbee

Los módulos XBee son soluciones integradas fabricadas por DIGI Internacional que brindan un medio inalámbrico para la interconexión y comunicación entre dispositivos los cuales se utilizan para crear redes Fast Point-To-Multipoint o redes Peer-To-Peer. Utilizan el estándar IEEE 802.15.4, el cual define las operaciones de nivel físico y control de acceso al medio en redes inalámbrica de área personal con bajas tasas de transmisión de datos (LR-WPAN). Están basados en el protocolo ZigBee el cual destaca por su eficiencia energética y de costo. A esto hay que añadirle que debido a su sencillez de configuración y a su capacidad para crear una o varias redes con múltiples dispositivos es una opción más que fiable para multitud de aplicaciones.

## 1.2 Antecedentes y estado actual del tema

Desde que el ser humano logró crear instrumentos que le facilitaran labores cotidianas la sociedad ha ido evolucionando en dirección a un mundo tecnológico en el que sensores, actuadores y redes forman ya parte de nuestra vida diaria. De esto se han aprovechado muchas industrias como la de la domótica, la cual ha visto en estos instrumentos una vía para facilitar la vida en el hogar.

Desde que en 1887 Heinrich Rudolf Hertz logró transmitir electricidad en forma de ondas electromagnéticas a través del aire consiguiendo la primera comunicación sin cableado ninguno el avance en comunicación inalámbrica no ha parado, llegando en 1971 a la creación de la primera red local inalámbrica (WLAN) en la Universidad de Hawaii y dando así el comienzo al uso de tecnologías para redes inalámbricas, y con ello, la necesidad de crear estándares para la comunicación de los distintos dispositivos. Estos estándares fueron siendo creados bajo el IEEE 802.11 y dividiéndolos según sus características como la velocidad de transmisión o el espectro de banda que emiten.

Y del conjunto de algunos de estos estándares nació ZigBee, que tal y como de indica en [2] es una solución inalámbrica global que de manera económica y conveniente puede controlar una amplia gama de dispositivos para mejorar la comodidad y seguridad de los consumidores.

Los puntos fuertes de esta solución es su baja potencia, la elevada duración de la batería de los dispositivos que la implementan y la gran variedad de topologías que pueden ser creadas con ella. Todo ello hace que sea una solución perfecta para la automatización del hogar y la administración de sistemas de

construcción.

Basándose en ZigBee se crea en 2005 el primer módulo XBee bajo el estándar 802.15 (LR-WPAN). Por su baja velocidad de transmisión de datos, bajo coste y alta eficacia se empieza a utilizar en múltiples aplicaciones, entre las cuales podemos encontrar:

- *Proyectos,en fase de prueba, por parte de la NASA* donde XBee se usaría como alternativa para crear nuevos modelos de comunicación ya que debido a su bajo coste y peso se podrían crear naves espaciales más ligeras y capaces de aumentar su capacidad de carga
- En agricultura, para crear *eficientes invernaderos* donde XBee se utiliza para transmitir los datos de temperatura, humedad, pH y luminosidad.
- En el mundo artístico, *un equipo de cámara 360º para smartphones* desarrollado por unos alumnos de la Middlesex University London en la que XBee se utiliza para conectividad remota con la plataforma que gira para poder crear la imagen de 360º.

Cabe mencionar que el departamento al cual pertenece el tutor de este TFG tiene experiencia previa en el uso de XBee y que el tutor ha desarrollado un código previo que proporciona.

## *1.3 Objetivos*

El objetivo general de este trabajo es utilizar estos módulos para realizar una aplicación domótica en la cual a través de una interfaz web o móvil se puedan controlar los distintos dispositivos conectados a estas placas en respuesta a las órdenes del usuario o de los datos obtenidos por otros elementos conectados a la misma o a otra placa (pulsadores, interruptores, sensores de presencia, temperatura, etc.). Todo ello con un bajo consumo y ofreciendo como objetivo principal una gran flexibilidad de configuración y ampliación.

Para conseguirlo, los objetivos específicos planteados al inicio de este TFG fueron:

- Realizar un diseño básico y estático en el cual se le enviaran los comandos literales a los módulos XBee a través de una interfaz web. Teniendo en cuenta que para esto debía crear primero un servidor web.
- Mejorar el diseño anterior para no tener que enviar los comando literales sino a partir de configuraciones. Con esto quiero decir que es el usuario el que elige el input y el parámetro que dan lugar a una reacción en un output. Todo esto sin tener que conocer en qué pin está cada elemento.
- Otra meta a alcanzar era diferenciar tipos de usuarios. Esto es debido a que si se quiere llevar a un entorno real de una casa lo más probable es que no todos los miembros de la familia deban tener los mismos accesos al sistema. Por ejemplo, un niño pequeño no debería tener permiso a elementos peligrosos.
- Por último, debido a que hoy en día la mayoría de los usuarios prefieren utilizar el móvil en lugar de otro dispositivo, otro objetivo era la creación de esta interfaz para móviles.

# Capítulo 2 Conocimientos previos

## 2.1 XBee

### 2.1.1 Comunicación

Los módulos XBee se comunican los unos con los otros a través del aire, recibiendo y enviando paquetes de datos inalámbricos.

El problema es que, aunque gracias al firmware los módulos pueden ejecutar pequeños programas para comunicarse entre ellos, esto es lo único que pueden hacer, enviar los paquetes. Por tanto, para gestionar los datos que se envían y reciben son necesarios otros dispositivos como microcontroladores u ordenadores. Por suerte, la comunicación entre los módulos XBee y estos otros dispositivos se puede hacer de forma sencilla a través de una interfaz serial. Por tanto, el proceso de comunicación es el siguiente:

- El dispositivo inteligente crea el paquete de datos que quiere enviar.
- El dispositivo XBee conectado por serial al dispositivo del punto anterior envía los datos de forma inalámbrica.
- De forma inversa, los datos enviados por otros módulos llegan a aquel conectado al dispositivo inteligente, el cual es capaz a través del serial de reenviar estos datos al dispositivo al que está unido para ser procesados.

De esta forma los ordenadores y microcontroladores pueden enviar y administrar los mensajes entre XBee.

Sin embargo, la comunicación entre el módulo XBee y el dispositivo inteligente debe seguir un protocolo para poder entender la información. Este protocolo puede ser creado por el desarrollador o, la opción más fácil y utilizada en este TFG, usar el modo API proporcionado por XBee.

Este modo proporciona una interfaz estructurada donde la información se comunica a través del serial en paquetes organizados y en un orden determinado, permitiendo así una comunicación compleja sin tener que definir un protocolo propio.

Como pueden haber más de un módulo XBee por red es necesario que cada uno de estos tenga un identificador. Este identificador puede ser de 64 bits, de 16 bits o de un identificador puesto por el programador para que sea más sencillo su reconocimiento.

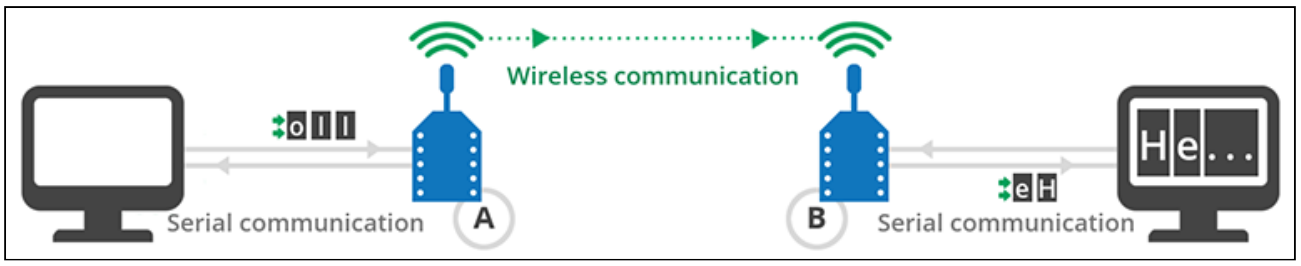



*Imagen 1: Descripción gráfica de la comunicación entre dispositivos XBee y otros dispositivos.* [3]

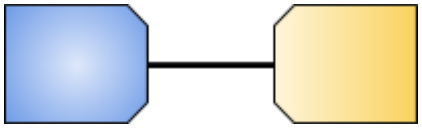
### 2.1.2 *Tipos de topologías*

Como se ha comentado anteriormente una de las grandes ventajas de XBee es la capacidad de crear diversas redes con diversas topologías gracias a ZigBee.

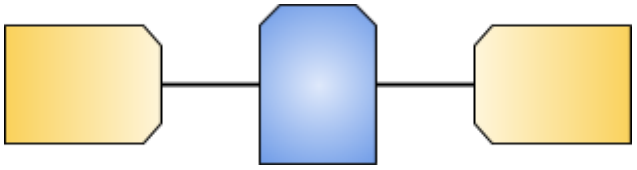
En todas las redes podemos diferenciar al menos dos tipos de dispositivos, el *Coordinador*, el cual debe ser único, y el *Dispositivo final*; sin embargo, en algunos casos se añade otro tipo, el *Enrutador*.

Las topologías que pueden ser creadas son:

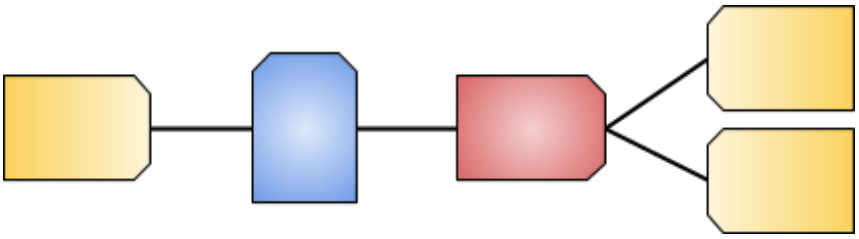
- Topología Peer-to-Peer: La más sencilla. Consiste únicamente en conectar dos módulos XBee. En este tipo de topología uno debe ejercer como coordinador y otro como esclavo.



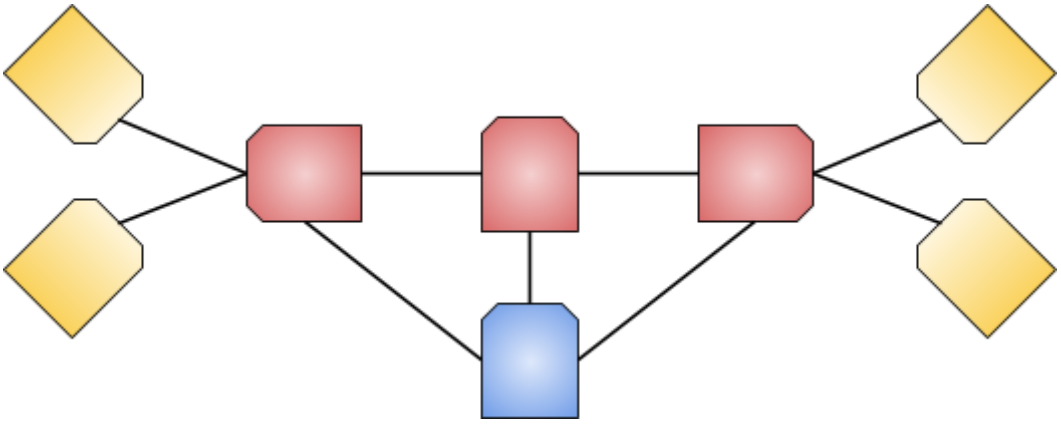
- Topología en estrella (star): En este caso hay más de dos módulos, de los cuales uno ejerce la función de coordinador, que recibe y envía los paquetes de datos al resto de módulos, que son esclavos.



- Topología en árbol: También intervienen más de dos dispositivos pero en este caso todos o algunos esclavos no se comunican con el coordinador directamente sino que lo hacen a través de un módulo intermedio mediante una única vía, el enrutador.



- Topología en malla: Similar a la topología en árbol pero con la diferencia de que los datos pueden llegar al coordinador por diferentes caminos.



Leyenda:


### 2.1.3 *Especificaciones técnicas*

En concreto, los módulos utilizados para este Trabajo de Fin de Grado son XBee ZNet 2.5. Descripción técnica encontrada en [1]

| | |
|---|---|
| ***Rango con obstáculos (Urban Range)*** | 40 metros |
| ***Rango en línea (Line Range)*** | 120 metros |
| ***Corriente para transmitir (TX current)*** | 40 mA |
| ***Corriente para recibir (RX current)*** | 40 mA |
| ***Corriente en reposo (Idle current)*** | 15 mA |
| ***Entradas/Salidas Digitales (Digital IO pins)*** | 11 |
| ***Entradas Analógicas ( Analog input pins)*** | 4 |
| ***Velocidad de transmisión (RF Data Rate)*** | 250 kbps |
| ***Número de equipos en la misma red*** | 65535 (16 bits) |
| ***Número de canales*** | 16 Direct Sequence Channels |
| ***Voltaje de alimentación (supply voltage)*** | 2.1-3.6 V |
| ***Frecuencia (Operating Frequency Band)*** | ISM 2.4 GHz |
| ***Dimensiones*** | 2.438cm x 2.761cm |

| | |
|---|---|
| ***Temperatura de funcionamiento*** | -40 to 85ºC (industrial) |

***Tabla 1: Características de XBee ZNet 2.5***

A todo esto hay que añadirle el componente software del módulo. Los XBee son capaces de ejecutar ciertos programas gracias a un pequeño microcontrolador. Estos programas son conocidos como el firmware y permiten que un módulo pueda conectarse con el resto además de decirle de qué forma.

### *2.1.4 Funcionamiento*

Cada módulo XBee cuenta con veinte pines de los cuales podemos configurar siete como entrada o salida digital y cinco como entrada o salida digital o entrada analógica. El resto de pines están reservados para funciones específicas como pueden ser los dispositivos UART (Universal Asynchronous Receiver-Transmitter) o para resetear el módulo.

Es a estos pines a los que debemos conectar las distintas electrónicas que queremos utilizar en nuestra red domótica como pueden ser pulsadores, sensores de cualquier tipos y LEDs. De esta forma, conociendo el identificador del módulo y el pin al que está conectado un elemento en concreto podemos enviar órdenes al mismo a través de XBee.

En la siguiente tabla se muestra la distribución de pines, obtenida de [1]:

| ***Nº Pin*** | ***Nombre*** | ***Dirección*** | ***Descripción*** |
|---|---|---|---|
| 1 | -VCC | - | Fuente de alimentación |
| 2 | -DOUT | salida | Emisor UART |
| 3 | -DIN<br>-CONFIG | entrada | Receptor UART |
| 4 | -DIO12 | ambos | Entrada/Salida digital 12 |
| 5 | -RESET | entrada | Módulo de reseteo |
| 6 | -DIO10<br>-RSSI<br>-PWMO | ambos | E/S digital 10<br>RX Signal Strength Indicator<br>Pulse Width Modulation |
| 7 | -DIO11 | ambos | E/S digital 11 |
| 8 | [reserved] | - | No conectar |
| 9 | -DTR<br>-SLEEP_RQ<br>-DIO8 | ambos | Línea de control de suspensión de pines<br>E/S digital 10 |

| | | | |
|---|---|---|---|
| 10 | -GND | - | Toma a tierra |
| 11 | -DIO4 | ambos | E/S digital 4 |
| 12 | -CTS<br>-DIO7 | ambos | Control de flujo CTS<br>E/S digital 7 |
| 13 | -ON<br>-SLEEP<br>-DIO9 | salida | Indicador de estado del módulo<br>E/S digital 9 |
| 14 | [reserved] | - | No conectar |
| 15 | -Associate<br>-DIO5 | ambos | Indicador de “asociado”<br>E/S digital 5 |
| 16 | -RTS<br>-DIO6 | ambos | Control de flujo RTS<br>E/S digital 6 |
| 17 | -AD3<br>-DIO3 | ambos | Entrada analógica o E/S digital 3 |
| 18 | -AD2<br>-DIO2 | ambos | Entrada analógica o E/S digital 2 |
| 19 | -AD1<br>-DIO1 | ambos | Entrada analógica o E/S digital 1 |
| 20 | -AD0<br>-DIO0<br>-CB | ambos | Entrada analógica o E/S digital 0.<br>Botón de puesta en marcha (para ayudar a desplegar dispositivos en una red) |

***Tabla 2: Distribución de pines***

### *2.1.5 Comandos AT*

Ya que uno de los objetivos es poder leer o cambiar la configuración del dispositivo XBee local, es por ello que se debe tener un modo de comando con tal fin. En el caso de XBee este modo de comandos es el modo comandos AT.

Los comandos AT, denominados así por la abreviatura de *attention*, son instrucciones codificadas que conforman un lenguaje de comunicación entre el hombre y un terminal modem.

Cada módulo XBee tiene una serie de configuraciones, como ID de canal o red, que definen su comportamiento. Estas configuraciones se identifican con dos caracteres, por ejemplo, CH para canal e ID para ID de red. Cuando desee leer o establecer cualquier configuración del módulo XBee, se debe enviar un comando AT. Cada comando AT comienza con las letras "AT" seguidas por los dos caracteres que identifican el comando que se está emitiendo y luego por algunos

valores de configuración opcionales.

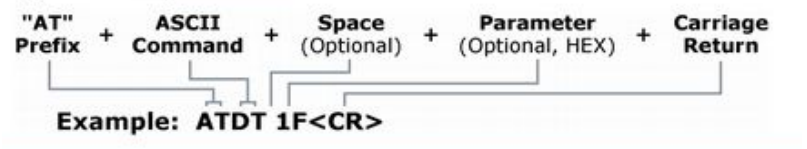



*Imagen 2: Estructura de un comando AT [1]*

Todos los comandos AT soportados por los módulos son:

- Comandos especiales

| *Comando AT* | *Nombre y descripción* |
|---|---|
| (vacío) | Comprueba la conexión con el módulo. |
| WR | Escribe la configuración actual en la memoria no volátil para que persista la próxima vez que se encienda el dispositivo. |
| WB | Write Binding Table. Escribe la tabla de enlace actual en la memoria no volátil. |
| RE | Restore Default. Restaurar los parámetros del módulo a los valores predeterminados de fábrica. |
| FR | Software Reset. Reinicia el módulo. |
| NR | Network Reset. Restablecer los parámetros de la capa de red en uno o más módulos dentro de un PAN. |

*Tabla 3: Comandos especiales*

- Direccionamiento

| *Comando AT* | *Nombre y descripción* |
|---|---|
| DH | Destination Address High. Establece u obtiene los 32 bits superiores de la dirección de destino de 64 bits. |
| DL | Destination Address Low. Establece u obtiene los 32 bits inferiores de la dirección de destino de 64 bits. |
| MY | 16-bit Network Address. Obtiene la dirección de red de 16 bits del módulo. |
| MP | 16-bit Parent Network Address. Obtiene la dirección de red principal de 16 bits del módulo. |
| NC | Number of Children. Lee la cantidad de dispositivos finales hijo que se han unido al dispositivo. |
| SH | Serial Number High. Lee los 32 bits superiores de la dirección IEEE única de 64 bits de los módulos. |

| | |
|---|---|
| SL | Serial Number Low. Lee los 32 bits inferiores de la dirección IEEE única de 64 bits de los módulos. |
| NI | Node Identifier. Almacena un identificador en un string. |
| DD | Device Type Identifier. Almacena el tipo de dispositivo XBee. |
| ZA | ZigBee Application Layer Addressing. Establece o lee el atributo de direccionamiento de la capa de aplicación ZigBee. |
| SE | Source Endpoint. Establece o lee el valor del endpoint de la capa de aplicación ZigBee. |
| DE | Destination Endpoint. Establece o lee el valor ID de destino de la capa de aplicación Zigbee. |
| CI | Cluster Identifier. Establece o lee el valor ID de clúster de la capa de aplicación Zigbee. |
| BI | Binding Table Index. Establece o lee el valor del índice de la tabla de enlace. |

***Tabla 4: Comandos de direccionamiento***

- Red y Seguridad

| *Comando AT* | *Nombre y descripción* |
|---|---|
| CH | Operating Channel. Lee el número del canal utilizado para transmitir y recibir datos entre los módulos. |
| ID | PAN ID. El coordinador establece el PAN ID; un enrutador o dispositivo final establece el PAN ID deseado. |
| BH | Broadcast Hops. Establece o lee el número máximo de saltos para cada transmisión broadcast. |
| OP | Operating PAN ID. Lee el PAN ID. |
| ND | Node Discover. Descubre e informa de todos los módulos encontrados. |
| NT | Node Discover Timeout. Establece o lee la cantidad de tiempo que un nodo pasará buscando otros nodos cuando se emite ND o DN. |
| NO | Network Discovery Options. Establece o lee el valor de las opciones para el comando de descubrimiento de red. |
| DN | Destination Node. Resuelve una cadena NI (identificador de nodo) en una dirección física (distingue entre mayúsculas y minúsculas). |
| SC | Scan Channels. Establece o lee la lista de canales para escanear |

| | |
|---|---|
| SD | Scan Duration. Establece o lee la duración del escaneo. |
| NJ | Node Join Time. Establece o lee el tiempo que un coordinador o enrutador permite que un nodo se una. |
| JV | Channel Verification. Establece o lee el parámetro de verificación del canal. |
| AR | Aggregate Routing Notification. Establece o lee el tiempo entre mensajes broadcast consecutivos. |
| AI | Association Indication. Lee la información sobre la última solicitud de unión de nodo. |

*Tabla 5: Comandos de red y seguridad*

- Seguridad

| *Comando AT* | *Nombre y descripción* |
|---|---|
| EE | Encryption Enable. Establece o lee la configuración de cifrado. |
| EO | Encryption Options. Para configurar las opciones de encriptado. |
| KY | Encryption Key. Establece la clave de cifrado AES de 128 bits. |

*Tabla 6: Comandos de seguridad*

- Interfaz de radiofrecuencia

| *Comando AT* | *Nombre y descripción* |
|---|---|
| PL | Power Level. Selecciona o lee el nivel de potencia al que el módulo transmite. |
| PM | Power Mode. Establece o lee el modo de alimentación del dispositivo. |
| DB | Received Signal Strength. Este comando informa la intensidad de la señal recibida del último paquete de datos recibido. |

*Tabla 7: Comandos del módulo de radiofrecuencia*

- Interfaz serial

| *Comando AT* | *Nombre y descripción* |
|---|---|
| AP | API Enable. Activa el modo API. |
| AO | API Options. Configura las opciones del modo API. |
| BD | Interface Data Rate. Configura o lee la velocidad de datos del serial para la comunicación entre el módulo y el host. |

| | |
|---|---|
| NB | Serial Parity. Establece o lee la configuración del bit de paridad (para detección de errores) en el módulo. |
| RO | Packetization Timeout. Establece o lee el número de tiempos de caracteres del silencio entre caracteres requeridos antes de la paquetización. |
| D7 | Establece o lee las opciones para la línea DIO7 del módulo. |
| D6 | Configura las opciones para la línea DIO6 del módulo. |

*Tabla 8: Comandos de la interfaz serial*

- Comandos de E/S

| *Comando AT* | *Nombre y descripción* |
|---|---|
| IS | Force Sample. Fuerza una lectura de todas las líneas de entrada digitales y analógicas habilitadas. |
| 1S | XBee Sensor Sample. Fuerza a que se tome una muestra de un dispositivo XBee. |
| IR | IO Sample Rate. Configura o lee periódicamente los sensores de E/S. |
| IC | IO Digital Change Detection. Configura o lee los pines de E/S digitales para monitorear los cambios en el estado de E/S. |
| P0 | Selecciona o lee la función para PWM0 |
| P1 | Configura las opciones para la línea DIO11 del módulo. |
| P2 | Configura las opciones para la línea DIO12 del módulo. |
| P3 | Configura las opciones para la línea DIO13 del módulo. Este comando no está disponible aún. |
| D0 | Selecciona o lee la función para AD0 o DIO0 |
| D1 | Selecciona o lee la función para AD1 o DIO1 |
| D2 | Selecciona o lee la función para AD2 o DIO2 |
| D3 | Selecciona o lee la función para AD3 o DIO3 |
| D4 | Selecciona o lee la función para DIO4 |
| D5 | Configura las opciones para la línea DIO5 del módulo. |
| LT | Assoc LED Blink Time. Establece o lee el tiempo de parpadeo del LED asociado. |
| D8 | DIO8 Configuration. Selecciona o lee la función para DIO8. |

| | |
|---|---|
| | Este comando no está disponible aún |
| PR | Pull-up Resistor. Configura o lee el bit que configura el resistor pull-up interno para las líneas de E/S. |
| RP | RSSI PWM Timer. Señal que se emitirá después de la última transmisión. |
| CB | Commissioning PushButton. Utilizado para simular por software el botón de puesta en marcha. |

*Tabla 9: Comandos de las E/S*

- Diagnósticos

| *Comando AT* | *Nombre y descripción* |
|---|---|
| VR | Firmware Version. Lee la versión de Firmware del módulo. |
| HV | Hardware Version. Lee la versión de Hardware del módulo |
| %V | Supply Voltage. Lee el voltaje en el pin VCC |

*Tabla 10: Comandos de diagnósticos*

- Opciones de los comandos AT

| *Comando AT* | *Nombre y descripción* |
|---|---|
| CT | Command Mode Timeout. Establece o lee el periodo de inactividad del módulo. |
| CN | Exit Command Mode. Sale del modo comando AT. |
| GT | Guard Times. Establece el periodo de silencio requerido antes y después del comando CC para evitar la entrada involuntaria en el modo comando AT. |
| CC | Command Sequence Character. Establece o lee el valor ASCII que se usará en la secuencia para entrar en el modo comando AT. |

*Tabla 11: Comandos de las opciones de los comandos AT*

- Comandos del modo Sleep

| *Comando AT* | *Nombre y descripción* |
|---|---|
| SM | Sleep Mode. Establece el módulo en modo sleep |
| SN | Number of Sleep Periods. Establece el número de períodos de inactividad para no activar el pin de Encendido/Suspensión en la activación si no hay datos de un módulo esperando el dispositivo final. |

| | |
|---|---|
| SP | Sleep Period. Tiempo que el dispositivo final permanece dormido. |
| ST | Time Before Sleep. Establece el tiempo antes de que el dispositivo final entre en modo sleep. |
| SO | Sleep Option. Configura opciones del modo sleep. |

*Tabla 12: Comandos del modo Sleep*

Los comandos AT que se han utilizado serán expuestos y explicados a lo largo del Capítulo 3.

### 2.1.6 *Frames*

Como se ha mencionado, los módulos XBee se comunican con el dispositivo inteligente y entre ellos a través de paquetes; estos paquetes de datos estructurados en modo API se denominan frames. Se envían y reciben a través de la interfaz serial del dispositivo y contienen el mensaje inalámbrico en sí, así como información adicional, como el destino u origen de los datos o la calidad de la señal.

Cuando un dispositivo está en modo API, todos los datos que entran y salen del módulo a través del serial están contenidos en frames que definen operaciones o eventos dentro del dispositivo.

| Start delimiter | Length | | Frame data | | | | | | | | Checksum |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Frame type | Data | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | n | n+1 |
| 0x7E | MSB | LSB | API frame type | Frame-type-specific data | | | | | | | Single byte |

*Imagen 3: Estructura de un frame [10]*

- Start delimiter: Secuencia especial de bits que indican que empieza un frame.
- Length: Tamaño total del frame.
- Frame data: Contiene la información recibida o que va a ser transmitida.
    - Frame type: Indica el tipo de frame y cómo está organizada la información del campo Data
    - Data: Contiene los datos del frame en sí.
- Checksum: Para verificar la integridad del frame

Para parsear estos frames utilizamos la librería xbee-api la cual explica muy bien como con dos funciones es capaz de transformar los datos. Estas dos funciones que se mostrarán en la siguientes imágenes sacadas de [6]:

```
var C = xbee_api.constants;
var xbeeAPI = new xbee_api.XBeeAPI();

// Something we might want to send to an XBee...
var frame_obj = {
  type: C.FRAME_TYPE.AT_COMMAND,
  command: "NI",
  commandParameter: [],
};
console.log(xbeeAPI.buildFrame(frame_obj));
// <Buffer 7e 00 04 08 01 4e 49 5f>
```

*Imagen 4: Función buildFrame() que crea un API frame a partir de un objeto JS*

```
// Something we might receive from an XBee...
var raw_frame = new Buffer([
    0x7E, 0x00, 0x13, 0x97, 0x55, 0x00, 0x13, 0xA2, 0x00, 0x40, 0x52, 0x2B,
    0xAA, 0x7D, 0x84, 0x53, 0x4C, 0x00, 0x40, 0x52, 0x2B, 0xAA, 0xF0
]);

console.log(xbeeAPI.parseFrame(raw_frame));
// { type: 151,
//   id: 85,
//   remote64: '0013a20040522baa',
//   remote16: '7d84',
//   command: 'SL',
//   commandStatus: 0,
//   commandData: [ 64, 82, 43, 170 ] }
```

*Imagen 5: Función parseFrame() que analiza y devuelve un objeto JS desde el búfer pasado.*

Los tipos de frames que han sido utilizados son:

- 0x88: AT Command Response (802.15.4, ZNet, ZigBee)
- 0x8A: Modem Status (802.15.4, ZNet, ZigBee)
- 0x97: Remote Command Response (802.15.4, ZNet, ZigBee)
- 0x91: ZigBee Explicit Rx Indicator (AO=1) (ZNet, ZigBee)
- 0x92: ZigBee IO Data Sample Rx Indicator (ZNet, ZigBee)
- 0x95: Node Identification Indicator (AO=0) (ZNet, ZigBee)
- 0x90: ZigBee Receive Packet (AO=0) (ZNet, ZigBee)

## 2.2 *JavaScript*

JavaScript (JS) es un lenguaje interpretado creado por *Brendan Eich* en 1995. Su objetivo principal es dotar a las webs de dinamismo, y es por esto que navegadores son capaces de de interpretar sus líneas de código. Es orientado a objetos y débilmente tipado, lo que lo hace un lenguaje potente y fácil de usar. A esto hay que añadir que es monohilo, por tanto no es permitido ejecutar más de

una operación a la vez, es decir, no permite realizar operaciones en paralelo.

A todo ello hay que añadir que es lenguajes de programación asíncrono. Por lo tanto, se basa en llamadas que pueden ser cumplidas ahora o en un futuro. Es decir, las variables pueden ser asignadas en cualquier momento de la ejecución del programa.

Para controlar la asincronía podemos usar las Promesas (*Promise*). Podemos ver las promesas como un valor que no está disponible en un momento pero que estará disponible (bien sea lo esperado o bien sea un error) en el futuro. Como es lógico, todas las operaciones dependientes de este valor serán pospuestas hasta tener el mismo. En resumen, se ejecutará nuestro código hasta llegar a una promesa; mientras se resuelve esta promesa (por ejemplo, enviar un mensaje al coordinador) se sigue ejecutando el código no dependiente de esta acción y cuando se haya resuelto la operación marcada entonces se ejecuta el código que hayamos especificado en la promesa que se debe realizar cuando tengamos el valor necesario.

Esto se puede ver mejor con un ejemplo. Supongamos que queremos realizar una suma de dos valores y a esta suma queremos restarle un tercer valor. Además, queremos enviar un mensaje de cada uno de los pasos. La resta no debería ejecutarse hasta tener el valor de la suma ya que sino los mensajes no se mostrarían en orden correcto. Sin embargo, en JS esto podría ocurrir. Si quisiéramos solucionar este problema sería tan sencillo como crear la siguiente promesa:

```
let addTwoNumbers = new Promise (() => {
	c = a + b;
	console.log("La suma de a + b es: " + c);
});

addTwoNumbers.then( () => {
	e = c - d;
	console.log("La resta de c - d es: " + e);
});
```

Además, como el tema de las promesas es algo muy utilizado, en la especificación del lenguaje ECMAScript 2017 se introdujo una forma sencilla de realizar el mismo trabajo que con las promesas, las palabras claves *async* y *await*.

La palabra clave *async* le dice al compilador de JavaScript que trate la función de manera diferente. El compilador se detiene cada vez que llega a la palabra clave *await* dentro de esa función. Asume que la expresión, después de esperar, devuelve una promesa y aguanta hasta que la promesa se resuelva o se rechace antes de seguir adelante. Por poner el mismo ejemplo anterior, con async y await se resolvería de la siguiente manera:

```
let addTwoNumbers = function(a, b) {
	return a + b;
});
```

```
let subtractTwoNumbers = function(c, d) {
	return c - d;
});

let result = async function() {
	let c = await addTwoNumbers(a, b);
	console.log("La suma de a + b es: " + c);
	let e = subtractTwoNumbers(c, d);
	console.log("La resta de c - d es: " + e);
}
```

Aunque en este caso concreto parezca más complicado este método que el anterior, en códigos más complejos es al contrario.

## 2.3 NodeJS

Tal y como se indica en su página oficial [4], NodeJS es: *"un entorno de ejecución para JavaScript"*, es decir, es una librería que nos permite ejecutar JS no solo en los navegadores sino también en los servidores. Entre sus puntos fuertes destaca que es dirigido por evento, asíncrono y permite crear sistemas escalables.

La principal ventaja que ofrece NodeJS es que permite crear servidores escalables de una manera sencilla. Además, gracias a su instalador de paquetes NPM podemos encontrar una gran variedad de librerías que facilitan el trabajo.

## 2.4 Vue JS

Framework de JS progresivo que permite crear interfaces web de usuario. Con él dotar del dinamismo que aporta JS a una web se hace de manera menos tediosa. VueJS inicialmente se enfoca en la capa visual de una aplicación web pero si se combina con otras librerías y herramientas es capaz de crear aplicaciones sofisticadas.

Para este trabajo, por ejemplo, lo he utilizado con Vuetify [9] para la realización del diseño y maquetado de la página. Este framework ofrece, a partir de componentes propios, crear una visual de la interfaz sencilla y elegante basada en *Material Design*, conservando toda la potencia de Vue.

VueJS abstrae al desarrollador de la interacción con el navegador o con el DOM, es decir, en lugar de manipular los elementos referenciándolos en el DOM se definen e interactúan con ellos de forma declarativa, en un nivel superior.

VueJS es probablemente el framework de Front-End más accesible porque

se introduce fácilmente en la aplicación a través de una etiqueta de script. De hecho, VueJS se creó seleccionando las mejores ideas de otros frameworks como Angular, React y Knockout, y seleccionando las mejores opciones de cada uno de ellos. El ejemplo más sencillo de VueJS es

```
<html>
 <body>
   <div id="example">
     <p>{{ msg }}</p>
   </div>

   <script src="https://unpkg.com/vue"></script>
   <script>
     new Vue({
          el: '#example',
            data: { msg: 'Hello World!' }
     })
   </script>
 </body>
</html>
```

En el se importa Vue mediante un CDN y se crea una instancia con su constructor (new Vue). Con " *el: '#example'* ", le estamos especificando que la instancia sea montada en aquel elemento con id *example*. Por último, definimos los datos que va a tener la instancia, los cuales pueden ser renderizados con la sintaxis {{nombreDelDato}}. En este ejemplo, al cargar el HTML en el navegador veremos *Hello World!*, el mensaje dentro de dato *msg*

VueJS es un framework basado en componentes. Los componentes son unidades individuales e independientes de una interfaz, los cuales pueden ser reutilizados dentro de un mismo proyecto o en proyectos distintos. Pueden tener su propio estado, etiquetas y estilo. Para crear un componente se hace uso de Vue.component. Por ejemplo, podríamos crear un componente que fuese un contador, tal y como se hace en la web oficial de Vue [11].

```
Vue.component('Button-counter', {
    data: function () {
        return {
 count: 0
    }
  },

  template:
    `<button v-on:click="count++">
        You clicked me {{ count }} times.
    </button>`
});
```

Este componente, de nombre Button-counter, es un botón (definido en el atributo *template* el cual define la plantilla del componente) en que al clickar,

establecido con el Event Listener v-on:click, ejecutar el código JS definido en este manejador de eventos, que en este caso es aumentar el dato 'count'.

Utilizar este componente en cualquier archivo HTML o incluso en otro componente (en su *template*) es tan sencillo como utilizar la sintaxis de HTML con el nombre del componente. Por ejemplo, si quisiéramos introducir este contador en un div sería tan sencillo como:

```
<div id="components-demo">
  <button-counter></button-counter>
</div>
```

Teniendo en cuenta que en este div debe estar cargada una instancia de Vue:

```
new Vue({ el: '#components-demo' })
```

## 2.5 *Webpack*

Tal y como se especifica en [7] Webpack es un módulo creado en NodeJS que genera un gráfico de dependencias que mapea cada módulo del proyecto y genera uno o más paquetes transpilados. Entendiéndose como módulo del proyecto todo aquello que se encuentra separado en diferentes ficheros dentro de nuestra aplicación, sea cual sea su extensión.

Es empleado en el desarrollo web porque da solución a dos problemas relevantes:

- Gestionar un amplio número de ficheros y
- Manejar ar la gestión las dependencias del proyecto y sus acoplamientos.

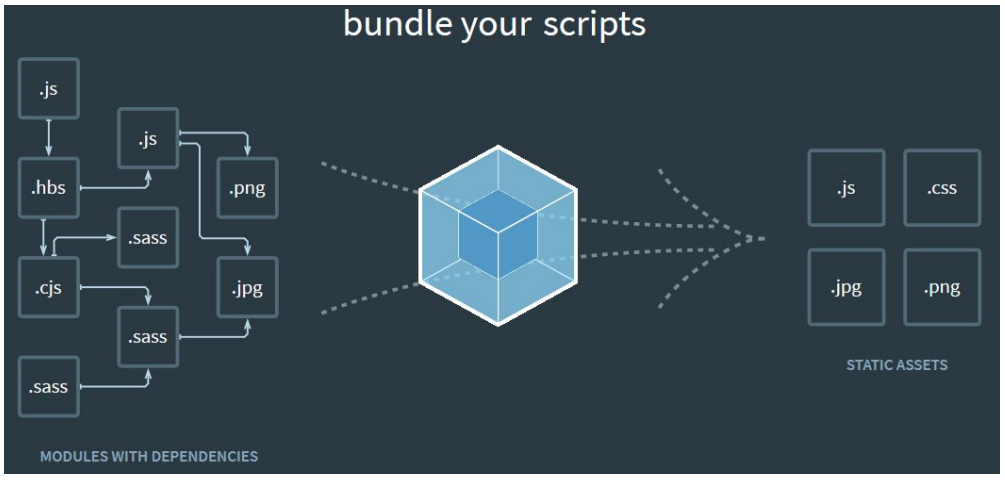



*Imagen 6: Descripción gráfica del uso de Webpack [7]*

A parte de transformar nuestro código, webpack ofrece otras funcionales adicionales como un compilador en "caliente" el cual nos permite ver las modificaciones en el código sin tener que refrescar el navegador.

# Capítulo 3 Desarrollo

## 3.1 Herramientas empleadas

En primer lugar cabe destacar que el TFG se hizo enteramente en el sistema operativo Ubuntu ya que este ofrece una mayor libertad y flexibilidad a la hora de descargar paquetes, abrir puertos para la conexión y un dominio total del sistema. Por este motivo he necesitado instalar en mi ordenador portátil una máquina virtual con el sistema operativo anteriormente señalado. La herramienta utilizada con este fin fue *Oracle VM VirtualBox.*

Como editor de código he utilizado VisualStudio Code ya que su amplia gama de extensiones ofrece al programador una gran ayuda a la hora de escribir código de cualquier lenguaje. A esto hay que añadir que ofrece funciones no solo para los programas sino también para el control de versiones o la depuración entre otros.

Con el fin de llevar una vigilancia de los progresos que se iban haciendo se utilizó la herramienta para el control de versiones *Git*. Además, para poder acceder a estos cambios desde cualquier sitio se utilizó el servicio web GitHub, con el repositorio *ULL-InformaticaIndustrial-Empotrados*/*TFG_JuanCarlos_XBee* proporcionado por el profesor Alberto Hamilton al comienzo del trabajo.

## 3.2 Progreso

### 3.2.1 Fase 1. Estudio del código y la electrónica

En primer lugar, se hizo un estudio del código brindado por el tutor para comprender el funcionamiento del módulo para NodeJS de XBee llamado *xbee-api* el cual facilita analizar y crear tramas API para para comunicar el ordenador con el coordinador y así poder dar comandos al coordinador u otros nodos.

Este código contaba con dos archivos principales. El primero, *dialogaapi.js, donde se define una clase que* se encarga de gestionar la comunicación, en modo API, entre el módulo coordinador y el dispositivo inteligente. El mismo también contaba con una tabla de direcciones para ir recordando los nodos que se han comunicado con el coordinador y con un emisor de eventos para cada tipo de mensaje utilizado. Para los mensajes más habituales se invocaban unos métodos en su recepción.

El segundo archivo es *consultaAPI.js* en el cual a través de una consola y de forma interactiva se pueden enviar comandos AT [5]. Al arrancar es necesario que se envíen ciertos comandos al coordinador, siendo la mayoría de ellos necesarios para reconocer la red y por tanto para tener la dirección de los módulos XBee. Estos comandos, extraída su información de [1], son:

- **SH** y **SL**: Para obtener los 64 bits de dirección de cada módulo XBee asignados de fábrica.
- **VR**: Para saber la versión del firmware.
- **AI**: Para obtener información sobre la última solicitud de unión al módulo XBee.
- **OP**: Lee la ID del PAN al que pertenece el módulo.
- **CH**: Lee el número del canal utilizado para transmitir y recibir entre los módulos dentro de su PAN.
- **NI**: Almacena un Identificador del Nodo en caracteres ASCII para poder ser identificados más fácilmente por los humanos.
- **ND**: Descubre y reporta todos los nodos encontrados, informándonos de los parámetros de cada nodo. De entre esta información cabe destacar como relevante la PAN y la dirección (tanto la de 64 bits como del NI) de cada nodo.

Una vez finalizado el proceso de comprensión del código inicial el siguiente paso fue familiarizarse con la electrónica. Para ello fue de gran ayuda la siguiente imagen ofrecida por el profesor referente a las conexiones de una de las dos PB que fueron dadas:

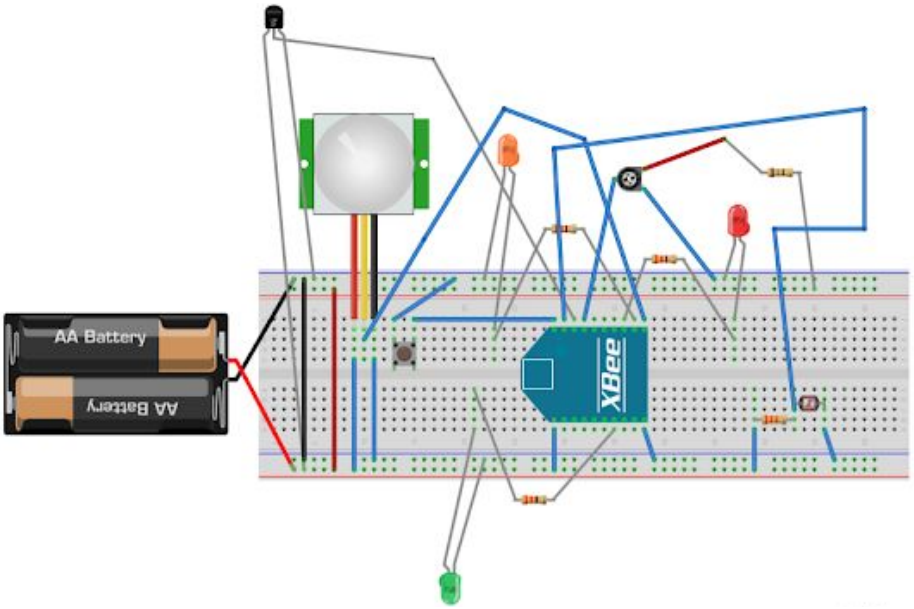



*Imagen 7: Conexiones Protoboard 1*

En la siguiente tabla se muestra la distribución de pines necesarios para este TFG y el dispositivo que hemos conectado al mismo:

| *Nº Pin* | *Comando AT* | *Opciones* | *Dispositivo Conectado* |
|---|---|---|---|
| 4 | p2 | 0, 3, 4, 5 | -No usado (PB1) |

| | | | |
|---|---|---|---|
| 6 | p0 | 0, 3, 4, 5<br>+<br>1→RSSI PWM | -Salida digital. Led rojo en el zócalo (PB1)<br><br>-Salida digital. Led verde (PB2) |
| 7 | p1 | 0, 3, 4, 5 | -Salida digital. Led verde (PB1)<br><br>-Entrada digital. Pulsador (PB2) |
| 11 | d4 | 0, 3, 4, 5 | -Entrada digital. Sensor PIR de presencia (PB1) |
| 12 | d7 | 0, 3, 4, 5<br>+<br>1→CTS flow control | -Salida digital. Led naranja (PB1) |
| 13 | / | / | -Salida digital. Led rojo indicador de No Sleep (PB1) y (PB2) |
| 15 | d5 | 0,3, 4, 5<br>+<br>1→Associated indication LED | -Salida digital. Led verde en el zócalo (PB1) y (PB2) |
| 17 | d3 | 0, 2, 3, 4, 5 | -Entrada analógica. Potenciómetro variable (PB1)<br><br>-Salida digital. Led naranja (PB2) |
| 18 | d2 | 0, 2, 3, 4, 5 | -Entrada analógica. LM35 voltaje variable con la temperatura (PB1) |
| 19 | d1 | 0, 2, 3, 4, 5 | -Entrada analógica. LDR resistencia variable con luz (PB1)<br><br>-Entrada analógica. Potenciómetro variable (PB2) |
| 20 | d0 | 0, 2, 3, 4, 5 | -Entrada digital. Pulsador (PB1)<br><br>-Entrada digital. Pulsador (PB2) |

***Tabla 13: Distribución y opciones de pines***

Como los pines pueden ser configurados de varias formas (entrada o salida analógica o digital) es necesario especificar en el comando dicha configuración. Es por ello que en la siguiente tabla se especifica qué supone cada opción:

| ***Opción*** | ***Descripción*** |
|---|---|
| ***o*** | Deshabilitado (Disabled) |

| | |
|---|---|
| 2 | Entrada analógica (Analog input) |
| 3 | Entrada digital (Digital input) |
| 4 | Salida digital, baja intensidad (Digital output, default low) |
| 5 | Salida digital, alta intensidad (Digital output, default high) |

*Tabla 14: Opciones*

### 3.2.2 Fase 2. Desarrollo de la primera versión

Adentrándonos en el desarrollo del software y con el fin de conseguir el primer objetivo tuvimos que crear el servidor el cual pudiera enviar la interfaz web para mandar comandos y recibir las peticiones. Para esto utilizamos la librería de NodeJS *Express* la cual nos permite crear una infraestructura web mínima y flexible para aplicaciones web.

El planteamiento fue crear una API REST, en la cual diferenciamos tres acciones principales: mandar comandos directamente, insertar nuevas acciones de configuración y borrar acciones de configuración ya existentes.

Sin embargo, en un primer momento solo se llevó a cabo el primer punto ya que lo que nos interesaba era comprobar que estaban llegando bien las peticiones al servidor. Para ello se utilizó el método '*POST*' de Express, el cual recibe como primer parámetro el nombre del endpoint (y posibles argumentos) y como segundo parámetro una función (llamada callback) la cual se ejecuta cuando se recibe una petición para ese endpoint.:

```
app.post("/direct_command/:command", function(req, res){
```

*Imagen 8: función post para enviar comando directos*

Con endpoint */direct_command* y pasando como parámetro el comando que se quiere ejecutar directamente (:command).

En cuanto al HTML, simplemente constaba de un input en el cual se introducía el comando y de un botón que al clickar ejecutaba la función que llamaba al endpoint del servidor.

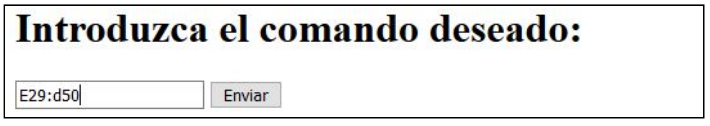



*Imagen 9: HTML inicial*

Con el trabajo realizado hasta ese momento el primer objetivo se cumplía, pero lo interesante de este TFG residía en el objetivo número dos, ya que el usuario final debía:

- ser ajeno al pin al que estaba conectada cada electrónica y
- poder configurar acciones automáticas más allá de simplemente cambiar el estado de un output pulsando un botón.

Adentrados en este punto, lo primero que se debía hacer era crear un modelo de datos conveniente para que, guardados en archivos de configuración, fueran útiles al fin expuesto en el párrafo anterior. Para ello, se crearon cuatro archivos de configuración.

El archivo de configuración *pins.config.json* contiene un objeto JS en el cual se recoge, dividido por entrada o salida, el input/output del que se trata, su comando para ser modificado y el nombre natural dado (sustituyendo los espacios por guión bajo para simplicidad a la hora de tratar los datos).

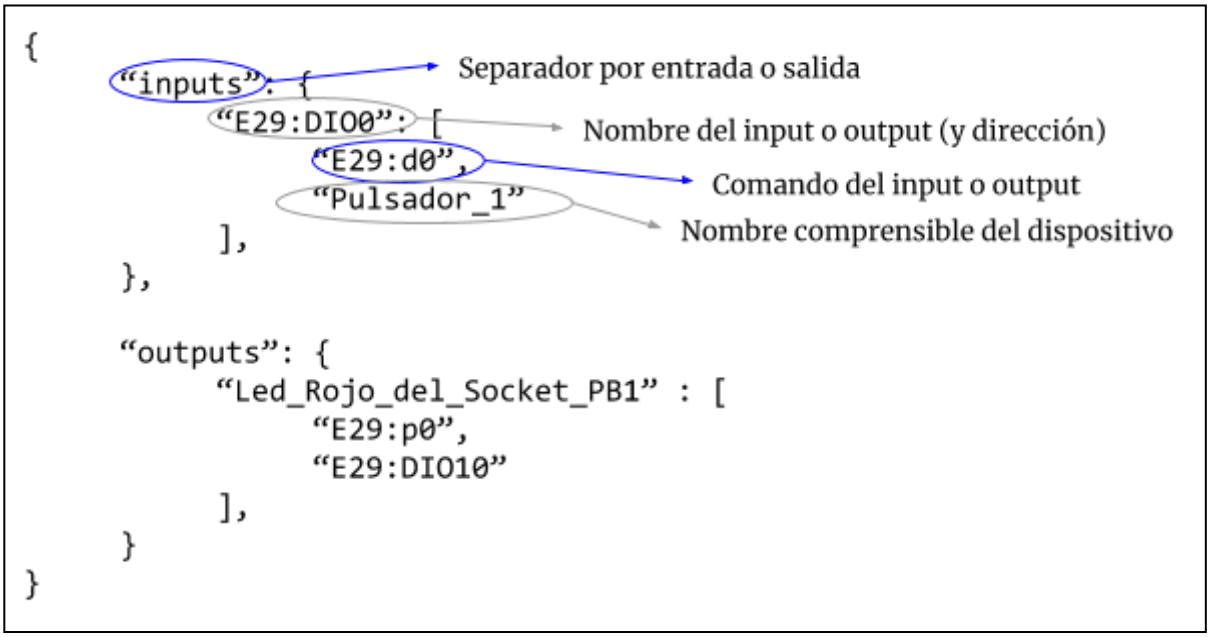



*Imagen 10: Contenido de ejemplo del archivo pins.config.json*

El archivo *options.json* contiene un objeto JS con las opciones que aceptan cada uno de los inputs. En este caso no hizo falta añadir el de los outputs porque para todos ellos siempre existen tres opciones, forzar encendido, forzar apagado y cambiar estado; es por ello que estas opciones se introdujeron directamente en la vista del Front-End donde se iban a tratar estos datos.

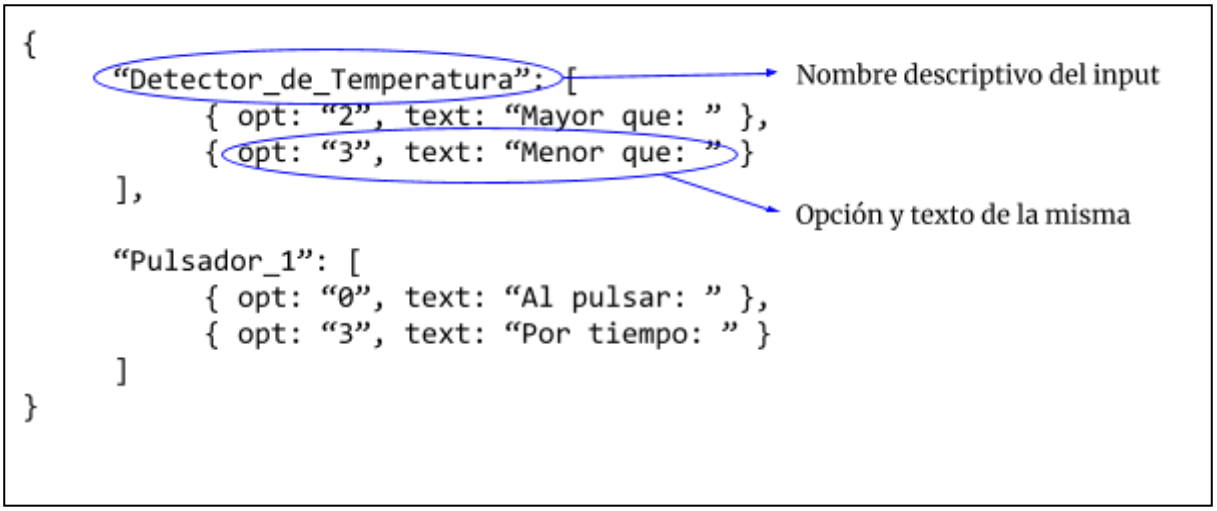



*Imagen 11: Contenido de ejemplo del archivo options.json*

El archivo *initial.conf.json* contiene un objeto JS el cual guarda el estado inicial de toda la electrónica con el objetivo de que cuando se arranque el servicio los dispositivos se encuentren en el estado correcto. Por ello, la opción debe estar dentro del dominio marcado por las opciones de cada pin establecidas en la tabla 2. También se ha añadido la key "*others*" la cual es otro objeto JS en el cual se guardan otros comando iniciales con sus opciones.

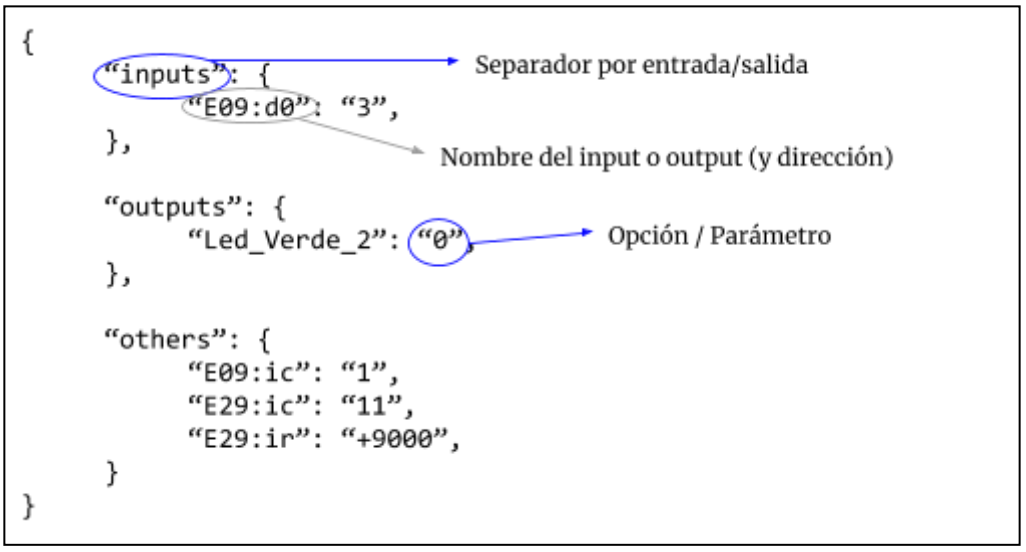



***Imagen 12: Contenido de ejemplo del archivo initial.conf.json***

Destacar los dos comando que aparecen en "*others*":

- *IC*: Detección de Cambio Digital. Este comando se utiliza para monitorear los cambios en pines establecidos como Inputs / Outputs digitales, aunque en el contexto de este TFG solo hace falta monitorear las entradas. Este comando recibe como parámetro un valor en hexadecimal en el cual se especifica a qué pines se les debe hacer un seguimiento.
- *IR*: Frecuencia de Muestreo. Este comando envía cada cierto tiempo el valor medido por las electrónicas establecidas como Inputs / Outputs tanto digitales como analógicas. Recibe como parámetro, tras el símbolo +, un número el cual expresa en milisegundos el intervalo de tiempo en el que queremos que nos envíe la información.

Por último, el archivo *actions.json* contiene un objeto JS en el cual se guarda toda la información de las configuraciones, siguiendo el orden de *input→parámetro→ output→acción*.

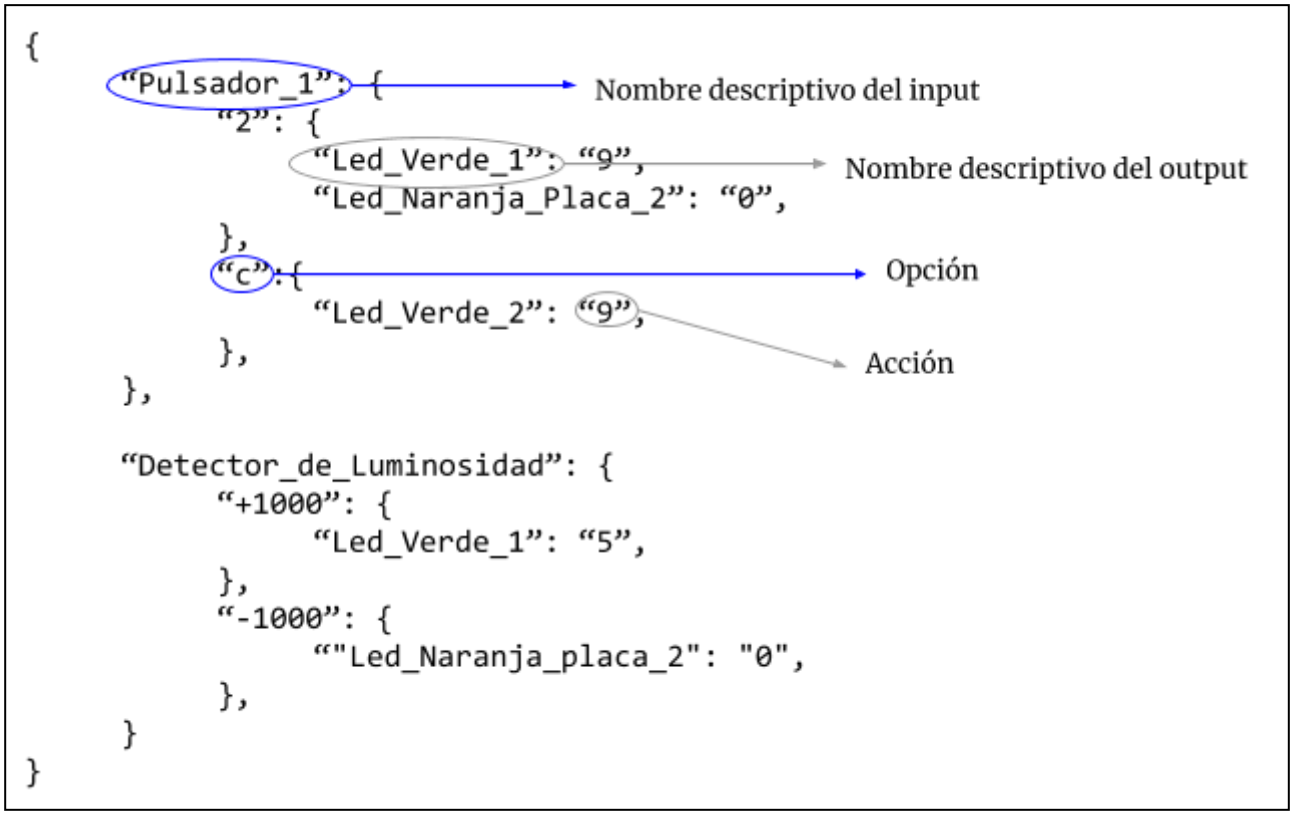



*Imagen 13: Contenido de ejemplo del archivo actions.json*

Para explicar un poco más en profundidad este archivo, comentar los distintos parámetros del mismo.

Opción:

- *"c"*: Hace referencia a cuando el input cambia. Por ejemplo, en el caso del Pulsador_1 tanto cuando se presiona como cuando se suelta. Válido solo en inputs digitales.
- *"+INT"*: Hace referencia a que la acción se ejecute cuando el valor medido sea mayor que *INT*. En el caso de la imagen, cuando el Detector_de_Luminosidad registre un valor superior a 1000 debe ejercer una acción sobre el Led_Verde_1. Válido solo en inputs analógicos.
- *"-INT"*: El mismo caso que el punto anterior pero esta vez cuando el valor medido sea menor que *INT*. Válido solo en inputs analógicos.
- *"INT"*: Hace referencia a los segundo indicados como tiempo. Por ejemplo, en el caso de la imagen con el Pulsador_1 cuando este haya sido pulsado alrededor de 2 segundos se ejercerá tanto una acción sobre Led_Verde_1 como sobre Led_Naranja_Placa_2. Válido solo en inputs digitales.

Acción:

- "0":Forzar apagado del output.
- "5":Forzar encendido del output.
- "9":Cambiar estado, es decir, si está apagado encenderlo y viceversa.

### *3.2.3 Fase 3. Permitiendo a los usuarios realizar configuraciones*

En un primer momento todas las configuraciones se introdujeron a mano para poder centrarnos en la elaboración del algoritmo que permitiera, a partir de estas configuraciones y los datos obtenidos de la red, funcionar realmente como un servicio domótico. Sin embargo, tal y como se comentó anteriormente, la idea inicial fue permitir a los usuarios crear y borrar acciones. Para ello, se crearon dos nuevos endpoints en el servidor, utilizando nuevamente el método *POST*. Con él se construyó:

- Insertar acciones:

  ```
  app.post("/insertAction/:selected_input/:si_param/:selected_output/:so_param",
  function(req, res){
  ```

  En este caso el endpoint es */insertAction* y recibe cuatro parámetros. El primero (:selected_input) hace referencia a la entrada que va a ejercer la acción y el segundo con qué parámetro (:si_param), teniendo en cuenta que los parámetros pueden ser "Por tiempo", "Mayor que", "Menor que" y al cambiar de estado, que se refleja de diferentes formas según el input seleccionado. Los dos últimos parámetros hacen referencia al output. Con el tercero (:selected_output) se especifica la salida la cual se va a ver afectada y con el último (:so_param) qué acción se quiere ejercer sobre ese output, forzar su encendido, forzar su apagado o cambiar su estado.
- Borrar acciones:

  ```
  app.post("/deleteAction/:device/:parameter", function(req, res){
  ```

  Con endpoint /deleteAction recibe dos parámetros, el primero (:device) el cual contiene la entrada a la cual se le va a borrar la acción. El segundo parámetro (:parameter) posee la opción la cual se quiere borrar.

En cuanto al algoritmo empleado para hacer que la electrónica siguiera las órdenes establecidas comentar que:

- Se captura el evento emitido por una recepción remota de un Input o Output: ' `da.on("recepcionIORemota", fr => {` ' guardando en fr el contenido del mensaje recibido.

- Una vez hecho esto se comprueba el tipo del mensaje, ya que todos aquello que traen consigo la modificación de un input o output pertenecen al tipo 146. Seguidamente lo que se hizo fue comprobar si el paquete recibido pertenecía al comando IC o al comando IR; esto se hizo de una manera simple y es que el comando IC, por defecto, devuelve vacío el campo analogSamples, el cual es un objeto JS con el sensor y su valor. En este punto el algoritmo difiere un poco dependiendo del comando que envía el paquete.

- Si el paquete fue enviado por el comando IC:
  - En una variable guardamos tanto el nodo como el input que sufrió un cambio. Esto lo averiguamos porque guardamos el estado anterior de todos los inputs, por tanto recorriendo ambos objetos podemos comprobar cual cambió.
  - En el momento en el que se activa un input comprobamos:

        - Si es un input que tiene acciones “al cambio” → se ejecutan dichas acciones.
        - Si es un input que tiene acciones por tiempo → se inicia un timer.
        - Cuando vuelve a llegar otro paquete del comando IC analizamos si es del input anterior o no y:
        - Si es distinto input, iniciamos otro timer para ese input.
        - Si es el mismo input entonces finalizamos el timer, calculamos el tiempo, buscamos la opción más cercana a ese tiempo y ejecutamos las acciones que se requerían para esa opción.

- Si el paquete fue enviado por el comando IR comprobamos el valor obtenido de todos los sensores establecidos como inputs analógicos y para cada uno de ellos:
    - Comprobamos si con el valor medido debemos ejecutar alguna acción porque sea mayor o menor a una configuración establecida.

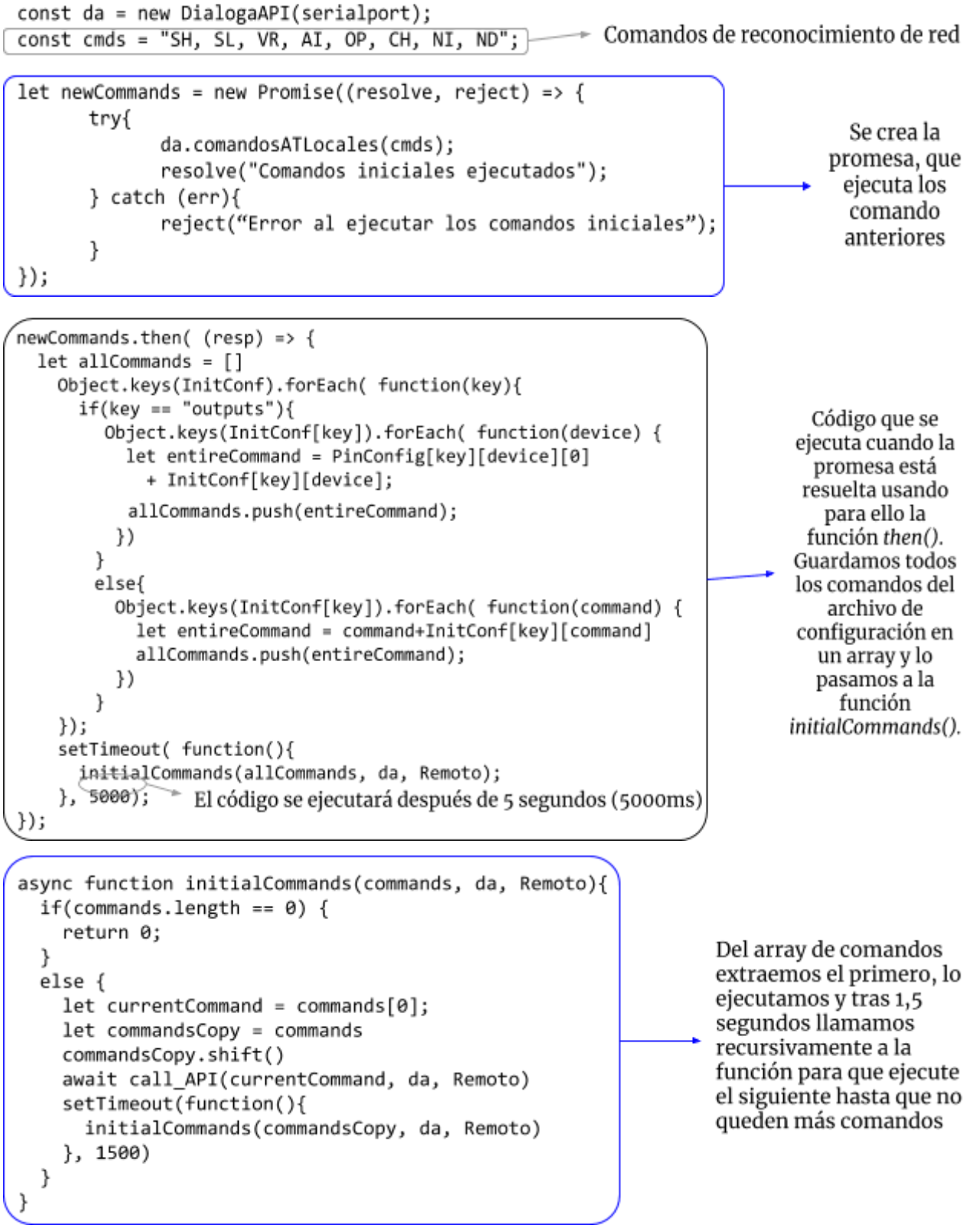
### *3.2.4 Fase 4. Comandos iniciales*

Para finalizar con la parte del servidor encargado de tratar los datos obtenidos de la red de sensores era necesario ejecutar unos comandos iniciales. Esto es debido a que al iniciar el servicio por primera vez o tras reiniciarlo los módulos no saben cómo está configurado cada uno de sus pines (como entrada, salida u otra opción especial). Además, también en necesario indicar que queremos recibir los datos de los sensores a través del comando IR e IC.

Los primeros comandos necesarios son los que se comentó al inicio de este apartado para el reconocimiento y direccionamiento de la red (SH, SL, VR, AI, OP, CH, NI y ND). Tras ellos, es necesario establecer cada pin con su configuración e invocar los comandos para recibir los datos de los sensores. Toda esta configuración es la que se estableció en el archivo *initial.config.json*. Por tanto, lo que debemos hacer es recorrer este archivo e ir ejecutando las configuraciones que ahí se hallan.

Sin embargo, esto no fue tan simple debido a la asincronía del lenguaje JS. Para poder ejecutar estas configuraciones fue necesario recurrir a las promesas (*Promise)* de este lenguaje.

Combinando las promesas, la recursividad y la función *setTimeout()* de JS, la cual ejecuta cierto código tras un tiempo que se le pasa como parámetro, logramos ejecutar en orden correcto y sin pérdida de paquetes los comandos necesarios.




***Imágenes 14, 15 y 16: Explicación del código para ejecutar los comandos iniciales***

La razón por la cual se espera 1,5 segundos entre comando y comando es para no llenar el buffer de los módulos XBee, ya que si esto pasa entonces empieza a desechar paquetes y no se ejecutan los comandos correctamente.

### *3.2.5 Fase 5. Diseño de la interfaz web*

En cuanto a la interfaz web, tal y como se comentó previamente se empezó con un archivo HTML muy básico el cual servía el propio Express. Pero esto solo fue hasta conseguir el primer objetivo. Una vez alcanzado el mismo se empezó con el framework VueJS.

Con esta herramienta creamos una SPA (Single Page Application), es decir, solo tenemos una página, en este caso index.html, la cual carga un componente, *App.vue*, en el que las distintas vistas se renderizan en lugar de cargar nuevas páginas desde el servidor.

Con esta idea clara, lo primero fue definir las diferentes vistas que queríamos tener. Tras una fase breve de análisis se llegó a la conclusión que eran necesarias al menos tres vistas, las cuales fueron creadas y que se explicarán a continuación:

- Vista *Home.vue*: Es la primera que se muestra y en ella se ofrecen todos los outputs conectados a la red para poder ver y cambiar su estado.


***Imagen 17: Vista Home***

- Vista *Rules.vue*: Donde se crean las "reglas". Es en esta vista donde se especifican las configuraciones que se quieren introducir. Consta de tres elementos esenciales; el primero que se encarga de recopilar los datos referentes al input. El segundo hace lo mismo pero en relación al output y el tercero es un botón el cual llama al endpoint en el servidor */insertAction* para introducir la configuración.

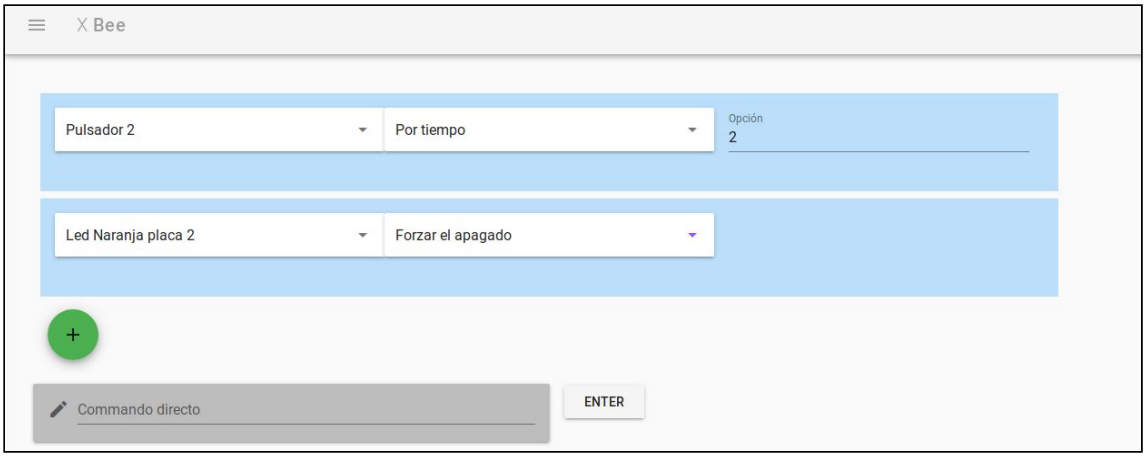



***Imagen 18: Vista Rules***

- Vista *Actions.vue*: Donde se visualizan y borran las acciones existente. En esta vista se pueden observar todas las configuraciones que hay, las cuales se obtienen cargando directamente el archivo *actions.json*. Estas configuraciones se dividen por inputs y cada acción se puede eliminar por separado llamando al endpoint */deleteAction.*

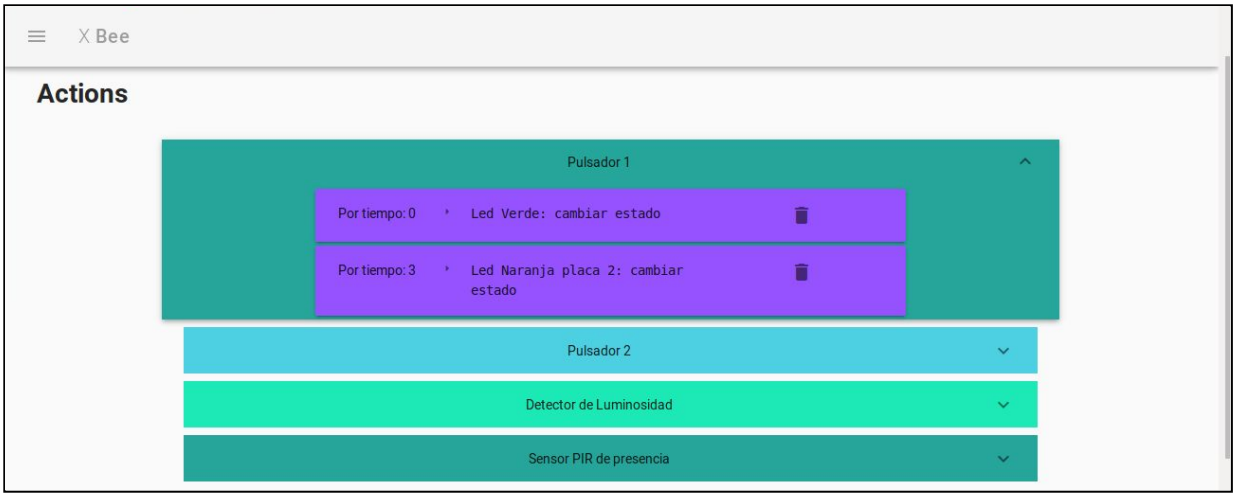



***Imagen 19: Vista Actions***

Además de las vistas, se crearon dos componentes, los cuales se encuentran en *App.vue* y que por tanto aparecen en todas las vistas:

- Componente *Nav.vue*: Carga un navbar, el cual permite navegar por las distintas vistas.

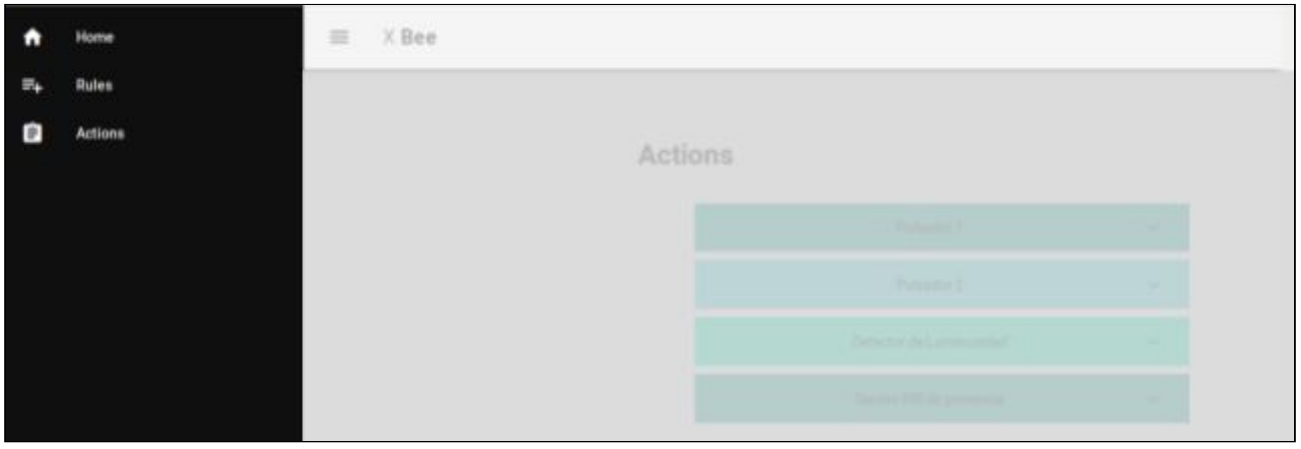



***Imagen 20: Componente Nav***

- Componente *Footer.vue*: Donde se encuentra el copyright del proyecto.


***Imagen 21: Componente Footer***

Para poder cambiar de vistas de forma dinámica se utilizó el paquete *Vue-Router*, el cual nos permitió de manera sencilla realizar el enrutamiento entre las vistas.

Para hacer que toda la interfaz funcione y sea renderizada por el navegador es necesario transpilar el código de los archivos de Vue a JS. Para ello utilizamos la herramienta webpack y una de sus funcionales adicionales, devServer.

Esta funcionalidad, devServer, como se describe en su documentación oficial [8], es usada para "*desarrollar rápidamente una aplicación*". En el contexto de este TFG utilicé esta característica de webpack para crear un pequeño servidor que se encargara de servir la interfaz web, separando de esta forma Front-End y Back-End para ofrecer un servicio mejor ya que, aunque en este caso están en la misma máquina, se podría desplegar en dos máquinas distintas y así resolver las peticiones de manera más eficiente. Siendo así porque el usuario se conectaría al servidor que le brinda la interfaz y es la interfaz la que se comunica con el servidor que realiza todo el procesamiento de datos.

Para conectar ambos servidores se usó la librería *Axios* la cual ofrece un cliente HTTP basado en promesas para el navegador y NodeJS. Es decir, nos permite crear promesas que ejecutan peticiones HTTP. En el caso concreto del TFG se usó para llamar a los endpoints *insertAction*, *deleteAction* y *direct_command*.

Por último, mencionar que gracias a devServer y Vuetify fue posible diseñar una interfaz responsiva para que pueda ser visualizada tanto en dispositivos de gran tamaño como en aquellos más pequeños.

## 3.3 Alcance físico

Uno de las dudas más interesantes de los módulos XBee era averiguar si realmente cumplían con las especificaciones de alcance que se mencionan en sus especificaciones técnicas.

En esta se comenta que el rango con obstáculos es de 40 metros, mientras que el rango en línea es de 120 metros. Quisimos averiguar si esto era cierto y se hizo una prueba dentro de un hogar obteniendo los siguientes resultados con matices:

| | |
|---|---|
| ***Rango con obstáculos (Urban Range)*** | 32 metros |
| ***Rango en línea (Line Range)*** | 20 metros |

***Tabla 15: Alcance medido***

En cuanto al rango con obstáculos comentar que se hizo entre dos habitaciones separadas por tres paredes por las cuales circulan circuitos

eléctricos y tuberías de agua.

En cuanto al rango en línea sí que se debe destacar que es tan reducido en comparación a los metros de las especificaciones técnicas debido a que esta era la mayor distancia que se podía conseguir dentro del hogar sin obstáculos.

Comentar también que las pruebas se hicieron únicamente con dos nodos, el coordinador y un dispositivo final.

## 3.4 Problemas detectados y Propuesta de Solución

El primer problema encontrado fue a la hora de entender el código inicial. Esto fue debido a que la documentación con respecto a la librería xbee-api, más allá de la que proporciona el autor, es un tanto insuficiente. La solución vino al leer detenidamente el código para saber qué hacía cada parte del mismo, apoyado por la documentación oficial de la librería [6].

La segunda dificultad vino de la mano de la electrónica y es que pocas han sido las bases que he tenido en este campo. Sin embargo, gracias a la ayuda ofrecida por el profesor pude sortear esta dificultad sin problemas.

El siguiente obstáculo fue a la hora de crear los archivos de configuración. Esto fue motivado porque debía no solo plantear un modelo que tuviera una estructura de datos adecuada para ser tratada por el servidor, sino también con una forma sencilla y coherente para ofrecer al Front-End la información y así brindar una vista adecuada. La solución fue, tras mucha reflexión y errores con modelos fallidos, la planteada en el apartado 3.3 con los diversos archivos de configuración.

Tras estos problemas, tocó lidiar con las promesas. Como se comentó en el apartado anterior, debíamos ejecutar una serie de comandos en cierto orden y dejando un margen de tiempo entre uno y otro. A esto hay que añadirle que la ejecución de un comando trae consigo una respuesta, por tanto, no consistía únicamente en enviar dichos comandos uno tras otro, porque de esa forma la respuesta podría no llegar por un desbordamiento del buffer en el módulo XBee que envía las respuestas, que en este caso sería el coordinador. Este margen debía ajustarse bien ya que:

- Un tiempo muy corto haría que el buffer del coordinador empezara a recibir respuestas que tiene que enviar a distintos módulos finales, llegando un momento en el que se produjera un desbordamiento y se empezaran a descartar paquetes, haciendo entonces que el coordinador no enviara cierta configuración a los dispositivos finales sobre sus pines.
- Un tiempo muy elevado haría que el servicio tardara mucho en arrancar, sobre todo en redes con muchas electrónicas conectadas.

Este problema se solucionó con muchas pruebas, tanto a la hora de realizar los primeros comandos como a la hora de ejecutar la configuración inicial de cada

pin. El resultado final fue dejar un margen de 5 segundos entre que se envían los comando de reconocimiento de la red y los comandos de configuración de los pines para poder recibir las respuestas de los primeros. Luego, dejar entre comandos de configuración un tiempo de 1,5 segundos.

Cambiando al entorno del Front-End, el primer problema encontrado fue a la hora de mostrar los distintos outputs en la vista *Rule*. Esto se debía a la forma del modelo de datos existentes para los outputs y el elemento de Vuetify utilizado para listar la electrónica. La solución fue cambiar el modelo de datos; de ahí que en los archivos *pins.config.json* y *initial.conf.json* tengas un estilo distinto para los inputs que para los outputs.

El siguiente problema fue en la vista *Action* ya que para aquellas acciones en las que se veían afectados más de un output la parte visual era bastante inaceptable. Esto era resultado de un desorden en el texto además de salir el mismo del contenedor en el que estaba. La solución fue aplicar un estilo propio para el elemento HTML en el que se encapsulaban cada una de las consecuencias de una opción.

# *Capítulo 4* Conclusiones

## *4.1 Objetivos logrados*

Una vez finalizado el TFG, de los objetivos planteados al comienzo del mismo se han conseguido la mayoría de ellos.

Se consiguió hacer un diseño básico a través de una interfaz web creando un servidor sencillo con Express.

Posteriormente se evolucionó la aplicación añadiendo los archivos de configuración, abstrayendo al usuario final del nivel físico del servicio y permitiéndole crear y borrar acciones automáticas para influir en los outputs.

Una vez conseguido esto se mejoró el aspecto visual de la interfaz para dar una experiencia más satisfactoria al usuario.

Por último, se logró que la interfaz pudiera ser vista desde cualquier dispositivo para dar una mayor libertad a la hora de acceder a la web.

Como conclusión adicional mencionar que se verificó que el alcance proporcionado en las especificaciones técnicas se acerca bastante al real, al menos para rangos con obstáculos.

## *4.2 Summary and Conclusions*

As summary, we can see that with XBee modules we could create a sensor network in an simple and highly configurable way. We divided the service in two different servers.

On the one hand, on Back-End side we used NodeJS, with a set of libraries and *Promises*, that allow us create an efficient daemon. This daemon is able to manage the received packages and to send packages to the correct XBee module.

On the other hand, the Front-End side. In this field which we started from scratch we utilized VueJS in order to make the creation of the web interface easier. With this tool we achieved to elaborate several views. Each view represent a functionality of the service, create and delete configurations and do a direct

action thanks to *Home* view.

In conclusion, from de aims that were proposed at the beginning of the assignment th ones which were achieved are:

- The basic design to send commands through a web interfaces.
- Add the configuration files. This point helps to abstract users from the physical layer. With these files users could create and delete actions automatically in order to influence in outputs.
- Make that the interface could be seen from any devices in order to give users more freedom for access the web.

In addition, we could see that the *urban Range* specified in the guide [1] is very similar to the urban range that we could measure.

The aim that we could not achieve is make different users class with different permissions. In addition, although it was not an aim, another point that could be improved is the *SLEEP* mode of XBee modules. With that mode the service would be better because it reduce the electric consumption, reducing electrical cost too.

## 4.3 Líneas futuras

Como aspecto más relevante a mejorar destacaría la seguridad tanto interna como externa.

Como seguridad interna me refiero al objetivo planteado y no logrado de realizar una separación de autorizaciones de usuarios. Tal y como se explicó en el apartado de objetivos si se llevara a un entorno real puede que no todos los miembros de una casa deban tener los mismos permisos a todos los elementos de la red.

Como seguridad externa hago referencia a que no se debería permitir que una persona ajena realice configuraciones. Para poder acceder desde todos los dispositivos lo que se debe hacer es conectarse a la ip y puerto correcto del servidor de la interfaz web, de momento sin ningún tipo de autenticación; esto crea el problema de que si una persona externa consigue entrar en la red en la que está el servidor podría realizar cambios en las configuraciones y electrónicas de forma ilícita.

Otro punto a realizar en un futuro sería la configuración en modo SLEEP. De este modo los módulos XBee entran en “reposo” durante una cantidad de tiempo determinada. De esta manera el consumo es aún menor, por tanto el tiempo de duración de baterías (en caso de estar conectados a una) y coste energético sería inferior.

## 4.4 *Valoración personal*

Mi elección por este TFG fue debido a que combinaba dos campos que tienen todo mi interés, la domótica y el desarrollo web.

Aunque si bien es cierto que el inicio del proyecto me costó un poco al no estar familiarizado con los módulos XBee y la electrónica no es menos verídico que disfruté mucho realizando este trabajo. Esto fue debido al ver resultados rápidos y eficaces con cada cambio que hacía. Tanto si se trataba de enviar comando como de crear vistas el resultado es inmediato ya que en el primer caso veía la electrónica reaccionar (en la mayoría de los casos encendiendo LEDs) y en el segundo caso al renderizar la página el navegador.

Además, debido a los problemas encontrados he mejorado como programador web, sobre todo en el lenguaje JavaScript y en el uso de frameworks como VueJS.

# Capítulo 5 Presupuesto

| *Concepto* | *Cantidad* | *Coste unidad* | *Coste total* |
|---|---|---|---|
| Módulo XBee | 3 | 27,99€ | 83,97€ |
| Protoboard (+jumpers) | 2 | 2,95€ | 5,90€ |
| Led | 6 | 0,20€ | 1,20€ |
| Resistencias | 7 | 0,10€ | 0,70€ |
| Pulsador | 3 | 0,84€ | 2,52€ |
| Potenciómetros variables | 2 | 0,44€ | 0,88€ |
| Sensor PIR de presencia | 1 | 3,30€ | 3,30€ |
| LDR resistencia variable con la luz | 1 | 0,38€ | 0,38€ |
| LM35 voltaje variable con la temperatura | 1 | 2,64€ | 2,64€ |
| Programador | 180 horas | 8,33 €/hora | 1500€ * |
| ***Coste total:*** | | | ***1601,49€*** |

***Tabla 16: Presupuesto***

* El presupuesto de 1500€ de gastos de programador se ha calculado tras visionar diversas webs y ofertas de trabajo real, y concluyendo un salario bruto anual de 21000€ en 14 pagas. Como el tiempo dedicado al TFG fue de aproximadamente 20 horas semanales se supuso un sueldo de media jornada, por tanto de 10500€ brutos anuales en 14 pagas. Ya que la dedicación fue de dos meses se llegó al presupuesto de 1500€. En las horas de trabajo sólo están contempladas aquellas dedicadas al desarrollo del software, sin contar la elaboración de la presente memoria.

# *Bibliografía*

[1]: Product Manual. XBee® ZNet 2.5/XBee-PRO® ZNet 2.5 OEM RF Modules, © 2012 Digi International, Inc. All rights reserved. 23rd January, 2012

[2]: *ZigBee Alliance*

[3]: Digi International Inc. *How XBee devices communicate*

[4]: *NodeJS*

[5]: *Comandos Hayes (comandos AT)*

[6]: *Librería de JavaScript xbee-api*

[7]: *Webpack Documentation*

[8]: *Webpack devServer*

[9]: *Vuetify Documentation*

*[10]: API frame*

*[11]: Vue Components*