Proyecto Fin de Grado
Ingeniería de las Tecnologías de Telecomunicación

Sistema IoT basado en el control y monitoreo de una red ZigBee. Aplicación en servicio domótico.

Autor: Francisco José Velasco Iglesias
Tutor: José Manuel Fornés Rumbao

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Proyecto Fin de Grado
Ingeniería de las Tecnologías de Telecomunicación

Sistema IoT basado en el control y monitoreo de una red ZigBee. Aplicación en servicio domótico.

Autor:
Francisco José Velasco Iglesias

Tutor:
José Manuel Fornés Rumbao
Profesor titular

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Proyecto Fin de Carrera: Sistema IoT basado en el control y monitoreo de una red ZigBee. Aplicación en servicio domótico.

Autor: Francisco José Velasco Iglesias

Tutor: José Manuel Fornés Rumbao

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mis abuelos

Agradecimientos

La finalización y entrega de este trabajo de fin de grado, supone para mí la culminación de un largo y duro periodo de investigación, infinitas pruebas y gran incertidumbre. Pero también tremendamente satisfactorio.

Un periodo cuyos inicios se remontan a principios de 2020, cuando recorrí con mi tutor José Manuel Fornés las azoteas de la ETSI en busca de los nodos sensores que debía implementar en mi anterior trabajo. Por aquellas fechas, no sabíamos aún lo que pocos meses después se nos vendría encima. Fue en pleno confinamiento, cuando me reuní por videollamada con José Manuel para consensuar el nuevo enfoque que debía de darle al proyecto, ya que el anterior era completamente inviable debido a la situación que atravesábamos.

Me gustaría por tanto, dar las gracias en primer lugar a mi tutor por su paciencia y atención. Agradecer por supuesto a mi familia, mis padres y mis hermanas, el incondicional apoyo. A mis compañeros (ya amigos) Juan, Pablo, Romera y Jose, con los que he luchado sin parar contra viento y marea en todos estos años de carrera. Sin la ayuda de cada uno de ellos, todo habría sido mucho más difícil.

Por último, acordarme también de mis abuelos, a los que dedico este proyecto y cuyo sueño era verme acabar la carrera. No me cabe duda de que lo están viendo desde alguna parte. Nadie creyó más en mí que ellos, y ojalá estén tremendamente orgullosos.

Francisco José Velasco Iglesias

Madrid, 2021

Resumen

En los últimos años, la domótica se ha consolidado como uno de los principales sectores de inversión por parte de las grandes empresas así como de estudio para muchos ingenieros y desarrolladores. La continua evolución y aumento de la accesibilidad que está experimentando dicha tecnología, está contribuyendo al incremento de la seguridad, confortabilidad y el ahorro energético de muchas viviendas, lo que se traduce en una mejor calidad de vida para cada vez más usuarios, con mención destacada para aquellos con necesidades especiales y/o personas de avanzada edad.

En este proyecto se plantea una solución alternativa a la actual oferta del mercado, la cual estará conformada por algunas de las tecnologías hardware y software más punteras del momento. Se buscará aplicar las mejores prácticas consideradas dentro del campo de la domótica, así como integrar el mayor número de funcionalidades posibles, las cuales serán puestas a prueba en un entorno real.

Por último, cabe destacar que dicho sistema se podría aplicar perfectamente, con las correspondientes adaptaciones, en otros ámbitos similares como oficinas, hospitales, centros de mayores, hoteles o cualquier tipo de inmueble (inmótica) e incluso en ambientes exteriores.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

1.1 Objetivos

La exponencial evolución del Internet de las cosas (IoT) en general y del sector de la domótica en particular, ha permitido dar respuesta a las nuevas tendencias que están surgiendo en una sociedad cada vez más digitalizada, en la que las últimas tecnologías están más presentes que nunca en todos los ámbitos de nuestras vidas. La automatización y el control inteligente de la vivienda o domótica, es por lo tanto un excelente contexto en el que aplicar los últimos avances tecnológicos, hecho que podemos ver reflejado en la amplia variedad de ofertas disponibles que aportan soluciones de todo tipo y para cualquier vivienda.

La principal motivación de este proyecto no es otra que la de introducir una nueva solución domótica alternativa a las ya existentes en el mercado. Está basado en una red de sensores (WSN) que interactuarán entre sí de manera inalámbrica y que estarán distribuidos por toda la vivienda. Cada nodo de dicha WSN, corresponde con un elemento domotizado de la casa (tales como luces, persianas, ventiladores…) y obedecen a las órdenes que se le indiquen de manera remota mediante una aplicación móvil.

De manera más específica, podríamos sintetizar los objetivos principales del proyecto en los siguientes puntos:

- Implementación de algunas de las soluciones hardware y software más punteras para lograr la funcionalidad propuesta.
- Cumplimiento de los principios básicos de la domótica como servicio contribuyente a la mejora de la calidad de vida del usuario:
    - Fomentar el ahorro energético.
    - Manejo simple e intuitivo (accesibilidad).
    - Aporte de seguridad frente a accidentes e intrusos.
    - Confort.
- Funcionamiento remoto del servicio, por lo que no será necesario situarse en el interior la vivienda para controlar/monitorizar los elementos que formen parte del sistema.
- Coexistencia del sistema con las formas tradicionales de control de los elementos del hogar, de manera que puedan ser usados alternativamente por los propios habitantes, invitados (especialmente, ya que no dispondrán de la aplicación móvil) y en caso de caída del servicio o de internet.
- Posibilidad de adaptación del sistema en otros ámbitos similares como oficinas, hospitales, centros de mayores, hoteles e incluso en ambientes exteriores.

1.2 Estructura del documento

A continuación, se detallará el contenido de los diferentes capítulos que forman este documento:

- Capítulo 1 – Introducción

  En este capítulo, se desarrolla una contextualización del proyecto, explicando los motivos y objetivos que se persiguen en el mismo.

- Capítulo 2 – Estado del arte

  Se definen los principales conceptos teóricos y tecnologías que se aplican en el proyecto y son necesarios para comprender este documento.

- Capítulo 3 – Caso de estudio

  Se describe *grosso modo* el funcionamiento del sistema domótico, así como el escenario en el que se integra.

- Capítulo 4 – Pliego de condiciones

  Se detallan las soluciones hardware y software necesarias para poner en marcha el proyecto. Así mismo, en él se analizan y comparan dichas soluciones con algunas alternativas existentes en el mercado, justificando los criterios que han llevado a su elección.

- Capítulo 5 – Arquitectura e implementación hardware del sistema

  Se describe la manera en la que se han implementado las soluciones hardware expuestas en el *Pliego de condiciones* para conformar los distintos elementos, así como el papel que desempeñan cada uno de ellos en el sistema.

- Capítulo 6 – Descripción del sistema a nivel software

  Se profundizará en los distintos lenguajes, librerías, entornos y protocolos aplicados, para lograr el correcto funcionamiento de cada elemento integrado en el escenario propuesto.

- Capítulo 7 – Evaluación final y propuestas de ampliaciones y mejoras

  Breve comentario sobre la fase en la que se encuentra el sistema domótico a la hora de la entrega el proyecto y valoración de las posibles ampliaciones y mejoras a introducir en el futuro.

- Capítulo 8 – Manual de usuario

  Serie de instrucciones a seguir en cuanto a preparación y montaje del escenario, para replicar el funcionamiento mostrado en el vídeo de prueba.

2 ESTADO DEL ARTE

La creciente obsesión por tratar de digitalizar y conectar a internet el mayor número de objetos cotidianos posibles, ha dado lugar al surgimiento de lo que hoy día conocemos como Internet de las Cosas (*Internet of the Things*, IoT).

Como resultado de esta fuerte tendencia, han surgido algunos conceptos directamente relacionados como las denominadas Redes Inalámbricas de Sensores (*Wireless sensor networks*, WSN), que consisten en un grupo de nodos autónomos especialmente distribuidos que intercambian información entre sí y/o con un nodo central de manera inalámbrica. ZigBee es uno de los principales estándares de comunicaciones inalámbricas que se usan para dar soporte a estas WSN.

Las WSN tienen su origen en iniciativas militares, no obstante, gracias al gran potencial de estas redes, actualmente su uso más extendido lo encontramos en campos como la industria, el sector sanitario, la automoción, el control de la infraestructura urbana o la domótica. Este proyecto, se centra en este último ámbito.

Con el fin de comprender la situación tecnológica, en este capítulo se analizan los siguientes términos:

- Internet de las Cosas.
- Redes Inalámbricas de Sensores.
- Domótica.
- La tecnología ZigBee.

2.1 Internet de las Cosas

Internet está considerado como una de las creaciones más relevantes de la historia de la humanidad debido a su impacto en el modo de vida de la sociedad y su uso se viene extendiendo exponencialmente durante los últimos años.

El concepto Internet de las Cosas [1] [2] se refiere a la implementación de internet en una colección de objetos ilimitados permanentemente conectados en un escenario digital, permitiendo la interacción entre ellos y con las personas. El término coloquial “cosas”, hace hincapié en la idea de tratar de conectar el mayor número de dispositivos y objetos que forman parte de todas las actividades diarias y profesionales de la vida cotidiana.

El desarrollo del Internet de las Cosas representa sin duda la evolución de Internet, y a grandes rasgos, supone un notable cambio en la actual manera de entender la realidad que nos rodea. La extensión de su uso está propiciando un incremento en cuanto a la fiabilidad, la sostenibilidad y la eficiencia de los sistemas, infraestructuras y aplicaciones que incorporan esta tecnología.

Sin embargo, son varias las barreras que amenazan con retrasar el desarrollo del IoT, como el establecimiento de un conjunto de normas comunes, la falta de recursos, la pandemia que nos afecta o la necesidad de resolver ciertas diferencias políticas. No obstante, mientras que las entidades correspondientes sean capaces de resolver dichas dificultades, la irrupción del IoT en el panorama global se producirá más pronto que tarde y supondrá sin duda un cambio de paradigma a nivel mundial.

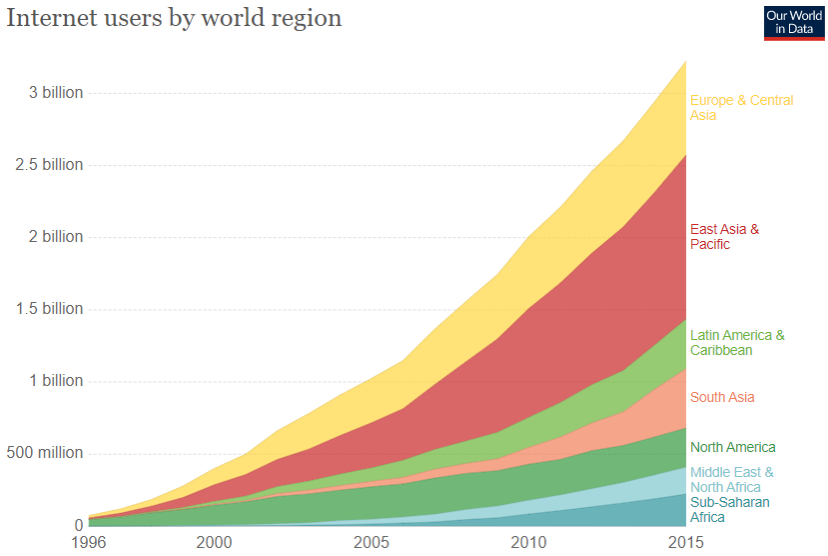

Figura 1. Usuarios de internet en el mundo. *Fuente: ITU-D.*

2.1.1 Origen

Las raíces del concepto IoT se remontan al año 1999, con el surgimiento de un grupo perteneciente al Instituto de Tecnología de Massachusetts (MIT), el cual realizaba investigaciones sobre la identificación por radiofrecuencia en red (RFID) y las tecnologías de sensores emergentes [3].

No obstante, no fue hasta el 2009, cuando el profesor Kevin Ashton del MIT, lo introdujo al gran público en su artículo *That "Internet of Things" Thing*, del que merece la pena citar este pequeño extracto traducido:

"Somos entes físicos, como también lo es nuestro entorno. Nuestra economía, sociedad y supervivencia no se basan en ideas o información, se basan en cosas (…). Necesitamos dotar a los dispositivos de sus propios medios para recopilar información, para que puedan ver, oír y oler el mundo por sí mismos."

Kevin Ashton, 2009

2.1.2 Características

El extenso rango de posibles aplicaciones de la tecnología IoT, hace que las funcionalidades específicas de los sistemas puedan diferir bastante según el contexto en los que trabajen. No obstante, existen una serie de rasgos comunes que comparten la mayoría de dichos sistemas a nivel de comportamiento:

Sensibilidad, conectividad e interacción

Es fundamental la implementación de funcionalidades que permitan la comunicación inalámbrica en tiempo real entre dispositivos, tales como la identificación, la toma de medidas, el almacenamiento y las actividades relacionadas con la interconexión.

Inteligencia

La clara tendencia es que los sistemas basados en IoT sean lo menos determinista posible y estén formados por entidades con cierto grado de inteligencia artificial que sean capaces actuar de manera autónoma según las circunstancias.

Seguridad

Resulta de vital importancia la incorporación de medidas de seguridad que protejan y garanticen la integridad y privacidad de la elevada cantidad de datos que son continuamente intercambiados y almacenados.

2.1.3 Aplicaciones

Transporte

La aplicación del IoT concierne a todos los aspectos relacionados con el campo del transporte, ya sean vehículos, mobiliario o conductores y peatones, mediante la interacción dinámica entre ellos. La comunicación intervehicular, el control del tráfico, los parkings inteligentes o la asistencia en carretera, son algunos ejemplos.

Figura 2. IoT aplicado al transporte.

Industria

Se conoce como IIoT (*Industrial Internet of the Things*) a todos aquellos sistemas implementados en el ámbito de la producción industrial que, combinados con la tecnología operacional, permiten conectar, monitorizar y regular maquinaria, localizar personal o automatizar procesos.

Agricultura

La toma de datos sobre la temperatura, lluvia, humedad, velocidad del viento o incidencia solar, puede ser usada para automatizar numerosos procesos en el ámbito de la agricultura, lo cual puede contribuir a mejorar la cantidad y calidad de las cosechas.

Medicina

La tecnología IoT puede utilizarse también en el campo sanitario para el rastreo remoto de pacientes y sistemas de notificación de emergencias. Estos dispositivos pueden variar desde monitores de presión sanguínea y control de pulsaciones, hasta equipamiento capaz de controlar implantes especializados, como marcapasos, pulseras electrónicas o audífonos. Estos sofisticados dispositivos pueden ser configurados para asegurar que el paciente tenga un soporte adecuado sin necesidad de atención continua por parte del personal de enfermería.

Tecnología militar

El IoMT (*Internet of Military Things*) es la aplicación de la tecnología IoT en el dominio miliar para propósitos de tareas de reconocimiento, supervivencia y otros objetivos de combate relacionados.

Domótica

Los dispositivos IoT pueden ser usados para la automatización y el control de los elementos mecánicos, eléctricos y electrónicos presentes en todo tipo de edificaciones y estancias públicas y privadas. La domótica será desarrollada en un apartado posterior, ya que es uno de los temas centrales de este proyecto.

2.1.4 Retos y limitaciones

A través del desarrollo de los diferentes sistemas basados en la tecnología IoT, han ido surgiendo numerosos retos a enfrentar por ingenieros y desarrolladores.

Centrándonos en la parte puramente técnica del IoT y dejando de lado el resto de los problemas políticos o burocráticos, cabe señalar su fuerte dependencia del desarrollo de otras tecnologías, como la identificación por

radiofrecuencia en red (RFID) o las redes inalámbricas de sensores (WSN), como el principal problema que se plantea. Sin entrar en detalle en este apartado, dichas tecnologías tienen como objetivo crear redes de nodos inalámbricos dotados de cierta inteligencia y capacidad de adaptación, lo cual les permiten ofrecer numerosas ventajas en materia de reducción de costes, flexibilidad de funcionamiento, facilidad de instalación y reemplazo…

Como retos más generales a resolver, podríamos citar los siguientes:

- Disponibilidad de internet.
- Problemas de seguridad.
- Consumo de potencia.
- Escalabilidad.
- Aceptación por parte de la sociedad.

Atendiendo a este último punto, es importante destacar que el IoT debe ser considerado y abordado como un fenómeno global, y como tal debe ser tratado para su óptimo desarrollo durante los próximos años.

2.1.5 Futuro

Como ya se ha comentado, el uso del Internet de las Cosas abarca todo tipo de sectores y es aplicable a cualquier área. Actualmente, la tecnología IoT se encuentra presente de forma dispersa en redes y sistemas independientes y específicos para distintos fines.

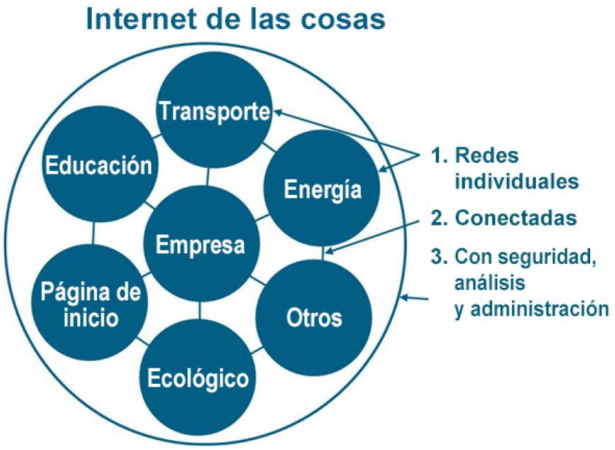

Figura 3. El IoT como red de redes.

Sin embargo, en un futuro no muy lejano, dispondremos de una cantidad mucho mayor de objetos conectados, gracias a la acelerada evolución de esta tecnología. A medida que este fenómeno se vaya produciendo, los sistemas aislados irán conectándose entre sí, incorporando las necesarias capacidades de seguridad, análisis y administración. Es entonces cuando empezaremos a ver el Internet de las Cosas como un fenómeno global, una red de redes que supondrá una poderosa herramienta de progreso para la humanidad.

"El Internet de las Cosas tiene el potencial de cambiar el mundo, tal como lo hizo el Internet. Tal vez incluso más."

Kevin Ashton, 2009

2.2 Redes Inalámbricas de Sensores

Las redes inalámbricas de sensores juegan un papel fundamental en el desarrollo del Internet de las Cosas. Como se ha comentado anteriormente, el IoT concibe un mundo de interconexión global donde cada vez son más numerosos los dispositivos u objetos que se van incorporando a esta gran red de redes. En este sentido, las redes de sensores inalámbricos [4] posibilitan el desarrollo, la escalabilidad y la expansión de dichas redes inteligentes de bajo costo y fácil implementación que se integran en el concepto de IoT para traer nuevas experiencias en las actividades de la vida diaria y en el resto de los ámbitos descritos en el apartado anterior.

Las WSN han recibido gran atención en los últimos años desde el punto de vista académico e industrial gracias a los avances tecnológicos relacionados con los microsensores, las comunicaciones inalámbricas y el procesamiento de dispositivos embebidos.

2.2.1 Elementos de una WSN

Los elementos fundamentales de los que constan cualquier WSN son los siguientes: dispositivos finales o nodos, nodo coordinador o pasarela y servidor.

Los nodos (también conocidos como "motas") son numerosos y cuentan con la capacidad de comunicarse entre ellos y con la pasarela gracias al uso de protocolos como Modbus, LoRaWAN, Bluetooth o Zigbee. Según su papel en la red, podemos clasificarlos en nodos sensores y nodos retransmisores o routers. Los nodos sensores se encargan de recolectar datos y enviarlos a los routers o a otros nodos sensores cuya función puede ser doble. Por su parte, estos últimos reenviarán la información a la pasarela. En definitiva, la función a grandes rasgos de los nodos o motas consiste en conectar el mundo real con el mundo digital, recolectando y transmitiendo inalámbricamente información de todo tipo.

La pasarela o nodo coordinador puede considerarse el dispositivo maestro de la red. Se encuentra conectado a internet o a otro tipo de redes y otorga a la WSN funcionalidades como el preprocesamiento y análisis de datos y la interacción con el usuario para la gestión del sistema.

Por último, es habitual que cada WSN tenga asociada un servidor con su correspondiente base de datos en la que almacenar los numerosos datos que son manejados.

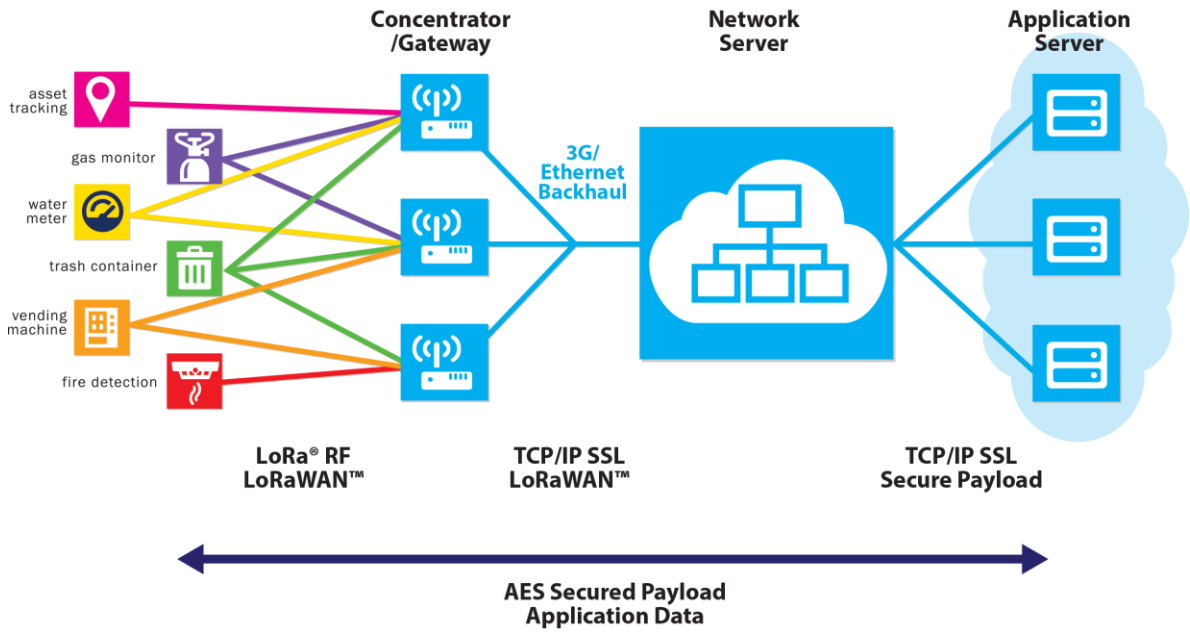

Figura 4. Arquitectura de una red WSN basada en el protocolo LoRaWAN.

2.2.2 Estructura de un nodo

Cada nodo, independientemente de su funcionalidad, se compone de una serie de elementos comunes. Dichos componentes son los sensores, el microcontrolador, el transceptor, la memoria y la fuente de alimentación.

Los sensores (pueden ser múltiples) captan una señal analógica que es digitalizada por un convertidor analógico-digital y enviada al microcontrolador. Este último se encarga de procesarla y controlar el resto de elementos que actuarán en consecuencia de la señal recibida. Los datos que representan la señal digitalizada pueden ser almacenados en la memoria y/o reenviados a otro nodo por medio del transceptor.

Los nodos pueden incorporar también una serie de elementos opcionales según su localización o función en la WSN como generadores de energía (normalmente paneles solares), sistemas de geolocalización, movilidad, etc.

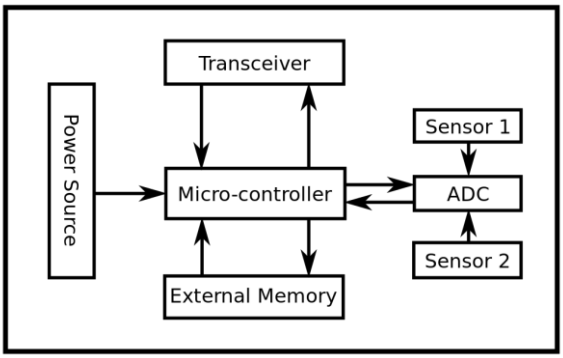

Figura 5. Arquitectura típica de un nodo sensor.

2.2.3 Características

Las WSN comparten una serie de características comunes:

- Tolerancia a errores y a condiciones ambientales desfavorables.
- Capacidad de los nodos de autogestionar cierto tipo de fallos.
- Topología dinámica y capacidad de adaptación frente a cambios.
- Consumo energético mínimo.
- Heterogeneidad y redundancia.
- Escalabilidad.
- Uso de técnicas de optimización *cross-layer*.
- Computación en tiempo real.

2.3 Domótica

Se entiende por domótica [5] al conjunto de dispositivos y técnicas que permiten integrar cierto grado de automatismos e inteligencia a las viviendas con el fin de mejorar la calidad de vida de sus habitantes en materia de seguridad, eficiencia energética, comunicaciones y confort.

2.3.1 Historia y evolución

El término domótica proviene del latín *domus* (casa) y *tica* (automática) que unidas significa literalmente “casa automática”.

El origen de la domótica se remota a la década de los 70, cuando aparecieron los primeros dispositivos de automatización de edificios basados tecnología X-10. Los primeros sistemas comerciales fueron instalados sobre todo en Estados Unidos y se limitaban a la regulación de la temperatura ambiente en edificios de oficinas.

Más tarde, tras el auge de los PC (*Personal Computer*), a finales de la década de los 80 y principios de los 90, se empezaron a incorporar los sistemas de cableado estructurado en oficinas, para facilitar la conexión de todo tipo de terminales y periféricos entre sí, utilizando un cableado estándar y tomas repartidas por todo el edificio.

Posteriormente, los automatismos destinados a edificios de oficinas junto con otros específicos, se fueron aplicando también a las viviendas de particulares, donde el número de necesidades a cubrir es mucho más amplio, dando lugar a la vivienda domótica.

2.3.2 Servicios y aplicaciones

Los servicios ofrecidos por la domótica se pueden aplicar principalmente en los siguientes aspectos:

Ahorro Energético

Aparte del uso de aparatos de bajo consumo en el hogar, resulta interesante el empleo de sistemas que posibiliten una gestión más eficiente de los mismos. Para ello, la domótica permite utilizar dispositivos temporizados, sensores y elementos programables mediante los cuales automatizar y controlar (apagar/encender, abrir/cerrar y regular) elementos domésticos como luces, climatización, puertas y ventanas, cerraduras…

Comunicación

La gestión de la comunicación se encarga del intercambio de información entre los habitantes y los equipos o aparatos domotizados de la propia vivienda, y de ésta con el exterior.

Entre sus principales funciones destacan:

- Control y monitoreo remoto de la instalación domótica mediante la red local privada y/o internet.
- Transmisión de alarmas activadas a centrales de seguridad, llamadas telefónicas y/o alertas por SMS o email.
- Intercomunicación interior de todos los servicios electrónicos del hogar como citofonía al televisor, portero automático al teléfono, difusión de audio y video, etc.
- Utilización del protocolo IP para el control de las redes domóticas y transmisión segura de datos sobre las mismas mediante internet, recurriendo a técnicas de encriptación y autenticación para el acceso remoto a dicha información, asegurando así la privacidad, seguridad e integridad.

Seguridad

Existen dos factores básicos que se deben tener en cuenta a la hora de garantizar la total seguridad de una vivienda domotizada: la seguridad personal y la seguridad del patrimonio presente en la misma. Los controles de intrusión, el cierre automático de aberturas, el empleo de cámaras de vigilancia, alarmas personales, sensores de incendios, fugas de gas, inundaciones, fallos del suministro eléctrico… son algunos de los métodos más empleados para evitar cualquier tipo de incidencia que atenten contra la seguridad de personas, animales y bienes que se encuentren en los hogares.

Confort y accesibilidad

La suma de los beneficios ya descritos proporciona comodidades que pueden mejorar notablemente la calidad de vida de los usuarios, sobre todo la de aquellos con necesidades especiales debido a una avanzada edad o a algún tipo de discapacidad. En definitiva, uno de los principales objetivos de la domótica es la mejora del bienestar de este grupo de personas, incentivando su independencia y aportando una mayor autonomía a su día a día.

2.3.3 Conclusiones

En la actualidad, la domótica se ha convertido en una tecnología que está al alcance de cada vez más hogares y avanza diariamente a pasos agigantados.

A menudo, va de la mano de otras tecnologías punteras como el IoT y las WSN, así como del uso de dispositivos móviles, los cuales son utilizados como unidades de control del propio sistema domótico. Esto hace que las tareas diarias sean más cómodas, versátiles, seguras, eficientes y, sobre todo, más accesibles para aquellas personas con dificultades o problemas de motricidad, cognitivos, o algún tipo de minusvalía.

2.4 La tecnología ZigBee

La tecnología ZigBee [6] [7] abarca un conjunto de protocolos que definen algunas de las especificaciones más usadas para dar soporte a las WSN. Está especialmente orientado a aplicaciones que precisen de comunicaciones inalámbricas seguras, con una baja tasa de datos y una maximización de la vida útil de sus baterías, permitiendo a su vez la monitorización y control remotos de los dispositivos que la forman.

2.4.1 ZigBee y el estándar IEEE 802.15.4

El estándar IEEE 802.15.4 fue desarrollado por el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) y, por otra parte, la organización responsable del protocolo ZigBee, es la ZigBee Alliance. Son por tanto dos estándares distintos; mientras que IEEE 802.15.4 define las capas físicas y de control de acceso al medio (MAC), ZigBee hace uso de esta última capa, definiendo las de red, seguridad y aplicación.

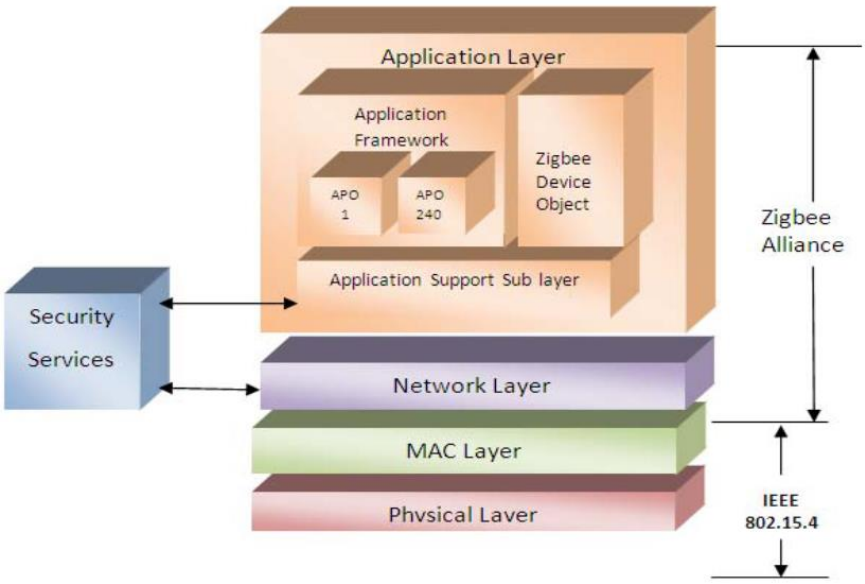

Figura 6. Torre de protocolos de IEEE 802.15.4 y ZigBee.

Como se puede observar en la *Figura 6*, la arquitectura de protocolos se basa en el sistema OSI (*Open system interconnection*).

Capa física

La capa física es la más cercana al hardware y mediante la cual, se controla la comunicación con el transceptor radio. Permite, por lo tanto, la administración del acceso al hardware ZigBee, incluyendo la inicialización del mismo o la selección del canal según la previa estimación de la calidad de señal en los canales disponibles. Puede trabajar en tres bandas de frecuencias distintas: 868 MHz (para Europa), 915 MHz (en Norteamérica y Australia) y 2.4 GHz (disponible en todo el mundo).

Física	Banda de frecuencia	Número de canales	Parámetros de propagación		Parámetros de los datos	
			Velocidad de procesamiento	Modulación	Bits por segundo	Volumen de información
868/915 MHz	868-868.6 MHZ	1	300 kchip/s	BPSK	20 kb/s	20 kbaud
	902-928 MHz	10	600 kchip/s	BPSK	40 kb/s	40 kbaud
2.4 GHz	2.4-2.4835 GHz	16	2.0 Mchip/s	O-QPSK	250 kb/s	62.5 kbaud

Tabla 1. Características técnicas de las bandas de frecuencia usadas por el IEEE 802.15.4. *Fuente:* [8].

Capa MAC

Ofrece a las capas superiores dos tipos de servicio:

- MAC data service.
- MAC management service.

Gracias a ellos, se permite la transmisión y recepción de unidades de datos (PDU) a través de la capa física, así como la necesaria sincronización de dispositivos.

Capa de red

Es la capa responsable de las tareas de enrutamiento. Así mismo, otorga seguridad y optimización de consumo a las aplicaciones ZigBee que ocupan la capa inmediatamente superior. Se definen tres topologías básicas de red (en estrella, en árbol y en malla), las cuales serán desarrolladas en un apartado posterior.

Capa de aplicación

Es la capa más alta de la tecnología ZigBee y su función básica es la de albergar los objetos de aplicación o parámetros en cada dispositivo de la red. Se puede configurar una amplia variedad de parámetros en cada dispositivo, lo que les otorga un alto grado de versatilidad y personalización.

Ofrece también la opción de utilizar perfiles para el desarrollo de aplicaciones, lo cual permite la interoperabilidad entre productos desarrollados por diferentes fabricantes que implementen el protocolo para sus comunicaciones. Esto es, que si dos dispositivos desarrollados por distintos fabricantes incluyen el mismo perfil de usuario, éstos serán capaces de conectarse e interactuar entre ellos como si ambos pertenecieses al mismo proveedor. La propia ZigBee Alliance define los principales perfiles de usuarios, aunque éstos también pueden ser desarrollados por fabricantes privados y puestos al servicio del resto de usuarios.

2.4.2 Topologías de una red ZigBee y tipos de nodos

Los dispositivos que conforman las redes ZigBee, se distribuyen sobre la misma en forma de nodos o motas. Según el rol que desempeñen, se definen nodos finales, enrutadores o coordinadores, siguiendo evidentemente, la línea descrita anteriormente en el apartado dedicado a las WSN.

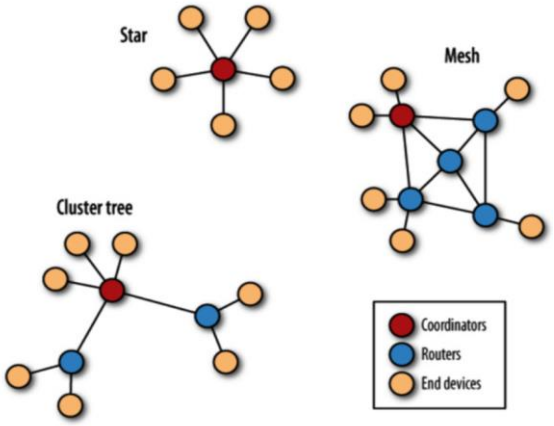

Figura 7. Tipos de topologías de una red ZigBee.

Los tipos de topologías de red soportados por la tecnología ZigBee son:

Topología en estrella

En la topología en estrella, se establece la comunicación entre cada nodo final y un único coordinador central, por lo que todas las comunicaciones pasan necesariamente por este último nodo (los dispositivos finales no podrán interactuar entre ellos de forma directa). El nodo coordinador a menudo está permanentemente conectado a la red eléctrica, mientras que el resto de nodos también pueden ser inalámbricos y estar alimentados por baterías.

Topología en árbol

Esta topología está compuesta por un nodo coordinador raíz, del que pueden partir tanto nodos finales como routers, de los cuales, a su vez pueden nacer múltiples nodos finales, formando una estructura en árbol. Los dispositivos finales dependen directamente de su nodo padre (ya sea un router o el coordinador), de manera que si este falla o se deshabilita, todos sus nodos hijos se verán afectados.

Topología en malla

La topología en malla, a diferencia de las otras dos, permite que todos de sus nodos se comuniquen directamente entre sí, siempre y cuando estén en rango y hayan establecido una conexión. Es la topología más compleja de mantener, a la vez que la más robusta y tolerante a fallos.

2.4.3 ¿Por qué ZigBee?

El nombre de ZigBee tiene su origen en el símil que representa la forma de transmitir información entre cada uno de los nodos, con el característico zigzagueo una abeja (*bee* en inglés), que va de flor en flor dejando polen.

Es uno de los protocolos que mejor se adapta a las necesidades de una WSN gracias a sus características:

- Fiabilidad y tolerancia ante fallos.
- Facilidad de despligue.
- Seguridad.
- Bajo coste.
- Interoperabilidad y escalabilidad.
- Maximización de la vida útil de las baterías.
- Gran número de nodos soportados.
- Estándar *open source* con alto grado de personalización.
- Baja tasa de datos.

2.4.4 Alternativas a ZigBee

Wi-Fi

El estándar WiFi posee una alta tasa de transferencia de datos, lo que posibilita el intercambio de mayor cantidad de datos en menor tiempo. Es por ello que el consumo energético es mucho más elevado. Su aplicación más común es la de proporcionar conexión inalámbrica a internet.

Una de las grandes desventajas a las que se enfrenta actualmente WiFi es la seguridad. Un elevado porcentaje de este tipo de redes, son instaladas de forma abierta debido a la simplicidad que supone su implementación. Muchas de estas redes no cumplen con los estándares de seguridad, incluso algunas de ellas son desplegadas a propósito por *hackers* con el objetivo de robar información de los usuarios que se conecten a ellas.

Bluetooth

Al igual que WiFi, Bluetooth posee una tasa de transferencia de datos mayor que ZigBee, así como, por contrapartida, un consumo mayor de potencia en las comunicaciones. Su rango de cobertura no es demasiado alto y está orientado a eliminar la necesidad de cables para la comunicación entre dispositivos electrónicos y accesorios, como puede ser el caso de un teléfono móvil y unos auriculares inalámbricos. Además, su tasa de datos es suficiente para permitir el intercambio de ficheros.

LoRaWAN

El estándar LoRaWan es, sin duda, la alternativa más similar a ZigBee. Está orientado a aplicaciones IoT más sencillas, debido a su menor tasa de datos, consumo de potencia, coste de implementación…

Su característica más destacada es su alto rango de cobertura (de ahí su nombre, *Long Range*).

	Wi-Fi	**Bluetooth**	**LoRaWAN**	**Zigbee**
Banda de frecuencia	2.4GHz y 5GHz	2.4 GHz	subGHz (868/915 MHz en Europa)	868/915 MHz y 2.4 GHz
Tasa de datos	Hasta 54 Mbps	1 Mbps	Hasta 50 kbps	Hasta 250 kbps
Número de canales	14/25	79	10	1/10/16
Rango de cobertura	100 m	10 m	10 Km	100 m
Topología de red	estrella	red de dispersión	estrella	estrella, malla o árbol
Requisitos de alimentación	Altos - Horas de Batería	Medios - Días de Batería	Muy Bajos - Años de Batería	Muy Bajos - Años de Batería
Modulación	BPSK, QPSK, COFDM, CCK y M-QAM	GFSK	CSS	BPSK y O-QPSK
Costes	altos	muy bajos	muy bajos	bajos

Tabla 2. Comparativa técnica de algunos estándares de comunicaciones inalámbricas alternativos a ZigBee.

2.4.5 ZigBee y la Domótica

El campo en el que se prevé que esta tecnología cobre más fuerza, es precisamente la domótica. Vendría a reemplazar a los sensores y actuadores basados en tecnologías obsoletas y que se instalan aisladamente en las viviendas, con el objetivo de dar soporte a toda una red domótica centralizada mucho más moderna y eficaz.

La tecnología ZigBee se está imponiendo en el mercado, siendo cada vez mayor el número de fabricantes que

la implementa y se está convirtiendo en la base de la domótica moderna. Una nueva domótica versátil y fácil de instalar y de hacer crecer progresivamente gracias al uso de dispositivos baratos, sencillos, inalámbricos y de rápida integración actuando al unísono bajo el escenario de una única red sensorial común.

3 CASO DE ESTUDIO

Como ya se ha introducido, el gran objetivo de este proyecto es la aplicación del IoT en un sistema domótico. Para ello, se ha hecho uso de una red inalámbrica de sensores (WSN) basada en la tecnología ZigBee.

La WSN planteada, está formada por varios nodos o dispositivos finales y un nodo coordinador, que hará de pasarela entre dicho ecosistema ZigBee y la red IP privada del domicilio en el que deseemos integrar el sistema domótico.

Los dispositivos finales de la red ZigBee, no son más que elementos domotizados de la casa, tales como bombillas, sensores, persianas o ventiladores inteligentes, cuyo funcionamiento desea ser controlado remotamente desde una aplicación móvil. Es en este contexto donde cobra sentido el uso de una pasarela que haga de puente entre los dos ecosistemas; el IP, hablado por los dispositivos móviles en los que corra la aplicación, pero no por los elementos domotizados, y el ZigBee, en el que ocurre exactamente lo contrario.

Para el soporte y comunicación entre la aplicación móvil y la pasarela, se hace uso de un servidor, que en este caso será ejecutado en un PC, situado también en la red IP privada de la casa.

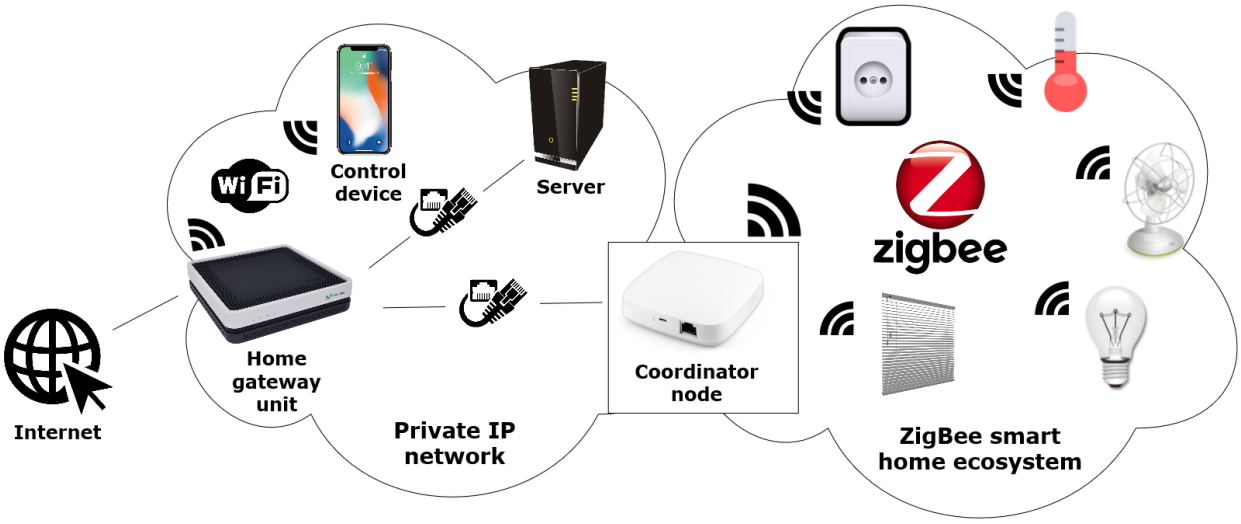

Figura 8. Escenario en el que se integra el sistema domótico y dispositivos que lo conforman.

El alcance del presente trabajo de fin de grado abarca por tanto los siguientes puntos principales:

- Desarrollo de la interfaz de usuario de la aplicación móvil.
- Desarrollo del servidor que da soporte a la aplicación móvil y al nodo coordinador o pasarela.
- Configuración e integración total del ecosistema ZigBee.

- Programación y configuración a nivel hardware del nodo coordinador o pasarela y de los dispositivos domóticos que lo requieran.
- Redacción del presente documento.

4 PLIEGO DE CONDICIONES

Para cumplir con los objetivos de este proyecto, se han seleccionado una serie de soluciones hardware y software, las cuales se definen como requisitos en este apartado.

Se detallan a continuación dichas soluciones implementadas, cuyas funciones concretas en el escenario descrito anteriormente, serán desarrolladas en apartados posteriores. El actual se limitará a detallar sus aspectos generales y especificaciones técnicas, así como a compararlas con las alternativas disponibles en el mercado, resaltando sus ventajas y justificando su elección frente al resto.

4.1 El microcontrolador Arduino UNO

El Arduino UNO [9] es una placa de microcontrolador basada en el chip ATmega328p y desarrollada por Arduino.cc.

Figura 9. Placa Arduino UNO.

4.1.1 Introducción a Arduino

Arduino es una plataforma de código abierto usada para el desarrollo de proyectos electrónicos. Está basada en placas de circuitos programables mediante un IDE (*Integrated Development Environment*) propio. Su facilidad y flexibilidad de uso, unido a su carácter *open-source* (tanto a nivel hardware como software), ha popularizado su uso entre artistas, diseñadores, aficionados y desarrolladores de todos los niveles y perfiles.

4.1.2 El IDE de Arduino

El entorno de desarrollo integrado (IDE) de Arduino es una aplicación multiplataforma (disponible para Windows, macOS y Linux) usada para escribir y cargar programas en las placas de Arduino y similares. Admite la programación mediante el lenguaje propio de Arduino (basado en C++), así como el uso de una

infinidad de bibliotecas desarrolladas tanto por la misma compañía, como por una extensa comunidad de desarrolladores ajenos.

La estructura principal de un programa o *sketch* en Arduino consta de dos funciones principales: *setup* y *loop*. *Setup* es llamada en primer lugar tras la ejecución del *sketch* y contiene la inicialización de variables, módulos de librerías y asignación de pines como entrada/salida o encendido/apagado. Esta función solo se ejecuta una vez, después de cada reinicio o encendido de la placa. La función *loop* se ejecuta a continuación de manera ininterrumpida en bucle y contiene el código que define el control activo de la placa.

El IDE de Arduino emplea la utilidad AVRDUDE para la conversión del código ejecutable a un archivo de texto en codificación hexadecimal, que es cargado directamente en la placa.

Figura 10. IDE de Arduino en Windows 10.

4.1.3 Especificaciones técnicas del Arduino UNO

En la siguiente tabla se recopilan una serie de características técnicas sobre la placa Arduino UNO:

Microcontrolador	ATmega328P
Frecuencia de operación	16 MHz
Tensión de operación	5 V
Tensión de entrada	7-12 V
Memoria FLASH	32 KB
Memoria SRAM	2 KB
Memoria EEPROM	1 KB
Dimensiones	68.6 x 53.4 mm
Peso	25 g

Tabla 3. Especificaciones técnicas del Arduino UNO.

4.1.4 Elementos del Arduino UNO

A continuación, se muestra información relativa a los elementos, puertos, buses y pines de entrada y salida disponibles en la placa:

14 pines digitales de entrada/salida (de los cuales, 6 permiten PWM de salida)
6 pines PWM digitales de entrada/salida
6 pines analógicos de entrada
20 mA de corriente DC por pin de entrada/salida
50 mA de corriente DC para el pin de 3.3V
Soporte de comunicación UART, I2C y SPI
1 puerto USB tipo B

Tabla 4. Información sobre las entradas y salidas del Arduino UNO.

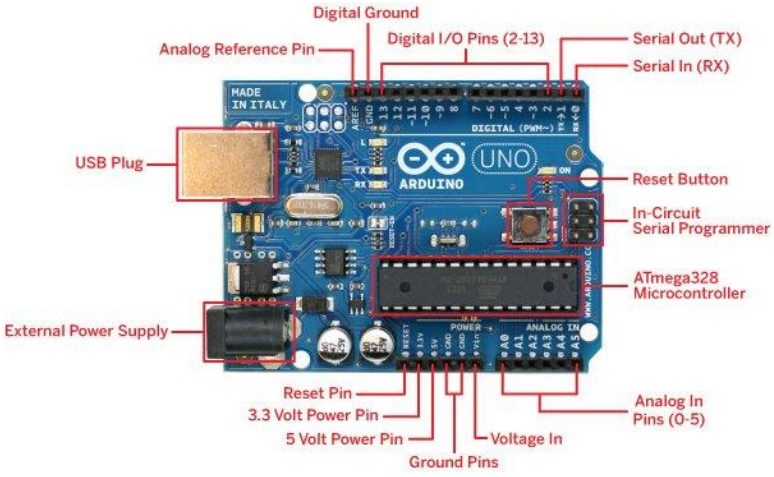

Figura 11. Elementos del Arduino UNO.

4.1.5 Las *shields* de Arduino

Las *shields* son placas electrónicas prefabricadas que se conectan directamente al arduino para ampliar las capacidades de este (control de motores y pantallas, conexión a internet, incorporación de sensores…). De nuevo, a menudo son desarrolladas por la comunidad. Suelen incluir una librería asociada, para facilitar su programación e integración con el arduino a nivel software.

En este proyecto se hace uso de las shields ethernet y xbee.

Ethernet *shield*

Permite conectar el arduino a internet mediante un cable ethernet. Se usa habitualmente junto con la librería *ethernet.h*, la cual permite asociar una dirección MAC e IP al arduino, así como alojar un servidor ligero en el mismo.

Xbee *shield*

Incluye un socket para la conexión de un módulo XBee. Dichos módulos permiten al arduino conectarse a una red ZigBee, convirtiéndolo en un nodo más de la misma.

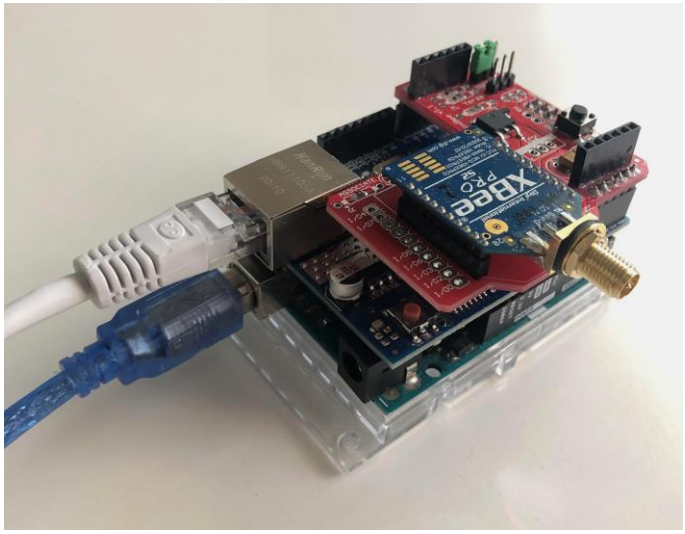

Figura 12. Shields XBee y Ethernet conectadas simultáneamente al Arduino UNO.

4.1.6 Alternativas al Arduino UNO

Los principales motivos de la elección de la placa Arduino UNO frente a otras de la misma familia o de otros fabricantes, es su precio y, sobre todo, su alta compatibilidad con librerías y las *shields* Ethernet y XBee, las cuales se usarán de forma simultánea, tal y como puede verse en la *Figura 12*.

A continuación, se enumera una serie de alternativas al Arduino UNO y el motivo de sus descartes:

Familia Arduino

Dentro de la familia Arduino las alternativas son numerosas, entre las cuales destacan los Nano, Mega o WiFi. La relación precio-prestaciones que ofrece, junto a su alta compatibilidad, han llevado al Arduino UNO a ser el más vendido y documentado de entre placas ofrecidas por la compañía. Esto decantó su elección frente al Nano, el cual debido a su estructura no permitía el uso de *shields*, al WiFi, tras decidirse que la conexión a internet se realizaría mediante Ethernet, o al Mega, cuyos pines y buses extras no compensaban su sobrecoste.

ESP8266

El ESP8266 [10] es un microprocesador con capacidad WiFi que está ganando popularidad en el desarrollo de este tipo de proyectos. Está disponible a un precio más económico que cualquier arduino y puede ser programado mediante su IDE. Sus capacidades limitadas y la decisión del uso de Ethernet frente a WiFi, descartaron finalmente al ESP8266.

Raspberry Pi

La Raspberry Pi [11] es un ordenador de bajo coste y tamaño reducido. Su uso está muy extendido en todo tipo de proyectos de electrónica, así como en tareas básicas que haría cualquier ordenador común, como navegar por internet, hojas de cálculo, procesar textos, reproducir vídeo en alta definición, e incluso *gaming*.

Arduino y Raspberry Pi son dos conceptos totalmente diferentes, aunque gracias a la versatilidad que ofrecen y la imaginación de la comunidad, ha hecho que ambos sean utilizados para crear todo tipo de proyectos de electrónica. La clara ventaja de Raspberry frente a Arduino es su potencia de cálculo y su conectividad. La gran desventaja radica en que su libertad de desarrollo está más limitada, sobre todo a nivel hardware, ya que la Raspberry Foundation mantiene el control sobre sus placas, y sólo ellos las diseñan y fabrican.

Las funcionalidades del arduino junto con el uso de las *shields*, son más que suficientes para abordar el proyecto, por lo que finalmente no se planteó asumir la diferencia de precio que suponía la raspberry.

4.2 El microcontrolador Waspmote v1.2

Waspmote v1.2 [12] es una placa de microcontrolador orientada al desarrollo de proyectos de IoT basados en redes inalámbricas de sensores, con requisitos muy específicos y destinados a ser desplegados en entornos reales.

Fue lanzada al mercado por la empresa española Libelium, la cual ofrece además un IDE propio basado en el de Arduino, por lo que pueden reutilizarse los códigos con algunos cambios en la sintaxis y en el soporte de ciertas librerías. Podemos decir pues, que la placa Waspmote v1.2 es una especie de arduino con capacidades especiales orientadas a la conectividad y a actuar como nodo sensor dentro de una WSN.

Figura 13. Placa Waspmote v1.2.

4.2.1 Especificaciones técnicas del Waspmote v1.2

Microcontrolador	ATmega1281
Frecuencia de operación	14.7456 MHz
Memoria FLASH	128 KB
Memoria SRAM	8 KB
Memoria EEPROM	4 KB
Dimensiones	73.5 x 51 x 13 mm
Peso	20 g
Rango de temperaturas	-10 ºC, + 65 ºC
Tensión de la batería	3.3 - 4.2 V
Carga mediante USB	5 V y 280 mA
Carga mediante panel solar	6 - 12 V y 280 mA
Consumo de corriente	15 mA (encendido) y 55 uA (sleep)

Tabla 5. Especificaciones técnicas del Waspmote v1.2.

4.2.2 Elementos del Waspmote v1.2

8 pines digitales de entrada/salida
1 PWM
7 pines analógicos de entrada
Soporte de comunicación UART, I2C y SPI
1 puerto micro USB
Admite tarjeta SD de hasta 2 GB
Sensor de temperatura de -40 ºC a + 85 ºC (error de ±0.25 ºC)
Acelerómetro en los 3 ejes (±2 g / ±4 g / ±8 g)

Tabla 6. Información sobre las entradas y salidas del Waspmote v1.2.

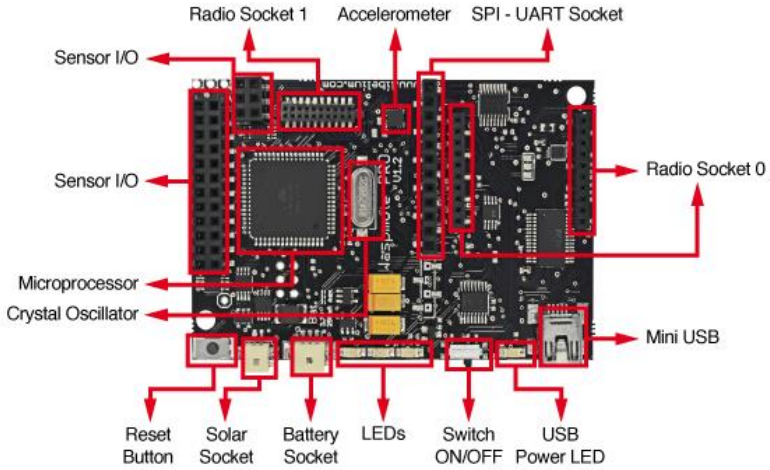

Figura 14. Elementos presentes en el anverso del Waspmote v1.2.

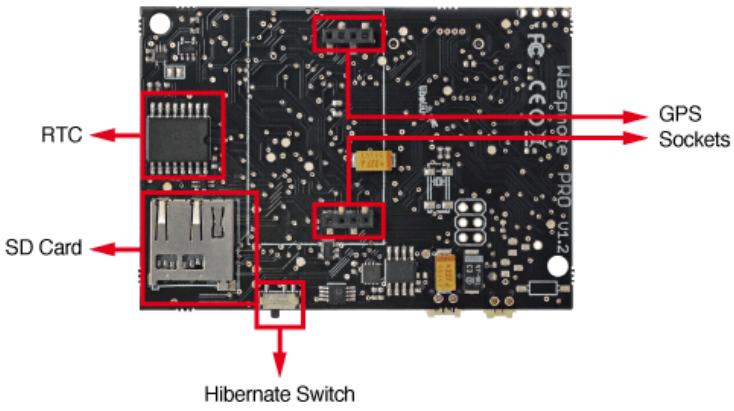

Figura 15. Elementos presentes en el reverso del Waspmote v1.2.

4.3 El módulo XBee PRO S2

Los módulos XBee [13] son soluciones integradas ofrecidas por el fabricante Digi International Inc. que brindan un medio inalámbrico para la interconexión y comunicación de dispositivos. Utilizan los protocolos IEEE 802.15.4 y ZigBee para la creación de redes. Fueron diseñados para aplicaciones que requieren de un alto tráfico de datos, baja latencia y una sincronización de comunicación simple.

Cuentan con un software dedicado con el que programar y configurar ciertos parámetros de cada módulo y son integrados en el arduino gracias a la ya comentada XBee *shield* y directamente al waspmote mediante el socket radio del que dispone (ver *Figura 14*).

Figura 16. Módulo XBee PRO S2.

4.3.1 El software XCTU

XCTU es una aplicación multiplataforma compatible con Windows, MacOS y Linux diseñada por Digi para permitir a los desarrolladores interactuar con los módulos XBee gracias a su simple e intuitiva interfaz gráfica. Para la conexión del módulo XBee con el PC donde corre XCTU es necesario un adaptador USB específico.

De entre las posibilidades que permite XCTU destacan:

- Programación y configuración por cable y remota de los módulos XBee.
- Carga de perfiles ZigBee predefinidos en los módulos.
- Actualización del firmware de los módulos.
- Creación y configuración de redes ZigBee.
- Testeo de comunicaciones inalámbricas mediante el analizador de área.
- Creación y prueba de tramas ZigBee mediante el generador y el intérprete de tramas.

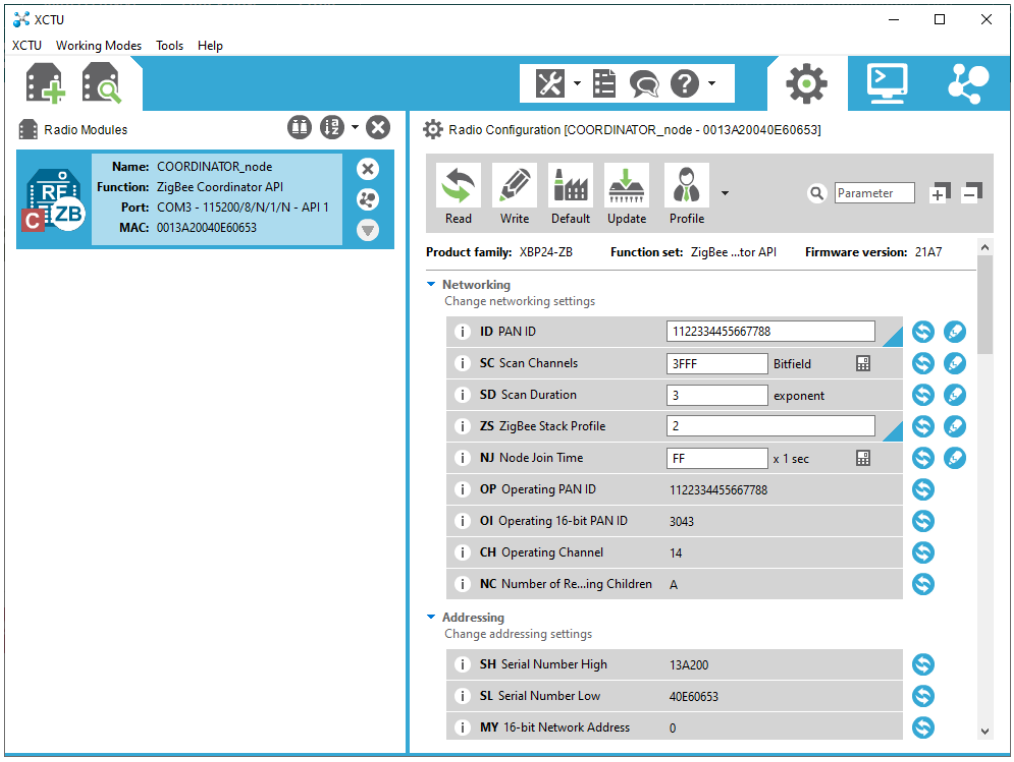

Figura 17. Pestaña de configuración del software XCTU.

4.3.2 Alternativas al XBee PRO S2

Los módulos XBee, son los más populares y usados para la creación de redes ZigBee y proyectos asociados, gracias a las ventajas ya comentadas y al apoyo directo de la ZigBee Alliance. Ahora bien, dentro de la familia XBee existe una amplia variedad de alternativas cuyas características técnicas de algunas de ellas son comparadas en la siguiente tabla:

Modelo	Frecuencia	Potencia de transmisión	Sensibilidad	Tipo de antena
XBee ZB S2C TH	2.4 GHz	6.3 mW	-101 dBm	Wire
XBee PRO ZB S2C TH	2.4 GHz	63 mW	-101 dBm	Wire
XBee PRO S2	2.4 GHz	50 mW	-102 dBm	RP-SMA
XBee PRO 900HP	900 MHz	250 mW	-100 dBm	RP-SMA
Digi XBee 3 ZigBee 3.0	2.4 GHz	6.3mW	-103 dBm	PCB

Tabla 7. Comparativa de módulos XBee de Digi.

4.4 El *framework* React Native

React Native [14] [15] es un *framework* de JavaScript orientado al desarrollo de aplicaciones móviles nativas en el lado del cliente (*frontend*), basado en la librearía de JavaScript React. Permite el desarrollo de

aplicaciones bajo código único para iOS y Android, obteniendo como resultado final una interfaz de usuario nativa real. Facebook es el principal impulsor de esta tecnología y empresas como Instagram, Skype, Tesla o Uber Eats ya están apostando por este *framework* para sus propias aplicaciones.

4.4.1 Características de React Native

Bajo esta filosofía de construcción de aplicaciones móviles, React Native ofrece las siguientes ventajas:

Compatibilidad *Cross-Platform*

Gracias a esta característica, es posible crear aplicaciones que pueden ser ejecutadas tanto en iOS como en Android simultáneamente bajo el mismo código base.

Funcionalidades nativas

Las aplicaciones creadas con React Native funcionan de la misma manera que una aplicación nativa real programadas para cada uno de los sistemas operativos usando su lenguaje propio. La unión de React Native con JavaScript permite la ejecución de aplicaciones complejas con un óptimo rendimiento.

Figura 18. React Native está cambiando el concepto de desarrollo de aplicaciones móviles.

Actualizaciones instantáneas para desarrollo y testeo

Los desarrolladores tienen la flexibilidad de hacer efectivas las actualizaciones directamente en el dispositivo del usuario sin tener que pasar por las tiendas de aplicaciones propias de cada sistema y sus tediosos procesos obligatorios previos.

Sencilla curva de aprendizaje

React Native es extremadamente fácil de leer y de aprender ya que se basa en conceptos fundamentales del lenguaje JavaScript, siendo especialmente intuitivo para los expertos en dicho lenguaje.

Experiencia positiva para el desarrollador

Aparte de su sencillez, el propio lenguaje motiva al desarrollador a probar e introducir mejoras más complejas a las aplicaciones, según se aumenta el conocimiento y dominio del mismo. Ofrece varias funcionalidades importantes como el *Hot reload* que refresca la aplicación en el momento en que se guardan cambios o el uso del *debugger* del navegador Google Chrome, para la tarea de depuración de código.

4.4.2 El *framework* Expo

Expo [16] es el mejor aliado para el desarrollado de aplicaciones móviles con React Native. Se trata de un *framework open-source* que proporciona una colección de herramientas que facilitan el arranque y testeo de aplicaciones React Native, así como la programación y administración de los códigos.

Es una gran alternativa al uso directo del React Native CLI (*Command Line Interface*) cuyo enfoque más complejo y engorroso, está orientado a programadores experimentados o a aplicaciones que requieran un extra de flexibilidad o nivel de customización en sus diseños.

La plataforma Expo, incluye una aplicación móvil disponible para iOS y Android, que permite correr nuestro proyecto de aplicación al instante en un dispositivo móvil físico, mediante el escaneo de un código QR, lo cual es ideal para testear instantáneamente los cambios realizados en la programación de los códigos.

4.4.3 El editor de código Visual Studio Code

El editor de código fuente usado para la programación de la aplicación móvil es Visual Studio Code de Microsoft. Algunas de las ventajas que ofrece dicho editor y que han llevado a la elección de este, son las siguientes:

- Uso de extensiones que amplían funcionalidades y facilitan la tarea de programación.
- Resaltado de sintaxis y finalización de código inteligente mediante atajos.
- Es gratuito y de código abierto.
- Alto grado de personalizaciones estéticas y funcionales.
- Soporte nativo para los principales lenguajes de programación.
- Depurador sencillo y eficiente.
- Gran soporte técnico, disponibilidad de documentación y ayuda en la página oficial, foros, etc.

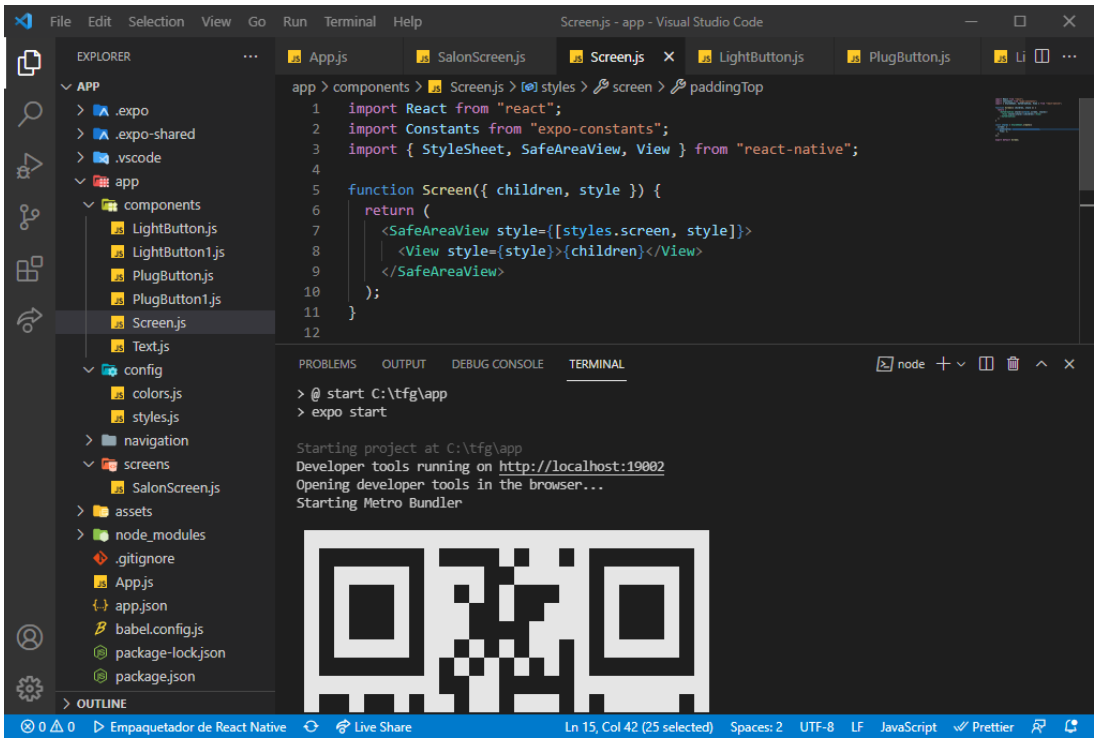

Figura 19. Editor Visual Estudio Code con código React Native y Expo.

4.4.4 Alternativas a React Native

Como ya se ha mencionado, React Native nace como alternativa a la programación nativa en iOS y Android, por lo que la comparativa obvia sería con los lenguajes Swift y Objective-C, usados para el desarrollo de aplicaciones destinadas a iOS y con Java y Kotlin para Android:

Lenguaje de programación	Desarrollador	Multiplataforma	Dificultad de aprendizaje	Hot Reload	Aplicaciones populares
Swift	Apple	No (solo iOS)	Media	No	Slack, LinkedIn, WhatsApp…
Objective-C	Tom Love y Brad Cox	No (solo iOS)	Alta	No	Casi todas las principales apps iOS hasta hace unos 4 años.
React Native (JavaScript)	Facebook	Sí	Baja	Sí	Facebook, Instagram, Skype…
Kotlin	JetBrains	No (sólo Android)	Alta	No	Pinterest, Evernote, Uber…
Java	Oracle	No (sólo Android)	Media	No	Spotify, Twitter, Telegram

Tabla 8. Comparativa de React Native frente a sus alternativas.

React Native es por tanto la opción perfecta para este proyecto, no obstante, existe una serie de contextos en los que podría ser más conveniente el uso de lenguajes nativos:

- Aplicación extremadamente compleja que necesite el uso de la API (*Application Programming Interface)* nativa.
- Desarrollo de una aplicación destinada a un solo sistema operativo.
- Necesidad de soporte inmediato de las últimas funcionalidades nativas lanzadas.

4.5 El servicio de *backend* Node.js

Node.js [17] es un entorno en tiempo de ejecución para la capa del servidor basado en el lenguaje de programación JavaScript. Es especialmente usado para la creación de servicios de *backend* o API de tipo REST.

4.5.1 Concepto de API REST

Una API REST (*Representational State Transfer*) [18] [19] es un tipo de servicio que da soporte al lado del cliente (*frontend*) de una aplicación de cualquier tipo (móvil en nuestro caso). Dichas aplicaciones cliente, son simplemente la superficie con la que el usuario interactúa. Es necesario pues, de un servicio situado en un servidor o en la nube, que almacene datos, envíe notificaciones y, en definitiva, se comunique con el *frontend* y le de todo el soporte necesario. Para que una API se considere de tipo REST, debe cumplir con los siguientes principios:

Estructura cliente/servidor sin estado

Las API REST siguen una estructura compuesta por el lado del cliente y el servidor, que comparten recursos mediante peticiones HTTP sin estado, lo cual implica que cada una de ellas es independiente y está desconectada del resto.

Almacenamiento y manipulación de recursos mediante URI

Una URI representa un identificador único para cada recurso, los cuales son almacenados en forma de objetos en un sistema REST. Esto nos facilita el acceso a la información para poder modificarla, borrarla o compartir su ubicación exacta a terceros.

Cuatro operaciones principales

Las operaciones o métodos más importantes relacionados con el intercambio de datos en cualquier sistema REST, en particular, y para el protocolo HTTP en general, son cuatro: POST (crear), GET (leer), PUT (editar) y DELETE (borrar).

Sistema de capas

Su arquitectura forma un sistema de capas jerarquizado e invisible para el cliente. Cada una de estas capas, cumple con una funcionalidad diferente dentro del sistema REST, lo cual favorece enormemente la escalabilidad, el equilibrio en el reparto de carga y la seguridad.

Independencia del tipo de plataformas o lenguajes

La API REST se adapta a cualquier tipo de sintaxis o plataforma con la que se desee trabajar. Esto ofrece una gran libertad a la hora de elegir entornos o lenguajes para su programación. Sin embargo, las respuestas a las peticiones se deben hacer siempre mediante un lenguaje de intercambio de recursos específico, ya sea XML o JSON.

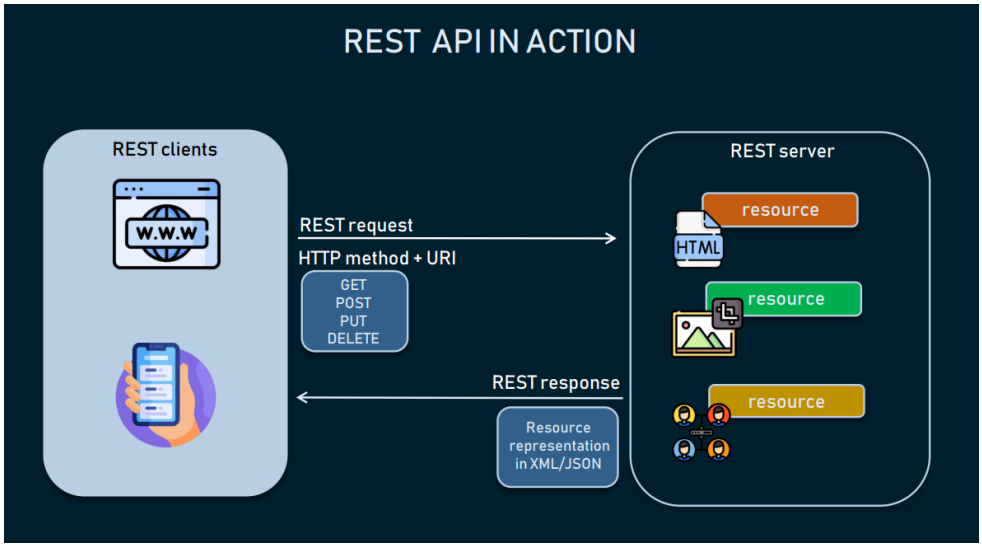

Figura 20. Comunicación entre el lado del cliente de una aplicación y su API REST. *Fuente: www.altexsoft.com.*

4.5.2 Características y ventajas del uso de Node.js como servicio de *backend* o API REST

Node.js es ideal para el desarrollo de servicios de *backend* que requieran de una alta escalabilidad y una transferencia continua de datos en tiempo real. Es usado en producción por grandes compañías como PayPal, Uber, Netflix, etc, debido a las características y ventajas que ofrece:

- Entorno multiplataforma y de código abierto basado en JavaScript.
- Uso del potente motor V8 de Google.
- Gran ecosistema de librerías disponibles que facilitan la tarea de desarrollo.
- Permite agilizar procesos gracias a su carácter asíncrono basado en la producción, detección, reacción y consumo de eventos (paradigma *event-driven architecture* o arquitectura orientada a eventos).

4.5.3 Alternativas a Node.js

Existen varias alternativas para el desarrollo de una API REST como PHP, Python o Django, siendo Node.js la más utilizada para la creación de este tipo de servicios de *backend*.

En este apartado no se realizará ninguna comparativa de características técnicas con la competencia, ya que tampoco se contempló el uso de ninguna de ellas, debido a que la idea era continuar con la programación en el lenguaje JavaScript. Se optó pues, por seguir con la filosofía *JavaScript everywhere*, unificando el desarrollo de las capas de usuario y servidor de la aplicación en torno a un solo lenguaje de programación, con lo que ello supone a nivel de consistencia y limpieza de código.

Así mismo, se continuó con el uso de la herramienta Visual Studio Code para la escritura y edición de código

pues, al fin y al cabo, la programación seguía siendo en JavaScript, por lo que la compatibilidad es total.

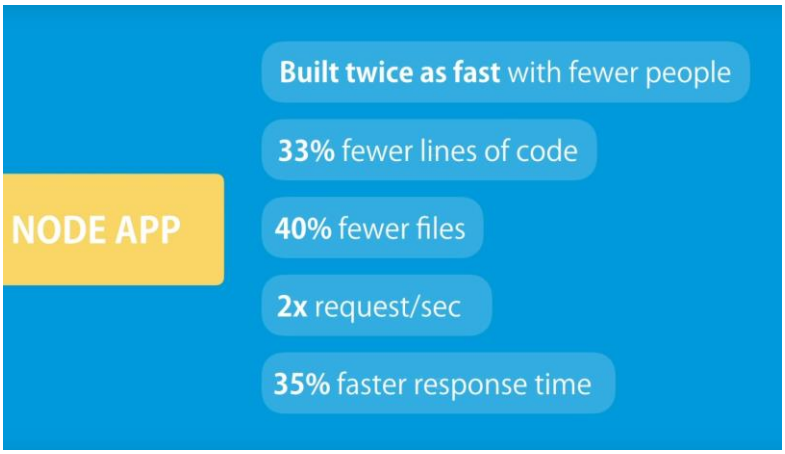

Figura 21. Datos sobre el rendimiento de la aplicación de PayPal tras reescribirse su *backend* usando Node.js.

4.6 Dispositivos adicionales

En este apartado se describirán las especificaciones técnicas de algunos dispositivos hardware adicionales que se integrarán en el sistema domótico. La elección de estos obedece a motivos económicos, de compatibilidad y/o de disponibilidad las tiendas, por lo que mientras cumplan con dichos requisitos, pueden ser sustituidos por cualquier otra opción equivalente.

4.6.1 Bombilla LED inteligente Philips Hue A60 E27

Bombilla LED inteligente con iluminación regulable a distancia directamente con Bluetooth o mediante su integración en una red ZigBee.

Tipo de producto	Bombilla LED inteligente
Tecnologías	ZigBee y Bluetooth
Tipo de casquillo	E27
Tensión de entrada	220 - 240 V de AC
Potencia de bombilla	7 W
Flujo luminoso	550 lm
Formato de lámpara	A60
Clasificación energética	A+
Duración de la lámpara	15000 h
Dimensiones	115 x 60 mm
Temperatura cromática máxima	2100 K
Ciclos de apagado y encendido	50000

Tabla 9. Especificaciones técnicas de la bombilla LED inteligente Philips Hue A60 E27.

Figura 22. Bombilla LED inteligente Philips Hue A60 E27.

4.6.2 Adaptador de enchufe inteligente Ledvance Smart+

Adaptador de enchufe inteligente con luz de estado y tecnología ZigBee, que permite controlar el apagado y encendido de dispositivos no inteligentes.

Tipo de producto	Enchufe inteligente
Tecnología	ZigBee
Material	Plástico
Tensión de entrada	230 V de AC
Corriente máxima	16 A
Clasificación energética	A+
Dimensiones y peso	8.4 x 6 x 6 cm; 130 g

Tabla 10. Especificaciones técnicas del adaptador de enchufe inteligente Ledvance Smart+.

Figura 23. Adaptador de enchufe inteligente Ledvance Smart+.

4.6.3 Sensor de humedad y temperatura DHT11

Sensor de humedad y temperatura de alta fiabilidad y estabilidad debido a su transmisión de datos por señal digital calibrada.

Tipo de producto	Sensor de humedad y temperatura
Señal de salida	Digital
Alimentación	5 V
Consumo	2,5 mA
Rango de temperatura	0 ºC - 50 ºC
Rango de humedad relativa	20 % - 95 %

Tabla 11. Especificaciones técnicas del sensor de humedad y temperatura DHT11.

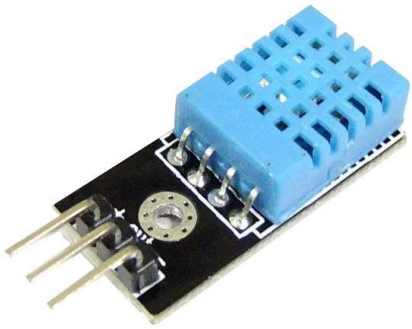

Figura 24. Sensor de humedad y temperatura DHT11.

4.6.4 Otros componentes electrónicos

- Diodo LED de luz blanca.
- Módulo controlador de motor L9110H.

5 ARQUITECTURA E IMPLEMENTACIÓN HARDWARE DEL SISTEMA

En este capítulo, se desarrollará en profundidad la arquitectura del sistema a nivel hardware conforme al escenario expuesto en el *Caso de estudio*.

Se explicará, por tanto, la manera en la que se han integrado las soluciones hardware expuestas en el capítulo anterior para conformar los distintos elementos, así como el papel que desempeñan cada uno de ellos en el sistema.

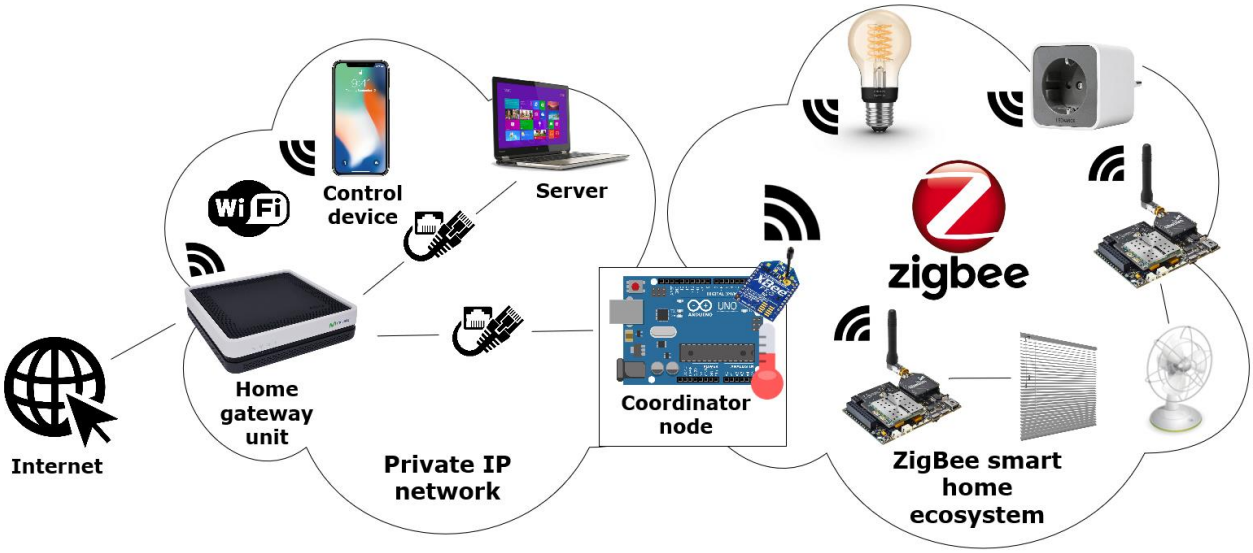

Figura 25. Representación del escenario expuesto en el *Caso de estudio* con los elementos hardware finalmente implementados.

5.1 Servidor

Se trata de un PC portátil en el que se ejecutará el servicio de *backend* o API REST, haciendo las veces de servidor del sistema domótico. Dicho equipo, forma parte de la red IP local del domicilio, a la cual podrá acceder inalámbricamente mediante WiFi o usando un cable Ethernet, siendo preferible la segunda opción, siempre que sea posible, para un funcionamiento óptimo.

El papel de este servidor es el de almacenar información descriptiva y parámetros del estado de los distintos dispositivos inteligentes de la casa (bombillas, ventiladores, persianas…) incorporados al sistema domótico. Dichos parámetros serán modificados por el dispositivo móvil de control y consultados por el nodo

coordinador, estableciéndose así una comunicación entre ellos orquestada por este servicio de *backend*.

5.2 Dispositivo móvil

El dispositivo móvil (ya sea teléfono, tableta, o dispositivo similar con sistema operativo Android o iOS), es el elemento encargado de correr la aplicación mediante la que será posible controlar el sistema domótico al completo (encender/apagar/atenuar/aumentar bombillas, atenuar/aumentar la velocidad de ventilador, consultar temperatura y humedad…).

Se integrará vía WiFi a la red IP local del domicilio, ofrecida evidentemente por el *home gateway unit*, más coloquialmente conocido como el router WiFi de la casa.

Se comunica directamente con el servidor, para modificar los parámetros de los dispositivos domóticos, así como con el nodo coordinador, avisándole cuando el estado de algún dispositivo haya sido modificado, para que este consulte en el servidor el parámetro en cuestión y actúe en consecuencia.

5.3 Nodo coordinador o pasarela

Es el elemento encargado de hacer de pasarela entre las redes IP y ZigBee. Recibe órdenes directas de la aplicación móvil, justo en el instante posterior de que se haya hecho una modificación en el estado de algún dispositivo ZigBee. La pasarela, junto con la orden de la aplicación, recibe el ID del elemento modificado en cuestión, por lo que es capaz de consultarlo directamente en el servidor y trasladarle la indicación en forma de trama ZigBee al dispositivo domótico correspondiente.

5.3.1 Arquitectura del nodo coordinador

La arquitectura del nodo coordinador está conformada por el Arduino UNO junto con las *shields* Ethernet y XBee, las cuales le permiten acceder a la red de área local de la casa y al ecosistema ZigBee, respectivamente. El arduino será alimentado mediante su puerto USB tipo B, gracias a un cable del mismo tipo conectado a la toma de corriente por medio de un adaptador USB para enchufe.

Montaje del nodo coordinador y conexión de las *shields*

Como ya se ha comentado, el gran beneficio de las *shields* es que se tratan de placas con circuitos integrados complejos que añaden distintas funcionalidades al arduino con tan solo conectarlas al mismo. Se ha aprovechado este punto a favor, que elimina prácticamente toda la complejidad a nivel hardware que tiene el desarrollo del nodo coordinador. No obstante, es importante seguir los siguientes pasos de montaje para asegurar el correcto funcionamiento de este:

- En primer lugar, conectar directamente al arduino la *shield* Ethernet.
- Tras alimentar la *shield* XBee (ver siguiente subapartado), colocarla en segundo lugar, justo encima de la *shield* Ethernet.
- Conectar el módulo XBee PRO S2 en el socket dedicado.
- Extraer el segundo *jumper* (ver *Figura 28*) para que el módulo XBee no ocupe permanentemente el único puerto serie UART (*universally asynchronous receiver/transmitter*) del arduino, lo que imposibilitaría la carga de programas y el uso del monitor serial.

Cabe señalar que tanto como el módulo XBee (para el envío/recepción de tramas ZigBee), como el puerto USB tipo B (para la carga de programas y la impresión de información en el monitor serial disponible en la IDE de Arduino), se alternan el uso del puerto serie UART del arduino.

Por otro lado, la *shield* Ethernet se comunica con el arduino mediante SPI (*Serial Peripheral Interface*) a través de los pines 13 (SCK), 12 (MISO), 11 (MOSI) y 10 (SS), mientras que la XBee accede al puerto serie UART a través los pines 0 (RX) y 1 (TX).

Por último, también podemos ver en la *Figura 28* un diodo LED en el pin 9, que señalizará el envío de tramas

ZigBee. Posteriormente se detallará su funcionamiento.

5.3.2 ¿Qué es el ICSP y cómo alimentar correctamente la shield XBee?

El ICSP (*In-Circuit Serial Programming*) [20] es un conector consistente en 6 señales: MOSI, MISO, SCK, RESET, VCC, GND. Permite la programación directa del chip ATmega328P mientras se encuentra integrado en la placa de Arduino. Además de ser un método alternativo a la programación vía USB, podemos usarlo también para comunicar y/o alimentar periféricos o *shields*.

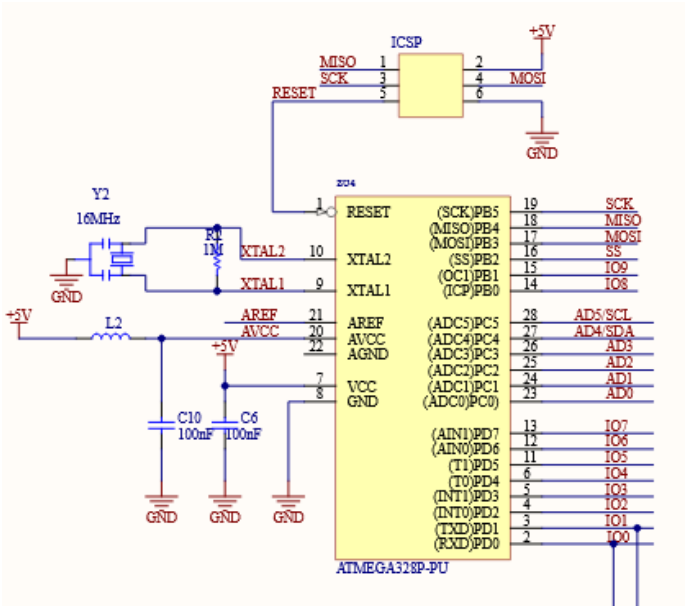

Figura 26. Esquema del ICSP y sus conexiones con el chip ATmega328P. *Fuente: store.arduino.cc*.

En este caso haremos uso del ICSP para esta segunda funcionalidad, siendo este ocupado por la *shield* Ethernet, por lo que surge una problemática a la hora de conectar simultáneamente dicha *shield* junto con la XBee. La solución es sencilla: sabiendo que la *shield* XBee solo precisa de las señales VCC, GND y RESET del ICSP, tomaremos dichas señales directamente del Arduino (trasladadas sin usar desde la *shield* Ethernet), tal y como podemos ver en la siguiente figura:

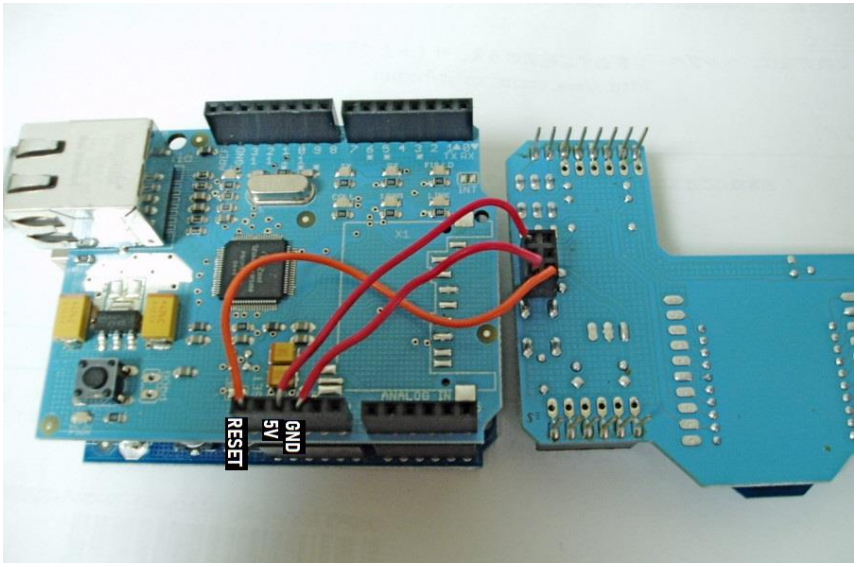

Figura 27. Alimentación de la XBee *shield*. *Fuente: maker.wiznet.io/2016/11/14*.

5.3.3 El sensor de humedad y temperatura

El sensor de humedad y temperatura es el único elemento domótico que se integra directamente en el propio nodo coordinador.

Los datos sobre humedad y temperatura que proporciona, son enviados directamente por el nodo coordinador al dispositivo móvil cuando dicha información es requerida. Posteriormente, será el mismo dispositivo móvil el que actualiza el valor de los parámetros relacionados con el sensor en el servidor.

Su arquitectura es sencilla pues dispone de solo tres pines (GND, DATA y VCC). Como puede verse en la *Figura 28*, el pin de datos (cable azul) se conecta directamente a cualquier pin digital disponible del arduino (el 2 en este caso). La alimentación y tierra del arduino, han tenido que ser trasladarlas desde el arduino hacia una placa de pruebas, ya que en un principio quedaban ocupadas por la XBee *shield*. Desde ahí se conectan tanto la *shield* como los pines GND y VCC del sensor.

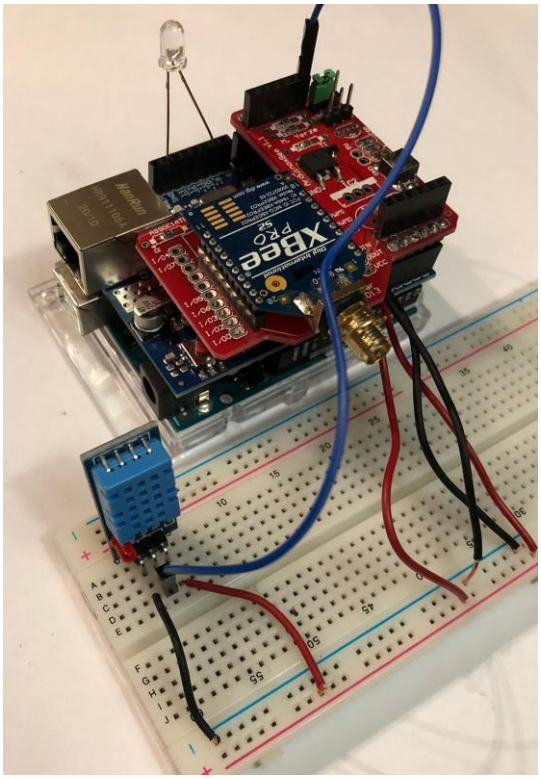

Figura 28. Implementación a nivel hardware del nodo coordinador al completo.

5.4 Elementos domóticos del ecosistema ZigBee

Corresponde cada uno a un dispositivo o nodo final de la red ZigBee. Se incluirán finalmente en el ecosistema los siguientes elementos: una bombilla, un adaptador de enchufe y un ventilador.

Estos dispositivos o nodos finales, reciben las órdenes introducidas por el usuario desde la aplicación móvil por medio de tramas ZigBee generadas en el nodo coordinador.

5.4.1 La bombilla y el adaptador de enchufe inteligentes

Tanto la bombilla como el adaptador de enchufe, resultan de menor interés a nivel de desarrollo hardware para este proyecto, debido a que son productos comerciales reales que han sido adquiridos tal cual en el mercado, listos para ser implementados. Dichos dispositivos incluyen tarjetas ZigBee propias de cada fabricante. Es precisamente este aspecto el que ha supuesto un mayor trabajo, sobre todo a la hora de poder comunicar la tarjeta XBee que usa el nodo coordinador con la de cada uno de los fabricantes de la bombilla y el adaptador

de enchufe. Dicha cuestión será abordada en profundidad en el próximo capítulo: *Descripción del sistema a nivel software*.

El adaptador estará listo para ser incorporado en el ecosistema ZigBee tras ser enchufado a la corriente, mientras que, en el caso de la bombilla, hará falta enroscarla en un portalámparas de tipo E27 y activar su correspondiente interruptor de corriente.

5.4.2 El ventilador inteligente

Se trata de una pequeña maqueta formada por el módulo controlador de motor L9110H y la placa de microcontrolador Waspmote v1.2. El mismo módulo incorpora un motor con una hélice de plástico que representa un ventilador cuya velocidad se controla mediante la señal de PWM del waspmote, situada en el pin digital 1. Para alimentar el motor se usará el pin de 5V que proporciona también el waspmote.

El funcionamiento de este dispositivo final de la red ZigBee es sencillo: el waspmote espera la posible llegada de tramas ZigBee por parte del nodo coordinador y alimenta el módulo controlador del motor aplicando hasta cuatro niveles distintos de ciclos de trabajo en el PWM según lo que se indique en las mismas. Esto propiciará que el ventilador gire con una mayor o menor velocidad.

Este dispositivo es totalmente inalámbrico, pues se alimenta a través de una batería. Bastaría con encender el waspmote para que se encuentre operativo.

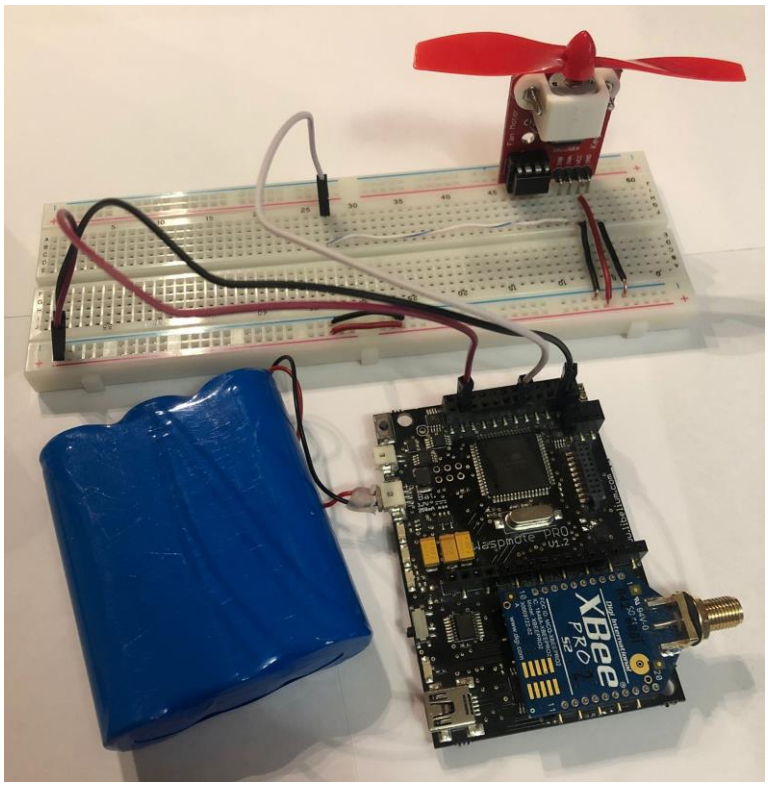

Figura 29. Implementación hardware del ventilador inteligente.

6 DESCRIPCIÓN DEL SISTEMA A NIVEL SOFTWARE

Con el fin de lograr las funcionalidades propias de cada elemento implementado en el sistema, se han llevado a cabo una serie de prácticas y procedimientos a nivel software que posibilitan la correcta integración y operatividad del escenario propuesto.

En este capítulo, se profundizará en los distintos lenguajes, librerías, entornos y protocolos aplicados a lo largo del desarrollo del proyecto.

6.1 Desarrollo del ecosistema ZigBee con el software XCTU

El uso de la aplicación XCTU junto con el adaptador USB específico para el módulo XBee, ha sido clave para el desarrollo del ecosistema ZigBee. Permitió convertir el PC en el nodo coordinador y dispositivo controlador del sistema domótico durante la fase de pruebas. Este hecho posibilitó el total desacople de la red ZigBee con respecto al resto del escenario, con las ventajas que ello supuso a nivel de simplificación del problema y testeo de prácticas que más tarde se extrapolarían al sistema domótico completo.

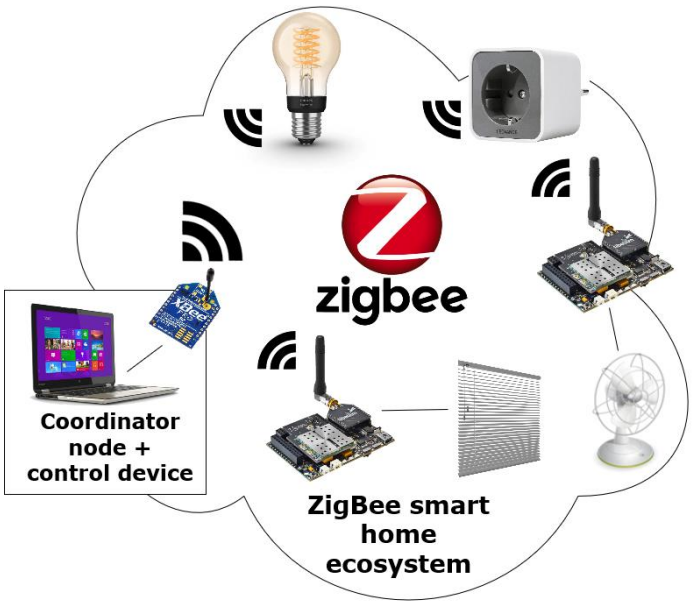

Figura 30. PC como nodo coordinador y dispositivo controlador del ecosistema ZigBee.

6.1.1 Preparación del entorno

Para poner en marcha el entorno que nos permita interactuar directamente con el módulo XBee, lo primero será insertarlo en el socket USB del adaptador y conectarlo al PC mediante el mismo. Trabajando ya con la interfaz gráfica del XCTU, debemos escanear el puerto del PC al que se ha conectado el adaptador y descubrir el módulo radio insertado. Tras esto, ya tendremos acceso a la pestaña *configuration working mode*, en la cual es posible modificar ciertos perfiles y parámetros de configuración relativos a la tarjeta XBee conectada (ver *Figura 17*).

6.1.2 Selección del perfil de configuración y parámetros adecuados

Mediante la correcta asignación del perfil de configuración y la elección de algunos parámetros clave, será posible otorgarle la condición de nodo coordinador de la red ZigBee a nuestro PC, así como dejar configurado el módulo XBee para cuando vaya a ser integrado posteriormente en el arduino.

El perfil o *function set* aplicado al módulo es el de *ZigBee Coordinator API*, mientras que el valor establecido en los parámetros más importantes son los siguientes [21]:

- **ID** (*Personal area ID*) ➔ **1122334455667788**.
- **ZS** (*ZigBee stack profile setting*) ➔ **2** (para el perfil ZigBee-PRO).
- **NI** (*Node ID*) ➔ **COORDINATOR_node**.
- **EE** (*Encryption enabled*) ➔ **ENABLED**.
- **EO** (*Encryption options*) ➔ **2** (para transmitir la clave desencriptada durante el escaneo de dispositivos).
- **KY** (*Encryption key*) ➔ **5A6967426565416C6C69616E63653039**.
- **BD** (*Baud rate*) ➔ **115200** (para la comunicación con el puerto serie del arduino).
- **AP** (*API mode*) ➔ **ENABLED** (para el formato de trama ZigBee usado en la comunicación).

6.1.3 Configuración y visualización de la red ZigBee

Pinchando en la pestaña *networking working mode*, se accederá al escáner del resto de dispositivos ZigBee disponibles en la red. Será posible escanear e incorporar a nuestra red cualquier dispositivo ZigBee al alcance y visualizar la topología formada.

En el caso de que los dispositivos a incorporar lo hagan a través de otra tarjeta XBee (como los waspmote), deben de poseer un perfil *ZigBee Router API* y estar desvinculados de otra red ZigBee, para que sea posible su escaneo y vinculación. En el caso de que sea un dispositivo Wasmote el que porte el XBee, es necesario además que tengan cargado un programa que cuente con una fase inicial de comprobación de los distintos parámetros de red. Durante dicha fase, es conveniente que el waspmote, en su inicialización, permita además al XBee detectar y conectarse a una nueva red ZigBee.

Si estos requisitos no se cumplen, la tarjeta XBee en cuestión deberá ser configurada mediante el XCTU y un adaptador USB conectado al PC, para posteriormente volver a colocarla en el dispositivo en el que se encontraba previamente. Si se diera el caso, adicionalmente habría que cargar un programa en el waspmote que permita al XBee la asociación a la red ZigBee con el *Personal area ID* proporcionado. Dicho ID deberá ser, evidentemente, el establecido por el nodo coordinador en el momento del inicio del escaneo. Tras esto, será posible acceder en la pestaña *configuration working mode*, a los perfiles de dichos módulos remotos y visualizar y/o editar sus parámetros de configuración de forma inalámbrica (si no se ha hecho ya mediante adaptador USB).

Los parámetros configurados serán los siguientes:

- **ID** (*Personal area ID*) ➔ **1122334455667788** (mismo que el configurado por el nodo coordinador).
- **ZS** (*ZigBee stack profile setting*) ➔ **2** (para el perfil ZigBee-PRO).

- **DH** (*Destination address high*) ➔ Establecer la correspondiente al nodo coordinador.
- **DL** (*Destination address low*) ➔ Establecer la correspondiente al nodo coordinador.
- **NI** (*Node ID*) ➔ Arbitrario (según funcionalidad).
- **EE** (*Encryption enabled*) ➔ **ENABLED**.
- **EO** (*Encryption options*) ➔ **0** (para recibir la clave desencriptada durante el escaneo de dispositivos por parte del nodo coordinador).
- **BD** (*Baud rate*) ➔ **115200** (para la comunicación con el puerto serie de waspmote).
- **AP** (*API mode*) ➔ **ENABLED** (para el formato de trama ZigBee usado en la comunicación).

Por otro lado, si el dispositivo a vincular a nuestra red pertenece a otro fabricante (no usa XBee como tarjeta ZigBee), como sería el caso de la bombilla o el adaptador de enchufe, podrán incorporarse directamente si no están asociados a ninguna otra red ZigBee. Si esto último ocurre y no es posible la visualización tras el escaneo en el *networking working mode*, será preciso reestablecer la configuración de fábrica de dichos dispositivos.

Para realizar esta acción, bastará con mantener pulsado durante 10 segundos el botón de ON/OFF del adaptador de enchufe, mientras que para el caso de la bombilla, habrá que hacer lo propio mediante la opción dedicada en la aplicación móvil Philips Hue Bluetooth, disponible para su descarga en cualquier dispositivo Android o iOS,

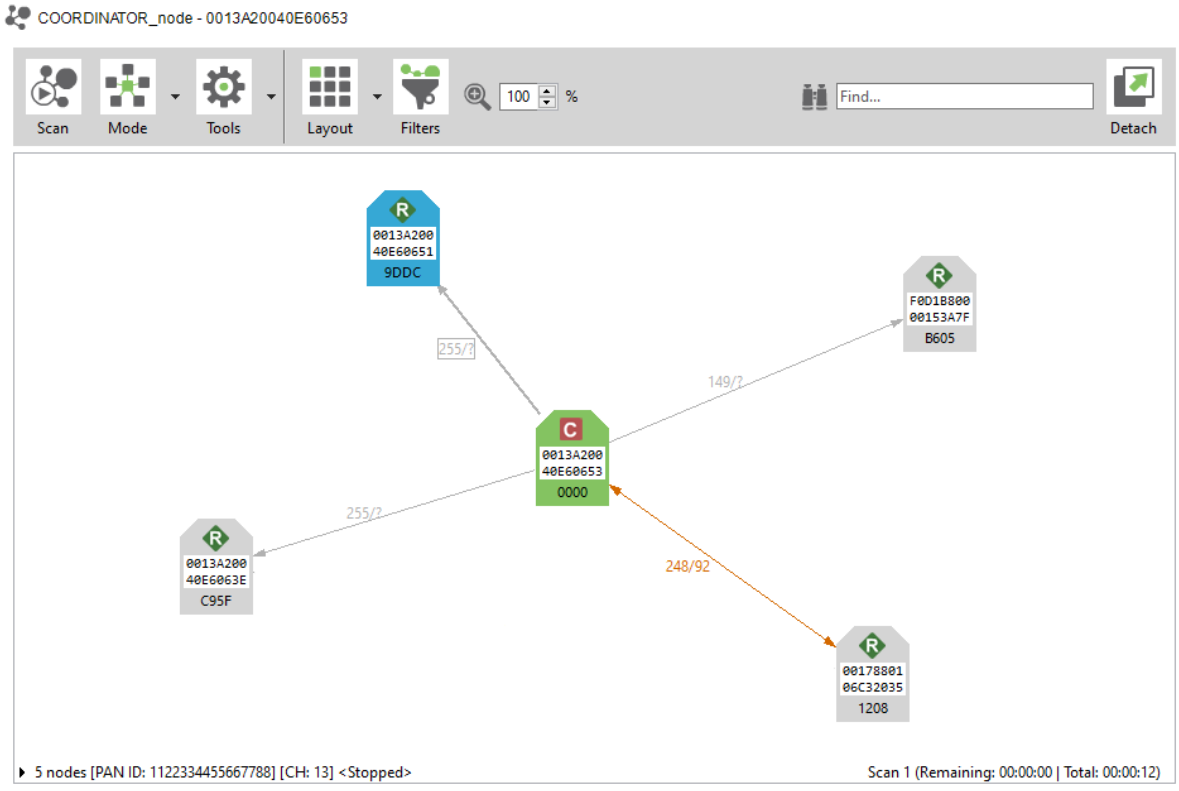

Figura 31. Topología en estrella de la red ZigBee visualizada en el *networking working mode* del XCTU.

Como se puede observar en la *Figura 31*, se obtiene una topología en estrella de la red ZigBee, tras la configuración de cada uno de los nodos que la integran. Los dispositivos finales aparecen registrados en el visualizador del *networking working mode* como *routers* debido a que cada uno de ellos están vinculados directamente solo con el nodo coordinador (por definición en la topología en estrella). Sin embargo, a todos los efectos funcionan como dispositivos finales en el presente escenario.

Por último, cabe destacar que las acciones descritas tanto en este subapartado como en el anterior, solo habrá que realizarlas una vez. Permiten dejar a los dispositivos ZigBee totalmente configurados y listos para su exportación al entorno domótico definitivo.

6.1.4 Generación de tramas ZigBee

Pinchando en la pestaña *consoles working mode* del XCTU, se tendrá acceso al analizador de tráfico de red y al generador de tramas ZigBee. Estas dos funcionalidades, han permitido experimentar directamente con los diferentes campos de las tramas. Aprovechado que la red ya está creada y que el PC está actuando como nodo coordinador, el XCTU posibilita la total personalización, generación y envío de tramas y, por lo tanto, el estudio del comportamiento y respuesta de cada uno de los dispositivos finales ante las órdenes enviadas desde el PC.

El formato de la trama ZigBee

Se van a enviar y recibir tramas del tipo *Explicit Addressing Zigbee Command frame* [22] [23] [24] [25]. Este tipo de trama se utiliza para el envío de datos a un destino específico utilizando los campos de direccionamiento de la capa de aplicación de la torre del protocolo ZigBee. Es la ideal para una comunicación de este tipo y está formada por los siguientes campos:

- **Delimitador de inicio de trama**: Toma siempre el valor `0x7E`.
- **Longitud**: Número de bytes totales de la trama (sin contar los del delimitador). Su valor es auto calculado por el generador de tramas.
- **Tipo de trama**: `0x11`, correspondiente al *Explicit Addressing Zigbee Command frame*.
- **ID de trama**: Asocia la petición a sus respuestas. Su valor es escogido arbitrariamente.
- **Dirección de 64-bit destino**: `0x0017880106C32035` para el envío hacia la bombilla, `0xF0D1B80000153A7F` para el adaptador de enchufe y `0x0013a20040E6063E` para el ventilador.
- **Dirección de 16-bit destino**: No usada. Se deja a `0xFFFE` (valor por defecto).
- ***Source endpoint***: Identifica el dispositivo transmisor hacia la tarjeta ZigBee que finalmente envía la trama. Se aplica el valor `0xE8` para comunicaciones por puerto serie (caso de módulos Xbee conectados al waspmote, arduino o PC).
- ***Destination endpoint***: Identifica el dispositivo receptor de la trama transmitida desde su tarjeta ZigBee. De nuevo, se establece `0xE8` para destinos XBee, mientras que la bombilla requiere el valor `0x0B` por parte de su fabricante. En caso de duda (es información difícil de encontrar, ya que el fabricante no siempre la proporciona), se puede aplicar el valor `0xFF` de difusión, siempre que la funcionalidad del dispositivo lo permita. Esto es lo que se ha hecho para el adaptador de enchufe.
- **ID de clúster**: ID del clúster usado según el tipo de dato que contenga la trama. Un clúster define un conjunto de especificaciones y comandos según la aplicación o funcionalidad requerida en la comunicación ZigBee. Se usa el ID de clúster `0x0011` para transmisiones por puerto serie (destinos XBee), el `0x008` para el control de niveles (caso de la bombilla para regular el brillo) y el `0x006` para la conmutación on/off (usada para apagar y encender el adaptador de enchufe).
- **ID de perfil**: ID del tipo de comunicación según la aplicación en el que se establece. `0x0104` para aplicaciones de domótica.
- **Radio de difusión**: Establece el número máximo de saltos que puede atravesar una transmisión de difusión. No aplica en este caso, por lo que se deja a `0x00`.
- **Campo de opciones**: Campo de bits de banderas para habilitar opciones adicionales. Se deja a `0x00` (no se usa ninguna).
- **Carga útil**: Conjunto de datos que contienen la orden a transmitir por parte del nodo coordinador.

  Para la bombilla se enviará `0x010004XX0000100010`, siendo `XX` el nivel del brillo en un rango de

intensidad de 0-255.

En el caso del adaptador de enchufe enviaremos `0x01000Y0010`, donde `Y` puede ser 0 o 1 según si queremos habilitar o no la alimentación a la corriente del dispositivo enchufado al adaptador.

Para el ventilador, la carga útil podrá ser de `0x30`, `0x31`, `0x32` o `0x33`, que no son más que las conversiones a hexadecimal de los caracteres ASCII "1", "2", "3" y "4", respectivamente. Dichos valores de *payload*, vendrían a indicarle al waspmote que lo controla el nivel de velocidad a aplicar en el motor, mediante el aumento o disminución del ciclo de trabajo del PWM, como ya se ha comentado anteriormente.

- ***Checksum***: Auto calculado por el generador de tramas.

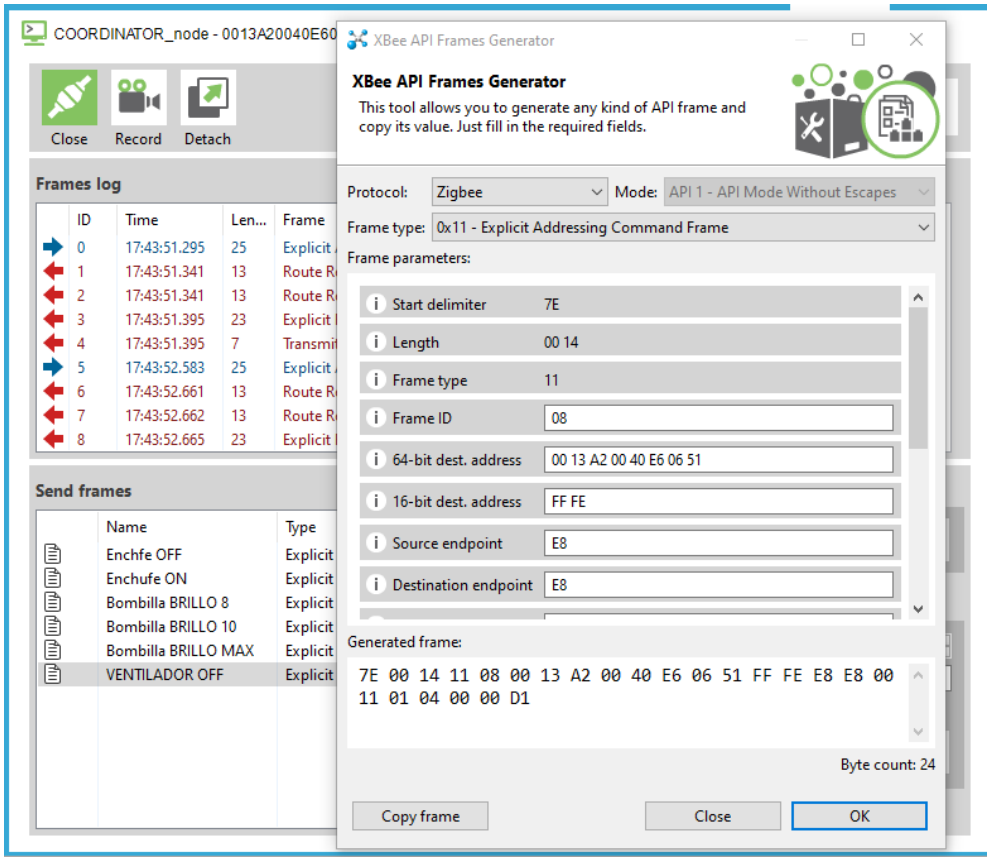

Figura 32. Ventana del generador de tramas en el *consoles working mode* del XCTU.

En la *Figura 32* se muestra la pestaña correspondiente con el generador de tramas. Se observa el alto grado de nivel de configuración y edición de estas, siendo posible elegir el valor en hexadecimal de cada uno de los parámetros que la forman, así como el tipo de trama, además de la versión concreta del protocolo ZigBee a aplicar en la comunicación. Detrás de la pestaña del generador de trama, pueden verse algunas tramas de prueba generadas, cuyo nombre corresponden con las acciones a realizar, junto con el nombre del dispositivo final ZigBee a las que afectan. También se observa la ventana de *logs*, la cual registra información adicional sobre las tramas enviadas y sus respuestas asociadas.

Por otro lado, cabe mencionar el interesante uso de la opción *Save frames list* que ofrece la sección *consoles working mode* del XCTU. Pinchando en dicho botón, se abrirá una pestaña con el sistema de archivos local, donde es posible guardar en la ubicación deseada, la lista con las distintas tramas generadas en la sesión actual. Esta lista podrá ser cargada directamente en sesiones de prueba futuras, con las comodidades que ello conlleva.

Finalmente, una vez que se ha comprobado que todos los dispositivos responden adecuadamente a las órdenes contenidas en las tramas probadas y se ha conformado el ecosistema ZigBee al completo, es momento de migrarlo al sistema domótico final a implementar (ver *Figura 25*). Para ello, desconectaremos el módulo XBee

del adaptador USB y lo integraremos en el verdadero nodo coordinador del sistema completo: el dispositivo formado por el arduino y sus periféricos (descrito en el *Capítulo 5*).

6.2 Desarrollo software del nodo coordinador

En este subapartado, se describirá el comportamiento y las tareas llevadas a cabo, así como las librerías y funciones usadas por el programa o *sketch* editado y cargado en el nodo coordinador mediante la IDE de Arduino, tras la conexión de este al PC por medio de un cable USB tipo B.

6.2.1 Descripción de las librerías incluidas

ArduinoJson.h

El servicio de *backend* del sistema domótico almacena la información en formato Json, por lo que resulta imprescindible el uso de una librería que incluya herramientas para facilitar la lectura y el manejo de dicho formato.

La librería Arduino Json [26], proporcona el objeto JsonDocument que abstrae un documento Json y permite su directa escritura (serialización) y lectura/parseo (deserialización). Dicho objeto podrá ser de tipo estático (StaticJsonDocument) o dinámico (DynamicJsonDocument). StaticJsonDocument se genera durante la compilación y se almacena en la pila ocupando un tamaño de memoria fijo según se le indique, mientras que DynamicJsonDocument emplea memoria dinámica almacenándose con el resto de variables. Este último tipo es algo más lento, pero permite escalabilidad y la generación de ficheros de mayor tamaño.

El uso básico de esta librería es sencillo: en primer lugar, se crea un JsonDocument del tipo deseado y a continuación, se usan los métodos serializeJson o deserializeJson según si se desee crear un nuevo objeto Json o interpretarlo, respectivamente.

Ethernet.h

La librería Ethernet [27], está diseñada para trabajar con la *shield* que le da nombre. Permite conectar al arduino a internet y trabajar tanto como de servidor, escuchando peticiones, como de cliente, realizándolas.

Los métodos usados de esta librería en el código son los siguientes:

- *begin*: Indica al servidor que comience a escuchar peticiones entrantes.
- *available*: Permite el acceso a un cliente que está tratando de conectarse al servidor y tiene datos disponibles para leer.
- *connected*: Devuelve `true` si algún cliente sigue conectado y `false` si no.
- *read*: Lee el siguiente byte del mensaje de petición recibido por parte del cliente o del servidor destino según el caso.
- *stop*: Desconecta al cliente del servidor.

SPI.h

Permite la comunicación con dispositivos SPI (*Serial Peripheral Interface*), con el arduino como dispositivo maestro. La incorporación de esta librería es necesaria para el funcionamiento de la *shield* Ethernet, la cual utiliza el puerto SPI para su comunicación con el arduino.

DHT.h

La librería DHT.h [28] permite mediante sus métodos suministrados, la comunicación sencilla con el sensor DHT11 para obtener la información de temperatura y humedad recogida y enviada por el mismo.

6.2.2 Descripción del comportamiento del código y sus funciones

La primera tarea que realiza el programa es la importación de las cuatro librerías citadas anteriormente. A

continuación, se declaran e inicializan las variables globales, entre las que cabe destacar la cadena que contiene la dirección IP del PC donde se ejecuta el servicio de *backend* Json.

Tras esto, comienza la ejecución de la primera función básica de Arduino: la función *setup*.

La función *setup*

Inicializa, en este orden:

- El LED de señalización de estado,
- el monitor serial,
- el sensor de humedad y temperatura DHT11
- y el servidor de Arduino, a la escucha, en la dirección IP que le otorguemos (importante que se encuentre dentro de la subred de la vivienda), de peticiones HTTP por parte del dispositivo móvil avisando de que consulte el servidor.

Por último, se llama a la función *flashLed*, la cual hace que parpadee el LED de forma rápida por tres veces indicando que ya se encuentra en funcionamiento el nodo coordinador a la espera de recibir órdenes de la aplicación móvil.

La función *flashLed*

Producirá el parpadeo del LED el número de veces y con el periodo que le indiquemos, señalizando éxito o error. Esta función será llamada también en varias ocasiones por el resto de las funciones del programa. Se ha establecido que los parpadeos rápidos indiquen éxito, mientras que los lentos alerten de alguna situación anómala.

La función *loop*

Es la otra función básica de Arduino. Se ejecuta en bucle de manera ininterrumpida estableciendo la rutina que definirá el comportamiento del nodo coordinador. Dicha rutina se basa en esperar a la conexión de algún cliente y responder en consecuencia.

Si el cliente accede al servidor del arduino mediante una petición HTTP GET, este responderá simplemente con el mensaje "Arduino SERVER. ID elemento inteligente actualizado: 0". Puede ser interesante acceder de esta manera al servidor del arduino (por ejemplo, a través de un navegador web, desde un dispositivo situado en la misma subred) para verificar la disponibilidad del mismo.

Por el contrario, si el cliente envía una petición de tipo HTTP PUT incluyendo un mensaje en formato Json conteniendo la clave ID y su valor, el nodo coordinador guarda dicho mensaje y comprueba el ID recibido. En el caso de que este tenga un valor de 1, el cliente estará preguntando por los datos de humedad y temperatura ofrecidos por el sensor que implementa el nodo coordinador. Se llamará a los métodos que permiten comunicarse con el sensor para obtener dicha información, se comprobará la integridad de los datos obtenidos, y se enviarán en forma de respuesta a la petición HTTP PUT recibida. Si el ID es mayor o igual que 2, el nodo coordinador responde igualmente a la petición HTTP a modo de ACK y llama a la función *httpRequest*.

Cabe aclarar que se espera que estas peticiones HTTP PUT con la información del ID contenida en el mensaje provengan de la aplicación móvil, la cual está expresamente preparada para el envío de peticiones con tal formato.

La función *httpRequest*

La petición HTTP PUT recibida, ha avisado al nodo coordinador de que el valor de algún parámetro del dispositivo domótico ZigBee con la ID indicada, ha sido modificado en el servicio de *backend*. En la función *httpRequest*, el arduino actuará como cliente de cara ha dicho servidor, realizando una petición HTTP GET hacia la IP del mismo, con la ruta donde se encuentran almacenados los parámetros del elemento domótico en cuestión. Si se recibe una respuesta satisfactoria, se procederá a la deserialización del mensaje contenido en la respuesta. Tanto si se ha producido un error en la deserialización, como si la respuesta no ha sido satisfactoria, se señalizará el error llamando a la función *flashLed*.

El siguiente y último paso será la creación y envío de las tramas ZigBee, emulando las pruebas realizadas en el XCTU:

- Si el ID recibido es igual a 2, se enviará una trama ZigBee que modifique el brillo de la bombilla en 8 posibles niveles según el valor indicado en el mensaje deserializado.
- En el caso de que el ID valga 3, se enviará hacia el adaptador de enchufe una trama ZigBee que lo enciende si se encontraba apagado y viceversa.
- Por último, un ID igual a 4 generará una trama ZigBee que aumentará o disminuirá en 4 niveles diferentes la velocidad del ventilador.

Si cada envío de trama ZigBee es exitoso, se señalizará mediante un parpadeo rápido del LED.

Finalmente, se volverá a ejecutar la función *loop* completándose el ciclo del programa.

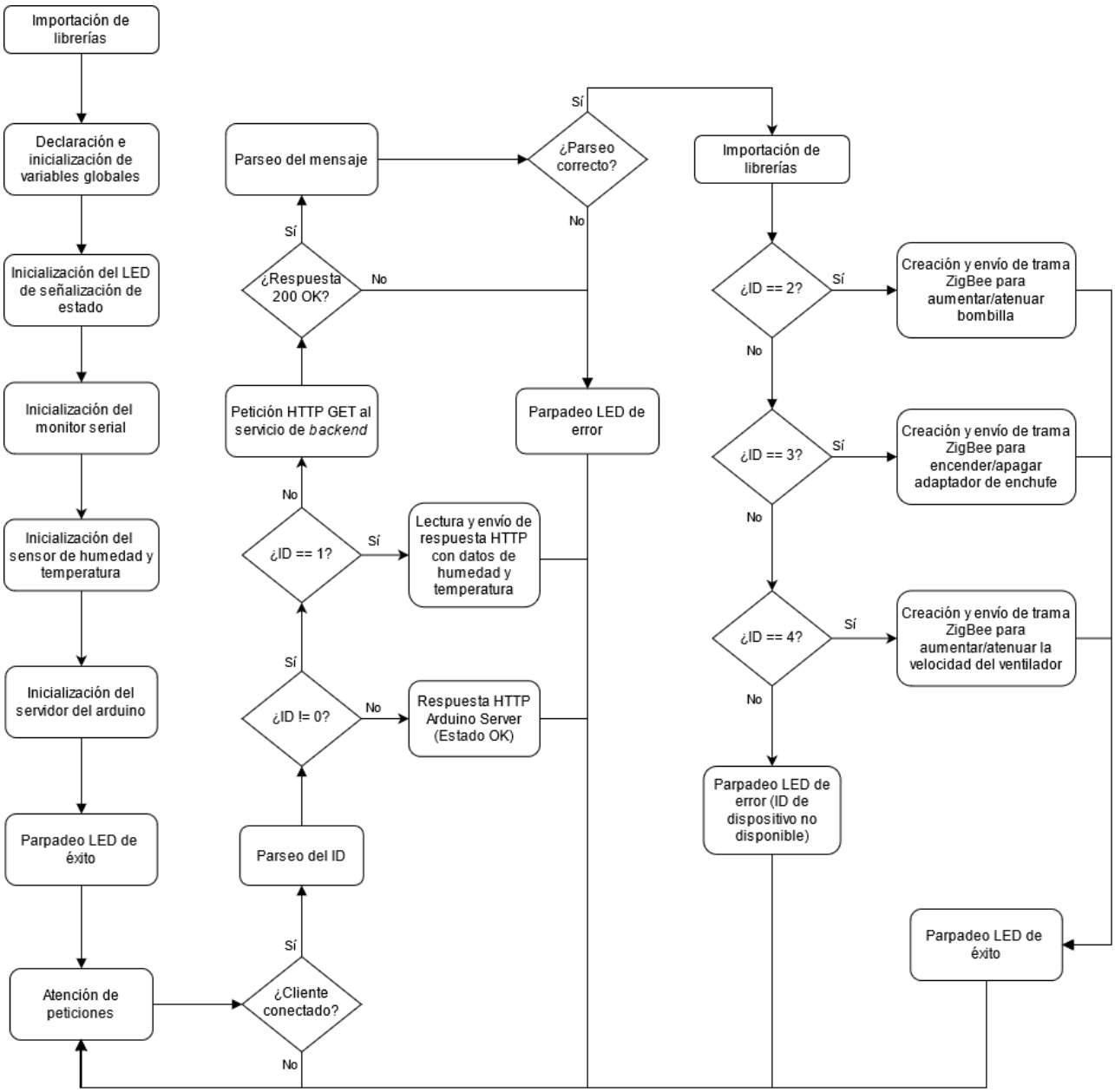

Figura 33. Diagrama de flujo que representa el comportamiento del nodo coordinador.

6.3 Implementación del servicio de *backend* del sistema

Como ya se ha comentado, la API REST que actuará como servicio de *backend* del sistema, adoptará el formato Json y estará basado en Node.js. Será ejecutado por línea de comandos en el equipo servidor, que en este caso será un PC.

En este apartado, se comentarán las herramientas software aplicadas en su código.

6.3.1 El *framework* Express.js

Express.js [17], o simplemente Express, es un *framework* usado en Node.js para facilitar el desarrollo de servicios de *backend*. Ofrece una serie de métodos básicos que corresponden con cada uno de los diferentes tipos de peticiones HTTP (GET, POST, PUT, DELETE…). El servidor podrá manejar de acuerdo con lo programado, cada petición HTTP del cliente según el método usado en la misma y la ruta del recurso solicitado.

Los recursos, se almacenarán al principio del código en forma de objetos tipo Json, en los que cada uno de ellos corresponde con un elemento o dispositivo domótico. El servicio programado será capaz de manejar los métodos HTTP GET y PUT.

En el caso de las peticiones tipo GET, se responderá a la ruta "`/api`" con un "`Hello World!`" y a la ruta "`/api/elements`" con el listado de los recursos Json almacenados, correspondientes a cada elemento o dispositivo domótico implementado en el sistema.

Por otro lado, tanto el dispositivo controlador del sistema, mediante la aplicación móvil, como el nodo coordinador, acceden como clientes a la ruta "`/api/elements/:id`". Esta ruta podrá ser visitada, bien para consultar los parámetros del elemento domótico en cuestión (siendo "`/:id`" su identificador correspondiente) o bien para modificar alguno de ellos, según si la petición es de tipo HTTP GET o PUT.

Por último, se establece la escucha del servidor en el puerto dado mediante el método *listen*. Es importante recalcar que para obtener alguna de las respuestas indicadas, es esencial que el cliente se encuentre en la misma subred que el servidor y que la dirección IP del mismo (junto con el puerto a la escucha) preceda a la ruta en la URL de la petición.

El paquete Joi

Como buena práctica de seguridad y fiabilidad, es necesario el uso de una herramienta que procese y verifique la información recibida por parte de los clientes. El paquete Joi proporciona un esquema de validación al servicio de *backend*. Gracias a ello, el servidor será capaz de verificar el correcto formato de la información almacenada o el tipo de los parámetros a modificar por parte del cliente mediante las peticiones HTTP PUT, a las que se responderá con un código de estado HTTP 400 (*Bad Request*), en caso de alguna incorrección.

Por otro lado, el servidor responderá con códigos de estado HTTP 404 (*Not Found*) cuando el cliente trate de acceder recursos inexistentes. Se incluirá el mensaje "`The element with the given ID was not found.`" en el caso de que se solicite una "`/:id`" con la que el servidor no cuenta.

6.4 Implementación de la aplicación móvil

El dispositivo controlador del sistema correrá la aplicación móvil o servicio de *frontend* desarrollado en React Native [29]. Aparte de React Native y su uso junto a Expo, se han incluido una serie de herramientas y paquetes adicionales que han facilitado la implementación de las funcionalidades requeridas.

Por otro lado, debido a su complejidad, la programación de la aplicación se ha dividido y estructurado en diferentes ficheros y directorios para favorecer su modularidad, escalabilidad y eficiencia a la hora de abordar su desarrollo.

6.4.1 Estructura del programa

Directorio *api*

Contiene los ficheros encargados de establecer el cliente del servicio de *backend* del sistema y del servidor del nodo coordinador. La aplicación móvil se conectará a dichos servidores mediante peticiones HTTP PUT, para modificar los parámetros de los dispositivos domóticos integrados en el sistema, y para avisar de dicha acción al nodo coordinador, proporcionándole la ID del mismo. También para solicitar directamente la información de humedad y temperatura.

Es muy importante establecer correctamente tanto la dirección IP como los puertos en los que se encuentran a la escucha cada uno de dichos servidores, para un correcto funcionamiento.

Directorio *app*

Incorpora los siguientes subdirectorios:

- *components*: En él, se definen los diferentes elementos o componentes estéticos y funcionales que serán importados por cada una de las pantallas que incluye la aplicación. Dichos componentes son desde botones, a los formatos de pantalla y texto.
- *config*: Contiene los ficheros que definen el estilo de la aplicación a nivel de colores y fuentes de texto.
- *navigation*: Define la navegación entre las distintas pantallas disponibles.
- *screens*: Incluye los ficheros de cada una de las pantallas renderizables.

Directorio *node_modules*

Guarda las configuraciones de los paquetes o librerías externas descargadas e incluidas por los ficheros de la aplicación.

Fichero *App.js*

Se encuentra en el directorio raíz del proyecto y es el primer fichero que se renderiza al inicializar la aplicación. A partir de él se van importando el resto de pantallas navegables y de componentes con los que el usuario podrá interactuar.

6.4.2 Descripción de los principales herramientas y paquetes usados

React Navigation

La gestión de la presentación y transición entre pantallas se realiza mediante lo que se conoce como la navegación. React Navigation ofrece las herramientas necesarias para la gestión de rutas y navegación en una aplicación React Native. Así mismo, está pensado para facilitar el intercambio de datos en forma de parámetros entre cada una de las pantallas ofrecidas.

Ofrece tres tipos de formatos de navegación:

- *Stack Navigator*: Permite la transición directa entre pantallas, mediante el uso de botones o componentes similares (equivalente a la tradicional navegación en navegadores web).
- *Tab Navigator*: Implementa una barra con pestañas que es común a todas las pantallas, permitiendo seleccionar en todo momento la pantalla deseada.
- *Drawer Navigator*: Incluye un cajón oculto en forma de menú arrastrable que funciona como cualquier otra pantalla. Se usa normalmente para incluir opciones adicionales o de configuración.

Api Sauce

La librería Api Sauce proporciona los métodos necesarios para el establecimiento de conexiones y realización de peticiones hacia servidores.

Permite el manejo eficiente de peticiones y respuestas en forma de objetos, así como la sencilla gestión de respuestas con códigos de error. De esta manera, una petición siempre será resuelta incluso aunque sea

errónea, por lo que permite evitar el uso de los típicos bloques de código *try-catch* y comprobar directamente el objeto de la respuesta.

React Hooks

Los React Hooks son herramientas funcionales que permiten sustituir y simplificar algunas tareas que tradicionalmente debían ser realizadas mediante clases. Se tratan de funciones predefinidas que permiten "enganchar" elementos o componentes funcionales a características propias de una de clase. Es decir, proporcionan un estado y un ciclo de vida a estos componentes evitando así el tener que crear y usar una clase específica para cada componente que lo requiera.

Los *hooks* usados en el código son:

- useState: Conocido como *hook* de estado. Devuelve una variable de estado local, a la que se "engancha" un componente funcional, y una función necesaria para actualizar el valor de dicho estado. Dicha función se asigna a eventos realizados sobre los componentes funcionales (por ejemplo, la pulsación de un botón), provocando la actualización de la variable de estado local.
- useEffect: Conocido como *hook* de efecto. Agrega efectos secundarios a un componente funcional dado, es decir, permite realizar modificaciones en su estado cada vez que se produzca un renderizado, y/o cambie el estado de otra variable dada, de la que depende directamente.
- useNavigation: Es ofrecido por la librería React Navigation. Proporciona un objeto que posibilita la navegación en aquellos contextos en los que no se tenga acceso directo a la misma.
- useRoute: También proporcionado por la librería React Navigation. Devuelve un objeto de ruta que permite a las pantallas destino de la navegación, el acceso a los parámetros compartidos por la pantalla de origen.

6.4.3 Ficheros de navegación, pantallas ofrecidas y descripción funcional de la aplicación móvil

Lo primero que se renderiza tras abrir la aplicación es el fichero App.js, el cual importa directamente AppNavigator.js. Dicho fichero de navegación crea el objeto Tab mediante la llamada al método *createBottomTabNavigator* de la librería React Navigation. El objeto Tab, define un *Tab Navigator* que incluye una barra inferior con cuatro pestañas mediante las que navegar entre las diferentes pantallas principales. Cada pantalla principal corresponde con una estancia de la casa: el salón, la cocina, el dormitorio y el cuarto de baño.

Es importante señalar que en la fase en la que se encuentra el proyecto en el momento de la entrega, la pantalla del salón es la única completamente funcional de la aplicación hasta el momento. Lo mostrado en las pantallas correspondientes con el dormitorio, la cocina y el cuarto de baño es meramente estético, a modo de demostración. La pantalla del salón estará asociada, por tanto, al resto del sistema domótico y será posible hacer uso de cada uno de sus componentes para enviar órdenes y controlar los diferentes dispositivos domóticos situados en dicha estancia.

Aparte de las cuatro pantallas principales asociadas a cada estancia del domicilio, la aplicación implementa una quinta pantalla accesible mediante la pantalla del salón. Esta pantalla muestra la información sobre los datos de la humedad y temperatura recogidos y enviados directamente desde el nodo coordinador, previa solicitud por parte de la aplicación móvil.

Los ficheros que definen las transiciones entre las distintas pantallas de la aplicación y que posibilitan, por tanto la navegación, son los siguientes:

AppNavigator.js

Como ya se ha indicado, es el primer y único elemento renderizado por el fichero App.js. Además de definir las pantallas hacia las que se puede acceder con cada pestaña incluida en el *Tab Navigator* que ofrece, en este fichero se configurará estéticamente cada una de estas pestañas. Para ello, se incluirán una serie de iconos vectoriales a situar en cada pestaña, que simbolizan cada estancia con la que se corresponden. En cuanto a configuración estética, también es posible modificar los colores y títulos de cada pestaña, así como la cabecera superior de cada pantalla. Esta última opción se ha dejado por defecto, mientras que la definición de la paleta de colores se ha derivado al fichero navigationTheme.js.

SalonNavigator.js

La pantalla del salón, cuenta también con la subpantalla del sensor de humedad y temperatura. El fichero SalonNavigator.js, crea un *Stack Navigator* con la instanciación del objeto Stack (mediante la llamada al método *createStackNavigator*). El objeto Stack, define por tanto un navegador formado por la pantalla del salón y la subpantalla del sensor de humedad y temperatura, la cual se despliega en forma de "presentación modal", tras interaccionar con el componente adecuado. La presentación modal permite el cierre directo mediante arrastre de la subpantalla desplegada. El uso de dicho estilo de presentación junto con su configuración estética, se realizarán en el fichero SalonNavigator.js, mientras que los pormenores de la asociación con el componente que despliega esta subpantalla, se definirá en el fichero del propio componente.

El fichero SalonNavigator.js se asocia a la pestaña del salón en el *Tab Navigator* configurado en AppNavigator.js.

navigationTheme.js

Define los colores que se aplicarán en cada una de las pestañas de la barra común a todas las pantallas que ofrece el *Tab Navigator*, definido en el fichero AppNavigator.js.

A continuación, se detallarán cada una de las pantallas que conforman la aplicación:

SalónScreen.js

Es la pantalla principal de la aplicación que aparece renderizada tras la inicialización de esta. Importa y distribuye por la pantalla los componentes correspondientes a cada elemento domótico integrado en el salón: la bombilla, el adaptador de enchufe, el ventilador, y el sensor de temperatura y humedad. Exporta dinámicamente el nombre y el ID que se asociará a cada componente, y que serán renderizados y visualizados por el usuario.

enDesarrolloScreen.js

Será la pantalla que se asociará desde el AppNavigator.js a las pestañas del dormitorio, la cocina y el cuarto de baño. Como ya se ha mencionado, dichas estancias no tienen asociadas ningún componente funcional, por lo que enDesarrolloScreen.js se limitará a mostrar y configurar estéticamente un componente de texto que es común para dichas estancias y que incluye el mensaje: "Pantalla en desarrollo".

SensorScreen.js

Incorpora el *hook* useRoute, el cual le permite recibir como parámetros de navegación los datos sobre la humedad y temperatura, recibidos por parte del botón que despliega a esta subpantalla. Dichos datos son mostrados en esta pantalla mediante un componente de texto, cuya configuración estética se realiza también en este fichero.

En la figura siguiente, puede observarse un esquema que representa las rutas de navegación disponibles entre las distintas pantallas de la aplicación. Cabe aclarar que, aunque en la figura no quede del todo bien reflejado, las pestañas de la barra inferior del *tab navigator* son completamente funcionales en las cuatro pantallas principales de la aplicación (las correspondientes a cada estancia de la casa). Es posible por tanto, acceder también desde la pantalla de la cocina, el dormitorio o el cuarto de baño, a cualquier otra pantalla, menos a la de que muestra la información sobre la humedad y la temperatura (la cual solo es accesible desde la pantalla del salón).

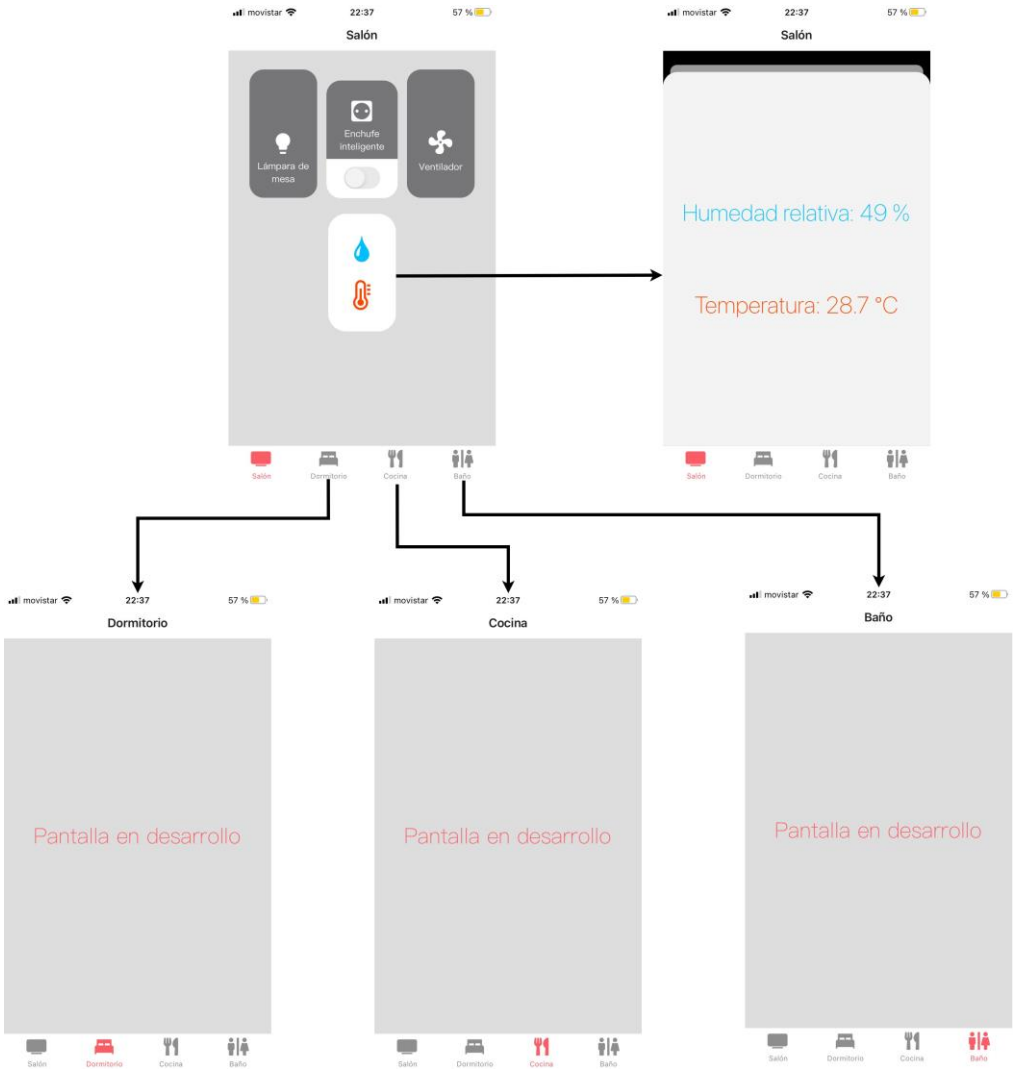

Figura 34. Esquema de las rutas de navegación entre las pantallas disponibles en la aplicación.

6.4.4 Ficheros del directorio *api*

El directorio *api*, introducido en el subapartado *Estructura del programa* de este mismo capítulo, contiene los siguientes ficheros:

client.js

Crea el cliente del servicio de *backend* del sistema domótico mediante el método *create* de la librería Api Sauce, al que se le proporciona la URL base (con puerto alternativo 3000 y dirección IP del PC) en la que está disponible dicho servidor.

clientArduino.js

Crea el cliente del servidor disponible en el arduino. Se usa también el método *create* de la librería Api Sauce, al que se le proporciona en este caso una URL base con la IP asociada al arduino y el puerto estándar para el protocolo HTTP (el 80).

callApi.js

Este fichero importa el cliente creado en client.js y define los métodos con los que se realizarán las peticiones HTTP hacia el servicio de *backend* del sistema. Dichos métodos son *putElement* y *putSensorElement*. Ambos emplean peticiones HTTP PUT para modificar los valores de los parámetros contenidos en el servicio de *backend*.

callArduino.js

Importa el cliente creado en clientArduino.js y define los métodos con los que se realizarán las peticiones HTTP hacia el servidor del arduino. Estos métodos son *putArduino* y *putArduinoSensor* y emplean también peticiones HTTP PUT, en este caso para enviar avisos al arduino justo después de haber modificado un parámetro del servicio de *backend*, o para solicitar directamente la información ofrecida por el sensor de humedad y temperatura que incorpora el nodo coordinador. El uso tanto de *putArduino* como de *putArduinoSensor*, va por lo tanto siempre de la mano de los métodos *putElement* y *putSensorElement*.

Por otro lado, el motivo en este contexto de usar HTTP PUT en lugar de HTTP GET, es poder incluir junto a la petición el ID del dispositivo domótico sobre el que se desea realizar alguna acción.

6.4.5 Componentes funcionales de la aplicación móvil

La pantalla del salón incluye una serie de elementos o componentes funcionales que importan los métodos definidos en los ficheros del directorio *api*. Gracias a esto, la interacción del usuario con dichos componentes supondrá el envío de mensajes HTTP, mediante los cuales la aplicación controlará los distintos dispositivos domóticos. Dichos componentes representan, en definitiva, la unión directa con el resto del sistema domótico.

La programación y el comportamiento, así como la estética de los mismos, se define en los siguientes ficheros:

PlugButton.js

Se trata del botón que controla el adaptador de enchufe inteligente.

Importa los métodos *putElement* y *putArduino* y el *hook* de estado. Incluye un *switch* cuyo estado puede ser apagado (`false`) o encendido (`true`), y que cambia con cada cambio de posición, llamando a la función setIsEnabled creada mediante el uso del useState *hook*. A su vez, con cada cambio de posición del *switch*, se llama en primer lugar al servicio de *backend* para dejar reflejado el nuevo estado mediante el método *putElement*, al que se le pasa como parámetro tanto el estado como el ID 3 correspondiente con el adaptador de enchufe. En segundo lugar, se llama a *putArduino* incluyendo también la ID 3.

Figura 35. PlugButton.js encendido.

LightButton.js

Es un botón en forma de barra deslizadora que controla la bombilla inteligente.

Importa, además de los ficheros del directorio *api* y el *hook* de estado, el *hook* de efecto. Funciona de forma similar al botón anterior, con la diferencia de que en este caso el *hook* de estado almacena el nivel de brillo de la bombilla (del 0 al 7). Con cada cambio de nivel de brillo, se enviarán las mismas peticionas descritas en el caso anterior, pero con los parámetros correspondientes a la bombilla (ID=2 y parámetro adicional con el nivel de brillo para *putElement*) y se coloreará en mayor o menor medida la barra deslizadora.

Como función adicional, el useEffect *hook* permite comprobar los cambios que se produzcan el brillo para que, en caso de que la bombilla tome el nivel de brillo 0, se le asigne el valor `false` a la variable de estado de la bombilla.

Figura 36. LightButton.js con el nivel máximo de brillo.

FanButton.js

Controla el ventilador.

Es similar al LightButton.js, con la diferencia de que permite modificar la velocidad del ventilador, donde LightButton.js variaba el brillo de la bombilla. Se permite aplicar cuatro niveles de velocidades diferentes. El valor del ID correspondiente con el ventilador es igual a 4.

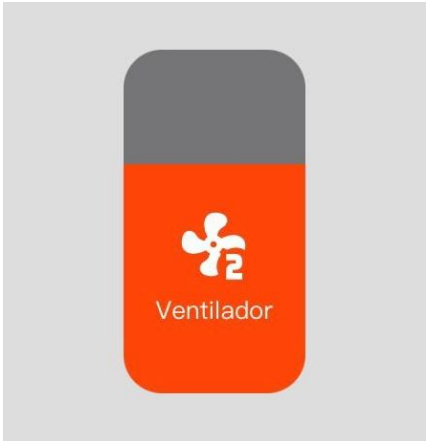

Figura 37. FanButton.js al nivel 2 de velocidad.

SensorButton.js

Permite solicitar y mostrar los niveles de humedad relativa y temperatura en grados centígrados existentes en el salón del domicilio y proporcionados por el sensor que incluye el nodo coordinador.

Importa el useNavigation *hook* y los ficheros del directorio *api*. Se trata de un botón que, tras su pulsación, llama al método *putArduinoSensor* pasándole como parámetro el ID 1. Como respuesta a la petición HTTP que acarrea la llamada a ese método, se recibe por parte del nodo coordinador, un objeto en formato Json que contiene los datos de la humedad y temperatura, los cuales serán reenviados mediante la sucesiva llamada al método *putSensorElement*, para la modificación de dichos valores en el servicio de *backend* del sistema.

Finalmente, gracias al uso del *hook* de navegación y el método *navigate* que proporciona, tras la pulsación del botón, se desplegará la pantalla SensorScreen.js que recibirá también los datos de humedad y temperatura y permititá al usuario la visualización de los mismos.

Figura 38. SensorButton.js

En cuanto a estética se refiere, cada componente emplea un color diferente, como se puede ver en cada una de las cuatro figuras anteriores. Dichos colores servirán para indicarle al usuario el estado de los diferentes botones. Así mismo, se usa un icono vectorial distinto para cada uno de ellos, que para el caso de LightButton.js y FanButton.js, también cambia según el nivel de brillo o velocidad del que dispongan en cada momento.

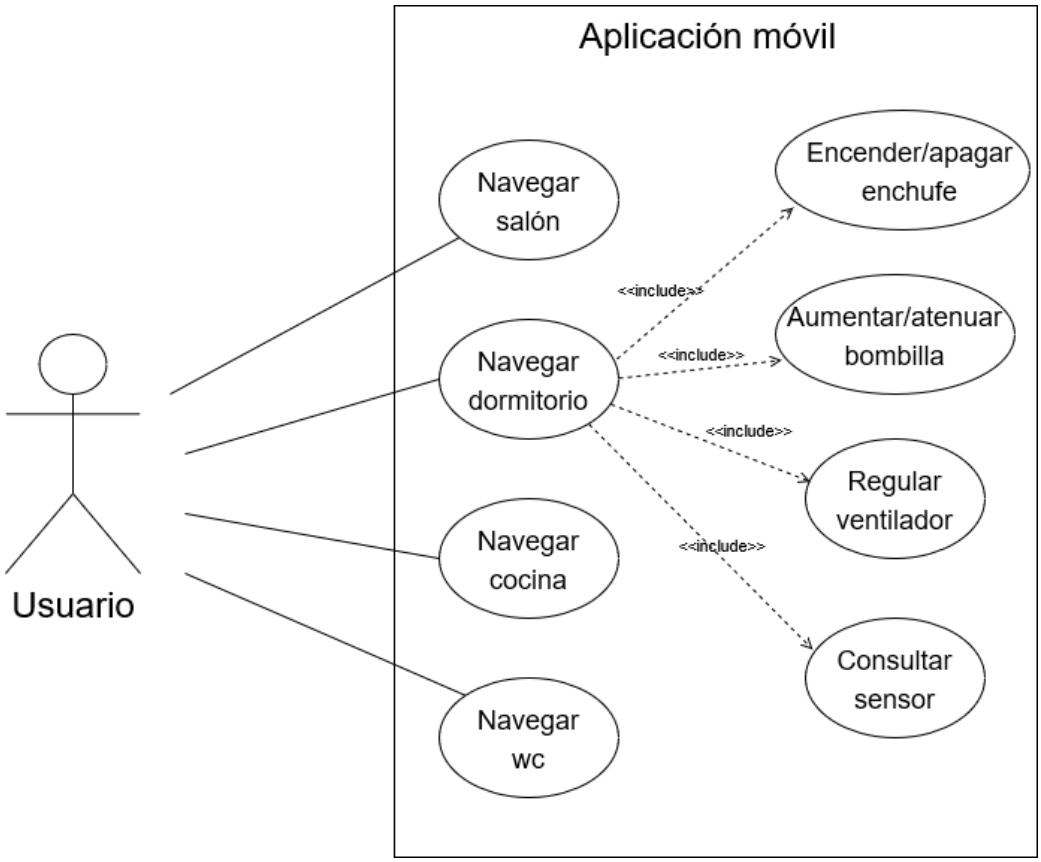

Figura 39. Diagrama de casos de uso de la aplicación móvil.

6.5 Implementación software del ventilador

El programa que implementa el nodo ventilador es cargado tras la conexión del waspmote al PC mediante un cable USB. Se usará para ello la IDE de Waspmote. Incluye la librería WaspXBeeZB.h que, como su propio nombre indica, proporciona las funciones necesarias para permitir al waspmote interaccionar con el módulo XBee que incorpora.

La estructura del programa o sketch es idéntica a la de los de Arduino y consta de las siguientes funciones:

setup

Se ejecuta una sola vez tras la inclusión de la librería WaspXBeeZB.h y la creación de la variable global que recoge los errores de la comunicación ZigBee. Realiza las siguientes operaciones en orden:

- Proporcionar un ciclo de trabajo nulo al pin `DIGITAL1` que proporciona el PWM que permitirá el control de la velocidad del ventilador. Esto se realiza para que el motor que incorpora el ventilador permanezca apagado al principio.
- Activar el pin generador de 5V para alimentar el módulo del ventilador.
- Iniciar el monitor serial y el módulo XBee.
- Llamar a la función *checkNetworkParams*.

checkNetworkParams

Intenta obtener un ID de red ZigBee adecuado y espera hasta obtener una indicación de asociación satisfactoria. Cuando la obtiene, imprime por monitor serial un mensaje de éxito seguido de unos parámetros relativos a la conexión (ID de red y número de canal).

loop

Función que se ejecuta en bucle a la espera de la recepción de tramas ZigBee. Imprime el mensaje (carga útil) contenido si alguna de ellas es recibida. Según si el valor del mensaje recibido sea 0, 1, 2 o 3, el waspmote aplicará un ciclo de trabajo mayor o menor al pin `DIGITAL1`.

7 EVALUACIÓN FINAL Y PROPUESTAS DE AMPLIACIONES Y MEJORAS

En el punto de la entrega del proyecto y echando la vista atrás, se puede afirmar con rotundidad que se han cumplido los principales objetivos marcados al inicio del mismo.

En cuanto a la implementación de soluciones, tanto a nivel software como hardware, se ha conseguido aplicar algunas de las tecnologías y herramientas más punteras en los campos del IoT en general, y la domótica en particular. Todas ellas perfectamente integradas y cohesionadas para conformar un escenario domótico totalmente funcional y con un manejo remoto simple e intuitivo para el usuario, gracias a la aplicación móvil asociada.

No obstante, el potencial del proyecto presentado es enorme, y se podría idear una infinidad de elementos y funcionalidades a añadir, con el fin de aumentar aún más las posibilidades del mismo. A continuación, se citarán algunas propuestas de ampliaciones y mejoras que finalmente no han podido ser incluidas en el proyecto final entregado, bien por falta de tiempo y/o de recursos:

Lo más inmediato y primero que se viene a la mente en este sentido, es la adición de un mayor número de elementos domóticos, tales como un mayor número de bombillas inteligentes, sensores de distinto tipo, control de persianas, alarma de seguridad, etc, sobre todo con el objetivo de cubrir y domotizar el resto de las estancias de la casa, no solo el salón como ocurre actualmente.

Por otro lado, durante la fase de desarrollo en la que aún se encuentra el sistema, se ha estado utilizando la aplicación de pruebas asociada a Expo y React Native, cuyo uso está limitado para dispositivos situados en la misma subred donde se ubica el sistema. Uno de los objetivos iniciales era el de poder hacer uso de la aplicación sin tener que estar dentro del domicilio y poder controlar los elementos domotizados desde cualquier lugar. Sería necesario pues, la publicación del servidor en internet, y de la aplicación móvil en las tiendas de aplicaciones de Android y/o iOS.

En el caso de que además se quisiera hacer extensible el sistema desarrollado a usuarios de todo el mundo, sería imprescindible dotarlo de la capacidad de personalización. Actualmente el sistema es estático. Todos los elementos domóticos a introducir han de ser previamente integrados en la red ZigBee, en el algoritmo del nodo coordinador, en el servidor y, por último, en el código de la aplicación móvil, mediante un botón dedicado. La solución en este sentido, pasaría en primer lugar por añadir una pantalla de personalización en la aplicación móvil que permita al usuario la inclusión de nuevos elementos domóticos, así como de pantallas de estancias que mejor se adapten a su domicilio. Dicha pantalla de personalización debe manejar el método HTTP POST, que deberá de ser soportado también por el servidor. Por último, se deberá dotar también al nodo coordinador de la capacidad de escanear e incorporar bajo demanda los dispositivos ZigBee de la zona, aparte de la creación dinámica de tramas ZigBee.

La introducción de las mejoras descritas en este apartado, sin duda llevarían al sistema domótico desarrollado a un nivel superior, permitiendo por tanto su uso y disfrute por parte de cualquier usuario, en un mundo futuro cada vez más familiarizado con este tipo de tecnologías.

8 MANUAL DE USUARIO

En este apartado, se citarán los sencillos pasos a seguir para poner en marcha el escenario implementado, replicando así la funcionalidad mostrada en el vídeo de prueba mostrado en la presentación del proyecto.

Dichos pasos son:

- Conectar el adaptador de enchufe a una toma de corriente. Se puede enchufar a su vez en ella cualquier aparato no domotizado como un cargador de móvil o una lámpara de escritorio para comprobar mejor su funcionamiento.
- Enroscar la bombilla inteligente en un casquillo compatible y encender su interruptor de corriente asociado.
- Encender el Waspmote que incorpora el ventilador inteligente.
- Alimentar el nodo coordinador con el cable USB de Arduino y conectarlo al router de la casa mediante un cable Ethernet. Importante cargar previamente el programa desde la IDE de Arduino y asegurarse de proporcionar en el mismo la dirección IP donde se ejecutará el servicio de *backend*. Esperar a los cuatro parpadeos rápidos del LED, que señalizan la correcta puesta en marcha del nodo coordinador.
- Ejecutar el servicio de *backend* desde el PC.
- Ejecutar el proyecto de React Native asociado a la aplicación móvil. Asegurarse de proporcionar las direcciones IP correctas, tanto del PC donde se ejecuta el servicio de *backend* como del servidor del arduino, en los archivos client.js y clientArduino.js, respectivamente. Escanear el código QR que aparece usando un dispositivo móvil, lo que llevará a la aplicación de Expo cuya descarga está disponible en las tiendas de aplicaciones de Android y iOS.
- Probar el funcionamiento de los dispositivos domóticos mediante la pantalla “salón”.

Adicionalmente, es posible acceder al servicio de *backend* desde un navegador web para comprobar cómo va cambiando el valor de cada parámetro a medida que lo ordenemos mediante la aplicación. Igualmente, se podrá acceder al servidor del arduino mediante un navegador web para comprobar la disponibilidad del mismo.

REFERENCIAS

[1] D. Evans, «Cisco Internet Business Solutions Group (IBSG)Cisco IBSG © 2011Cisco aThe Internet of Things: How the Next Evolution of the Internet Is Changing Everything,» 2011.

[2] S. C. Mukhopadhyay, Internet Of Things, vol. 2, 2014.

[3] P. R. d. l. Santos, «Breve historia de Internet de las cosas (IoT),» Telefónica Tech, [En línea]. Available: https://empresas.blogthinkbig.com/breve-historia-de-internet-de-las-cosas-iot/.

[4] S.-H. Yang, Wireless Sensor Networks, 2014.

[5] R. M. R. Tituana, «Domótica. Estado del Arte.».

[6] P. Dhillon y D. H. Sadawarti, «A Review Paper on Zigbee (IEEE 802.15.4),» *International Journal of Engineering Research & Technology (IJERT)*, vol. 3, 2014.

[7] M. Ramya, S. Madasamy y R. Prabakaran, «STUDY ON ZIGBEE TECHNOLOGY,» *Conference: Electronics Computer Technology (ICECT)*, vol. 6, 2011.

[8] C. A. Vera Romero, J. E. Barbosa Jaimes y D. C. Pabón González, «Estudio de las características de la capa física de la tecnología ZigBee.,» *Scientia et Technica*, vol. 22, nº 3, 2017.

[9] Arduino, [En línea]. Available: https://www.arduino.cc/.

[10] «ESP8266,» Wikipedia, [En línea]. Available: https://es.wikipedia.org/wiki/ESP8266.

[11] E. R. d. Luis, «De cero a maker: todo lo necesario para empezar con Raspberry Pi,» Xakata, [En línea]. Available: https://www.xataka.com/makers/cero-maker-todo-necesario-para-empezar-raspberry-pi.

[12] G. M. Muñoz, Red de sensores Waspmote. Aplicación real para la mejora de las centrales solares térmicas., Sevilla, 2016.

[13] «¿Qué es XBee?,» Digi, [En línea]. Available: https://xbee.cl/que-es-xbee/.

[14] A. R. Cebrián, «¿Qué es React Native? La forma de desarrollar apps está cambiando,» cuatroochenta, [En línea]. Available: https://cuatroochenta.com/que-es-react-native-el-modo-de-desarrollar-apps-esta-cambiando/.

[15] J. A. Blanes, «¿Qué es React Native?,» Deloitte, [En línea]. Available: https://www2.deloitte.com/es/es/pages/technology/articles/que-es-react-native.html.

[16] F. Ontiveros, «¿Por qué desarrollar apps de React Native en Expo?,» Medium, [En línea]. Available: https://medium.com/leopark-lab/por-qu%C3%A9-desarrollar-apps-de-react-native-en-expo-2e1b83e4d00a.

[17] M. Hamedani, «The Complete Node.js Course,» Code With Mosh, [En línea]. Available:

https://codewithmosh.com/p/the-complete-node-js-course.

[18] E. Ribas, «Qué es Api Rest y por qué debes de integrarla en tu negocio,» IEBS, [En línea]. Available: https://www.iebschool.com/blog/que-es-api-rest-integrar-negocio-business-tech/.

[19] «¿Qué es una API de REST?,» Red Hat, [En línea]. Available: https://www.redhat.com/es/topics/api/what-is-a-rest-api.

[20] J. Crespo, «ICSP,» Aprendiendo Arduino, [En línea]. Available: https://aprendiendoarduino.wordpress.com/2016/11/06/icsp/.

[21] «Zigbee Home Automation,» DIGI, [En línea]. Available: https://www.digi.com/support/knowledge-base/zigbee-home-automation?q=Home+Automation&l=en_US&fs=Search&pn=1.

[22] «Explicit Addressing Command Request,» DIGI, [En línea]. Available: https://www.digi.com/resources/documentation/Digidocs/90002002/Reference/r_frame_0x11.htm.

[23] «ZigBee Cluster Library Specification,» ZigBee Alliance, [En línea]. Available: https://zigbeealliance.org/wp-content/uploads/2019/12/07-5123-06-zigbee-cluster-library-specification.pdf.

[24] J. Richards, «ratmandu/node-red-contrib-zblight,» github, [En línea]. Available: https://github.com/ratmandu/node-red-contrib-zblight.

[25] «ZigBee Home Automation Hacking,» [En línea]. Available: https://wor.io/zigbee/.

[26] L. Llamas, «Cómo usar ficheros Json en Arduino con Arduino Json,» [En línea]. Available: https://www.luisllamas.es/arduino-json/.

[27] «Ethernet library,» Arduino, [En línea]. Available: https://www.arduino.cc/en/reference/ethernet.

[28] L. d. V. Hernández, «Cómo utilizar el sensor DHT11 para medir la temperatura y humedad con Arduino,» Programarfacil.com, [En línea]. Available: https://programarfacil.com/blog/arduino-blog/sensor-dht11-temperatura-humedad-arduino/.

[29] M. Hamedani, «The Ultimate React Native Series,» Code With Mosh, [En línea]. Available: https://codewithmosh.com/p/the-ultimate-react-native-course.