

TRABAJO FIN DE GRADO

DISEÑO Y DESARROLLO DE UN SISTEMA DE GESTIÓN DOMÓTICA MULTI-UBICACIÓN

TRABAJO FIN DE GRADO PARA LA
OBTENCIÓN DEL TÍTULO DE
GRADUADO EN INGENIERÍA EN
TECNOLOGÍAS INDUSTRIALES

SEPTIEMBRE 2024

Alejandro Amor Mourelle

DIRECTORES DEL TRABAJO FIN DE GRADO:

Ángel García-Beltrán
Ascensión López Vargas

I. AGRADECIMIENTOS

Quiero comenzar con mi más sincero agradecimiento a mis tutores, por la confianza que tuvieron en mí al permitirme desarrollar este proyecto, a partir de una motivación personal. Sin su ayuda, este proyecto no habría sido posible y les estaré eternamente agradecido.

A mi familia, especialmente a mi hermana, que tantas noches ha pasado estudiando conmigo. Por animarme a apuntar siempre alto sin miedo y no desesperarse aun cuando parecía que se torcía un poco el camino. Sin ellos, no sería la persona que soy ahora.

Finalmente, quiero darle las gracias a Teresa. Gracias por ser siempre mi refugio en tiempos difíciles y llenar mi vida de felicidad. Sin ti, nada de esto tendría sentido.

II. RESUMEN

La industria 4.0 lleva años en constante desarrollo, siendo uno de sus componentes fundamentales el “*Internet of Things*” o IoT. El desarrollo de tecnologías IoT como las redes de comunicación 5G, la computación en la nube o los avances en cuanto a protocolos de comunicación, no solo han servido como base de la innovación en la industria, sino que también han tenido un gran impacto en otros ámbitos como, por ejemplo, la domótica, abriendo en ellos un gran abanico de nuevas posibilidades de innovación.

Los sistemas domóticos profesionales ofrecen actualmente un servicio básico, a un coste elevado a través de cuotas mensuales. Por ello se han ido desarrollando distintos sistemas domóticos de libre uso, que permiten al usuario final crear su propio sistema con un coste muy inferior.

Algunos de estos nuevos sistemas de libre uso se han implementado utilizando tecnologías de comunicación inalámbricas, mediante protocolos de comunicación como Zigbee o Z-Wave. Otros han implementado servicios de almacenamiento de datos en la nube, permitiendo la gestión de estos sistemas desde cualquier lugar. Sin embargo, es difícil encontrar sistemas que hayan implementado las mejores prácticas de las distintas tecnologías del IoT en su conjunto.

En este contexto, surge la idea de desarrollar este proyecto, a partir de la motivación personal del autor, por crear su propio sistema domótico. El objetivo de este proyecto es, por tanto, la creación de un sistema domótico, con prestaciones técnicas similares a uno profesional, manteniendo el enfoque de minimizar el coste para el usuario final e implementando todas las ventajas de los desarrollos tecnológicos en el área de la domótica y la informática.

De esta forma, el objetivo principal del proyecto se puede concretar en:

Creación de un sistema de gestión domótica, multi-ubicación, con capacidad remota de gestión, intuitivo, personalizable, optimizado en el coste por usuario, con alta capacidad de escalar, de uso masivo y de arquitectura hardware fácilmente migrable, si el crecimiento lo requiriera.

Este objetivo incluye la creación de un sistema extremo a extremo, de uso masivo, que permita a cada usuario instalar, gestionar y personalizar sistemas de gestión domótica en múltiples ubicaciones, desde una única aplicación final, de manera remota. Todo ello, manteniendo la búsqueda de un coste mínimo por usuario.

Para alcanzar este objetivo, en primer lugar, se analizaron los protocolos de comunicación existentes relacionados con los sensores domóticos. De entre aquellos válidos, se ha seleccionado Zigbee por su razonable alcance, arquitectura de redes malladas, su bajo consumo y la gran variedad de dispositivos disponibles.

Una vez decidido el protocolo de comunicación, se realiza un diseño conceptual del sistema sobre el que realizar el posterior análisis y desarrollo.

En el diseño a alto nivel, se identifican un conjunto de ubicaciones. En cada ubicación del sistema se considera que existen múltiples sensores, así como un “**Homehub**” o **consola de control** del sistema, la cual se comunica con el gestor domótico (que es el núcleo del proyecto) y éste con el usuario final a través de una interfaz web HTML.

Una vez conocido el esquema básico del sistema, se realizan las fases de análisis y diseño. En ellas se definen los requisitos funcionales y no funcionales, y la labor de cada uno de los

componentes del sistema. A continuación, se desarrollan con más detalle aquellos componentes más relevantes.

Empezando por la descripción de las piezas más sencillas, los sensores se comunican con el *Homehub* a través del protocolo de comunicación Zigbee. Para evitar la complejidad de gestionar cada sensor a través de dicho protocolo, se implementa en el *Homehub* la librería de Python “Zigbee2MQTT”. Esta librería traduce los mensajes de los sensores al protocolo MQTT, el cual es ideal para la comunicación entre el Homehub y el gestor domótico, ya que permite la comunicación asíncrona mediante un sistema de colas de mensajería.

Por tanto, la labor del *Homehub* es gestionar los sensores de su ubicación, recibir y enviar mensajes en protocolo Zigbee del lado de los sensores, recibir y enviar mensajes en protocolo MQTT del lado del gestor domótico y ser capaz de traducir de un protocolo al otro.

Para analizar las funciones que debe realizar el **gestor domótico**, se realiza un diseño de bajo nivel de cada uno de sus componentes. De esta forma, se identifican qué funciones debe realizar cada uno y cómo se gestionan las conexiones entre ellos.

En este diseño de bajo nivel, el gestor domótico está compuesto por 4 servicios. El primero de ellos, el **servicio de mensajería**, se encarga de permitir la comunicación con los *Homehubs*. Por ello, su función es implementar la parte servidor del protocolo estándar MQTT. De esta forma se convierte en un “broker” o intermediario de mensajería, que permite la comunicación a través de sus canales, a los distintos clientes MQTT que se conecten a él.

El **servicio de escucha o “Listener”**, se encarga de recibir mensajes MQTT en relación con los *Homehubs*. Por ello, su función principal es la implementación de la parte cliente del protocolo MQTT.

Cada vez que el servicio de escucha recibe un mensaje, se comunica con la base de datos para guardar la información relevante y llama a una función del servidor de aplicaciones para comprobar si debe realizarse alguna acción adicional.

El **servicio de almacenamiento de datos** debe ser capaz de almacenar y gestionar todos los datos necesarios para el sistema y permitir sesiones de lectura y escritura de datos a los demás servicios del sistema.

El servicio se compone de una base de datos SQL, gestionada mediante PostgreSQL, en la que se ha implementado un modelo de datos adaptado a las necesidades del proyecto.

El **modelo de datos** está formado principalmente por las clases: usuarios, ubicaciones, sensores y eventos.

Cada usuario tiene asociados, en un primer nivel, una serie de ubicaciones. Estas ubicaciones a su vez están vinculadas, en un segundo nivel, a una serie de sensores. Por otro lado, cada usuario también tiene asociados en el primer nivel una serie de eventos.

De entre estas clases, destacan: “Sensores” y “Eventos”.

En primer lugar, “Sensores” se ha estructurado a través de una tabla de tipos de sensor y otra tabla de campos de cada tipo. Esto permite incorporar al sistema cualquier sensor, independientemente del número de campos o tipología de los mismos, sin necesidad de realizar ningún cambio en el modelo de datos.

Por su parte, el campo “Eventos”, surge de la agrupación de una serie de condiciones y acciones. La introducción de eventos al sistema permite a los usuarios definir bajo qué

circunstancias quieren que el sistema realice automáticamente alguna acción y, de esta forma, gestionar de manera automática y personalizada su conjunto de ubicaciones.

Por último, el **servidor de aplicaciones-web** es el núcleo central del gestor domótico y de este proyecto. En él se encuentra toda la lógica necesaria para su funcionamiento. Su función principal es la creación y gestión de un servidor de aplicaciones-web.

Este servicio debe ser capaz de gestionar la creación, lectura, actualización y borrado (**CRUD**) de usuarios, ubicaciones, sensores y eventos. El envío de los mensajes MQTT necesarios para la gestión de los sensores también forma parte de este servicio. Además, también es el encargado de la generación y envío de páginas HTML personalizadas para cada usuario.

Para poder realizar todas estas labores se utilizan múltiples librerías de Python, siendo las más destacadas las siguientes:

1. Unicorn: permite la creación del servidor de aplicaciones.
2. FastAPI: *framework* que permite un fácil desarrollo, optimizado para suministrar un alto rendimiento.
3. SQLAlchemy: permite la fácil gestión de los modelos y conexiones a la base de datos.
4. Paho: permite la creación de un cliente MQTT para el envío de mensajes.
5. Jinja2: permite la personalización de los archivos HTML.
6. HTMX: permite mejorar la experiencia del usuario al interactuar con la página web.

El uso de todas estas herramientas permite cumplir con los distintos objetivos de este proyecto.

Para terminar con el análisis y diseño del sistema, debe destacarse el uso del *software Docker*. Este *software* permite desarrollar el sistema a través de “dockers” o “**contenedores**”, entendiéndose por contenedor, un “paquete” de *software* con todo lo necesario para ejecutar un servicio de manera independiente.

Diseñando el sistema a través de contenedores, se garantiza que éste pueda migrarse de una arquitectura *hardware* a otra, y en caso de ser necesario escalar la solución, se puedan replicar los servicios, simplemente aumentando el número de contenedores.

Para llevar a cabo el desarrollo del proyecto, han sido necesarias 1.423 líneas de Python y 1.995 líneas de código HTML. Su desarrollo ha permitido alcanzar los objetivos del proyecto de manera satisfactoria.

Una vez terminado el proyecto con éxito, se plantean numerosas mejoras posibles, dentro de estas, destacan la implementación de sistemas de seguridad, la ampliación del sistema a otros protocolos de comunicación y la creación de una aplicación móvil que permita una fácil gestión del sistema desde cualquier lugar.

III. PALABRAS CLAVE Y CÓDIGOS UNESCO

Palabras clave: Domótica, IoT, Sensores, Zigbee, MQTT, Bases de datos, HTML, HTMX, FastAPI, Contenedores, Docker, SQLAlchemy, Paho, Zigbee2MQTT, Sistema Domótico.

Códigos UNESCO: 120312, 120314, 120317, 120323, 120325, 330417, 330413, 332505

ÍNDICE

1. INTRODUCCIÓN	3
2. OBJETIVOS	5
3. ESTADO DEL ARTE	7
3.1. Principales protocolos de comunicación existentes en el mercado domótico	7
3.2. Comparativa entre los diferentes protocolos de comunicación	9
3.3. Antecedentes	9
3.4. Selección de un determinado protocolo de comunicación	12
4. ANÁLISIS DE REQUISITOS	13
4.1. Requisitos funcionales	13
4.1.1. Consola de control (<i>Homehub</i>)	14
4.1.2. Servicio de mensajería (MQTT)	15
4.1.3. Servicio de escucha (Listener)	16
4.1.4. RF Servicio de almacenamiento de datos (PostgreSQL)	17
4.1.5. Servidor de aplicaciones-web (FastAPI)	18
4.2. Requisitos no funcionales	21
4.2.1. Sistema global	21
4.2.2. Servicio de Respuesta HTML	21
4.2.3. RNF Servicio de almacenamiento de datos (PostgreSQL)	22
5. DISEÑO Y DESARROLLO	23
5.1. Diseño del sistema global	23
5.2. Diseño y desarrollo del <i>Homehub</i>	23
5.2.1. Diseño hardware	23
5.2.1.1. Consola de control	23
5.2.1.2. Adaptador Zigbee y antena receptora	24
5.2.2. Diseño <i>software</i>	24
5.3. Diseño y desarrollo del gestor domótico	25
5.3.1. Diseño hardware	25
5.3.2. Diseño <i>software</i>	25
5.3.2.1. Servicio de mensajería (MQTT)	25
5.3.2.2. Servicio de escucha (Listener)	26
5.3.2.3. Diseño del Servicio de almacenamiento de datos (PostgreSQL)	27
1. Modelo de datos	27
2. Desarrollo del contenedor del Servicio de Almacenamiento de Datos "SAD"	29
5.3.2.4. Servidor de aplicaciones-web (FastAPI)	30
1. Desarrollo del contenedor	30
2. Librerías necesarias	31
3. Desarrollo del programa principal	32
6. RESULTADOS	35
6.1. Sistema domótico obtenido	35
6.2. Resumen de los resultados obtenidos	36
6.3. Funciones desarrolladas	37
6.4. Interfaz web de usuario	43
6.5. Valoración de impactos y aspectos de responsabilidad social, legal, ética y profesional	58
7. VALIDACIÓN	59
8. CONCLUSIONES Y LÍNEAS FUTURAS	61
8.1. Conclusiones	61
8.2. Líneas futuras	62
1. Funcionalidades fuera de alcance	62
2. Implantación de sistemas de ciberseguridad	62

3.	Desarrollo de nuevas funcionalidades	63
4.	Creación de una app móvil	63
5.	Extensión del sistema a otros protocolos	63
6.	Realización de análisis de carga y gestión de la escalabilidad	63
9.	BIBLIOGRAFÍA	65
10.	PLANIFICACIÓN TEMPORAL	67
11.	PRESUPUESTO	71
11.1.	Presupuesto del material	71
11.2.	Presupuesto del coste personal.....	72
12.	EVALUACIÓN DE IMPACTOS	73
13.	ANÁLISIS DE ASPECTOS LEGALES Y ÉTICOS	75
14.	CONTRIBUCIÓN A LOS OBJETIVOS DE DESARROLLO SOSTENIBLE.....	77
16.	ÍNDICE DE FIGURAS	79
17.	ÍNDICE DE TABLAS.....	81
18.	ABREVIATURAS Y ACRÓNIMOS.....	83
19.	GLOSARIO	85

1. INTRODUCCIÓN

El mundo de la tecnología, y en especial el mundo del “*Internet of Things*” o IoT, lleva años en constante desarrollo. Se han desarrollado nuevos protocolos de comunicación, mejorado la variedad y fiabilidad de todo tipo de sensores, creado gemelos digitales a escala de fábricas reales, introducido las ventajas de la industria 4.0... Esto ha generado un contexto de cambio constante, con mucho espacio para la innovación en otros ámbitos, a través de la aplicación de dichas nuevas tecnologías.

Dentro de dicho contexto, surge este proyecto a raíz de la motivación personal por utilizar estas nuevas tecnologías, aplicadas a un entorno más cercano, una vivienda. A pesar de que esta idea ya ha sido desarrollada ampliamente, en este caso se pretende dar un enfoque distinto a las soluciones finales que existen en el mercado, se busca crear un sistema, que permita al usuario final personalizar todo lo posible el uso de los dispositivos disponibles, a través de una única plataforma y controlando múltiples ubicaciones de forma simultánea.

En un principio, esto puede parecer algo sencillo, pero debido al uso de distintos sistemas de comunicación por parte de cada empresa comercializadora, la interconexión de múltiples dispositivos a través de un único sistema o aplicación no siempre ha sido una opción viable. Por ello, los usuarios finales terminaban siendo limitados por el fabricante escogido inicialmente, o debían utilizar múltiples sistemas, cada uno con sus propios aplicativos para gestionar unos pocos sensores, aumentando enormemente la complejidad de su uso y su coste.

Este proyecto busca crear un entorno que permita al usuario final, utilizar al menor coste posible cualquier dispositivo comercial compatible, y a su vez, diseñar él mismo las normas lógicas para su uso. Buscándose que este entorno sea de fácil uso para cualquier persona, independientemente de su experiencia tecnológica.

A lo largo de esta memoria se explicarán las distintas tecnologías existentes que permitirán crear el sistema deseado y los distintos criterios o factores claves, que implicarán la elección concreta de cada tecnología. Se entrará más en detalle en los distintos objetivos mencionados previamente y se incluirán criterios básicos como el coste y la disponibilidad.

Por último, este proyecto no se diseñará para un usuario único. Una vez creado el sistema para un único usuario, la generalización a su uso masivo parece un paso lógico a la vez que no conlleva un trabajo mucho mayor, por ello se optó por desarrollar todo este proyecto con la intención de ser un sistema multiusuario.

2. OBJETIVOS

A lo largo de esta sección se describirán más en detalle los objetivos comentados previamente y se incluirán varios más para definir claramente lo que se busca alcanzar con el desarrollo de este proyecto.

El principal objetivo del proyecto es:

Creación de un sistema de gestión domótica, multi-ubicación, con capacidad remota de gestión, intuitivo, personalizable, optimizado en el coste por usuario, con alta capacidad de escalar, de uso masivo y de arquitectura hardware fácilmente migrable, si el crecimiento lo requiriera.

Como el propio nombre del proyecto indica, el principal objetivo es la creación de un sistema extremo a extremo, de uso masivo, que permita a cada usuario: instalar, gestionar y personalizar sistemas de gestión domótica en múltiples ubicaciones, desde una única aplicación final de manera remota. Y que a la vez tenga un coste mínimo por usuario cuando se utilice masivamente.

Para poder lograr este objetivo, se deben cumplir una serie de objetivos de segundo orden:

1. **Conseguir que el sistema sea capaz de gestionar múltiples ubicaciones por usuario:** se añade como criterio de diseño que el sistema debe permitir a un usuario gestionar distintas ubicaciones a partir de una única interfaz de manera transparente, es decir, sin necesidad de cambiar de una interfaz a otra en función de la ubicación deseada.
2. **Permitir un alto grado de personalización del sistema:** se pretende crear un sistema que permita al usuario final personalizar lo máximo posible el uso de los sensores y las posibles acciones que surjan a partir de sus valores.
3. **Crear una interfaz de usuario de fácil uso:** la aplicación de usuario final debe ser de fácil manejo tanto para un usuario experimentado como para una persona sin experiencia en el manejo de sensores o informática.
4. **Conseguir un sistema de fácil escalabilidad y migración:** se diseñará el sistema pensando en su implantación en sistemas de procesamiento a través de la “nube”. Permitiendo de esta forma evitar la necesidad de un servidor físico y la fácil migración a arquitecturas con mayor potencia. Además, se diseñará el sistema mediante módulos, “contenedores” o “*containers*”, que sean fácilmente replicables y escalables en función de la demanda prevista.
5. **Crear un sistema de fácil interconexión:** este proyecto busca desarrollar un sistema que permita la interconexión del mayor número de dispositivos compatibles entre sí, independientemente del fabricante de este. Existirán límites de compatibilidad en función del protocolo de comunicación que utilice cada dispositivo, pero se buscará evitar cualquier otra limitación impuesta por los fabricantes o desarrolladores de sistemas de uso.

Al mismo tiempo, se buscará cumplir con algunos de los criterios básicos de cualquier proyecto:

6. **Buscar el mínimo coste posible por usuario:** el sistema se diseñará centralizando los procesos en la medida de lo posible y se diseñará utilizando *hardware* con la potencia mínima necesaria que permita ejecutar los programas desarrollados, abaratando así el coste de los equipos. También se utilizará *software* de libre uso que evite el coste de la compra de licencias y ligue el proyecto a un desarrollador concreto.

7. **Conseguir alta disponibilidad y fiabilidad:** los paquetes *software* y protocolos escogidos deberán garantizar una gran fiabilidad y tener disponibles una variedad suficiente de sensores y sistemas compatibles para cubrir las necesidades del usuario final promedio. De esta forma, se evitarán aquellos desarrollos más novedosos que no hayan sido ampliamente testados y de los cuales aún no haya gran cantidad de sensores comercializados.

3. ESTADO DEL ARTE

En este apartado se va a analizar el estado del arte de los dispositivos y tecnologías relacionados con el proyecto. Concretamente, se va a realizar una comparativa de los distintos protocolos de comunicación presentes en la domótica y el mundo IoT. De esta forma se podrán ver sus ventajas e inconvenientes a la hora de elegir el protocolo adecuado para este proyecto.

También se realiza un análisis de otros proyectos similares desarrollados en los últimos años, para entender sus puntos fuertes y débiles, aprendiendo de ellos para mejorar el desarrollo de este proyecto.

3.1. Principales protocolos de comunicación existentes en el mercado domótico

Los principales protocolos de comunicación utilizados en la industria domótica son los siguientes [1]:

1. **Wi-Fi HaLow (802.11):** Este protocolo se basa en el estándar IEEE 802.11 de comunicación inalámbrica. Una de sus ventajas, es su amplia utilización y su compatibilidad con ordenadores, móviles...

Para garantizar la seguridad durante las comunicaciones, implementa distintos protocolos de seguridad: WEP, WPA y WPA2. Por su gran popularidad, se ha desarrollado durante años. Actualmente, existe la versión Wi-Fi HaLow, que mejora el consumo energético y permite su implantación en sistemas IoT.

Permite tasas de transmisión de datos de entre 11 y 40 kbps. Su rango máximo oscila entre 20 metros para entornos cerrados y 240 para entornos al aire libre. A pesar de que la versión Wi-Fi más moderna utiliza la banda de los 5 GHz, Wi-Fi HaLow se mantiene en la banda de los 2.4 GHz, permitiendo así un consumo menor y una mejor penetración de los obstáculos.

2. **Bluetooth Low Energy (BLE):** Este protocolo se basa en el estándar IEEE 802.15.1. Desde la creación del Bluetooth, el protocolo se ha desarrollado continuamente hasta crear el estándar BLE o Bluetooth LE, enfocado a minimizar el consumo de energía para aplicaciones distintas a la concebida inicialmente para este protocolo.

BLE opera en la banda de los 2.4 GHz. Posee 79 canales, cada uno de ellos de 1 MHz. La comunicación está protegida a través del uso del protocolo estándar de comunicación AES-CCM y del uso de certificados digitales (CSRK).

A pesar de permitir inicialmente solo redes de conexión en estrella, el desarrollo del Bluetooth 5.0 introdujo las redes malladas, permitiendo su aplicación en ámbitos de IoT y domótica.

3. **Zigbee:** Este protocolo se basa en el estándar IEEE 802.15.4, diseñado para el uso en redes PAN (Personal Area Networks), y por ello orientado a dispositivos personales. Utiliza la banda de 868 MHz en Europa, con un ancho de banda máximo de 20 kbps, la de 915 MHz en los Estados Unidos, con un ancho de banda máximo de 40 kbps y la de 2.4 GHz en el resto del mundo, con un ancho de banda máximo de 250 kbps. Su rango máximo es de alrededor de 100 metros y destaca por su bajo consumo.

Para garantizar la seguridad durante las comunicaciones, utiliza el estándar de encriptación AES-128. Permite la creación de amplias redes con un único coordinador de la misma, ya que los sensores pueden pasar mensajes de uno a otro hasta alcanzar al coordinador. De esta forma, el rango efectivo de la red puede extenderse.

4. **Z-Wave:** Este protocolo fue desarrollado por una empresa privada en 1999, la cual posee la patente. Este protocolo utiliza tecnología de radiofrecuencia para conectar sensores y actuadores. Para garantizar la seguridad durante la comunicación, utiliza el estándar de encriptación AES-128. Funciona con un controlador maestro que gestione a través de órdenes a los dispositivos esclavos.

5. **LoRaWAN:** Este protocolo ha sido desarrollado por “The LoRa Alliance” usando como base la tecnología LoRa. Destaca por su amplio rango, de hasta 1 km en zonas urbanas y su muy bajo consumo de energía para comunicación de baja carga.

Utiliza el estándar de encriptación AES-128 para garantizar la seguridad de las comunicaciones.

Este protocolo se usa ampliamente en la industria agrícola, en las ciudades inteligentes... por su gran alcance y la larga duración de la batería de los dispositivos por su bajo consumo.

6. **6LoWPAN:** Este protocolo fue desarrollado por una empresa privada y se basa en el estándar IEEE 802.15.4. Al igual que Zigbee, opera en las bandas de: 2.4 GHz en gran parte del mundo, 913 MHz en Estados Unidos y 868 MHz en Europa.

Su rango máximo es de aproximadamente 100 metros y la máxima tasa de transmisión de datos posible es de 250 kbps. El protocolo ha sido optimizado para requerir la menor cantidad de bytes posible y funciona mediante una topología de red mallada.

Para garantizar la seguridad durante la comunicación, implementa el estándar de encriptación AES-128.

Además de los protocolos mencionados previamente, existen muchos otros que no se han recogido. Esto se debe a que su ámbito de aplicación no está relacionado con el proyecto o a que su desarrollo es aún incipiente y no apto para un sistema que busque garantizar la fiabilidad del sistema.

Una vez conocidos los principales protocolos de comunicación del ámbito de la domótica, se debe escoger uno de ellos para desarrollar el sistema. La mayor parte de los sistemas comerciales de gestión domótica evitan al usuario final la necesidad de escoger entre unos protocolos de comunicación u otros. El usuario final se preocupa únicamente de la compatibilidad de los sensores con la consola de control del fabricante.

Como contrapartida, el usuario final se debe vincular de forma exclusiva a una determinada consola de control, lo que condiciona y limita las capacidades de interacción entre el usuario y los sensores controlados por dicha consola.

Debido a que este proyecto cubre, extremo a extremo, un “gestor domótico multi-ubicación”, sí se debe analizar el estado del arte de los diferentes protocolos de comunicación y seleccionarse (al menos) uno de ellos. Claramente, esta decisión condicionará el universo de sensores disponibles para la propia aplicación y supondrá una serie de limitaciones en cuanto a alcance, precio, etc.

3.2. Comparativa entre los diferentes protocolos de comunicación

En la Tabla 1 se hace una comparativa entre los distintos protocolos de comunicación próximos al mundo de la domótica y del IoT, describiendo sus principales características a tener en cuenta para poder elegir el protocolo adecuado.

Tabla 1: Comparativa de los protocolos de comunicación [1]

Características	Bluetooth LE	Z-Wave	ZigBee	LoRa	6LoWPAN	Wi-Fi
Estándar	IEEE 802.15.1	ITU G-9959	IEEE 802.15.4	IEEE 802.15.4g	IEEE 802.15.4	IEEE 802.11
Red	WPAN	WPAN	WPAN	WAN	WPAN	WLAN
Topología	Estrella, malla	Malla	Estrella, malla, árbol	Malla	Estrella, Malla	Malla
Potencia	Baja	Baja	Baja	Ultra-baja	Baja	Media
Bandas de Frecuencia	2.4 GHz	868 MHz–908 MHz	2.4 GHz	869/915 MHz	868 MHz (UE), 915 MHz (EE. UU.), 2.4 GHz (Global)	2.4/5/6 GHz
Tasa de Datos	1–2 Mbps	40 kbps	250 kbps	50 kbps	250 kbps	11–9600 Mbps
Rango	15–30 m	30 m (interiores), 100 m (exteriores)	10–100 m	Urbano (2–5 km) suburbano (15 km)	10–100 m	100 m
Dispersión	FHSS	-	DSSS	CSS	DSSS	DSSS
Seguridad	E0 stream, AES-128	AES-128	AES-128	AES-128	AES-128	WPA2/3
Aplicaciones Comunes	Aplicaciones de audio y auriculares inalámbricos	Monitoreo y control del hogar	Control y monitoreo industrial	Monitoreo de la contaminación del aire, procesamiento agrícola...	Monitoreo y control a través de internet	Móviles, empresas, hogar...

3.3. Antecedentes

En este apartado se van a comentar los resultados de otros sistemas domóticos desarrollados previamente. Se comentarán sus principales objetivos, cómo fue llevado a cabo el desarrollo y cuáles fueron sus resultados.

De esta forma se pretende ver qué enfoques han tenido un buen resultado y pueden servir como base del desarrollo de este sistema y qué ventajas o ideas pueden ofrecer.

En primer lugar, se van a estudiar dos proyectos de domótica más antiguos, que fueron la base de desarrollos futuros. Ambos tienen como uno de sus objetivos principales la

disminución del coste de un sistema domótico, que en la época en la que se publicaron era un problema aún mayor al que es actualmente. De esta forma, pueden ayudar al proyecto a buscar formas de disminuir el coste por usuario final (criterio de diseño de nuestro sistema).

El primero de ellos: **“Diseño de un sistema de control domótico basado en la plataforma Arduino “** [2]. Este proyecto buscaba crear un sistema de bajo coste, a través del uso de microprocesadores, sustituyendo a los ordenadores personales, ya que estos infrutilizaban gran cantidad de su potencia.

El autor plantea el uso de placas Arduino para llevar a cabo el sistema, y utiliza una serie de sensores conectados a través de cables o directamente a los puertos de la placa Arduino. Además, también estudia las distintas posibles arquitecturas de comunicación mediante múltiples microprocesadores y un servidor central.

Este proyecto permite ver cómo los desarrollos domóticos de bajo coste llevan estudiándose muchos años y cómo los microprocesadores antiguos eran, ya en su época, suficientemente potentes como para poder llevar a cabo un proyecto de características similares. Sin embargo, no aprovecha el desarrollo incipiente de la tecnología inalámbrica con el protocolo Zigbee y los sensores desarrollados para dicho protocolo.

Por otro lado, está el proyecto: **“Análisis y diseño de un prototipo de sistema domótico de bajo costo”** [3]. Este proyecto también tenía como objetivo principal, permitir el uso de sistemas domóticos a un precio razonable.

Para ello, realiza un análisis de los protocolos de comunicación del mercado y concluye que, en el momento de la publicación del artículo, el mejor protocolo existente era Zigbee. Debido a su bajo coste en cuanto a sensores y coordinadores, su bajo consumo de energía y su amplio catálogo de dispositivos compatibles.

Además, para minimizar el coste total del sistema, también opta por un microprocesador. En este caso, el microprocesador “Microchip 16F688”, este dispositivo cumple con los requisitos del proyecto y tiene un bajo consumo de energía.

Nuevamente, este proyecto opta por utilizar un microprocesador con las características mínimas necesarias. Minimizando al máximo el coste del hardware utilizado. Además, utiliza las tecnologías inalámbricas, en este caso Zigbee, de forma que no hay necesidad de cableado y la instalación del sistema es más cómoda para el usuario.

Ambos proyectos fueron pioneros de la domótica de bajo coste en su época, y comparten con el proyecto buscado, el afán por minimizar dicho coste del sistema domótico para el usuario final. Como punto débil, las interfaces de usuario desarrolladas son muy primitivas y no permiten un uso cómodo por parte del usuario. Además, el grado de personalización sigue siendo muy bajo, permitiendo pocas opciones para el usuario final.

En cuanto a los trabajos más recientes revisados, destacan dos de ellos por la aplicación de nuevas tecnologías como la IA o las bases de datos en la nube.

El primero de ellos es: **“DomoHome: un sistema domótico inteligente”** [4]. Este proyecto desarrolla un sistema domótico similar a los descritos previamente, sin embargo, el sistema se conecta mediante un servidor MQTT a una base de datos en la nube. Esto permite la gestión del sistema desde cualquier lugar.

Por ello, el autor desarrolla también una aplicación móvil sencilla que permita conectarse desde cualquier móvil con el sistema para gestionarlo.

Este sistema tiene como punto fuerte el uso de las nuevas tecnologías de red para permitir la gestión del sistema desde cualquier lugar. Sin embargo, el sistema conecta los sensores por cable al microprocesador principal, lo cual supone una gran desventaja a la hora de instalar el sistema y disminuye la comodidad del usuario final al incorporar nuevos sensores al sistema.

El segundo de ellos es: **“SISTEMA DOMÓTICO A ESCALA CON ACCESO REMOTO”** [5]. Este proyecto consiste en la creación de una vivienda inteligente sostenible. Destaca por su aplicación de las tecnologías más novedosas como el *“Machine Learning”* y por la búsqueda de un alto grado de sostenibilidad.

El proyecto utiliza como núcleo del sistema una Raspberry Pi 4, debido a que sus especificaciones parecen ser las mejores para la creación de una vivienda inteligente. Junto con dicho microprocesador, utiliza múltiples sensores conectados directamente por cable a la Raspberry Pi. Destaca la implementación de una cámara capaz de retransmitir imagen en vivo a la aplicación final.

El proyecto incluye también el desarrollo de una aplicación web para gestionar los dispositivos a distancia. Utiliza una base de datos MongoDB, no SQL, para la gestión de los datos del usuario y de los sensores.

Por último, este proyecto aplica tecnologías emergentes como el *“Machine Learning”* para la identificación biométrica del usuario.

Respecto a la búsqueda de un diseño más sostenible, el sistema busca utilizar energía solar como forma de alimentación de los distintos dispositivos.

De nuevo, el sistema diseñado tiene como punto débil el uso de conexiones por cable entre el microprocesador central y los distintos dispositivos, lo que supone un inconveniente para el usuario final.

El estudio de todos estos proyectos permite sacar una serie de conclusiones:

1. El uso de microprocesador como núcleo central de los sistemas domóticos es una práctica ya habitual que permite minimizar el coste del hardware necesario y su consumo.
2. El uso de las nuevas tecnologías de red es necesario para dar al usuario final la mejor experiencia posible, permitiéndole gestionar el sistema domótico desde cualquier lugar.
3. El uso de una base de datos en la nube permite a la domótica actual crear sistemas más sofisticados y personalizables para el usuario.
4. Los sistemas hasta ahora estudiados están enfocados únicamente a un usuario, lo que no permite optimizar todo lo posible la capacidad del sistema.
5. Los sistemas actuales no están diseñados para unificar en una única interfaz de usuario múltiples ubicaciones de un usuario.
6. No todos los sistemas utilizan conexiones inalámbricas para los sensores, siendo su utilización un criterio de diseño de cada sistema.
7. Los sistemas estudiados no están desarrollados para su uso masivo por parte de múltiples usuarios finales.
8. Los sistemas estudiados no están diseñados para poder migrar a otros *hardwares* futuros que ofrezcan mejores prestaciones.

Teniendo todo esto en cuenta, los objetivos del proyecto buscan cumplir una serie de necesidades de los usuarios finales que no parecen haber sido ampliamente exploradas.

3.4. Selección de un determinado protocolo de comunicación

Una vez conocidas las principales características de los protocolos de comunicación, y los antecedentes del proyecto, se va a seleccionar el protocolo de comunicación.

A la hora de seleccionar un determinado protocolo de comunicaciones para los sensores gestionados por nuestra aplicación, se debe tener en consideración el cumplimiento de los siguientes factores:

1. Amplia disponibilidad en el mercado de los sensores que lo utilizan, tanto en tipología como en cantidad como en fabricantes.
2. Coste económico, asequibilidad de los sensores a utilizar,
3. Minimización de la “potencia consumida”, debido a que gran parte de los sensores domóticos utilizan baterías.
4. Protocolo orientado a temas domóticos (es decir, que no sea un protocolo genérico orientado a temas IoT industriales)
 1. Ancho de banda adecuado a soluciones domóticas
 2. Alcance, distancia máxima a la que se pueden comunicar los sensores, adecuada a soluciones domóticas
 3. Minimización de interferencias en ambientes domóticos
 4. Velocidad de transmisión de datos adecuada a ambientes domóticos
 5. Capacidad para crear redes de sensores razonablemente amplias para ambientes domóticos.

Siguiendo los anteriores criterios, se pueden descartar una serie de protocolos de comunicación:

Wi-Fi: ha sido **descartado** por su alto consumo de energía. La batería de este tipo de sensores se agota rápidamente, por lo que esta solución no es aplicable en ubicaciones que no están supervisadas durante meses.

Bluetooth: ha sido **descartado** por problemas de alcance, su alcance en entornos domóticos con gran cantidad de obstáculos puede suponer un problema. Además, su implantación en el mercado domótico es menor que la de otros protocolos orientados a dicho mercado.

LoRaWAN: ha sido **descartado** por estar más orientado a entornos de IoT industriales que a entornos domóticos. La disponibilidad de sensores domóticos es reducida. Si se utiliza el entorno estándar TTN, se depende de pasarelas (“gateways”) de terceros, lo que haría que este proyecto no sea 100% autocontenido.

6LoWPAN: ha sido **descartado** por su menor alcance que otras alternativas, también se ha tenido en cuenta su menor implantación actual respecto a otros protocolos.

Por lo tanto, los dos protocolos que, atendiendo a los criterios previamente establecidos, serían los más adecuados para el desarrollo de este proyecto son: Z-Wave y Zigbee

Finalmente, el protocolo seleccionado es ZigBee, debido a la existencia de una ventaja adicional (que se detalla más adelante), la cual simplifica la interacción con los sensores. Esta ventaja es la existencia de un *software* de libre uso denominado “Zigbee2MQTT” que enmascara o apantalla toda la interacción con los sensores ZigBee, permitiendo que esta comunicación se realice utilizando mensajes MQTT.

4. ANÁLISIS DE REQUISITOS

En este capítulo se va a desarrollar el análisis de requisitos funcionales y no funcionales. Este análisis permite definir claramente las características y funcionamiento deseado del sistema a desarrollar.

Estructurar de manera adecuada los requisitos del sistema puede contribuir a cumplir con las necesidades y expectativas de los usuarios finales.

Para facilitar la comprensión de estos, se presenta en la Figura 1 un diagrama de contexto de muy alto nivel. De esta forma, se pretende lograr una mejor visualización del sistema buscado en su totalidad.

La Figura 1, ilustra cómo un sistema completo para un usuario final, está formado por un cúmulo de ubicaciones remotas, con sus sensores correspondientes, las cuales se comunican con el gestor domótico y le permiten al usuario final gestionar todo su sistema a través de una sencilla interfaz gráfica.

Posteriormente, se presentarán diagramas de bajo nivel para cada componente del sistema, desglosando los distintos requisitos funcionales y no funcionales que debe cumplir cada componente.

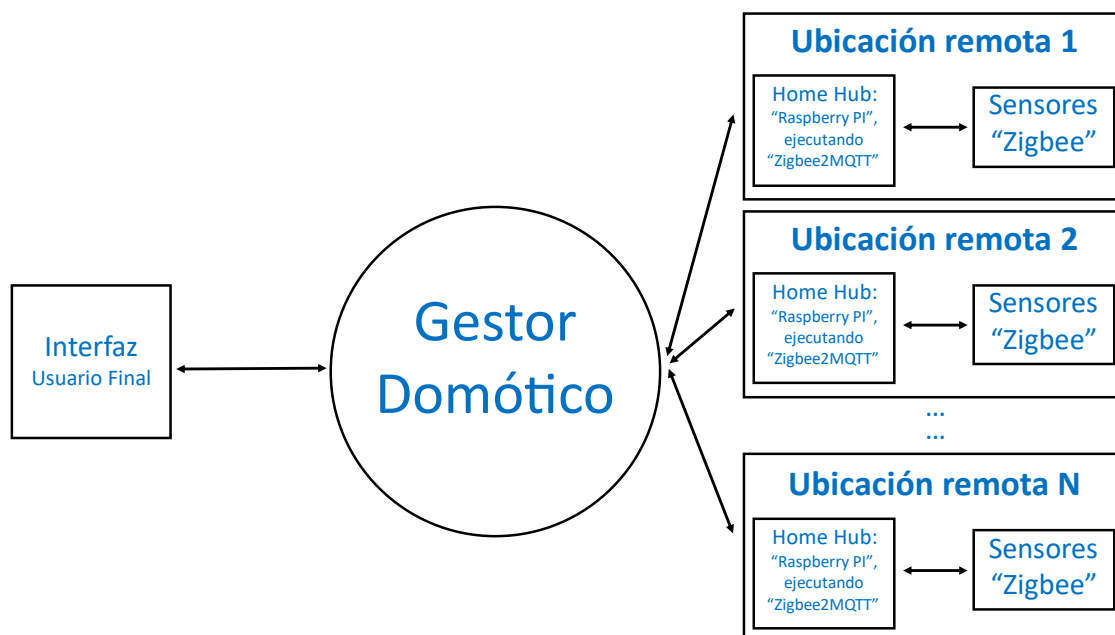


Figura 1: Diagrama de contexto de alto nivel

4.1. Requisitos funcionales

A continuación, se va a analizar cada componente por separado.

4.1.1. Consola de control (*Homehub*)

La Figura 2 desarrolla en profundidad el concepto de ubicación remota. Se empieza explicando el concepto de ubicación remota por ser la pieza del sistema más básica.

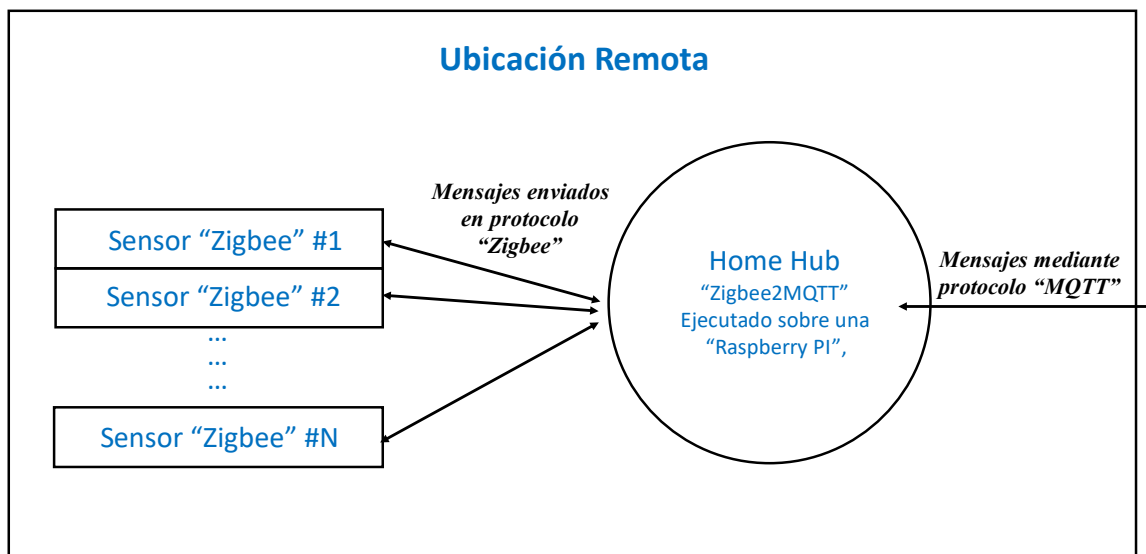


Figura 2: Diagrama de bajo nivel de una ubicación remota

La labor del *Homehub* consiste en la transmisión de los datos de cada sensor de una determinada ubicación al gestor domótico. El *Homehub* está formado por un microordenador, en este caso una Raspberry PI, y una antena receptora con el *software* “Zigbee2MQTT” [6].

Como se observa en la Figura 2, para poder realizar su labor, debe ser capaz de recibir los mensajes escritos en protocolo “Zigbee” [7] de los distintos sensores y traducirlos al protocolo “MQTT” [8], protocolo utilizado por el servicio de mensajería.

Para realizar esta labor, los requisitos funcionales que debe cumplir el *Homehub* de una ubicación remota son los expuestos en la Tabla 2:

Tabla 2: Requisitos funcionales de un *Homehub* (HH)

ID	Descripción del requisito
RF-HH-1	El “HH” debe ser capaz de incorporar y registrar nuevos sensores en cualquier momento.
RF-HH-2	El “HH” debe ser capaz de establecer comunicación con aquellos sensores registrados.
RF-HH-3	En caso de perder la conexión con algún dispositivo, el “HH” debe ser capaz de restablecerla, si fuera técnicamente posible.
RF-HH-4	El “HH” debe ser capaz de “traducir” la información recibida mediante el protocolo de comunicación “Zigbee” al protocolo de mensajes “MQTT”.
RF-HH-5	El “HH” debe ser capaz de crear un cliente “MQTT”.
RF-HH-6	El “HH” debe ser capaz de establecer una conexión cliente-servidor con el “MQTT” server.
RF-HH-7	El “HH” debe ser capaz de subscribirse a los “topics MQTT” correspondientes a su ubicación.
RF-HH-8	El “HH” debe ser capaz de realizar el envío de mensajes “MQTT” al servidor.
RF-HH-9	En caso de perder la conexión con el servidor “MQTT” el “HH” debe ser capaz de restablecerla si fuera técnicamente posible.

La mayor parte de estos requisitos funcionales los cumple el *software* “Zigbee2MQTT”, por ello, el trabajo a realizar en relación con los *Homehubs* en este proyecto será principalmente de instalación y configuración de distintos paquetes *software*.

El siguiente componente es el gestor domótico en sí, ilustrado con la Figura 3. Éste, a su vez, está compuesto por cuatro grandes bloques funcionales. Cada uno de estos bloques posee diferentes funciones y, por tanto, distintos requisitos.

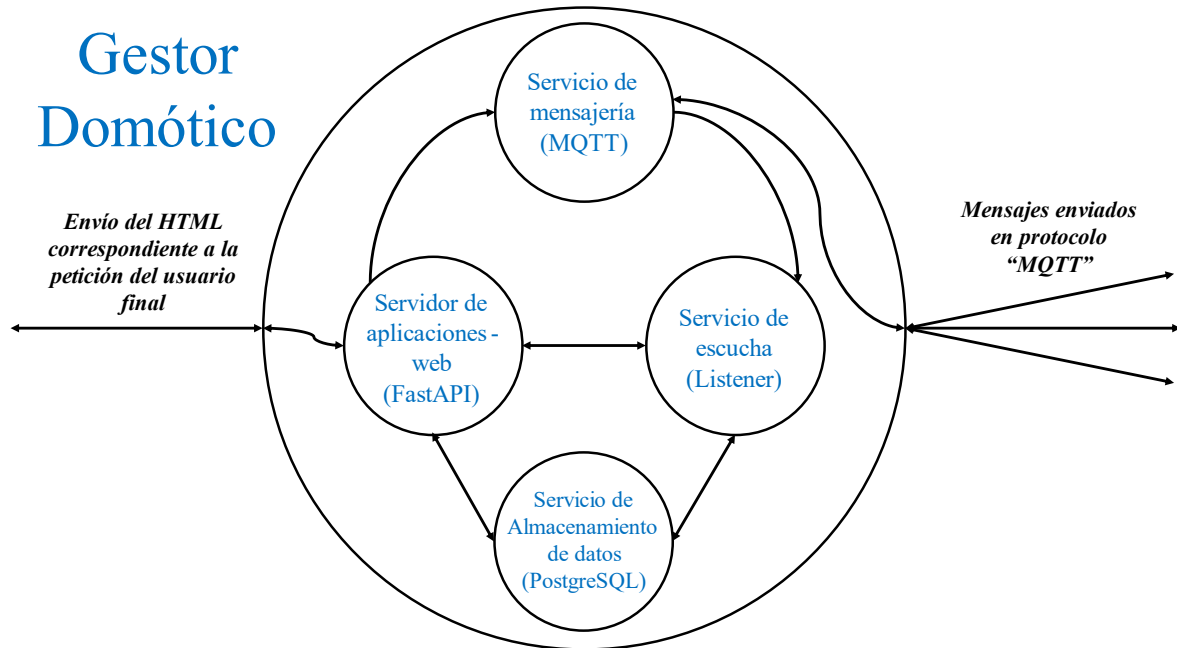


Figura 3: Diagrama de bajo nivel del gestor domótico

En la Figura 3, se pueden observar los principales componentes del gestor domótico, y los canales de comunicación existentes entre ellos. A continuación, se explica cada uno de estos componentes, su labor y los requisitos funcionales que debe cumplir.

4.1.2. Servicio de mensajería (MQTT)

El servicio de mensajería será un intermediario o “*broker*” de mensajería que implemente el protocolo estándar MQTT. En este proyecto se opta por la implementación del “*broker* MQTT Mosquitto” [9] por motivos que se explicarán más adelante.

Como se observa en la Figura 3, el servicio de mensajería debe ser capaz de transmitir los mensajes de entrada y salida relacionados con los distintos *Homehubs*, debe permitir la recepción de los mensajes al servicio de escucha y, por último, debe ser capaz de transmitir los mensajes de entrada del servidor de aplicaciones. Por ello, además de los requisitos funcionales definidos por el propio estándar MQTT [8], el servicio de mensajería debe cumplir con los requisitos funcionales expuestos en la Tabla 3:

Tabla 3: Requisitos funcionales del Servicio de Mensajería (SM)

ID	Descripción del requisito
RF-MQTT-1	El "SM" debe ser capaz de registrar nuevos <i>Homehubs</i> (crear nuevos canales).
RF-MQTT-2	El "SM" debe ser capaz de establecer comunicación con cualquier <i>Homehub</i> registrado.
RF-MQTT-3	En caso de perder la conexión con algún <i>Homehub</i> , el "SM" debe intentar restablecer la conexión si fuera técnicamente posible.
RF-MQTT-4	El "SM" debe ser capaz de registrar nuevos sensores (crear nuevos subcanales).
RF-MQTT-5	El "SM" debe ser capaz de crear y gestionar gran cantidad de canales de comunicación para cada <i>Homehub</i> y para cada sensor asociado a este.
RF-MQTT-6	En caso de fallo del sistema, el "SM" debe ser capaz de rearmarse y volver al funcionamiento normal de manera autónoma.
RF-MQTT-7	El "SM" debe ser capaz de recibir gran cantidad de mensajes por parte del servidor de aplicaciones con una tasa de error mínima.
RF-MQTT-8	El "SM" debe permitir la recepción de los mensajes por parte del Servicio de escucha.

Nuevamente, la mayor parte de estos requisitos funcionales los cumple la implementación "Mosquitto" del intermediario MQTT. De esta forma, la labor realizada durante el proyecto en relación con el servicio de mensajería será principalmente de instalación y configuración de este paquete *software*.

4.1.3. Servicio de escucha (Listener)

El servicio de escucha o "*Listener*" será el encargado de recibir los mensajes enviados desde los sensores remotos de cada ubicación al servicio de mensajería y transmitirlos al servicio de almacenamiento de datos o al servidor de aplicaciones-web.

Por ello, el servicio de escucha consta de un cliente MQTT capaz de comunicarse con el servicio de mensajería y múltiples funciones que le permitan realizar peticiones de lectura y escritura a la base de datos, o ejecutar rutinas del servidor de aplicaciones.

Los requisitos funcionales del servicio de escucha se exponen en la Tabla 4:

Tabla 4: Requisitos funcionales del Servicio de Escucha (SE)

ID	Descripción del requisito
RF-SE-1	El "SE" debe ser capaz de implementar un cliente MQTT.
RF-SE-2	El "SE" debe ser capaz de conectarse al servicio de mensajería.
RF-SE-3	El "SE" debe ser capaz de subscribirse a todos los "topics" asociados a cada usuario, <i>Homehub</i> y sensor.
RF-SE-4	El "SE" debe ser capaz de establecer una conexión con la base de datos.
RF-SE-5	El "SE" debe ser capaz de realizar peticiones de lectura y escritura a la base de datos.
RF-SE-6	El "SE" debe ser capaz de guardar en la base de datos la información de los mensajes recibidos, estructurada según el modelo de datos.
RF-SE-7	En caso de fallo, el "SE" debe ser capaz de rearmarse y volver al funcionamiento habitual de manera autónoma.
RF-SE-8	El "SE" debe ser capaz de llamar a funciones del servidor de aplicaciones-web con la llegada de nuevos mensajes.

Los requisitos funcionales relacionados con la implementación de un cliente MQTT se gestionarán mediante la librería “*Paho*” [10] de Python [11], como se explicará más adelante. De esta forma, este componente necesitará configuración del cliente MQTT y desarrollar las funciones que lo comunican con la base de datos y con el servidor de aplicaciones-web.

4.1.4. RF Servicio de almacenamiento de datos (PostgreSQL)

El servicio de almacenamiento de datos es uno de los pilares fundamentales de este proyecto. En él se almacenan todos los datos relativos a usuarios, ubicaciones, sensores y eventos. Este servicio será llevado a cabo por una base de datos relacional, gestionada mediante el sistema de gestión PostgreSQL [12] por motivos explicados posteriormente. Esta base de datos debe ser capaz de gestionar gran cantidad de solicitudes de escritura y lectura por parte del servidor de aplicaciones-web y el servicio de escucha.

Uno de los requisitos más importantes de las bases de datos es la ciberseguridad de ésta, sin embargo, los requisitos relacionados con la seguridad del gestor domótico quedan fuera del alcance de este TFG para un futuro desarrollo. Teniendo esto en cuenta, los requisitos funcionales que debe cumplir el servicio de almacenamiento de datos quedan recogidos en la Tabla 5:

Tabla 5: Requisitos funcionales del Servicio de Almacenamiento de Datos (SAD)

ID	Descripción del requisito
RF-SAD-1	El “SAD” debe permitir el registro de nuevos usuarios, <i>Homehubs</i> , sensores y eventos.
RF-SAD-2	El “SAD” debe garantizar la integridad de los datos.
RF-SAD-3	El “SAD” debe tener una serie de permisos y roles que garanticen que solo usuarios autorizados tienen acceso a los datos.
RF-SAD-4	El “SAD” debe tener una alta disponibilidad, al menos suficiente para no ser un cuello de botella del sistema.
RF-SAD-5	En caso de fallo del sistema, el “SAD” debe ser capaz de rearmarse y volver al funcionamiento habitual de manera autónoma.
RF-SAD-6	El “SAD” debe poseer un modelo de datos adaptado a las necesidades del sistema. (Se desarrollará en capítulos posteriores).
RF-SAD-7	El “SAD” debe permitir la creación de múltiples sesiones concurrentes de consulta.
RF-SAD-8	El “SAD” buscará optimizar, en la medida de lo posible, el tiempo de lectura y escritura.

El cumplimiento de los requisitos funcionales relacionados con la alta disponibilidad vendrá marcado por la potencia del *hardware* sobre el que se ejecute este servicio. En este proyecto, se diseña el sistema pensando en su ejecución sobre una máquina virtual que pueda adaptarse a las necesidades del sistema, dando por garantizado este requisito.

En cuanto a los requisitos de integridad y gestión de los datos, el sistema de gestión de bases de datos PostgreSQL será el encargado de llevarlos a cabo.

Por último, el desarrollo del modelo de datos óptimo para este sistema se verá en capítulos posteriores.

4.1.5. Servidor de aplicaciones-web (FastAPI)

El desarrollo del servidor de aplicaciones-web será la parte principal de este TFG. En este componente del sistema, se programará toda la lógica asociada a usuarios, ubicaciones, sensores y eventos.

Mientras que en otros muchos componentes de este sistema la labor es principalmente de instalación y configuración, en este caso, se desarrollará desde cero la lógica necesaria para cumplir con los requisitos funcionales de este componente. Para entender más fácilmente el funcionamiento del servidor de aplicaciones, se presenta la Figura 4 con los distintos componentes:

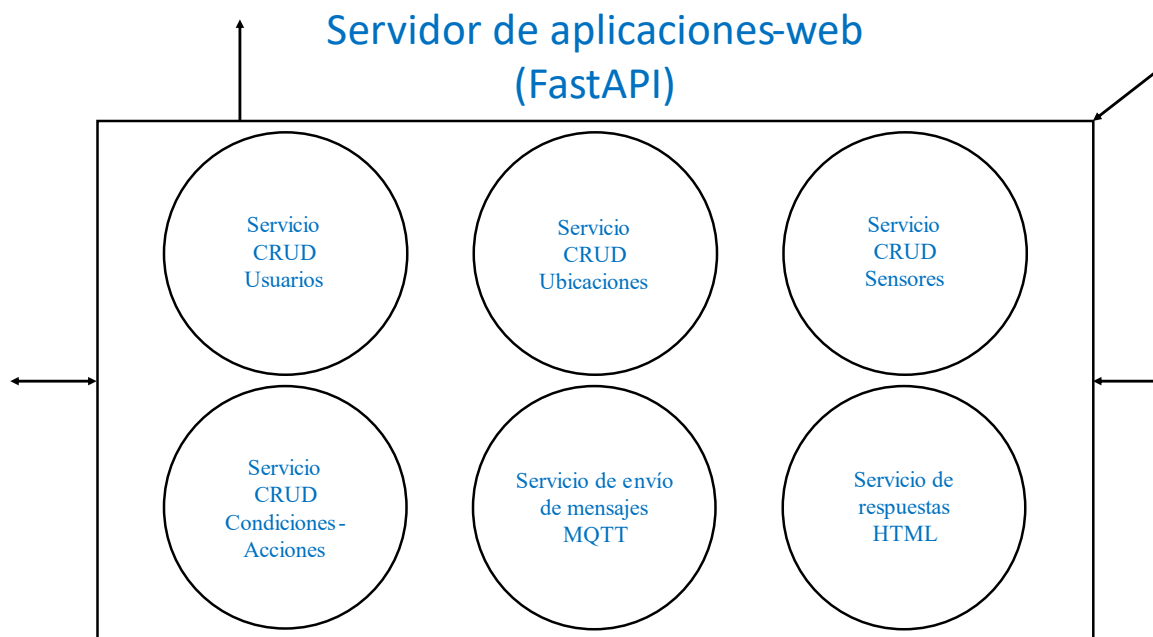


Figura 4: Diagrama de bajo nivel del Servidor de aplicaciones-web

Como se puede observar en la Figura 4, el servidor de aplicaciones-web está formado por 6 componentes, cada uno de ellos encargado de una serie de funciones. Los 3 componentes superiores y el inferior izquierdo hacen referencia a los “CRUDs” de usuarios, ubicaciones, sensores y eventos o condiciones-acciones.

El acrónimo “CRUD” (Create, Read, Update, Delete) hace referencia a las labores de creación, lectura, actualización y borrado. Estos 4 componentes se encargarán de dichas labores asociadas a usuarios, ubicaciones, sensores y eventos respectivamente. Adicionalmente, se engloba dentro del “CRUD” de usuarios las funciones necesarias para el acceso de usuarios a su página web personal.

El servicio de envío de mensajes MQTT será el encargado de implementar un cliente MQTT y conectarse al intermediario de mensajería para enviar las órdenes de los usuarios finales a los *Homehubs* y sensores correspondientes.

Por último, el servicio de respuestas HTML [13] será el encargado de crear y servir los distintos códigos HTML personalizados para cada usuario. Este servicio es el encargado de crear la interfaz de usuario y que esta sea intuitiva y cómoda.

Aunque en la Figura 1, se presente un bloque referente a la interfaz de usuario final, este es meramente visual, ya que las funciones de creación y gestión de la interfaz de usuario son

llevadas a cabo exclusivamente por el servidor de aplicaciones-web a través del componente de servicio de respuestas HTML.

A continuación, se presentan las tablas: Tabla 6, Tabla 7, Tabla 8, Tabla 9, Tabla 10, Tabla 11, recogiendo los requisitos funcionales de cada componente del servidor de aplicaciones-web:

Tabla 6: Requisitos funcionales del Servicio "CRUD" de Usuarios (SCUS)

ID	Descripción del requisito
RF-SAW-CR-US-1	El "SCUS" debe ser capaz de conectarse con el servicio de almacenamiento de datos.
RF-SAW-CR-US-2	El "SCUS" debe ser capaz de recibir peticiones HTML de la interfaz de usuario.
RF-SAW-CR-US-3	El "SCUS" debe permitir la creación de nuevos usuarios.
RF-SAW-CR-US-4	El "SCUS" debe permitir la lectura de datos relacionados con los usuarios.
RF-SAW-CR-US-5	El "SCUS" debe permitir la actualización de los datos de los usuarios.
RF-SAW-CR-US-6	El "SCUS" debe permitir el borrado de los datos asociados a un usuario.
RF-SAW-CR-US-7	El "SCUS" debe garantizar que solo un usuario con las credenciales correctas puede acceder a su página personal.
RF-SAW-CR-US-8	El "SCUS" debe ser capaz de crear un listado de usuarios en caso de que el administrador lo requiera.

Tabla 7: Requisitos funcionales del Servicio "CRUD" de Ubicaciones (SCUB)

ID	Descripción del requisito
RF-SAW-CR-UB-1	El "SCUB" debe ser capaz de conectarse con el servicio de almacenamiento de datos.
RF-SAW-CR-UB-2	El "SCUB" debe ser capaz de recibir peticiones HTML de la interfaz de usuario.
RF-SAW-CR-UB-3	El "SCUB" debe permitir la creación de nuevas ubicaciones asociadas a un usuario.
RF-SAW-CR-UB-4	El "SCUB" debe permitir la lectura de datos relacionados con las ubicaciones.
RF-SAW-CR-UB-5	El "SCUB" debe permitir la actualización de los datos de las ubicaciones.
RF-SAW-CR-UB-6	El "SCUB" debe permitir el borrado de los datos asociados a una ubicación.
RF-SAW-CR-UB-7	El "SCUB" debe ser capaz de crear un listado de ubicaciones en caso de que el administrador lo requiera.

Tabla 8: Requisitos funcionales del Servicio "CRUD" de Sensores (SCS)

ID	Descripción del requisito
RF-SAW-CR-S-1	El "SCS" debe ser capaz de conectarse con el servicio de almacenamiento de datos.
RF-SAW-CR-S-2	El "SCS" debe ser capaz de recibir peticiones HTML de la interfaz de usuario.
RF-SAW-CR-S-3	El "SCS" debe permitir la creación de nuevos sensores asociados a una ubicación.
RF-SAW-CR-S-4	El "SCS" debe permitir la lectura de datos relacionados con los sensores.

ID	Descripción del requisito
RF-SAW-CR-S-5	El "SCS" debe permitir la actualización de los datos de los sensores.
RF-SAW-CR-S-6	El "SCS" debe permitir el borrado de los datos asociados a un sensor.
RF-SAW-CR-S-7	El "SCS" debe ser capaz de crear un listado de sensores en caso de que el administrador lo requiera.

Tabla 9: Requisitos funcionales del Servicio "CRUD" de Condiciones-Acciones (SCCA)

ID	Descripción del requisito
RF-SAW-CR-CA-1	El "SCCA" debe ser capaz de conectarse con el servicio de almacenamiento de datos.
RF-SAW-CR-CA-2	El "SCCA" debe ser capaz de recibir peticiones HTML de la interfaz de usuario.
RF-SAW-CR-CA-2	El "SCCA" debe permitir la creación de nuevas condiciones-acciones asociadas a un usuario.
RF-SAW-CR-CA-3	El "SCCA" debe permitir la lectura de datos relacionados con las condiciones-acciones.
RF-SAW-CR-CA-4	El "SCCA" debe permitir la actualización de los datos de las condiciones-acciones.
RF-SAW-CR-CA-5	El "SCCA" debe permitir el borrado de los datos asociados a una condición-acción.
RF-SAW-CR-CA-6	El "SCCA" debe ser capaz de crear un listado de condiciones-acciones en caso de que el administrador lo requiera.

Tabla 10: Requisitos funcionales del Servicio de Envío de Mensajes MQTT (SEM) del servidor de aplicaciones

ID	Descripción del requisito
RF-SAW-MQTT-1	El "SEM" debe ser capaz de conectarse con el servicio de almacenamiento de datos.
RF-SAW-MQTT-2	El "SEM" debe ser capaz de recibir peticiones HTML de la interfaz de usuario.
RF-SAW-MQTT-3	El "SEM" debe ser capaz de implementar un cliente MQTT
RF-SAW-MQTT-4	El "SEM" debe ser capaz de establecer conexión con el servicio de mensajería.
RF-SAW-MQTT-5	El "SEM" debe ser capaz de enviar mensajes MQTT a la ubicación y sensor adecuados.
RF-SAW-MQTT-6	El "SEM" debe ser capaz de recibir y ejecutar peticiones de envío de mensajes de los otros servicios CRUD.

Tabla 11: Requisitos funcionales del Servicio de Respuestas HTML (SR)

ID	Descripción del requisito
RF-SAW-SR-1	El "SR" debe ser capaz de conectarse con el servicio de almacenamiento de datos.
RF-SAW-SR-2	El "SR" debe ser capaz de recibir peticiones HTML de la interfaz de usuario.
RF-SAW-SR-3	El "SR" debe ser capaz de servir páginas HTML a los usuarios.
RF-SAW-SR-4	El "SR" debe ser capaz de personalizar las páginas HTML servidas en función de los datos del usuario.
RF-SAW-SR-5	El "SR" debe crear páginas HTML que eviten la introducción de datos erróneos en la medida de lo posible.

4.2. Requisitos no funcionales

Además de los requisitos funcionales, también se van a analizar los requisitos no funcionales existentes en el sistema, orientados a garantizar que este realiza las funciones necesarias de una forma concreta.

4.2.1. Sistema global

Para cumplir con los objetivos expuestos previamente, el sistema debe ser diseñado con ellos en mente.

Uno de los principales objetivos de este proyecto es minimizar el coste del sistema por usuario, asumiendo que el sistema alcanzase un nivel de uso masivo en régimen permanente.

Por ello, el sistema se diseña buscando centralizar, siempre que sea posible, los procesos realizados en un único servidor central, el “gestor domótico” presentado en la Figura 1. Mediante la centralización de los procesos del sistema, se “aligera” la carga de trabajo de cada ubicación remota, permitiendo que la potencia de procesamiento instalada en ellas sea menor y de esta forma abaratando el coste del *hardware* necesario en ellas.

Otro de los objetivos del proyecto es crear un sistema con gran escalabilidad, es decir, en caso de que su uso aumente de forma rápida, el sistema debe ser capaz de adaptarse y replicarse para dar servicio a la creciente demanda.

En consecuencia, el sistema será diseñado mediante módulos independientes. Esta forma de diseño se asemeja a la actual corriente de diseño de los sistemas masivos, estructurados en microservicios fácilmente replicables.

Además, se debe tener en cuenta que el sistema debe ser intuitivo y de fácil uso para cualquier persona, y permitir un alto grado de personalización. Teniendo todo esto en cuenta, los requisitos no funcionales del sistema global se presentan en la Tabla 12:

Tabla 12: Requisitos no funcionales del sistema global

ID	Descripción del requisito
RNF-SG-1	El sistema debe minimizar el coste por usuario, asumiendo el uso masivo del mismo.
RNF-SG-2	El sistema debe estar diseñado mediante módulos independientes que puedan ser replicables cuando, por el crecimiento de los usuarios, se requiera escalar la solución en potencia de cálculo.
RNF-SG-3	El sistema debe ser intuitivo y de fácil uso.
RNF-SG-4	El sistema debe permitir, en la medida de lo posible, la personalización de este por parte del usuario final.

4.2.2. Servicio de Respuesta HTML

Además de los requisitos funcionales, el servicio de respuestas de HTML debe cumplir con una serie de requisitos no funcionales. Esto se debe a que es el servicio encargado de interactuar con el usuario final y, por tanto, no son solo importantes las funciones que desarrolla, sino también como las lleva a cabo.

En la Tabla 13 se presentan los requisitos no funcionales del servicio de respuestas HTML:

Tabla 13: Requisitos no funcionales del Servicio de Respuesta HTML (SR)

ID	Descripción del requisito
RNF-SAW-SR-1	Las páginas creadas por el “SR” deben ser visualmente agradables para los usuarios finales.
RNF-SAW-SR-2	Las páginas creadas por el “SR” deben ser sencillas e intuitivas. El usuario debe saber utilizar el sistema sin necesidad de una explicación avanzada.
RNF-SAW-SR-3	El diseño de las páginas debe ser consistente en todo el sistema.
RNF-SAW-SR-4	El “SR” debe ser capaz de crear páginas HTML que se actualicen parcialmente según el usuario interactúe con ellas. Evitando la necesidad de actualizar la página completa.

4.2.3. RNF Servicio de almacenamiento de datos (PostgreSQL)

Durante la gestión del servicio de almacenamiento de datos, la incorporación de nuevos sensores puede suponer un desafío. Esto se debe a la compleja variedad de sensores existentes en el mercado y a su distinta tipología en cuanto a campos y variables medidas.

Por ello, para facilitar la gestión de este servicio, uno de sus requisitos no funcionales debe ser la creación de un modelo de datos que permita incorporar cualquier sensor, independientemente de su tipo, de manera sencilla y ordenada.

Esto implica que el modelo de datos deberá constar de múltiples tablas relacionadas con los sensores que permitan relaciones 1 a N. De esta forma se podrán incorporar nuevos sensores sin necesidad de modificar ni crear nuevas tablas en el modelo, simplemente creando instancias del modelo definido inicialmente.

Por otro lado, el diseño del servicio de almacenamiento de datos puede condicionar el grado de personalización del sistema. Para cumplir con el objetivo de máxima personalización del sistema, el modelo de datos debe incorporar relaciones (relaciones SQL) entre las distintas tablas del modelo que permitan generar lógicas de gestión a medida para cada usuario.

Para cumplir con estos objetivos, el servicio de almacenamiento de datos debe satisfacer los requisitos no funcionales de la Tabla 14:

Tabla 14: Requisitos no funcionales del Servicio de Almacenamiento de Datos (SAD)

ID	Descripción del requisito
RNF-SAD-1	El modelo de datos del “SAD” deberá estar diseñado para que el trabajo de incorporación de nuevos tipos de sensores al sistema sea mínimo y evite la modificación del modelo de datos existente.
RNF-SAD-2	El modelo de datos del “SAD” deberá incorporar relaciones SQL entre las distintas tablas del modelo, permitiendo la creación de lógicas complejas para maximizar la personalización del sistema por parte del usuario.

5. DISEÑO Y DESARROLLO

A lo largo de este capítulo se explicará el diseño y desarrollo del sistema y de cada una de sus partes. Éste se lleva a cabo de manera que permita cumplir con todos los requisitos mencionados en el capítulo anterior.

5.1. Diseño del sistema global

En primer lugar, se debe diseñar el marco sobre el que construir cada una de las piezas del sistema. Uno de los objetivos del sistema, es conseguir que tenga una fácil escalabilidad (cuando sea sometido a un uso masivo) y gran capacidad de migración de un entorno *hardware* de ejecución a otro (por ejemplo, de nube pública como AWS a otra como Microsoft Azure). Este objetivo queda plasmado en el requisito “RNF-SG-2”.

El sistema debe diseñarse mediante pequeños módulos o “**contenedores**” [14] que sean replicables en función de la carga del sistema. Para cumplir con este objetivo, se utiliza el *software Docker* [14].

Docker es una plataforma de libre uso, que permite al usuario ejecutar programas y aplicaciones en forma de “paquetes” de manera aislada. Estos “paquetes” son lo que se conoce como contenedores Docker o simplemente “dockers”.

Un contenedor debe incorporar todo lo necesario para poder ejecutar la aplicación deseada (librerías, archivos de datos...). De esta forma, el usuario final no necesita nada más que instalar la plataforma Docker y conocer el contenedor que desea ejecutar (su imagen).

Mediante la utilización de Docker a lo largo del trabajo se programarán las distintas partes del sistema en contenedores. Esto permitirá no solo crear un sistema independiente de la arquitectura *hardware* escogida, sino también garantizar el objetivo previamente mencionado: la fácil migración y escalabilidad.

Una vez establecidos los criterios generales del diseño o bases del sistema, se procede con el diseño y desarrollo de cada parte.

5.2. Diseño y desarrollo del *Homehub*

En este apartado se explica el proceso de diseño y desarrollo de los *Homehubs*.

5.2.1. Diseño *hardware*

Se comienza explicando el desarrollo de la parte *hardware* del *Homehub*.

5.2.1.1. Consola de control

El primer paso es elegir la infraestructura *hardware* óptima para el *Homehub*.

De nuevo, para ello se deben tener en cuenta los objetivos del proyecto. Se busca un sistema con el mínimo coste económico posible por usuario, que, a la vez, permita minimizar las labores de programación a realizar sobre el *Homehub* y que así se pueda centralizar la mayor parte posible de estas labores de programación en el “Back End”.

Las labores del *Homehub* se resumen principalmente en la recepción y transmisión de unos pocos mensajes procedentes de los sensores bajo su control, por ello su carga de trabajo estimada es baja. Teniendo todo lo anterior en cuenta, se busca un *hardware* ampliamente usado en el mercado, de bajo coste, pero con cierto margen en cuanto a potencia. Además, debe garantizar amplia compatibilidad con diferentes paquetes o librerías de *software* de libre

uso que se usarán como parte del gestor domótico, y por ello debe ser capaz de ejecutar un sistema operativo abierto y estándar como Linux.

Cumpliendo con estos criterios, se selecciona la **Raspberry Pi 4 Model B**. Que utiliza su sistema operativo propio “Raspberry Pi OS”, una adaptación del Linux Debian.

Esta selección de *hardware* permite cumplir con los objetivos y criterios previamente comentados y deja cierto margen de seguridad en cuanto a potencia para el desarrollo del proyecto.

Es decir, y como resumen de este apartado, se ha seleccionado:

1. Raspberry Pi 4 Model B como plataforma *hardware* para el *Homehub*.
2. Raspberry Pi OS como sistema operativo (una variante de Linux Debian).
3. Dockers como *software* para soportar desarrollos realizados sobre contenedores.

5.2.1.2. Adaptador Zigbee y antena receptora

Por otro lado, también será necesario el uso de un adaptador o “*dongle*” que permita la recepción de los mensajes Zigbee que emiten y reciben los sensores domóticos. Este adaptador necesita una antena receptora adecuada para la recepción de mensajes en dicho protocolo.

Como se comentará más adelante, se usará el *software* “Zigbee2MQTT” el cual recomienda una serie de modelos de adaptador y antena compatibles. De las opciones posibles, se elige el componente “SONOFF ZBDongle-E” por su aparente robustez y su fácil instalación y uso. Este componente incluye el adaptador y la antena adecuados.

5.2.2. Diseño *software*

Una vez conocido el *hardware* necesario, se procede a analizar el *software* que deberá ejecutar. En primer lugar, la Raspberry Pi ejecuta el sistema operativo “**Raspberry Pi OS**” [15], el cual se actualiza junto con sus librerías a la última versión disponible.

Como se explica al principio del capítulo, sobre el sistema operativo se instala la plataforma comercial de libre uso denominada **Docker**. Sobre esta capa *software* se ejecutarán los distintos contenedores que llevarán a cabo las tareas.

El uso de Docker en el *Homehub* podría ser prescindible en una versión final del sistema, sin embargo, al utilizarlo se garantiza la posibilidad de migrar en un futuro a otro *hardware* con mejores prestaciones, sin necesidad de cambios en el programa.

Sobre la plataforma Docker, se crea un contenedor que ejecuta la imagen específica del *software* “Zigbee2MQTT”. Este *software* es una de las piezas fundamentales del proyecto. “Zigbee2MQTT” permite evitar toda la complejidad de gestionar los distintos sensores Zigbee de cada ubicación al convertir los mensajes recibidos directamente a otro protocolo estándar, MQTT. De esta manera se ahorra una enorme cantidad de trabajo en la gestión de los sensores, enfocando la mayor parte del proyecto al desarrollo del *software* del “Back End”.

El uso de este contenedor requiere la configuración necesaria para adaptarlo a las necesidades del *Homehub*.

Entre las labores necesarias están:

1. Configuración de puertos de entrada y salida.
2. Configuración IP del contenedor.

3. Configuración del registro de nuevos sensores.

Por su parte, la antena receptora necesita tener instalado el firmware para realizar el “rol de coordinador” de una red Zigbee.

Por último, para aumentar la comodidad durante las labores de desarrollo (en todo lo que tiene que ver, fundamentalmente, con las labores de administración), se instala un *software* de creación de escritorios remotos que permita utilizar la Raspberry Pi desde otro ordenador.

5.3. Diseño y desarrollo del gestor domótico

En este apartado se explica el diseño y desarrollo del gestor domótico como componente del sistema global.

5.3.1. Diseño hardware

El diseño del gestor domótico se hará teniendo en mente que debe garantizarse una fácil migración de un entorno *hardware* a otro, así como una alta escalabilidad cuando sea necesario por una hipotética utilización masiva del mismo.

Por ello, el gestor domótico se desarrolla pensando en su futura ejecución en una nube pública. De esta forma, el sistema podría adaptarse fácilmente a las fluctuaciones de demanda que pudieran existir, replicando los contenedores necesarios, es decir, “encendiendo” o “apagando” contenedores en función de la carga de trabajo (demanda de actividad) existente en cada momento.

Sin embargo, durante el desarrollo del proyecto, se simulará el uso de la nube pública a través de otra Raspberry Pi 4 Model B. De esta forma se facilita el desarrollo del sistema (trabajando en un entorno 100% controlado) sin incurrir en los costes que supondría la utilización de una nube pública.

A pesar del uso de una Raspberry Pi como gestor domótico, el sistema se diseña y programa utilizando el paradigma de contenedores, por lo que su futura implantación en un servidor en la nube no debería suponer ningún esfuerzo adicional.

5.3.2. Diseño software

El gestor domótico se divide en las 4 partes que ilustra la Figura 3. Cada una de ellas estará desarrollada en un contenedor o “Docker” independiente. En el caso de aquellos servicios estándar como es el servicio de mensajería (MQTT), la imagen Docker usada por su contenedor es pública y solo hace falta descargarla y posteriormente configurar el contenedor.

En el caso de otros contenedores como el servicio de escucha, la imagen tendrá que ser desarrollada (programada) desde cero y posteriormente se configura el contenedor en el que se ejecuta dicha imagen.

5.3.2.1. Servicio de mensajería (MQTT)

El servicio de mensajería es una implementación de la parte servidor del protocolo estándar MQTT. Existen múltiples implementaciones válidas de dicho protocolo: EMQX, nanoMQ, VerneMQ, Mosquitto...

Se ha optado por el uso de la implementación Mosquitto. Esto se debe a que esta implementación es de libre uso, tiene un consumo de recursos bajo (permite su uso en microordenadores como la Raspberry PI) y es de fácil uso mediante su imagen oficial Docker.

Una vez decidida la implementación de MQTT, en este caso Mosquitto [9], se procede a crear su contenedor utilizando la imagen Docker “eclipse-mosquitto:2.0”. Además, deben configurarse los siguientes parámetros:

1. Versión de Docker a utilizar por el contenedor.
2. Reinicio del servicio, de forma autónoma, en caso de fallo técnico, hasta restablecerse.
3. Puertos de entrada y salida del contenedor.
4. Volúmenes de datos asociados al contenedor.
5. Configuración de los mensajes o “logs” que permiten ver errores durante el desarrollo.

5.3.2.2. Servicio de escucha (Listener)

El servicio de escucha, también llamado “*Listener*”, es una implementación de la parte cliente del protocolo estándar MQTT.

Este módulo también se desarrolla en un contenedor independiente, sin embargo, no existe para él una imagen estándar, al ser un programa adaptado al proyecto. Por ello, en primer lugar, se crea la imagen del contenedor. Para ello, se utiliza como imagen base la versión del Docker de Python 3.11.2 estándar y sobre esta se instalan una serie de librerías.

En este caso, la librería más relevante es “**Paho**” [10]. Esta librería es la que permite crear un cliente MQTT programado en Python. Traduce los mensajes recibidos por el gestor domótico en protocolo MQTT, para que puedan ser usados por los otros componentes del sistema.

La librería incluye una serie de funciones que permite al servicio de escucha:

1. Conectarse al servidor MQTT.
2. Suscribirse a los temas relacionados con los usuarios.
3. Llamar a otra función al recibir mensajes.

Otra librería relevante es “**SQLModel**” [16], esta librería se explicará en detalle en capítulos posteriores. En este servicio, “SQLModel” permite al servicio de escucha realizar peticiones de lectura y escritura a la base de datos.

De esta forma, el servicio de escucha es capaz de buscar en la base de datos a todos los usuarios, sus ubicaciones asociadas y los sensores asociados a estas últimas. Con esta información, el servicio de escucha se suscribe a todos los canales MQTT necesarios. Por otro lado, las peticiones de escritura le permiten guardar en la base de datos todos los mensajes recibidos, asociados al sensor correspondiente.

Teniendo todo esto en cuenta y a modo de resumen, para desarrollar este módulo deben llevarse a cabo las siguientes labores:

1. Crear un programa en lenguaje Python que cubra al 100% la funcionalidad requerida del servicio de escucha. Se debe configurar la dirección IP y los puertos abiertos, tanto del servidor MQTT, como de la base de datos. Además, se debe crear el código necesario para llevar a cabo sus funciones básicas (conectarse, suscribirse...).
2. Crear y compilar el archivo Docker personalizado a partir de la imagen base de Python, añadiendo las librerías necesarias (Paho-mqtt, SQLModel...) y el programa de escucha.
3. Configurar en el contenedor los volúmenes de datos asociados a él y la función de reinicio autónoma.

5.3.2.3. Diseño del Servicio de almacenamiento de datos (PostgreSQL)

Otro de los grandes pilares de este proyecto es su servicio de almacenamiento de datos. En primer lugar, se debe definir el modelo de datos adecuado para el proyecto.

1. Modelo de datos

Por la tipología de los datos, y las relaciones existentes entre ellos a primera vista, se decide crear un modelo de datos basado en tablas relacionales. De acuerdo con los objetivos del sistema y los requisitos no funcionales: “RNF-SAD-1” y “RNF-SAD-2”, se diseña el modelo de datos de la Figura 5. Se presenta la figura con orientación vertical para una mejor visualización.



Figura 5: Modelo de datos

En la Figura 5, se observa cómo el modelo de datos está compuesto por una serie de tablas relacionadas entre sí a través de **claves primarias** o “*primary keys*” y **claves foráneas** o “*foreign keys*”. Estas claves son las encargadas de permitir la búsqueda de datos entre las tablas, relacionando los sensores con su ubicación correspondiente y ésta a su vez con el usuario al que pertenece.

El modelo consta de cinco grandes bloques: **usuarios, ubicaciones, sensores, eventos y registro histórico de datos**. Los usuarios, las ubicaciones y el registro de datos están formados cada uno de ellos por una única tabla de datos. Mientras que los sensores y los eventos necesitan del funcionamiento conjunto de varias tablas para cumplir con los requisitos establecidos previamente.

El punto fuerte de este modelo radica en cómo se han estructurado las relaciones entre las tablas de sensores y las relaciones entre las tablas de eventos.

El requisito “RNF-SAD-1” exige al sistema contar con la capacidad de ampliar los sensores compatibles de manera fácil, sin necesidad de mucho esfuerzo. Esta tarea no es sencilla, ya que en un ambiente domótico existen dispositivos de ámbitos muy distintos, cada uno de ellos con un número distinto de campos de entrada y variables de distinto tipo.

La solución a este problema pasa en primer lugar, por diseñar el sistema de forma que no se necesiten añadir nuevas tablas para incorporar otros sensores, ni existan tablas con una cantidad de columnas variables, ya que esto complicaría tanto la incorporación de nuevos sensores como la gestión de sus datos por parte de otros módulos.

Por ello, se ha creado la tabla “sensor_types” de la Figura 5. Esta tabla, cuyo único dato es el tipo de sensor, lleva asociados a través de su clave primaria un número indefinido de elementos de la tabla “sensor_fields”. Esto permite que cada sensor tenga asociados un número variable de campos sin necesidad de realizar cambios en el modelo de datos.

Para cumplir con el requisito funcional: “RF-SAD-6”, se ha diseñado el sistema de forma que un usuario puede tener “n” ubicaciones y ésta a su vez “m” sensores.

Adicionalmente, la estructura del modelo de datos permite crear “**eventos**” los cuales se han definido como la unión entre una serie de “**condiciones**” que, al cumplirse, deben desencadenar una serie de “**acciones**”.

Se entiende como **condición**, que el valor de una variable numérica medida por un sensor sea mayor, menor, igual o distinta que un umbral establecido; o en caso de una variable booleana o una enumeración, que el valor sea distinto o igual que un valor dado.

Se entiende como **acción**, el envío de uno o varios mensajes MQTT al sensor o sensores correspondientes, para que dicho sensor o sensores cambien al estado deseado.

De esta forma, un usuario puede utilizar los eventos para combinar condiciones y acciones relacionadas con cualquier sensor de cualquier ubicación. Cumpliendo así con el objetivo de buscar la máxima personalización posible del sistema por parte del usuario.

2. Desarrollo del contenedor del Servicio de Almacenamiento de Datos “SAD”

Al igual que los otros componentes, el servicio de almacenamiento de datos está programado en un contenedor independiente. Para crear dicho contenedor, el primer paso es elegir el gestor de bases de datos adecuados.

Teniendo en cuenta que el modelo de datos se basa en tablas relacionales, se escoge un gestor de bases de datos cuyo lenguaje de programación sea SQL. Actualmente, existen múltiples gestores de bases de datos válidos para un proyecto como el que se está desarrollando.

Se descartan grandes gestores como “Oracle DB” y “Microsoft SQL Server” al ser opciones de pago. De entre las opciones restantes de libre uso, se escoge PostgreSQL [12] por tener una imagen Docker oficial actualizada periódicamente, aunque también hubieran sido válidas otras opciones como MariaDB o MySQL.

Una vez seleccionado el gestor de bases de datos, en este caso PostgreSQL, se crea un contenedor a partir de su imagen oficial. Para ello se llevan a cabo las siguientes labores:

1. Utilización de la imagen Docker “postgres”.
2. Configuración de los puertos de comunicación.
3. Configuración del rol de administrador de la base de datos (nombre, contraseña, permisos...).
4. Incorporación de los volúmenes de datos necesarios del contenedor.

Para la futura validación del proyecto, se lleva a cabo la inicialización de la base de datos con una serie de instancias de las clases: usuarios, ubicaciones, sensores... que permitan probar las distintas funciones programadas.

Por último, con relación a la base de datos, se instala el programa “DBBeaver” [17], un *software* de libre uso que permite visualizar con facilidad los datos contenidos en una base de datos y las relaciones entre tablas existentes.

5.3.2.4. Servidor de aplicaciones-web (FastAPI)

Este componente es el núcleo central del proyecto, en él se desarrollan las principales funciones del sistema.

1. Desarrollo del contenedor

El servidor de aplicaciones-web, es el núcleo de este proyecto.

La centralización de todos los procesos posibles en este servicio permite cumplir con el objetivo del proyecto, que busca minimizar el coste para el usuario final.

El servidor de aplicaciones-web, tendrá su propio contenedor Docker. Al ser un servicio personalizado, se deberá crear una imagen del contenedor a partir de la imagen base de Docker de Python 3.11.2. A esta imagen se le añaden las librerías necesarias requeridas por el programa principal, y se compila para obtener la imagen Docker del servidor.

Una vez obtenida la imagen, se debe configurar el contenedor. Para ello se llevan a cabo las siguientes labores:

1. Crear el programa principal en Python. En él se configuran las direcciones IP y los puertos necesarios para comunicarse con los otros módulos del sistema.
2. Se define el puerto de conexión para el contenedor del servidor.
3. Se asocian al contenedor los volúmenes de datos necesarios para ejecutar el programa principal.

2. Librerías necesarias

Para poder llevar a cabo el desarrollo del programa principal, se han usado múltiples librerías cuyas funcionalidades serán claves en este proyecto y sin las cuales no hubiera sido posible.

Se van a describir a continuación aquellas más importantes, explicando cuáles han sido las ventajas de su uso:

1. **Uvicorn** [18]: Uvicorn es una librería que se encarga de implementar un servidor ASGI (Asynchronous Server Gateway Interface) en Python. Una vez levantado el servidor, Uvicorn se encarga de la gestión de las solicitudes HTTP y se las transmite, en este caso, a la aplicación creada sobre FastAPI. Esta librería es la recomendada por FastAPI como servidor, al estar diseñada para ser rápida y eficiente con frameworks como Starlette o el propio FastAPI.
2. **FastAPI** [19]: FastAPI es un framework que permite el fácil y rápido desarrollo de APIs basadas en Python. Una de sus principales ventajas es su alto rendimiento, siendo FastAPI uno de los frameworks más rápidos [20].

Sin embargo, las ventajas de FastAPI van más allá del rendimiento. Este framework destaca también por su facilidad de aprendizaje, su capacidad de ayudar al programador en las labores de desarrollo con las funciones de autocompletado y las funciones de validación.

También permite la validación automática a través de la definición de los tipos de datos esperados y genera documentación interactiva de la API automáticamente. Permitiendo probar a los desarrolladores el correcto funcionamiento de las rutas o “*endpoints*” definidos.

Por último, permite una fácil integración de otras dependencias en los endpoints creados.

3. **SQLModel** [16]: SQLModel es la librería que se encarga principalmente de gestionar la interacción de los programas desarrollados en Python con bases de datos SQL. Ha sido desarrollada por el creador de FastAPI por lo que está diseñada para que dicha librería sea completamente compatible y explote al máximo sus ventajas.

Mediante el uso de esta librería, se gestionan las labores de escritura y lectura de la base de datos directamente en Python, sin necesidad de utilizar SQL en ningún momento.

Permite la creación de clases que funcionan al mismo tiempo como modelos de datos, evitando la duplicación de código y mejorando la claridad del mismo. Al igual que FastAPI, permite la definición de tipos de datos para llevar a cabo una validación automática. SQLModel también permite la integración de otras dependencias ampliamente usadas como SQLAlchemy.

4. **Paho** [10]: Paho es la librería utilizada para poder implementar un cliente MQTT en Python. Esta librería permite al servidor de aplicaciones mandar mensajes MQTT a los sensores de cada usuario y de esta forma realizar acciones sobre ellos.

5. **Jinja2** [21]: Jinja2 es la librería que permite la creación de una interfaz de usuario HTML personalizada. Esta librería permite embeber código dentro de una plantilla HTML y al llamar a una ruta con los datos de un usuario, renderiza el HTML en función de sus datos.
6. **HTMX** [22]: HTMX es una biblioteca que permite modificar el HTML de un usuario de manera rápida y cómoda para él.

Mediante su uso, se puede diseñar una plantilla HTML para que se refresquen solo los elementos necesarios de una página web en función de la actividad del usuario o programar cambios automáticos en la página en función del tiempo u otras variables. Todo esto se realiza de manera no intrusiva para el usuario final, al evitar recargar la página en su totalidad.

El uso de todas estas herramientas en conjunto permite llevar a cabo el desarrollo buscado para el servidor de aplicaciones-web cumpliendo con los objetivos del proyecto.

3. Desarrollo del programa principal

El programa principal debe incluir los servicios descritos en la Figura 4. Debido a la gran cantidad de funciones, el programa principal se va a organizar a través de múltiples módulos, cada uno de ellos enfocado en realizar una serie de funciones.

Cada servicio de la Figura 4 tiene su propio módulo, con la excepción del servicio de respuestas HTML que se encuentra embebido en los otros módulos a través del uso de la librería Jinja2.

Además de los módulos ligados a alguno de los servicios de la Figura 4, también existen varios módulos extra necesarios para gestionar el modelo de la base de datos y la inicialización de la misma.

A continuación, se describen los distintos módulos en profundidad:

1. **Módulo principal:** este módulo consta de las importaciones de los otros módulos y se encarga de la creación del servidor a través de la librería Uvicorn. En él se programa la dirección IP del servidor y los puertos de conexión necesarios.
2. **Módulo del modelo de datos:** en este módulo se programan las distintas clases de datos asociadas al modelo definido en la Figura 5. Mediante el uso de SQLAlchemy, estos modelos servirán para crear las tablas de datos de la base de datos, pero también podrán ser utilizados como clases estándar en Python.
En este módulo también se programan las relaciones entre tablas que permiten la búsqueda de información mediante claves primarias y foráneas.
3. **Módulo de inicialización y conexión:** en este módulo se introducen las credenciales de acceso a la base de datos y se crea una conexión. Dicha conexión permitirá posteriormente crear múltiples sesiones de acceso a la base de datos para la lectura o escritura.
En este módulo también se realiza la inicialización de la base de datos mediante un conjunto de usuarios, ubicaciones, sensores... datos ficticios que permitan la posterior prueba y validación de las funciones programadas.

4. **Módulo de acceso:** en este módulo se definen las funciones relativas al acceso a la página web, por parte de los usuarios, mediante credenciales.
5. **Módulos de usuarios, ubicaciones y sensores:** en estos módulos se programan las distintas funciones relativas al **CRUD** (funciones “Create”, “Read”, “Update y “Delete”) de usuarios, ubicaciones y sensores. En cada uno de los módulos se programan las funciones relativas a la creación, actualización, y borrado de usuarios, ubicaciones y sensores. También se incluyen una serie de funciones orientadas a la administración del sistema, como la creación de un listado de cada uno de estos grupos, la búsqueda de un elemento en función de su ID, o la creación a partir de un ID, de un diccionario (entendiendo diccionario en el sentido de Python) de un elemento y todos aquellos elementos relacionados con él.
6. **Módulo de envío de mensajes:** en este módulo se encuentran las funciones relacionadas con el envío de mensajes MQTT por parte del servidor de aplicaciones-web a los sensores. También incorpora varias funciones de transformación de datos necesarias para enviar los mensajes MQTT de forma correcta.
7. **Módulo de Eventos (condiciones-acciones):** este módulo es el de mayor tamaño y el más complejo. Se encarga del CRUD de las condiciones-acciones. Su dificultad radica en la necesidad de crear una página web que no permita al usuario final crear condiciones-acciones erróneas, ya sea por el uso de un tipo de sensor que no posee, o por la introducción de una comparación lógica errónea. Por ello, la labor principal de este módulo es garantizar que toda condición-acción ha sido creada de manera adecuada y de manera consistente.

Este módulo también incorpora la importante función de chequear, cada vez que se recoge un cambio en el estado de los sensores de un usuario, si se cumple alguna de las condiciones definidas por dicho usuario, y en ese caso llevar a cabo las correspondientes acciones.

Además de los módulos comentados, existe un grupo de archivos de alta importancia. Previamente, se ha comentado que el **servicio de respuesta HTML** está embebido en las sentencias de los otros módulos. Sin embargo, también es necesario crear una serie de plantillas HTML, que el programa sea capaz de personalizar durante su renderización, mediante la librería Jinja2.

Existen múltiples plantillas, desde las más básicas, para el acceso y el registro de nuevos usuarios, sensores... hasta las más complejas, completamente personalizadas con la ayuda de Jinja2, que también incorporan HTMX para una mejor experiencia de usuario. Cumpliendo así con los objetivos del sistema relacionados con el fácil uso y la comodidad del usuario. Es decir, cumpliendo con los requisitos: “RNF-SG-3”, “RNF-SAW-SR-1”, “RNF-SAW-SR-2 “, “RNF-SAW-SR-4”.

6. RESULTADOS

En este capítulo se recogen los resultados obtenidos tras el desarrollo del proyecto.

Primero, se expone el sistema domótico obtenido con sus componentes implementados.

En segundo lugar, se hace un breve resumen de la cantidad de líneas de código y archivos desarrollados. También se indican métricas como el tiempo durante el cual se ha probado el sistema, los fallos surgidos, tiempo entre “pings” de los sensores...

Después, se enumeran las distintas funciones creadas en cada módulo, realizando una breve descripción de cada una de ellas.

A continuación, se explica el funcionamiento de la interfaz web para los usuarios, adjuntando una serie de capturas que enseñan cómo se navega entre las distintas secciones de la misma y qué funcionalidades ofrecen.

Por último, se analizan los posibles impactos sociales, legales, éticos... que pueda tener el proyecto.

6.1. Sistema domótico obtenido

El principal resultado de este proyecto es el sistema domótico obtenido tras su desarrollo. Para ilustrar de manera gráfica el sistema, se adjunta en la Figura 6 los elementos que componen el sistema real utilizado.



Figura 6: Sistema Domótico obtenido

En la Figura 6, se encuentran a la izquierda todos los sensores domóticos utilizados en el sistema. Enumerados por filas, los sensores son los siguientes:

1. Sensor de proximidad
2. Sensor de temperatura y humedad
3. Pulsador Sonoff 1
4. Sensor de presencia Marmitek

5. Alarma
6. Enchufe inteligente
7. Pulsador Sonoff 2
8. Sensor de presencia Sonoff

A su derecha, se encuentra la Raspberry Pi, que actúa como *homehub*, junto con el adaptador y la antena receptora que permite recibir los mensajes Zigbee de los sensores.

Por último, se encuentra la Raspberry Pi encargada de simular el servidor en la nube.

Una vez terminado el desarrollo del sistema, su funcionamiento ha sido probado durante un periodo de tiempo suficientemente extenso como para asegurar su funcionamiento correcto.

Todo este sistema es fácilmente exportable y replicable. Para ello, es preciso disponer de un dispositivo capaz de ejecutar Linux, que actúe como *homehub*, de los sensores que se quieran implementar en el sistema y de otro dispositivo Linux o un servidor en la nube que sea capaz de ejecutar Linux para llevar a cabo las labores del gestor domótico.

El *software* desarrollado para el sistema, al estar diseñado mediante contenedores, sería fácilmente exportable. Solo sería necesario instalar el *software* Docker en ambos dispositivos Linux y ejecutar sobre ellos los contenedores correspondientes al *homehub* y el gestor domótico.

6.2. Resumen de los resultados obtenidos

A lo largo del proyecto, se han desarrollado **1.423 líneas de código Python**. Estas se encuentran distribuidas entre los distintos módulos comentados en apartados anteriores.

El código realizado se compone principalmente de las funciones que se comentarán más adelante, las cuales permiten cumplir con los requisitos funcionales del sistema.

Para el desarrollo de la interfaz de usuario, han sido escritas **1.995 líneas de código HTML**. Éstas se encuentran distribuidas entre **24 archivos de HTML**. Su desarrollo permite alcanzar los objetivos del proyecto relacionados con la interfaz de usuario y su facilidad de uso por parte del usuario.

Respecto al funcionamiento del sistema final, éste ha sido probado de manera real durante 35 días de manera continuada. A lo largo de este periodo, se han realizado múltiples pruebas de funcionamiento del sistema y éste ha estado siempre disponible para los usuarios, con la excepción de varias desconexiones del sistema por fallos en el suministro eléctrico. Durante estas desconexiones por falta de suministro eléctrico, el sistema quedó momentáneamente inaccesible para los usuarios, no permitiendo la gestión de sus sensores.

Por otro lado, el sistema también fue reiniciado de manera voluntaria para actualizar las librerías software a una nueva versión disponible durante el periodo de prueba del sistema.

En cuanto a los sensores, la mayor parte de ellos han funcionado correctamente durante el periodo de pruebas, sus medidas enviadas eran correctas (temperatura, humedad...) y no daban falsos positivos (sensor de movimiento o de proximidad). Sin embargo, el sensor de movimiento Sonoff tuvo que ser reemplazado al funcionar de manera errónea y no detectar movimiento en multitud de ocasiones, razón por la cual fue sustituido por el sensor de movimiento de la marca Marmitek.

Los reportes de los sensores cumplían con los valores por defecto de la librería "Zigbee2MQTT". Dicha librería establece, que el tiempo mínimo entre reportes será de 0

segundos, el tiempo máximo de una hora y para aquellos sensores con medidas numéricas, estables, que se reportarán cambios de 10 unidades.

Todos estos valores afectan al consumo energético de los sensores del sistema, y es de gran importancia al ser sensores alimentados a través de una batería. Por ello, la librería “Zigbee2MQTT” permite cambiar dichos valores en función de la configuración deseada por el usuario.

6.3. Funciones desarrolladas

A continuación, se adjuntan una serie de tablas con las funciones correspondientes a cada módulo del servidor de aplicaciones-web y al servicio de escucha. Casi la totalidad de las funciones desarrolladas durante el proyecto, son parte del servidor de aplicaciones-web, las pocas funciones restantes son parte del servicio de escucha.

En la Tabla 15 se describen las funciones desarrolladas para el correcto funcionamiento del **servicio de escucha**.

Tabla 15: Funciones del servicio de escucha

Nombre	Descripción
List_01 “on_message”	Al detectar un mensaje MQTT, identifica el sensor emisor, guarda el mensaje en la tabla de datos históricos y realiza una llamada a la función de chequeo de condiciones-acciones.
List_02 “on_connect”	Al iniciarse el servicio de escucha, chequea que la conexión con el servicio de mensajería MQTT ha sido exitosa, y en caso afirmativo busca en la base de datos todos los sensores existentes y se subscribe a los canales MQTT asociados.

Todas las funciones restantes pertenecen al servidor de aplicaciones-web y se adjuntan en distintas tablas. Cada una de ellas relativa a uno de los módulos descritos en la sección anterior.

En la Tabla 16 se recoge la única función desarrollada en el **módulo principal**:

Tabla 16: Funciones del módulo principal

Nombre	Descripción
Main_01 “login”	Al ser llamada a través de una ruta, devuelve la página HTML de inicio que permite registrarse o acceder a través de unas credenciales.

El **módulo del modelo de datos** está definido a través de la creación de distintas clases, en vez de funciones. Por ello, para recoger los resultados del desarrollo de este módulo, se adjuntan las tablas: Tabla 17, Tabla 18, Tabla 19, Tabla 20, Tabla 21, Tabla 22, Tabla 23 y Tabla 24.

Estas tablas recogen las estructuras de datos principales utilizadas durante el proyecto. Existen múltiples tablas que derivan de estas, necesarias para cumplir con ciertas funciones del proyecto, que no se adjuntan para mejorar la legibilidad del texto.

Tabla 17: Campos de la clase Usuarios

Clase Usuarios	
Campo	Tipo de variable
ID	Optional int
Nickname	Optional str
First_name	Optional str
Last_name	Optional str
Legal_id	Optional str
Phone_number	Optional str
Email	Str, unique
Access_login_name	Str, unique
Access_password	Str
Email_validated	Bool
Created_at	Optional datetime

Tabla 18: Campos de la clase Ubicaciones

Clase Ubicaciones	
Campo	Tipo de variable
ID	Optional int
User_id	Int
Nickname	Str
Country	Optional str
Province	Optional str
City	Optional str
Street	Optional str
Number	Optional str
Floor	Optional str
Door	Optional str
Postcode	Optional str

Tabla 19: Campos de la clase Sensores

Clase Sensores	
Campo	Tipo de variable
ID	Optional int
Location_id	Int
Nickname	Str
Type	Int
Serial_number	Optional str
Vendor	Optional str
Model	Optional str
Mac_address	Optional str

Tabla 20: Campos de la clase atributos de un tipo de sensor

Clase atributos de un tipo de sensor	
Campo	Tipo de variable
ID	Optional int
Sensor_types	Int
Field_name	Str

Clase atributos de un tipo de sensor	
Field_values	Str
Field_values_to_print	Str
Name_type	Str
Field_updatable	Bool

Tabla 21: Campos de la clase datos históricos de los sensores

Clase datos históricos de los sensores	
Campo	Tipo de variable
ID	Optional int
Sensor_id	Int
Date_time	Optional datetime
Field_name	Str
Field_value	Str

Tabla 22: Campos de la clase activador de eventos

Clase activador de eventos	
Campo	Tipo de variable
ID	Optional int
User_id	Int
Nickname	Str
Is_active	Bool

Tabla 23: Campos de la clase grupo de eventos

Clase grupo de eventos	
Campo	Tipo de variable
ID	Optional int
Trigger_id	Int
Nickname	Optional str
Type_event	Str

Tabla 24: Campos de la clase evento

Clase eventos	
Campo	Tipo de variable
ID	Optional int
Grouping_id	Int
Sensor_id	Int
Field_id	Int
Field_name	Str
Field_comparison	Str
Field_value	Str
Field_values_to_print	Str

Estas clases, junto con las relaciones definidas en el apartado referente al modelo de datos, permiten crear, almacenar y gestionar todos los datos necesarios para el sistema de manera ordenada.

El **módulo de inicialización y conexión** cuenta con una única función: “create_DB_and_tables”. Esta función es la encargada de inicializar la base de datos.

Para ello, en primer lugar, borra cualquier dato previo existente en la base de datos (para evitar datos erróneos generados durante las pruebas de funcionamiento), y a continuación, crea las distintas tablas de datos del modelo y las inicializa con una serie de instancias de cada clase. Esto permite tener una base de limpia y conocida cada vez que se reinicia el sistema, realizando de manera más sencilla las pruebas del sistema una vez desarrollado.

El **módulo de acceso** cuenta también con una única función: “Access”. Esta función se encarga de permitir o no el acceso de un usuario a la página web. Para ello, la función verifica que el nombre de usuario y la contraseña coinciden con los de un usuario de la base de datos a través de una consulta SQL. En caso de realizar un acceso correcto, la función devuelve el HTML de la página principal, personalizado con los datos del usuario correspondiente.

Las funciones desarrolladas para los **módulos de usuarios, ubicaciones y sensores** siguen una estructura muy similar, ya que estos módulos buscan cumplir con los requisitos funcionales impuestos por un CRUD (Create, Read, Update, Delete). Se adjuntan las tablas Tabla 25, Tabla 26 y Tabla 27 que recogen todas las funciones desarrolladas de dichos módulos.

Tabla 25: Funciones del módulo de usuarios

Nombre	Descripción
User_01 “register_user”	Esta función devuelve el HTML correspondiente al registro de nuevos usuarios.
User_02 “create_user”	Esta función recoge los datos del formulario de registro HTML y crea un nuevo usuario con dichos datos en la base de datos. Además, devuelve al usuario a la página de acceso.
User_03 “show_users”	Esta función permite al administrador del sistema obtener un listado de todos los usuarios existentes en el momento de la consulta.
User_04 “Search_user”	Esta función permite buscar los datos de un usuario a través de su campo ID, identificador único de cada usuario.
User_05 “Search_user_AD”	Esta función permite buscar todos los datos de un usuario y de cualquier elemento vinculado a este a través del ID del usuario. Devuelve todos los datos de ubicaciones, sensores, condiciones-acciones... vinculados a un usuario.
User_06 “HTML_Update_user”	Esta función devuelve el HTML correspondiente para la actualización de los datos de un usuario.
User_07 “Update_user”	Esta función recoge los datos introducidos en un formulario HTML de actualización de usuario. A continuación, realiza una consulta a la base de datos, actualizando los datos del usuario.
User_08 “Delete_user”	Esta función se encarga de borrar a un usuario de la base de datos.

Tabla 26: Funciones del módulo de ubicaciones

Nombre	Descripción
Location_01 “register_location”	Esta función devuelve el HTML correspondiente al registro de nuevas ubicaciones.
Location_02 “create_location”	Esta función recoge los datos del formulario de registro HTML y crea una nueva ubicación con dichos datos en la base de datos. Además, devuelve al usuario a la página de acceso.

Nombre	Descripción
Location_03 “show_locations”	Esta función permite al administrador del sistema obtener un listado de todas las ubicaciones existentes en el momento de la consulta.
Location_04 “Search_location_AU”	Esta función permite buscar todos los datos relacionados con una ubicación y sus elementos superiores (usuarios y eventos) asociados a través del ID de la ubicación.
Location_05 “Search_location_AD”	Esta función permite buscar todos los datos de una ubicación y de cualquier elemento inferior (sensores) vinculado a esta a través del ID de la ubicación.
Location_06 “HTML_Update_location”	Esta función devuelve el HTML correspondiente para la actualización de los datos de una ubicación.
Location_07 “Update_location”	Esta función recoge los datos introducidos en un formulario HTML de actualización de ubicación. A continuación, realiza una consulta a la base de datos, actualizando los datos de la ubicación.
Location_08 “Delete_location”	Esta función se encarga de borrar una ubicación de la base de datos.

Tabla 27: Funciones del módulo de sensores

Nombre	Descripción
Sensor_01 “register_sensor”	Esta función devuelve el HTML correspondiente al registro de nuevos sensores.
Sensor_02 create_sensor”	Esta función recoge los datos del formulario de registro HTML y crea un nuevo sensor con dichos datos en la base de datos. Además, devuelve al usuario a la página de acceso.
Sensor_03 “show_sensors”	Esta función permite al administrador del sistema obtener un listado de todos los sensores existentes en el momento de la consulta.
Sensor_04 “Search_sensor”	Esta función permite buscar los datos de un sensor a través de su campo ID, identificador único de cada sensor.
Sensor_05 “HTML_Update_sensor”	Esta función devuelve el HTML correspondiente para la actualización de los datos de un sensor.
Sensor_06 “Update_sensor”	Esta función recoge los datos introducidos en un formulario HTML de actualización de sensor. A continuación, realiza una consulta a la base de datos, actualizando los datos del sensor.
Sensor_07 “Delete_sensor”	Esta función se encarga de borrar a un sensor de la base de datos.

El **módulo de envío de mensajes** consta de 2 funciones. La primera de ellas “from_string_to_Type” se encarga de convertir una cadena de caracteres a una variable del tipo correspondiente.

La segunda función: “Paho_action” es la que lleva a cabo las principales funciones de este módulo. Conocidos el ID del sensor al que se le quiere mandar un mensaje, el nombre del campo afectado, y el estado de dicho campo deseado, esta función se encarga de buscar en la base de datos la ubicación y usuario correspondiente al sensor. A partir de estos datos, genera la dirección MQTT correspondiente al sensor y envía un mensaje con la carga o “payload” prevista.

El **módulo de eventos o de condiciones-acciones** es el más complejo. Esto se debe a la necesidad de desarrollar una serie de funciones auxiliares diseñadas para manejar la creación dinámica de nuevas ventanas a través de HTMX. El uso de HTMX permite crear una interfaz

de usuario que evite los errores humanos a la hora de crear o modificar las condiciones-acciones. En la Tabla 28 se recogen y explican las principales funciones de este módulo:

Tabla 28: Funciones del módulo de eventos

Nombre	Descripción
Events_01 “process_event_dict”	Esta función procesa los datos correspondientes a un evento, almacenados en una variable tipo diccionario.
Events_02 “handle_entity_creation”	Esta función gestiona la creación de nuevos eventos y nuevos grupos de condiciones y acciones.
Events_03 “create_event_structures”	Esta función se encarga de la inicialización de las condiciones y acciones a partir de los datos recogidos por la interfaz de usuario.
Events_04 “find_grouping_ids_by_trigger_id”	Esta función se encarga de buscar todos los grupos de condiciones y acciones asociados a un evento a través de su ID.
Events_05 “create_events”	Esta función redirige al usuario desde la página HTML principal a la página correspondiente a la creación y gestión de eventos.
Events_06 “new_event”	Esta función devuelve el HTML parcial correspondiente para la creación de un nuevo evento.
Events_07 “event_expand”	Esta función devuelve el HTML parcial correspondiente a la configuración de un evento y sus condiciones-acciones.
Events_08 “event_shrink”	Esta función devuelve el HTML parcial que permite colapsar la parte del HTML correspondiente a un evento.
Events_09 “event_sensor”	Esta función devuelve el HTML parcial que permite escoger un sensor válido para una acción o condición.
Events_10 “event_fields”	Esta función actualiza los campos del HTML para que solo se puedan seleccionar como campos válidos los asociados al sensor escogido.
Events_11 “event_op_value”	Esta función actualiza los campos del HTML para que la operación de comparación y los valores posibles, sean válidos para el sensor y campos escogidos.
Events_12 “event_submit”	Esta función recoge los datos de la interfaz de usuario a través de un formulario HTML y los procesa. Además, devuelve el HTML parcial que permite colapsar el evento correspondiente.
Events_13 “event_remove”	Esta función se encarga de borrar un evento existente, también sirve para interrumpir la creación de un evento nuevo. Devuelve el HTML parcial que permite colapsar el evento correspondiente.
Events_14 “check_statement”	Esta función se encarga de realizar la comparación entre el último valor guardado de un sensor y el valor que debería tener para desencadenar una acción o condición.
Events_15 “Check_Trigger_Html”	Esta función es la más importante del módulo. Se encarga de, una vez creados los eventos, condiciones y acciones, chequear si ante un cambio de estado de un sensor, se cumplen las condiciones necesarias para desencadenar una acción. En caso de cumplirse las condiciones de un evento, utiliza al módulo de envío de mensajes para desencadenar las acciones correspondientes.
Events_16 “return”	Esta función permite volver a la página principal del usuario.

6.4. Interfaz web de usuario

La interfaz de usuario es el componente del sistema que permite al usuario final gestionar su sistema domótico desde cualquier lugar. Como se mencionó previamente, han sido necesarias alrededor de 2.000 líneas de código HTML, y el uso de múltiples librerías como Jinja2 y HTMX.

En este apartado, se va a exponer la experiencia de un usuario durante el uso de la interfaz de usuario. Para ello, se utilizarán múltiples figuras del sistema real desarrollado por el que navegaría un usuario.

En primer lugar, cuando un usuario se conecta al servidor, este le devuelve la página de acceso mostrada en la Figura 7.

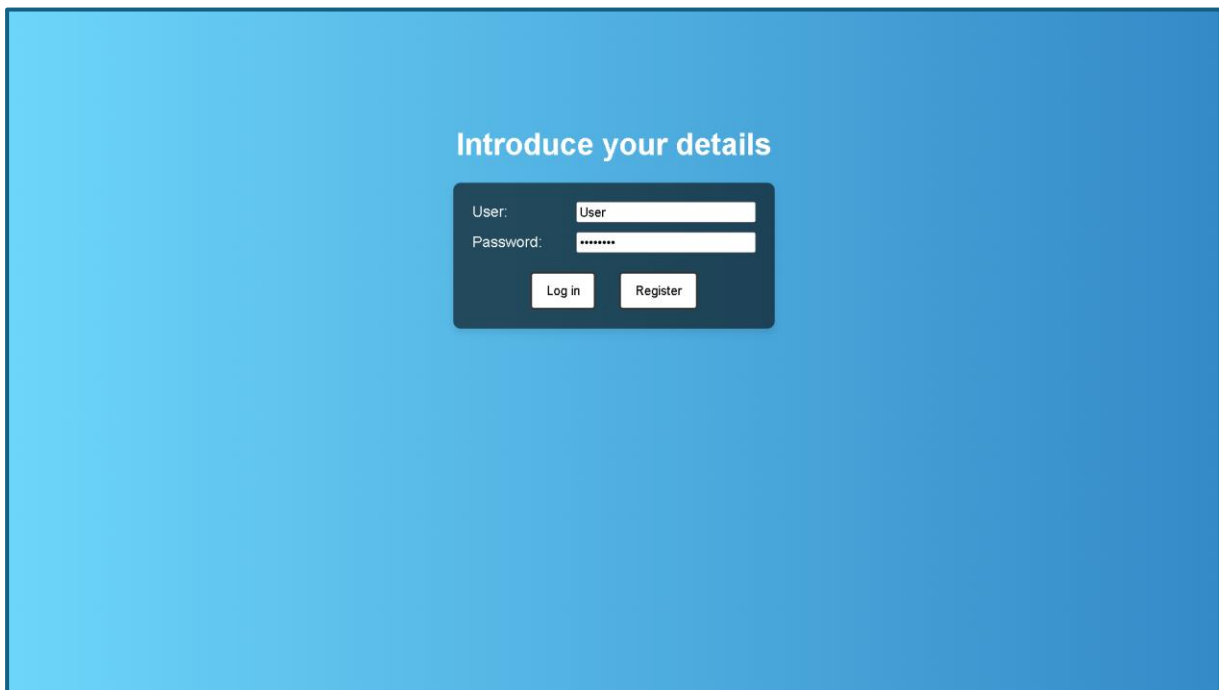
The image shows a web interface for user login. It features a light blue gradient background. In the center, there is a dark blue rounded rectangle containing the text 'Introduce your details' in white. Below this text, there are two input fields: 'User:' with a white input box containing the text 'User', and 'Password:' with a white input box containing seven asterisks. At the bottom of the dark blue rectangle, there are two white buttons: 'Log in' and 'Register'.

Figura 7: Página web de acceso

En caso de que el usuario acceda por primera vez, el botón “*Register*” le permitirá crear un nuevo perfil como se muestra en la Figura 8

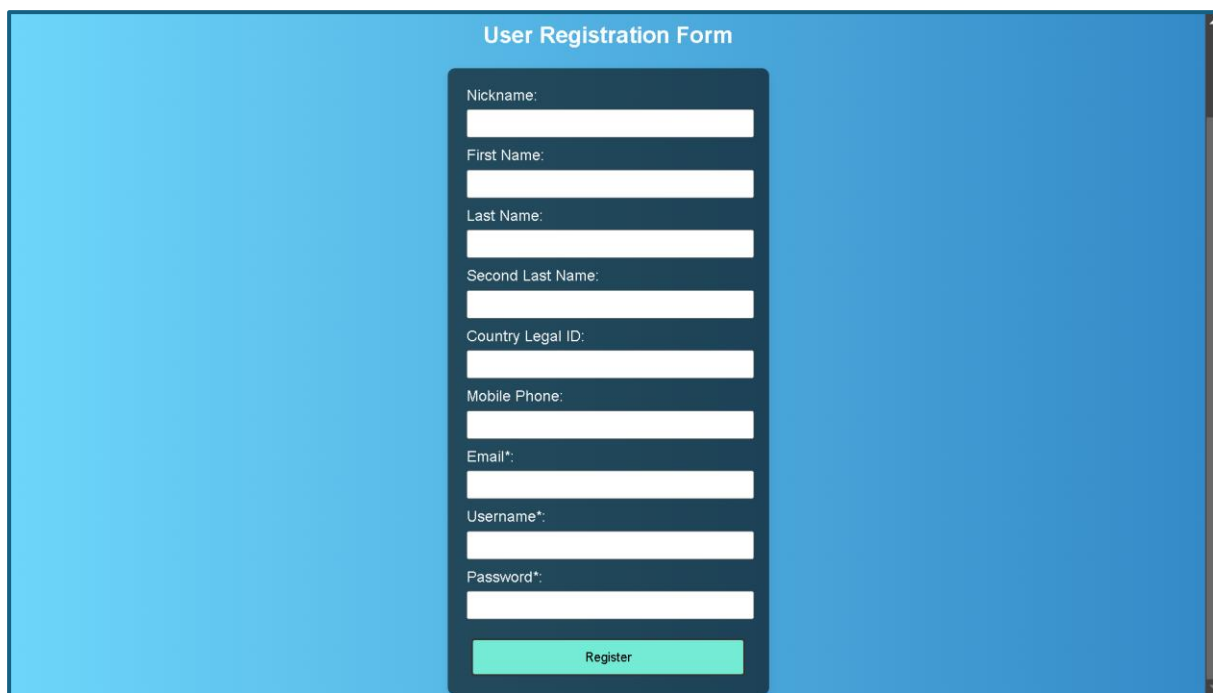
A screenshot of a web browser displaying a 'User Registration Form'. The form is centered on a blue gradient background. It contains several input fields with labels: 'Nickname:', 'First Name:', 'Last Name:', 'Second Last Name:', 'Country Legal ID:', 'Mobile Phone:', 'Email*', 'Username*', and 'Password*'. The asterisk indicates required fields. At the bottom of the form is a green button labeled 'Register'. The browser's address bar and scrollbar are visible on the right side.

Figura 8: Formulario de registro de nuevos usuarios

Esta página web permite al usuario introducir todos sus datos para la creación, de un nuevo perfil. Indica qué datos son imprescindibles mediante un asterisco, y el botón de “*Register*” envía el formulario HTML al servidor con los datos introducidos. Además, devuelve al usuario a la página de acceso.

Si el usuario ya tuviera una cuenta existente, y hubiese pulsado el botón de registro por error, puede volver a la página principal mediante el botón de “*Home*” en la esquina superior izquierda (no visible en la captura para mejorar la visibilidad de la misma).

Si el usuario ya posee una cuenta, pero introduce de manera equivocada su usuario, la página web se actualiza a la mostrada en la Figura 9.

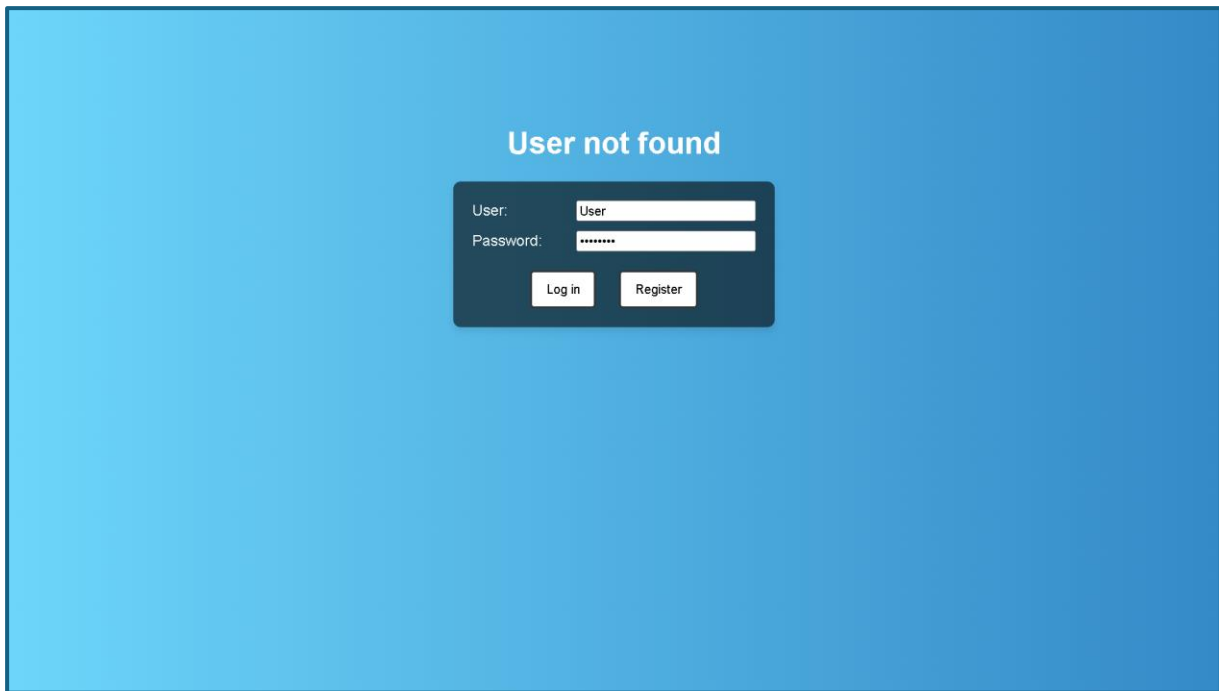


Figura 9: Página web de acceso en caso de no encontrar al usuario

En caso de que el usuario introduzca sus credenciales de manera errónea, la página se actualiza a la mostrada en la Figura 10

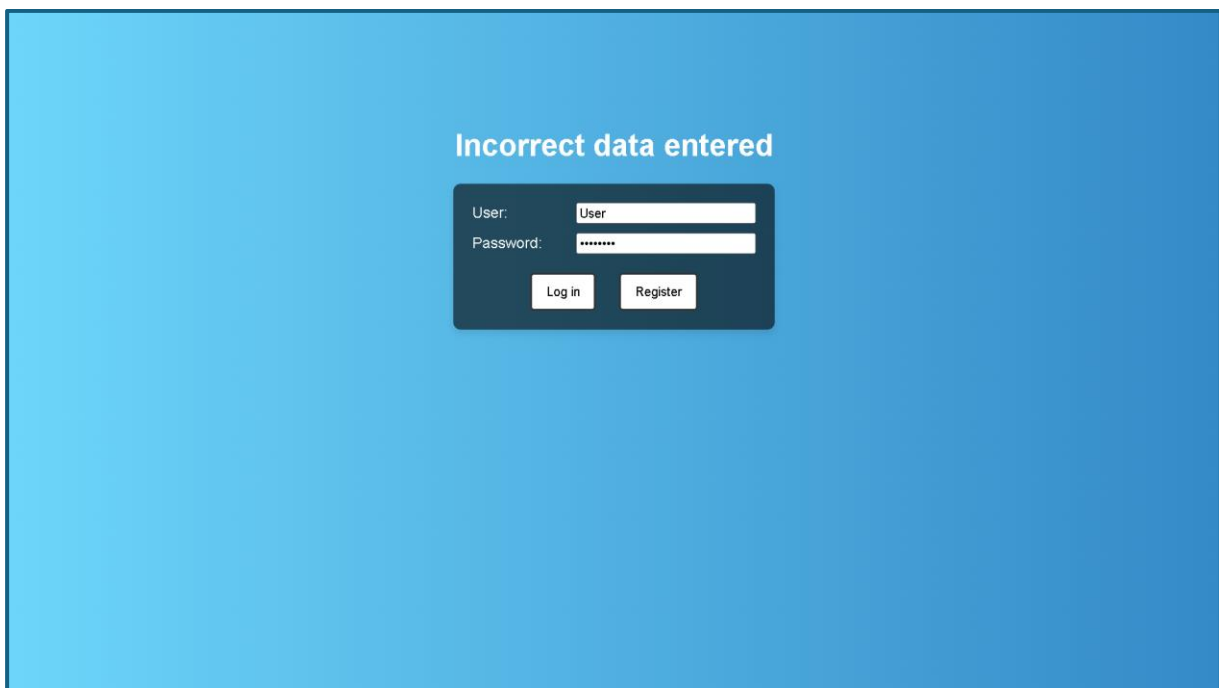


Figura 10: Página web de acceso en caso de credenciales incorrectas

Si el usuario introduce de manera correcta sus credenciales, el servidor de aplicaciones-web le redirigirá a su página principal. Ésta se encuentra personalizada en función de sus datos, ubicaciones, sensores y eventos. Para un usuario nuevo, la página principal sería la mostrada en la Figura 11.

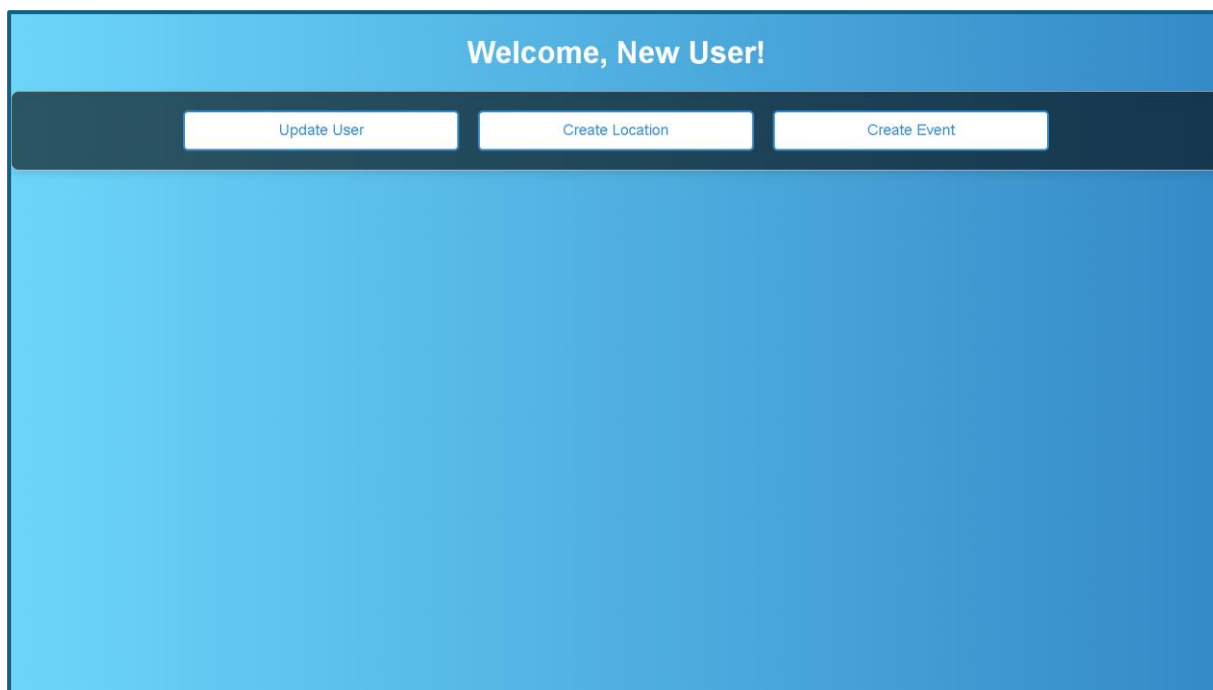


Figura 11: Página web principal de un usuario nuevo

Para mostrar las funcionalidades del sistema, se van a usar los datos de un usuario ficticio con múltiples ubicaciones, sensores y eventos. Para dicho usuario, la página principal sería la mostrada en la Figura 12.

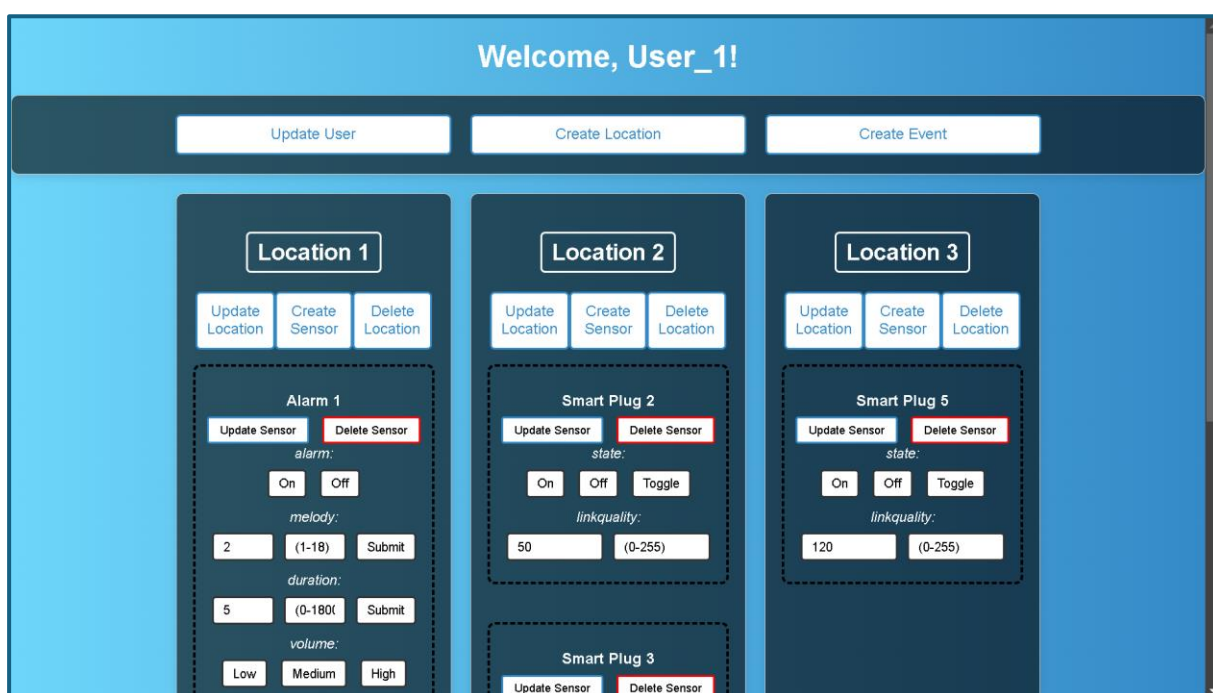


Figura 12: Página web principal de un usuario ficticio

En la Figura 12 se puede observar cómo el servidor de aplicaciones-web realiza una búsqueda de los datos del usuario, a partir de sus credenciales y personaliza la página web de acuerdo con sus datos. En este caso concreto, el usuario de prueba posee 3 ubicaciones, cada una de ellas con una serie de sensores.

Si el usuario desea actualizar sus datos, el botón “*Update User*” le redirige al formulario web mostrado en la Figura 13.

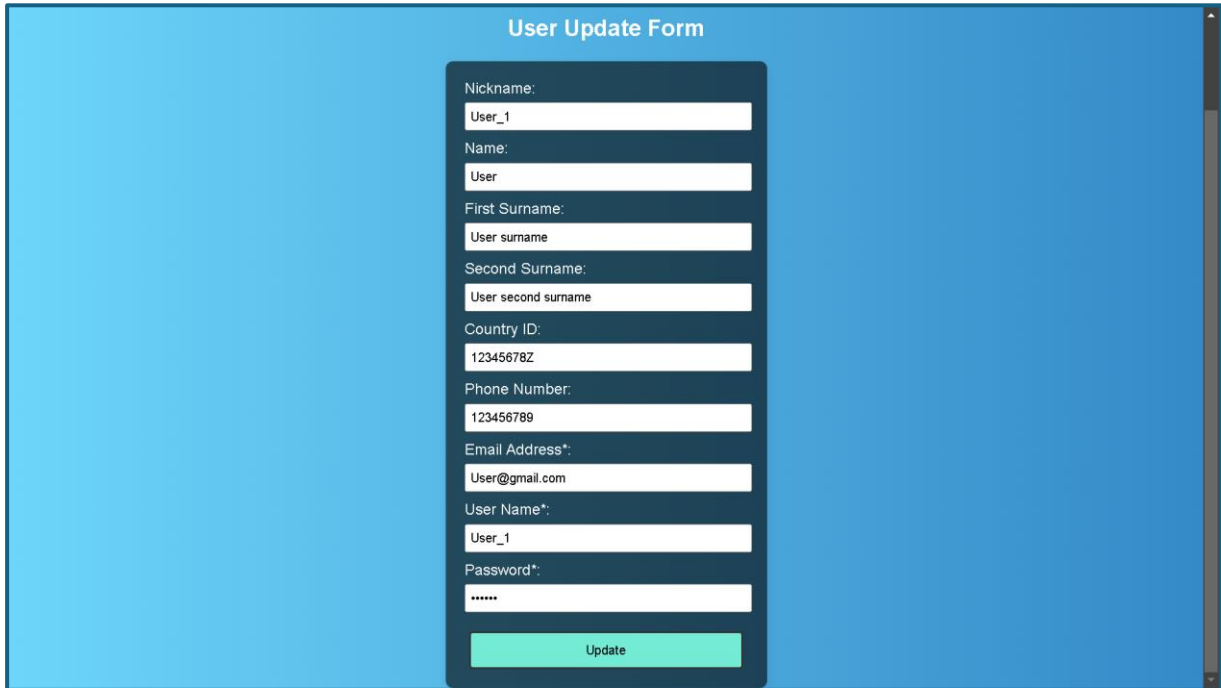
A screenshot of a web browser displaying a 'User Update Form'. The form is centered on a blue gradient background. It contains several input fields with pre-filled text: Nickname (User_1), Name (User), First Surname (User surname), Second Surname (User second surname), Country ID (12345678Z), Phone Number (123456789), Email Address* (User@gmail.com), User Name* (User_1), and Password* (*****). A green 'Update' button is at the bottom.

Figura 13:Formulario web de actualización de los datos de usuario

Una vez relleno el formulario, o al menos los campos marcados como imprescindibles, el botón “*Update*” envía los datos al servidor de aplicaciones-web y devuelve al usuario a su página principal.

Si el usuario necesita crear una ubicación nueva, el botón “*Create Location*”, visible en la Figura 12, le redirige al formulario web mostrado en Figura 14.

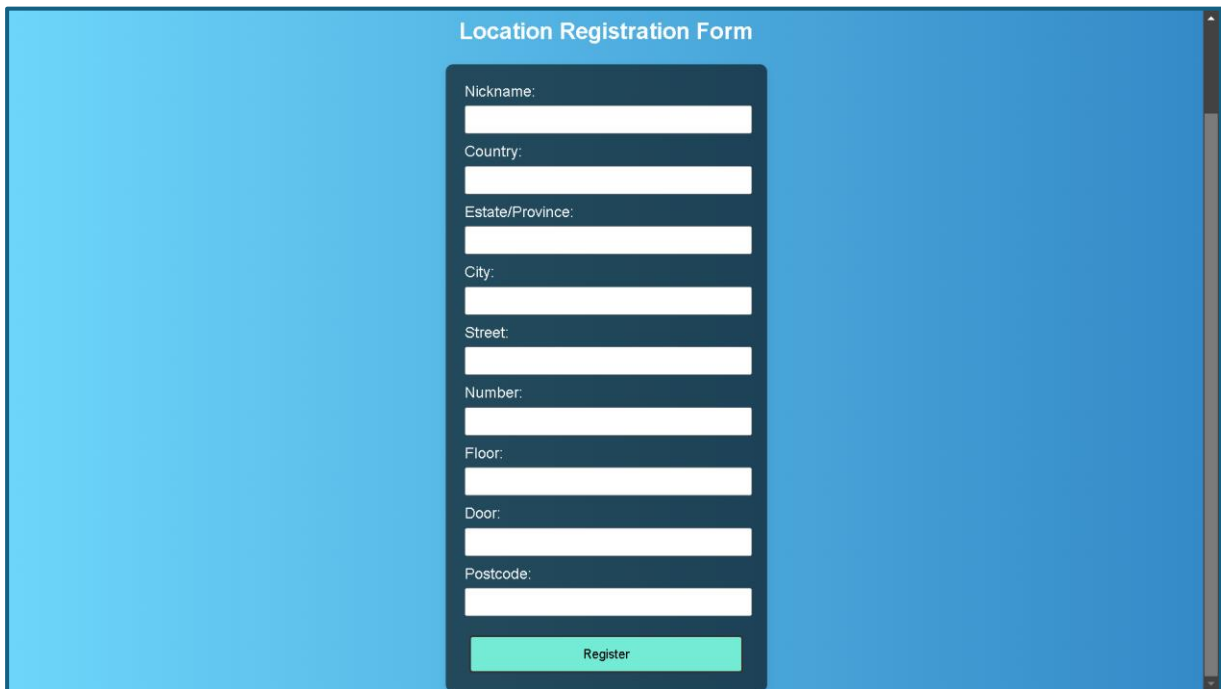
A screenshot of a web browser displaying a 'Location Registration Form'. The form is centered on a blue gradient background. It contains several empty input fields: Nickname, Country, Estate/Province, City, Street, Number, Floor, Door, and Postcode. A green 'Register' button is at the bottom.

Figura 14:Formulario web de creación de una nueva ubicación

Una vez relleno el formulario, el botón “*Register*” envía los datos al servidor de aplicaciones-web y devuelve al usuario a su página principal.

Como se puede observar en la Figura 12, en el menú de cada ubicación, existen una serie de botones, que permiten al usuario final gestionar dichas ubicaciones y los sensores existentes en ella. A través del botón “*Update Location*” el usuario es redirigido a un formulario web que permite actualizar los datos de dicha ubicación, como se muestra en la Figura 15.

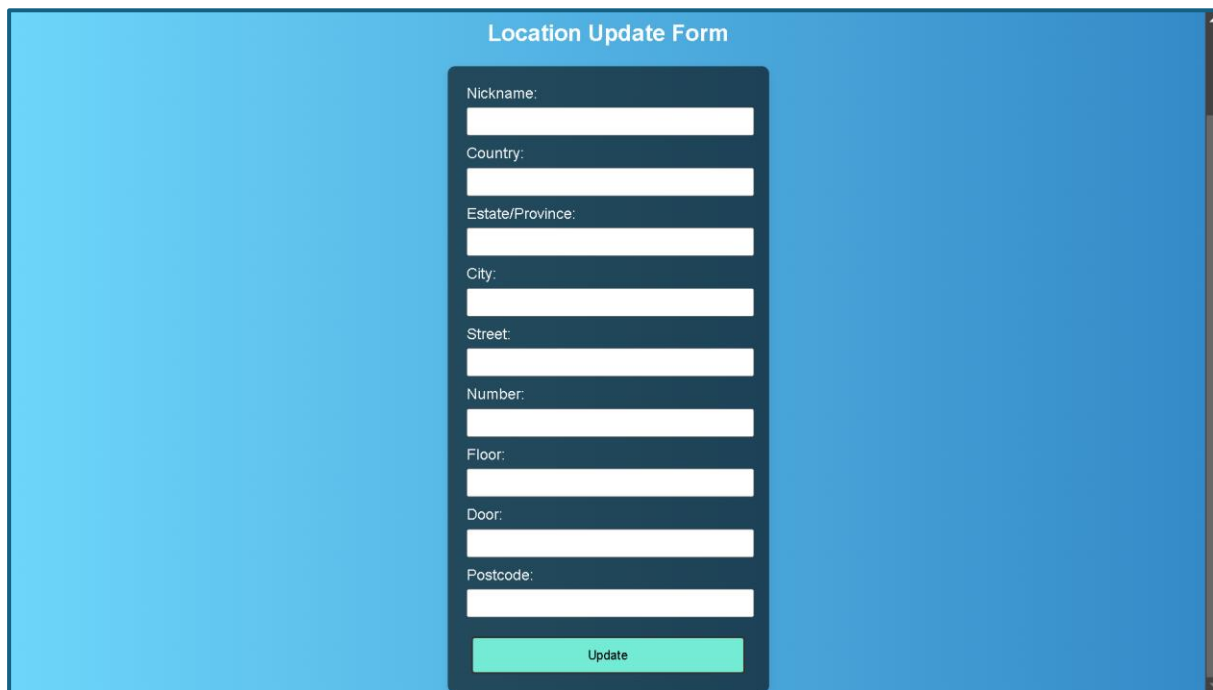
The image shows a web browser window with a blue gradient background. In the center, there is a dark blue rectangular form titled "Location Update Form" in white text. The form contains several input fields with white text labels: "Nickname:", "Country:", "Estate/Province:", "City:", "Street:", "Number:", "Floor:", "Door:", and "Postcode:". Each label is followed by a white rectangular input box. At the bottom of the form is a green rectangular button with the word "Update" in white text.

Figura 15: Formulario web de actualización de una ubicación ya existente

Si el usuario necesita crear un nuevo sensor asociado a una ubicación, deberá pulsar el botón “*Create sensor*”, visible en la Figura 12, de la ubicación deseada. Esto le redirigirá al formulario web mostrado en la Figura 16.

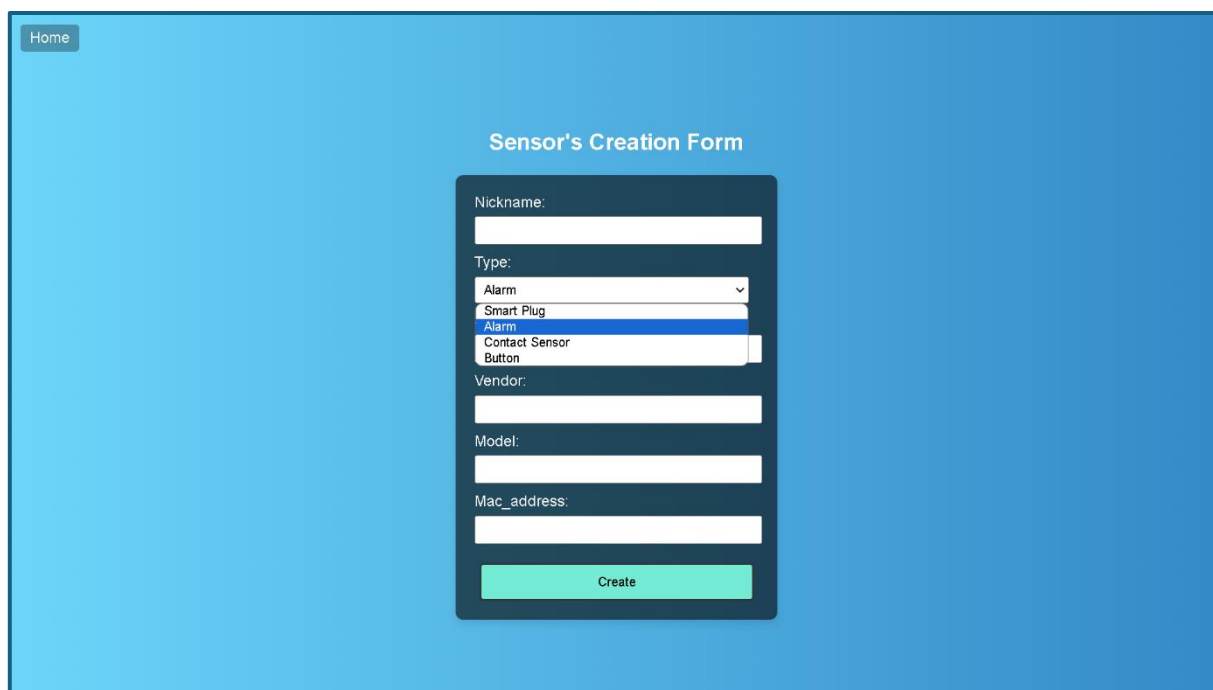
The image shows a web browser window with a blue gradient background. In the top-left corner, there is a small grey button labeled "Home". In the center, there is a dark blue rectangular form titled "Sensor's Creation Form" in white text. The form contains several input fields with white text labels: "Nickname:", "Type:", "Vendor:", "Model:", and "Mac_address:". The "Type:" label is followed by a dropdown menu showing a list of options: "Alarm", "Smart Plug", "Alarm", "Contact Sensor", and "Button". The "Alarm" option is currently selected and highlighted in blue. At the bottom of the form is a green rectangular button with the word "Create" in white text.

Figura 16: Formulario web de creación de nuevos sensores

Este formulario cuenta con un campo especial, el campo “*Type*”. Este campo se ha programado en forma de desplegable al ser un campo que, en caso de ser introducido erróneamente, impediría el correcto funcionamiento del sistema.

Los tipos de sensores compatibles con el sistema están guardados en la base de datos. El formulario despliega como opciones posibles todos los tipos guardados en la base de datos y en caso de actualizarse ésta con la incorporación de nuevos tipos de sensores, también se actualiza automáticamente el listado del campo “*Type*”.

Una vez rellenado el formulario, el botón “*Create*” guarda los datos en la base de datos y devuelve al usuario a la página principal.

Si un usuario desea borrar definitivamente una ubicación, el botón “*Delete Location*”, visible en la Figura 12, le permite hacerlo. Para asegurar que el usuario no se ha equivocado al pulsar este botón, el sistema muestra un mensaje de advertencia al usuario como el mostrado en la Figura 17.

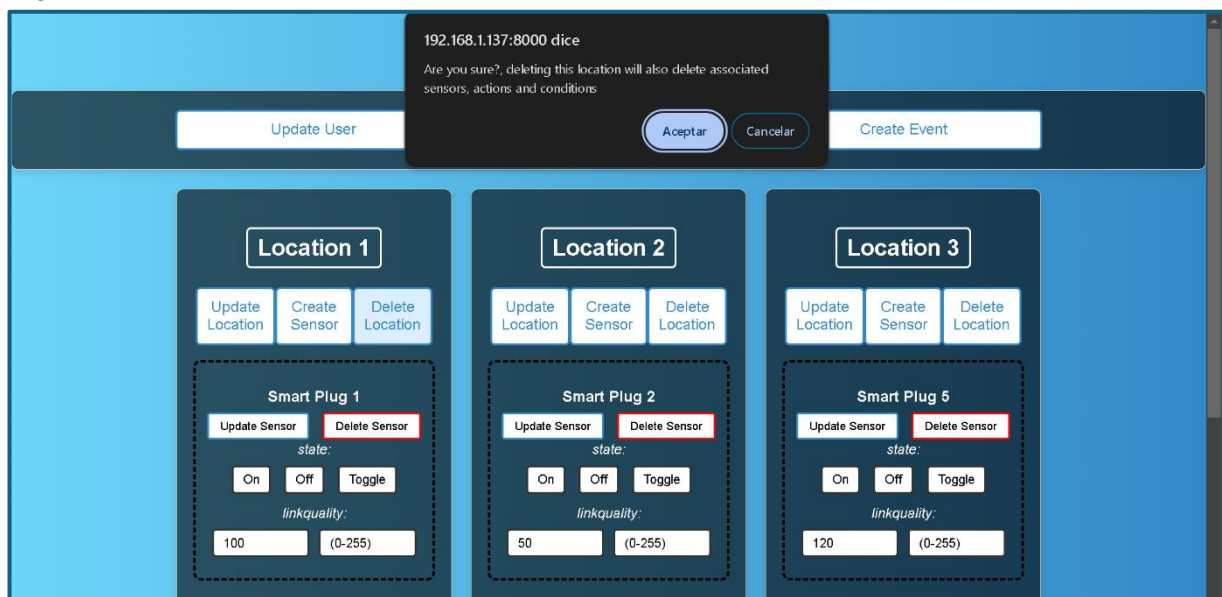


Figura 17: Mensaje de advertencia al borrar una ubicación existente

Este mensaje advierte al usuario de que, en caso de borrar una ubicación, se borrarán también todos aquellos sensores asociados a esta y los eventos que dependan de dichos sensores.

Dentro de cada ubicación, se encuentran representados los distintos sensores existentes en ella. Estos sensores también tienen una serie de campos que permiten gestionarlos. Las opciones existentes se muestran más en detalle en la Figura 18.

The screenshot displays three sensor configuration panels on a dark blue background. Each panel includes a 'linkquality' field with a value and a range (0-255). The first panel, 'Alarm 1', has an 'alarm' field with 'On' and 'Off' buttons, a 'melody' field with a value and range (1-18), a 'duration' field with a value and range (0-180), a 'volume' field with 'Low', 'Medium', and 'High' buttons, and a 'battpercentage' field with a value and range (0-100). The second panel, 'Smart Plug 3', has a 'state' field with 'On', 'Off', and 'Toggle' buttons, and a 'linkquality' field with a value and range (0-255). The third panel, 'Smart Plug 4', has a 'state' field with 'On', 'Off', and 'Toggle' buttons, and a 'linkquality' field with a value and range (0-255). Each panel also has an 'Update Sensor' button and a 'Delete Sensor' button, with the latter highlighted by a red border.

Figura 18: Campos asociados a los sensores en la página web

El botón “*Update Sensor*” redirige al usuario a un formulario web para la actualización de los datos. Este formulario se muestra en la Figura 19.

The screenshot shows a 'Sensor's Update Form' on a light blue background. The form is a dark blue box with white text and input fields. It contains five input fields: 'Nickname:', 'Serial number:', 'Vendor:', 'Model:', and 'Mac_address:'. Below these fields is a green 'Update' button. A 'Home' button is visible in the top left corner of the page.

Figura 19:Formulario de actualización de un sensor ya existente

Este formulario no permite actualizar el “*Type*” del sensor, ya que esto podría provocar un error en caso de tener eventos asociados al mismo. Además, si un sensor es registrado correctamente, el “*Type*” del mismo debería permanecer constante en el tiempo.

Si el usuario desea borrar un sensor, puede utilizar el botón “*Delete Sensor*” para ello. Al igual que con el botón de borrado de una ubicación, este botón mostrará por pantalla una advertencia indicando que, en caso de borrar un sensor, toda acción o condición asociada a

dicho sensor será también borrada. Esto se ilustra en la Figura 20.

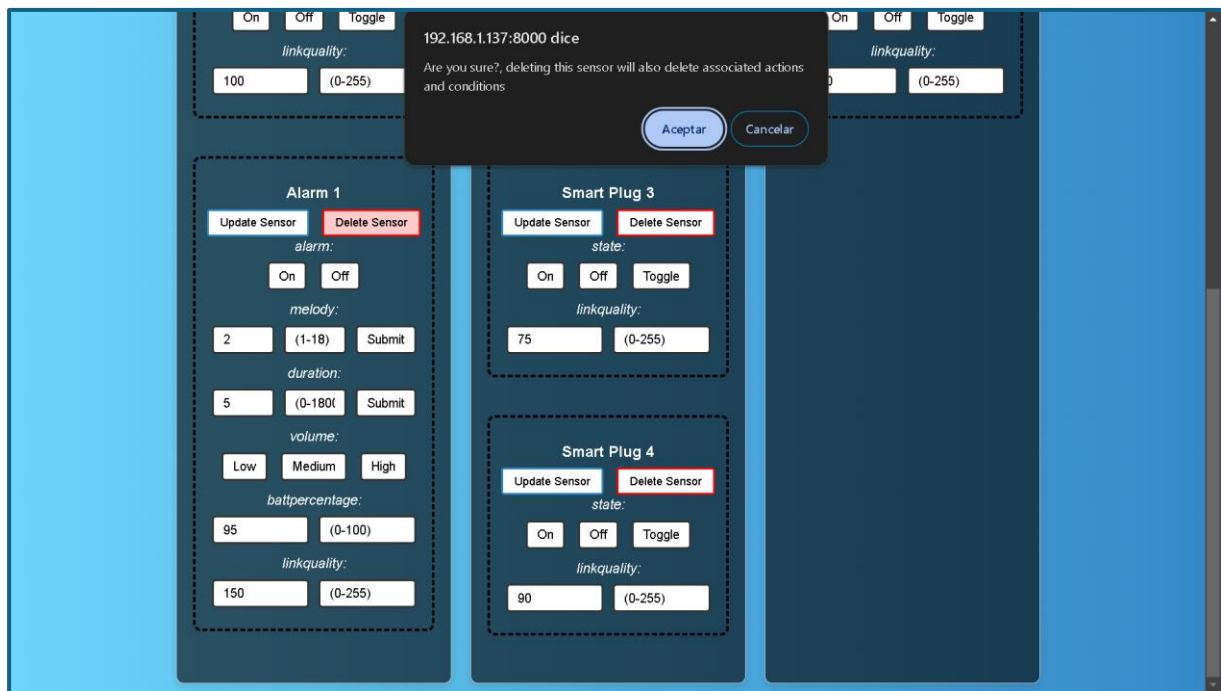


Figura 20: Mensaje de advertencia al borrar un sensor existente

En cuanto a los campos de cada sensor, estos se pueden clasificar en primer lugar en función de si son campos únicamente de lectura como por ejemplo el campo “*linkquality*” del sensor “*Alarm 1*”, que indica cómo de buena es la conexión del sensor con el *Homehub*, o campos que permiten al usuario mandar una orden a un sensor, como por ejemplo el campo “*state*” del sensor “*Smart plug 3*”, que permiten al usuario encender o apagar el enchufe.

Aquellos campos diseñados únicamente para la lectura muestran dos valores, usando como ejemplo de nuevo el mismo campo “*linkquality*”, el campo inferior de la izquierda, con valor 150, indica el último valor registrado en la base de datos de dicho campo y dicho sensor. A su derecha, se muestra el intervalo de valores posibles para dicho campo, en este caso de 0 a 255.

Los campos diseñados para enviar órdenes son de tipo booleano o enumeración, como, por ejemplo, el campo “*state*” anterior o el campo “*volume*” del sensor “*Alarm 1*”, y se componen de una serie de botones. Cada uno de ellos, al ser pulsado, manda una orden vía mensaje MQTT, en caso de ser recibida correctamente por el sensor correspondiente, cambia su valor al valor marcado por el botón.

Los campos de envío de órdenes de tipo numérico, como por ejemplo “*melody*” del sensor “*Alarm 1*”, muestran nuevamente dos campos, el de la izquierda indica el último valor registrado en la base de datos para dicho campo y sensor, y el campo de su derecha indica el intervalo de valores posibles para dicho campo.

A la derecha de estos campos, se incluye un campo de “*Submit*” que permite modificar el valor deseado para dicha propiedad. Enviando un mensaje MQTT al sensor correspondiente con el valor registrado por el usuario en el primer campo.

Una vez que el usuario ya ha creado y configurado su perfil, sus ubicaciones y sus sensores, puede pasar a personalizar sus eventos.

Para ello, el usuario debe pulsar el botón “*Create Event*”, de la Figura 12. Este le redirigirá a

la página de creación y gestión de los eventos. Esta página se muestra en la Figura 21.

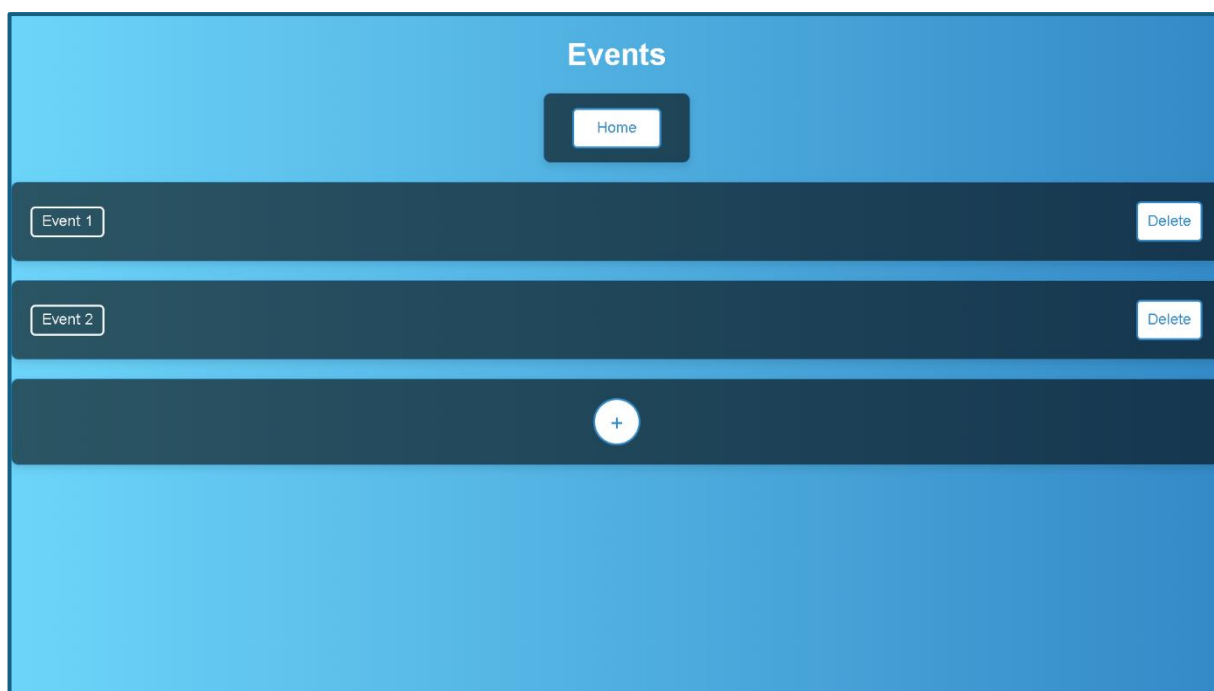


Figura 21: Página web de eventos de un usuario ficticio

Como se puede observar en la Figura 21, la página consta de un botón superior “Home” que permite al usuario volver a su página principal. Debajo de este se encuentran los eventos guardados. El usuario creado a modo de ejemplo tiene 2 eventos guardados. Estos se encuentran colapsados en una única línea para mejorar la estética de la página.

Si el usuario desea borrar alguno de sus eventos guardados, puede usar el botón “Delete” que se encuentra a la derecha de cada evento.

En caso de querer ver los datos de un evento o modificarlo, el usuario puede expandir la sección de la web dedicada a dicho evento pulsando el botón con el nombre del evento. El cambio de la página web al pulsar dicho botón se muestra en la Figura 22.

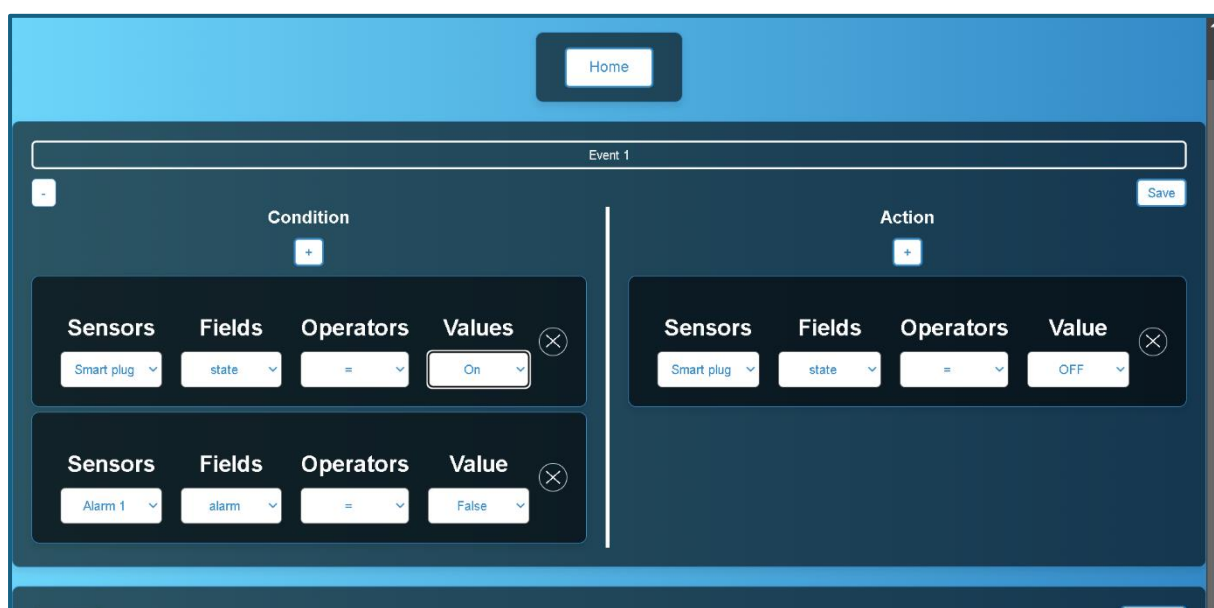


Figura 22: Página web de creación de eventos, con un evento expandido

Una vez desplegado el menú de un evento, si desea volver a colapsarlo, puede pulsar el botón “-“ para ello. Si se desea cambiar el nombre del evento, basta con clicar en su nombre y reescribirlo.

El menú desplegado muestra todas las condiciones y acciones guardadas para el evento seleccionado. Cada condición o acción consta de 4 campos:

1. **Sensors:** Este campo permite al usuario seleccionar uno de los sensores que tenga registrados dentro de sus ubicaciones. Para ello, el sistema lee los datos del usuario y crea un desplegable que permite únicamente seleccionar un sensor registrado por el usuario.
2. **Fields:** Una vez seleccionado el sensor, el sistema busca en la base de datos el tipo de sensor seleccionado. Estando determinado el tipo, crea un desplegable con los distintos campos posibles de dicho sensor.
3. **Operators:** Este campo es un desplegable de operadores lógicos. Posee los principales operadores que permiten realizar comparaciones básicas entre el valor registrado por un sensor y el valor marcado para la condición o acción.
4. **Values:** Este campo varía en función del tipo de dato del desplegable “Fields”. Si se desea actuar a través de una condición o acción en un campo de tipo booleano o de tipo enumeración, el campo “Values” creará un desplegable con la lista de opciones válidas.

En caso de actuar sobre un campo de tipo numérico, se convertirá en un campo de input indicando el intervalo de valores posibles y permitiendo al usuario introducir el valor deseado.

Para guardar cualquier cambio realizado sobre un evento, se debe pulsar el botón “Save” en la esquina superior derecha del evento.

A continuación, se adjunta una serie de figuras para mostrar el funcionamiento del sistema durante la creación de un evento junto con sus condiciones y acciones.

En primer lugar, se debe pulsar el botón “+” mostrado en la parte inferior de la Figura 21. De esta forma se crea una nueva línea para dicho evento, como se muestra en la Figura 23.

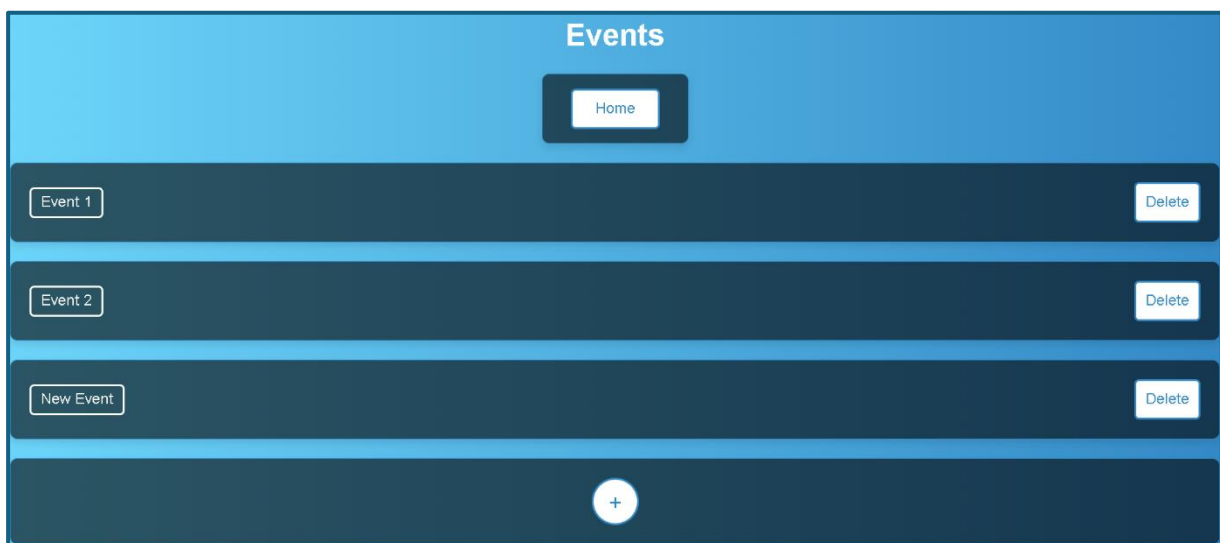


Figura 23: Página web de creación de eventos, con un nuevo evento creado

El siguiente paso será desplegar el menú del nuevo evento “*New event*” y cambiarle el nombre por otro, en este caso “*Event 3*”. Este proceso se muestra en la Figura 24.

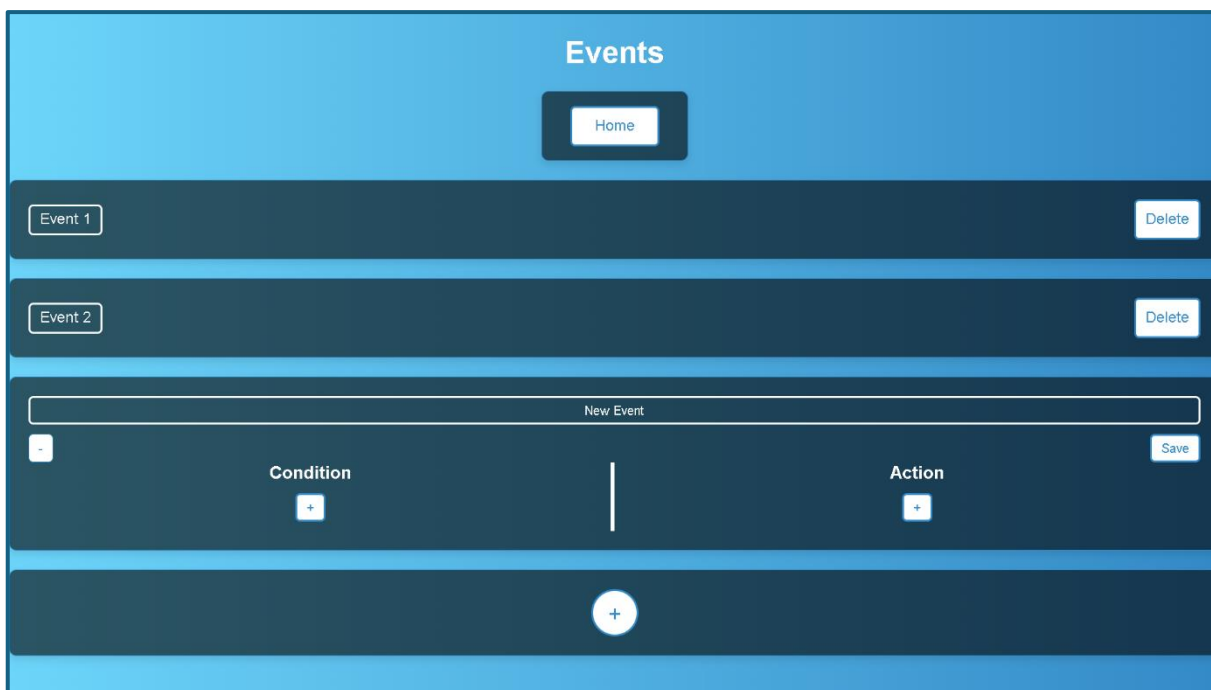


Figura 24: Página web de creación de eventos, con un nuevo evento desplegado

Después se debe añadir una condición pulsando el botón “+” situado bajo “*Condition*”, lo que desplegará un menú como el definido previamente, pero con los valores por defecto de cada campo. Esto se muestra en la Figura 25.

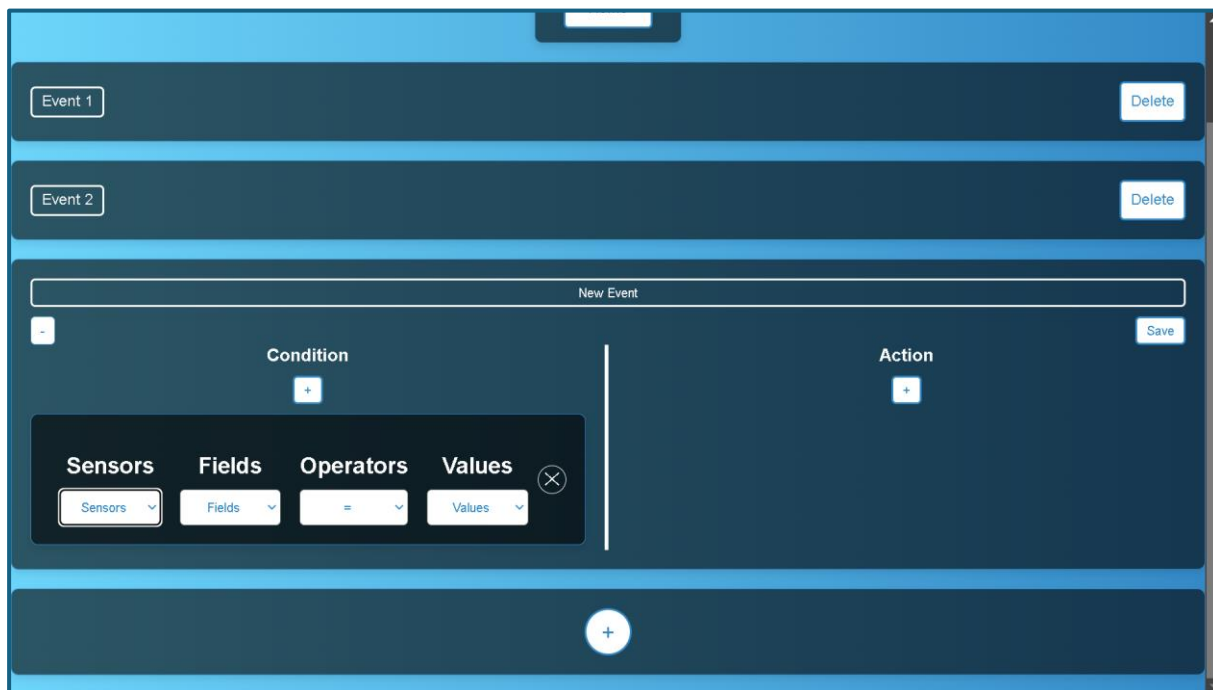


Figura 25: Página web de eventos, al desplegar el menú de una nueva condición

El siguiente paso es abrir el desplegable de sensores y elegir sobre qué sensor se creará la condición, este paso se muestra en la Figura 26. En este caso, se selecciona el sensor “*Alarm 1*”.

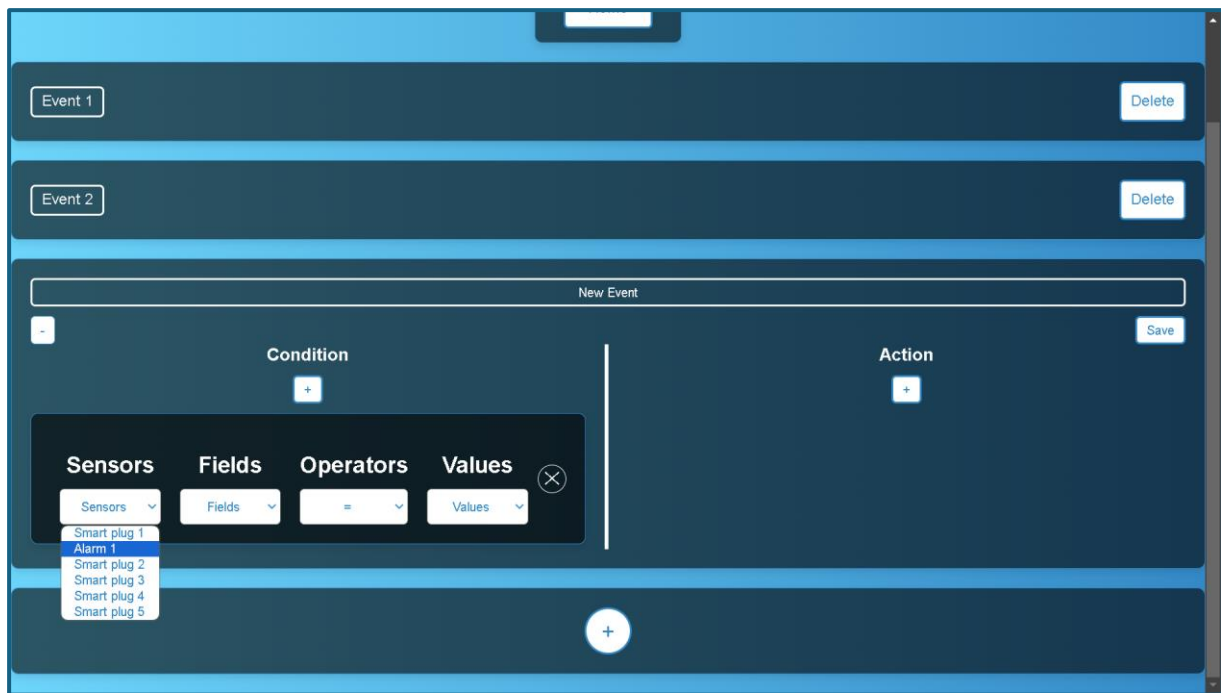


Figura 26: Página web de creación de eventos, selección del sensor

Después se selecciona el campo del sensor del cual dependerá la condición. Esto se muestra en la Figura 27. En este caso, se escoge el campo “*melody*”.

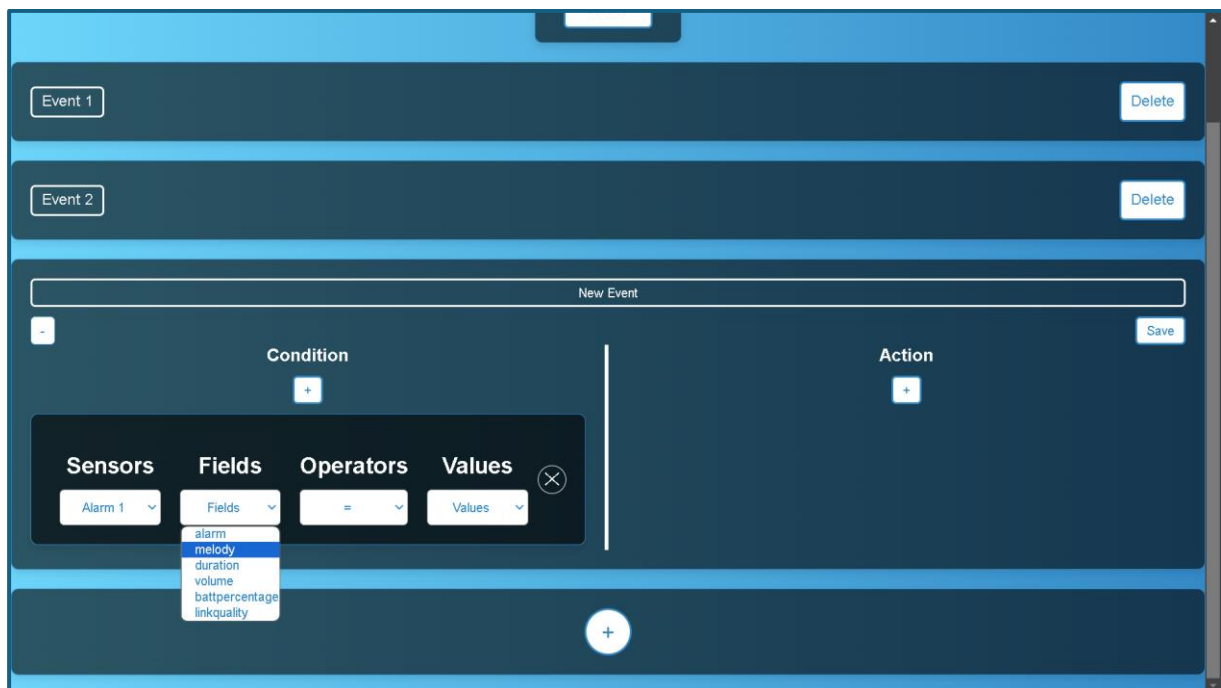


Figura 27: Página web de creación de eventos, selección del campo objetivo de la condición

Al ser “*melody*” un campo de tipo numérico, el campo “*Values*” se actualiza a un campo de input. El último paso es seleccionar el operador lógico deseado e insertar en el campo “*Values*” el valor objetivo de la condición. El campo “*Values*” muestra los valores posibles que puede introducir el usuario. Esto se muestra en la Figura 28.

Figura 28: Página de creación de eventos, selección del operador y del valor de comparación

Para completar la configuración de un nuevo evento, se repite el proceso creando una acción y renombrando el evento. Esta situación se ilustra en la Figura 29.

Figura 29: Página web de creación de eventos, creación de una acción

Finalmente, una vez terminada la configuración, el usuario debe pulsar el botón “Save” para guardar los datos del nuevo evento en la base de datos y colapsar el evento. Siendo la situación final la mostrada en la Figura 30.

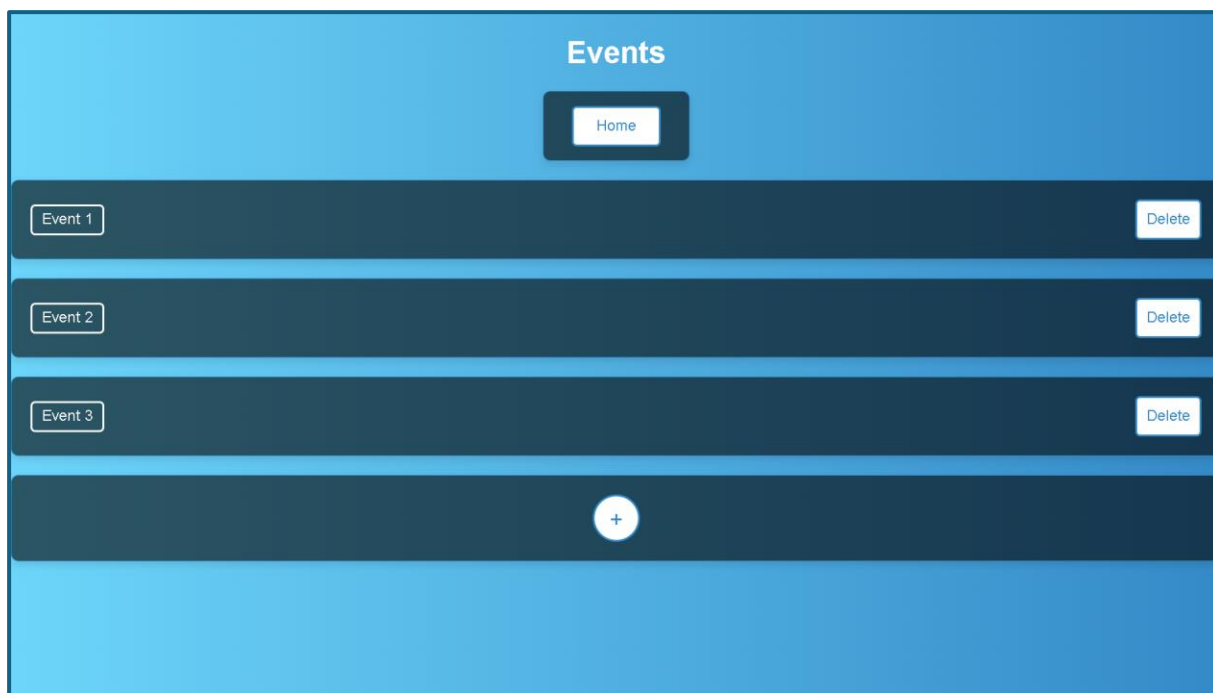


Figura 30: Página web de creación de eventos, situación final tras guardar un nuevo evento

6.5. Valoración de impactos y aspectos de responsabilidad social, legal, ética y profesional

Este proyecto se enmarca en el ámbito tecnológico. Al ser un proyecto principalmente de desarrollo *software*, puede parecer que no existen grandes impactos o responsabilidades relacionadas con el mismo.

Sin embargo, se pueden destacar varios aspectos en los que merece la pena profundizar:

En primer lugar, el **aspecto social y económico**. Este sistema busca tener un impacto social positivo al permitir a cualquier persona disponer de un sistema domótico. Esto se consigue gracias a haber definido como objetivo del sistema el minimizar el coste para el usuario final.

De esta forma, mientras que actualmente los sistemas comerciales de domótica y seguridad tienen un elevado coste de instalación y muchas veces también coste mensual de servicio elevado. Así, este sistema podría ayudar a la democratización de la tecnología y de los sistemas de domótica y seguridad.

Otro ámbito de gran importancia para el proyecto es el **ámbito legal**. El proyecto necesita conocer datos de los usuarios para su funcionamiento. Esto implica la necesidad de guardarlos de manera correcta, conforme con la **Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales**.

Actualmente, el proyecto no goza de sistemas de seguridad, por tanto, para poder cumplir con la legislación vigente se deberán desarrollar medidas de seguridad en todas las piezas del sistema para evitar cualquier posible filtración o pérdida de datos.

Por último, el sistema busca minimizar la potencia de cálculo necesaria en cada ubicación de los usuarios, concentrando las operaciones realizadas en el servidor central. De esta forma, se puede minimizar el *hardware* necesario por cada usuario. Esto podría permitir una reducción de los recursos necesarios por usuario, generando de esta forma un **impacto ambiental** positivo.

Por otro lado, este sistema se suma a otros ya desarrollados que permiten conectar cualquier dispositivo Zigbee, de esta forma se evita el utilizar múltiples consolas de control en función del fabricante de los dispositivos deseados y nuevamente se contribuye a disminuir los recursos necesarios.

En conclusión, si se desarrollan en un futuro los sistemas de seguridad necesarios que permitiesen cumplir con la legalidad vigente, el sistema podría tener un impacto positivo para la sociedad. Permitiendo el uso de sistemas domóticos por parte de cualquier usuario, independientemente de su nivel económico y disminuyendo los recursos necesarios para llevarlo a cabo.

7. VALIDACIÓN

En este capítulo se explica la validación del sistema desarrollado. Para comprobar el buen funcionamiento de cada una de las partes del sistema y de este como conjunto, se han realizado una serie de pruebas de validación durante el desarrollo del proyecto.

Estas pruebas se pueden dividir en 3 grandes grupos:

1. Pruebas unitarias
2. Demos presenciales
3. Pruebas de sistema

Las **pruebas unitarias** se han llevado a cabo durante todo el desarrollo del proyecto. Estas sirven para comprobar el buen funcionamiento de partes aisladas del sistema. Según se han ido desarrollando los distintos módulos, se han realizado pruebas de su correcto funcionamiento. Se han realizado pruebas unitarias de los siguientes módulos o partes del sistema:

1. **Homehubs**: se ha comprobado que sus contenedores funcionan y su configuración es correcta.
2. **Servicio de mensajería MQTT**: se ha comprobado el buen funcionamiento de su contenedor y de las funciones *software* que sobre él se ejecutan, verificando que es capaz de recibir mensajes MQTT y enviarlos a todos los clientes conectados a un canal determinado.
3. **Servicio de escucha**: se ha comprobado el buen funcionamiento de su contenedor y de las funciones *software* que sobre él se ejecutan, verificando que es capaz de conectarse al servicio de mensajería, subscribirse a todos los canales necesarios y recibir los mensajes MQTT enviados por los sensores.
4. **Servicio de almacenamiento de datos**: se ha comprobado el buen funcionamiento de su contenedor y de las funciones *software* que sobre él se ejecutan, verificando que es capaz de realizar consultas de escritura y lectura de datos. Las tablas del modelo de datos son correctas y el servicio permite la conexión de las demás piezas del sistema a la base de datos.
5. **Servidor de aplicaciones-web**: se han realizado pruebas del correcto funcionamiento de las distintas funciones mediante la documentación automática provista por la librería "FastAPI".
6. **Interfaz de usuario**: se ha comprobado el correcto funcionamiento de todos los códigos HTML y de las funcionalidades de sus botones, despleables, formularios, etc. También se ha comprobado la correcta personalización de la interfaz de usuario en función de los datos de cada usuario.

Estas pruebas han permitido garantizar que cada pequeña pieza del proyecto funciona correctamente por separado, de esta forma, los futuros problemas encontrados serán afrontados con la casi total seguridad de que son fruto de un problema de interconexión entre piezas y no de un error de un módulo concreto.

La **demo presencial** fue una prueba presencial llevada a cabo una vez habían sido desarrolladas las piezas básicas del proyecto que permitían comprobar que el proyecto diseñado era viable.

Fue llevada a cabo en un entorno real utilizando una red **LAN** (Local Area Network), las dos Raspberry Pi (una de ellas simulando una nube pública y la otra simulando una ubicación real de un usuario) y varios sensores. El resultado favorable de esta demo permitió pasar a la fase

de desarrollo, con la garantía de que el sistema permitía la interconexión de todas las piezas y que el futuro desarrollo no sería en vano.

Por último, las **pruebas del sistema** permitieron ir comprobando paso a paso que los distintos módulos funcionaban de manera correcta unos con otros. Dentro de estas pruebas se engloban pruebas como el correcto funcionamiento de la conexión IP entre contenedores, la comprobación del correcto funcionamiento de la base de datos a la hora de permitir la lectura de datos por parte de los otros servicios existentes, etc.

También se encuentran en este grupo las pruebas de funcionamiento del sistema real, es decir, una vez comprobado el funcionamiento de las distintas funciones con datos de entrada introducidos manualmente, se ha comprobado el funcionamiento del sistema final en una situación real de funcionamiento. Para ello se han utilizado una serie de sensores y las Raspberry Pi, y se ha comprobado el correcto funcionamiento del sistema como conjunto en una situación real durante varios días.

Debido al alcance definido para este proyecto, el entorno *hardware* de la nube pública en la que se debería de ejecutar todo el “**back-end**” del proyecto, se simula a través de una Raspberry Pi, por lo que no es posible realizar pruebas de carga realistas que permitan conocer las limitaciones del sistema. Sin embargo, al haber diseñado el proyecto para su funcionamiento en la nube (utilizando el concepto de “contenedores” y un diseño *software* orientado a “**microservicios**”), no deberían existir problemas de potencia por parte del servidor (o servidores), ya que la potencia de éste (o éstos) podría redimensionarse en función de la demanda real de los usuarios o, alternatively, aumentarse el número de contenedores requeridos para cada microservicio, forma actual en la que se redimensionan los temas de potencia de cálculo en los actuales sistemas de información.

8. CONCLUSIONES Y LÍNEAS FUTURAS

En este capítulo se va a comentar el resultado del proyecto en relación con los objetivos marcados inicialmente. También se comentarán las principales líneas futuras de desarrollo que permitirán mejorar el sistema creado.

8.1. Conclusiones

Al comienzo de esta memoria, se fijaron una serie de **objetivos técnicos** que era necesario alcanzar para poder realizar este proyecto de manera satisfactoria. El desarrollo ha sido enfocado en cumplir con dichos objetivos, lo cual se ha conseguido. Las principales conclusiones en relación con los objetivos técnicos del proyecto son:

1. Enfocar el proyecto hacia el cumplimiento de los objetivos marcados ha impulsado la introducción de diferentes mejoras que en un primer momento no estaban previstas, como por ejemplo el uso de HTMX en la interfaz de usuario. Estas han permitido garantizar el cumplimiento de dichos objetivos. De esta forma se ha superado la idea inicial de la que surgió este proyecto.
2. El resultado obtenido tras el desarrollo cumple satisfactoriamente el objetivo principal del proyecto. Se ha creado un sistema domótico, de fácil uso para el usuario final, que permite la gestión de sensores domóticos distribuidos en múltiples ubicaciones y con un alto grado de personalización.
3. Además, a pesar de no contar aún con pruebas de carga (tal como se explicó en secciones previas, no se usa la nube pública, sino que se simula a través de una Raspberry Pi, no siendo posible realizar pruebas del sistema final real actualmente), por su diseño a través de contenedores, se puede afirmar que, en caso de alcanzarse un uso masivo del sistema, el sistema no debería tener problemas de funcionamiento.
4. También es importante destacar que la arquitectura y el diseño del sistema se han orientado a minimizar todo lo posible el coste para el usuario final. Objetivo que también se ha conseguido, ya que el coste del sistema domótico para un hipotético usuario final se reduce al coste de los sensores que desee cada usuario y a la compra de un *Homehub*, con unos requisitos computacionales muy bajos.

Por otro lado, a los objetivos técnicos definidos inicialmente, se les puede añadir también una serie de **objetivos formativos** por parte del autor del texto.

Este proyecto le ha permitido desarrollar una serie de competencias relacionadas con el mundo de la programación y el desarrollo *software*.

Para una persona con poca experiencia en este campo como el autor, el desarrollo de este proyecto le ha permitido comprender el funcionamiento de las aplicaciones en el contexto actual, desde las piezas más pequeñas, en este caso sensores, hasta el extremo contrario con la interfaz de usuario, pasando por todo el núcleo del sistema al que se ha denominado “gestor domótico”.

Durante el desarrollo, el autor ha podido aprender Python, el principal lenguaje de programación actual, HTML, Docker, SQL y múltiples librerías, herramientas de gran utilidad para un futuro.

También ha podido aprender sobre los procesos de desarrollo iniciales del *software* a través del análisis de requisitos funcionales y el diseño y desarrollo.

De esta forma, para el autor del texto este proyecto ha servido de introducción al mundo de la ingeniería software.

Una vez terminado el proyecto, y a pesar del resultado satisfactorio, se plantean numerosas mejoras que podrían implementarse, desde un desarrollo más avanzado de la seguridad en el sistema, hasta el desarrollo de una app.

En el siguiente apartado se desarrollan las posibles mejoras que se podrían desarrollar.

8.2. Líneas futuras

Debido a que la complejidad del desarrollo del proyecto fue mayor de la esperada en un primer momento, existen múltiples líneas de desarrollo que quedan pendientes para su futura realización.

1. Funcionalidades fuera de alcance

En primer lugar, algunos de los requisitos funcionales deseados, quedaron fuera del alcance de proyecto.

Entre ellos están los requisitos funcionales relacionados con el rol de gestor o administrador del sistema. Cómo pueden ser la creación de informes y la capacidad de realizar análisis de los datos del conjunto de usuarios.

También queda pendiente el desarrollo de las funcionalidades relacionadas con la gestión del servidor MQTT por parte del administrador.

Se ha preferido dar prioridad en el desarrollo a la parte destinada al usuario final, alcanzando de esta forma todas las funcionalidades relacionadas con el usuario para probar si el sistema genera interés y es de utilidad.

2. Implantación de sistemas de ciberseguridad

Este apartado es posiblemente el de mayor complejidad para su desarrollo futuro. La ciberseguridad es uno de los temas de moda hoy en día y la implantación de sistemas de seguridad fiables en un proyecto con múltiples elementos puede suponer un gran reto.

Se han analizado los principales puntos débiles del proyecto en relación con la ciberseguridad, dejando para un futuro desarrollo los siguientes puntos:

1. Configuración avanzada de los sistemas de seguridad de “Zigbee2MQTT” que garantice la imposibilidad de conexiones no deseadas al sistema “Zigbee2MQTT”. Es decir, garantizar que un hipotético “*hacker*” no puede utilizar “sensores falsos” para conectarse y manipular el sistema de gestión domótica.
2. Configuración avanzada de los sistemas de seguridad del servidor MQTT que garantice que ningún usuario puede acceder a los canales de otros usuarios, ni un agente externo puede acceder a los canales del servidor MQTT. Es decir, garantizar que el servidor MQTT solo trabajará con mensajes MQTT cuyo origen o destino sean los “*Homehubs*” validados en el sistema domótico, evitando que un posible intruso pueda manipular el sistema a través de mensajes MQTT no autorizados.
3. Codificación de los datos sensibles de los usuarios a la hora de almacenarlos, evitando su revelación en caso de una posible filtración de los datos guardados en la base de datos central.

4. Creación de un sistema de regeneración de la base de datos que permita la recuperación de los datos perdidos en caso de un fallo grave del sistema. Esto se podría realizar bien a través de un sistema de gestión de la base de datos que se ejecute en modalidad “alta disponibilidad” (en un entorno redundante) o bien a través de un entorno de “*back up*” que se utilizaría ante una emergencia o situación de desastre. Ambos entornos son compatibles entre sí y también podrían coexistir.
5. Creación de un sistema de verificación de usuario mediante certificados digitales y tokens que permitan garantizar que el usuario que utiliza la aplicación web es quien dice ser en todo momento.
6. Rediseñar la aplicación web para el envío de los datos imprescindibles en cada petición web, evitando de esta forma en la medida de lo posible el envío repetido de información sensible.

3. Desarrollo de nuevas funcionalidades

También se plantea para un futuro desarrollo, la creación de funcionalidades no previstas en el análisis del proyecto. Entre ellas destaca la creación de un sistema de avisos, externo al propio sistema, vía email, llamada de teléfono y/o SMS o WhatsApp, que permita al usuario definir bajo qué situaciones quiere ser notificado de alguna incidencia de manera urgente.

4. Creación de una app móvil

Otro importante desarrollo futuro de gran utilidad es la creación de una app móvil. Con ella no solo el usuario podrá recibir notificaciones en todo momento, sino que además se facilitará enormemente el uso del sistema en cualquier lugar.

Mediante la aplicación también se intentará facilitar la rápida implantación de un nuevo sistema por parte del usuario, permitiendo realizar la configuración inicial del sistema a través de la aplicación móvil.

5. Extensión del sistema a otros protocolos

Una vez desarrollado el sistema completo, y teniendo en cuenta que uno de los objetivos del proyecto era crear un sistema que permitiese conectar el mayor número de sensores compatibles. Parece lógico buscar formas de incluir otros protocolos ampliamente utilizados, como puede ser Z-Wave, en el sistema. Al estar ya diseñado el sistema completo, la labor necesaria para ello consistiría en encontrar una forma de convertir los mensajes recibidos en protocolos como Z-Wave a protocolo MQTT, de esta forma el sistema actual ya sería capaz de gestionarlos.

6. Realización de análisis de carga y gestión de la escalabilidad

Por último, se pretende realizar un análisis de carga que permita ver cuáles son las limitaciones reales del sistema funcionando en la nube, y a partir de este análisis estudiar cuál es la mejor estrategia de escalabilidad que permita explotar al máximo la arquitectura en contenedores del sistema.

9. BIBLIOGRAFÍA

- [1] M. Mansour y otros, «Internet of Things: A Comprehensive Overview on Protocols, Architectures, Technologies, Simulation Tools, and Future Directions,» *Energies*, p. 39, 14 Abril 2023.
- [2] E. L. Sánchez, «Diseño de un sistema de control domótico basado en la plataforma Arduino,» Escola Tècnica Superior d'Enginyeria Informàtica, Universitat Politècnica de València , Valencia, 2012.
- [3] M. B. Durango y otros, «Análisis y diseño de un prototipo de sistema domótico de bajo costo,» *Revista Facultad de Ingeniería Universidad de Antioquia*, n° 63, pp. 117-128, Junio 2012.
- [4] D. Fenocchio y otros, «DomoHome: un sistema domótico inteligente,» de *Libro de actas - XXVIII Congreso Argentino de Ciencias de la Computación - CACIC 2022*, La Rioja, 2022.
- [5] E. M. Thabet, «SISTEMA DOMÓTICO A ESCALA CON ACCESO REMOTO,» Escuela Politécnica Superior de Linares, Ingeniería de Telecomunicación, Linares, 2024.
- [6] K. Kanters, «Zigbee2MQTT,» [En línea]. Available: <https://www.zigbee2mqtt.io/>. [Último acceso: 02 09 2024].
- [7] P. Dhillon y Dr H. Sadawarti, «A Review Paper on Zigbee (IEEE 802.15.4),» *International Journal of Engineering Research & Technology (IJERT)*, vol. 3, n° 4, pp. 141-145, Abril 2014.
- [8] Oasis, «mqtt.org,» [En línea]. Available: <https://mqtt.org/mqtt-specification/>. [Último acceso: 7 8 2024].
- [9] Eclipse, «mosquitto.org,» [En línea]. Available: <https://mosquitto.org/>. [Último acceso: 6 08 2024].
- [10] P. Fersing y R. Light, «pypi.org/project/paho-mqtt/,» [En línea]. Available: <https://pypi.org/project/paho-mqtt/>. [Último acceso: 10 8 2024].
- [11] Python Software Foundation, «python.org,» [En línea]. Available: <https://www.python.org/>. [Último acceso: 02 09 2024].
- [12] The PostgreSQL Global Development Group, «postgresql.org,» [En línea]. Available: <https://www.postgresql.org/>. [Último acceso: 02 09 2024].
- [13] Web Hypertext Application Technology Working Group, «html.spec.whatwg.org,» [En línea]. Available: <https://html.spec.whatwg.org/>. [Último acceso: 02 09 2024].
- [14] Docker, «docker.com,» [En línea]. Available: <https://www.docker.com/resources/what-container/>. [Último acceso: 25 Julio 2024].
- [15] The Raspberry Pi Foundation, «raspberrypi.com,» [En línea]. Available: <https://www.raspberrypi.com/software/>. [Último acceso: 02 09 2024].
- [16] S. R. Montaña, «sqlmodel.tiangolo.com,» [En línea]. Available: <https://sqlmodel.tiangolo.com/>. [Último acceso: 9 8 2024].
- [17] DBeaver, Community, «dbeaver.io,» [En línea]. Available: <https://dbeaver.io/>. [Último acceso: 5 8 2024].
- [18] Uvicorn, «uvicorn.org,» [En línea]. Available: <https://www.uvicorn.org/>. [Último acceso: 7 8 2024].
- [19] S. R. Montaña, «fastapi.tiangolo.com,» [En línea]. Available: <https://fastapi.tiangolo.com/>. [Último acceso: 9 8 2024].
- [20] S. R. Montaña, «fastapi.tiangolo.com/,» [En línea]. Available: <https://fastapi.tiangolo.com/#performance>. [Último acceso: 9 8 2024].

- [21] organization, The Pallets, «jinja.palletsprojects.com,» [En línea]. Available: <https://jinja.palletsprojects.com/en/3.1.x/>. [Último acceso: 10 8 2024].
- [22] C. Gross, «htmx.org/,» [En línea]. Available: <https://htmx.org/>. [Último acceso: 9 8 2024].
- [23] talent, «[talent.com](https://es.talent.com/),» [En línea]. Available: <https://es.talent.com/salary?job=programador+.net>. [Último acceso: 18 7 2024].
- [24] talent, «[talent.com](https://es.talent.com/)/,» [En línea]. Available: <https://es.talent.com/salary?job=ingeniero+junior#:~:text=%C2%BFCu%C3%A1nto%20gana%20un%20Ingeniero%20junior%20en%20Espa%C3%B1a%3F&text=El%20salario%20ingeniero%20junior%20promedio,hasta%20%E2%82%AC%2031.895%20al%20a%C3%B1o..> [Último acceso: 30 07 2024].
- [25] Agencia Estatal Boletín Oficial del Estado, «Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales,» 2018. [En línea]. Available: <https://www.boe.es/eli/es/lo/2018/12/05/3>. [Último acceso: 04 09 2024].
- [26] AENOR, «EA 0026:2006,» 2006. [En línea]. Available: <https://tienda.aenor.com/norma-une-ea-0026-2006-n0037689>. [Último acceso: 04 09 2024].
- [27] Agencia Estatal Boletín Oficial del Estado, «[boe.es](https://www.boe.es/),» 18 09 2002. [En línea]. Available: <https://www.boe.es/eli/es/rd/2002/08/02/842>. [Último acceso: 04 09 2024].
- [28] UNE normalizacion española, «[une.org](https://www.une.org/),» 2020. [En línea]. Available: <https://www.une.org/encuentra-tu-norma/busca-tu-norma/norma?c=N0063753>. [Último acceso: 04 09 2024].
- [29] Real Academia Española, «dle.rae.es,» [En línea]. Available: <https://dle.rae.es/>. [Último acceso: 7 8 2024].

10. PLANIFICACIÓN TEMPORAL

En este capítulo se va a describir la planificación temporal del proyecto. Para realizarlo de manera más visual se utilizarán una **EDP** y un **diagrama de Gantt**.

En un proyecto con gran cantidad de pequeños módulos como este, una buena planificación temporal es esencial para poder cumplir todos los objetivos. Además, con ella se establece un cierto orden para el desarrollo de cada módulo, disminuyendo el tiempo y esfuerzo necesarios.

En primer lugar, la Figura 31 muestra la EDP o **Estructura de Descomposición del Proyecto**. En ella se pueden apreciar las distintas fases que lo componen. En color verde se muestran las fases de alto nivel, y en color morado las subfases del proyecto de más bajo nivel. Por último, los recuadros en color naranja hacen referencia a los distintos paquetes de trabajo que deben realizarse para alcanzar los objetivos planificados.

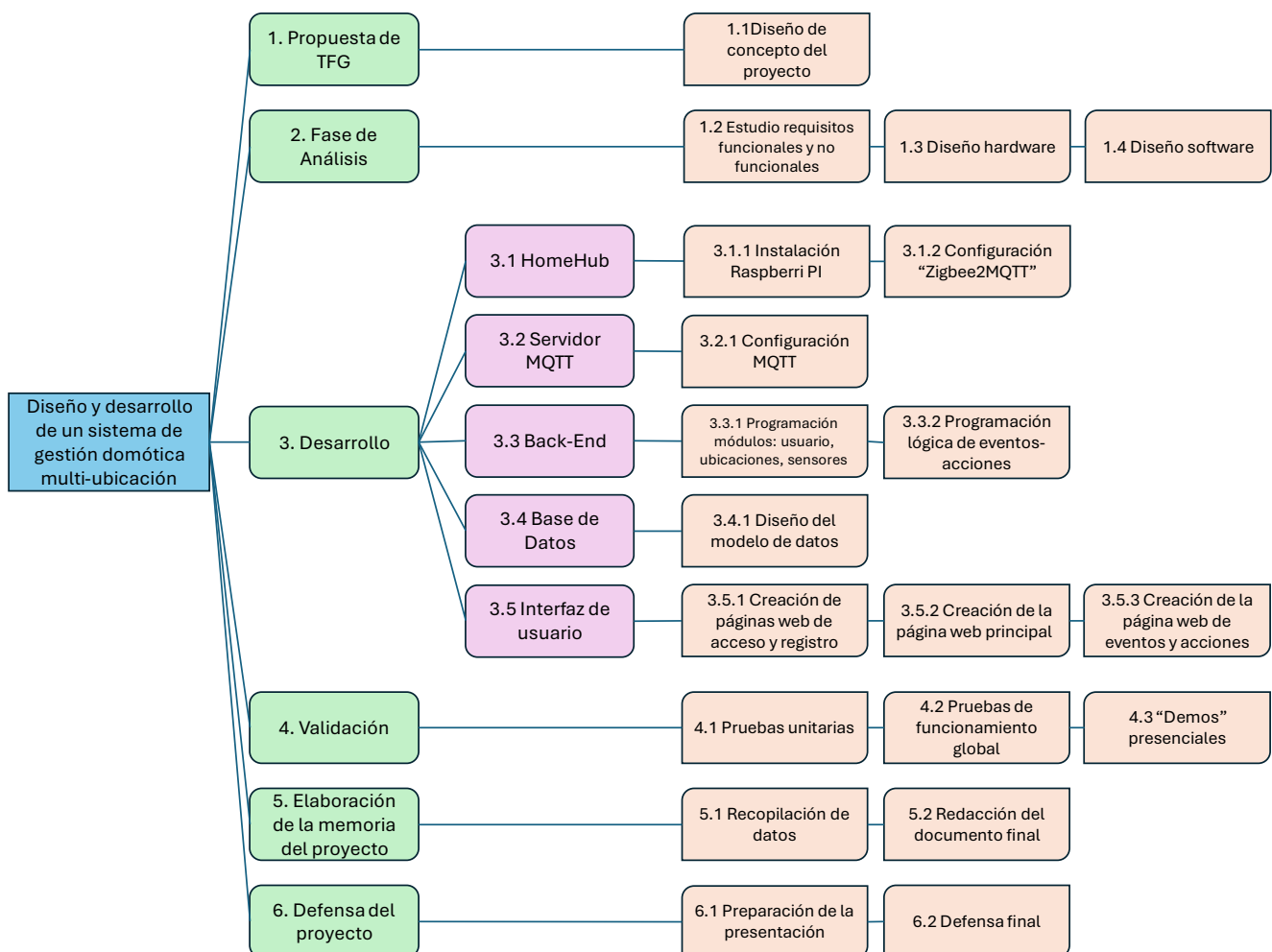


Figura 31: Estructura de Descomposición del Proyecto

PLANIFICACIÓN TEMPORAL

A continuación, se presenta una tabla con información adicional sobre cada uno de los paquetes incluidos en la EDP. La Tabla 29 muestra tanto el inicio como el final aproximado de cada paquete y las horas aproximadas necesarias para la finalización de cada apartado. Mantiene el código de colores asociado a cada paquete de la EDP para mejorar la comprensión de la misma.

Tabla 29: Información sobre los componentes de la EDP

Nombre Actividad	Fecha Inicio	Duración (días)	Fecha Fin	Horas estimadas
Propuesta de TFG	9-nov.	7	16-nov.	12,0
Diseño de concepto del proyecto	9-nov.	7	16-nov.	12,0
Fase de Análisis	17-nov.	28	15-dic.	60,0
Análisis de requisitos	17-nov.	14	1-dic.	30,0
Diseño <i>hardware</i> y <i>software</i>	1-dic.	14	15-dic.	30,0
Periodo de exámenes	15-dic.	72	25-feb.	
Desarrollo	25-feb.	89	24-may.	336,0
Desarrollo <i>HomeHub</i>	25-feb.	14	10-mar.	40,0
Instalación Raspberry Pi	25-feb.	5	1-mar.	15,0
Configuración "Zigbee2MQTT"	1-mar.	9	10-mar.	25,0
Desarrollo MQTT	11-mar.	14	25-mar.	20,0
Configuración MQTT	11-mar.	14	25-mar.	20,0
Desarrollo Base de Datos	25-mar.	29	23-abr.	50,0
Diseño del modelo de datos	25-mar.	29	23-abr.	50,0
Desarrollo Back-End	25-mar.	21	15-abr.	105,0
Programación Módulos	25-mar.	14	8-abr.	70,0
Programación eventos y acciones	8-abr.	7	15-abr.	35,0
Desarrollo Interfaz de Usuario	16-abr.	38	24-may.	121,0
Páginas de acceso y registro	16-abr.	8	24-abr.	16,0
Página principal	24-abr.	15	9-may.	45,0
Página eventos y acciones	9-may.	15	24-may.	60,0
Validación	25-feb.	92	27-may.	42,0
Pruebas unitarias	25-feb.	89	24-may.	30,0
Prueba funcionamiento global	24-may.	3	27-may.	10,0
"Demos" presenciales	25-mar.	1	26-mar.	2,0
Periodo de exámenes	27-may.	49	15-jul.	
Elaboración de la memoria	15-jul.	16	31-jul.	53,0
Recopilación de datos	15-jul.	2	17-jul.	8,0
Redacción de la memoria	17-jul.	14	31-jul.	45,0
Periodo de descanso	1-ago.	45	15-sep.	
Defensa de la memoria	15-sep.	6	21-sep.	11,0
Preparación de la Presentación	15-sep.	5	20-sep.	10,0
Defensa final	20-sep.	1	21-sep.	1,0
Total	9-nov.	317	21-sep.	514,0

Se calcula que en total han sido necesarias unas 500 horas aproximadamente. Habiendo sido destinadas la mayor parte de estas al apartado de desarrollo, el núcleo central de este proyecto, y más concretamente a la programación del Back-End y la interfaz de usuario.

Una vez establecidas las fechas de inicio y finalización de cada paquete, la Figura 32 presenta el diagrama de Gantt que permite apreciar visualmente el desarrollo del proyecto, aunque se debe tener en cuenta la distinta distribución de horas por día en cada fase del proyecto. El diagrama se presenta de forma vertical para una mejor visualización.



Figura 32: Diagrama de Gantt del proyecto

11. PRESUPUESTO

En este apartado se realiza un análisis del presupuesto requerido para el desarrollo del proyecto. Se tienen en cuenta tanto los gastos asociados al material necesario, como los gastos que supondría pagar a los trabajadores que llevasen a cabo este proyecto.

11.1. Presupuesto del material

Para la realización de este proyecto, han sido necesarias una gran cantidad de piezas, principalmente sensores y componentes, que permitiesen probar las distintas funcionalidades y hacer una prueba real del sistema.

La Tabla 30 muestra el material necesario para la prueba de un *Homehub*:

Tabla 30: Presupuesto del Homehub

Elemento	Precio
Raspberry Pi 4 Model B	70,99 €
Pack LEICKE Raspberry Pi 4 Modelo B + Fuente de alimentación	14,99 €
Cable Rankie micro-HDMI to HDMI	5,99 €
SONOFF Universal Zigbee 3.0 USB Dongle Plus	29,99 €
2 x Tarjeta Netac 32 GB Tarjeta Micro SD	14,99 €

La Tabla 31 muestra el material necesario para la prueba de un servidor en la “nube”:

Tabla 31: Presupuesto del servidor en la nube

Elemento	Precio
Raspberry Pi 4 Model B 8 GB	92,61 €
Pack LEICKE Raspberry Pi 4 Modelo B + Fuente de alimentación	14,99 €
Cable Rankie micro-HDMI to HDMI	5,99 €
2 x Tarjeta Netac 32 GB Tarjeta Micro SD	14,99 €

La Tabla 32 muestra los sensores comprados para realizar pruebas:

Tabla 32: Presupuesto de los sensores

Elemento	Precio
Zigbee 3.0 Enchufe Inteligente, SONOFF S26R2ZBTPF	21,99 €
Sonoff SNZB-04 Sensor de Puerta	11,88 €
Sonoff SNZB-03 Sensor de movimiento	13,29 €
Sensor de movimiento Marmitek SM0202_1	42,99 €
Zigbee Interruptor Inalámbrico SONOFF SNZB-01	18,99 €
SONOFF Zigbee Wireless Switch SNZB-01P	19,50 €
SONOFF ZigBee Sensor De Temperatura y Humedad SNZB-02P	12,63 €
Alarma Neo NAS-AB02B2	8,50 €

Por último, la Tabla 33 muestra los materiales necesarios para las pruebas de concepto y la interconexión de las distintas partes:

Tabla 33: Presupuesto de interconexión

Elemento	Precio
TP-Link TL-SF1005D 5-Port 10/100 Mbps Desktop Ethernet Switch	13,89 €
Cable Matters Paquete de 5, 10G Cable Ethernet Cat 6	9,99 €

El presupuesto total material del proyecto asciende a **439,08 €**.

11.2. Presupuesto del coste personal

El presupuesto relacionado con el personal del proyecto se puede calcular a partir de los datos de la planificación temporal. Se debe tomar como dato el número de horas totales empleadas en el desarrollo del proyecto, teniendo en cuenta que el autor, a la hora de arrancar el proyecto, era una persona con una experiencia entre moderada y escasa en todo lo relativo a los temas de programación, utilización de gestores de bases de datos, utilización de servidores Web y servidores de aplicaciones, conocimiento del protocolo MQTT y del Servidor o Broker de dicho protocolo, etc., y que, por lo tanto, ha debido autoformarse mientras elaboraba el proyecto.

Por lo tanto, para el cálculo del coste de personal se utiliza como salario el de un ingeniero junior, teniendo en cuenta solo las horas dedicadas al desarrollo del proyecto como tal, no incluyendo aquellas destinadas a formación. También se incluyen los costes relacionados con la dirección del proyecto, realizando una estimación de las horas dedicadas y usando como salario, el de un ingeniero senior. Los datos relativos a estos costes de personal se presentan en la Tabla 34:

Tabla 34: Presupuesto coste de personal

Categoría del personal	Nº de personas	Horas	€/h	Coste Total €
Ingeniero Junior	1	500 aprox	14,62 [23]	7310
Ingeniero Senior	2	30	19,23 [24]	1153,8
Total				8463,8

Por tanto, el coste total del desarrollo del proyecto sería **8902,88 €**.

12. EVALUACIÓN DE IMPACTOS

La implantación en los hogares de un sistema como el desarrollado en este proyecto, tiene el potencial de generar impactos positivos en numerosos ámbitos. Entre ellos, destacan el social, el económico y el medioambiental.

1. Impacto medioambiental: El uso de sistemas domóticos inteligentes, como el diseñado durante este proyecto, puede, a través de la automatización y el control preciso de los distintos dispositivos, mejorar la eficiencia energética de los hogares. De esta forma, la energía consumida por cada hogar se vería disminuida, contribuyendo a la reducción de la huella de carbono de cada hogar. También puede ayudar a disminuir el gasto de otros recursos como el agua o el gas natural, mejorando la eficiencia de sistemas de riego o calefacción. Teniendo de esta forma un impacto medioambiental positivo.
2. Impacto económico: en el ámbito económico, dicha reducción energética, ayuda a disminuir los costes energéticos mensuales de cada usuario.

Al mismo tiempo, este proyecto, en comparación con otras alternativas, consigue la reducción de los costes tanto de compra inicial de los dispositivos necesarios como de funcionamiento y gestión del gestor domótico. De esta forma, se cumple el objetivo inicial del proyecto que buscaba minimizar el coste por usuario final y se tiene un impacto económico muy positivo para los usuarios.

Además, el aumento de la demanda de tecnologías domóticas podría incentivar el desarrollo de la industria y la creación de empleos, teniendo un impacto positivo en la economía.

3. Impacto social: respecto al impacto social, además del posible impacto a través de la creación de nuevos empleos, la implantación de este sistema domótico puede mejorar la calidad de vida de las personas, permitiendo a los usuarios llevar a cabo las tareas del día a día de forma más sencilla y saber en todo momento el estado de sus ubicaciones. Además, la reducción de la contaminación y la huella de carbono también tienen un impacto positivo en la salud de la población.

13. ANÁLISIS DE ASPECTOS LEGALES Y ÉTICOS

Respecto al ámbito legal, los sistemas domóticos responden ante una serie de normativas. Al ser la domótica un área relativamente reciente, aún se están desarrollando normativas futuras que puedan crear un marco legislativo adecuado para su regulación.

Como ya se comentó previamente, una de las normativas que debe ser cumplida es la **Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales** [25], esta ley regula el almacenamiento y gestión de los datos de los usuarios por parte del gestor domótico.

Respecto a la legislación relativa con los dispositivos domóticos en sí, existen una serie de normativas a tener en cuenta:

1. Norma EA0026 de AENOR [26]: aprobada en septiembre de 2013, se ha convertido en un documento de referencia en este ámbito. Indica los requisitos mínimos a cumplir por cualquier instalación domótica.
2. Reglamento Electrotécnico de Baja Tensión (Real Decreto 842/2002) [27]: Específicamente, la ITC-51 se encarga de regular los sistemas de automatización y seguridad en las viviendas. Busca asegurar la correcta instalación y funcionamiento de los sistemas domóticos.
3. UNE-EN 50090-5-1:2020 [28]: es un estándar europeo que se enfoca en la comunicación abierta entre distintos dispositivos electrónicos.

El cumplimiento de estas normas es necesario para poder desarrollar un sistema domótico como el diseñado hasta un nivel comercial. De esta forma podría comercializarse a gran escala cumpliendo con la normativa vigente

14. CONTRIBUCIÓN A LOS OBJETIVOS DE DESARROLLO SOSTENIBLE

Los sistemas domóticos integran nuevas tecnologías que permiten optimizar la gestión de los hogares, y pueden tener un papel fundamental para alcanzar varios de los Objetivos de Desarrollo Sostenible.

Estos sistemas no solo mejoran la calidad de vida de la sociedad, sino que también pueden tener un fuerte impacto en otras áreas como el medioambiente y la economía. Por ello, están relacionados con una serie de ODS, entre ellos los más importantes son:

1. ODS 6: AGUA LIMPIA Y SANEAMIENTO

Aunque no todos los sistemas domóticos destacan por tener un impacto notable en este objetivo, el uso de sistemas automáticos en la gestión de recursos hídricos en los hogares puede tener un alto impacto. No solo pueden ayudar a reducir el consumo de agua en tareas básicas como el riego, sino que también pueden ayudar a detectar posibles fugas de agua y alertar a los usuarios. Contribuyendo a la conservación del agua y disminuyendo los costes asociados para el usuario

2. ODS 7: ENERGÍA ASEQUIBLE Y NO CONTAMINANTE

Los sistemas domóticos están diseñados para controlar permanentemente la evolución de los hogares. De esta forma se pueden minimizar los consumos energéticos de iluminación, climatización y los distintos dispositivos conectados. Esto ayuda a disminuir el consumo energético por parte de los usuarios y contribuye directamente al ODS 7.

3. ODS 9: INDUSTRIA, INNOVACIÓN E INFRAESTRUCTURA

El desarrollo de este proyecto y de otros sistemas domóticos contribuye a fomentar la innovación y desarrollo de nuevas tecnologías y ayuda en la modernización de la infraestructura doméstica. La domotización contribuye al desarrollo e integración de múltiples tecnologías emergentes como las relacionadas con el IoT, la inteligencia artificial... Esto contribuye al desarrollo de la industria tecnológica y la creación de nuevas infraestructuras más sostenibles.

4. ODS 11: CIUDADES Y COMUNIDADES SOSTENIBLES

Otro de los ODS más directamente relacionados con este proyecto es el ODS 11. La implementación de las nuevas tecnologías del IoT y las mejoras de los protocolos de comunicación son un paso crucial hacia la creación de ciudades inteligentes y sostenibles. Los hogares domotizados contribuyen a aumentar la eficiencia de las ciudades y su seguridad, mejorar la calidad de vida y el medioambiente.

5. ODS 13: ACCIÓN POR EL CLIMA

La reducción del consumo energético y de las emisiones contaminantes obtenida mediante la implementación de sistemas domóticos puede suponer una gran ayuda para alcanzar el ODS 13. La capacidad de estos sistemas para minimizar el consumo de los hogares puede ser un factor significativo para reducir la huella de carbono de las grandes ciudades.

16. ÍNDICE DE FIGURAS

FIGURA 1: DIAGRAMA DE CONTEXTO DE ALTO NIVEL	13
FIGURA 2: DIAGRAMA DE BAJO NIVEL DE UNA UBICACIÓN REMOTA	14
FIGURA 3: DIAGRAMA DE BAJO NIVEL DEL GESTOR DOMÓTICO	15
FIGURA 4: DIAGRAMA DE BAJO NIVEL DEL SERVIDOR DE APLICACIONES-WEB.....	18
FIGURA 5: MODELO DE DATOS	28
FIGURA 6: SISTEMA DOMÓTICO OBTENIDO	35
FIGURA 7: PÁGINA WEB DE ACCESO	43
FIGURA 8: FORMULARIO DE REGISTRO DE NUEVOS USUARIOS	44
FIGURA 9: PÁGINA WEB DE ACCESO EN CASO DE NO ENCONTRAR AL USUARIO	45
FIGURA 10: PÁGINA WEB DE ACCESO EN CASO DE CREDENCIALES INCORRECTAS	45
FIGURA 11: PÁGINA WEB PRINCIPAL DE UN USUARIO NUEVO.....	46
FIGURA 12: PÁGINA WEB PRINCIPAL DE UN USUARIO FICTICIO	46
FIGURA 13: FORMULARIO WEB DE ACTUALIZACIÓN DE LOS DATOS DE USUARIO	47
FIGURA 14: FORMULARIO WEB DE CREACIÓN DE UNA NUEVA UBICACIÓN	47
FIGURA 15: FORMULARIO WEB DE ACTUALIZACIÓN DE UNA UBICACIÓN YA EXISTENTE	48
FIGURA 16: FORMULARIO WEB DE CREACIÓN DE NUEVOS SENSORES	48
FIGURA 17: MENSAJE DE ADVERTENCIA AL BORRAR UNA UBICACIÓN EXISTENTE	49
FIGURA 18: CAMPOS ASOCIADOS A LOS SENSORES EN LA PÁGINA WEB	50
FIGURA 19: FORMULARIO DE ACTUALIZACIÓN DE UN SENSOR YA EXISTENTE	50
FIGURA 20: MENSAJE DE ADVERTENCIA AL BORRAR UN SENSOR EXISTENTE	51
FIGURA 21: PÁGINA WEB DE EVENTOS DE UN USUARIO FICTICIO	52
FIGURA 22: PÁGINA WEB DE CREACIÓN DE EVENTOS, CON UN EVENTO EXPANDIDO	52
FIGURA 23: PÁGINA WEB DE CREACIÓN DE EVENTOS, CON UN NUEVO EVENTO CREADO	53
FIGURA 24: PÁGINA WEB DE CREACIÓN DE EVENTOS, CON UN NUEVO EVENTO DESPLEGADO.....	54
FIGURA 25: PÁGINA WEB DE EVENTOS, AL DESPLEGAR EL MENÚ DE UNA NUEVA CONDICIÓN	54
FIGURA 26: PÁGINA WEB DE CREACIÓN DE EVENTOS, SELECCIÓN DEL SENSOR	55
FIGURA 27: PÁGINA WEB DE CREACIÓN DE EVENTOS, SELECCIÓN DEL CAMPO OBJETIVO DE LA CONDICIÓN	55
FIGURA 28: PÁGINA DE CREACIÓN DE EVENTOS, SELECCIÓN DEL OPERADOR Y DEL VALOR DE COMPARACIÓN	56
FIGURA 29: PÁGINA WEB DE CREACIÓN DE EVENTOS, CREACIÓN DE UNA ACCIÓN	56
FIGURA 30: PÁGINA WEB DE CREACIÓN DE EVENTOS, SITUACIÓN FINAL TRAS GUARDAR UN NUEVO EVENTO	57
FIGURA 31: ESTRUCTURA DE DESCOMPOSICIÓN DEL PROYECTO	67
FIGURA 32: DIAGRAMA DE GANTT DEL PROYECTO	69

17. ÍNDICE DE TABLAS

TABLA 1: COMPARATIVA DE LOS PROTOCOLOS DE COMUNICACIÓN [1]	9
TABLA 2: REQUISITOS FUNCIONALES DE UN HOMEHUB (HH)	14
TABLA 3: REQUISITOS FUNCIONALES DEL SERVICIO DE MENSAJERÍA (SM)	16
TABLA 4: REQUISITOS FUNCIONALES DEL SERVICIO DE ESCUCHA (SE)	16
TABLA 5: REQUISITOS FUNCIONALES DEL SERVICIO DE ALMACENAMIENTO DE DATOS (SAD)	17
TABLA 6: REQUISITOS FUNCIONALES DEL SERVICIO "CRUD" DE USUARIOS (SCUS)	19
TABLA 7: REQUISITOS FUNCIONALES DEL SERVICIO "CRUD" DE UBICACIONES (SCUB)	19
TABLA 8: REQUISITOS FUNCIONALES DEL SERVICIO "CRUD" DE SENSORES (SCS)	19
TABLA 9: REQUISITOS FUNCIONALES DEL SERVICIO "CRUD" DE CONDICIONES-ACCIONES (SCCA)	20
TABLA 10: REQUISITOS FUNCIONALES DEL SERVICIO DE ENVÍO DE MENSAJES MQTT (SEM) DEL SERVIDOR DE APLICACIONES	20
TABLA 11: REQUISITOS FUNCIONALES DEL SERVICIO DE RESPUESTAS HTML (SR)	20
TABLA 12: REQUISITOS NO FUNCIONALES DEL SISTEMA GLOBAL	21
TABLA 13: REQUISITOS NO FUNCIONALES DEL SERVICIO DE RESPUESTA HTML (SR)	22
TABLA 14: REQUISITOS NO FUNCIONALES DEL SERVICIO DE ALMACENAMIENTO DE DATOS (SAD)	22
TABLA 15: FUNCIONES DEL SERVICIO DE ESCUCHA	37
TABLA 16: FUNCIONES DEL MÓDULO PRINCIPAL	37
TABLA 17: CAMPOS DE LA CLASE USUARIOS	38
TABLA 18: CAMPOS DE LA CLASE UBICACIONES	38
TABLA 19: CAMPOS DE LA CLASE SENSORES	38
TABLA 20: CAMPOS DE LA CLASE ATRIBUTOS DE UN TIPO DE SENSOR	38
TABLA 21: CAMPOS DE LA CLASE DATOS HISTÓRICOS DE LOS SENSORES	39
TABLA 22: CAMPOS DE LA CLASE ACTIVADOR DE EVENTOS	39
TABLA 23: CAMPOS DE LA CLASE GRUPO DE EVENTOS	39
TABLA 24: CAMPOS DE LA CLASE EVENTO	39
TABLA 25: FUNCIONES DEL MÓDULO DE USUARIOS	40
TABLA 26: FUNCIONES DEL MÓDULO DE UBICACIONES	40
TABLA 27: FUNCIONES DEL MÓDULO DE SENSORES	41
TABLA 28: FUNCIONES DEL MÓDULO DE EVENTOS	42
TABLA 29: INFORMACIÓN SOBRE LOS COMPONENTES DE LA EDP	68
TABLA 30: PRESUPUESTO DEL HOMEHUB	71
TABLA 31: PRESUPUESTO DEL SERVIDOR EN LA NUBE	71
TABLA 32: PRESUPUESTO DE LOS SENSORES	71
TABLA 33: PRESUPUESTO DE INTERCONEXIÓN	72
TABLA 34: PRESUPUESTO COSTE DE PERSONAL	72
TABLA 35: TABLA DE ABREVIATURAS Y ACRÓNIMOS	83
TABLA 36: GLOSARIO DEL PROYECTO	85

18. ABREVIATURAS Y ACRÓNIMOS

Tabla 35: Tabla de abreviaturas y acrónimos

Abreviatura o Acrónimo	Significado
3GPP	3rd Generation Partnership Project
API	<i>Application Programming Interfaces</i>
AWS	<i>Amazon Web Services</i>
CR-CA	CRUD Condiciones-Acciones
CR-S	CRUD Sensores
CR-UB	CRUD UBicaciones
CRUD	Create, Read, Update, Delete
CR-US	CRUD Usuarios
CSRK	Connection Signature Resolving Key
EDP	Estructura de Descomposición del Proyecto
HH	<i>HomeHub</i>
HTML	HyperText Markup Language
IoT	Internet of Things
LoRaWAN	Long Range Wide Area Network
MQTT	Message Queuing Telemetry Transport
NB-IoT	Narrowband Internet of Things
PAN	Personal Area Networks
RF	Requisito Funcional
RNF	Requisito No Funcional
SAD	Servicio de Almacenamiento de Datos
SAW	Servidor de Aplicaciones-Web
SCCA	Servicio CRUD Condiciones-Acciones
SCS	Servicio CRUD de Sensores
SCUB	Servicio CRUD de UBicaciones
SCUS	Servicio CRUD USuarios
SE	Servicio de Escucha
SEM	Servicio de Envío de Mensajes
SG	Sistema Global
SR	Servicio de Respuestas
TTN	The Things Network
Wi-Fi	Wireless Fidelity
WPAN	Wireless Personal Area Network

19. GLOSARIO

Tabla 36: Glosario del proyecto

Término	Definición
Back-End	Parte del sistema informático encargado del almacenamiento y manipulación de los datos. Típicamente el usuario final no tiene acceso a ella.
Back-up	Copia de seguridad de cierta información.
Broker	Software intermediario en un proceso de comunicación que permite el envío y recepción de mensajes entre dos partes.
Contenedor	Unidad estándar de <i>software</i> que empaqueta el código y todas sus dependencias para que la aplicación se ejecute de forma rápida y fiable de un entorno informático a otro. [14]
Domótica	Conjunto de sistemas que automatizan las diferentes instalaciones de una vivienda. [29]
Framework	Esquema o marco de trabajo para el desarrollo software.
Homehub	Consola de control central de cada ubicación.
Protocolo	Conjunto de reglas que se establecen en el proceso de comunicación entre dos sistemas. [29]
Servidor	Unidad informática que proporciona diversos servicios a computadoras conectadas con ella a través de una red. [29]