Universidad de Valladolid

Escuela de Ingeniería Informática

Grado en Ingeniería Informática
Mención De Computación

TRABAJO FIN DE GRADO

Diseño e implementación de un sistema para la gestión domótica usando MQTT

Autor:
D. Fernando González Sanz

Tutor:
D. Jesús María Vegas Hernández

Agradecimientos

Este trabajo no habría sido posible sin el acompañamiento y el apoyo constante de todas las personas que, de una forma u otra, han estado presentes durante estos años. A todas ellas, gracias.

A mi familia, por estar siempre ahí, y en especial a mis padres, por su apoyo incondicional, por creer en mí incluso en los momentos más difíciles y por animarme a seguir adelante cada día.

A mis amigos, por su compañía, por preocuparse, y por hacer más llevadero el camino con su cercanía y su ánimo constante.

A mis compañeros de carrera, que con el tiempo se han convertido en amigos, y con quienes compartir clases, proyectos y largas horas de estudio ha hecho todo este proceso mucho más llevadero.

A mi tutor Jesús, por su orientación durante el desarrollo de este proyecto, y por su cercanía y disponibilidad siempre que la he necesitado.

Resumen

Este proyecto presenta el desarrollo de un sistema domótico modular basado en el protocolo MQTT, enfocado a la monitorización de variables ambientales y a la detección de movimiento en entornos domésticos. El sistema se compone de nodos con ESP32 que integran sensores DHT22 y HC-SR501, además de una cámara digital, y una Orange Pi Zero 2W que actúa como broker MQTT y servidor de visualización mediante Node-RED.

Se ha conseguido establecer una infraestructura funcional que permite la transmisión en tiempo real de temperatura, humedad, detección de movimiento e imagen. El sistema es accesible tanto desde una red local como de forma remota, lo que amplía sus posibilidades de uso y supervisión. Se han evaluado aspectos clave como la estabilidad de las comunicaciones, el comportamiento de los sensores y el rendimiento del streaming de vídeo. Además, el diseño del dashboard se ha optimizado para ofrecer una visualización clara, estructurada por nodos y adaptada a distintos dispositivos.

El trabajo proporciona una base sólida sobre la que construir futuras extensiones, incluyendo la incorporación de autenticación de usuarios, grabación de vídeo y procesamiento inteligente de eventos.

Abstract

This project presents the development of a modular home automation system based on the MQTT protocol, focused on monitoring environmental variables and detecting motion in domestic environments. The system consists of ESP32 nodes that integrate DHT22 and HC-SR501 sensors, as well as a digital camera, and an Orange Pi Zero 2W acting as the MQTT broker and the visualization server through Node-RED.

A functional infrastructure has been implemented to enable real-time transmission of temperature, humidity, motion detection, and images. The system is accessible both from a local network and remotely, which enhances its usability and supervision capabilities. Key aspects such as communication stability, sensor performance, and video streaming quality have been evaluated. Additionally, the dashboard design has been optimized to provide a clear, node-structured visualization adaptable to different devices.

This work provides a solid foundation for future extensions, including user authentication, video recording, and intelligent event processing.

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

Capítulo 1

Introducción y objetivos

1.1. Introducción

En el contexto actual de la transformación digital, la automatización del hogar o domótica se ha convertido en una de las aplicaciones más accesibles y útiles del Internet de las Cosas (IoT). La creciente disponibilidad de microcontroladores económicos y potentes, como el ESP32, junto con la proliferación de sensores y actuadores, ha hecho posible desarrollar sistemas domóticos personalizados sin depender de soluciones comerciales cerradas y costosas.

Este trabajo de fin de grado tiene como objetivo el diseño e implementación de un sistema domótico distribuido, basado en tecnologías de bajo coste y software libre, que permita el control y monitoreo remoto de dispositivos del hogar mediante una arquitectura basada en el protocolo de mensajería MQTT. Este protocolo ha sido elegido por su ligereza, eficiencia y escalabilidad, características que lo hacen ideal para entornos con restricciones de red y energía, como los propios de dispositivos embebidos.

El sistema desarrollado se compone de dos nodos basados en placas ESP32, cada uno equipado con una cámara de videovigilancia, un sensor de temperatura y humedad (DHT22) y un sensor de movimiento (HC-SR501). La comunicación entre los nodos y el servidor central se realiza a través de un broker MQTT alojado en una Orange Pi Zero 2W, actuando como núcleo del sistema. Los datos capturados se publican en diferentes topics jerárquicos y pueden ser visualizados en tiempo real a través de una interfaz web, desarrollada también como parte del proyecto.

Este trabajo pretende mostrar cómo es posible construir un sistema funcional sin recurrir a plataformas comerciales propietarias. Además, el diseño planteado es modular y escalable, permitiendo la integración futura de nuevos sensores o funcionalidades adicionales con un mínimo esfuerzo.

En los capítulos siguientes se describen los fundamentos teóricos del protocolo MQTT, el diseño del sistema, la planificación y ejecución del desarrollo, así como la implementación final del sistema.

1.2. Motivación

La motivación principal de este trabajo radica en la voluntad de diseñar y construir un sistema domótico completo, funcional y personalizable, sin depender de soluciones comerciales propietarias. Los

sistemas disponibles en el mercado suelen estar limitados en cuanto a flexibilidad y posibilidades de personalización, además de implicar costes elevados tanto en adquisición como en instalación y mantenimiento.

Este proyecto representa una oportunidad para aplicar conocimientos adquiridos a lo largo del grado en ingeniería informática, abarcando áreas como la programación embebida, las comunicaciones en red, el diseño de interfaces gráficas y la gestión de sistemas distribuidos. A su vez, permite profundizar en el protocolo MQTT, ampliamente adoptado en entornos IoT por su bajo consumo de recursos, simplicidad de implementación y fiabilidad en entornos con conectividad limitada.

Además del interés técnico, existe una motivación pedagógica: demostrar que es posible, desde un enfoque autodidacta y con recursos limitados, implementar una solución domótica robusta, eficiente y escalable. La posibilidad de acceder a los datos de sensores en tiempo real, así como de visualizar las cámaras de videovigilancia desde una interfaz web, ofrece un valor añadido al proyecto y pone de manifiesto el potencial del enfoque modular adoptado.

Finalmente, este trabajo también responde a una inquietud personal por explorar las posibilidades que ofrecen las tecnologías abiertas, fomentando el aprendizaje activo, la innovación y la reutilización del conocimiento para futuras mejoras o nuevos desarrollos.

1.3. Objetivos

El objetivo general de este trabajo de fin de grado es diseñar y desarrollar un **prototipo funcional** de sistema domótico modular, accesible y económico, basado en el protocolo MQTT. Este prototipo busca demostrar la viabilidad técnica de una solución que permita la monitorización remota de sensores y la transmisión de imágenes desde dispositivos distribuidos en el entorno doméstico.

Desde un principio, se ha priorizado la sencillez, escalabilidad y bajo coste de la solución, utilizando componentes asequibles como el ESP32, sensores DHT22 y HC-SR501, y una Orange Pi Zero 2W como servidor central. El objetivo no es construir un sistema comercial cerrado, sino validar un modelo que pueda ser ampliado o adaptado en el futuro según las necesidades del usuario.

Para cumplir con este objetivo principal, se han planteado los siguientes objetivos específicos:

- Analizar el protocolo MQTT, su arquitectura y funcionamiento, con especial atención a su idoneidad en entornos de IoT y sistemas embebidos.
- Seleccionar y configurar tecnologías de hardware y software adecuadas para la construcción de nodos sensores y del servidor central de comunicaciones.
- Diseñar la arquitectura del prototipo, incluyendo la estructura de topics MQTT, los flujos de datos y el modelo de interacción entre nodos y servidor.
- Implementar los componentes clave del prototipo: programación de los nodos ESP32, instalación y configuración del broker MQTT, y desarrollo de una interfaz visual básica mediante Node-RED.
- Validar el funcionamiento básico del sistema a través de pruebas que permitan verificar la recolección, envío y visualización de datos en tiempo real.

- Documentar el proceso de desarrollo con el fin de facilitar su comprensión, replicabilidad y futuras mejoras por parte de terceros.

Capítulo 2

Estado del arte

Este capítulo ofrece una revisión de los conceptos y tecnologías fundamentales sobre los que se apoya el desarrollo del sistema propuesto. Se analiza el contexto actual de la domótica y el Internet de las Cosas (IoT), así como el papel central que desempeña el protocolo MQTT como mecanismo de comunicación entre los distintos dispositivos del sistema.

2.1. Domótica e Internet de las Cosas

La domótica engloba el conjunto de tecnologías orientadas a la automatización y control de viviendas y edificios, con el fin de mejorar el confort, la eficiencia energética, la seguridad y la accesibilidad. Gracias a la evolución de las redes de comunicaciones, los sensores y los sistemas embebidos, ha sido posible desarrollar soluciones domóticas cada vez más asequibles y flexibles, que pueden adaptarse a las necesidades particulares de cada entorno.

El concepto de Internet de las Cosas (IoT, por sus siglas en inglés) se refiere a la interconexión de objetos físicos a través de redes digitales, permitiendo que dispositivos cotidianos recojan datos, se comuniquen entre sí y respondan a condiciones del entorno. En este contexto, la domótica representa una de las aplicaciones más consolidadas y con mayor proyección, ya que permite extender las capacidades de monitorización y control del hogar más allá del entorno físico, mediante interfaces accesibles desde cualquier lugar.

Para que estos sistemas funcionen de manera eficiente, es fundamental contar con protocolos de comunicación que sean ligeros, fiables y adaptables a dispositivos con recursos limitados. Entre las distintas alternativas existentes, MQTT se ha consolidado como una de las opciones más utilizadas en soluciones IoT, tanto a nivel doméstico como industrial.

2.2. El protocolo MQTT como base para sistemas IoT

2.2.1. Introducción

Uno de los pilares tecnológicos sobre los que se sustenta este proyecto es el protocolo de comunicación MQTT (Message Queuing Telemetry Transport). Esta sección realiza un análisis detallado de

dicho protocolo, abordando su origen, evolución, arquitectura y características técnicas, con el objetivo de justificar su idoneidad en el contexto de soluciones domóticas distribuidas basadas en IoT.

MQTT (Message Queuing Telemetry Transport) es un protocolo de mensajería ampliamente adoptado en aplicaciones de Internet de las Cosas (IoT), debido a su eficiencia, simplicidad y bajo consumo de recursos. Se basa en un modelo de comunicación publicador/suscriptor que permite desacoplar emisores y receptores de mensajes, lo que facilita la integración de múltiples dispositivos en arquitecturas distribuidas.

Entre sus principales características se encuentran su ligereza, flexibilidad, escalabilidad, uso eficiente del ancho de banda, baja latencia, soporte para sesiones persistentes, comunicación bidireccional y su naturaleza agnóstica respecto al formato de los datos transmitidos. Estas propiedades lo convierten en una solución particularmente adecuada para sistemas como el desarrollado en este proyecto, donde intervienen sensores, microcontroladores y servidores conectados en red.

2.2.2. Origen e historia

El protocolo MQTT (Message Queuing Telemetry Transport) fue desarrollado en 1999 por Andy Stanford-Clark (IBM) y Arlen Nipper (Arcom, actualmente Eurotech), con el propósito de ofrecer una solución de mensajería eficiente que redujera el consumo de ancho de banda y energía. Su primera aplicación se centró en sistemas de supervisión vía satélite para el transporte de petróleo en zonas desérticas del Medio Oriente, donde la infraestructura era limitada y la conectividad extremadamente restringida [6].

En 2013, MQTT fue adoptado como estándar por la Organization for the Advancement of Structured Information Standards (OASIS), lo que consolidó su uso en entornos industriales y comerciales. Posteriormente, en 2014, se publicó la versión 3.1.1 del protocolo, que introdujo mejoras orientadas a la interoperabilidad entre distintas implementaciones y a la claridad de la especificación.

La versión más reciente, MQTT 5.0, fue publicada en 2019 y supuso una evolución significativa respecto a sus predecesoras. Entre sus principales novedades destacan una mayor escalabilidad, la inclusión de mecanismos detallados para el reporte de errores y la posibilidad de incorporar propiedades personalizadas en las cabeceras de los mensajes [14].

2.2.3. Características generales

El protocolo MQTT incorpora una serie de propiedades que lo convierten en una solución especialmente adecuada para aplicaciones en entornos IoT. Estas características explican su amplia adopción en sistemas distribuidos con restricciones de ancho de banda, latencia o consumo energético [14, 6].

Ligereza

MQTT se distingue por su baja sobrecarga en la transmisión de datos. Utiliza cabeceras mínimas y estructuras de mensaje simples, lo que permite reducir significativamente el tamaño de los paquetes. Esta eficiencia resulta clave en redes con ancho de banda limitado y dispositivos con recursos restringidos, como microcontroladores y sensores típicos en entornos IoT.

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type				Flags specific to each MQTT Control Packet type			
byte 2…	Remaining Length							

Figura 2.1: Cabecera de un mensaje MQTT [14]

La Figura 2.1 muestra la estructura de la cabecera fija que acompaña a todos los paquetes de control en MQTT. Esta cabecera, de tamaño mínimo de dos bytes, está optimizada para maximizar la eficiencia:

- **Byte 1:** Contiene el tipo de paquete (por ejemplo, `CONNECT`, `PUBLISH` o `SUBSCRIBE`) y los flags específicos asociados, como `DUP`, `QoS` o `RETAIN`.
- **Byte 2 en adelante:** Codifica el campo Remaining Length, que indica la longitud total del mensaje restante. Este campo puede ocupar hasta cuatro bytes, según el tamaño del contenido.

Flexibilidad

El modelo publicador/suscriptor empleado por MQTT permite un alto grado de desacoplamiento entre emisores y receptores, lo que facilita la integración de dispositivos heterogéneos. Esta flexibilidad se extiende a la gestión de fiabilidad mediante tres niveles de *Quality of Service* (QoS):

- **QoS 0:** El mensaje se envía una única vez, sin confirmación. Adecuado para datos no críticos.
- **QoS 1:** El mensaje se entrega al menos una vez, con posible duplicación.
- **QoS 2:** El mensaje se entrega exactamente una vez, garantizando máxima fiabilidad.

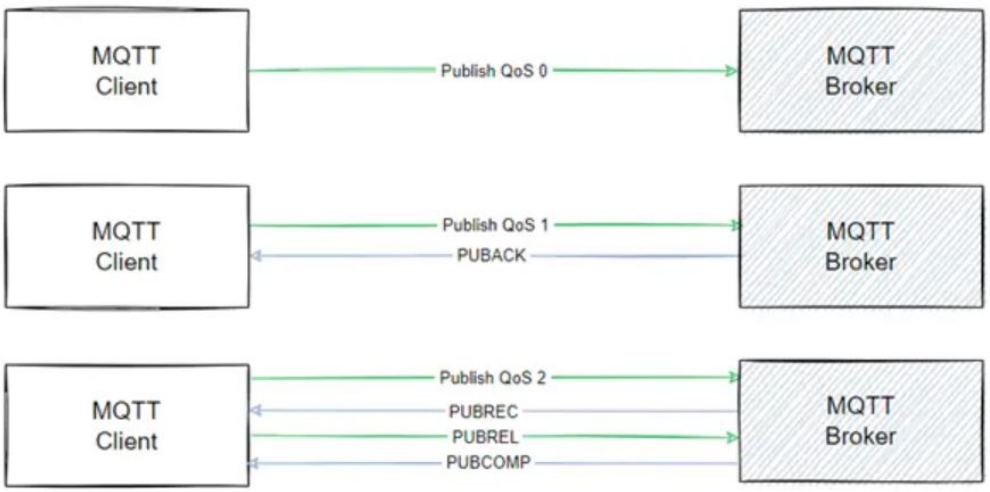

Figura 2.2: Esquema de QoS en MQTT [21]

Otras capacidades que refuerzan la flexibilidad del protocolo son:

- **Topologías adaptables:** Permite arquitecturas centralizadas o distribuidas, con múltiples brokers y pasarelas.
- **Mensajes retenidos:** La opción `retained` facilita que los nuevos suscriptores reciban la última información disponible al conectarse.
- **Compatibilidad multiplataforma:** Existen implementaciones en numerosos lenguajes y sistemas operativos.

Escalabilidad

Gracias a su arquitectura desacoplada, MQTT permite construir redes con gran número de dispositivos sin que estos deban conocerse entre sí. Esta propiedad, unida al uso de topics jerárquicos para organizar los mensajes, facilita el crecimiento progresivo del sistema.

El protocolo ha demostrado su capacidad para escalar en entornos industriales, agrícolas y logísticos, y existen brokers comerciales capaces de gestionar millones de mensajes por segundo [15].

Sesiones persistentes

MQTT permite mantener el estado de los clientes a través de sesiones persistentes. Cuando un cliente se reconecta, puede recuperar automáticamente sus suscripciones y recibir mensajes que quedaron pendientes durante su desconexión, en función del QoS configurado. Esta característica resulta esencial en dispositivos con conectividad intermitente.

Agnóstico de datos

El protocolo no impone ningún formato específico al contenido de los mensajes, lo que permite transportar desde texto plano hasta binarios complejos, como imágenes o estructuras serializadas. Esta neutralidad permite que el broker se limite a reenviar los mensajes sin interpretarlos, dejando el procesamiento en manos de los extremos, lo que contribuye a mejorar el rendimiento y simplificar el diseño.

Bajo consumo energético

El diseño ligero de MQTT, combinado con sesiones persistentes y configuraciones de QoS adaptables, favorece un uso eficiente de la energía. Esto permite a los dispositivos IoT permanecer en modo de bajo consumo durante largos periodos, activándose solo cuando deben transmitir o recibir datos. Por este motivo, MQTT es especialmente útil en escenarios donde la autonomía energética es crítica, como en sensores remotos o aplicaciones alimentadas por batería.

2.2.4. Arquitectura publicador/suscriptor

El protocolo MQTT se basa en una arquitectura de tipo publicador/suscriptor, que se diferencia notablemente del modelo tradicional cliente-servidor. Este enfoque permite una mayor flexibilidad y es-

calabilidad, ya que los emisores y receptores de los mensajes no necesitan estar conectados entre sí directamente, sino que se comunican a través de un intermediario común: el bróker.

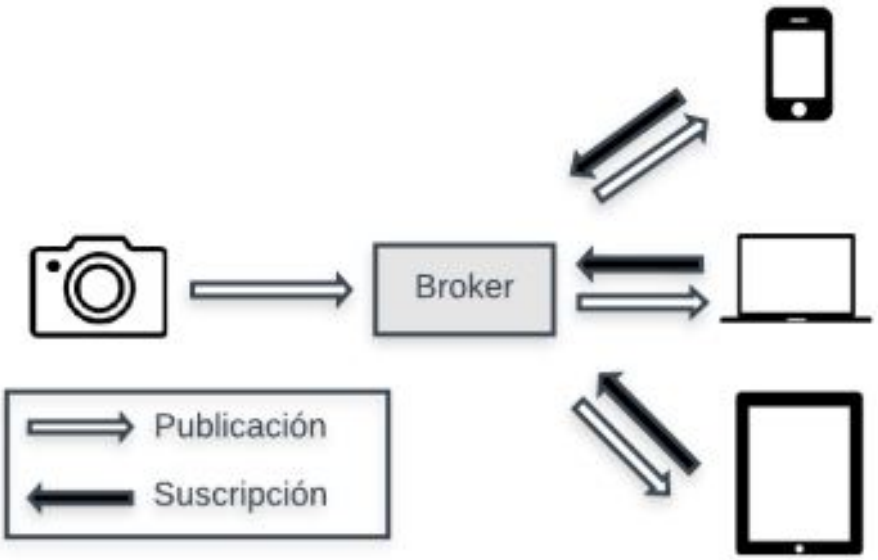

Figura 2.3: Ejemplo de arquitectura publicador/suscriptor [5]

Elementos principales del modelo

La arquitectura se compone de tres elementos fundamentales:

- **Bróker MQTT:** Componente central encargado de recibir los mensajes publicados por los clientes y reenviarlos únicamente a aquellos suscriptores que estén registrados en el *topic* correspondiente. Además, gestiona la autenticación, autorización, sesiones persistentes y el almacenamiento de mensajes pendientes según el nivel de calidad de servicio (QoS).
- **Cliente MQTT:** Cualquier dispositivo que se conecta al bróker, ya sea para publicar, suscribirse o ambas cosas. Cada cliente debe autenticarse para establecer la conexión.
- **Topic:** Canal lógico a través del cual se organiza y distribuye la información. Su estructura es jerárquica, utilizando barras inclinadas (`/`) como separadores. Cada mensaje va asociado a un topic, y los clientes se suscriben a uno o varios para recibir la información deseada.

Ejemplos de topics comunes:

- `casa/cocina/humedad`
- `casa/salon/luz`
- `casa/baño/humedad`

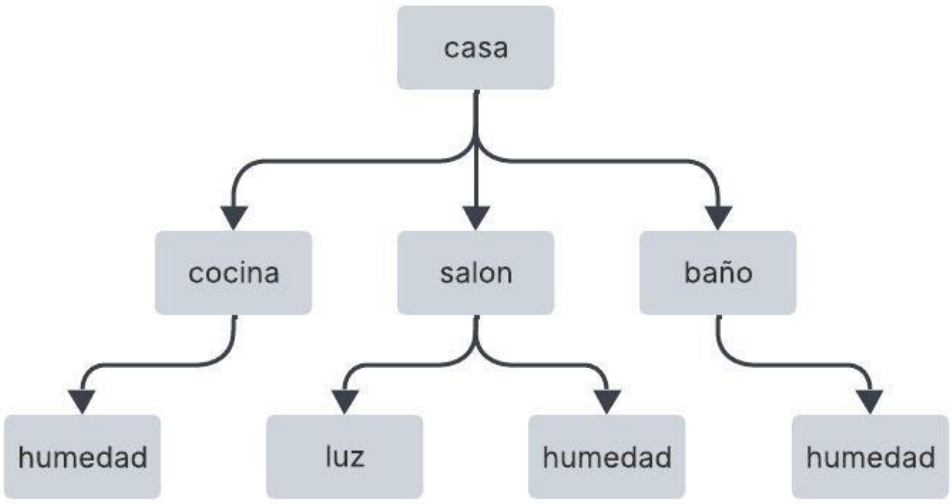

Figura 2.4: Ejemplo de jerarquía de topics

MQTT también permite el uso de comodines para ampliar la flexibilidad de suscripciones:

- **+**: Sustituye un único nivel jerárquico.
  Ejemplo: `casa/+/humedad` abarca `casa/cocina/humedad`, `casa/salon/humedad`, etc.
- **#**: Sustituye cero o más niveles, pero solo puede colocarse al final del topic.
  Ejemplo: `casa/#` incluye todos los topics que comienzan con `casa/`, como `casa/salon/luz`.
  La suscripción `casa/#/humedad` no es válida, ya que el símbolo `#` solo puede aparecer al final.

Desacoplamiento

Una de las principales ventajas del modelo publicador/suscriptor es el alto grado de desacoplamiento que introduce entre los componentes. Al comunicarse únicamente con el bróker, los clientes no necesitan conocer la identidad ni el estado de otros dispositivos. Este desacoplamiento se manifiesta en tres niveles:

- **Espacial:** Los clientes no necesitan conocerse ni intercambiar direcciones.
- **Temporal:** No es necesario que los clientes estén activos al mismo tiempo, gracias al soporte de sesiones persistentes y mensajes retenidos.
- **De sincronización:** No se requiere coordinación temporal entre los dispositivos; el bróker se encarga de gestionar y entregar los mensajes de forma asíncrona.

Filtrado de mensajes

El sistema de filtrado basado en topics permite distribuir la información de forma dirigida, evitando tráfico innecesario. La jerarquía en la nomenclatura de los topics facilita la organización lógica del

sistema y mejora la mantenibilidad. No obstante, una mala definición de la estructura de topics puede comprometer la claridad y escalabilidad del diseño, por lo que es fundamental establecerla correctamente desde el inicio del proyecto.

Escalabilidad

El modelo está diseñado para admitir un gran número de dispositivos sin comprometer el rendimiento. El bróker actúa como procesador de eventos, gestionando suscripciones y publicaciones de forma asincrónica. El filtrado eficiente basado en topics y la entrega dirigida permiten optimizar el uso de red, reducir la carga en los dispositivos y mantener una arquitectura distribuida capaz de crecer sin necesidad de cambios estructurales.

2.2.5. Conexión e intercambio de mensajes

El protocolo MQTT define una serie de mensajes específicos para gestionar el establecimiento de conexiones, suscripciones, publicaciones y otros aspectos clave del flujo de comunicación entre clientes y bróker [16, 14, 22]. A continuación, se describen los mecanismos principales que conforman este proceso.

Conexión

La conexión entre un cliente y el bróker se inicia con el envío de un mensaje `CONNECT`, que debe ser respondido por el bróker con un mensaje `CONNACK` confirmando el resultado de la operación.

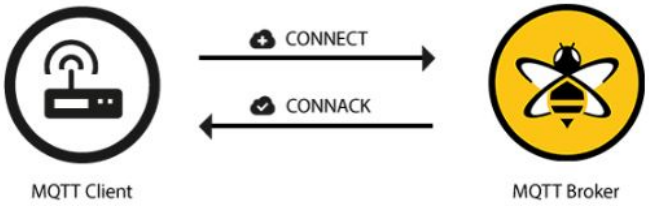

Figura 2.5: Establecimiento de conexión cliente-bróker [14]

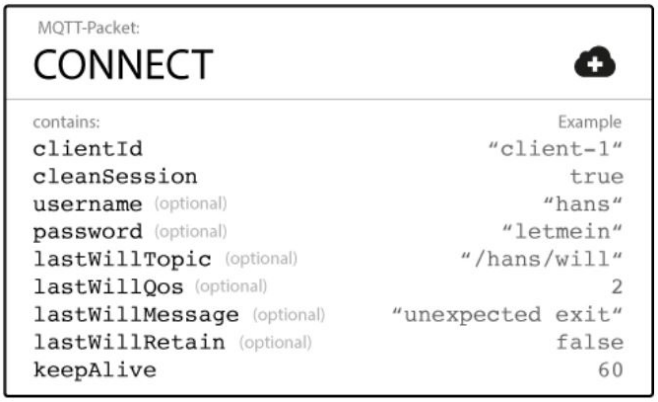

Figura 2.6: Contenido del mensaje CONNECT [14]

Entre los campos más relevantes del mensaje `CONNECT` se encuentran:

- **clientId:** Identificador único del cliente, utilizado por el bróker para mantener el estado de la sesión.
- **cleanSession:** Indica si la sesión debe ser persistente. En caso de ser `false`, el bróker almacena suscripciones y mensajes pendientes.
- **username y password:** Opcionales. Permiten realizar autenticación, aunque se recomienda usar cifrado TLS para proteger estas credenciales.
- **lastWill:** Define el mensaje que se publicará si el cliente se desconecta inesperadamente.
- **keepAlive:** Intervalo máximo permitido sin actividad antes de que el bróker cierre la conexión. Se supervisa mediante mensajes `PINGREQ` y `PINGRESP`.

El mensaje `CONNACK` confirma la conexión e informa si ya existía una sesión activa.

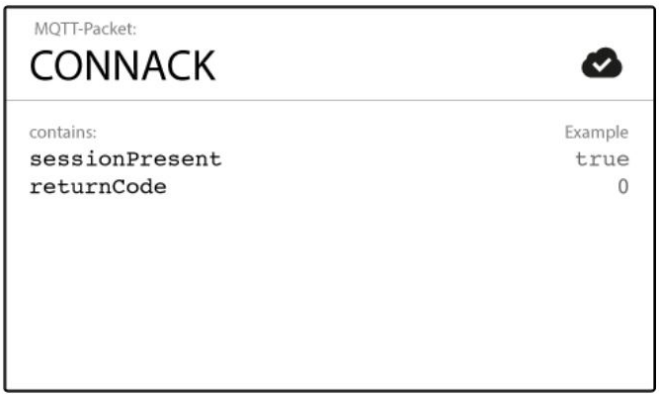

Figura 2.7: Contenido del mensaje CONNACK [14]

Suscripción

Una vez conectado, el cliente puede enviar un mensaje `SUBSCRIBE` para recibir mensajes de uno o varios topics.

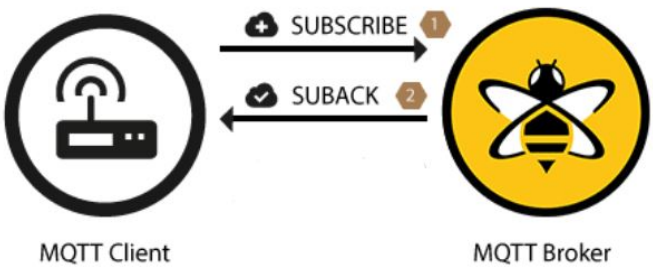

Figura 2.8: Intercambio de mensajes para la suscripción [14]

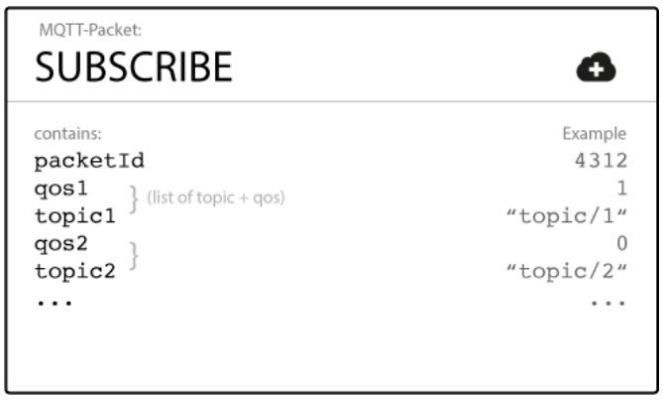

Figura 2.9: Contenido del mensaje SUBSCRIBE [14]

Cada mensaje `SUBSCRIBE` incluye:

- **packetId:** Identificador del mensaje.
- **topic y QoS:** Lista de pares (topic, nivel de calidad) para cada suscripción.

El bróker responde con un mensaje `SUBACK` indicando el resultado:

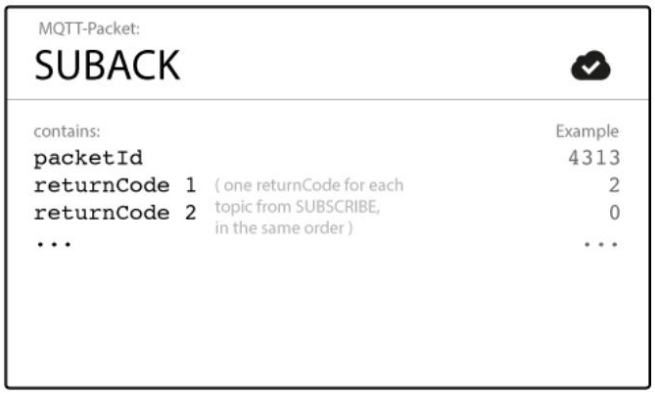

Figura 2.10: Contenido del mensaje SUBACK [14]

Publicación

Para enviar información, los clientes utilizan el mensaje `PUBLISH`. Este contiene el topic, el contenido (payload), y diversos metadatos:

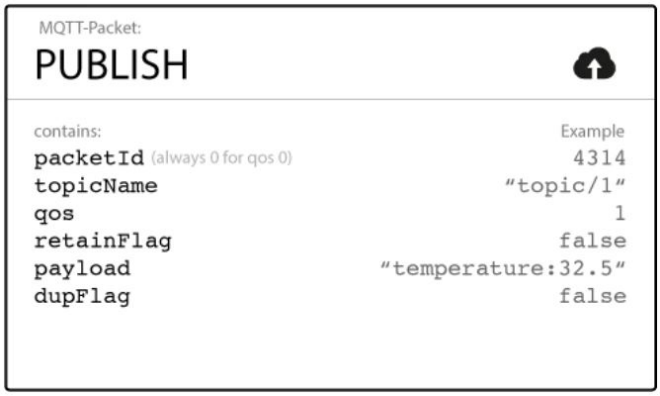

Figura 2.11: Contenido del mensaje PUBLISH [14]

- **packetId:** Identificador único del mensaje.
- **topicName:** Topic de destino.
- **QoS:** Nivel de calidad de servicio.
- **retainFlag:** Indica si el mensaje debe ser retenido por el bróker.
- **payLoad:** Contenido del mensaje, que puede adoptar cualquier formato.
- **dupFlag:** Marca los mensajes reenviados tras fallo en la entrega.

Calidad de Servicio (QoS)

El protocolo define tres niveles de entrega garantizada:

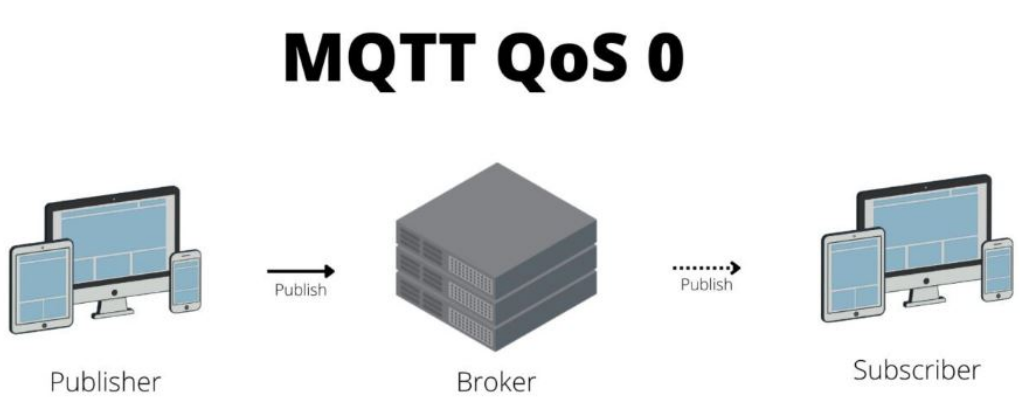

Figura 2.12: Esquema QoS 0 [7]

QoS 0: "Disparar y olvidar" El mensaje se envía una sola vez, sin confirmación. Es el nivel más eficiente, pero sin garantía de entrega.

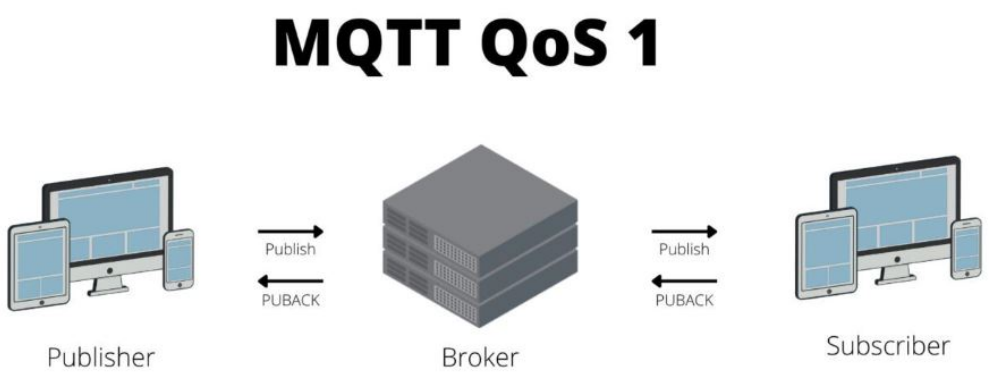

Figura 2.13: Esquema QoS 1 [7]

QoS 1: Entrega al menos una vez El publicador conserva el mensaje hasta recibir el `PUBACK` del bróker. Pueden producirse duplicados.

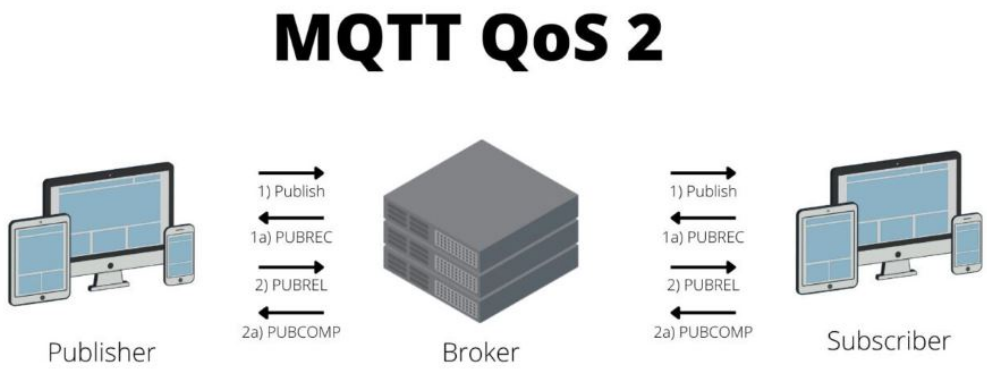

Figura 2.14: Esquema QoS 2 [7]

QoS 2: Entrega exactamente una vez Proceso en cuatro pasos que garantiza que el mensaje no se duplique ni se pierda. Es el nivel más fiable, aunque el más exigente en recursos.

Cancelación de suscripción

Para dejar de recibir mensajes, el cliente envía un mensaje `UNSUBSCRIBE`, al que el bróker responde con `UNSUBACK`.

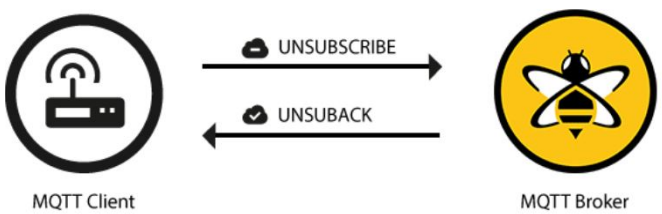

Figura 2.15: Intercambio de mensajes para cancelar la suscripción [14]

Mensajes PING

Para mantener la conexión activa, el cliente puede enviar mensajes `PINGREQ` al bróker, quien debe responder con `PINGRESP`. Si no se recibe respuesta dentro del margen definido por `keepAlive`, la conexión se considera inactiva.

Desconexión

Cuando el cliente desea cerrar la sesión, envía un mensaje `DISCONNECT`. Si se había establecido una sesión persistente, el bróker conserva su estado y sus suscripciones para futuras reconexiones.

Resumen de tipos de mensajes

A continuación, se muestra una lista de los principales tipos de mensajes utilizados en MQTT.

Tipo de mensaje Tipo de mensaje	Código	Descripción
CONNECT	0X10	Petición del cliente para conectarse al servidor
CONNACK	0X20	Reconocimiento de la conexión
PUBLISH	0X30	Edición del mensaje
PUBACK	0X40	Reconocimiento de la edición
PUBREC	0X50	Edición recibida
PUBREL	0X60	Edición liberada
PUBCOMP	0X70	Edición completa
SUSCRIBE	0X80	Petición de suscripción del cliente
SUBACK	0X90	Reconocimiento de suscripción
UNSUSCRIBE	0XA0	Petición de desuscripción del cliente
UNSUBACK	0XB0	Reconocimiento de desuscripción
PINGREQ	0XC0	Petición de ping
PINGRESP	0XD0	Respuesta de ping
DISCONNECT	0XE0	Cliente desconectado

Figura 2.16: Lista de mensajes MQTT [5]

2.2.6. Seguridad

El diseño original del protocolo MQTT no incluía mecanismos de seguridad integrados, ya que estaba destinado inicialmente a su uso en redes privadas y entornos controlados. No obstante, su adopción en contextos abiertos y distribuidos, propios del Internet de las Cosas (IoT), ha hecho imprescindible incorporar medidas de protección que garanticen la confidencialidad, integridad y autenticación de las comunicaciones.

Actualmente, existen diversas soluciones externas orientadas a mitigar riesgos como accesos no autorizados, interceptación de datos, suplantación de identidad o ataques de denegación de servicio. Estas medidas permiten proteger tanto al bróker como a los distintos clientes conectados, asegurando una transmisión segura de la información.

Autenticación

MQTT admite mecanismos básicos de autenticación mediante el uso de credenciales, concretamente un nombre de usuario y una contraseña incluidos en el mensaje `CONNECT`. Esta medida permite al bróker validar la identidad de cada cliente antes de establecer la sesión.

Sin embargo, si no se utiliza una capa de cifrado adicional, estas credenciales se transmiten en texto plano, lo que representa un riesgo de seguridad considerable. Para evitarlo, se recomienda emplear cifrado TLS, que asegura la confidencialidad de las credenciales durante su transmisión.

Autorización

Una vez autenticado el cliente, el sistema puede aplicar políticas de autorización que definan qué operaciones tiene permitido realizar. El bróker puede restringir las acciones de publicación y suscripción según el identificador del cliente y los topics implicados.

Por ejemplo, se puede permitir a un cliente publicar únicamente en `casa/salon/luz`, sin autorizarle a suscribirse a otros topics. Esta funcionalidad es clave para mantener la seguridad y el aislamiento lógico entre dispositivos dentro de una red compartida.

Cifrado de mensajes

MQTT, por sí mismo, no proporciona cifrado de los datos transmitidos. Por este motivo, es necesario incorporar una capa adicional de seguridad, siendo la solución más habitual el uso de TLS (Transport Layer Security). Este protocolo permite cifrar las comunicaciones entre cliente y bróker, evitando que terceros no autorizados puedan acceder al contenido.

TLS actúa entre la capa de transporte y la de aplicación, estableciendo un canal seguro sobre el que se ejecuta MQTT. En la actualidad, se desaconseja el uso de su predecesor SSL debido a sus vulnerabilidades conocidas. Por tanto, cualquier implementación moderna debe recurrir exclusivamente a versiones actualizadas de TLS.

Conclusión

Aunque MQTT fue concebido inicialmente como un protocolo ligero y sin seguridad integrada, su aplicación en entornos IoT hace imprescindible incorporar medidas que protejan tanto la infraestructura como los datos intercambiados.

La combinación de autenticación, autorización y cifrado a través de TLS permite desplegar sistemas robustos, preparados para resistir accesos indebidos e intentos de interceptación. La seguridad debe considerarse desde el diseño del sistema como un componente esencial, especialmente cuando se gestionan datos personales o se controlan dispositivos con capacidad de actuación sobre el entorno.

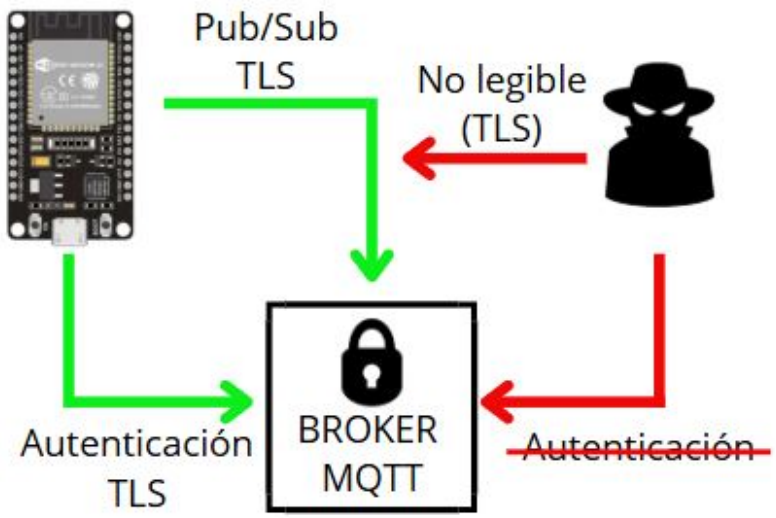

Figura 2.17: Esquema de seguridad en MQTT

2.3. Alternativas al uso de MQTT en sistemas IoT

Aunque en este proyecto se ha optado por utilizar el protocolo MQTT debido a su bajo consumo de recursos, fiabilidad y buena adaptación a arquitecturas distribuidas, existen otras opciones que también se emplean en sistemas de comunicación para dispositivos IoT. En esta sección se describen algunas de ellas, resaltando sus principales características y los motivos por los que no han sido seleccionadas para esta implementación.

2.3.1. CoAP (Constrained Application Protocol)

CoAP está diseñado específicamente para entornos IoT con restricciones de red y hardware. Funciona sobre UDP, lo que reduce la latencia y la sobrecarga en la transmisión de mensajes. Además, adopta un modelo RESTful con comandos como GET, POST, PUT y DELETE, pero en un formato binario más eficiente que HTTP. Otra ventaja es que permite comunicación multicast y admite niveles de fiabilidad ajustables [11]. No obstante, su menor adopción práctica y la complejidad que puede suponer su depuración hacen que resulte menos accesible que MQTT.

2.3.2. AMQP (Advanced Message Queuing Protocol)

AMQP se caracteriza por ofrecer una entrega de mensajes altamente fiable, incluyendo colas, confirmaciones y transacciones. Es un protocolo muy sólido para aplicaciones empresariales en las que la integridad de los datos es crítica, como en banca o gestión de procesos industriales [14]. Sin embargo, su peso y complejidad lo convierten en una opción poco adecuada para dispositivos embebidos o redes con recursos limitados, como las utilizadas en este trabajo.

2.3.3. HTTP/REST

La arquitectura REST sobre HTTP es una de las soluciones más conocidas y extendidas, también dentro del ámbito IoT. Su simplicidad y compatibilidad con la infraestructura web existente facilitan su integración, especialmente cuando se trata de consultar o modificar el estado de dispositivos mediante APIs [18]. Aun así, la falta de persistencia en las conexiones y el mayor consumo energético hacen que no sea la opción más adecuada para sistemas que requieren comunicación eficiente y en tiempo real.

2.3.4. WebSockets

WebSockets permiten establecer una conexión bidireccional y persistente entre cliente y servidor utilizando una sola conexión TCP. Esto los convierte en una opción muy interesante para aplicaciones que necesitan actualizaciones en tiempo real, como paneles de control o interfaces web dinámicas [14]. Sin embargo, no cuentan con características propias de fiabilidad como los niveles de QoS de MQTT, y su implementación puede resultar más exigente desde el punto de vista de recursos.

2.3.5. XMPP (Extensible Messaging and Presence Protocol)

XMPP es un protocolo basado en XML que originalmente fue desarrollado para mensajería instantánea, pero que también ha sido adaptado a entornos IoT. Destaca por su capacidad de extensibilidad y por su arquitectura descentralizada orientada a eventos [12]. A pesar de estas ventajas, el uso de XML implica una mayor carga de red y procesamiento, lo que puede dificultar su adopción en dispositivos con recursos limitados. Además, su presencia en sistemas IoT sigue siendo menos común que la de otros protocolos más ligeros como MQTT o CoAP.

2.3.6. Comparativa de protocolos

A modo de resumen, en la Tabla 2.1 se recoge una comparativa de los principales protocolos analizados en función de varios criterios relevantes para sistemas IoT: persistencia de la conexión, bidireccionalidad, ligereza, soporte para calidad de servicio (QoS) y grado de idoneidad general.

Protocolo	**Persistente**	**Bidireccional**	**Ligero**	**QoS**	**Idóneo IoT**
MQTT	Sí	Sí	Sí	Sí	**Sí**
HTTP/REST	No	No	Medio	No	Parcial
WebSockets	Sí	Sí	Medio	No	Parcial
CoAP	No	Parcial	Sí	Básico	Sí
AMQP	Sí	Sí	No	Sí	No
XMPP	Sí	Sí	No	No	Parcial

Tabla 2.1: Comparativa entre protocolos de comunicación para IoT

Capítulo 3

Planificación

Para llevar a cabo la planificación del proyecto, este se ha dividido en varias fases que representan los distintos bloques de trabajo a realizar. Aunque algunas de estas fases se corresponden parcialmente con capítulos del documento, la división responde principalmente a criterios organizativos y de ejecución. A continuación, se detallan dichas fases.

Se ha optado por un enfoque de metodología en cascada, en el cual cada fase se inicia una vez completada la anterior, manteniendo así una secuencia lógica en el desarrollo. No obstante, se han planificado algunas fases para ejecutarse en paralelo de forma controlada, como es el caso de las pruebas iniciales de hardware y la redacción de la memoria. Esta ejecución simultánea no rompe el modelo en cascada, ya que dichas fases no presentan dependencias críticas con las restantes y su desarrollo paralelo responde a razones de eficiencia, aprovechamiento del tiempo y prevención de riesgos.

A continuación se muestran todas las fases del proyecto en un diagrama de Gantt.

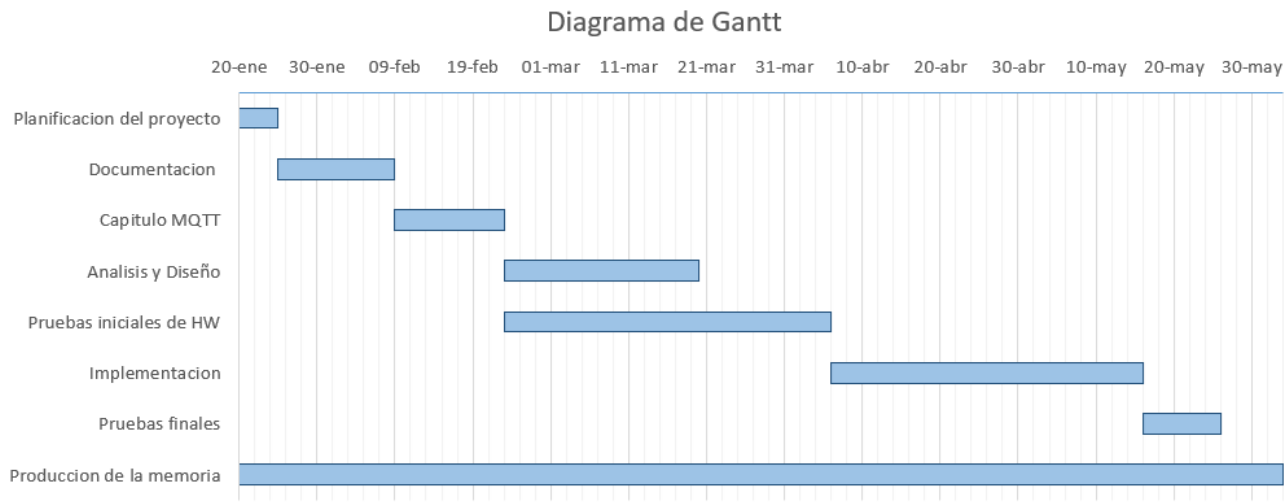

Figura 3.1: Diagrama de Gantt

La duración estimada del proyecto es de 134 días, comenzando el 20 de enero y finalizando aproximadamente el 3 de junio. No obstante, no se trabajará todos los días de forma continua. Teniendo en cuenta fines de semana, festivos y otras obligaciones personales, se estima una dedicación efectiva de unos 100 días.

El esfuerzo total previsto es de unas 300 horas de trabajo, lo que implica una media aproximada de 3 horas diarias durante los días activos del proyecto.

3.1. Fases de la planificación

3.1.1. Fase 1: Planificación del proyecto

Durante esta fase se lleva a cabo la organización inicial del trabajo. Se definen los objetivos generales del proyecto, se identifican las tareas principales y se establecen estimaciones temporales para cada una de ellas. Esta etapa resulta fundamental para orientar correctamente el desarrollo posterior, garantizar una distribución equilibrada del esfuerzo y anticipar posibles dificultades o dependencias entre tareas.

- Fecha de inicio esperada: 20 de Enero.
- Fecha de fin esperada: 25 de Enero.

3.1.2. Fase 2: Documentación

En esta fase se lleva a cabo una labor de investigación y revisión bibliográfica para obtener una comprensión profunda del contexto del proyecto. Se estudian trabajos previos relacionados (estado del arte), se analiza el funcionamiento del protocolo MQTT, se evalúan distintas opciones de hardware y se exploran posibles enfoques de implementación. Esta etapa proporciona la base teórica y técnica necesaria para tomar decisiones fundamentadas en las fases posteriores.

- Fecha de inicio esperada: 25 de Enero.
- Fecha de fin esperada: 9 de Febrero.

3.1.3. Fase 3: Análisis del protocolo MQTT

Esta fase está dedicada al estudio detallado del protocolo de comunicación MQTT, pieza clave en la arquitectura del proyecto. Se analiza su funcionamiento, características, ventajas frente a otros protocolos y casos de uso típicos. Además, se investiga cómo integrar MQTT de forma efectiva en el sistema propuesto, evaluando su aplicabilidad a nivel de rendimiento, escalabilidad y compatibilidad con el hardware y software seleccionados.

- Fecha de inicio esperada: 9 de Febrero.
- Fecha de fin esperada: 23 de Febrero.

3.1.4. Fase 4: Análisis y Diseño

Durante esta fase se lleva a cabo un análisis exhaustivo del sistema a desarrollar. Se identifican y definen los requisitos funcionales y no funcionales, así como las necesidades específicas del entorno de aplicación. A partir de este análisis, se diseña la estructura general del sistema, incluyendo la selección de tecnologías, componentes de hardware y software, y la definición de la arquitectura del proyecto. Esta fase es clave para sentar las bases técnicas del desarrollo e implementar soluciones viables y sostenibles.

- Fecha de inicio esperada: 23 de Febrero.
- Fecha de fin esperada: 20 de Marzo.

3.1.5. Fase 5: Pruebas iniciales de hardware

Aunque el proyecto sigue un enfoque secuencial basado en el modelo en cascada, esta fase se planificó para desarrollarse en paralelo con otras. Se centra en la realización de pruebas individuales sobre los distintos componentes de hardware, con el objetivo de verificar su correcto funcionamiento de forma aislada, identificar posibles limitaciones técnicas y garantizar su compatibilidad con el sistema. Esta ejecución paralela permite detectar errores tempranos y minimizar riesgos en fases posteriores, optimizando así el tiempo global del desarrollo sin comprometer la coherencia del modelo adoptado.

- Fecha de inicio esperada: 23 de Febrero.
- Fecha de fin esperada: 6 de Abril.

3.1.6. Fase 6: Implementación

Esta fase abarca el desarrollo completo del sistema. Se procederá a la implementación del bróker MQTT, los clientes encargados de publicar y suscribirse a los mensajes, y la lógica de comunicación entre los distintos componentes. También se diseñará e implementará la interfaz gráfica de usuario, permitiendo la interacción con el sistema de forma intuitiva. Al finalizar esta etapa, el sistema debe estar completamente funcional y listo para su validación final mediante pruebas completas.

- Fecha de inicio esperada: 6 de Abril.
- Fecha de fin esperada: 16 de Mayo.

3.1.7. Fase 7: Pruebas finales

Una vez finalizada la implementación del sistema, se llevará a cabo un conjunto de pruebas integrales con el objetivo de verificar su correcto funcionamiento. Estas pruebas contemplan tanto aspectos funcionales como no funcionales: se evaluará la comunicación entre dispositivos, la estabilidad del sistema, el rendimiento bajo diferentes condiciones y la usabilidad de la interfaz. El propósito es detectar posibles errores, validar el cumplimiento de los requisitos definidos y garantizar la fiabilidad del sistema antes de su entrega final.

- Fecha de inicio esperada: 16 de Mayo.
- Fecha de fin esperada: 26 de Mayo.

3.1.8. Fase 8: Producción de la memoria

Esta fase se planificó para desarrollarse en paralelo al resto del proyecto, como parte del propio modelo adoptado. La redacción de la memoria se lleva a cabo de forma progresiva, conforme se completan las distintas fases técnicas. Este enfoque permite documentar cada bloque de trabajo con mayor precisión, evitando la pérdida de detalles importantes y reduciendo la carga acumulada al final del proyecto.

Durante los primeros días se dedica tiempo a familiarizarse con LaTeX, la herramienta seleccionada para la elaboración del documento. Además, al final del cronograma se reservan varios días exclusivamente para completar secciones como el resumen, los agradecimientos, las conclusiones y las líneas futuras, así como para realizar una revisión y corrección integral del contenido final.

- Fecha de inicio esperada: 20 de Enero.
- Fecha de fin esperada: 3 de Junio.

3.2. Riesgos

En esta sección se analizarán los posibles riesgos que pueden surgir durante el desarrollo del trabajo, sus probabilides e impacto y se diseñarán acciones para mitigarlos y corregirlos de la mejor forma posible.

Riesgo R01	**Probabilidad**	**Impacto**	**Acciones Mitigantes**	**Acciones de Contingencia**
Falta de dominio en tecnologías clave (MQTT, sensores, servidor)	Media	Alto	Realizar formación previa en MQTT, servidores y sensores. Consultar documentación oficial y realizar pruebas.	Buscar ayuda en foros especializados, contactar con expertos o cambiar a tecnologías más conocidas si es viable.

Tabla 3.1: Riesgo: Dominio de la tecnología (R01)

Riesgo R02	**Probabilidad**	**Impacto**	**Acciones Mitigantes**	**Acciones de Contingencia**
El alumno está un tiempo de baja por enfermedad	Media	Medio	Planificar tareas con margen de tiempo extra para imprevistos. Mantener comunicación con el tutor para ajustes.	Reorganizar el calendario, priorizar tareas esenciales y solicitar extensión de plazos si es necesario.

Tabla 3.2: Riesgo: Baja por enfermedad (R02)

Riesgo R03	**Probabilidad**	**Impacto**	**Acciones Mitigantes**	**Acciones de Contingencia**
Confinamiento por distintos motivos	Baja	Bajo	Asegurar acceso remoto a todas las herramientas necesarias. Mantener un plan de trabajo flexible.	Reorganizar tareas para enfocarse en aquellas que se puedan realizar desde casa. Solicitar ajustes en plazos si es necesario.

Tabla 3.3: Riesgo: Confinamiento (R03)

Riesgo R04	**Probabilidad**	**Impacto**	**Acciones Mitigantes**	**Acciones de Contingencia**
No está disponible el hardware deseado	Media	Alto	Identificar alternativas compatibles con el proyecto. Verificar disponibilidad con anticipación.	Adaptar el desarrollo a otro hardware disponible.

Tabla 3.4: Riesgo: Indisponibilidad de hardware (R04)

Riesgo R05	**Probabilidad**	**Impacto**	**Acciones Mitigantes**	**Acciones de Contingencia**
Problemas con el envío del hardware (No llega o no llega en buenas condiciones)	Baja	Alto	Asignar tiempo de sobra a realizar las pruebas con los dispositivos. Comprobar las valoraciones de los productos por otros compradores.	Si algún producto llega defectuoso (o no llega) se vuelve a pedir, ya que tenemos tiempo de sobra asignado, esto no afectará a las siguientes etapas.

Tabla 3.5: Riesgo: Problemas de comunicación (R05)

Riesgo R06	**Probabilidad**	**Impacto**	**Acciones Mitigantes**	**Acciones de Contingencia**
Falta de tiempo para completar el proyecto	Baja	Alto	Planificar correctamente las tareas, estableciendo plazos realistas.	Reducir la complejidad de algunas funcionalidades o solicitar una extensión de plazo si es viable.

Tabla 3.6: Riesgo: Falta de tiempo para completar el proyecto (R06)

Riesgo R07	**Probabilidad**	**Impacto**	**Acciones Mitigantes**	**Acciones de Contingencia**
Pérdida de datos o código del proyecto	Baja	Alto	Usar control de versiones como Git. Realizar backups periódicos en la nube y localmente.	Recuperar la última copia de seguridad disponible y reconstruir el código perdido. En caso de pérdida parcial, intentar restaurar archivos con herramientas de recuperación.

Tabla 3.7: Riesgo: Pérdida de datos o código del proyecto (R07)

Riesgo R08	Probabilidad	Impacto	Acciones Mitigantes	Acciones de Contingencia
Aumento del estrés	Baja	Medio	Planificar descansos regulares. Gestionar el tiempo de manera eficiente. Mantener una buena organización y evitar la sobrecarga de tareas.	Realizar ejercicios de relajación y técnicas de manejo del estrés. Ajustar la planificación para reducir la carga de trabajo en momentos críticos.

Tabla 3.8: Riesgo: Aumento del estrés (R08)

Con los riesgos ya definidos, vamos a recalcar cuáles son los más importantes de acuerdo a la probabilidad de que ocurran y el impacto que tendrían en el proyecto. Para ello, atenderemos a la siguiente tabla:

Probabilidad	Impacto		
	Bajo	Medio	Alto
Alta	Medio-Alto	Alto	Alto
Media	Medio	Medio-Alto	Alto
Baja	Bajo	Medio	Medio-Alto

Tabla 3.9: Cruce de probabilidad e impacto para determinar el nivel de importancia del riesgo

Los riesgos que muestran un mayor nivel de importancia, y por tanto, lo que se deben tener más en cuenta son los siguientes:

- **Riesgo R01:** Falta de dominio en tecnologías clave (MQTT, sensores, servidor).
  Para prevenir este riesgo, se realizará una amplia formación en MQTT consultando otros trabajos similares, tutoriales en internet y documentación oficial

- **Riesgo R04:** No está disponible el hardware deseado.
  Se realizará el pedido del hardware lo antes posible, y se realizarán pruebas individuales básicas con el objetivo de comprobar que funciona correctamente. Las pruebas se realizarán en paralelo a otras fases del proyecto. De esta forma, si se produce algún problema con el envío o el hardware en sí mismo, no afectará al tiempo final del proyecto.

3.3. Costes

En esta sección se detallarán los costes del proyecto, incluyendo el coste de los productos hardware, el tiempo estimado de trabajo y otros costes adicionales que conlleva el trabajo.

3.3.1. Coste de hardware

Componente de Hardware	Precio (€)
Orange Pi Zero 2W 4GB + expansión de red	52.99
Sensor de temperatura DHT22 (X2)	2.72
Tarjeta MicroSd 32 GB	3.79
Adaptador microSd - USB	2.31
Sensor de movimiento HC-SR501 (X2)	2.50
Cables Dupont	2.31
ESP32-S3 WROOM + camara OV5640 (X2)	20.38
Cargador 5V 3A tipo C (X2)	8.98
Total	95.98

Tabla 3.10: Lista de componentes de hardware y su coste total

3.3.2. Coste de horas de trabajo

El coste asociado a las horas de trabajo se ha estimado tomando como referencia el salario medio de un desarrollador IoT en España, que asciende a 46 000 €/año (aproximadamente 22.12 €/hora) según datos de Glassdoor [13].

$$300 \text{ horas} \times 22{,}12\,/h = 6636$$

Este valor representa una estimación del coste que tendría el desarrollo del proyecto si se externalizara a un perfil profesional vinculado al desarrollo de soluciones IoT, permitiendo así cuantificar el esfuerzo invertido en términos económicos de forma más realista.

Perfil profesional	Precio/hora	Horas	Total (€)
Desarrollador IoT	22.12 €	300	6636.00

Tabla 3.11: Coste estimado en horas de trabajo

3.3.3. Otros costes

Como equipo de trabajo, se ha decidido usar un ordenador portátil Asus Zenbook 2021. Este ordenador tiene un coste de precio de 899 €y se estima que tiene una vida útil de 5 años. Por lo tanto su coste para el proyecto se fijará en 899 €/ 5 años / 12 meses = 14.98 €/mes. El proyecto durará 4 meses, por lo que el coste del ordenador para el proyecto será de 14.98 €/mes * 4 meses = 59.92 €.

Otro coste a tener en cuenta es la electricidad, para estimar el coste, se ha tomado como referencia el precio medio de la electricdidad en España a 0.1381 €/kWh. La energía consumida se calcula con la potencia del ordenador (65 W) y el tiempo de uso (300 horas). Por lo tanto, la energía consumida es de (65 W * 300 h) / 1000 = 19.5 kWh. Para calcular el coste final, lo multiplicamos por el precio de la electricidad: 19.5 kWh * 0.1381 €/kWh = 2.70 €. [20]

Concepto	Precio	Cantidad	Total (€)
Ordenador portátil (Asus Zenbook 2021)	14.98 €/mes	4 meses	59.92
Electricidad	0.1381 €/kWh	19.5 kWh	2.70
		Total general	**62.62**

Tabla 3.12: Costes de servicios por unidad de tiempo y total estimado

3.3.4. Coste total del proyecto

Tras haber analizado por separado los costes de hardware, el tiempo de trabajo invertido y otros gastos asociados, es posible obtener una estimación global del coste del proyecto. Esta cifra final refleja no solo los materiales necesarios, sino también el valor del tiempo, el conocimiento y el esfuerzo dedicados durante el desarrollo.

Concepto	Coste total (€)
Hardware	95.98
Horas de trabajo	6636.00
Otros costes (ordenador y electricidad)	62.62
Total del proyecto	**6794.60**

Tabla 3.13: Resumen del coste total estimado del proyecto

Como podemos observar en la tabla 3.13, el mayor gasto del proyecto corresponde al tiempo de trabajo, que representa un 97 % del total. Cabe recalcar el bajo coste en cuanto a hardware adquirido, uno de los objetivos del proyecto.

3.4. Desarrollo final del proyecto

El desarrollo del proyecto se ha ajustado satisfactoriamente a la planificación establecida, respetando tanto los plazos como la estructura secuencial de fases definida inicialmente bajo un enfoque en cascada. Cada bloque de trabajo se ha iniciado en las fechas previstas y se ha completado conforme al cronograma, sin que se hayan registrado desviaciones significativas en la duración global del proyecto.

Durante la fase de planificación se definieron con claridad los objetivos, entregables y estimaciones temporales, lo que ha facilitado una correcta gestión del tiempo y una distribución equilibrada del esfuerzo. Este planteamiento inicial ha sido determinante para poder avanzar con fluidez en las siguientes etapas, incluyendo la documentación teórica, el análisis técnico, la implementación y la elaboración de la memoria.

Las dos fases que se planificaron en paralelo —las pruebas iniciales de hardware y la redacción de la memoria— han respondido bien a esta estrategia. En concreto, la realización anticipada de pruebas sobre los componentes permitió detectar a tiempo un error en el pedido del hardware: se había adquirido por equivocación una Orange Pi Zero 2W con 1GB de RAM en lugar de los 4GB necesarios. Al tratarse de

una fase ejecutada en paralelo al análisis y diseño, y al haberse previsto un margen de tiempo suficiente, este incidente se resolvió sin afectar al desarrollo general. Se gestionó un nuevo pedido correctamente y se pudo continuar con normalidad.

Este tipo de problema se había contemplado como riesgo R05: 3.5 en la planificación inicial ("Problemas con el envío del hardware"), y contaba con acciones mitigantes y de contingencia definidas. En particular, se había anticipado la posibilidad de recibir productos defectuosos o incorrectos, y se había asignado tiempo extra precisamente para este tipo de eventualidades. Esta preparación previa permitió actuar con rapidez y eficacia, demostrando la utilidad de una gestión de riesgos bien planteada.

El resto del proyecto se ha desarrollado sin incidencias graves. La carga de trabajo ha sido constante pero asumible, en parte gracias a una estimación realista del tiempo efectivo disponible (aproximadamente 100 días reales de trabajo dentro de una duración total de 134 días). La planificación detallada, junto con el uso de herramientas como el diagrama de Gantt, ha facilitado la supervisión y el control del progreso en todo momento.

En cuanto a la memoria del proyecto, su elaboración paralela al desarrollo técnico ha permitido documentar cada fase en el momento en que se realizaba, lo que ha contribuido a reducir la carga de trabajo final y ha evitado posibles omisiones. Esta estrategia ha resultado especialmente útil para mantener un equilibrio entre el desarrollo práctico y la calidad de la documentación.

En conjunto, se puede afirmar que el proyecto ha evolucionado según lo previsto, cumpliendo los hitos en tiempo y forma, y resolviendo los imprevistos dentro de los márgenes definidos. La previsión de riesgos, junto con la flexibilidad introducida en ciertas fases, ha sido clave para garantizar una evolución estable y controlada del trabajo.

Capítulo 4

Análisis y diseño del sistema

4.1. MQTT en detalle

En esta sección se explicará cómo se aplica el protocolo MQTT en nuestra sistema, se explicará los tópicos que se van a usar, los mensajes que se van a publicar y el nivel de calidad de servicio (QoS) que se va a usar.

4.1.1. Topics

A la hora de diseñar el sistema, se ha optado por una solución muy sencilla en la que cada ESP32 publicará los datos de los sensores de la siguiente forma: 'baseX'/'nombreSensor'. Por ejemplo: 'base1/temperatura'.

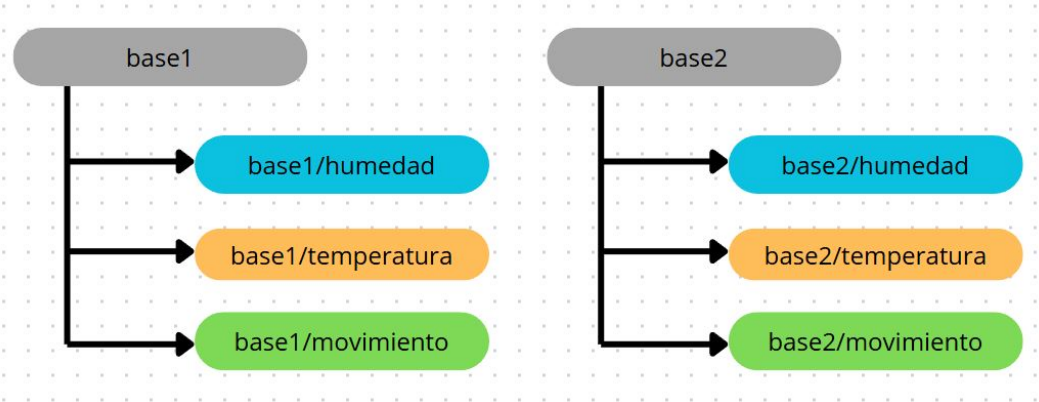

Figura 4.1: Tópicos de los sensores

En la figura 4.1 se puede observar la estructura de los tópicos elegida. Cada 'base' corresponde a un ESP32. Y en cada ESP32 los sensores publican de la siguiente forma:

- HC-SR501: 'baseX/movimiento'
- DHT22: 'baseX/temperatura' y 'baseX/humedad'

4.1.2. Mensajes

Procedemos con la descripción de los mensajes que se van a publicar en cada uno de los tópicos. En la siguiente tabla se pueden observar las características de los mensajes que se van a publicar.

Topic	Tipo de mensaje	Descripción	Ejemplos de mensaje
`baseX/movimiento`	Texto (`ALERTA` / `NO_ALERTA`)	El sensor HC-SR501 devuelve 'HIGH' or 'LOW' y el código lo adapta para publicar el texto correspondiente	`'ALERTA','NO_ALERTA'`
`baseX/temperatura`	Número (texto, 2 decimales)	Valor de la temperatura medido por el sensor DHT22	`'24.53','30.12'`
`baseX/humedad`	Número (texto, 2 decimales)	Valor de la humedad medido por el sensor DHT22	`'58.47','65.22'`

Tabla 4.1: Topics MQTT y características de los mensajes publicados.

4.1.3. QoS

En cuanto a la selección del nivel de calidad de servicio (QoS), se ha optado por el QoS 0 para los mensajes de temperaturay humedad, ya que no es necesario asegurarse de que el mensaje llegue al broker, ya que sino llega, el ESP32 volverá a publicar en un intervalo de tiempo determinado. Respecto a la información del sensor de movimiento, es importante que no se pierda porque es una parte vital del sistema. Además, tampoco podemos permitirnos la recepción de duplicados porque no podríamos diferenciar si el sensor sigue detectando movimiento o es un duplicado del anterior. Por lo tanto, se ha optado por el QoS 2, que garantiza la entrega del mensaje exactamente una vez.

Con esta elección de QoS, no sobrecargamos el sistema en exceso, ya que la temperatura y la humedad no son datos críticos y no es necesario asegurarse de que lleguen al broker, por lo que les podemos asignar un QoS menos pesado para el sistema. Por otro lado, el sensor de movimiento es un dato crítico y no podemos permitirnos perderlo ni recibir duplicados.

4.2. Arquitectura general

En la sección anterior se ha descrito cómo se comunican los nodos ESP32 con el bróker MQTT, lo que constituye la base del sistema de adquisición y transmisión de datos de sensores. Sin embargo, dicha comunicación no cubre el flujo de información relacionado con las imágenes capturadas por las cámaras. En esta sección se describe la arquitectura general completa del sistema, integrando tanto el canal MQTT como el mecanismo de captura y visualización de imágenes.

Inicialmente, el diseño del sistema preveía que cada ESP32 ofreciera un endpoint HTTP que sirviera las imágenes en streaming desde su propia dirección IP. Node-RED, ejecutado en la Orange Pi, accedería a este flujo y lo mostraría en su dashboard. Esta solución, aunque conceptualmente válida, presentó diversos problemas técnicos durante la implementación.

Debido a esas dificultades, se optó por una solución alternativa más estable, que se detalla en la sección 5.6.

Con la solución definitiva aplicada, el sistema queda representado en la Figura 4.2:

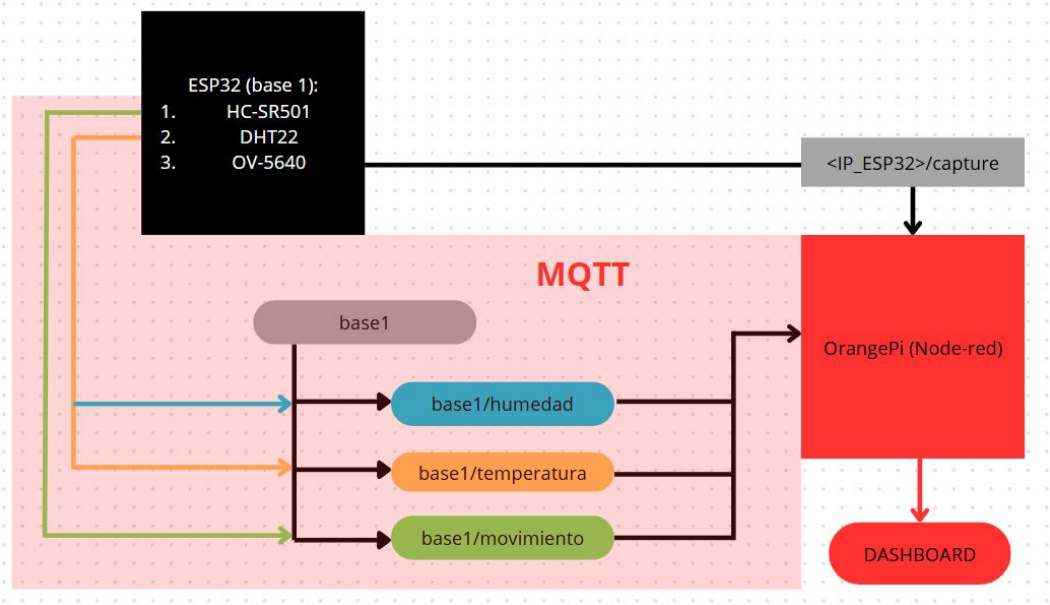

Figura 4.2: Arquitectura general del sistema

Para una mayor claridad visual, la figura muestra únicamente un nodo ESP32 con sus sensores y su cámara. En la práctica, el sistema está compuesto por dos nodos idénticos que operan de forma paralela e independiente, pero utilizando la misma arquitectura.

Se distinguen claramente dos subsistemas:

- **Subsistema MQTT:** Cada ESP32 está conectado a sensores de movimiento (HC-SR501), temperatura y humedad (DHT22). Los datos obtenidos son publicados periódicamente en el bróker Mosquitto, alojado en la Orange Pi, utilizando una estructura jerárquica de topics MQTT. Node-RED se suscribe a estos topics para procesar y mostrar la información en el dashboard.

- **Subsistema de imágenes:** Node-RED también se encarga de mostrar imágenes capturadas por la cámara OV5640 conectada al ESP32. Para ello, se define una ruta en Node-RED (por ejemplo, `/camara1.jpg`), que actúa como proxy. Cada vez que el navegador accede a esta ruta, Node-RED realiza una petición HTTP al ESP32 en la ruta `/capture`, recibe la imagen y la entrega al cliente. Este mecanismo, ejecutado de forma periódica (cada 0.5 segundos), permite simular un flujo de vídeo mediante la sucesión de imágenes estáticas.

Ambos subsistemas convergen en la Orange Pi, donde Node-RED actúa como núcleo de procesamiento y visualización, proporcionando al usuario una interfaz centralizada desde la que supervisar en tiempo real todos los parámetros del sistema.

4.3. Requisitos del sistema

4.3.1. Requisitos funcionales

Los requisitos funcionales describen las funciones específicas que debe realizar el sistema para cumplir sus objetivos. A continuación se enumeran los principales requisitos funcionales del sistema de domótica desarrollado:

1. **RF01:** El sistema debe detectar movimiento mediante sensores PIR conectados a los nodos ESP32.
2. **RF02:** El sistema debe medir y publicar datos de temperatura y humedad a través del sensor DHT22 integrado en cada nodo ESP32.
3. **RF03:** El sistema debe capturar imágenes utilizando una cámara OV5640 en los nodos ESP32.
4. **RF04:** Los nodos ESP32 deben publicar los datos recogidos (movimiento, temperatura, humedad) en topics MQTT específicos del sistema.
5. **RF05:** El broker MQTT (Mosquitto) debe recibir y distribuir los mensajes publicados por los nodos ESP32 a los clientes suscritos.
6. **RF06:** Node-RED, como cliente MQTT, debe suscribirse a los topics y procesar los datos recibidos para visualización, almacenamiento o actuación.
7. **RF07:** El sistema debe generar alertas en la interfaz gráfica cuando se detecte movimiento en alguno de los nodos.
8. **RF08:** El sistema debe estar operativo de forma continua y tolerar reinicios de los componentes sin pérdida de funcionalidad.
9. **RF09:** El sistema debe implementar seguridad en las comunicaciones mediante cifrado TLS en las conexiones MQTT.
10. **RF10:** El sistema debe utilizar una red privada virtual (VPN) para garantizar el acceso seguro a la red desde el exterior.
11. **RF11:** Cada nodo debe publicar los datos en intervalos definidos, incluso si no se detecta movimiento, para confirmar su operatividad.
12. **RF12:** El sistema debe estructurar los topics MQTT de forma jerárquica y modular para facilitar su gestión y escalabilidad.

4.3.2. Requisitos no funcionales

Los requisitos no funcionales definen criterios de calidad y restricciones del sistema que no están directamente relacionados con funciones específicas, pero que son esenciales para su correcto funcionamiento, mantenimiento y escalabilidad. A continuación, se enumeran los principales requisitos no funcionales del sistema desarrollado:

1. **RNF01: Disponibilidad:** El sistema debe estar disponible de forma continua durante al menos el 95 % del tiempo de operación previsto.
2. **RNF02: Escalabilidad:** El sistema debe permitir la incorporación de nuevos nodos ESP32 sin modificar la lógica principal ni el esquema de topics.
3. **RNF03: Mantenibilidad:** La configuración del sistema (topics, scripts, flujo de Node-RED, etc.) debe estar documentada para facilitar su mantenimiento y actualización futura.
4. **RNF04: Compatibilidad:** El sistema debe ser compatible con otros dispositivos IoT que soporten MQTT 3.1.1 o superior.
5. **RNF05: Seguridad:** Todas las comunicaciones MQTT deben estar cifradas mediante TLS, y el acceso remoto a la Orange Pi debe realizarse únicamente a través de una VPN.
6. **RNF06: Integridad:** Los datos transmitidos por los sensores no deben ser alterados durante el tránsito. El sistema debe garantizar la integridad mediante TLS y control de errores en los clientes.
7. **RNF07: Fiabilidad:** El sistema debe tolerar desconexiones temporales de los nodos sin pérdida de configuración ni necesidad de reinicio manual del broker o clientes.
8. **RNF08: Usabilidad:** La interfaz gráfica proporcionada por Node-RED debe ser intuitiva y permitir una monitorización clara del estado del sistema y los eventos capturados.
9. **RNF09: Consumo energético:** Los nodos ESP32 deben operar con un consumo mínimo.

4.4. Modelo de dominio

El modelo de dominio representa conceptualmente las entidades principales que forman parte del sistema domótico y las relaciones existentes entre ellas. Este modelo permite abstraer la estructura lógica del sistema, facilitando la comprensión de su diseño y su futura mantenibilidad.

En la Figura 4.3 se muestra una representación simplificada del modelo de dominio del sistema. Se identifican tanto los dispositivos físicos como los componentes lógicos implicados en la gestión y monitorización de datos.

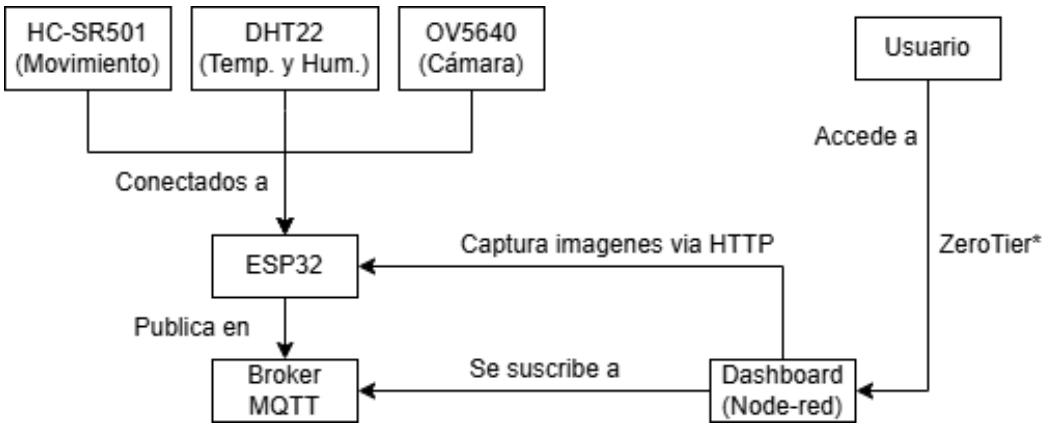

Figura 4.3: Modelo de dominio del sistema

Las principales entidades del sistema son:

- **ESP32:** Dispositivo encargado de recoger datos del entorno mediante sensores y capturar imágenes. Publica dicha información mediante MQTT.
- **Sensores:** Componentes físicos conectados al ESP32. Pueden ser de tipo temperatura y humedad (DHT22) o de detección de movimiento (HC-SR501).
- **Cámara:** Módulo OV5640 conectado al ESP32 que permite capturar imágenes bajo demanda, accesibles a través de peticiones HTTP.
- **Broker MQTT:** Nodo central (Mosquitto en la Orange Pi) encargado de recibir y distribuir los mensajes publicados por los nodos.
- **Dashboard (Node-RED):** Interfaz gráfica web que actúa como cliente MQTT y consumidor HTTP. Permite visualizar los datos de sensores y las imágenes de las cámaras.
- **Usuario:** Actor externo que accede al dashboard y supervisa el estado del sistema en tiempo real.

Este modelo refleja una arquitectura distribuida y modular, donde las entidades están desacopladas gracias al uso de MQTT y servicios HTTP, favoreciendo así la escalabilidad del sistema.

Se puede observar que en la conexión entre el usuario y el dashboard aparece indicado ZeroTier*. Esto se debe a que, al tratarse de una Orange Pi conectada en red local, el acceso remoto requiere una VPN. En este caso se ha utilizado ZeroTier, que permite conectar al usuario con la interfaz de Node-RED desde el exterior de forma segura.

4.5. Modelo de casos de uso

A continuación se presenta un diagrama general de casos de uso que resume las principales interacciones entre el usuario y el sistema. Se identifican dos funcionalidades principales: la visualización de datos en tiempo real y la visualización de imágenes en el dashboard. Esta última incluye internamente la captura de imágenes desde la cámara del nodo ESP32.

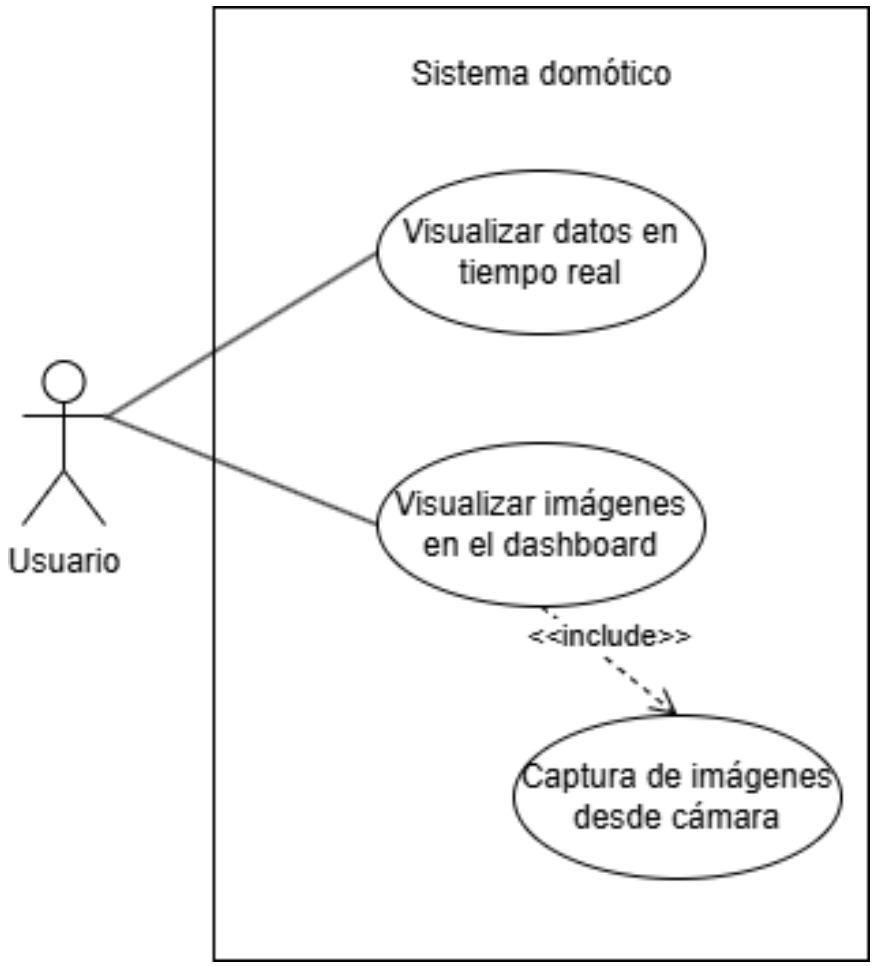

Figura 4.4: Diagrama de casos de uso

4.5.1. Visualizar datos en tiempo real

Este caso de uso describe el proceso completo mediante el cual el sistema recoge datos de sensores (temperatura, humedad y movimiento), los publica mediante MQTT y los visualiza en la interfaz de Node-RED, accesible por el usuario.

- **Nombre:** Lectura, publicación y visualización de datos de sensores
- **Actores:** Nodo ESP32, Broker MQTT (Mosquitto), Node-RED, Usuario
- **Descripción:** El nodo ESP32 recoge los datos desde los sensores conectados (DHT22 para temperatura y humedad, HC-SR501 para movimiento), y los publica en los topics MQTT correspondientes. Node-RED, actuando como cliente MQTT, los recibe y los presenta gráficamente. El usuario, al acceder al dashboard, puede visualizar los datos en tiempo real sin necesidad de realizar ninguna acción.
- **Precondiciones:**
  - El nodo ESP32 debe estar correctamente conectado a la red y a los sensores.

- El broker MQTT debe estar en ejecución y accesible.
- Node-RED debe estar suscrito a los topics correspondientes.

- **Postcondiciones:**
  - Los datos deben visualizarse correctamente en la interfaz de usuario.
- **Flujo principal:**
  1. El sensor DHT22 mide temperatura y humedad.
  2. El sensor PIR detecta (o no) movimiento.
  3. El ESP32 publica los valores en los topics:
     - `baseX/temperatura`
     - `baseX/humedad`
     - `baseX/movimiento`
  4. El broker MQTT (Mosquitto) distribuye los mensajes.
  5. Node-RED los recibe.
  6. Node-RED procesa los datos (movimiento) y los muestra en el dashboard.
  7. El usuario visualiza la información actualizada al acceder al dashboard.
- **Excepciones:**
  - Si el ESP32 pierde conexión con el broker, intentará reconectarse automáticamente.
  - Si Node-RED no está conectado, los datos no se visualizarán hasta que se restablezca.

Diagrama de secuencia

La Figura 4.5 muestra el proceso mediante el cual el nodo ESP32 recoge datos de sensores y los publica en diferentes topics MQTT. Node-RED, como cliente suscrito, recibe estos datos y actualiza automáticamente la interfaz que el usuario tiene abierta en el dashboard.

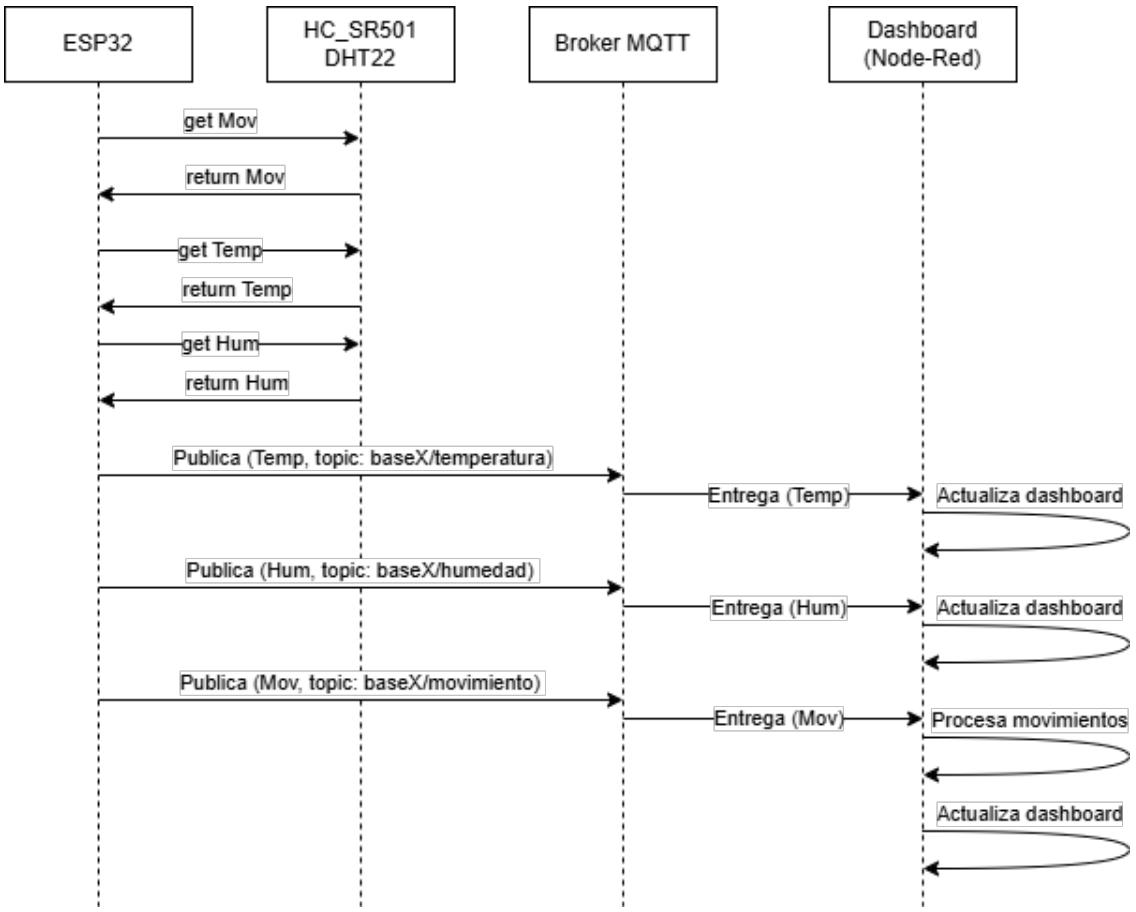

Figura 4.5: Diagrama de secuencia del proceso de visualización de datos en tiempo real

Este flujo se ejecuta de forma periódica aproximadamente cada 500 ms desde el bucle principal del código cargado al ESP32.

4.5.2. Visualizar imágenes en el dashboard

Este caso de uso describe el proceso mediante el cual el sistema muestra imágenes actualizadas periódicamente en el dashboard de Node-RED. El usuario no interactúa activamente, sino que simplemente accede a una interfaz donde se renderizan imágenes capturadas automáticamente desde el nodo ESP32.

- **Nombre:** Visualización de imágenes en el dashboard
- **Actores:** Usuario, Node-RED, Nodo ESP32 con cámara OV5640
- **Descripción:** Al acceder al dashboard de Node-RED, la interfaz activa automáticamente un flujo que realiza peticiones periódicas al ESP32. Este, a su vez, captura imágenes con su cámara y las devuelve como respuesta. Mientras el usuario mantiene abierta la vista del dashboard, las imágenes se actualizan de forma continua; si no hay ningún usuario conectado, no se realiza ninguna captura.
- **Precondiciones:**
  - Node-RED debe estar en funcionamiento.

  - La cámara del ESP32 debe estar conectada y operativa.
  - El flujo de Node-RED debe estar configurado para reaccionar a peticiones del dashboard.

- **Postcondiciones:**
  - Las imágenes se muestran en el dashboard con frecuencia regular mientras haya usuarios conectados.

- **Flujo principal:**
  1. El usuario accede al dashboard de Node-RED (`/ui`).
  2. Un componente de la interfaz lanza peticiones periódicas (cada 0.5 segundos) a la ruta `/camara1.jpg`.
  3. Node-RED intercepta la petición con un nodo `http in`.
  4. Node-RED realiza una solicitud HTTP al nodo ESP32 (`/capture`).
  5. El ESP32 responde con una imagen JPEG recién capturada.
  6. Node-RED reenvía la imagen como respuesta al navegador del usuario.
  7. El dashboard actualiza la imagen mostrada.

- **Excepciones:**
  - Si el ESP32 no responde, puede mostrarse una imagen anterior o una imagen vacía.
  - Si Node-RED está detenido, no se podrán realizar peticiones ni mostrar imágenes.

Diagrama de secuencia

En la Figura 4.6 se representa el flujo de comunicación entre los elementos implicados en la visualización de imágenes. El proceso se inicia cuando el usuario accede al dashboard, momento en el que se activan peticiones automáticas hacia una ruta de Node-RED que actúa como intermediario. Esta ruta realiza a su vez una solicitud al nodo ESP32, el cual captura una imagen y la devuelve como respuesta. Finalmente, la imagen se muestra en el dashboard.

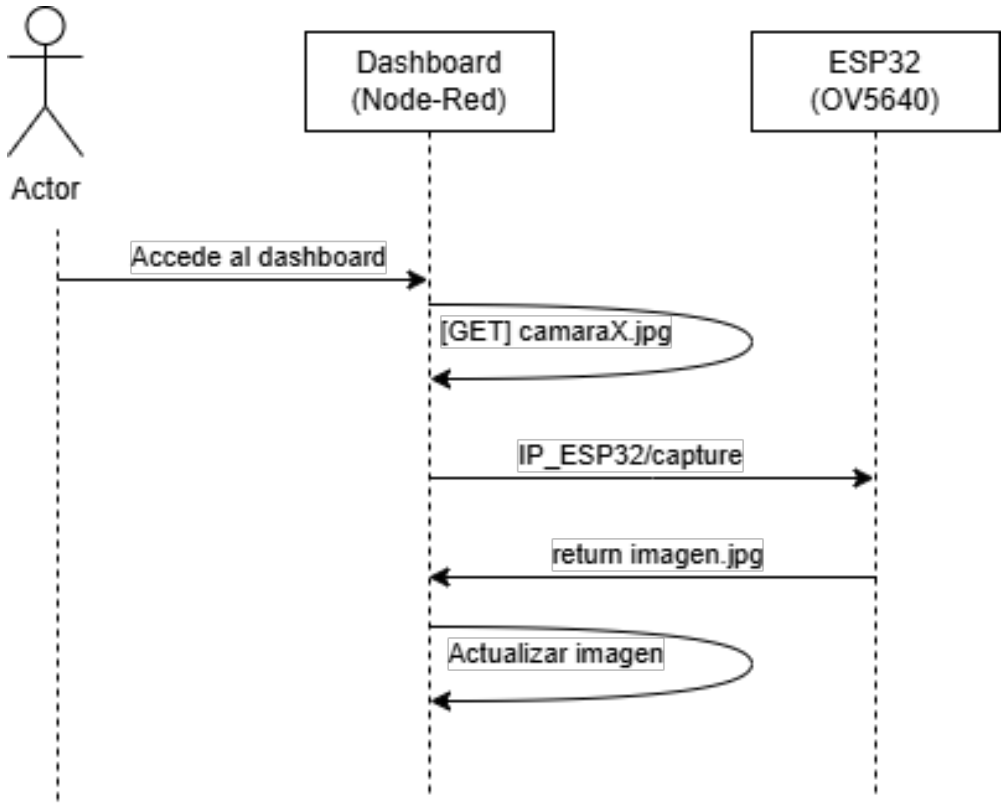

Figura 4.6: Diagrama de secuencia del proceso de visualización de imágenes

Este flujo se ejecuta de forma periódica, aproximadamente cada 0.5 segundos, mientras el usuario mantiene abierta la interfaz del dashboard.

4.6. Diseño del dashboard

El diseño del dashboard se planteó desde las primeras fases del desarrollo como un componente esencial del sistema, ya que representa el punto de interacción principal entre el usuario y la infraestructura domótica. El objetivo era construir una interfaz clara, accesible y funcional, que permitiese monitorizar el estado de los sensores en tiempo real y visualizar las imágenes capturadas por las cámaras integradas en los nodos ESP32.

Objetivos de diseño

Los principales criterios establecidos durante el diseño del dashboard fueron:

- **Modularidad visual:** Dividir la interfaz en bloques independientes para cada nodo ESP32, de modo que se puedan visualizar los datos y el estado de cada base por separado.
- **Claridad informativa:** Priorizar la representación sencilla y directa de la información relevante: temperatura, humedad, alertas de movimiento e imágenes.

- **Actualización automática:** Garantizar que los datos mostrados se actualicen sin intervención del usuario, empleando suscripciones MQTT y peticiones periódicas HTTP.
- **Compatibilidad multiplataforma:** Asegurar que el diseño sea responsive y pueda consultarse tanto desde ordenadores como desde dispositivos móviles.
- **Jerarquía visual:** Utilizar colores y tipografías que permitan al usuario identificar rápidamente estados de alerta o información destacada.

Estructura propuesta

Siguiendo los objetivos anteriores, se definió una disposición basada en cuadros visuales independientes, uno por cada nodo ESP32, denominados *Base 1* y *Base 2*. Cada uno de estos cuadros agrupa la información correspondiente a su nodo de forma clara y compacta.

En dispositivos de escritorio, ambos cuadros se dispondrán horizontalmente, permitiendo una visión simultánea de las dos bases. En cambio, en dispositivos móviles, la interfaz se adapta mediante diseño responsive, colocando los cuadros de forma vertical y permitiendo la navegación mediante desplazamiento (*scroll*) hacia abajo.

Cada cuadro de base está compuesto por los siguientes bloques de información:

- **Indicadores de temperatura y humedad:** Valores actualizados en tiempo real mediante MQTT, presentados con unidades claras y acompañados de iconos representativos.
- **Historial de detección de movimiento:** Tabla que muestra las tres últimas detecciones registradas, indicando fecha y hora.
- **Visualización de cámara:** Imagen capturada por el nodo ESP32 correspondiente, actualizada cada 0.5 segundos mediante una petición HTTP automática.

Este diseño modular y adaptativo permite al usuario supervisar el sistema desde cualquier dispositivo con una experiencia de uso fluida y sin perder funcionalidad.

Bocetos del diseño propuesto

A continuación se muestran tres ilustraciones que permiten visualizar el diseño planteado para el dashboard:

Figura 4.7: Cuadro de una base

Figura 4.8: Disposición de los cuadros en pantalla de ordenador

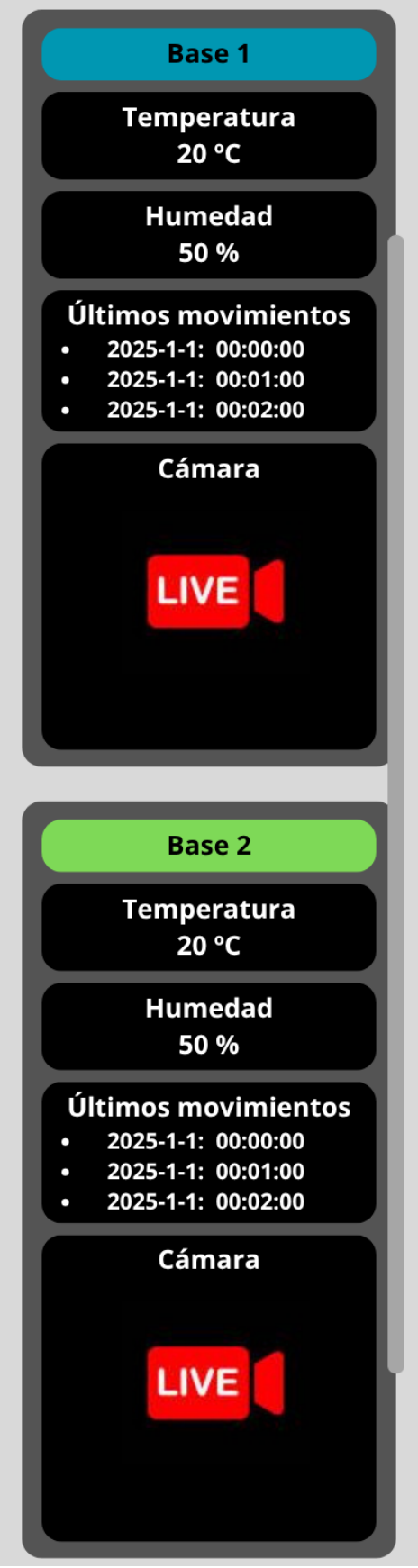

Figura 4.9: Disposición de los cuadros en pantalla de móvil

- La Figura 4.7 muestra un esquema individual de uno de los cuadros que representa una base.
- La Figura 4.8 representa cómo se dispondrían los cuadros de ambas bases en una pantalla de ordenador, colocados horizontalmente para facilitar la supervisión simultánea.
- La Figura 5.20 muestra la distribución vertical de los cuadros en un dispositivo móvil, donde el usuario podría desplazarse verticalmente para visualizar cada base.

4.7. Componentes hardware del sistema

4.7.1. Orange Pi Zero 2W

Se usará un mini ordenador Orange Pi Zero 2W como servidor MQTT. Las principales características de esta placa son:

- CPU: H618 Quad-Core 64-bit. 1.5 GHz high-performance Cortex-A53 processor
- GPU: Mali-G31 MP2
- RAM: 4 GB LPDDR4
- Almacenamiento: TF card slot, en nuestro caso se usará una tarjeta microSD de 32 GB.
- Video Output: Mini HDMI 2.0.
- Cargador: Tipo C - 5V/2A.
- Conectividad: 2 x USB 2.0 Tipo C.
- Expansión: Se ha optado por usar la placa de expansión que contiene 2 puertos USB 2.0 y un adaptador de red Ethernet

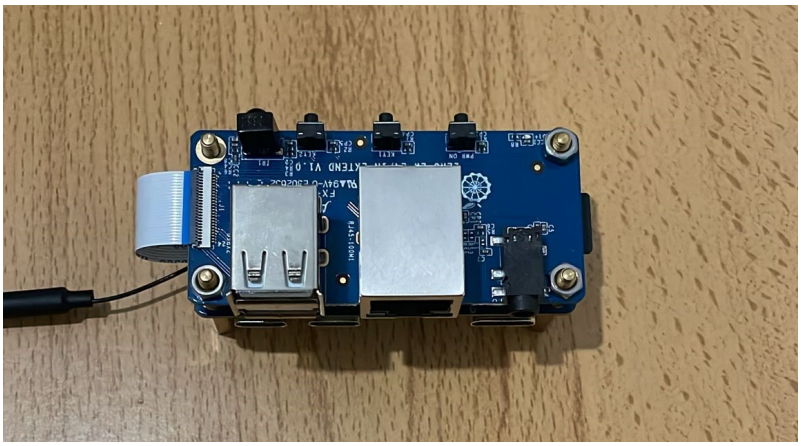

Figura 4.10: Orange Pi Zero 2W - vista de la placa de expansión

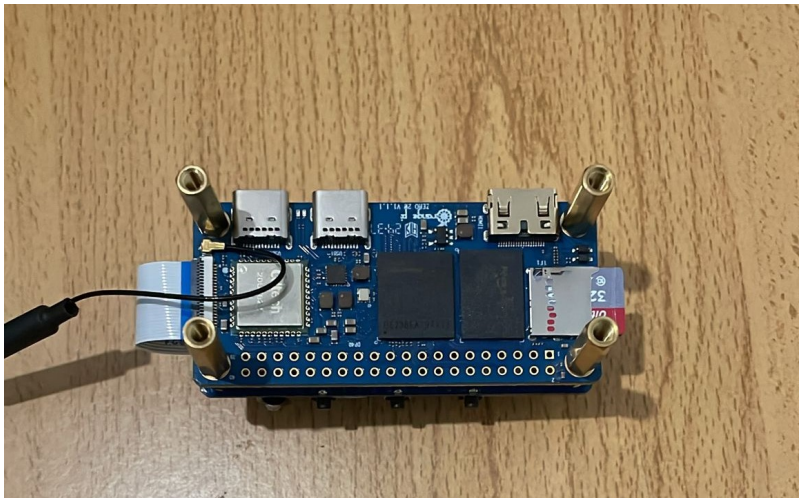

Figura 4.11: Orange Pi Zero 2W - vista de la placa base

En ambas imágenes se puede observar la placa base de la Orange Pi ya conectada a la placa de expansión y con la tarjeta microSD insertada.

Cabe destacar que la expansión es necesaria para poder conectar la Orange Pi a la red local y programarla desde el ordenador pórtatil. Otra opción hubiese sido comprar un adaptador para el Mini HDMI y realizar la programación desde un monitor, con teclado y ratón.

Enlace de compra: `https://es.aliexpress.com/item/1005006064209654.html`

4.7.2. Esp32W-S3 WROOM

El ESP32-WROOM-32 es una placa de desarrollo basada en el módulo ESP32, tiene un procesador Xtensa dual-core 32-bit LX7 CPU, con una frecuencia de 240 MHz. En cuanto al almacenamiento, tiene 384 KB de memoria ROM, 512 KB de SRAM, 16 KB de RTCSRAM y 8 MB de PSRAM. Trabaja con un voltaje entre 3 y 3.6 V. En el apartado de conectividad inalambrica, tiene Wi-Fi y Bluetooth LE. En este proyecto usaremos la conectividad a través de Wi-Fi.

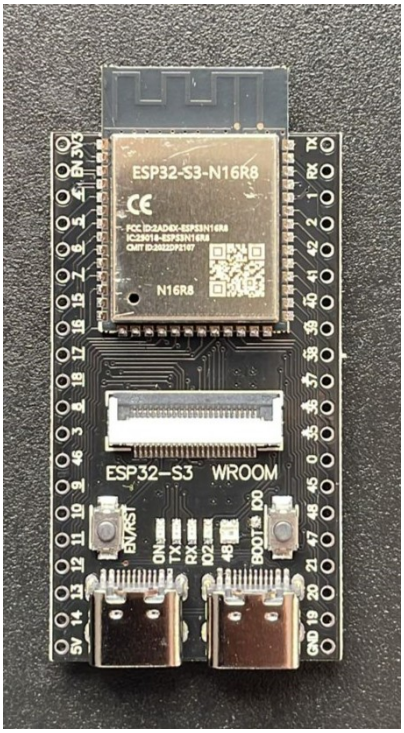

Figura 4.12: ESP32

Enlace de compra: `https://es.aliexpress.com/item/1005007308040075.html`

4.7.3. Cámara OV5640

Para la captura de imágenes se ha utilizado el módulo de cámara OV5640, compatible con el ESP32-S3. Este sensor de imagen es ampliamente utilizado en sistemas embebidos debido a su bajo consumo, resolución adecuada y compatibilidad con microcontroladores.

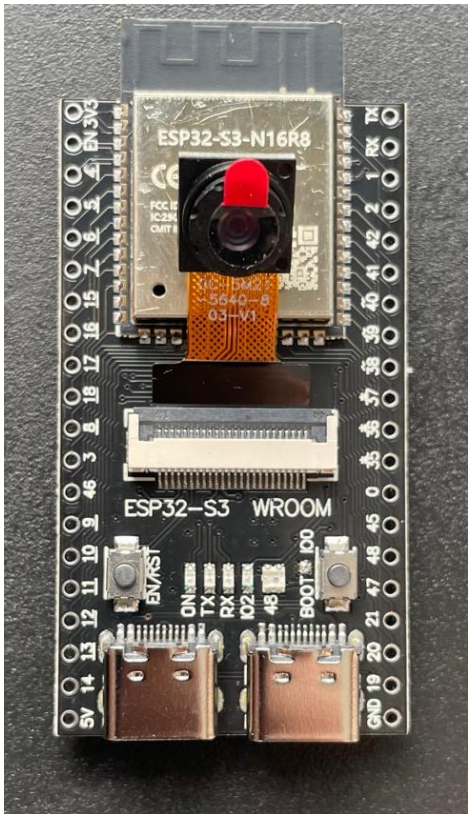

Figura 4.13: Módulo de cámara OV5640

La cámara OV5640 incorpora un sensor CMOS de 5 megapíxeles capaz de capturar imágenes fijas a una resolución máxima de 2592x1944 píxeles, aunque en este proyecto se ha trabajado con resoluciones inferiores para optimizar el rendimiento del sistema. El módulo se conecta al ESP32 a través de la interfaz DCMI (Digital Camera Interface) utilizando pines específicos para datos, reloj y sincronización.

Entre las principales características técnicas destacan:

- Sensor de imagen: CMOS de 5MP (1/4”)
- Resolución máxima: 2592x1944 píxeles
- Interfaz: DCMI o SCCB/I2C (control)
- Formatos de imagen soportados: JPEG, RGB, YUV
- Control de exposición y balance de blancos automático

En el sistema desarrollado, la cámara se mantiene en espera y sólo captura una imagen cuando Node-RED realiza una petición HTTP a la ruta ‘/capture‘ del ESP32. Esta arquitectura permite prescindir del uso de streaming continuo, reduciendo así el uso de ancho de banda y procesamiento. Node-RED actúa como intermediario: recibe la imagen desde el ESP32 y la entrega al navegador del usuario como una respuesta binaria a la ruta ‘/camara1.jpg‘.

Enlace de compra: `https://es.aliexpress.com/item/1005007308040075.html`

4.7.4. Sensores de temperatura y humedad DHT22

Para medir la temperatura y la humedad usaremos el sensor DHT22.

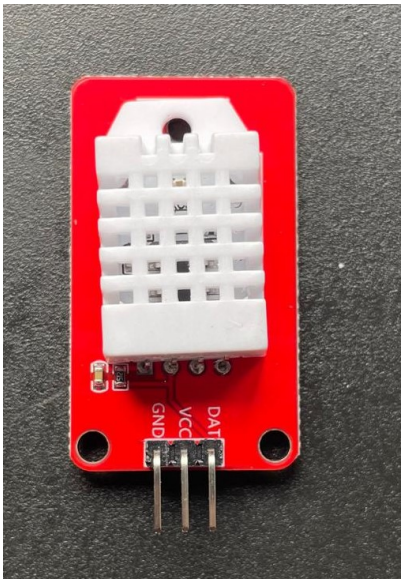

Figura 4.14: Sensor DHT22 con módulo AM2302

Este sensor es capaz de medir temperaturas desde -40ºC hasta 80ºC con una precisión de ±0.5°C y humedad relativa desde 20 % hasta 90 % con una precisión de ±2 %. Este sensor originalmente tiene 4 pines de conexión, pero en la versión que usaremos, el DHT22 con módulo AM2302, tiene 3 pines de conexión, VCC, GND y DATA. Además, este módulo trae incorporada una resistencia de pull-up de 10kΩ en el pin DATA. Por lo tanto la forma de conectarlo con un ESP32 es la siguiente:

DHT22	ESP32
VCC	3.3V
GND	GND
DATA	GPIO

Tabla 4.2: Conexión del sensor DHT22 con el ESP32

Enlace de compra: `https://es.aliexpress.com/item/1005007632667715.html`

4.7.5. Sensores de movimiento HC-SR501

Con el objetivo de detecar movimiento para activar la cámara, usaremos el sensor de movimiento HC-SR501.

Figura 4.15: Sensor de movimiento HC-SR501

Este sensor tiene un rango de detección entre 3 y 7 metros y un ángulo de detección de 120º. Cabe destacar que el rango de detección es ajustable. El sensor tiene 2 modos de funcionamiento principales:

- **Un solo disparo (Trigger mode)**: El sensor se activa cuando detecta movimiento y mantiene activa la señal HIGH durante el tiempo que se haya programado. Después de ese tiempo, la señal vuelve a LOW y se mantiene así hasta que detecta un nuevo movimiento.
- **Disparo continuo (Retriggerable mode)**: En este modo,cada vez que el sensor detecta movimiento, el sensor activa la señal HIGH. Si sigue detectando movimiento, mantendrá la señal HIGH continuamente hasta que deje de detectar movimiento.

Este sensor tiene 3 pines de conexión, VCC, GND y OUT. La conexión con el ESP32 es la siguiente:

HC-SR501	ESP32
VCC	3.3V
GND	GND
OUT	GPIO

Tabla 4.3: Conexión del sensor HC-SR501 con el ESP32

Enlace de compra: `https://es.aliexpress.com/item/1005006039243790.html`

4.8. Entorno software y tecnologías utilizadas

4.8.1. Programa para grabar la imagen: Balena Etcher

Para grabar la imagen del sistema operativo Armbian en la tarjeta microSD que se utilizará en la Orange Pi, se ha empleado el programa *Balena Etcher*. Este software es de código abierto, multiplataforma y gratuito, y permite la creación de medios de arranque de forma sencilla y segura. [4]

Balena Etcher presenta una interfaz gráfica minimalista que guía al usuario a través de tres pasos: selección de la imagen del sistema (por ejemplo, un archivo `.img` o `.iso`), selección del dispositivo de destino (la tarjeta microSD) y ejecución del proceso de grabación mediante la opción *Flash!*.[24]

Entre sus ventajas destacan la verificación automática de la imagen tras el proceso de escritura, la detección automática de unidades extraíbles y la protección frente a la sobreescritura del disco principal del sistema.

Gracias a su compatibilidad con Windows, macOS y Linux, se convierte en una herramienta muy versátil en entornos de desarrollo multiplataforma.

4.8.2. Sistema operativo: Armbian

Para el sistema operativo de la Orange Pi Zero 2W se ha optado por utilizar *Armbian*, una distribución ligera basada en Debian [9], diseñada específicamente para placas de desarrollo como Orange Pi, Banana Pi y Odroid [23].

En concreto, se ha seleccionado la versión *Minimal/IoT*, que proporciona un entorno sin interfaz gráfica, ideal para sistemas embebidos y aplicaciones donde se prioriza la eficiencia de recursos. Esta elección permite un menor consumo de CPU y RAM, lo que optimiza el rendimiento en tareas como la ejecución del broker MQTT y el entorno Node-RED [2].

Una característica relevante de esta versión es el uso de `systemd-networkd` como gestor de red en lugar de `NetworkManager`, lo que proporciona una configuración más ligera y eficiente para entornos sin escritorio. Esta configuración se adapta perfectamente al enfoque del proyecto, donde toda la interacción con la placa se realiza desde un equipo con Windows a través de la red local, sin necesidad de periféricos conectados a la Orange Pi.

Además, Armbian cuenta con una comunidad activa de desarrolladores, lo que garantiza soporte continuo, actualizaciones frecuentes y acceso a un amplio repositorio de paquetes. Al ser de código abierto y estar basado en Debian, hereda también su estabilidad y compatibilidad con herramientas estándar del ecosistema GNU/Linux [3].

4.8.3. Plataforma de desarrollo: Arduino IDE

Para el desarrollo del software de los nodos ESP32, encargados de gestionar los sensores, capturar imágenes y publicar datos mediante MQTT, se ha utilizado el entorno de desarrollo *Arduino IDE*. Este entorno es ampliamente utilizado en proyectos de electrónica y sistemas embebidos, debido a su sencillez y facilidad de integración con múltiples plataformas de hardware [25].

Arduino IDE permite escribir, compilar y cargar programas directamente en microcontroladores como el ESP32. Es un entorno multiplataforma y de código abierto, compatible con Windows, macOS y Linux[1].

El lenguaje de programación utilizado es el lenguaje Arduino, una variante simplificada de C++ que proporciona un conjunto de funciones accesibles para el control de pines, sensores y comunicaciones. Esta abstracción facilita el desarrollo de prototipos rápidos y es suficiente para cubrir las necesidades funcionales del sistema diseñado.

Además, el ecosistema de Arduino cuenta con una amplia comunidad y numerosas librerías ya desarrolladas (como ‘WiFi.h‘ o ‘PubSubClient.h‘), lo que agiliza significativamente el proceso de programación y depuración.

4.8.4. Broker MQTT: Mosquitto

Para la implementación del protocolo MQTT en la red local del sistema, se ha utilizado *Mosquitto*, un broker MQTT de código abierto, ligero y altamente eficiente. Mosquitto es compatible con las versiones 5.0, 3.1.1 y 3.1 del estándar MQTT, lo que lo convierte en una solución robusta y ampliamente adoptada en sistemas IoT [10].

Una de sus principales ventajas es su bajo consumo de recursos, lo que permite ejecutarlo sin problemas en dispositivos de hardware limitado como la Orange Pi Zero 2W, donde se ha instalado sobre el sistema operativo Armbian. Además, al ser multiplataforma, puede integrarse fácilmente con otros sistemas, como los nodos ESP32.

Los ESP32, por su parte, se conectan al broker mediante la librería 'PubSubClient', que implementa el cliente MQTT compatible con Mosquitto y facilita la publicación y suscripción a topics desde el entorno de desarrollo Arduino IDE. Esta arquitectura permite establecer una comunicación eficiente y en tiempo real entre los sensores y la interfaz de usuario.

4.8.5. TLS: Cifrado de las comunicaciones

TLS (Transport Layer Security) es un protocolo criptográfico diseñado para proporcionar comunicaciones seguras a través de redes no confiables, como Internet. Su principal objetivo es garantizar la confidencialidad, integridad y autenticidad de los datos transmitidos entre dos puntos [8].

En el contexto de sistemas IoT, donde dispositivos distribuidos intercambian información sensible, el uso de TLS resulta fundamental. Este protocolo permite cifrar los mensajes, de modo que un atacante que intercepte la comunicación no pueda leer su contenido. Además, TLS impide modificaciones no autorizadas de los datos durante el tránsito y permite verificar la identidad de las partes implicadas mediante certificados digitales [14].

TLS opera sobre protocolos de aplicación como HTTP o MQTT, y requiere un proceso inicial de establecimiento de conexión conocido como *handshake*, durante el cual se negocian los algoritmos criptográficos a utilizar y se validan los certificados. Una vez establecida la conexión segura, los datos se transmiten mediante cifrado simétrico, lo que permite mantener un buen rendimiento incluso en dispositivos con recursos limitados.

Gracias a su amplio soporte, robustez y estándares abiertos, TLS se ha consolidado como la solución más común para asegurar las comunicaciones en redes modernas.

4.8.6. Interfaz gráfica: Node-RED

Para la visualización, gestión y procesamiento de los datos provenientes del broker MQTT se ha utilizado *Node-RED*, una herramienta de desarrollo basada en programación visual por flujos. Esta plataforma, desarrollada originalmente por IBM, está orientada a la creación de aplicaciones de integración entre dispositivos IoT, APIs y servicios online mediante una interfaz gráfica intuitiva [19].

Node-RED se ejecuta localmente en la Orange Pi, pero su interfaz gráfica es accesible desde cualquier navegador web conectado a la misma red local. A través de esta interfaz, el usuario puede construir flujos

de trabajo (workflows) de forma modular, arrastrando y conectando nodos que representan acciones, entradas, salidas y procesos.

En el contexto de este proyecto, Node-RED se emplea para:

- Recibir datos publicados por los sensores mediante suscripciones a topics MQTT.
- Obtener imágenes capturadas desde las cámaras integradas en los ESP32 mediante peticiones HTTP periódicas.
- Visualizar todos estos datos en un dashboard accesible desde otros dispositivos en la red.

Una vez definidos y desplegados los flujos, el sistema queda operativo sin necesidad de intervención manual, permitiendo el acceso al dashboard desde cualquier cliente web dentro de la red local.

4.8.7. VPN: ZeroTier

Una parte fundamental del sistema es la posibilidad de acceder a los datos de forma remota, especialmente al dashboard de Node-RED que se ejecuta en la Orange Pi Zero 2W. Para habilitar este acceso sin necesidad de abrir puertos en el router o depender de configuraciones complejas, se ha optado por utilizar *ZeroTier*, una solución de red definida por software que permite establecer redes virtuales privadas (VPN) de forma sencilla y segura [26].

ZeroTier destaca por su facilidad de uso y compatibilidad multiplataforma, lo que permite instalarlo en sistemas Linux, Windows, macOS, Android e iOS. En este proyecto se ha utilizado el plan gratuito, que permite hasta 10 dispositivos conectados simultáneamente, más que suficiente para las necesidades del sistema.

Los dispositivos conectados a la red ZeroTier en este proyecto son:

- **Orange Pi Zero 2W:** Aloja el broker MQTT y el dashboard de Node-RED. Debe estar accesible desde cualquier cliente remoto autorizado.
- **Ordenador portátil:** Desde él se configura, monitoriza y programa la Orange Pi y se visualiza el dashboard.
- **Teléfonos móviles:** Permiten acceder al sistema desde cualquier lugar, ofreciendo una experiencia portátil y en tiempo real.

Además del acceso remoto, ZeroTier aporta una capa adicional de seguridad. El administrador de la red debe aprobar explícitamente cada dispositivo que desee unirse, lo que evita accesos no autorizados. Adicionalmente, todas las comunicaciones dentro de la red virtual están cifradas de extremo a extremo, garantizando la confidencialidad e integridad de los datos transmitidos, incluso en redes no confiables.

Capítulo 5

Desarrollo e implementación

5.1. Piloto de funcionamiento individual de todos los elementos del proyecto

Antes de abordar la integración definitiva de todos los módulos, se diseñó un **piloto de funcionamiento individual** cuyo objetivo es verificar, de manera aislada, que cada uno de los elementos hardware y software del sistema cumple con los requisitos especificados. Este piloto incluye la instalación y configuración del bróker MQTT, la puesta en marcha de los nodos ESP32 con sus sensores (DHT22 y HC-SR501), la validación de la cámara OV5640 y la visualización de los datos en Node-RED. Realizar estas pruebas por separado es imprescindible por dos motivos:

1. Reduce la complejidad de la depuración, ya que permite identificar y corregir fallos sin la interferencia del resto de componentes

2. Proporciona evidencias objetivas del correcto funcionamiento antes de pasar a la fase de integración, minimizando así los riesgos y el tiempo de desarrollo.

5.1.1. Instalación de Mosquitto en Windows

Con el objetivo de probar el funcionamiento de los ESP32 junto a las cámaras y sensores y debido a un problema con el envío de la OrangePi. Se ha optado por instalar Mosquitto en un ordenador con Windows. Para ello, se ha descargado el instalador desde la página oficial de Mosquitto[17]. Se ha instalado correctamente y se ha comprobado que el servicio funciona correctamente, se han abierto 2 terminales, una como suscriptor del topic "pruebaz la otra ha publicado un mensaje para ese mismo topic. El mensaje ha llegado instantáneamente al suscriptor. Cabe recalcar que el puerto elegido ha sido el 1884, ya que el 1883 daba varios problemas con el firewall de Windows. Las imágenes de las pruebas se pueden ver a continuación:

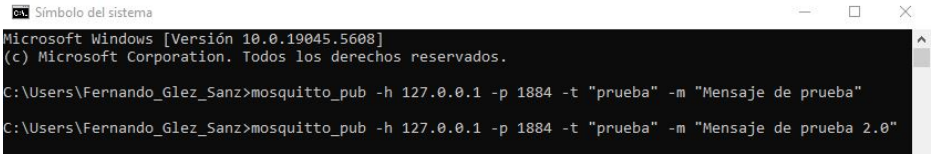

Figura 5.1: Mensajes enviados por el publicador de Mosquitto

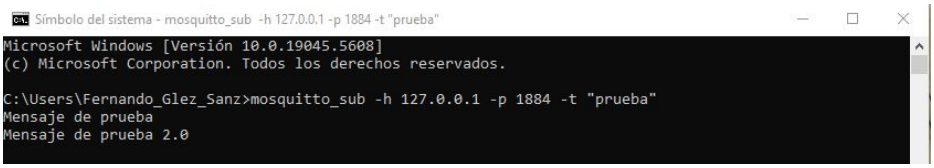

Figura 5.2: Mensajes recibidos por el suscriptor de Mosquitto

Con esto comprobado, se puede avanzar con las pruebas para el resto de dispositivos.

5.1.2. Pruebas con ESP32 y DHT22

PubSubClient de Nick O'Leary https://github.com/adafruit/DHT-sensor-library/actions Se va a empezar probando el sensor de humedad y temperatura DHT22 conectado al ESP32. Primero se ha instalado Arduino IDE y las librerías necesarias para el ESP32 y el DHT22. Después se ha cargado un código de prueba en el que se realiza la conexión con la red WiFi, con el bróker MQTT y se envían los datos periódicamente. Esto se ha hecho con un código muy básico [1]. Se ha comprobado que todo funciona correctamente, el sensor publica en el topic "pruebaz desde un suscriptor de una terminal del ordenador se pueden ver las mediciones.

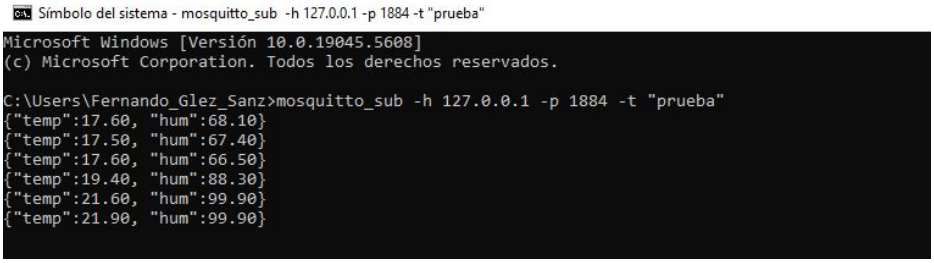

Figura 5.3: Mensajes recibidos por el suscriptor de Mosquitto

5.1.3. Pruebas con ESP32 y HC-SR501

A continuación, se va a probar el sensor de movimiento HC-SR501 de la misma forma que anteriormente. Se ha modificado levemente el código para que envíe un mensaje cuando detecte movimiento. 2 LIBRERIA DE HC-SR501: PIRSensor de Dean Gienger

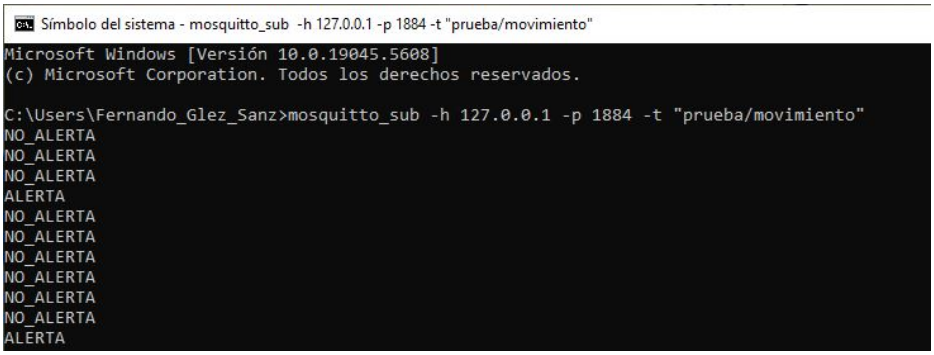

Figura 5.4: Mensajes recibidos por el suscriptor de Mosquitto

5.1.4. Instalación y configuración de Mosquitto en OrangePi

Instalación de Sistema Operativo: Armbian

Se ha elegido Armbian como sistema operativo para instalar en nuestra OrangePi. La imagen se ha descargado de la página oficial de Armbian[2]. Posteriormente, a través de BalenaEtcher 4.8.1 se ha flasheado la imagen en una tarjeta microSD. Hemos introducido la tarjeta microSd en la OrangePi y hemos conectado la tabla de expansión y el cable ethernet. Hemos buscado la ip de nuestra OrangePi en la página web de nuestro router a través de la dirección: 192.168.0.1. Introduciendo las claves de acceso correspondientes hemos localizado la ip de la OrangePi.

Figura 5.5: IP de la OrangePi a través de cable ethernet

A continuación, se ha accedido a la Orangepi a través de una terminal de Windows con el comando:

```
ssh root@192.168.0.21
```

Al ser la primera vez que se accede a la OrangePi, tenemos que configurrar la clave del host. Simplemente se responde ƶes". Después, ya nos solicita la contraseña para el usuario root, que por defecto es "1234". Seguidamente, se nos pide elegir una nueva contraseña, crear un nuevo usuario y configuraciones iniciales básicas como idioma y zona horaria.

Cuando finalizamos este proceso, ya estamos dentro de la OrangePi y podemos a empezar a instalar Mosquitto.

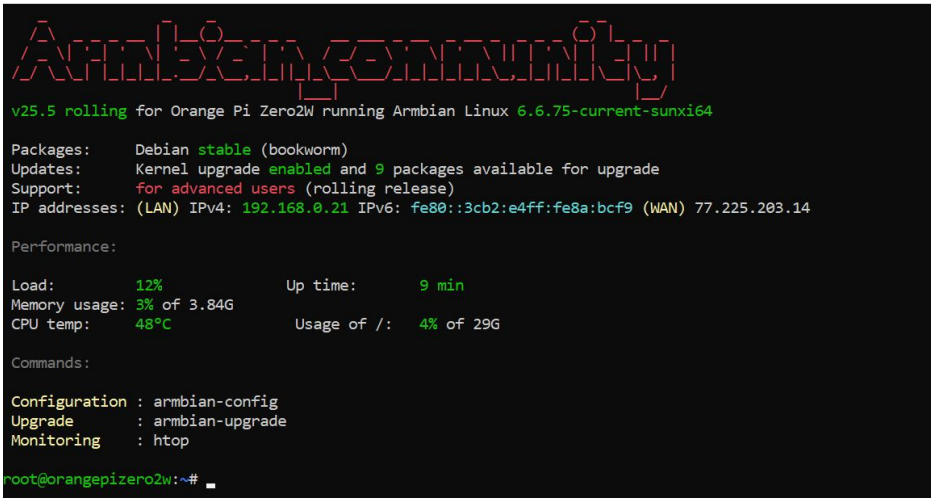

Figura 5.6: Terminal de la OrangePi

Instalación de Mosquitto y pruebas con cable ethernet

Para empezar, se ha actualizado el sistema operativo con los siguientes comandos:

```
sudo apt update && sudo apt upgrade
```

Después, se ha instalado Mosquitto con el siguiente comando:

```
sudo apt install mosquitto mosquitto-clients
```

Ahora vamos a abrir el puerto 1883 para que Mosquitto pueda recibir los mensajes. Para ello, se ha instalado la libreria iptables y se han utilizado los siguientes comandos:

```
sudo iptables -A INPUT -p tcp --dport 1883 -j ACCEPT
sudo iptables-save > /etc/iptables/rules.v4
sudo ss -tuln | grep 1883
```

Con el último comando comprobamos que el puerto 1883 está abierto y escuchando. Sin embargo, estaba escuchando en la dirección 127.0.0.1, que corresponde únicamente a localhost. Para solucionar esto, se ha creado un archivo de configuración custom.conf en la carpeta /etc/mosquitto/conf.d/ con la siguiente línea:

```
bind_address 0.0.0.0
```

Después, se ha creado un archivo de configuración aut.conf para no pedir autorización a los clientes que se conecten al bróker, en la misma carpeta con la siguiente línea:

```
allow_anonymous true
```

Por último, se ha reiniciado el servicio de Mosquitto con el siguiente comando:

```
sudo systemctl restart mosquitto
```

Ahora ya funciona correctamente, se han hecho pruebas desde dos terminales diferentes de Windows, una como publicador y otra como suscriptor y se ha comprobado que los mensajes se envían correctamente y se reciben en el suscriptor.

Figura 5.7: Mensajes enviados y recibidos a través del Mosquitto instalado en la OrangePi

Pruebas de funcionamiento con wifi

Ya hemos comprobado que Mosquitto funciona correctamente con conexión ethernet, ya que nuestra OrangePi no va a estar conectada por cable, vamos a configurar la conexión wifi. De este modo, reducimos el cableado ya que la orangePi solo necesitará el cable de alimentación.

Nos conectamos a la OrangePi a través de una terminal de Windows y su dirección IP por cable, igual que en el apartado anterior. Una vez nos hemos conectado, vamos a configurar la red wifi.

Primero, instalamos un cliente DHCP con el siguiente comando:

```
sudo apt install isc-dhcp-client -y
```

Este cliente DHCP es necesario para que la OrangePi pueda obtener una dirección IP de forma automática.

Posteriormente, editamos el archivo /etc/wpa_supplicant/wpa_supplicant.conf y añadimos las siguientes líneas:

```
network={
    ssid=*******
    psk=*******
    key_mgmt=WPA-PSK
}
```

En la siguiente imagen se puede ver como en el primer comando no había conexión wifi, después se ha modificado el archivo wpa_supplicant.conf como se ha indicado anteriormente, después se ha inciado la conexión con el comando "sudo wpa_supplicant -B -i wlan0 -c /etc/wpa_supplicant/wpa_supplicant.confz por último se ha obteniod una dirección IP con el comando "sudo dhclient wlan0". Finalmente, se ejecuta el primer comando de nuevo y se ve como la OrangePi ya tiene asignada una dirección IP.

```
root@orangepizero2w:~# ip a | grep wlan0
3: wlan0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
root@orangepizero2w:~# sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
root@orangepizero2w:~# sudo wpa_supplicant -B -i wlan0 -c /etc/wpa_supplicant/wpa_supplicant.conf
Successfully initialized wpa_supplicant
root@orangepizero2w:~# sudo dhclient wlan0
root@orangepizero2w:~# ip a | grep wlan0
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    inet 192.168.0.25/24 brd 192.168.0.255 scope global dynamic wlan0
root@orangepizero2w:~# sudo systemctl enable wpa_supplicant
```

Figura 5.8: Pruebas de conexión wifi en la OrangePi

La conexión se ha realizad correctamente y nuestra OrangePitiene asiganda la ip 192.168.0.25.

Ahora, vamos a programar la OrangePi para que se conecte automáticamente a la red wifi al reiniciar. Para ello, modificamos el archivo /etc/rc.local y añadimos las siguientes líneas:

```
sudo wpa\_supplicant -B -i wlan0 -c /etc/wpa\_supplicant/wpa\_supplicant.
   conf
sudo dhclient wlan0
exit 0
```

Y ya está, ahora la OrangePi se conecta automaticamente a la red wifi.

5.1.5. Pruebas en Node-RED

Instalación de Node-RED en OrangePi

Primero, se ha instalado Node.js, npm y Node-Red 4.8.6con los siguientes comandos:

```
    sudo apt install -y nodejs npm
    sudo npm install -g --unsafe-perm node-red
```

Seguidamente, se he ejecutado el comando "node-red"para iniciar el servidor de Node-Red. Hemos accedido a través de un navegador web a la dirección http://192.168.0.24:1880 y hemos configurado un flujo sencillo en el que se recibe un mensaje de un topic de Mosquitto y se muestra en un medidor en el dashboard de Node-Red.

El flujo que hemos implementado simplemente consiste en un receptor MQTT, suscrito al mismo topic en el que publica el ESP32 (solo publicamos la temperatura en esta prueba) y con salida un objeto JSON. Después se ha añadido un analizador JSON y finalmente un gauge (indicador) para mostrar la temperatura. Con el flujo ya diseñado, lo hemos instanciado y hemos visto el resultado en la direccion http://192.168.0.24:1880/ui , donde se puede ver correctamente el medidor de temperatura.

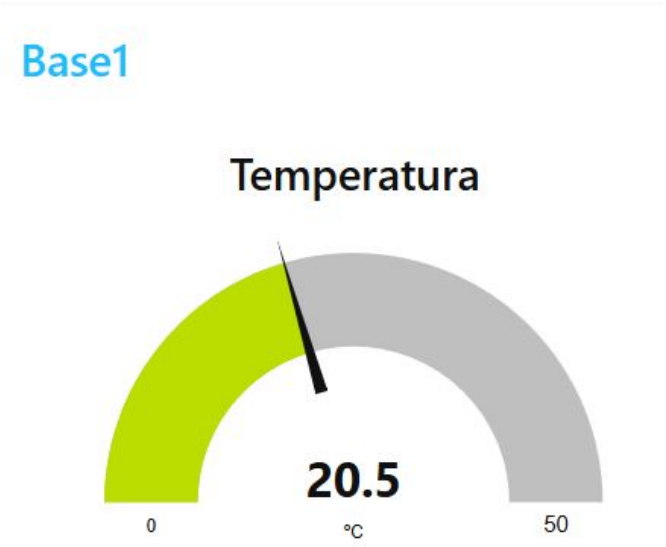

Figura 5.9: Visualización de la temperatura en el dashboard de Node-Red

5.1.6. Pruebas con ESP32 y cámara

Para comprobar si la camara funciona correctamente, se ha cargado un código de ejemplo de la librería ESP32-CAM. Este código se encarga de inicializar la cámara y enviar las imágenes a través de un servidor web. Para ello, se ha utilizado el siguiente código de ejemplo **??**:

Accedemeos a la camara buscando en el navegador la ip de la camara (obtenida a traves del monitor serie de arduino): 192.168.0.18.

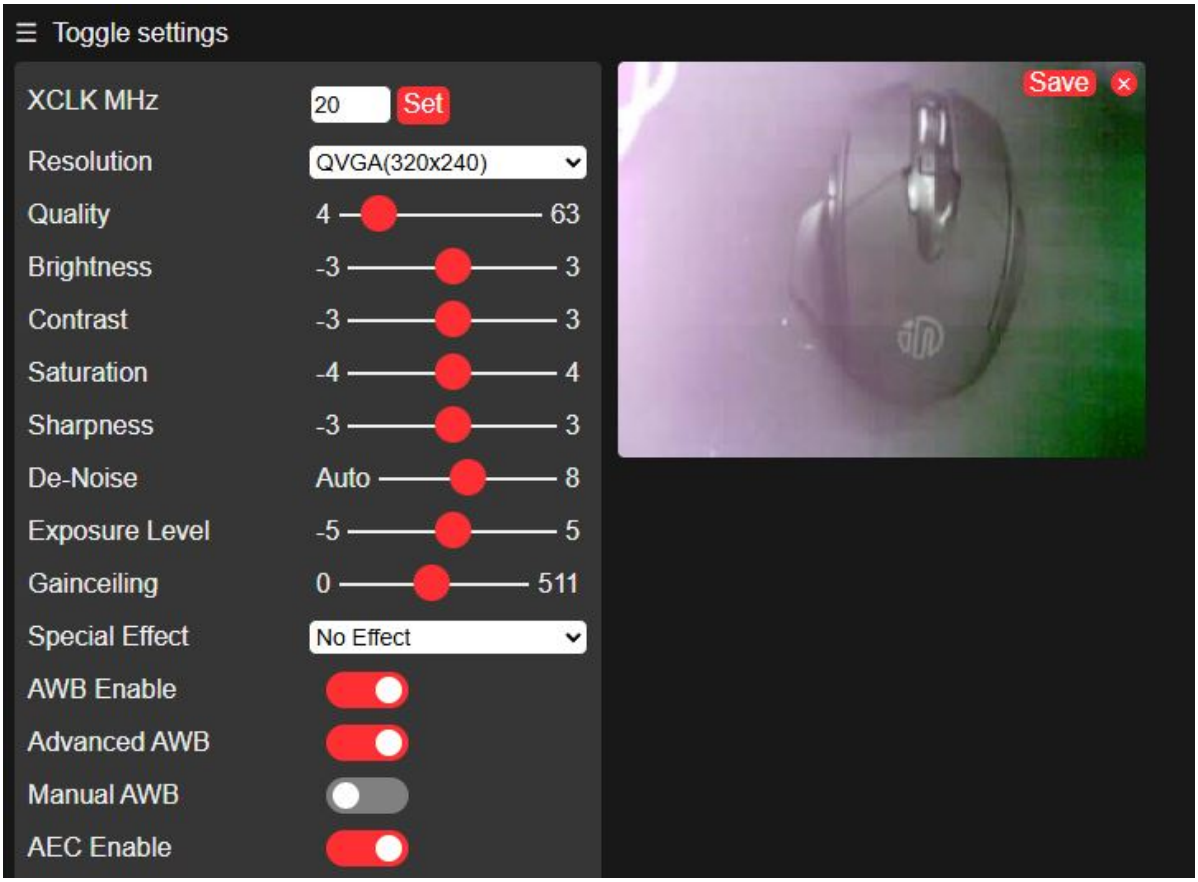

Figura 5.10: Stream Server de la cámara OV5640

5.2. Conexiones entre ESP32 y los distintos sensores en protoboard

En primera instancia, se realizó la conexión de los sensores al ESP32, para ello se utilizó un protoboard y cables de conexión. Para saber a qué pines de la placa conectar los sensores, se ha consultado el archivo camera_pins.h de la librería ESP32-CAM, donde se encuentran los pines que usa la cámara y los que quedan libres para ser asignados a las salidas de datos de los sensores. En cuanto a la alimentación de los sensores, se ha usado la alimentación de 3.3V para los 2 sensores. También la toma de tierra(GND) de la placa ESP32 se ha conectado con la de los 2 sensores. En la figura se puede observar el esquema de conexiones entre el ESP32 y los distintos sensores.

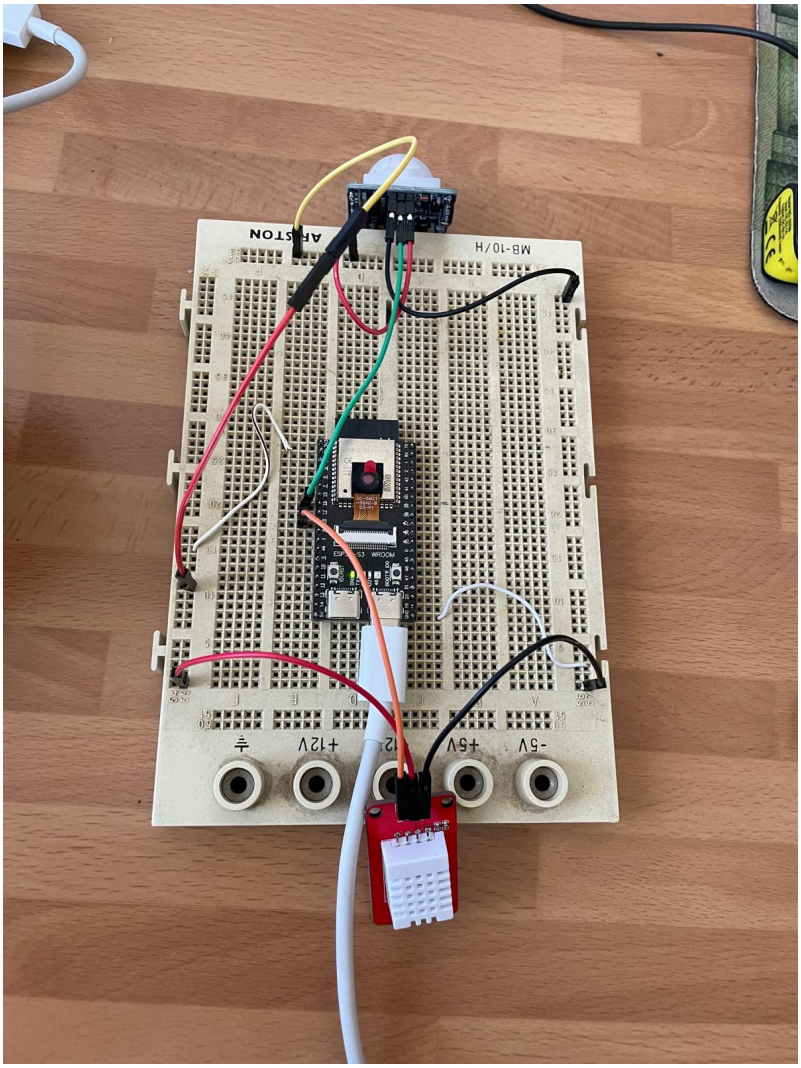

Figura 5.11: Conexiones entre el ESP32 y los sensores mediante un protoboard.

Con el montaje realizado, se procedió a programar el ESP32 para que hiciese todo lo que habíamos visto en la anterior sección, pero ahora todo junto. Por una parte, los datos de los sensores son publicados en el broker MQTT en su correspondiente topic y la cámara envía sus imágenes al stream server. El código que hemos cargado en el ESP32 es el siguiente: 4.

5.3. Pruebas de visualización en node-red

En esta sección se van a mostrar las pruebas de visualización de los datos obtenidos por el ESP32 en node-red. Con los 2 ESP32 programados para stremear las imágenes y publicar los datos de los sensores, se ha diseñado el siguiente flujo de node-red:

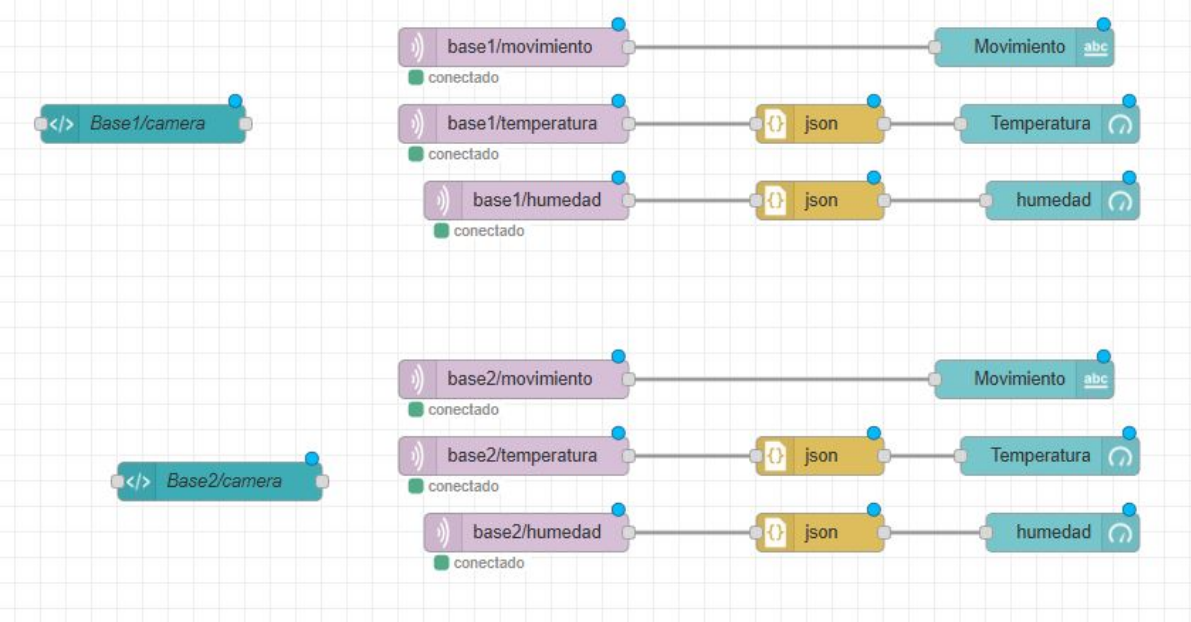

Figura 5.12: Flujo de node-red para la visualización de los datos obtenidos por el ESP32.

1. En la parte derecha de la figura vemos los nodos con los que implementamos la visualización de los sensores. En el MQTT-in del movimiento recibimos una cadena de texto que mostramos directamente. En los MQTT-in de tempertaura y humedad recibimos un JSON, por lo que lo pasamos a un analizador JSON y después lo mostramos en un nodo de gauge (medidor).

2. En la parte izquierda de la figura vemos los nodos con los que implementamos la visualización de las imágenes. Son 2 template en los que obtenemos la imagen del stream server y la ponemos dentro de un contenido con el siguiente código.

```
1        <div style="text-align: center;">
2    <img src="http://192.168.0.18:81/stream" style="width: 100%; height
        : 300px; object-fit: cover;">
3        </div>
```

Con este diseño de flujo, obtenemos el siguiente dashboard:

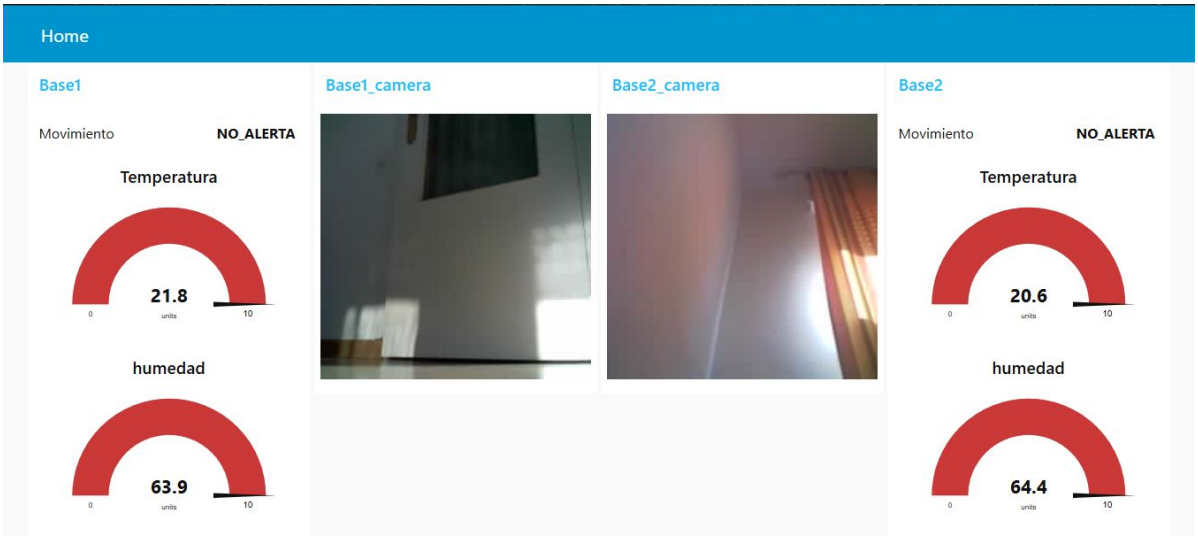

Figura 5.13: Dashboard de node-red para la visualización de los datos obtenidos por el ESP32.

5.4. Montaje de los ESP32 sin protoboard

Una vez que hemos comprobado que el sistema funciona correctamente vamos a proceder a realizar el cableado de los ESP32 con sus sensores. Esto nos permitirá tener un sistema más compacto y fácil de transportar e instalar. Para cada ESP32 se ha realizado el montaje de la siguiente manera:

1. GND: Se han modificado los cables para obtener un cable nuevo con 3 extremos hembra. Uno de ellos se conecta a la toma de tierra del ESP32 y los otros 2 a la toma de tierra de los sensores. Este cable es negro.

2. 3.3V: Se han modificado los cables para obtener un cable nuevo con 3 extremos hembra. Uno de ellos se conecta a la alimentación del ESP32 y los otros 2 a la alimentación de los sensores. Este cable es rojo.

3. DHT22 Data: Conectamos un cable hembra-hembra desde el pin 47 del ESP32 al pin de datos del sensor. Esta cable es naranja.

4. HC-SR 501 Data: Conectamos un cable hembra-hembra desde el pin 21 del ESP32 al pin de datos del sensor. Este cable es verde.

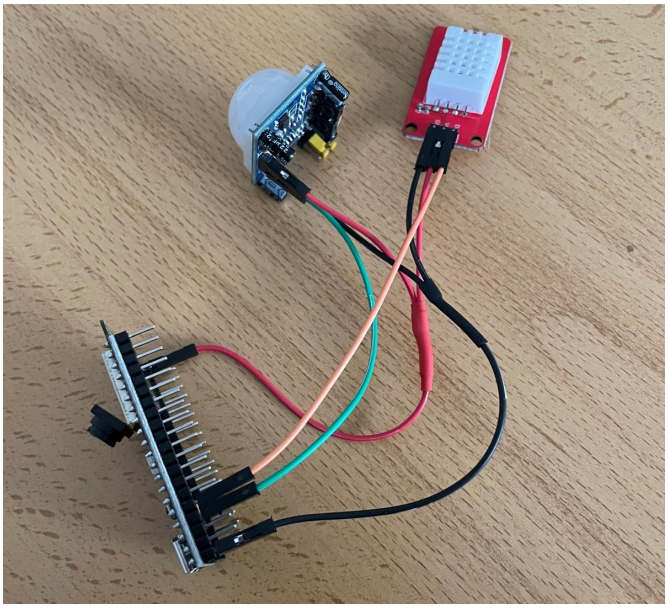

Figura 5.14: Montaje final del ESP32.

Para obtener el cable de 3 extremos hembras se ha pelado un cable hembra-hembra por el centro y ahí se le une otro que tenga un extremo hembra. Para fijar esta unión, se sueldan los cables y se utiliza tubo termoretractil para cubrir la unión.

Llegados a este punto, podemos confirmar que el sistema funciona correctamente. Los proximos pasos consistirán en permitir el acceso remoto al dashboard de node-red, ya que ahora mismo solo podemos acceder a él desde la red local. También debemos refinar el diseño del dashboard para que sea mas atractivo e intuitivo para el usuario.

5.5. Acceso remoto al dashboard de node-red

En este punto del proyecto, sabemos que el sistema funciona correctamente, sin embargo solo podemos acceder a la visualización del dashboard desde un dispositivo conectado a la misma red que la orangePi. Para lograr este objetivo vamos a usar la herramiento gratuita zeroTier. 4.8.7 El primer paso es crear una cuenta en su página web y crear una red. Una vez creada la red, se nos proporcionará un ID de red que es lo que usaremos para conectar los distintos dispositivos a nuestra red. Será necesario conectar la Orange Pi, el ordenador portátil y un móvil para hacer las pruebas correspondientes. Hay que tener en cuenta que zeroTier permite hasta un máximo de 10 dispositivos conectados a la red (en su plan gratuito).

Conexión del portátil a la red de zeroTier

Este paso es sencillo, solo hay que descargar e instalar el cliente de zeroTier en el portátil. Posteriormente, hay que introducir el ID de la red que nos proporcionó zeroTier. Para finalizar, al igual que en todos los dispositivos que se quieran conectar a la red, hay que autorizar la conexión desde la página web de zeroTier.

Conexión de la Orange Pi a la red de zeroTier

La configuración de zeroTier en la OrangePi la vamos a dividir en 2 partes. La primera consistirá en la instalación de zeroTier y la conexión a la red. La segunda parte se trata de cómo configurar la OrangePi para que inice automáticamente la conexión al encender la placa.

Para instalar zeroTier hay que ejecutar el siguiente comando en la terminal de Armbian:

```
curl -s https://install.zerotier.com | sudo bash
```

Una vez instalado, hay que ejecutar el siguiente comando para unirse a la red:

```
sudo zerotier-cli join <ID de la red>
```

<ID de la red>es el ID de la red que nos proporcionó zeroTier al crearla. Una vez ejecutado el comando, hay que autorizar la conexión desde la página de zeroTier de la misma forma que hicimos con el portátil.

En este punto, consultamos la dirección IP que se ha asignado a la OrangePi dentro de la red de zeroTier desde su página. En este punto se puede acceder al dashboard desde cualquier dispoitivo conectado a la red desde la dirección:

<IP de la OrangePi>:1880/ui

Ahora continuamos con la configuración de la OrangePi para que se conecta automáticamente a la red de zeroTier cada vez que se encienda de nuevo la orangePi. Para ello, vamos a crear un servicio systemd personalizado. Creamos el archivo con el siguiente comando:

```
sudo nano /etc/systemd/system/zerotier.service
```

En el que pondremos el siguiente contenido:

```
    [Unit]
Description=Zerotier Client
After=network.target

    [Service]
ExecStart=/usr/sbin/zerotier-one
Restart=always
User=root
Group=root
TimeoutStartSec=0

    [Install]
WantedBy=multi-user.target
```

En el apartado [Unit] se define el nombre del servicio y se le pide que se inicie después de que la red esté disponible (No tiene sentido que intente conectarse a la red de zeroTier si no ha establecido todavía la conexión Wifi). En la sección [Service] se define, en la primera linea, el comando que se ejecutará cuándo se inicie el servicio. En el resto de líneas se indica que si falla, debe volver a intentarlo automáticamente y que el servicio se ejecutará como usuario root. La última línea indica que no hay tiempo de espera para que el servicio se inicie. En la última sección [Install] se indica que el servicio se iniciará cúando alcanza el objetivo multi-user.

Con el servicio ya definido ejecutamos los siguientes comandos:

```
    sudo systemctl daemon-reload
    sudo systemctl restart zerotier.service
```

Estos comandos recargan los servicios del sistema y reinician el servicio de zeroTier que acabamos de definir.

Cómo último paso, modificamos el script de inicio del sistema con el comando:

```
    sudo nano /etc/rc.local
```

Y añadimos la siguiente líneas antes de la línea final exit 0;

```
    sudo systemctl start zerotier.service
    exit 0
```

Ahora vamos a definir también un servicio para que se inicie node-red automáticamente al encender la OrangePi. Para ello, creamos el archivo con el siguiente comando:

```
    sudo nano /etc/systemd/system/nodered.service
```

En el que pondremos el siguiente contenido:

```
    [Unit]
Description=Node-RED
After=network-online.target zerotier.service
Wants=network-online.target

    [Service]
ExecStart=/usr/bin/env node-red
Restart=on-failure
User=root
Environment=NODE_OPTIONS=--max_old_space_size=256

    [Install]
WantedBy=multi-user.target
```

Definimos el nombre del servicio e indicamos que debe iniciarse cuándo la red esté disponible y el servicio de zeroTier esté activo. El resto de líneas son iguales que para el anterior servicio, pero en este caso el comando que se ejecuta es el de node-red.

Instalación de un cliente de zeroTier en el móvil

Como último paso, vamos a instalar la aplicación de zeroTier en un dispositivo móvil. Para ello, hay que descargar la aplicación desde la tienda de aplicaciones correspondiente (Google Play Store o App Store). Una vez instalado el procedimiento es muy sencillo, simplemente hay que clicar en añadir una nueva red, introducir el ID de la red y autorizar la conexión desde la página web de zeroTier.

Para comprobar que funciona correctamente, nos desconectamos de la red Wifi y nos conectamos a la red de datos del móvil. Una vez conectados a la red privada de zeroTier, abrimos el navegador con la dirección IP de la OrangePi asignada en la red de zeroTier y el puerto 1880/ui.

El dashboard se ve correctamente desde el móvil, podemos consultar los datos de temperatura, humedad y movimiento, sin embargo, no se pueden ver las imágenes. Esto nos lleva a la siguiente sección, que no estaba contemplada en la hoja de ruta inicial del proyecto.

5.6. Resolución del problema de la visualización de imágenes fuera de la red local

Haciendo pruebas tanto con el portátil como con el móvil, se llegó a la conclusión de que el problema de la visualización del stream de las cáramas era que en los dos 'template' de node-red para las cámaras, estaba puesta la dirección IP de los ESP32 en la red local. Por este motivo, no se podían ver las imágenes desde fuera de la red local. Por lo que se identificaron 2 posibles soluciones:

1. Conectar los ESP32 a la red de zeroTier y usar la dirección IP que se les asigna en la red de zeroTier.

2. Hacer capturas de las imágenes desde la orangePi y publicarlas en el dashboard de node-red.

Sin embargo, haciendo más pruebas se vió otro problema que había que resolver. Y es que el stream server de la cámara no permite conexiones simultáneas, por lo que si hay un dispositivo viendo el stream, no se puede acceder desde otro dispositivo.

Por esta razón se optó por la segunda opción, ya que soluciona los 2 problemas detectados.

Se ha creado un nuevo flujo para la captura y publicación de las imágenes en el dashboard. Vamos a explicar el funcionamiento sólo para una cámara, ya que el funcionamiento es el mismo para las 2 cámaras.

El flujo es el siguiente:

Figura 5.15: Flujo de node-red para la captura y publicación de imágenes.

Estos nodos se definen de la siguiente manera:

1. http in: Tiene cómo método de acceso GET y el URL es /camara1.jpg. Esto significa que si accedemos a la dirección IP de la OrangePi y al puerto 1880/camara1.jpg, se ejecutará el flujo.

2. http request: Este nodo se encarga de hacer la petición HTTP a la cámara. En este caso, la URL es <IP de la cámara>/capture. Y tiene cómo tipo de dato a devolver un búfer binario.

3. http response: En este nodo no se define nada.

Este es el flujo que se encarga de capturar la imagen de la cámara y publicarla en el dashboard. Ahora, para mostrar esa imagen en el dashboard, se ha creado un nuevo template con el siguiente código:

```
<div style="text-align: center;">
    <img id="camara1" src="/camara1.jpg" style="width: 100%; height: 300px;
        object-fit: cover;">
</div>

<script>
    setInterval(() => {
    const cam = document.getElementById("camara1");
    cam.src = "/camara1.jpg?t=" + new Date().getTime();
    }, 500);
</script>
```

Este código se encarga de mostrar la imagen en el dashboard. La imagen se actualiza cada 500ms, por lo que parece un stream.

El punto de añadir la fecha y hora al final de la URL es para evitar que el navegador almacene en caché la imagen y no se vea la imagen actualizada. De esta froma cada nueva imagen tiene una URL diferente y el navegador no almacena la imagen en caché.

Llegados a este punto, podemos confirmar que el dashboard se ve correctamente, ya sea desde la red local o a través de la red de zeroTier.

El siguiente paso consistirá en mejorar la interfaz gráfica del dashboard.

5.7. Implementación de Comunicación Segura MQTT con TLS

En esta sección se describe detalladamente el procedimiento seguido para implementar un sistema de comunicación seguro basado en el protocolo MQTT usando TLS.

5.7.1. Configuración y generación de certificados TLS en Mosquitto

Para asegurar la comunicación MQTT se optó por usar TLS con certificados propios. Se siguieron los siguientes pasos:

- Se creó una nueva Autoridad Certificadora (CA) autofirmada usando OpenSSL, garantizando una clave RSA de 2048 bits y firma con SHA-256 para máxima compatibilidad con el ESP32.

```
openssl genrsa -out ca.key 2048
openssl req -x509 -new -nodes -key ca.key -sha256 -days 365 -out ca.crt
```

```
root@orangepizero2w:/etc/mosquitto/certs# openssl genrsa -out ca.key 2048
root@orangepizero2w:/etc/mosquitto/certs# openssl req -x509 -new -nodes -key ca.key -sha256 -days 365 -out ca.crt
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Valladolid
Locality Name (eg, city) []:Valladolid
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UVA
Organizational Unit Name (eg, section) []:Infor
Common Name (e.g. server FQDN or YOUR name) []:192.168.0.24
Email Address []:fernando.gonzalez.sanz@estudiantes.uva.es
```

Figura 5.16: Generación de la clave privada de la CA

- Se generó una nueva clave privada para el broker MQTT también con RSA 2048 bits:

```
openssl genrsa -out server.key 2048
```

- Se creó una solicitud de certificado (CSR) para el broker, estableciendo explícitamente el `Common Name` con la dirección IP local de la Orange Pi para evitar problemas de validación.

```
openssl req -new -key server.key -out server.csr
```

```
root@orangepizero2w:/etc/mosquitto/certs# openssl genrsa -out server.key 2048
root@orangepizero2w:/etc/mosquitto/certs# openssl req -new -key server.key -out server.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Valladolid
Locality Name (eg, city) []:Valladolid
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UVA
Organizational Unit Name (eg, section) []:Infor
Common Name (e.g. server FQDN or YOUR name) []:192.168.0.24
Email Address []:fernando.gonzalez.sanz@estudiantes.uva.es

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:passwordTFG25
An optional company name []:.
```

Figura 5.17: Generación de la CSR del broker

- El certificado del broker fue firmado con la CA recién creada:

```
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -
   CAcreateserial -out server.crt -days 365 -sha256
```

```
root@orangepizero2w:/etc/mosquitto/certs# openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CA
createserial -out server.crt -days 365 -sha256
Certificate request self-signature ok
subject=C = ES, ST = Valladolid, L = Valladolid, O = UVA, OU = Infor, CN = 192.168.0.24, emailAddress =
 fernando.gonzalez.sanz@estudiantes.uva.es
```

Figura 5.18: Firma del certificado del broker

- Los archivos `ca.crt`, `server.crt` y `server.key` se ubicaron en `/etc/mosquitto/certs` con permisos estrictos para garantizar la seguridad y accesibilidad solo al usuario `mosquitto`.

```
sudo chown mosquitto: /etc/mosquitto/certs/server.key
sudo chmod 600 /etc/mosquitto/certs/server.key
sudo chown mosquitto: /etc/mosquitto/certs/*.crt
sudo chmod 644 /etc/mosquitto/certs/*.crt
```

5.7.2. Configuración del broker Mosquitto

Se configuró Mosquitto para que escuche en el puerto estándar TLS 8883, habilitando la comunicación cifrada mediante TLS versión 1.2. Para ello, se añadió la siguiente configuración en `/etc/mosquitto/conf` donde se especifican las rutas de los certificados generados, se desactiva la exigencia de certificado para los clientes.

```
listener 8883
cafile /etc/mosquitto/certs/ca.crt
certfile /etc/mosquitto/certs/server.crt
keyfile /etc/mosquitto/certs/server.key
require_certificate false
tls_version tlsv1.2
bind_address 0.0.0.0
```

Tras configurar los certificados y la seguridad, se reinició el servicio Mosquitto ejecutando los siguientes comandos en la Orange Pi:

```
sudo systemctl restart mosquitto
sudo systemctl status mosquitto
sudo ss -tulpen | grep 8883
```

Con esto se comprobó que el servicio Mosquitto estaba activo y escuchando correctamente en el puerto 8883 con TLS habilitado.

5.7.3. Configuración y desarrollo del cliente ESP32

Para la comunicación segura, se actualizó el código del ESP32 con las siguientes modificaciones:

- Se utilizó la librería `WiFiClientSecure` para establecer una conexión TLS segura con el broker MQTT.
- El certificado público de la Autoridad Certificadora (CA) fue añadido en el código fuente en formato PEM como una cadena multilínea, lo que permite al ESP32 validar correctamente la identidad del broker durante la conexión TLS.
- Se configuró la conexión MQTT para utilizar el puerto seguro 8883 mediante un cliente seguro que integra `WiFiClientSecure` con la librería `PubSubClient`.

- Se ajustó el tiempo de espera para el handshake TLS, mejorando la estabilidad de la conexión en entornos con latencias variables.
- Se mantuvieron las funcionalidades originales del ESP32 para la lectura de sensores (PIR, DHT22) y captura con la cámara, publicando los datos a través de MQTT en el canal seguro establecido.

Las líneas clave para conseguir esto fueron:

```
WiFiClientSecure espSecureClient;

PubSubClient client(espSecureClient);

espSecureClient.setCACert(ca_cert);
espSecureClient.setHandshakeTimeout(30);
```

- `WiFiClientSecure espSecureClient;` Esta línea crea una instancia del cliente seguro que gestiona la conexión TLS entre el ESP32 y el broker. Se define como una variable global para que pueda ser utilizada en toda la aplicación.
- `PubSubClient client(espSecureClient);` Aquí se instancia el cliente MQTT `PubSubClien` asociándolo al cliente TLS `espSecureClient`. Esto permite que todas las comunicaciones MQTT se realicen sobre el canal seguro proporcionado por TLS.
- `espSecureClient.setCACert(ca_cert);` Esta línea configura el cliente TLS para que utilice el certificado de la Autoridad Certificadora (`ca_cert`) para validar el certificado presentado por el broker, asegurando que la conexión sea confiable y segura.
- `espSecureClient.setHandshakeTimeout(30);` Ajusta el tiempo máximo que el cliente espera para completar el proceso de handshake TLS. Esta línea se ubicó en la función `setup()`, justo después de establecer la conexión WiFi.

El resultado fue que el ESP32 logró conectarse de forma segura y estable al broker Mosquitto usando TLS con validación de certificados.

5.7.4. Configuración del consumidor Node-RED y alcance de la seguridad TLS

Para recibir los mensajes MQTT cifrados por TLS, se configuró Node-RED con los siguientes aspectos:

- Se creó una nueva configuración de servidor MQTT apuntando a la IP y puerto TLS del broker (`192.168.0.24:8883`).
- Se activó TLS en la configuración del broker MQTT dentro de Node-RED.

- Para evitar problemas de conexión relacionados con la validación del certificado, se **desactivó la opción "Verificar certificado del servidor"**. Por tanto, aunque la conexión está cifrada, Node-RED no valida que el broker sea legítimo.5.7.6
- Esta configuración permite que Node-RED establezca una conexión cifrada con el broker, asegurando la confidencialidad e integridad de los datos transmitidos entre ambos.

5.7.5. Problemas y dificultades encontradas durante la implementación de TLS

Durante la implementación del sistema MQTT seguro basado en TLS, se presentaron diversas dificultades técnicas y conceptuales que afectaron tanto a la configuración del broker Mosquitto como a los clientes ESP32 y Node-RED. A continuación, se describen las principales incidencias y la forma en que fueron abordadas.

Problemas iniciales con certificados y permisos en Mosquitto

- Al iniciar la configuración de TLS en Mosquitto, el servicio no pudo arrancar debido a errores relacionados con la carga de los certificados, principalmente por permisos insuficientes en los archivos de clave privada y certificados.
- Se solucionó estableciendo los permisos correctos en la carpeta de certificados y asignando la propiedad al usuario `mosquitto`, con permisos restrictivos para garantizar la seguridad.

Incompatibilidades y limitaciones en la generación de certificados

- El ESP32 mostró problemas para validar certificados que usaban claves mayores de 2048 bits o algoritmos y formatos de firma no compatibles.
- Se optó por generar explícitamente certificados con claves RSA de 2048 bits y firmas SHA-256, asegurando la compatibilidad con la librería `WiFiClientSecure` utilizada en el firmware del ESP32.

Problemas con la configuración TLS en Node-RED

- La validación del certificado del broker en Node-RED causaba errores de conexión persistentes, a pesar de usar el mismo certificado CA que el ESP32.
- Se diagnosticó que la versión de Node-RED y Node.js instaladas presentaban limitaciones en la gestión de certificados TLS personalizados, y que la conexión solo funcionaba desactivando la verificación del certificado del servidor.

5.7.6. Alcance y limitaciones de la seguridad implementada

El sistema ofrece un canal cifrado TLS entre el broker y los clientes, con las siguientes características y limitaciones:

- La comunicación entre los ESP32 y Mosquitto está cifrada y valida correctamente la identidad del broker mediante el certificado CA embebido, protegiendo contra ataques de tipo *Man-in-the-Middle*.
- Node-RED utiliza cifrado TLS para conectarse al broker, pero la verificación del certificado del servidor está desactivada, lo que implica riesgo potencial de suplantación si el entorno no es controlado.
- Esta configuración es adecuada para el entorno local y cerrado en el que se despliega el sistema (red Wi-Fi privada sin acceso externo), mitigando riesgos habituales en redes domésticas.
- Para entornos con mayores exigencias de seguridad, se recomienda activar la validación estricta en todos los clientes, implementar autenticación mutua con certificados de cliente y restringir el acceso al broker.

5.8. Dashboard de Node-RED

Uno de los componentes clave del sistema desarrollado es el dashboard implementado con Node-RED, que permite la monitorización en tiempo real de los sensores y las cámaras. Este panel de control ha sido diseñado con especial atención a la estética, la claridad de la información y, sobre todo, la adaptabilidad a dispositivos móviles.

5.8.1. Estructura del dashboard

El dashboard se ha organizado en dos columnas principales, una por cada base (**Base 1** y **Base 2**). Cada columna presenta los siguientes bloques verticales:

- **Título visual:** Se muestra un encabezado con el nombre de la base acompañado de un icono, destacando con un color distinto por base (azul para Base 1 y verde para Base 2) y fondo redondeado. Esto proporciona una rápida identificación y una presentación profesional.
- **Sensores:** Se muestran la temperatura y la humedad mediante bloques visuales con tipografía aumentada y colores simbólicos (termómetro rojo y gota azul). Se diseñaron con estilo oscuro, bordes suaves y espaciado para maximizar la legibilidad.
- **Últimos movimientos:** Este bloque presenta una tabla con las tres detecciones más recientes del sensor de movimiento, indicando fecha y hora.

- **Cámara:** Finalmente, se muestra la última imagen capturada por cada cámara conectada a las bases. Esta imagen se actualiza dinámicamente cada 500 ms mediante una etiqueta `<img>` con refresco automático en JavaScript, y se presenta dentro de un contenedor visual con bordes redondeados, sombra y título.

5.8.2. Diseño visual y adaptabilidad

El tema seleccionado fue el estilo **Dark**, con color base `#1e1e1e` y la fuente `Tahoma`, con el objetivo de ofrecer una apariencia moderna, coherente con el resto del sistema, y adecuada para visualización nocturna o prolongada. Los colores oscuros contrastan eficazmente con los textos claros y los iconos, facilitando la lectura.

Para garantizar una buena experiencia de usuario en dispositivos móviles, se utilizó un diseño responsivo vertical. En la vista móvil, cada base se presenta en una sola columna vertical, manteniendo el orden lógico de bloques: título, sensores, movimientos e imagen de cámara. Esta adaptación se realizó ajustando el ancho de los elementos a 7 columnas del sistema de grid de Node-RED Dashboard, lo cual asegura que ocupen el ancho completo sin provocar scroll horizontal.

5.8.3. Resultado obtenidos

El resultado es una interfaz clara, moderna y eficiente, accesible desde cualquier navegador web tanto en ordenador como en móvil a través de la VPN ZeroTier.

Aqui van las imágenes del dashboard en ordenador y móvil:

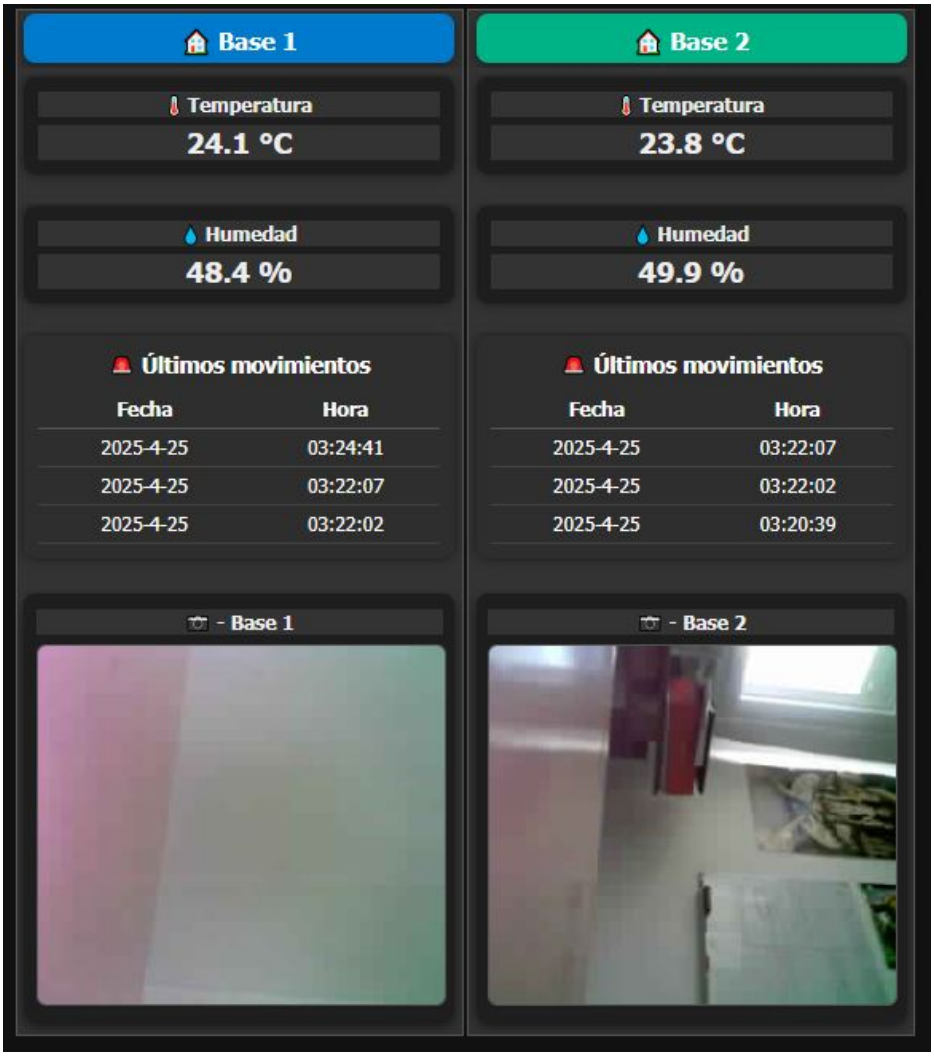

Figura 5.19: Dashboard de Node-RED en ordenador

Figura 5.20: Dashboard de Node-RED en móvil

Capítulo 6

Pruebas del sistema

6.1. Plan y alcance de las pruebas

El objetivo de esta fase es demostrar que la solución cumple los requisitos funcionales y de rendimiento antes de su despliegue definitivo. Siguiendo las buenas prácticas de Ingeniería del Software, se han diseñado y ejecutado los siguientes tipos de prueba:

- **Pruebas unitarias** (U): verifican cada sensor, el bróker MQTT y la cámara por separado, garantizando su correcto funcionamiento de forma aislada.
- **Pruebas del sistema** (S): evalúan el comportamiento completo en un escenario representativo de operación continua durante 8 h.
- **Pruebas de integración** (I): combinan los componentes hardware y software, verificando la interacción entre ellos y la correcta publicación de datos en el dashboard.

6.2. Entorno de pruebas

- **Hardware**: **OrangePi Zero 2W** (quad-core Cortex-A53 1.8 GHz, 1 GB RAM), 2 × ESP32-S3, sensor DHT22, sensor PIR HC-SR501, cámara OV5640.
- **Software**: Armbian 24.05 (`5.10.188`), Mosquitto `2.0.18`, Node-RED `3.1.2`, dashboard `node-red-3.5.1`.
- **Red**: Wi-Fi 2.4 GHz (IEEE 802.11n, 60 Mbps máx.), router TP-Link Archer VR600; subred 192.168.0.0/24.
- **Condiciones**: temperatura ambiente 20 ± 10 °C; dispositivos alimentados mediante adaptadores 5 V / 2 A.

6.3. Casos de prueba

Convenciones

ID Identificador único `CP-XX-YZ`, donde `XX` es el número secuencial y `Y/Z` indican el tipo (U/S/I).

Resultado `OK` = cumple criterios, `KO` = no cumple, `PEND` = pendiente de repetir.

ID	Objetivo	Datos de entrada	Resultado esperado	Res.
CP-01-U	Verificar acceso al dashboard desde distintos dispositivos	URL dashboard (`http://192.168.0.24:1880/ui`)	Carga $<$ 3 s; widgets actualizan datos cada 0.5 s	OK
CP-02-U	Comprobar publicación estable del DHT22	Alimentación al sensor, intervalo 10 s	100 % de mensajes sin pérdida; rango 10–50 °C / 20–90 %HR	OK
CP-03-U	Detectar movimiento con PIR HC-SR501	Mano a 1 m durante 5 s	Publica ALERTA o NO_ALERTA correctamente	**OK**
CP-04-I	Visualizar lecturas del DHT22 en el dashboard	Publicaciones cada 10 s; widget de texto T / HR	Texto actualiza en <2 s con 100 % de mensajes	OK
CP-05-I	Mostrar últimas alertas del HC–SR501	Mano a 1 m durante 5 s; lista de eventos	Debe listar las **3** últimas alertas con fecha/hora	**KO**[1]
CP-06-I	Verificar streaming de la cámara OV5640 en el dashboard	Visualizar flujo 30 s	Vídeo continuo con retardo <1 s	OK[2]
CP-07-S	Estabilidad 8 h	Funcionamiento continuo	Sin reinicios; pérdida $<$ 1 % mensajes	OK

Tabla 6.1: Matriz resumen de resultados

[1] El widget de Node-RED conserva a veces 4 entradas; pendiente de corrección.

[2] Se observan congelaciones puntuales de $\approx$ 1 s, aceptables en esta fase.

6.4. Conclusiones de las pruebas

Los resultados de la matriz permiten extraer las siguientes conclusiones generales:

- **Accesibilidad y rendimiento del dashboard** (CP-01-U) El dashboard carga en menos de 3 s desde distintos dispositivos y actualiza sus widgets cada 0.5 s sin interrupciones, garantizando una experiencia de usuario fluida.
- **Sensores ambientales DHT22**
  - Publicación en el bróker (CP-02-U): 100 % de mensajes recibidos dentro de los rangos esperados.
  - Visualización en Node-RED (CP-04-I): el widget de texto refresca las lecturas en menos de 2 s, cerrando el ciclo extremo $\rightarrow$ extremo sin pérdidas ni retardos apreciables.
- **Sensor PIR HC-SR501**
  - Funcionamiento del nodo ESP32 (CP-03-U): genera correctamente la secuencia `ALERTA/NO_ALERT`
  - Listado de alertas en el dashboard (CP-05-I): el widget debería mostrar las **tres** últimas alarmas, pero ocasionalmente mantiene cuatro entradas. La causa se localiza en la cola interna del nodo de UI; se clasifica como incidencia menor (`KO`) y quedará pendiente de ajuste.
- **Cámara OV5640** (CP-06-I) El flujo de vídeo se recibe con un retardo medio inferior a 1 s. Se detectan congelaciones puntuales de $\approx$ 1 s, consideradas aceptables para esta fase.
- **Estabilidad prolongada** (CP-07-S) La plataforma se mantuvo operativa durante 8 h continuas sin reinicios, cumpliendo los objetivos de robustez.

En conjunto, el sistema satisface los requisitos funcionales y de rendimiento. La única desviación relevante es la longitud variable de la cola de alertas PIR en el dashboard; se tratará como línea de mejora en futuras iteraciones. El resto de componentes se consideran validados y listos para despliegue.

Capítulo 7

Conclusiones y líneas futuras

7.1. Conclusiones

A lo largo del desarrollo de este Trabajo de Fin de Grado se ha conseguido materializar un **prototipo funcional** de sistema domótico modular y accesible, basado en el protocolo MQTT y orientado a la monitorización remota de dispositivos distribuidos en un entorno doméstico. El prototipo ha mostrado un comportamiento estable durante las pruebas realizadas, validando así los principios técnicos en los que se fundamenta.

El trabajo comenzó con un análisis detallado del protocolo MQTT, abordando sus características clave y sus ventajas frente a otras alternativas de comunicación en el ámbito del Internet de las Cosas. Este estudio permitió confirmar su idoneidad en escenarios con recursos limitados, debido a su ligereza, bajo consumo y simplicidad en la gestión de mensajes.

La elección de tecnologías de hardware y software se orientó hacia soluciones económicas y de fácil integración. La combinación de placas ESP32 con sensores DHT22 y HC-SR501, junto con una Orange Pi Zero 2W como nodo central, ha resultado ser adecuada para este tipo de sistemas. Sobre este último dispositivo se ha desplegado tanto el broker MQTT como la interfaz de visualización, desarrollada mediante Node-RED.

La arquitectura diseñada, basada en una estructura clara de tópicos MQTT y en la comunicación desacoplada entre nodos, ha facilitado la modularidad del sistema. Esta organización permite futuras ampliaciones o modificaciones sin necesidad de rediseñar el conjunto.

Durante el proceso de implementación se ha logrado interconectar todos los componentes, desde el firmware de los nodos hasta el entorno de visualización. Las pruebas han confirmado el correcto funcionamiento del flujo de datos, lo que demuestra que el prototipo cumple con los objetivos funcionales establecidos en la fase inicial del proyecto.

Desde una perspectiva formativa, el desarrollo de este trabajo ha contribuido al fortalecimiento de competencias técnicas en áreas como la programación de sistemas embebidos, redes, automatización, comunicaciones y aspectos relacionados con la ciberseguridad. Además, ha supuesto una experiencia integradora, al combinar distintas disciplinas propias de la ingeniería informática.

En definitiva, el prototipo desarrollado constituye una base sólida sobre la que construir versiones más completas de un sistema domótico real, manteniendo un enfoque adaptable, de bajo coste y con

potencial de aplicación en hogares, oficinas u otros entornos de pequeña escala.

7.2. Líneas futuras

El prototipo desarrollado constituye una base sólida sobre la que se pueden plantear nuevas investigaciones y proyectos dentro del ámbito de la domótica y el Internet de las Cosas. A partir de esta implementación funcional, se abren múltiples posibilidades de evolución que pueden ser abordadas en futuros Trabajos de Fin de Grado, Trabajos de Fin de Máster o proyectos de investigación aplicada. A continuación, se recogen algunas de las líneas más relevantes:

- **Desarrollo de un sistema completo de domótica:** Una de las principales líneas de continuación consiste en evolucionar el prototipo hacia un sistema completo, robusto y preparado para un uso real. Esto implicaría incorporar funcionalidades como autenticación de usuarios, control de acceso por roles, almacenamiento persistente de datos históricos, gestión de configuraciones remotas o registros de actividad (logs)

- **Captura y análisis inteligente de vídeo:** Se podría añadir una funcionalidad de grabación y análisis de vídeo activada por eventos, permitiendo aplicar técnicas de visión por computador para la detección automática de movimiento, reconocimiento de objetos o clasificación de situaciones relevantes.

- **Ampliación del conjunto de sensores:** El sistema podría enriquecerse con sensores adicionales (humo, gas, calidad del aire, ruido, etc.), lo que abriría la puerta a nuevos casos de uso y al análisis del comportamiento del sistema en contextos más exigentes o especializados, como hogares inteligentes avanzados o espacios industriales ligeros.

- **Sistemas de notificación y respuesta automatizada:** El desarrollo de sistemas de alerta mediante correo electrónico, notificaciones móviles o integración con servicios de mensajería permitiría una mayor reactividad ante eventos críticos. Esta línea puede complementarse con reglas configurables, automatizaciones personalizadas y análisis del comportamiento del usuario.

- **Control activo desde la interfaz:** Otra posible mejora sería permitir al usuario ejecutar acciones sobre dispositivos físicos desde el panel de control, como encender luces, abrir cerraduras electrónicas o activar sistemas de seguridad. Esto requeriría evaluar mecanismos de control seguro, prevenir accesos no autorizados y garantizar una interfaz usable y confiable.

- **Adaptación a nuevos contextos de uso:** El prototipo puede evolucionar hacia soluciones específicas para contextos como invernaderos, aulas, comercios, laboratorios o viviendas de asistencia. Estas adaptaciones pueden requerir ajustes en la interfaz, protocolos de comunicación o mecanismos de respaldo energético, entre otros.

Estas líneas de trabajo no solo suponen una evolución natural del prototipo actual, sino que también permiten abordar retos actuales en ámbitos como la comunicación distribuida, la interoperabilidad, la

ciberseguridad y la eficiencia energética, manteniendo un enfoque técnico con aplicaciones prácticas reales.

Anexo: Códigos para ESP32

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>
// Configuración del WiFi
const char* ssid = "*******";
const char* password = "*******";

// Configuración MQTT
const char* mqtt_server = "*******"; // IP del broker
const int mqtt_port = *******;                // Puerto MQTT (por defecto
   1883)
const char* mqtt_topic = "prueba";          // Topic donde publicará

// Configuración del DHT22
#define DHTPIN 17         // Pin GPIO donde está conectado el DHT22
#define DHTTYPE DHT22    // Tipo de sensor (DHT11 o DHT22)

DHT dht(DHTPIN, DHTTYPE);
WiFiClient espClient;
PubSubClient client(espClient);

// Función para conectar a WiFi
void setup_wifi() {
  delay(100);
  Serial.print("Conectando a WiFi...");
  WiFi.begin(ssid, password);
  while (WiFi.status() ¡= WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("¡Conectado a WiFi¡");
}

// Función para conectar a MQTT
```

```
34 void reconnect() {
35   while (¡client.connected()) {
36     Serial.print("Conectando a MQTT...");
37     if (client.connect("ESP32\\_DHT22")) {
38       Serial.println("¡Conectado¡");
39     } else {
40       Serial.print("Error, rc=");
41       Serial.print(client.state());
42       Serial.println(" Reintentando en 5s...");
43       delay(5000);
44     }
45   }
46 }
47
48 void setup() {
49   Serial.begin(115200);
50   setup_wifi();
51   client.setServer(mqtt_server, mqtt_port);
52   dht.begin();
53 }
54
55 void loop() {
56   if (¡client.connected()) {
57     reconnect();
58   }
59   client.loop();
60
61   // Leer temperatura y humedad
62   float temperatura = dht.readTemperature();
63   float humedad = dht.readHumidity();
64
65   if (isnan(temperatura) || isnan(humedad)) {
66     Serial.println("Error leyendo DHT22");
67     return;
68   }
69
70   // Crear mensaje JSON
71   char mensaje[50];
72   snprintf(mensaje, 50, "{\"temp\":%.2f, \"hum\":%.2f}", temperatura,
       humedad);
73
74   // Publicar en MQTT
75   client.publish(mqtt_topic, mensaje);
```

```
  Serial.print("Publicado: ");
  Serial.println(mensaje);

  delay(5000); // Publica cada 5 segundos
}
```

Código 1: Código de pruebas para DHT22

```
#include <WiFi.h>
#include <PubSubClient.h>

// Configuración de la red WiFi
const char* ssid = "*******";
const char* password = "*******";

// Configuración del Broker MQTT
const char* mqtt_server = "*******";  // Dirección IP de tu broker MQTT
const int mqtt_port = *******;                // Puerto del broker (1883 es
    el estándar)


// Configuración del HC-SR501
#define PIR_PIN 17  // Pin donde está conectado el OUT del HC-SR501

WiFiClient espClient;
PubSubClient client(espClient);

void setup_wifi() {
  Serial.print("Conectando a WiFi...");
  WiFi.begin(ssid, password);
  while (WiFi.status() ¡= WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("¡Conectado a WiFi¡");
}

void reconnect() {
  // Intentar reconectar si no estamos conectados
  while (¡client.connected()) {
    Serial.print("Conectando a MQTT...");
    if (client.connect("ESP32_Sensor")) {
      Serial.println("¡Conectado al broker MQTT¡");
    } else {
```

```
      Serial.print("Error, rc=");
      Serial.print(client.state());
      Serial.println(" Reintentando en 5 segundos...");
      delay(5000);
    }
  }
}

void setup() {
  // Inicializar Monitor Serial
  Serial.begin(115200);

  // Configuración de WiFi
  setup_wifi();

  // Configurar el cliente MQTT
  client.setServer(mqtt_server, mqtt_port);

  // Configurar el pin del PIR como entrada
  pinMode(PIR_PIN, INPUT);

  // Enviar mensaje inicial
  Serial.println("Esperando detección de movimiento...");
}

void loop() {
  // Reconectar si no estamos conectados al broker
  if (¡client.connected()) {
    reconnect();
  }
  client.loop();  // Mantener el cliente MQTT activo

  // Leer el estado del sensor PIR
  int estadoPIR = digitalRead(PIR_PIN);

  // Publicar un mensaje si detecta movimiento
  if (estadoPIR == HIGH) {
    Serial.println("Movimiento detectado¡");
    client.publish("prueba/movimiento", "ALERTA"); // Publica en el topic "
       prueba/movimiento"
  } else {
    Serial.println("No hay movimiento.");
    client.publish("prueba/movimiento", "NO_ALERTA"); // Publica en el
```

```
      topic "prueba/movimiento"
  }

  // Esperar antes de leer nuevamente
  delay(2000);
}
```

Código 2: Código de pruebas para HC-SR501

```
#include "esp_camera.h"
#include <WiFi.h>

//
// WARNING¡¡¡ PSRAM IC required for UXGA resolution and high JPEG quality
//            Ensure ESP32 Wrover Module or other board with PSRAM is selected
//            Partial images will be transmitted if image exceeds buffer size
//
//            You must select partition scheme from the board menu that has at least 3MB APP space.
//            Face Recognition is DISABLED for ESP32 and ESP32-S2, because it takes up from 15
//            seconds to process single frame. Face Detection is ENABLED if PSRAM is enabled as well

// ===================
// Select camera model
// ===================
//#define CAMERA_MODEL_WROVER_KIT // Has PSRAM
//#define CAMERA_MODEL_ESP_EYE  // Has PSRAM
#define CAMERA_MODEL_ESP32S3_EYE // Has PSRAM
//#define CAMERA_MODEL_M5STACK_PSRAM // Has PSRAM
//#define CAMERA_MODEL_M5STACK_V2_PSRAM // M5Camera version B Has PSRAM
//#define CAMERA_MODEL_M5STACK_WIDE // Has PSRAM
//#define CAMERA_MODEL_M5STACK_ESP32CAM // No PSRAM
//#define CAMERA_MODEL_M5STACK_UNITCAM // No PSRAM
//#define CAMERA_MODEL_M5STACK_CAMS3_UNIT  // Has PSRAM
//#define CAMERA_MODEL_AI_THINKER // Has PSRAM
//#define CAMERA_MODEL_TTGO_T_JOURNAL // No PSRAM
//#define CAMERA_MODEL_XIAO_ESP32S3 // Has PSRAM
// ** Espressif Internal Boards **
//#define CAMERA_MODEL_ESP32_CAM_BOARD
//#define CAMERA_MODEL_ESP32S2_CAM_BOARD
```

```
//#define CAMERA_MODEL_ESP32S3_CAM_LCD
//#define CAMERA_MODEL_DFRobot_FireBeetle2_ESP32S3 // Has PSRAM
//#define CAMERA_MODEL_DFRobot_Romeo_ESP32S3 // Has PSRAM
#include "camera_pins.h"

// ===========================
// Enter your WiFi credentials
// ===========================
const char *ssid = "*******";
const char *password = "*******";

void startCameraServer();
void setupLedFlash(int pin);

void setup() {
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();

  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.frame_size = FRAMESIZE_UXGA;
  config.pixel_format = PIXFORMAT_JPEG;  // for streaming
  //config.pixel_format = PIXFORMAT_RGB565; // for face detection/
     recognition
```

```
  config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
  config.fb_location = CAMERA_FB_IN_PSRAM;
  config.jpeg_quality = 12;
  config.fb_count = 1;

  // if PSRAM IC present, init with UXGA resolution and higher JPEG quality
  //                      for larger pre-allocated frame buffer.
  if (config.pixel_format == PIXFORMAT_JPEG) {
    if (psramFound()) {
      config.jpeg_quality = 10;
      config.fb_count = 2;
      config.grab_mode = CAMERA_GRAB_LATEST;
    } else {
      // Limit the frame size when PSRAM is not available
      config.frame_size = FRAMESIZE_SVGA;
      config.fb_location = CAMERA_FB_IN_DRAM;
    }
  } else {
    // Best option for face detection/recognition
    config.frame_size = FRAMESIZE_240X240;
#if CONFIG_IDF_TARGET_ESP32S3
    config.fb_count = 2;
#endif
  }

#if defined(CAMERA_MODEL_ESP_EYE)
  pinMode(13, INPUT_PULLUP);
  pinMode(14, INPUT_PULLUP);
#endif

  // camera init
  esp_err_t err = esp_camera_init(&config);
  if (err ¡= ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }

  sensor_t *s = esp_camera_sensor_get();
  // initial sensors are flipped vertically and colors are a bit saturated
  if (s->id.PID == OV3660_PID) {
    s->set_vflip(s, 1);        // flip it back
    s->set_brightness(s, 1);   // up the brightness just a bit
    s->set_saturation(s, -2);  // lower the saturation
```

```
116   }
117   // drop down frame size for higher initial frame rate
118   if (config.pixel_format == PIXFORMAT_JPEG) {
119     s->set_framesize(s, FRAMESIZE_QVGA);
120   }
121
122 #if defined(CAMERA_MODEL_M5STACK_WIDE) || defined(CAMERA_MODEL_M5STACK_
      ESP32CAM)
123   s->set_vflip(s, 1);
124   s->set_hmirror(s, 1);
125 #endif
126
127 #if defined(CAMERA_MODEL_ESP32S3_EYE)
128   s->set_vflip(s, 1);
129 #endif
130
131 // Setup LED FLash if LED pin is defined in camera_pins.h
132 #if defined(LED_GPIO_NUM)
133   setupLedFlash(LED_GPIO_NUM);
134 #endif
135
136   WiFi.begin(ssid, password);
137   WiFi.setSleep(false);
138
139   Serial.print("WiFi connecting");
140   while (WiFi.status() ¡= WL_CONNECTED) {
141     delay(500);
142     Serial.print(".");
143   }
144   Serial.println("");
145   Serial.println("WiFi connected");
146
147   startCameraServer();
148
149   Serial.print("Camera Ready¡ Use 'http://");
150 ␣␣Serial.print(WiFi.localIP());
151 ␣␣Serial.println("' to connect");
152 }
153
154 void loop() {
155   // Do nothing. Everything is done in another task by the web server
156   delay(10000);
```

```
}
```

Código 3: Codigo de pruebas para OV5640

```
#include "esp_camera.h"
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>

// Configuración del Broker MQTT
const char* mqtt_server = "*****";  // Dirección IP de tu broker MQTT
const int mqtt_port = 1883;                // Puerto del broker (1883 es el
    estándar)

// Configuración del HC-SR501
#define PIR_PIN 21  // Pin donde está conectado el OUT del HC-SR501

// Configuración del DHT22
#define DHTPIN 47         // Pin GPIO donde está conectado el DHT22
#define DHTTYPE DHT22    // Tipo de sensor (DHT11 o DHT22)

DHT dht(DHTPIN, DHTTYPE);

WiFiClient espClient;
PubSubClient client(espClient);


#define CAMERA_MODEL_ESP32S3_EYE
#include "camera_pins.h"

// Configuración de la red WiFi
const char* ssid = ********;          // Tu red WiFi
const char* password = ********; // Tu contraseña WiFi

void startCameraServer();
void setupLedFlash(int pin);


void reconnect() {
  // Intentar reconectar si no estamos conectados
  while (¡client.connected()) {
    Serial.print("Conectando a MQTT...");
    if (client.connect("ESP32_Sensor")) {
      Serial.println("¡Conectado al broker MQTT¡");
```

```
    } else {
      Serial.print("Error, rc=");
      Serial.print(client.state());
      Serial.println(" Reintentando en 5 segundos...");
      delay(5000);
    }
  }
}

void setup() {

  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();
  dht.begin();
  Serial.println("DHT_inic");

  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.frame_size = FRAMESIZE_UXGA;
  config.pixel_format = PIXFORMAT_JPEG;  // for streaming
  //config.pixel_format = PIXFORMAT_RGB565; // for face detection/
     recognition
  config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
  config.fb_location = CAMERA_FB_IN_PSRAM;
```

```
  config.jpeg_quality = 12;
  config.fb_count = 1;

  // if PSRAM IC present, init with UXGA resolution and higher JPEG quality
  //                      for larger pre-allocated frame buffer.
  if (config.pixel_format == PIXFORMAT_JPEG) {
    if (psramFound()) {
      config.jpeg_quality = 10;
      config.fb_count = 2;
      config.grab_mode = CAMERA_GRAB_LATEST;
    } else {
      // Limit the frame size when PSRAM is not available
      config.frame_size = FRAMESIZE_SVGA;
      config.fb_location = CAMERA_FB_IN_DRAM;
    }
  } else {
    // Best option for face detection/recognition
    config.frame_size = FRAMESIZE_240X240;
#if CONFIG_IDF_TARGET_ESP32S3
    config.fb_count = 2;
#endif
  }

#if defined(CAMERA_MODEL_ESP_EYE)
  pinMode(13, INPUT_PULLUP);
  pinMode(14, INPUT_PULLUP);
#endif

  // camera init
  esp_err_t err = esp_camera_init(&config);
  if (err ¡= ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }

  sensor_t *s = esp_camera_sensor_get();
  // initial sensors are flipped vertically and colors are a bit saturated
  if (s->id.PID == OV3660_PID) {
    s->set_vflip(s, 1);        // flip it back
    s->set_brightness(s, 1);   // up the brightness just a bit
    s->set_saturation(s, -2);  // lower the saturation
  }
  // drop down frame size for higher initial frame rate
```

```
125   if (config.pixel_format == PIXFORMAT_JPEG) {
126     s->set_framesize(s, FRAMESIZE_QVGA);
127   }
128 
129 #if defined(CAMERA_MODEL_M5STACK_WIDE) || defined(CAMERA_MODEL_M5STACK_
      ESP32CAM)
130   s->set_vflip(s, 1);
131   s->set_hmirror(s, 1);
132 #endif
133 
134 #if defined(CAMERA_MODEL_ESP32S3_EYE)
135   s->set_vflip(s, 1);
136 #endif
137 
138 // Setup LED FLash if LED pin is defined in camera_pins.h
139 #if defined(LED_GPIO_NUM)
140   setupLedFlash(LED_GPIO_NUM);
141 #endif
142 
143   WiFi.begin(ssid, password);
144   WiFi.setSleep(false);
145 
146   Serial.print("WiFi connecting");
147   while (WiFi.status() ¡= WL_CONNECTED) {
148     delay(500);
149     Serial.print(".");
150   }
151   Serial.println("");
152   Serial.println("WiFi connected");
153 
154 
155   // Configurar el cliente MQTT
156   client.setServer(mqtt_server, mqtt_port);
157 
158   // Configurar el pin del PIR como entrada
159   pinMode(PIR_PIN, INPUT);
160 
161   startCameraServer();
162 
163   Serial.print("Camera Ready¡ Use http://");
164   Serial.print(WiFi.localIP());
165   Serial.println(" to connect");
166 }
```

```
167
168 void loop() {
169
170   if (¡client.connected()) {
171     reconnect();
172   }
173   client.loop();  // Mantener el cliente MQTT activo
174
175   // Leer el estado del sensor PIR
176   int estadoPIR = digitalRead(PIR_PIN);
177
178   // Leer temperatura y humedad
179   float t = dht.readTemperature();
180   float h = dht.readHumidity();
181
182
183   if (isnan(t) || isnan(h)) {
184     Serial.println("Error leyendo DHT22");
185     return;
186   }
187
188   // Publicar un mensaje si detecta movimiento
189   if (estadoPIR == HIGH) {
190     Serial.println("Movimiento detectado¡");
191     client.publish("base2/movimiento", "ALERTA"); // Publica en el topic "
        prueba/movimiento"
192   } else {
193     Serial.println("No hay movimiento.");
194     client.publish("base2/movimiento", "NO_ALERTA"); // Publica en el topic
         "prueba/movimiento"
195   }
196
197   String tempStr = String(t, 2);  // Convertir a String con 2 decimales
198   String humStr = String(h, 2);
199
200   client.publish("base2/temperatura", tempStr.c_str());
201   client.publish("base2/humedad", humStr.c_str());
202
203   // Esperar antes de leer nuevamente
204   delay(500);
205 }
```

Código 4: Código de pruebas para el ESP32 con todos los sensores

Bibliografía

[1] Arduino. *Arduino - Home*. Accedido: 2025-03-24. 2025. URL: https://www.arduino.cc/.

[2] Armbian. *Orange Pi Zero 2W*. Accedido: 2025-03-22. 2025. URL: https://www.armbian.com/orange-pi-zero-2w/.

[3] Armbian Project. *Armbian Documentation*. Accedido: 2025-03-24. 2025. URL: https://docs.armbian.com/.

[4] Balena. *balenaEtcher - Flash OS images to SD cards and USB drives safely and easily*. Accedido: 2025-03-24. 2025. URL: https://etcher.balena.io/#about.

[5] Francisco Mahedero Biot. «Desarrollo de una aplicación IoT para el envío de imágenes mediante el protocolo MQTT». Accedido: 2025-05-16. Trabajo de Fin de Grado. Valencia, España: Universitat Politècnica de València, jul. de 2020. URL: https://riunet.upv.es/bitstream/handle/10251/152408/Mahedero%20-%20Desarrollo%20de%20una%20aplicaci%c3%b3n%20IoT%20para%20el%20env%c3%ado%20de%20im%c3%a1genes%20mediante%20el%20protocolo%20MQTT..pdf?sequence=1&isAllowed=y.

[6] Robert Bryce, Thomas Shaw y Gautam Srivastava. «Mqtt-g: A publish/subscribe protocol with geolocation». En: *2018 41st international conference on telecommunications and signal processing (TSP)*. Accedido: 2025-05-16. IEEE. 2018, págs. 1-4.

[7] Cedalo. *Understanding MQTT QoS*. Accedido: 2025-01-31. 2024. URL: https://cedalo.com/blog/understanding-mqtt-qos/#MQTT_QoS_0.

[8] Cloudflare. *What is Transport Layer Security (TLS)?* Accedido: 2025-05-19. 2024. URL: https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/.

[9] Debian Project. *Acerca de Debian*. Accedido: 2025-03-24. 2025. URL: https://www.debian.org/intro/about.es.html.

[10] Eclipse Mosquitto. *Eclipse Mosquitto*. Accedido: 2025-03-24. 2025. URL: https://mosquitto.org/.

[11] EMQX. *Understanding the CoAP Protocol for IoT Applications*. Accedido: 2025-05-26. 2023. URL: https://www.emqx.com/en/blog/coap-protocol.

[12] XMPP Standards Foundation. *Internet of Things with XMPP*. Accedido: 2025-05-26. 2022. URL: https://xmpp.org/uses/internet-of-things/.

[13] Glassdoor. *Sueldo de Desarrollador IoT en Madrid*. Accedido: 2025-06-06. 2025. URL: https://www.glassdoor.es/Sueldos/madrid-desarrollador-iot-sueldo-SRCH_IL.0,6_IM1030_KO7,24.htm.

[14] HiveMQ. *MQTT - The Standard for IoT Messaging*. Accedido: 2025-01-31. 2024. URL: https://www.hivemq.com/mqtt/.

[15] HiveMQ. *Scaling MQTT to Millions of Connections*. Accedido: 2025-05-01. 2021. URL: https://www.hivemq.com/blog/scaling-mqtt-to-millions-of-connected-devices/.

[16] Raynel Moreno Hernández. «Desarrollo de una aplicación IoT para la gestión de un hogar inteligente mediante el protocolo MQTT y Sistemas en chip (SoC) ESP32». Accedido: 2025-05-16. Tesis doct. Universitat Politècnica de València, 2020.

[17] Eclipse Mosquitto. *Mosquitto MQTT Broker - Download*. Accedido: 2025-03-05. 2025. URL: https://mosquitto.org/download/.

[18] Nabto. *REST API for IoT: The Ultimate Guide*. Accedido: 2025-05-26. 2023. URL: https://www.nabto.com/rest-api-iot-guide/.

[19] OpenJS Foundation. *Node-RED: Low-code programming for event-driven applications*. Accedido: 2025-04-21. 2025. URL: https://nodered.org/.

[20] TarifasGasLuz.com. *¿Cuál es el precio de la luz hoy y las horas más baratas?* Accedido: 2025-04-21. 2025. URL: https://tarifasgasluz.com/comparador/precio-kwh.

[21] Software Toolbox. *MQTT Quality of Service and the DataHub*. Accedido: 2025-01-31. 2024. URL: https://blog.softwaretoolbox.com/mqtt-quality-of-service-datahub.

[22] Juan Sebastian Trujillo Loaiza et al. «Implementación de una red inalámbrica de cámaras integrada a un sistema mecatrónico para el monitoreo de entornos residenciales y el manejo remoto de puertas usando MQTT». En: (2022). Accedido: 2025-05-16.

[23] Wikipedia contributors. *Armbian - Wikipedia, la enciclopedia libre*. Accedido: 2025-03-24. 2025. URL: https://es.wikipedia.org/wiki/Armbian.

[24] Xataka. *Cómo grabar una ISO en un USB*. Accedido: 2025-03-24. 2025. URL: https://www.xataka.com/basics/como-grabar-iso-usb.

[25] Xataka. *Qué es Arduino, cómo funciona y qué puedes hacer con uno*. Accedido: 2025-03-24. 2025. URL: https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno.

[26] ZeroTier, Inc. *ZeroTier*. Accedido: 2025-04-27. 2025. URL: https://www.zerotier.com/.