UNIVERSIDAD DE CASTILLA-LA MANCHA

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

Protocolo de mensajería ligero inspirado en MQTT para redes ESP-NOW

MQTT-inspired lightweight messaging protocol for ESP-NOW networks

Rubén Gómez Villegas

Julio, 2024

UNIVERSIDAD DE CASTILLA-LA MANCHA

TRABAJO FIN DE GRADO

Departamento de Tecnologías y Sistemas de Información

Tecnología Específica de Sistemas de Información

Protocolo de mensajería ligero inspirado en MQTT para redes ESP-NOW

Autor: Rubén Gómez Villegas

Tutor Académico: Rubén Cantarero Navarro

Cotutor Académico: Ana Rubio Ruiz

Julio , 2024

*D*edicado a mi familia y a todos
aquellos ...

Declaración de Autoría

Yo, Rubén Gómez Villegas con DNI 02318379W declaro que soy el único autor del trabajo fin de grado titulado “Protocolo de mensajería ligero inspirado en MQTT para redes ESP-NOW” y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual y que todo el material no original contenido en dicho trabajo está apropiadamente atribuido a sus legítimos autores.

Talavera de la Reina, a…..

Fdo: ……

Resumen

Esta plantilla puede modificarse para adaptarse a las particularidades de cada Proyecto, tanto en contenido como en formato, siempre y cuando se respete las directrices básicas indicadas en la guía de estilo y formato para la elaboración de TFG del Grado en Ingeniería Informática de la Facultad de Ciencias Sociales y Tecnologías de la Información de Talavera de la Reina.

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Agradecimientos

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Índice general

Índice de figuras

Índice de Tablas

Índice de Listados

Acrónimos

ACK Acknowledgement, Acuse de recibo

AES Advanced Encryption Standard, Estándar de Cifrado Avanzado

AIoT Artificial Intelligence of Things, Inteligencia Artificial de las Cosas

AMQP Advanced Message Queuing Protocol, Protocolo Avanzado de Colas de Mensajes

ARPANET Advanced Research Projects Agency Network, Red de la Agencia de Proyectos de Investigación Avanzada

API Application Programming Interface, Interfaz de Programación de Aplicaciones

BLE Bluetooth Low Energy, Bluetooth de Baja Energía

CoAP Constrained Application Protocol, Protocolo de Aplicación Restringida

CPU Central Processing Unit, Unidad Central de Procesamiento

CSS Cascading Style Sheets, Hojas de Estilo en Cascada

DDS Data Distribution Service, Servicio de Distribución de Datos

DTLS Data Transport Layer Security, Protocolo de Seguridad de la Capa de Transporte

ECC Elliptic Curve Cryptography, Criptografía de Curva Elíptica

ECDH Elliptic-curve Diffie–Hellman

ETS Erlang Term Storage, Almacenamiento de términos de Erlang

GB Gigabyte

GHz Gigahercios

GPIO General Purpose Input/Output, Entrada/Salida de Propósito General

HTML HyperText Markup Language, Lenguaje de Marcado de Hipertexto

HTTP HyperText Transfer Protocol, Protocolo de Transferencia de Hipertexto

ID Identificador, Identifier

IDE Integrated Development Environment, Entorno de Desarrollo Integrado

IEEE Institute of Electrical and Electronics Engineers, Instituto de Ingenieros Eléctricos y Electrónicos

IETF Internet Engineering Task Force, Grupo de Trabajo de Ingeniería de Internet

IoT Internet of Things, Internet de las Cosas

IP Internet Protocol, Protocolo de Internet

ITU International Telecommunication Union, Unión Internacional de Telecomunicaciones

JPEG Joint Photographic Experts Group

JSON JavaScript Object Notation, Notación de Objeto de JavaScript

Kbps Kilobits por segundo

KiB Kibibyte

LAN Local Area Network, Red de Área Local

mA Miliamperios

MAC Media Access Control, Control de Acceso a Medios

MAN Metropolitan Area Network, Red de Área Metropolitana

Mbps Megabits por segundo

MHz Megahercios

MIB Management Information Base, Base de Información Gestionada

MiB Mebibyte

MQTT Message Queue Telemetry Transport, Transporte de Telemetría en Colas de Mensajes

MTU Maximum Transmission Unit, Unidad de Transmisión Máxima

mW Milivatios

NACK Negative Acknowledgement, Acuse de recibo Negativo

OASIS Organization for the Advancement of Structured Information Standards, Organización para el Avance de Estándares de Información Estructurada

OSI Open Systems Interconnection, Interconexión de Sistemas Abiertos

PDF Portable Document Format, Formato de Documento Portátil

PNG Portable Network Graphics, Gráficos de Red Portátiles

QoS Quality of Service, Calidad de Servicio

RAM Random Access Memory, Memoria de Acceso Aleatorio

REST Representational State Transfer, Transferencia de Estado Representacional

RFID Radio Frequency Identification, Identificación por Radiofrecuencia

RISC Reduced Instruction Set Computer, Computador con Conjunto de Instrucciones Reducido

RSA Rivest–Shamir–Adleman

SHA Secure Hash Algorithm, Algoritmo de Hash Seguro

SoC System on a Chip, Sistema en un Chip

SNMP Simple Network Management Protocol, Protocolo Simple de Gestión de Redes

SSL Secure Sockets Layer, Capa de Puertos Seguros

SVG Scalable Vector Graphics, Gráficos Vectoriales Escalables

TB Terabyte

TCP Transmission Control Protocol, Protocolo de Control de Transmisión

TLS Transport Layer Security, Seguridad de la Capa de Transporte

TFG Trabajo Final de Grado

UCLM Universidad de Castilla-La Mancha

UDP User Datagram Protocol, Protocolo de Datagramas de Usuario

V Voltios

VoIP Voice over Internet Protocol, Voz sobre Protocolo de Internet

WEP Wired Equivalent Privacy, Privacidad Equivalente a Cableado

WPA Wi-Fi Protected Access, Acceso Wi-Fi Protegido

XML Extensible Markup Language, Lenguaje de Marcado Extensible

XMPP Extensible Messaging and Presence Protocol, Protocolo Extensible de Mensajería y Comunicación de Presencia

µs Microsegundos

ºC Grados centígrados

Capítulo 1

Introducción

El capítulo de Introducción debe describir el problema que se pretende resolver con el desarrollo del Trabajo Fin de Grado (TFG). Debe dar respuesta al qué sin especificar cómo se va a realizar, para lo cual se usarán el resto de los capítulos del documento. El lector de este documento debe tener claro el alcance del proyecto habiendo leído únicamente el capítulo de Introducción.

1.1. Motivación

Esta sección aborda la motivación del trabajo. Se trata de señalar la necesidad que lo origina, su actualidad y pertinencia. Puede incluir también un estado de la cuestión (o estado del arte) en la que se revisen estudios o desarrollos previos y en qué medida sirven de base al trabajo que se presenta.

En este capítulo debería introducirse el contexto disciplinar y tecnológico en el que se desarrolla el trabajo de modo que pueda entenderse con facilidad el ámbito y alcance del TFG. Puesto que un TFG no tiene que ser necesariamente un trabajo con aportes novedosos u originales, solo es necesario la inclusión de estado del arte cuando este contribuya a aclarar aspectos clave del TFG o se desee justificar la originalidad del trabajo realizado. Si la sección estado del arte es muy extensa, considera la opción de introducirla como un capítulo independiente.

1.2. Redacción de la Memoria

Durante la realización de la memoria del TFG es importante tener presente respetar la guía de estilo de la institución. Por tanto, el empleo de plantillas para un sistema de procesamiento de textos (por ejemplo, Word o LaTeX) puede requerir su adaptación cuando la plantilla mencionada no haya sido suministrada en la institución a la que se dirige el trabajo.

Para redactar un trabajo académico de modo efectivo se deben tener presentes una serie de normas que

ayuden a conseguir un resultado final que sea claro y de fácil lectura.

A la hora de redactar el texto se debe poner especial atención en no cometer plagio y respetar los derechos de propiedad intelectual. En particular merece gran atención la inclusión de gráficos e imágenes procedentes de Internet que no sean de elaboración propia. En este sentido se recomienda consultar el manual de la Universidad de Cantabria [1] en el que se explica de modo conciso cómo incluir imágenes en un trabajo académico.

1.3. Estructura del Documento

Este capítulo suele incluir una sección que indica la estructura (capítulos y anexos) del documento y el contenido de cada una de las partes en que se divide. Por tanto, las secciones que suelen acompañar este capítulo son:

- Motivación. Responde a la pregunta sobre la necesidad o pertinencia del trabajo.
- Objetivo. Determina de modo claro el propósito del trabajo descrito que puede desglosarse en subobjetivos cuando el objetivo principal se puede descomponer en módulos o componentes. Es muy importante definir el objetivo de modo apropiado. El Capítulo Objetivos de esta guía explica cómo definir el objetivo.
- Antecedentes o Contexto disciplinar/tecnológico. También puede denominarse Estado del Arte cuando se trata de comentar trabajos relacionados que han abordado la cuestión u objetivo que se plantea.
- Estructura del documento. Resumen de los capítulos y anexos que integran el documento.

1.4. Abreviaturas y Acrónimos

Un TFG que utiliza muchas abreviaturas y acrónimos puede añadir esta sección dónde se muestra el conjunto de abreviaturas y acrónimos y su significado.

1.5. Objetivos

Para hacer un planteamiento apropiado de los objetivos se recomienda utilizar la Guía para la elaboración de propuestas de TFG en la que se explica cómo definir correctamente los objetivos de un TFG.

1.5.1. Objetivo General

Introduce y motiva la problemática (i.e ¿cuál es el problema que se plantea y por qué es interesante su resolución?).

[1] Guía de Imágenes: https://web.unican.es/buc/Documents/Formacion/guia_imagenes.pdf

Debe concretar y exponer detalladamente el problema a resolver, el entorno de trabajo, la situación y qué se pretende obtener. También puede contemplar las limitaciones y condicionantes a considerar para la resolución del problema (lenguaje de construcción, equipo físico, equipo lógico de base o de apoyo, etc.). Si se considera necesario, esta sección puede titularse Objetivos del TFG e hipótesis de trabajo. En este caso, se añadirán las hipótesis de trabajo que el/la estudiante pretende demostrar con su TFG.

Una de las tareas más complicadas al proponer un TFG es plantear su Objetivo. La dificultad deriva de la falta de consenso respecto de lo que se entiende por objetivo en un trabajo de esta naturaleza.

En primer lugar, se debe distinguir entre dos tipos de objetivo:

- La finalidad específica del TFG que se plantea para resolver una problemática concreta aplicando los métodos y herramientas adquiridos durante la formación académica. Por ejemplo, Desarrollo de una aplicación software para gestionar reservas hoteleras on-line.
- El propósito académico que la realización de un TFG tiene en la formación de un graduado. Por ejemplo, la adquisición de competencias específicas de la intensificación cursada.

En el ámbito de la memoria del TFG se tiene que definir el primer tipo de objetivo, mientras que el segundo tipo es el que se añade en el Capítulo de Conclusiones y que justifica las competencias específicas de la intensificación alcanzadas y/o reforzadas con la realización del trabajo.

La categoría del objetivo planteado justifica modificaciones en la organización genérica de la memoria del trabajo. Así en el caso de estudios y validación de hipótesis el apartado de resultados y conclusiones debería incluir los resultados de experimentación y los comentarios de cómo dichos resultados validan o refutan la hipótesis planteada.

1.5.2. Objetivos Específicos

Generalmente, el objetivo general puede ser descompuesto en varios objetivos más específicos que se pretenden alcanzar. En esta sección se enumeran y describen cada uno de ellos.

Junto con la definición de estos objetivos se puede especificar los requisitos que debe satisfacer la solución aportada. Estos requisitos especifican características que debe poseer la solución y restricciones que acotan su alcance. En el caso de un trabajo cuyo objetivo es el desarrollo de un artefacto los requisitos pueden ser funcionales y no funcionales.

Al redactar el objetivo de un TFG se debe evitar confundir los medios con el fin. Así es habitual encontrarse con objetivos definidos en términos de las acciones (verbos) o tareas que será preciso realizar para llegar al verdadero objetivo. Sin embargo, a la hora de planificar el desarrollo del trabajo si es apropiado descomponer todo el trabajo en hitos y estos en tareas para facilitar dicha planificación.

La categoría del objetivo planteado justifica modificaciones en la organización genérica de la memoria

del trabajo. Así en el caso de estudios y validación de hipótesis el apartado de resultados y conclusiones debería incluir los resultados de experimentación y los comentarios de cómo dichos resultados validan o refutan la hipótesis planteada.

Capítulo 2

Estado del arte

2.1. Internet de las Cosas

Dado que una de las principales motivaciones de este Trabajo de Fin de Grado es el aporte que puede ofrecer a la facilidad de implementación del Internet de las Cosas, Internet of Things (IoT) en inglés, es importante conocer el joven concepto que nos rodea en la actualidad.

El siglo XX dio lugar al desarrollo de numerosos inventos que impulsaron una revolución y un avance ágil en la sociedad, que en la actualidad son frecuentemente usados y facilitan la vida humana. Este es el caso del Internet, que desde 1969 ha permitido la comunicación entre personas y el acceso a información. Hoy en día, también millones de objetos están conectados a la red, cuyas funcionalidades dependen de esta.

IoT es un término utilizado por primera vez en un discurso realizado en septiembre de 1985, cuando Peter T. Lewis predijo que "no sólo los humanos, sino también las máquinas y otras cosas se comunicarán interactivamente a través de Internet" /**TODO: mencionar al podcast**/; también empleado por Kevin Ashton en 1999 al realizar una presentación del uso de etiquetas de identificación por radiofrecuencia, o radio frequency identification (RFID) en inglés, y otros sensores en los productos de la cadena de suministro. Sin embargo, es esencial tener una clara comprensión del significado del IoT, en mayor parte por la confusión inherente al término en sí y por las aplicaciones cotidianas de esta tecnología. A primera vista, "Internet de las Cosas" podría dar la impresión de ser un término moderno para referirse a "conectar a Internet algo para controlarlo", una definición bastante simple para alguien que por ejemplo controla las luces de su hogar desde su teléfono móvil. Es tanta la confusión que no existe una definición formal única, sino que se pueden encontrar una disparidad de definiciones que no todos los sistemas IoT cumplen. En 2012, la Unión Internacional de Telecomunicaciones (ITU) recomendó definir el IoT como una infraestructura global que permite ofrecer servicios avanzados a todo tipo de aplicaciones conectando objetos entre sí e interoperando tecnologías de la información y comunicación, aprovechando capacidades de identificación,

obtención de datos, procesamiento y comunicación y cumpliendo con requisitos de seguridad y privacidad. Sin embargo, por lo general, podemos entender que el IoT trata de dotar de capacidades de comunicación además de procesamiento, captura y/o análisis de datos a distintos tipos de entes, como dispositivos físicos, objetos, edificaciones, terrenos, sistemas, hardware, software, e incluso contextos y situaciones, ya sea añadiéndoles dispositivos o integrando ciertas capacidades en los propios objetos. Estos entes pueden estar compuestos de sensores, que recopilan datos, o actuadores, que controlan otros objetos, y a través de redes, privadas o públicas, pueden intercambiar información con otros dispositivos, recopilar la información en un mismo dispositivo y transferir órdenes, además que existe la posibilidad de formar una agrupación de dispositivos para identificarlos como un único sistema que trata los datos. Estas redes de conexión de dispositivos IoT pueden utilizar la información transferida para poder automatizar sus comportamientos, pero el nodo final de la conexión suelen ser las personas, ya que utilizan los objetos como fuentes de información, para a su vez utilizar los datos recibidos como descubrimiento de oportunidades de negocio y nuevos servicios, monitorización y evaluación del estado y el comportamiento de los objetos y el entorno, y la toma de decisiones sobre los propios objetos manipulándolos remotamente y programándolos. Dependiendo del entorno en que se implemente, estos sistemas deben cumplir requisitos de seguridad y privacidad, evitando manipulaciones y accesos indebidos o incluso daños en los propios objetos y en el entorno que los rodea.

Con el objetivo de ilustrar cómo funciona un sistema IoT, la Figura 2.1 muestra un sistema de riego por aspersión inteligente en un jardín. El jardín está dividido en zonas, cada una con un dispositivo compuesto por sensores de temperatura y humedad y actuadores que activan y desactivan los aspersores que riegan la vegetación. Estos dispositivos están conectados y se comunican con un único gateway o puerta de enlace, teniendo la capacidad de recibir información de los dispositivos y mandar órdenes a estos. El gateway está comunicado a través de Internet con un servidor compuesto por una base de datos, donde se almacenan los datos históricos y registros, y un servicio que le permite ser controlado desde otro dispositivo conectado a Internet en cualquier sitio y momento. Este último dispositivo, denominado cliente, puede ser un teléfono móvil o un ordenador, y puede utilizarse para ver el estado del jardín y controlar los dispositivos manualmente desde una interfaz. La siguiente arquitectura, abstraída completamente de las limitaciones y los problemas que puede ofrecer su implementación, podrá dar lugar a dos casos de uso:

- El usuario quiere activar los aspersores. El usuario, desde la interfaz el dispositivo cliente como puede ser un botón, enviará la orden de activar los aspersores al servicio, que a su vez se lo enviará al gateway. El gateway enviará una orden compatible a los dispositivos instalados en el jardín, haciendo que los aspersores comiencen a funcionar. Esta secuencia de ejecución será parecida si el usuario desea desactivar los aspersores.
- Los aspersores funcionan cuando la temperatura es muy elevada y la humedad es baja. El usuario previamente, desde la interfaz del cliente, ha establecido que los aspersores funcionen de manera automática cuando, por ejemplo, el ambiente supere una temperatura de 42 ºC y la humedad sea con-

siderada baja. Estos parámetros los recibe el servidor y se los pasa al gateway, el cual los recordará. A partir de ese momento, el gateway irá recibiendo de los sensores de cada zona lecturas de temperatura y humedad, y las comparará con los parámetros establecidos. En el caso de que se superen la temperatura y humedad, se activarán los aspersores de la zona. Una vez activados, se mantendrá la lectura de temperatura y humedad, y cuando las lecturas sean inferiores, se desactivarán los aspersores. Además, con cada activación de los aspersores, el gateway notificará al servicio, que a su vez notifica al usuario, especificando la zona activada.

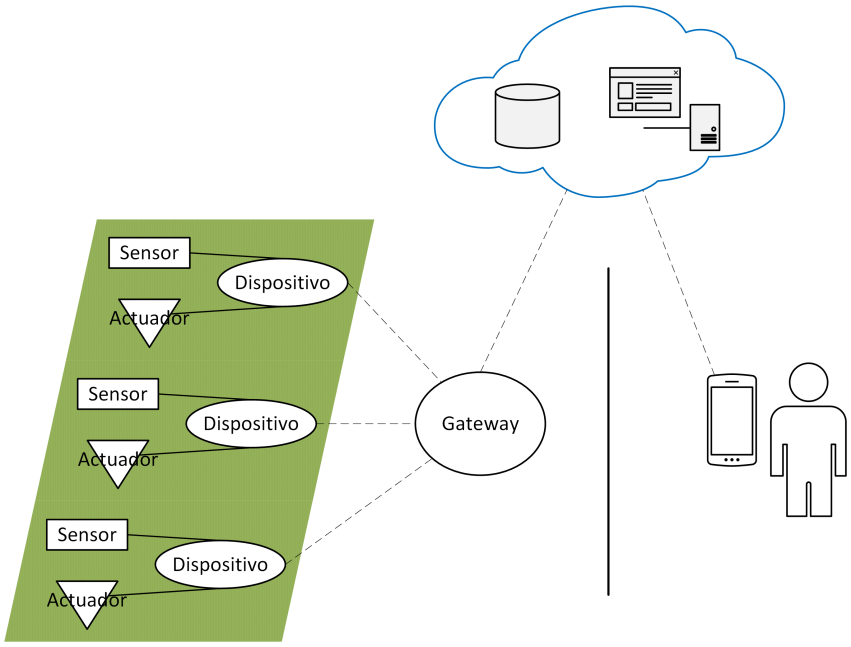

Figura 2.1: Esquema de un sistema de riego por aspersión utilizando dispositivos IoT

Cabe recalcar que el mencionado ejemplo puede volverse más complejo, por ejemplo añadiendo comprobaciones de previsión meteorológica en los próximos días, sensores de luz, control del caudal del agua, pero igualmente cumple con los requisitos para ser un sistema que implementa el IoT, ya que dota a un jardín de capacidad de comunicación, captura y análisis de datos, y se comunica con otros dispositivos y con personas para informar del estado y recibir órdenes.

2.1.1. Ventajas y usos

Hoy en día vivimos en un mundo conectado, en el cual estamos rodeados de numerosos dispositivos, gran parte de ellos interconectados como sistemas, ya que la información es crucial para la sociedad. Por ejemplo, a nivel empresarial es la que permite crear nuevas posibilidades, modelos, interacciones y soluciones únicas, entender y anticiparse a tendencias, identificar nuevas oportunidades, encontrar ventajas competitivas y detectar problemas internos, entre otras aplicaciones beneficiosas para la sociedad. Esta información se obtiene a partir del procesamiento y análisis de colecciones de datos, recopilados de diversas fuentes

potenciales, como es el caso de los sistemas IoT. El IoT permite conectar dispositivos a la red con el fin de enviar y transmitir datos de forma rápida y en tiempo real, mejorando la recopilación de datos del entorno y el control e interacción con el mismo. La rapidez permite que los datos y el control sean accesibles en cualquier momento y lugar, según requiera el usuario, siempre que el sistema haya sido configurado para ello. Sumado a lo anterior, tiene la característica de poder implementarse de diferentes maneras en una gran variedad de entornos, adaptándose al uso que se le vaya a dar y a otras necesidades.

El control y los datos generados tras aplicar técnicas adecuadas de análisis, como las que emplean inteligencia artificial, convierten al IoT en un factor fundamental en el funcionamiento de las empresas y en su transformación digital En términos generales, gracias al IoT es posible monitorear los procesos que ocurren en un negocio para controlar y optimizar los recursos utilizados, ya sean energéticos o de personal humano; además de encontrar tareas manuales o repetitivas que pueden ser automatizadas mediante el propio IoT, mejorando así la eficiencia, el tiempo y la productividad. Asimismo, permite la toma de decisiones informadas a partir de datos de negocio, tanto internos como externos, que permitan agregar valor al negocio de manera más rápida, y la mejora de experiencia y de calidad de vida de los usuarios o clientes basándose en los datos recibidos sobre estos.

En cuanto a tareas más específicas que IoT mejora y habilita, pueden estar la administración del inventario, localizar errores fácilmente, o incluso aún más específicas según el sector en el que se aplica:

- Sanidad: monitorizar signos vitales, hacer recomendaciones de salud adecuadas, comprobar autenticidad y dosis recomendadas en medicamentos.
- Agricultura y ganadería: controlar las condiciones del suelo y el crecimiento de los cultivos, localizar el ganado, controlar la calidad de la leche recolectada en un tanque.
- Transporte, conducción y logística: frenado automático de emergencia, cálculo de rutas óptimas, búsqueda de aparcamiento, mantener condiciones óptimas de remolques que transporten productos perecederos.
- Fabricación: controlar temperatura y humedad de centros de fabricación, identificar áreas con necesidad de personal, crear controles de calidad a productos.
- Comercio: identificar posición óptima de los productos en las estanterías, guiar a clientes hasta determinados productos, trazabilidad en el producto desde la llegada en el almacén hasta su venta.
- Ciudades y edificios inteligentes: mejor vigilancia y actuación por los equipos de emergencia, localizar áreas de contaminación, automatizar iluminación y calefacción.

El IoT no solo se reduce a estos sectores, es prácticamente aplicable a todos, como la educación, la construcción o el turismo, y a cualquier fase, desde el diseño hasta ofrecer y consumir el servicio o producto, gracias a la popularización de esta tecnología y al aumento de dispositivos que aparecen a nuestro alrededor.

{ https://es.wikipedia.org/wiki/Internet_de_las_cosas https://www.ibm.com/es-es/topics/internet-of-things

https://blog.ubisolutions.net/es/iot-tecnologias-casos-de-uso-ventajas-y-limitaciones-guia-completa https://www.tokioschool.com/noticias/ventajas-desventajas-internet-cosas/ https://kryptonsolid.com/principales-ventajas-y-desventajas-de-iot-en-los-negocios/ https://www.campusbigdata.com/big-data-blog/item/101-relacion-iot-con-big-data https://www.ibm.com/es-es/topics/internet-of-things https://www.nougir.com/index.php/blog-3/item/13-que-es-iot-o-internet-de-las-cosas-y-sus-aplicaciones https://innovayaccion.com/blog/aplicando-el-internet-de-las-cosas-a-las-empresas-2 https://cdn.ihs.com/www/pdf/IoT_ebook.pdf https://www2.deloitte.com/content/dam/Del estate/deloitte-nl-fsi-real-estate-smart-buildings-how-iot-technology-aims-to-add-value-for-real-estate-companies.pdf https://www.researchgate.net/publication/325373920_Internet_of_things_IoT_a_survey_on_architecture_enabling_
}

2.1.2. Factores, limitaciones y desafíos a tener en cuenta

Al desarrollar una arquitectura del IoT, es necesario tener en cuenta una serie de factores que afectan tanto a los dispositivos como a la comunicación entre ellos.

La primera consideración se refiere a los datos, el artefacto más importante en estos sistemas. Deben ser adecuados y el personal debe estar preparado para tratarlos y analizarlos. Sin embargo, debido a la gran cantidad de datos producidos, se presenta un reto significativo al analizarlos y extraer información, especialmente en casos de falta de herramientas o experiencia en análisis.

Dependiendo del dispositivo, el uso que se le da y el entorno donde se instala, puede verse limitado por varios factores energéticos y físicos interdependientes, tales como la manera de alimentarlo, el consumo (coste de la energía y tiempo de uso), el rango de movilidad, el peso y el tamaño. Dar preferencia a un factor puede perjudicar a otro, por lo que es necesario buscar un equilibrio que satisfaga todas las necesidades o priorizar algunas sobre otras y aplicar cambios en las propias necesidades. Por ejemplo, puedo tener un dispositivo instalado en un hueco pequeño que consuma poco pero que tenga una batería pequeña, o puedo hacer el hueco más grande para añadir una batería de mayor capacidad.

Las condiciones ambientales (como la lluvia, la humedad, las vibraciones y la temperatura), la disponibilidad de conexión en una zona (cobertura de red o electrificación), las interferencias electromagnéticas y otros factores del área donde se despliega el sistema pueden provocar un replanteamiento de ciertas partes. Por ejemplo, éstas pueden implicar la fiabilidad en el transporte de los datos mediante mecanismos de acuses de recibo y verificación de mensajes, o la frecuencia de transmisión y tamaño de los datos.

Debido a la heterogeneidad de los dispositivos en cuanto a fabricantes, características y capacidades, y a la falta de un estándar internacional de compatibilidad para dispositivos en IoT, es necesario dedicar un esfuerzo a identificar las tecnologías de comunicación compatibles (y que cumplan las necesidades de velocidad y alcance), configurar los dispositivos y usar dispositivos adicionales si es necesario. Esto convierte al sistema en una fuente de datos complicada de analizar e integrar en otros.

Es importante además adaptar el despliegue al uso que se le vaya a dar, como puede ser tener una topología de conexión y comunicación adecuada según los dispositivos que interactúen, prepararlo para una futura escalabilidad, habilitar la gestión remota de los dispositivos para actualizarlos, reconfigurarlos, localizarlos o desconectarlos, y realizar una correcta recolección de los datos.

Los dispositivos pueden recopilar datos sensibles de usuarios y empresas que son transmitidos constantemente, y los dispositivos destinados a IoT no suelen tener altas capacidades de procesamiento para establecer controles de seguridad. Además, los fabricantes pueden haberse centrado en optimizar los recursos, descuidando aspectos como la seguridad en el cifrado o en los controles de acceso. Junto a posibles creaciones de redes insuficientes, esto convierte a los sistemas IoT en un cebo para infiltraciones, sustracciones de datos, modificaciones de los datos transmitidos y corrupción de configuraciones y de dispositivos, entre otros tipos de ataques. Al implementar IoT, es preferible mantener una buena ciberseguridad y respetar la privacidad de los datos implementando medidas que aseguren la confidencialidad, la integridad y la disponibilidad (como el control de acceso, autenticación y cifrado), siendo importante conocer las regulaciones de la zona en la que opera el sistema para poder cumplirlas.

De manera general, los desarrolladores de software en estos sistemas tienen la responsabilidad de que tanto las capacidades como las limitaciones influyan en el código, aplicando métodos como la suspensión del funcionamiento del dispositivo cuando no está en uso o la optimización del código para reducir el consumo de recursos.

Como consideración final, el aumento de la brecha tecnológica que generan los sistemas IoT impacta en la sociedad y limita el grupo social que tiene la capacidad de comprender su funcionamiento y sabe acceder y hacer uso de esta tecnología correctamente. Esto implica una necesidad de educación hacia los usuarios.

Un sistema IoT se puede llevar a cabo en una gran cantidad de situaciones y entornos, pero es una complejidad que debe ser gestionada. Está conformado por dispositivos que por sí solos realizan tareas sencillas, pero al formar parte de un sistema requieren una inversión en dispositivos, una instalación correcta, un mantenimiento realizado por personal capacitado y tener en cuenta otros factores técnicos, del entorno y de la propia implementación. Para obtener una arquitectura eficiente que aporte los resultados esperados, es recomendable analizar los requisitos y las posibles limitaciones, listar las soluciones, diseñar la arquitectura del sistema, planificar un presupuesto, desarrollar una estrategia de implementación y de mantenimiento en caso de errores o mal funcionamiento, y realizar pruebas y evaluaciones. { “IoT fundamentals: Networking technologies, protocols, and use cases for the internet of things”. David Hanes, Gonzalo Salgueiro, Patrick Grossetete, Jerome Henry, Robert Barton https://www.redeweb.com/articulos/sensores/los-sensores-inalambricos-iot-y-el-problema-de-la-corta-duracion-de-las-baterias https://neurona-ba.com/iot-a-tu-alcance/ https://www.ittrends.es/infraestructura/2021/12/los-problemas-de-conectividad-estan-dificultando-los-despliegues-de-iot https://deingenierias.com/curso/iot/1-4-desafios-y-limitaciones-de-iot/ https://www.chakray.com/es/5-requesitos-de-una-arquitectura-iot/ https://www.incibe.es/incibe-

cert/blog/riesgos-y-retos-ciberseguridad-y-privacidad-iot https://es.wikipedia.org/wiki/Internet_de_las_cosas https://www.ibm.com/es-es/topics/internet-of-things https://blog.ubisolutions.net/es/iot-tecnologias-casos-de-uso-ventajas-y-limitaciones-guia-completa https://www.tokioschool.com/noticias/ventajas-desventajas-internet-cosas/ https://kryptonsolid.com/principales-ventajas-y-desventajas-de-iot-en-los-negocios/ }

2.1.3. Primeros ejemplos de IoT

El primer dispositivo IoT conocido fue una máquina de Coca-Cola conectada a ARPANET a principios de la década de los 80 en la Universidad Carnegie Mellon de Pittsburgh, Pensilvania. Esta máquina era mantenida por los usuarios de la universidad y tenía cierta popularidad que hacía que normalmente estuviera vacía o recién cargada con botellas calientes, lo que molestaba al departamento de Ciencias de la Computación. Para resolver este problema, instalaron microinterruptores en la máquina para detectar las botellas que había en cada ranura, los conectaron al ordenador Programmed Data Processor model 10 (PDP-10) del departamento y diseñaron un programa para registrar la hora cada vez que ocurría una transacción en cada ranura, así como otro programa para recibir información de la máquina. Utilizando el último programa desde la red del departamento, se podía consultar si había botellas en las ranuras, cuánto tiempo había transcurrido desde que se recargó con una nueva botella o si estaba fría.

Más tarde, hicieron público el acceso a esta información, haciendo que cualquier persona conectada a ARPANET en cualquier parte del mundo podía consultar el estado de la máquina de Coca-Cola durante más de una década utilizando el comando `finger coke@cmua`, hasta que la máquina fue reemplazada por ser incompatible con el nuevo diseño de las botellas de Coca-Cola.

Incluir imágenes: - Xerox Alto, ordenador fabricado en 1973, usado en la CMU para ver el estado de la maquina
- Máquina de cocacola https://engineercom.wpenginepowered.com/wp-content/uploads/2016/02/IIot1.png
- PDP-10

Posteriormente, en 1990, apareció otro notable objeto conectado a un Internet más similar al actual, una tostadora. La idea surgió en la edición de 1989 de Interop, una conferencia en la que se pone a prueba la interoperabilidad de dispositivos, cuando el organizador retó a John Romkey, cocreador de la primera pila de protocolos de Internet TCP/IP, a conectar un dispositivo a Internet. En esa época, Romkey trabajaba en el Protocolo Simple de Gestión de Redes o *Simple Network Management Protocol* (SNMP) junto a sus compañeros de Epilogue, un protocolo utilizado para leer y escribir variables en un agente remoto y comúnmente empleado para gestionar routers. Romkey quiso aprovechar la oportunidad para demostrar de forma sencilla que el SNMP podía controlar dispositivos físicos, con una tostadora. Utilizó un modelo de tostadora capaz de bajar el pan insertado al momento de recibir energía, lo cual hacía que necesitase controlar la alimentación del dispositivo en vez de emplear robótica que accione la típica palanca que tienen

las tostadoras comunes. La alimentación se controlaba con un relé, accionado a su vez por un interruptor accionado a través del puerto paralelo de su ordenador portátil, el cual recibía órdenes de un agente SNMP que detectaba los cambios. SNMP permite controlar dispositivos remotos definiendo objetos en su base de información gestionada (Management Information Base o MIB) con valores posibles, acceso permitido, estado y descripción. En este caso, se especificaron los objetos necesarios para que el usuario pudiera modificarlos a través de SNMP y así especificar la tostadora que se está controlando, el inicio del tostado, el tipo de pan y el nivel de tostado final.

La tostadora de Internet se presentó en la Interop de 1990, y se mejoró en la edición de 1991 añadiendo un brazo robótico de LEGO también controlado por Internet para insertar rebanadas de pan en la tostadora, consiguiendo así un sistema completamente automatizado. En la actualidad, la tostadora creada por Romkey sigue en su posesión.

- TODO: Incluir imágenes tostadora (sacarla de url)

Como último ejemplo, también de la misma época de Internet, está XCoffee o la cafetera de la Sala Trojan. En noviembre de 1991, Quentin Stafford-Fraser y Paul Jardetzky trabajaban junto a sus compañeros investigadores agrupados en el antiguo laboratorio de informática, también llamado Sala Trojan, de la Universidad de Cambridge, mientras que el resto de compañeros estaban dispersos por la universidad. Todos compartían un problema en común: una única máquina de café de goteo-filtro a la salida del laboratorio, cuya jarra estaba vacía o contenía café con un sabor horrible a excepción de cuando estaba recién hecho, siendo esta última opción solo para aquellos que se sentaban cerca. Para solucionar este problema, Stafford-Fraser y Jardetzky idearon el sistema XCoffee, y el Dr. Martyn Johnson y Daniel Gordon lo mejoraron. Fijaron una cámara apuntando a la jarra de la cafetera, la conectaron a una capturadora de fotogramas del ordenador Acorn Archimedes del rack de la sala y crearon un software para que cada vez que el servidor recibía una solicitud HTTP a través de la World Wide Web devolvía una página web con la imagen de la cafetera capturada más reciente. Desde el navegador se podía comprobar si valía la pena ir a por café (en el caso de estar en el mismo edificio que la cafetera), y esto convertía al sistema en la primera webcam de la historia, otorgándole una fama internacional. Estuvo operativa su última versión desde el 22 de noviembre de 1993 hasta el 22 de agosto de 2001 a las 09:54 UTC, cuando mostró su última imagen pocos segundos antes de su apagado debido a su difícil mantenimiento y al traslado de las instalaciones del laboratorio.

Actualmente, la máquina de café se encuentra restaurada y expuesta en el museo de informática Heinz Nixdorf MuseumsForum de Paderborn, Alemania.

https://www.nougir.com/index.php/blog-3/item/13-que-es-iot-o-internet-de-las-cosas-y-sus-aplicaciones https://informationmatters.net/internet-of-things-definitions/ (DOCUMENTO) https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=11559&lang=es (LIBRO) https://www.taylorfrancis.com/chapters/oa-edit/10.1201/9781003337584-3/internet-things-cognitive-transformation-technology-research-trends-

applications-ovidiu-vermesan-markus-eisenhauer-harald-sundmaeker-patrick-guillemin-martin-serrano-elias-tragos-javier-vali %C3 %B1o-arthur-van-derwees-alex-gluhak-roy-bahr (LIBRO) https://books.google.es/books/about/La_Re (DOCUMENTO) https://cdn.ihs.com/www/pdf/IoT_ebook.pdf (DOCUMENTO) https://www.researchgate.net/publication/352541 https://www.microsoft.com/insidetrack/blog/transforming-microsoft-buildings-with-iot-technology-and-indoor-mapping/ (LIBRO) https://www.se.com/us/en/download/document/998-20233517/ (DOCUMENTO) https://www.researchgate.net/publication/320135661_Efficient_IoT-based_Sensor_BIG_Data_Collection-Processing_and_Analysis_in_Smart_Buildings (DOCUMENTO) https://www2.deloitte.com/content/dam/Deloitte/nl/Documents/r estate/deloitte-nl-fsi-real-estate-smart-buildings-how-iot-technology-aims-to-add-value-for-real-estate-companies.pdf https://innovayaccion.com/blog/aplicando-el-internet-de-las-cosas-a-las-empresas-2 (DOCUMENTO) https://ieeexplore.ieee.org/document/7879243 https://www.intel.com/content/www/us/en/internet-of-things/overview.html https://www.fundacionbankinter.org/ftf-informes/internet-de-las-cosas/ https://www.statista.com/statistics/ number-of-connected-devices-worldwide/ (VIDEO) https://www.youtube.com/watch?v=RnasX1bFBh8 (LIBRO) https://www.amazon.com/IoT-Fundamentals-Networking-Technologies-Protocols/dp/1587144565 (LIBRO) https://www.ra-ma.es/libro/internet-de-las-cosas_93304/ https://es.wikipedia.org/wiki/Internet_de_las_cosas https://www.eexcellence.es/expertos/kevin-ashton-un-tecnologo-visionario https://www.redhat.com/es/topics/internet-of-things/what-is-iot https://at3w.com/blog/iot-internet-of-things-tecnologia-proteccion-contra-rayo-tomas-tierra/ https://www.itop.es/blog/item/iot-origen-importancia-en-el-presente-y-perspectiva-de-futuro.html https://www.linkedin.com/pulse/el-origen-del-internet-de-las-cosas-iot-comnet-s-a-/ https://www.ui1.es/blog-ui1/historia-y-origen-del-iot https://blog.avast.com/es/kevin-ashton-named-the-internet-of-things https://www.datacenterdynamics.com/en/podcasts/zero-downtime/episode-18-the-origin-of-the-internet-of-things-with-peter-lewis/ https://swifttalk.net/2021/10/06/the-concept-of-iot-internet-of-things/ https://www.cs.cmu.edu/~coke/ https://www.cs.cmu.edu/~coke/history_long.txt https://www.ibm.com/docs/es/aix/7.1?topic=protoc namefinger-protocol https://www.livinginternet.com/i/ia_myths_toast.htm https://en.wikipedia.org/wiki/John_Romkey https://ieeexplore.ieee.org/document/7786805 Romkey, J. (2017). Toast of the IoT: The 1990 Interop Internet Toaster. IEEE Consumer Electronics Magazine, 6(1), 116–119. doi:10.1109/mce.2016.2614740 https://romkey.com/about/ https://blog.avast.com/the-internets-first-smart-device https://en.wikipedia.org/wiki/Trojan_Room_coffe https://www.cl.cam.ac.uk/coffee/qsf/coffee.html https://www.cl.cam.ac.uk/coffee/coffee.html https://www.youtube.com/watch?v=u https://www.bbc.com/news/technology-20439301 https://www.historyofinformation.com/detail.php?id=1507 https://quentinsf.com/coffeepot/metcalfe/ https://www.cl.cam.ac.uk/coffee/qsf/switchoff.html https://owl.museum-digital.de/object/3761

2.1.4. Tipos de dispositivos y redes

Como se ha mencionado anteriormente, el IoT se basa en la transmisión de datos entre dispositivos. Estos datos pueden ser información recibida del entorno u órdenes de actuación, que salen o llegan a un dispo-

sitivo ubicado en un extremo de la red de conexión y que tiene conectado uno o ambos de los siguientes tipos de objetos:

- Sensores: miden una propiedad física y la representan digitalmente, siendo útiles para detectar cambios en el entorno. Tienen mayor precisión y capacidad de percepción que los órganos sensoriales humanos, y pueden integrarse en cualquier objeto físico para interpretar su entorno. Actualmente están experimentado una proliferación debido a su disminución en tamaño y coste.
- Actuadores: dispositivos que interpretan una señal eléctrica y desencadenan un efecto físico tras interpretarla, siendo útiles para producir cambios en el entorno. Si un sensor se puede comparar con los órganos sensoriales humanos, un actuador se puede comparar con las acciones que pueden ejecutar las extremidades.

En una red IoT, tanto sensores como actuadores son el corazón del sistema, ya que permiten interactuar con el mundo físico. Pueden ser parte de objetos inteligentes, dotados de procesador, fuente de energía y capacidad para comunicarse con otros componentes, y son objetos cotidianos que integran todo lo necesario para interactuar con el entorno.

Todos estos dispositivos heterogéneos están interconectados en una red que permite detectar, medir y actuar en relación con el entorno. Esta red, con el fin de ser efectiva y tolerante a fallos, debe pasar por un análisis de criterios como la topología de conexión entre los dispositivos, la distancia entre estos y cómo se procesan los datos.

Una red IoT, puede dividirse en capas según cómo se procesen los datos [34] [43]:

- Computación en la nube o *cloud*: los datos se procesan en un servidor central remoto, lejos de su origen. Requiere una conexión a Internet para su acceso, lo cual implica lidiar con la latencia de la red y el uso del ancho de banda. Ofrece una gran capacidad de procesamiento, almacenamiento y análisis, adecuada para aquellas aplicaciones que no requieren respuestas rápidas. Puede agrupar y procesar los datos de todo un sistema.
- Computación en la niebla o *fog*: los datos se procesan cerca de donde se originan, en nodos *fog* de la red local. Estos nodos evitan el uso de Internet para enviar datos a la nube, mejorando la eficiencia y ofreciendo una respuesta más rápida que la computación *cloud*. Se pueden distribuir varios nodos *fog* para jerarquizar la red, agrupandos y procesando datos de ciertos nodos.
- Computación en el borde o *edge*: los datos se procesan en los nodos donde se originan. Ofrece un procesamiento en tiempo real sin necesidad de comunicarse con otros dispositivos, liberando el uso de ancho de banda.

{ https://www.ibm.com/es-es/topics/internet-of-things (LIBRO, el de los apuntes) “IoT fundamentals: Networking technologies, protocols, and use cases for the internet of things”. David Hanes, Gonzalo Salgueiro, Patrick Grossetete, Jerome Henry, Robert Barton. https://www.cloudflare.com/es-es/learning/network-

layer/what-is-a-personal-area-network/ https://www.cloudflare.com/es-es/learning/network-layer/what-is-a-lan/ https://www.cloudflare.com/learning/network-layer/what-is-a-metropolitan-area-network/ https://www.cloudflare.com/learning/network-layer/what-is-a-wan/ https://www.gadae.com/blog/tipos-de-redes-informaticas-segun-su-alcance/ }

2.2. Middleware y protocolos de mensajería usados en IoT

Versión original: El uso más común de IoT es desplegar una arquitectura compuesta por varios dispositivos IoT. En mayor parte, estos dispositivos se designarán simplemente como dispositivos IoT, ya sean sensores o actuadores, mientras que habrá pocos dispositivos (al menos uno) con el rol de centro de mensajería, un intermediario encargado de retransmitir los datos a los dispositivos adecuados. Dependiendo del uso que se le dé a la arquitectura, estos objetos se comunicarán, y dependiendo de quién sea el emisor y el receptor, dan lugar a estos escenarios: - Un mensaje ha llegado al centro de mensajería IoT. Este mensaje es procesado y se actúa en consecuencia, por ejemplo, enviando la información necesaria a ciertos dispositivos IoT o almacenándola a una base de datos. - Un dispositivo IoT ha generado datos. Estos datos son procesados y luego enviados al centro de mensajería. - Un dispositivo IoT ha recibido datos del centro de mensajería. Estos datos son procesados y se actúa en consecuencia.

Versión 2:

El uso más común de IoT es desplegar una arquitectura compuesta por varios dispositivos IoT. En mayor parte, estos dispositivos se designarán simplemente como dispositivos IoT, ya sean sensores o actuadores, que emiten o reciben datos. Adicionalmente, habrá unos pocos dispositivos (al menos uno) con el rol de centro de mensajería, que actuarán como intermediarios encargados de retransmitir los datos a los dispositivos correspondientes.

/*TODO: elegir entre las dos versiones. El comentario de la v. original decia que era un caso de uso muy específico, pero es que* ***pongo que es un uso común****, es decir,* ***especifico que es concreto***/

Tras escoger una tecnología para conectar los dispositivos entre sí, en el desarrollo de aplicaciones IoT se requiere un protocolo de mensajería para intercambiar esos grandes volúmenes de datos que tratan.

Un protocolo de comunicación permite que los dispositivos se comuniquen y transmitan mensajes entre los dispositivos IoT y el centro de mensajería. Además, proporciona cierta fiabilidad a la comunicación, ya

que permite que los mensajes lleguen y sus datos sean entendidos y procesados correctamente.

La elección del protocolo se basa en cómo se adecua al escenario en el que se quiere implementar, considerando requisitos a tener en cuenta como la ubicación, las limitaciones físicas, el consumo y el coste. Por lo general, no cualquier protocolo de comunicación es apropiado. Los protocolos que se mencionan en este apartado se adecuan a la mayoría de escenarios IoT debido a su rapidez y su fácil implementación, y es posible escoger aquel que se adapte mejor a los requisitos.

2.2.1. Message Queue Telemetry Transport (MQTT)

El protocolo MQTT es uno de los más populares en el ámbito del IoT. Diseñado para ser ligero y adecuado para redes con ancho de banda limitado y dispositivos con pocos recursos, este estándar del comité técnico OASIS permite el transporte bidireccional de mensajes con datos entre múltiples dispositivos.

MQTT utiliza el patrón de comunicación publicación-suscripción. En este patrón, los publicadores categorizan los mensajes, y los suscriptores recibirán mensajes de las categorías de su interés, a diferencia de la comunicación directa en la que los participantes se envían los mensajes directamente. En el caso de MQTT, el patrón está basado en temas o *topics*, siendo posible que los suscriptores se interesen por uno o varios.

En una red MQTT, se definen dos roles principales: el broker o intermediario de mensajes y los clientes. El broker MQTT es un servidor comparable a una oficina de correos, que recibe todos los mensajes publicados por los clientes y los dirige a los clientes de destino apropiados. Por otra parte, un cliente es cualquier dispositivo conectado al broker a través de una red, y puede producir y recibir datos al publicar y suscribirse respectivamente. Este mecanismo es útil para compartir datos, controlar y gestionar dispositivos. Por ejemplo, un dispositivo cliente puede publicar datos de sensores y además recibir información de configuración o comandos de control. El enrutamiento de mensajes realizado por el broker proporciona transparencia de localización y desacoplamiento en el espacio, ya que el publicador no necesita conocer las direcciones de los suscriptores y los suscriptores no necesitan conocer al publicador, ambos interactúan únicamente con el broker.

Los mensajes están organizados en una jerarquía de temas o *topics*. Al publicar un mensaje, se publica en un tema específico, y en el caso de querer publicar en varios se deben realizar varias publicaciones. En cambio, un suscriptor puede suscribirse a un tema específico o a varios simultáneamente y recibirá una copia de todos los mensajes compatibles con los temas suscritos. La manera de indicar varios temas es mediante el uso de los siguientes caracteres comodín:

- Comodín de un nivel '+': coincide con un nivel de tema y puede utilizarse más de una vez en la especificación del tema.
- Comodín de varios niveles '#': coincide con cualquier número de niveles y debe ser el último carácter en la especificación del tema.

Por ejemplo, cuando se publica un mensaje en el tema "`edificioA/sensor1/temperatura`", el broker enviará una copia del mensaje los clientes suscritos a los temas "`edificioA/sensor1/temperatura` ↪ ", "`edificioA/+/temperatura`" y "`edificioA/#`", pero no a un cliente suscrito a "`edificioB`" o a "`edificioA/+/humedad`".

La transmisión de mensajes se realiza de forma asíncrona, sin detener la ejecución de ambos componentes a la hora de publicar o recibir, y se puede realizar una comunicación uno a muchos (un publicador y varios suscriptores), muchos a uno (varios publicadores y un suscriptor), uno a uno (un publicador y un suscriptor, menos común) y muchos a muchos (varios publicadores y varios suscriptores).

En el caso de que el broker reciba una publicación de un tema en el cual no hay nadie suscrito, el broker por defecto descarta el mensaje. Es posible activar la retención de mensajes configurando un campo en el mensaje para evitar esto, consiguiendo así que el broker almacene el último mensaje retenido de cada tema y lo distribuya inmediatamente a cualquier nuevo cliente suscrito, permitiendo así que el suscriptor reciba el valor más reciente en lugar de esperar a una nueva publicación, y además añadiendo soporte a una comunicación desacoplada en el tiempo, donde publicadores y suscriptores no necesitan estar conectados simultáneamente.

El protocolo soporta un mecanismo de limpieza de sesión. Por defecto, un cliente tras desconectarse y volverse a conectar no recibe los mensajes publicados durante su desconexión y el broker olvida las suscripciones del mismo cliente. Pero al desactivar dicho mecanismo, el broker mantiene tanto las relaciones de suscripción como los mensajes offline, enviándolos al cliente al momento de reconectarse, lo cual es útil para dispositivos que se conectan y desconectan constantemente, común en redes IoT. Además, MQTT enfrenta la inestabilidad de la red con un mecanismo *Keep Alive* que, al transcurrir un prolongado periodo sin interacción, ocurre un ping entre el cliente y el broker para evitar la desconexión. Si el ping falla y se identifica el cliente como desconectado, aplicará un mecanismo *Last Will*, que publica un último mensaje a un tema específico debido a una desconexión anormal, en el caso de estar configurado.

MQTT dispone de 14 tipos de mensajes diferentes, la mayoría utilizados para mecanismos internos y flujos de mensajes:

- CONNECT: establece una conexión con el broker, y si está configurado, se debe proporcionar un usuario y contraseña.
- DISCONNECT: finaliza una sesión MQTT enviando este mensaje para cerrar la conexión. Esta desconexión se denomina "graceful shutdown" o "apagado elegante", porque está la posibilidad de conectarse al broker con la misma sesión y reanudar el progreso.
- PINGREQ/PINGRESP: una operación de ping utilizada para saber si está viva la conexión y mantenerla.
- PUBLISH: contiene un mensaje para publicarlo en un tema específico.

- SUBSCRIBE: utilizado por los clientes para suscribirse a un tema específico y recibir las actualizaciones de este.
- UNSUBSCRIBE: mensaje que utiliza un cliente para indicar la pérdida de interés y anular la suscripción a un tema específico
- LWT: este mensaje "last will and testament" (última voluntad y testamento) se configura en un cliente para publicarse automáticamente si ocurre una desconexión inesperada. El broker mantiene un temporizador, y si comprueba que recientemente el cliente no ha publicado ni ha mandado un PINGREQ, se publica el mensaje LWT especificado notificando así a los suscriptores.

El diseño de MQTT se basa en la simplicidad y en minimizar el ancho de banda, dejando la interpretación del contenido del mensaje en manos del desarrollador. Los mensajes transmitidos a través de la red tienen la posibilidad de configurar el QoS o calidad de servicio por cada tema, asociados con distintas garantías de entrega. Aunque MQTT funciona sobre TCP, el cual tiene su propia garantía de entrega, históricamente los niveles QoS eran necesarios para evitar la pérdida de datos en redes antiguas y poco fiables, una preocupación válida para las redes móviles actuales. Estos son los siguientes tipos de QoS:

- QoS 0, a lo sumo una vez: los mensajes se envían y no se tiene en cuenta si llegan o no. Está la posibilidad de la pérdida de mensajes y no se hacen retransmisiones.
- QoS 1, al menos una vez: el receptor recibe el mensaje por lo menos una vez. El receptor debe enviar un acuse de recibo al emisor en cuanto reciba el mensaje, y si este ACK nunca llega (ya sea debido a que el mensaje nunca llegó o que el ACK se perdió), el emisor retransmitirá el mensaje, pudiendo producirse mensajes duplicados.
- QoS 2, exactamente una vez: asegura que el mensaje llegue exactamente una vez, manejado mediante la sobrecarga en la comunicación y el envío de una serie de acuses de recibo, y es la mejor opción cuando no se aceptan ni la pérdida ni la duplicidad de mensajes.

La transmisión de datos se realiza principalmente sobre la capa TCP/IP, pero existe la posibilidad de operar encima de otros protocolos de red que ofrezcan conexiones ordenadas, sin pérdidas y bidireccionales, y se transmiten en un tamaño reducido de paquetes de datos, estructurado por los siguientes campos mostrados en la figura 2.2:

- Cabecera fija, en la que se especifica el tipo de mensaje, si el mensaje es un duplicado, el QoS, si es un mensaje que retener y el tamaño del paquete.
- Cabecera variable, no siempre presente en los mensajes, y puede transportar información adicional de control.
- Payload o carga útil.

Por defecto, este protocolo envía credenciales y mensajes en texto plano sin medidas de seguridad, pero admite utilizar conexiones TLS/SSL protegidas por certificado, nombre de usuario y contraseña para cifrar

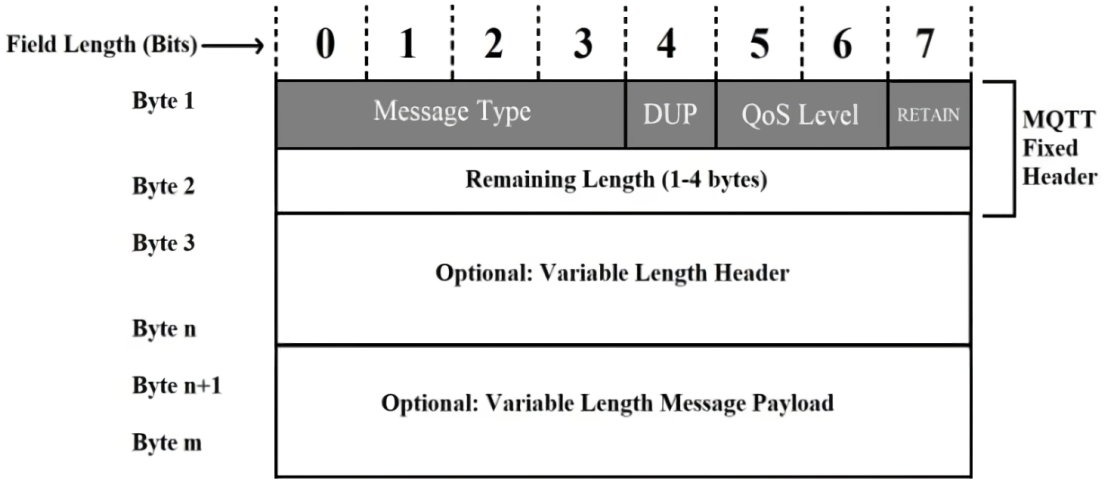

Figura 2.2: Formato de una trama MQTT (Fuente: [37])

y proteger la información transferida contra la intercepción, modificación o falsificación. Además, un broker MQTT tiene soporte para conectar dispositivos IoT a escala masiva, un factor tenido en cuenta durante su diseño y que resulta en una alta capacidad de concurrencia, rendimiento y escalabilidad, características deseables en una red IoT. Implementaciones como EMQX[1] y HiveMQ[2] han alcanzado hitos notables, con 100 y 200 millones de conexiones respectivamente, y un pico de 1 millón de mensajes gestionados por segundo. A esto se le puede sumar la capacidad de implementar múltiples brokers, para así compartir la carga de clientes y tratar la redundancia y la copia de seguridad de los mensajes en caso de fallo.

{ https://en.wikipedia.org/wiki/MQTT https://en.wikipedia.org/wiki/Publish %E2 %80 %93subscribe_pattern https://www.emqx.com/en/blog/what-is-the-mqtt-protocol https://www.gotoiot.com/pages/articles/mqtt_intro/index.html https://dzone.com/refcardz/getting-started-with-mqtt https://www.hivemq.com/resources/achieving-200-mil-concurrent-connections-with-hivemq/ http://www.steves-internet-guide.com/mqtt-protocol-messages-overview/ }

2.2.2. Advanced Message Queuing Protocol (AMQP)

AMQP es un protocolo binario avanzado que opera sobre la capa de aplicación, cuyo estándar abierto permite desarrollar mensajería y patrones de comunicación entre dispositivos. Facilita la comunicación entre servicios definiendo el formato de los datos enviados a través de la red como un flujo de bytes, así como la creación de mensajes, el encolamiento y enrutamiento de los mensajes producidos, y la manera de entregarlos a los consumidores.

Este protocolo se basa en el concepto de publicar y consumir de colas de mensajes a través de una conexión

[1] https://www.emqx.com/en
[2] https://www.hivemq.com/

fiable, persistente y orientada al envío de flujos de datos. Además, es compatible con el envío de múltiples flujos de datos simultáneos mediante múltiples canales en una única conexión. Permite hacer uso de estas colas de mensajes mediante distintos tipos de comunicación, como la entrega directa punto a punto y el modelo publicación-suscripción, y, pese a no ser un protocolo diseñado originalmente para su uso en el IoT, funciona muy bien en este ámbito y en la gran variedad de escenarios de comunicación y mensajería posibles.

En AMQP se definen las dos siguientes entidades principales que interactúan entre sí:

- Cliente: del tipo suscriptor o publicador (o consumidor y productor, respectivamente), se conecta a un broker a través de credenciales y, en caso de estar autorizado, puede recibir o publicar mensajes.
- Broker: servidor de mensajería al que los clientes se conectan y que se encarga de distribuir los mensajes. Internamente, posee *exchanges* o intercambiadores, donde se conectan los productores de mensajes, y colas, vinculadas a los exchanges dependiendo de varios criterios y a las que se conectan los consumidores para extraer los mensajes producidos.

De manera sencilla, se puede resumir el funcionamiento de este protocolo como un modelo en el que los mensajes son publicados y enviados a exchanges, los cuales enrutan los mensajes a las colas apropiadas según reglas o bindings, y los consumidores reciben los mensajes a través de las mismas colas.

Los exchages, además de recibir mensajes de los productores, se encargan de enviar los datos a las colas apropiadas, ya sean a una o a varias dependiendo del exchange y la clave de enrutamiento o routing key con la que se publica el mensaje. A este funcionamiento se le puede asociar la analogía del funcionamiento del envío de emails, ya que estos se envían a direcciones “routing key@exchange”, siendo la clave de enrutamiento la dirección de correo y el exchange el servidor. En AMQP existen cuatro tipos de exchanges:

- Topic, que permite una comunicación publicación-suscripción: los mensajes se enrutan a las colas adecuadas en función de la clave de enrutamiento y el patrón de vinculación al exchange.
- Direct, que permite una comunicación punto a punto: los mensajes se reciben en un exchange con una clave de enrutamiento específica y son enrutados directamente a una cola creada con la misma clave. Es posible que una cola tenga varias claves, conectándose a varios exchange simultáneamente, al igual que varias colas pueden compartir una misma clave de enrutamiento, enviando los mensajes a varios destinatarios.
- Fanout, que permite una comunicación parecida a broadcast: los mensajes son enrutados a todas las colas vinculadas al exchange.
- Cabeceras, que permite una comunicación publicación-suscripción: los mensajes se encaminan a las colas dependiendo de la coincidencia de las cabeceras de los mensajes.

Las colas son fragmentos de memoria creados por el cliente suscriptor identificadas unívocamente mediante un nombre, definido previamente por el cliente o automáticamente por el broker. Intrínsicamente, AMQP

garantiza la recepción y procesamiento de mensajes, ya que dispone de un mecanismo de ACK o acuse de recibo que permite confirmar la recepción y procesamiento del mensaje. En el caso de no recibir el ACK de un mensaje por parte de un consumidor, por ejemplo, porque perdió la conexión o no se procesó correctamente, el broker encola de nuevo el mensaje para reintentar la entrega. Junto a este mecanismo, también está la posibilidad de rechazar mensajes mediante un mensaje NACK o acuse de recibo negativo, útil para indicar que no se ha procesado bien el mensaje, pero que el broker puede borrar el mensaje de esa cola. Además, las colas admiten persistencia, para mantener la existencia de la cola incluso luego de que ocurra un reinicio en el broker.

Tras crear el exchange y la cola, se indica al broker la vinculación de ambas mediante un binding, especificando una clave de enrutamiento que, según el tipo de exchange, enruta adecuadamente las colas. El exchange entregará como máximo una copia de mensaje a una cola si corresponden las propiedades del mensaje con las propiedades del binding. Con los bindings es posible vincular varias colas a un mismo exchange, al igual que una cola a varios exchanges.

Con los tres conceptos mencionados previamente, exchange, binding y cola, se produce un desacople en tiempo y espacio entre los productores y los consumidores, ya que ambos están aislados entre sí y desconocen su existencia y ubicación.

Este protocolo especifica el comportamiento del servidor y de los clientes conectados, junto al formato de los mensajes enviados a través de la red, permitiendo interoperar independientemente del lenguaje o del proveedor. Los mensajes se transmiten sobre la capa TCP, y disponen de propiedades como las siguientes:

- Clave de enrutamiento.
- Modo de entrega, si es un mensaje persistente.
- Prioridad del mensaje en un rango entre 0 y 9.
- Vencimiento, indicado en milisegundos, es el tiempo que el broker debe esperar antes de descartar el mensaje si no fue consumido.
- Contador de intentos de entrega.
- Anotaciones de mensaje.
- Propiedades, como el ID del mensaje, usuario y tiempo de creación.
- Payload o carga útil.
- Footer, que contiene detalles del mensaje como hashes o firmas.

AMQP es extenso en cuanto a funciones, ofreciendo cifrado de extremo a extremo, múltiples propiedades de mensajes y modos de entrega, fiabilidad en la entrega, persistencia de mensajes, enrutamiento basado en criterios, capacidad de escalabilidad y definición de topologías, entre otros. Sin embargo, pese a su idoneidad para sistemas distribuidos, es un protocolo que conlleva un alto consumo de recursos como energía y memoria.

2.2.3. Extensible Messaging and Presence Protocol (XMPP)

Otra manera de comunicar varios dispositivos IoT es utilizar el XMPP, anteriormente conocido como Jabber. Este protocolo se basa en la transmisión de datos estructurados en formato XML dentro de una red de arquitectura cliente-servidor, en la cual los dispositivos están identificados por un Jabber ID, cuyo formato es similar al de una dirección de correo electrónico (por ejemplo, "abc@example.com"). En esta red, el cliente establece una conexión TCP/IP con el servidor. Posteriormente, el cliente se autentica con el servidor, y tras una autenticación exitosa, se habilita la posibilidad de enviar y recibir mensajes.

Una característica notable de XMPP es que cualquiera puede tener su propio servidor XMPP, no restringiendo a los usuarios a conectarse únicamente con otros usuarios en el mismo servidor central. Al ser un protocolo abierto formalizado por la Internet Engineering Task Force (IETF)[3], los desarrolladores disponen de un protocolo bien documentado y fiable, de este modo, es posible interoperar entre diferentes implementaciones de XMPP a través de Internet, independientemente del proveedor. En el caso de querer comunicarse con otro servidor, ambos servidores XMPP intercambian la información necesaria, habilitando un modelo federado.

Este protocolo está diseñado para ofrecer mensajería instantánea o casi en tiempo real a través de la red, sin importar la distancia entre los dispositivos, uno de los problemas más comunes en IoT. Además, permite obtener información sobre los usuarios conectados y mantener una lista de contactos para cada usuario. XMPP también es extensible, permitiendo a los desarrolladores añadir características y funcionalidades personalizadas, adaptando el protocolo a necesidades específicas de aplicaciones, como la transmisión de señales VoIP, video, ficheros, chat grupal, conferencias multiusuario, suscripción de presencia y comunicación publicación-suscripción para recibir actualizaciones sobre temas específicos de interés.

En los mensajes XMPP se utilizan estructuras XML denominadas *stanzas* para transportar los datos. Existen 3 tipos principales de stanzas:

- Stanza de mensaje (`message`): utilizado para enviar mensajes instantáneos entre clientes. Contiene los campos remitente, destinatario, cuerpo del mensaje y otros metadatos opcionales. Después de recibir el mensaje, el servidor utiliza el campo de destinatario para enrutar el propio mensaje.
- Stanza de presencia (`presence`): permite a las entidades conocer el estado y la disponibilidad online/offline de otros clientes. También puede transportar información adicional, como la actividad del cliente o su ubicación. Cuando un cliente se conecta o desconecta del servidor, envía una stanza de presencia para notificar a otros clientes de su lista de contactos.
- Stanza de IQ o info/query (`iq`): se usa para consultar al servidor, gestionar suscripciones o intercambiar datos estructurados entre clientes y servidores. Funciona de manera similar a los métodos HTTP GET y POST, siguiendo un patrón de petición-respuesta, en el cual un cliente envía una petición al

[3] https://www.ietf.org/

servidor y este responde con la información solicitada o con una confirmación.

El protocolo XMPP es altamente escalable debido a su capacidad de manejar multitud de conexiones y mensajes simultáneos. Además, al ser descentralizado, permite implementar fácilmente más servidores para gestionar el aumento de usuarios y altos picos de uso. En cuanto a seguridad, XMPP es compatible con cifrado de extremo a extremo mediante TLS o SSL, garantizando así la confidencialidad de los mensajes. Por último, cuenta con una amplia comunidad de usuarios, diversas implementaciones y guías que facilitan a los desarrolladores la creación de aplicaciones que integren este protocolo.

2.2.4. Data Distribution Service (DDS)

DDS es un estándar de middleware y API máquina-máquina que facilita la comunicación y el intercambio de datos. Fue desarrollado por el Object Management Group con el fin de responder a las necesidades específicas de aplicaciones que requieren intercambios de datos fiables y de alto rendimiento en sistemas distribuidos en tiempo real, sin dejar de lado la eficiencia. Su arquitectura se basa en un modelo de publicación-suscripción sin servidor, ya que los dispositivos se conectan entre sí, y donde los datos son publicados en un dominio y los suscriptores se conectan a este para recibir la información que les interesa.

Este middleware se corresponde con la capa de software que se encuentra entre el sistema operativo y las aplicaciones, abstrayendo la comunicación entre ambos y, por tanto, los detalles del tranporte de red y de los datos a bajo nivel. Permite que los distintos componentes de un sistema compartan y comuniquen datos, gestionando automáticamente el formato de los mensajes, el protocolo de transporte a usar, la fiabilidad, la calidad de sevicio y la seguridad, y simplificando así el desarrollo. DDS se centra completamente en los datos, asegurando un buen transporte e incluyendo información contextual de los mismos, lo que lo hace ideal para el IoT aplicado en entornos industriales.

DDS funciona con el concepto de espacio de datos global, un almacén de datos que a ojos del programador parece una memoria local accedida mediante APIs. Sin embargo, este espacio es solo una ilusión, ya que no existe un lugar donde residan todos los datos, se refiere a una colección de distintos almacenes locales en cada nodo por los cuales se reparten los datos. El programador piensa que está enviando mensajes a un almacén global, pero en realidad DDS envía mensajes a los almacenes locales apropiados.

Aunque este espacio global es tan característico, no se pierde compatibilidad, ya que el middleware es independiente del lenguaje de programación y la plataforma, posibilitando la interoperabilidad entre distintos sistemas y aplicaciones y una implementación que se adapte a las necesidades sin afectar a las comunicaciones entre dispositivos. Tampoco pierde efectividad, ya que su velocidad, baja latencia, baja sobrecarga en la comunicación, optimización del transporte y capacidad de transmitir millones de mensajes a multitud de receptores instantáneamente, lo convierten en una tecnología ideal para sistemas de alto rendimiento en tiempo real.

Al ser descentralizado, es decir, al no requerir un nodo central, el servicio DDS es mucho más eficiente y eficaz, ya que los mensajes no deben atravesar un dispositivo intermediario, ejecutando la comunicación punto a punto, y resultando en una mayor velocidad. Además, dispone de un servicio de descubrimiento dinámico, haciendo que las aplicaciones DDS sean extensibles y escalables. La aplicación no necesita conocer ni configurar los puntos finales de los dispositivos para la comunincación, ya que estos se descubren automáticamente en ejecución, y es capaz de identificar si el punto final se utiliza para publicar datos, para suscribirse o para ambos, el tipo de dato publicado o suscrito, y las características específicas de la comunicación.

DDS soporta principalmente el modelo de publicación-suscripción, intercambiando información basada en un tema o topic identificado por su nombre. En este modelo, cualquier nodo conectado puede publicar mensajes con el tema especificado o suscribirse a un tema, y DDS se encarga de que los datos se entreguen a los suscriptores correctos en el momento adecuado mediante comunicación peer-to-peer. Al suscribirse, es posible especificar filtros de tiempo y subconjuntos de datos para obtener solo los requeridos, y tiene la capacidad de detectar cambios para que los suscriptores reciban actualizaciones adecuadas de los datos. Por otro lado, al publicar, es DDS quien gestiona la complejidad de la transmisión y se encarga de almacenar los datos de manera segura. También ofrece compatibilidad con RPC o llamadas a procedimientos remotos.

El middleware es independiente del transporte, y puede funcionar sobre UDP, TCP y memoria compartida, entre otros. Entre las características opcionales que ofrece, como el filtrado de grandes datos, se encuentra la gestión de calidad de servicio o QoS, donde se pueden especificar requisitos de rendimiento y confiabilidad, como la latencia, el ancho de banda, la prioridad, la disponibilidad de los datos, el uso de recursos y la sincronización. Además, incluye mecanismos de seguridad que proporcionan autenticación, encriptación, confidencialidad, control de acceso e integridad en la distribución de información.

2.2.5. Constrained Application Control (CoAP)

CoAP es un protocolo de la capa de aplicación que permite a dispositivos con recursos limitados, como los que se encuentran en una red IoT, comunicarse entre sí. Funciona en un marco cliente-servidor, en el cual el cliente realiza una solicitud a un punto de comunicación del dispositivo servidor, y este responde, permitiendo la interoperabilidad entre los dispositivos uno a uno.

Este protocolo opera sobre el protocolo de transporte UDP, que, a diferencia de TCP, no requiere que los dispositivos establezcan una conexión de datos previa al envío de datos. Esto trae tanto consecuencias positivas como negativas. La consecuencia negativa radica en la poca fiabilidad en la comunicación, ya que el protocolo UDP no garantiza la entrega de los mensajes, sino que esta garantía se gestiona desde la implementación de CoAP. Es posible establecer garantía de entrega mediante acuses de recibo (ACK) y tiempos de espera. La consecuencia positiva del uso de UDP es la posibilidad de funcionar en redes con pérdidas o inestables, adecuado para sistemas IoT, ya que suelen operar en entornos de red difíciles, y

la rapidez en la comunicación, pues no requiere una conexión de datos previa, enviando directamente el mensaje.

Esta comunicación utiliza una arquitectura RESTful, en la cual los datos y las funcionalidades se consideran recursos a los que se accede mediante una interfaz estándar y uniforme. Estos recursos se acceden y se interactúa con ellos mediante métodos HTTP estándar (GET, POST, PUT, DELETE, que realizan las funciones "obtener", "crear", "actualizar" y "eliminar" recursos, respectivamente), permitiendo una interoperabilidad sencilla entre distintos tipos de dispositivos y facilitando a los desarrolladores el uso del protocolo. No es necesario que los recursos de la red sean conocidos por el dispositivo que vaya a utilizarlos, ya que CoAP implementa un mecanismo de descubrimiento integrado, útil en redes IoT en las que los dispositivos constantemente se conectan y desconectan. Esta función de descubrimiento trata de consultar un recurso conocido como "núcleo" en la red, el cual provee la lista de los recursos de los dispositivos en la red. Es decir, si un dispositivo IoT quiere interactuar con los recursos de otro, puede consultar al núcleo y comprobar qué recursos hay disponibles y cómo puede interactuar con ellos.

El intercambio de mensajes CoAP entre dispositivos es asíncrono, lo que significa que un dispositivo puede enviar una solicitud a otro y continuar ejecutando otras tareas mientras que la respuesta puede recibirla en cualquier momento. Esto se logra mediante un id en los mensajes, permitiendo al dispositivo relacionar peticiones con respuestas, asegurando un alto nivel de fiabilidad en el intercambio de mensajes. Esta comunicación asíncrona es crucial en redes IoT, ya que los dispositivos pueden no estar siempre conectados o disponibles para responder en el momento de la solicitud.

CoAP se basa en el intercambio de mensajes compactos codificados en un formato binario simple. El tamaño de estos mensajes no puede superar al necesario para encapsularlos dentro de un datagrama IP, y tienen distintos campos:

- Versión de CoAP.
- Tipo de mensaje.
- Longitud del Token.
- Código, en formato "c.dd", siendo "c" la clase indicando si es una solicitud, una respuesta satisfactoria, un error del cliente o un error del servidor, y "dd" el detalle.
- ID de mensaje.
- Token, usado para correlacionar solicitudes y respuestas.
- Opciones.
- Payload o carga útil.

Los distintos tipos de mensajes que se pueden transmitir son los siguientes:

- Mensajes confirmables (CON): utilizados cuando se necesita asegurar que el mensaje llegue al destinatario. Contienen un temporizador y un mecanismo de retroceso. Al transmitir una petición CON,

se espera recibir un mensaje ACK con el mismo ID de la petición o una respuesta en un mensaje CON y con un ID distinto.

- Mensajes no confirmables (NON): son mensajes menos fiables, usados para enviar información no crítica, que no requieren un acuse de recibo (ACK). En el caso de enviarse una solicitud como un mensaje NON, la respuesta también se recibirá como un mensaje NON (en el caso que el servidor tenga la información necesaria para responder).
- Mensajes de acuse de recibo (ACK): son transmitidos para reconocer que ha llegado un mensaje confirmable específico identificado por su ID de transacción. Estos mensajes pueden tener su propio payload y algunas opciones para detallar la recepción.
- Mensaje de reinicio (RST): cuando al receptor le falta información para procesar una solicitud, transmite un mensaje RST. Esto ocurre cuando el receptor se ha reiniciado y no ha persistido adecuadamente la petición recibida anteriormente, o cuando cancela una transacción.

Además es un protocolo diseñado para utilizar un bajo consumo de recursos en la transferencia, y permite hacer uso del protocolo de seguridad de la capa de transporte (Data Transport Layer Security, DTLS) para aumentar la seguridad, además de extender su implementación con funcionalidades adicionales. Por el contrario, es menos maduro que sus alternativas, resultando en una menor cantidad de recursos, guías y herramientas, además de una compatibilidad reducida con otros dispositivos IoT.

https://webbylab.com/blog/mqtt-vs-other-iot-messaging-protocols-detailed-comparison/ https://www.techtarget.com/iotagenda/ti
12-most-commonly-used-IoT-protocols-and-standards https://build5nines.com/top-iot-messaging-
protocols/ https://www.a3logics.com/blog/iot-messaging-protocols/ https://en.wikipedia.org/wiki/Advanced_Message_Queuing_
https://www.gotoiot.com/pages/articles/amqp_intro/index.html https://www.emqx.com/en/blog/mqtt-vs-
amqp-for-iot-communications#what-is-amqp http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-
messaging-v1.0-os.html#section-message-format https://en.wikipedia.org/wiki/XMPP https://www.pubnub.com/guides/xmpp/
https://blazeclan.com/blog/xmpp-for-dummies-part-3-stanzas-in-detail/ https://slixmpp.readthedocs.io/en/latest/api/stanza/presen
https://slixmpp.readthedocs.io/en/latest/api/stanza/iq.html https://www.dds-foundation.org/omg-dds-
standard/ https://www.utpl.edu.ec/proyectomiddleware/?q=tutorial-dds https://www.emqx.com/en/blog/coap-
protocol0 https://www.gotoiot.com/pages/articles/coap_intro/index.html (DOCUMENTO) https://datatracker.ietf.org/doc/html/rf

2.2.6. Comparación de middleware y protocolos

A partir de la información de los apartados anteriores, se pueden recopilar las diferencias y similitudes en los siguientes puntos:

1. *MQTT*:
   - **Descripción corta:** Basado en colas de mensajes y topics.
   - **Patrón de comunicación:** Publicación-suscripción.
   - **Necesita intermediario:** Sí, el broker.

- **Protocolo de transporte:** TCP.
- **Ventajas:** Muy utilizado, ligero, eficiente, útil en dispositivos y redes de recursos limitados, soporta distintas calidades de servicio.
- **Desventajas:** Encriptación y enrutación limitadas, bajo soporte para tipos de datos complejos.
- **Casos de uso:** Telemetría, mensajería ligera, automatización industrial, monitorización del entorno, hogares inteligentes, soluciones energéticas.

2. *AMQP*:
   - **Descripción corta:** Basado en colas de mensajes y exchanges.
   - **Patrón de comunicación:** Publicación-suscripción, directa o fanout.
   - **Necesita intermediario:** Sí, el broker.
   - **Protocolo de transporte:** TCP.
   - **Ventajas:** Alto rendimiento, seguro, ampliamente usado, parecido a MQTT.
   - **Desventajas:** Alto consumo de recursos, difícil aprendizaje.
   - **Casos de uso:** Servicios financieros, procesamiento de transacciones, envío de datos en tiempo real, comunicación segura entre entidades.
3. *XMPP*:
   - **Descripción corta:** Comunicación de datos y presencia mediante mensajes XML.
   - **Patrón de comunicación:** Cliente-servidor.
   - **Necesita intermediario:** Sí, el servidor XMPP.
   - **Protocolo de transporte:** TCP.
   - **Ventajas:** Robusto, extensible, muy ampliable de funciones.
   - **Desventajas:** No optimizado para entornos limitados, complejo de implementar.
   - **Casos de uso:** Mensajería instantánea, redes sociales, plataformas de colaboración.
4. *DDS*:
   - **Descripción corta:** Comunicación sin servidor.
   - **Patrón de comunicación:** Publicación-suscripción.
   - **Necesita intermediario:** No.
   - **Protocolo de transporte:** TCP y UDP, compatible con otros.
   - **Ventajas:** Alto rendimiento, fácil de escalar, comunicación centrada en datos en tiempo real, soporte de tipos de datos complejos, configurable la calidad de servicio.
   - **Desventajas:** Complejo de implementar, altos recursos de dispositivos y de ancho de banda.
   - **Casos de uso:** Sistemas de tiempo real, aplicaciones de misión crítica, automatización industrial, automoción, gestión de cadenas de suministro, sistemas distribuidos de gran escala.
5. *CoAP*:
   - **Descripción corta:** Basada en recursos compartidos y accesibles mediante REST.
   - **Patrón de comunicación:** Cliente-servidor.

- **Necesita intermediario:** No.
- **Protocolo de transporte:** UDP.
- **Ventajas:** Eficiente en redes y dispositivos de recursos limitados, soporte nativo para tecnologías web.
- **Desventajas:** Bajo soporte de clientes concurrentes.
- **Casos de uso:** Dispositivos de recursos limitados, automatización en hogares.

{ https://webbylab.com/blog/mqtt-vs-other-iot-messaging-protocols-detailed-comparison/ https://www.a3logics.com/blog/iot-messaging-protocols/

}

2.3. Espressif y sus dispositivos

Espressif Systems[4] es una multinacional de semiconductores sin fábrica fundada en 2008, que opera como líder mundial en el ámbito del IoT y está comprometida en proporcionar a millones de usuarios algunos de los mejores dispositivos y plataformas de software de la industria, junto con una variedad de soluciones IoT seguras.

La empresa se identifica como una empresa compuesta por especialistas, ingenieros y científicos dedicados al desarrollo de soluciones de vanguardia de bajo consumo que aprovechan la comunicación inalámbrica, el IoT y la inteligencia artificial de las cosas (AIoT). Estas soluciones se caracterizan por su seguridad, robustez, eficiencia energética, versatilidad, asequibilidad y enfoque código abierto.

Con el surgimiento de la inteligencia artificial y la evolución del IoT, la demanda de productos con conectividad inalámbrica segura ha ido creciendo considerablemente, y Espressif Systems ha respondido a este desafío desarrollando soluciones adaptadas a las necesidades del mercado. Espressif emplea los nodos de tecnología avanzada, la informática de bajo consumo, la comunicación inalámbrica, así como la tecnología de malla, para crear conjuntos de chips y módulos de alto rendimiento, que son más inteligentes, adaptables y versátiles.

El compromiso de esta empresa china con el código abierto se refleja en su oferta de una variedad de frameworks y herramientas de desarrollo para construir aplicaciones en diferentes ámbitos, como IoT, audio, reconocimiento facial y asistentes de voz. Las tecnologías y soluciones abiertas de Espressif permiten acercar el IoT a sus clientes, comerciales y no comerciales, y que desarrolladores de todos los ámbitos puedan utilizarlas a nivel mundial y construir sus propios dispositivos inteligentes y soluciones con una conectividad inalámbrica de rendimiento óptimo y un tiempo de desarrollo reducido.

Los productos de Espressif se utilizan ampliamente en dispositivos de electrónica de consumo, y es cono-

[4] https://www.espressif.com/

cido por sus populares series de chips, módulos y placas de desarrollo ESP8266 y ESP32, los cuales se analizarán en los siguientes apartados.

{ https://www.espressif.com/en/company/about-espressif https://www.eurotronix.com/es/fabricantes/espressif https://www.digikey.es/es/supplier-centers/espressif-systems }

2.3.1. ESP8266

El ESP8266 es un SoC o sistema en un chip diseñado para dispositivos móviles, electrónica portátil y aplicaciones del IoT. Lanzado en agosto de 2014, integra un procesador mononúcleo Tensilica L106 con una arquitectura *Reduced Instruction Set Computer* (RISC) de 32 bits de bajo consumo y una velocidad de reloj de entre 80 y 160 MHz.

Presenta una arquitectura para el ahorro de energía, permitiendo establecer el chip en modo activo, reposo y reposo profundo, lo cual es útil para que los dispositivos diseñados para alimentarse por batería funcionen durante mucho más tiempo.

En cuanto a memoria, no dispone de una memoria flash para almacenar programas, la cual debe ser proporcionada por el módulo que implemente este chip y puede tener un tamaño máximo de 16 MiB. Integra una RAM para instrucciones de 32 KiB, una caché de instrucciones 32 KiB, 80 KiB para almacenar datos del usuario y 16 KiB para datos del sistema de ETS.

Su bajo voltaje operativo oscila entre 2,5 y 3,6 V, con una corriente de operación alrededor de los 80 mA. Cuenta con la capacidad de funcionar en entornos industriales gracias a su amplio rango de temperatura de operación, que va de -40 y 125 ºC.

Admite distintos tipos de protocolos de comunicación, como IPv4, TCP, UDP y HTTP. Es un dispositivo certificado para funcionar por Wi-Fi y compatible con los protocolos 802.11 b/g/n en una frecuencia de 2,4 GHz. Tiene la capacidad de actuar como cliente en redes protegidas por claves WEP, WPA y WPA2, además de poder actuar como punto de acceso inalámbrico.

También integra en sus dimensiones compactas 16 pines GPIO para conectar dispositivos de entrada y salida, un conversor analógico de 10 bits, conmutadores de antena, un amplificador de potencia y de recepción, un balun de radiofrecuencia y módulos de gestión de potencia.

Este sistema admite varios IDEs y lenguajes de programación, como C y C++, utilizando Arduino IDE o PlatformIO; MicroPython, utilizando Mu Editor, Thonny IDE o Pymakr; y Lua, utilizando LuaLoader.

{ https://www.espressif.com/en/products/socs/esp8266 https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf https://en.wikipedia.org/wiki/ESP8266 https://www.luisllamas.es/esp8266/ https://programarfacil.com/podcast/esp8266-wifi-coste-arduino https://randomnerdtutorials.com/micropython-ides-esp32-esp8266/ https://www.danielmartingonzalez.com/es/usando-lua-en-esp8266-nodemcu-con-

lualoader-y-esplorer/ }

2.3.2. ESP32

El ESP32 es el SoC sucesor del ESP8266. Igual de apto para electrónica portátil e IoT, comparte muchas características y añade mejoras que lo convierten en un sistema muy superior.

Integra un procesador Tensilica Xtensa LX6 de doble núcleo (o de uno, dependiendo de la variante utilizada) cuya frecuencia de reloj oscila entre los 160 y 240 MHz, que trabaja en conjunto con un coprocesador de ultra baja energía.

La memoria experimenta un significativo aumento respecto a su predecesor, con un total de 520 KiB de SRAM, 448 KiB de memoria de solo lectura, 32 KiB de caché y hasta 4 MiB de memoria de almacenamiento (dependiendo del modelo).

Este chip, lanzado en septiembre de 2016, añade en comparación con el ESP8266 una mejora de potencia, soporte de Bluetooth 4.2 y BLE, sensor de temperatura, sensor hall, sensor táctil, reloj de tiempo real, más pines GPIO (hasta 34) y varios modos de energía.

Además, incorpora arranque seguro, encriptado de la flash y soporte de aceleración por hardware para los algoritmos y protocolos de cifrado y encriptación AES, SHA-2, RSA, ECC y el generador de números aleatorios.

El ESP32 tiene la posibilidad de funcionar como un sistema autónomo o como parte de un puente e interconexiones, y tiene la capacidad de interactuar con otros sistemas para proveer funcionalidad Wi-Fi y Bluetooth a través de sus interfaces.

Desde el lanzamiento del ESP32 original, han ido apareciendo variantes con distintas capacidades y procesadores, pero gran parte del código del ESP32 es compatible. Estas variantes son:

- ESP32-S2: enfocado en el consumo, integra un procesador mononúcleo LX7, reduce la memoria disponible y no tiene soporte de Bluetooth.
- ESP32-S3: utiliza el mismo procesador que el anterior, contiene más memoria y da soporte a Bluetooth 5 y BLE, enfocado al soporte de inteligencia artificial.
- ESP32-C3: contiene un procesador RISC-V mononúcleo y admite Bluetooth 5 y BLE, enfocado en la seguridad.
- ESP32-C6: centrado en la conectividad, la principal diferencia con el anterior es el soporte de Bluetooth 5.3, Wi-Fi 6 (802.11ax) y conectividad de radio (802.15.4) compatible con los protocolos Thread, Zigbee y Matter.
- ESP32-C2: incorpora un procesador RISC-V mononúcleo y admite Bluetooth 5 y BLE. Es un chip de pequeñas dimensiones que mantiene una conectividad robusta y estándares de seguridad.

- ESP32-C5: es la versión más reciente con mayor velocidad de reloj y capacidad de memoria, y es el primero que soporta Wi-Fi 6 a 5 GHz. Su enfoque en la conectividad también proviene de la capacidad de conexión con Bluetooth 5.2.

{ https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf https://www.espressif.com/en/products/socs/ https://en.wikipedia.org/wiki/ESP32 https://www.luisllamas.es/esp32/ https://www.espboards.dev/blog/esp32-soc-options/ }

2.4. ESP-NOW

El IoT se forma a partir de la conectividad entre objetos, donde surge la necesidad de un protocolo que equilibre las necesidades de latencia, uso de energía, capacidad de transmisión, confiabilidad y seguridad en la transmisión de datos. Son factores determinantes en el futuro desarrollo de esta tecnología, y aparecen como candidatos un gran número de tecnologías y protocolos, destinados tanto para la comunicación en área local, como Wi-Fi, para áreas amplias, como LoRa y LoRaWAN, y para transmisión a corta distancia, como RFID. Recientemente, al mencionado grupo se ha añadido ESP-NOW, el cual, pese a sus limitaciones, tiene características notables y es adecuado para IoT.

ESP-NOW es un protocolo de comunicación inalámbrica diseñado por Espressif para su uso entre sus dispositivos, como los ESP8266 y ESP32. Con el objetivo de sustituir a Wi-Fi y a otras tecnologías, ESP-NOW es capaz de realizar transmisiones de información y control rápidas, estables y con un bajo consumo de recursos de CPU y memoria flash entre dispositivos inteligentes, sin necesidad de un enrutador.

Se caracteriza por la rapidez de la transmisión, lograda evitando la necesidad de establecer una conexión previa entre dispositivos. Permite a su vez poner a disposición los dispositivos para transmitir datos y recibir órdenes instantáneamente tras el encendido. Además, este protocolo está basado en la capa de enlace de datos y es capaz de omitir las capas de red, transporte, sesión, presentación y aplicación del modelo OSI, reduciendo el consumo de energía (mejorando la autonomía en dispositivos con batería) y el retardo en la recepción y en el procesamiento de mensajes debido a la nula necesidad de cabeceras de paquete o desempaquetadores de cada capa. En redes congestionadas, es una característica beneficiosa, ya que brinda la capacidad de respuestas rápidas que reducen el retraso causado por la pérdida de paquetes. En el modelo utilizado en ESP-NOW en comparación con el modelo OSI estándar de la figura 2.3 se puede observar la ausencia de las 5 capas mencionadas anteriormente.

Pese al objetivo de ESP-NOW de reemplazar Wi-Fi, en los dispositivos de Espressif es capaz de coexistir simultáneamente junto a Wi-Fi y Bluetooth. Esto es útil para utilizar un dispositivo como gateway y exportar los datos intercambiados entre ESP-NOW hacia otras redes.

Tiene la capacidad de transmitir datos de manera máquina a máquina o broadcast, y para establecer la comunicación solo se requiere la dirección MAC del dispositivo de destino y establecer un canal de trans-

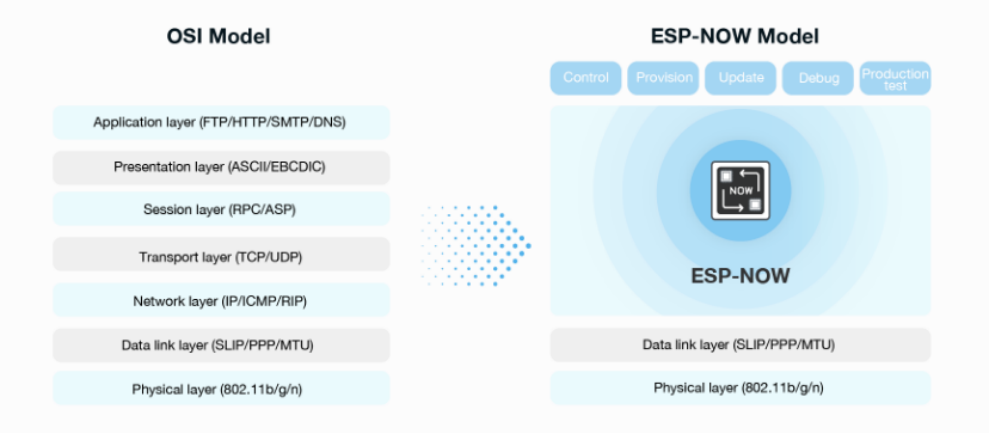

Figura 2.3: Comparación de las capas del modelo OSI con las del modelo ESP-NOW **TODO: REFERENCIAR**

misión. No obstante, dispone de una cantidad limitada de dispositivos con los que se puede emparejar. En general, el número de dispotivos emparejados no puede exceder de 20, y la cantidad de estos con los que se puede establecer una comunicación cifrada es configurable. Por defecto, este valor es 7, y admite un valor máximo de 17. Esta limitación puede ser un inconveniente en caso de necesitar una gran cantidad de dispositivos interconectados, pero una solución sería formar jerarquías de dispositivos.

Para el envío de datos, permite establecer una función de callback que será llamada instantáneamente tras el envío para poder gestionarlo. Esto puede llegar a ser útil debido a los posibles fallos que puedan ocurrir, por ejemplo, si el dispositivo de destino no existe, los canales de transmisión establecidos en ambos dispositivos no son los mismos o la trama de acción se pierde por interferencias. De la misma manera, se puede establecer una función de callback que sea llamada al recibir datos para poder tratarlos. Cabe remarcar que el protocolo no garantiza que se reciban los datos correctamente, pero existe la posibilidad de establecer ACKs para confirmar la correcta recepción y procesamiento de los datos, además de números de secuencia para afrontar la duplicidad.

ESP-NOW utiliza tramas de acción específicas del proveedor para encapsular y transmitir datos de una longitud máxima de 250 bytes, con un alcance de transmisión de entre 100 y 500 metros, dependiendo de las condiciones atmosféricas, y con una tasa de velocidad de bits de 1 megabit por segundo. Esto es beneficioso para la comunicación a larga distancia debido a su gran alcance en dispositivos al aire libre o incluso separados por paredes o pisos. Sin embargo, su uso puede estar limitado por la pequeña carga útil que puede transmitir, por lo que en otros casos podría ser mejor utilizar otras tecnologías como Wi-Fi. ESP-NOW utiliza tramas de un tamaño entre 43 y 293 bytes, cuyo formato, mostrado en la figura 2.4, está compuesto por los siguientes campos:

- Cabecera MAC, distinta de una cabecera MAC común debido a su funcionamiento sin conexión.

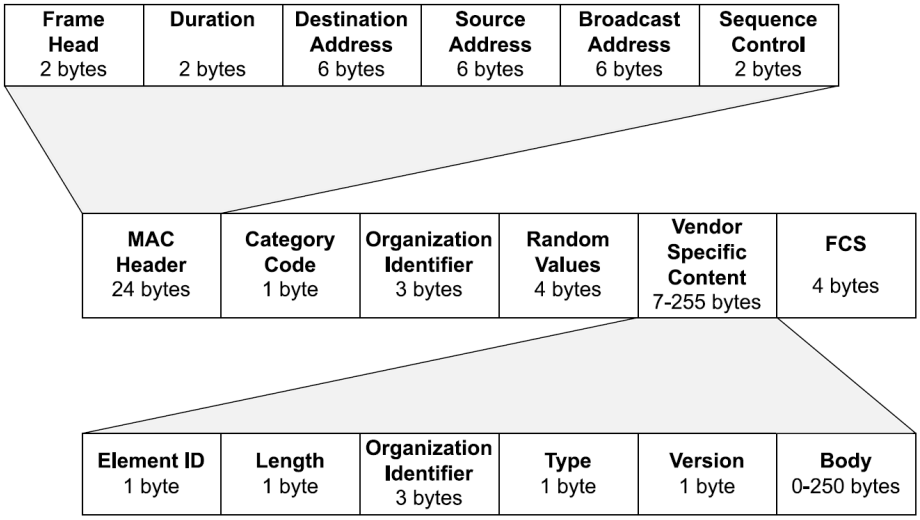

Figura 2.4: Formato de una trama ESP-NOW (Fuente: [48])

- Código de categoría, establecido a 127 para indicar la categoría específica del proveedor.
- Identificador de la organización único, que son los 3 primeros bytes de la dirección MAC aplicada por Espressif.
- Valores aleatorios, utilizados para prevenir ataques de retransmisión.
- Contenido específico del proveedor, que ocupa entre 7 y 257 bytes y contiene los siguientes campos específicos del proveedor:
  - ID del elemento, establecido a 221 para indicar que se trata del elemento específico del proveedor.
  - Longitud total del resto de campos.
  - Identificador de la organización, igual que el mencionado antes, los 3 primeros bytes de la dirección MAC.
  - Tipo, con valor 4 para indicar ESP-NOW.
  - Versión de ESP-NOW.
  - Cuerpo, que contiene los datos de ESP-NOW y puede ocupar entre 0 y 250 bytes.
- Frame Check Sequence, utilizado para verificar la integridad de la información recibida.

Existe la posibilidad de asegurar la transmisión de datos a través de ESP-NOW utilizando algoritmos de encriptación ECDH y AES y el método CBC-MAC Protocol (CCMP) que protegen las tramas de acción. El funcionamiento de estos se realiza mediante dos tipos de claves en los dispositivos: una Clave Maestra Principal (PMK) y varias Claves Maestras Locales (LMK) que corresponden a cada dispositivo emparejado. La PMK se utiliza para cifrar la LMK, y la LMK del dispositivo emparejado se utiliza para cifrar la trama de acción específica del proveedor. Esto está limitado a comunicaciónes entre pares, ya que no se admite el cifrado de tramas utilizadas para la multidifusión.

En cuanto a la gestión de dispositivos, puede utilizarse como un protocolo que ayude al aprovisionamiento de datos y configuraciones a dispositivos, depurarlos y actualizar su firmware.

ESP-NOW no necesita ningún procedimiento especial aparte de la implementación para poder utilizarse con fines comerciales. En la actualidad, se encuentra ampliamente utilizado en electrodomésticos inteligentes, iluminación inteligente, control remoto, sensores y otros.

{ (imagen, fuente): https://www.espressif.com/sites/all/themes/espressif/images/esp-now/model-en-mobile.png https://www.espressif.com/en/solutions/low-power-solutions/esp-now https://docs.espressif.com/projects/espressif-esp-faq/en/latest/application-solution/esp-now.html https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html https://emariete.com/esp8266-esp32-espnow/ }

2.4.1. Comparaciones con otras tecnologías

2.4.1.1. Wi-Fi

"Wi-Fi" es el nombre que otorga la Wi-Fi Alliance a esta tecnología de red inalámbrica basado en los estándares IEEE 802.11. Esta tecnología es ampliamente utilizado para enlazar dispositivos en redes LAN y proveer de acceso a Internet utilizando ondas de radio de 2,4 o 5 GHz (dependiendo de la versión) para transmitir la información, cumpliendo con la misión de ser una alternativa al envío de datos a través de cables.

En una red Wi-Fi se pueden encontrar una variedad de dispositivos cliente, que son los que aprovechan las características de la red, y dispositivos que distribuyen la red. Estos últimos consisten en routers o enrutadores, que brindan la conexión a Internet a los dispositivos y enrutan los mensajes enviados a través de la red; puntos de acceso, que transmiten la señal inalámbrica y es donde se conectan los dispositivos introduciendo las credenciales de la red; y repetidores, utilizados para extender el área de cobertura de una red. Entre los dispositivos, tanto clientes como los distribuidores de red, se utilizan adaptadores de red inalámbrica, que convierten los datos en una señal de radio y viceversa.

Es una de las tecnologías más utilizadas a nivel mundial, siendo que la mayoría de hogares y establecimientos disponen de una red Wi-Fi, y que los dispositivos que integran esta tecnología son fabricados con un Certificado Wi-Fi otorgado por la Wi-Fi Alliance tras superar las pruebas homologadas de interoperabilidad. Esta popularidad es beneficiosa para el IoT, ya que ofrece una capa de compatibilidad con una amplia gama de dispositivos sin necesidad de antenas, adaptadores de red ni otro tipo de hardware adicional. Además, Wi-Fi no es una tecnología nueva, tiene un sólido legado de interoperabilidad, y permite enviar información entre dispositivos con baja latencia.

Entre sus características adicionales se encuentra la topología flexible, que permite conectar los dispositivos de distintas maneras; la seguridad, ya que es posible tener redes protegidas con contraseñas cifradas mediante distintos protocolos (WEP, WPA y WPA2), el bajo coste de instalación, que en comparación con

la instalación de una red cableada, resulta más económico; y la capacidad de llegar a donde los cables no pueden llegar.

Existen distintos estándares de Wi-Fi que definen cómo actúa la red, y que cambian cada pocos años trayendo mejoras en el alcance, la velocidad y la conectividad. Por lo general, los dispositivos certificados para un estándar son intercomunicables con los certificados para otro estándar Wi-Fi siempre que compartan la misma banda de frecuencia, por lo que no es una preocupación tener todos los dispositivos con la versión más reciente. Los estándares Wi-Fi se muestran en la tabla 2.1.

Tabla 2.1: Lista de estándares Wi-Fi *TODO: REFERENCIAR*

Nombre comercial	Estándar IEEE	Año	Frecuencia (GHz)	Velocidad máxima	Rango (metros)
Wi-Fi 1	802.11a	1999	5	54 Mbps	120
Wi-Fi 2	802.11b	1999	2,4	11 Mbps	140
Wi-Fi 3	802.11g	2003	2,4	54 Mbps	140
Wi-Fi 4	802.11n	2009	2,4 y 5	450 Mbps	250
Wi-Fi 5	802.11ac	2014	5	2,3 Gbps	35
WiGig	802.11ad	2016	60	7 Gbps	100
Wi-Fi 6	802.11ax	2019	2,4, 5 y 6	9,6 Gbps	240
Wi-Fi 7	802.11be	2024	2,4, 5 y 6	46 Gbps	(Por determinar)

La popularidad de Wi-Fi abarca una gran variedad de dispositivos soportados, como teléfonos inteligentes, ordenadores, televisores inteligentes, impresoras e incluso placas de desarrollo como ESP8266 y ESP32.

{ https://en.wikipedia.org/wiki/Wi-Fi https://es.wikipedia.org/wiki/Wifi https://emariete.com/esp8266-esp32-espnow/ https://www.wi-fi.org/discover-wi-fi/internet-things https://www.adslzone.net/reportajes/tecnologia/que-es-wifi-como-funciona/ https://www.proofpoint.com/es/threat-reference/wifi#:~:text=Wifi %2C %20que %20es %20una %20contra

```
Fuentes de tabla
https://www.intel.la/content/www/xl/es/support/articles/000005725/wireless/
    ↪ legacy-intel-wireless-products.html
https://www.monolitic.com/cual-es-la-mejor-tecnologia-wifi-para-desarrollos-iot/
https://www.makeuseof.com/tag/understanding-common-wifi-standards-technology-
    ↪ explained/
https://www.netspotapp.com/es/blog/wifi-standards/
https://www.xataka.com/nuevo/nuevo-wifi-7-informacion
```

```
https://www.geckoandfly.com/10041/wireless-wifi-802-11-abgn-router-range-and-
    ↪ distance-comparison/
https://www.business.com/articles/what-is-802-11-ax-wi-fi/
```

}

2.4.1.2. Bluetooth

Bluetooth es un estándar de tecnología que facilita el intercambio de datos entre dispositivos a través de distancias cortas, con un máximo de 10 metros.

Desde su introducción en 1998, ha pasado por múltiples revisiones, siendo las más relevantes las de la última década:

- Versión 4.0 hasta 4.2: aumentó la velocidad de transferencia de datos a 24 Mbps y el alcance hasta 100 metros, y añadió soporte al protocolo IPv6. Además, introdujo BLE o Bluetooth Low Energy, una nueva variante de esta tecnología.
- Versión 5.0 hasta 5.2: aumentó la velocidad de transferencia de datos a 50 Mbps y el alcance hasta 200 metros, y realizó mejoras en la transmisión de audio y en el consumo de energía.

Bluetooth Low Energy fue diseñado para operaciones de bajo consumo de energía, capaz de soportar diferentes tipologías de comunicación (punto a punto, difusión y malla), a diferencia del clásico que solo admite punto a punto. Mientras que el Bluetooth clásico se usa para transferir datos y sonido, BLE es capaz de utilizarse para analizar con alta precisión la presencia, distancia y dirección del dispositivo.

Bluetooth utiliza ondas de radio UHF en las bandas ISM sin licencia de 2,4 GHz, y se utiliza como alternativa a las conexiones por cable para intercambiar ficheros y conectar transmisores de audio. Gracias a su bajo consumo de energía, seguridad, capacidad contra interferecias, compatibilidad con varios sistemas operativos y facilidad de implementación, esta tecnología se convierte en una buena opción para la implementación del IoT. Además, tiene la capacidad de agregar capas de cifrado, autenticación y verificación, y de construir redes PAN entre dispositivos al interconectar varios entre sí. Es común encontrarlo en pulseras y relojes inteligentes, teléfonos inteligentes, ordenadores, reproductores de música, altavoces, auriculares, y placas de desarrollo como ESP32.

{ https://www.bluetooth.com/learn-about-bluetooth/tech-overview/ https://en.wikipedia.org/wiki/Bluetooth https://www.xatakahome.com/curiosidades/bluetooth-su-evolucion-estas-diferencias-distintas-versiones https://www.mokosmart.com/es/what-is-bluetooth-iot-and-why-choose-it/ }

2.4.1.3. Más tecnologías y protocolos IoT

En esta sección se hacen menciones honoríficas a otras tecnologías y protocolos relevantes al IoT, sin entrar en un análisis detallado.

Zigbee: es un protocolo basado en la especificación IEEE 802.15.4 que permite formar redes PAN sencillas. Estas redes son en malla y están formadas por un nodo coordinador. Zigbee opera en la banda de 2,4 GHz y se utiliza en proyectos a pequeña escala que requieren una conexión inalámbrica de baja potencia, baja velocidad de transmisión (hasta 250 Kbps) y un rango próximo (hasta 10-100 metros) [52].

Thread: es una tecnología que utiliza 6LoWPAN y la especificación IEEE 802.15.4, por lo que opera en la banda de 2,4 GHz con una velocidad de hasta 250 Kbps. Permite crear redes malladas de bajo consumo que comunica de manera encriptada más de 250 dispositivos, los cuales deben ser certificados y creados por empresas miembros del Thread Group [51] [22].

LoRa: es una tecnología inalámbrica de conexión punto a punto que emplea tecnología de modulación en radiofrecuencia. Es ideal cuando se necesitan conexiones que cubran largas distancias con dispositivos sin fácil acceso a la corriente eléctrica de red ni cobertura, ya que ofrece un bajo consumo de uso y un largo alcance de 10-20 km, con alta tolerancia a interferencias. Opera en bandas inferiores a 1 GHz (dependiendo de la región), y transmite un máximo de 255 bytes a velocidades entre 0,3 Kbps y 50 Kbps. Esta tecnología es conocida por ser utilizada en LoRaWAN [11] [41].

2.4.1.4. Comparaciones entre Wi-Fi, Bluetooth y ESP-NOW

Es de alta relevancia comparar estas tecnologías y ESP-NOW entre sí, ya que son los más populares en el ámbito del IoT y son compatibles con las placas ESP32 que incorporan el reciente e innovador ESP-NOW. En particular, ESP32 es compatible con Bluetooth 4.2 y con Wi-Fi b, g y n de 2,4 GHz, por lo que a lo largo de este apartado se comparan estas versiones. Estos tres protocolos son similares en varios aspectos, ya que utilizan ondas de radio para transmitir datos de forma inalámbrica a una amplia gama de dispositivos, de manera rápida y fiable. Esto puede resultar en una decisión compleja para elegir entre los tres, aunque hay escenarios en los que no es necesario elegir, ya que en un ESP32 pueden trabajar en conjunto.

De manera más resumida, estas son las características teóricas que ofrecen los protocolos:

1. *ESP-NOW*:
   - **Alcance**: 220 metros.
   - **Cantidad de dispositivos conectables**: 20 a cada nodo.
   - **Unidad de Transmisión Máxima (MTU)**: 250 bytes.
   - **Velocidad de transmisión**: 1 Mbps.
   - **Uso**: IoT y comunicación entre dispositivos de Espressif.
2. *Wi-Fi b/g/n (2,4 GHz)*:

- **Alcance**: 250 metros.
- **Cantidad de dispositivos conectables**: depende de la configuración de la red y la asignación de direcciones IP.
- **Unidad de Transmisión Máxima (MTU)**: 1460 bytes, configurado en la librería de red de ESP32.
- **Velocidad de transmisión**: 54 Mbps.
- **Uso**: conexión a internet, acceso a dispositivos e IoT.

3. *Bluetooth 4.2*:
   - **Alcance**: 50 metros.
   - **Cantidad de dispositivos conectables**: 7 a cada nodo.
   - **Unidad de Transmisión Máxima (MTU)**: 251 bytes.
   - **Velocidad de transmisión**: 1 Mbps.
   - **Uso**: audio, dispositivos personales e IoT.

/*TODO: referenciar*/

Para detallar las comparaciones aplicando los protocolos en escenarios de uso real, se han tomado los datos de distintas pruebas realizadas y detalladas en una publicación de 2021 llevada a cabo por Dania Eridani, Adian Fatchur Rochim y Faiz Noerdiyan Cesara, miembros del Departamento de Ingeniería Informática de la Universidad de Diponegoro[5], en Indonesia.

En esta publicación [26] se realizaron pruebas de rango, velocidad, latencia, consumo y resistencia a barreras. Para ello, se utilizaron una placa ESP32 Development Board, una ESP32-CAM y una ESP32U, además de una antena externa de 2.4GHz conectada a la última placa mencionada y utilizada en ciertas pruebas.

Tabla 2.2: Resultados de las pruebas comparativas realizadas entre los protocolos (Fuente: [26])

Prueba realizada	ESP-NOW	Wi-Fi	Bluetooth
Rango máximo (metros), más valor es mejor:			
· con antena interna	**185**	84	15
· con antena externa	**220**	88	25
Velocidad de transmisión, más valor es mejor:			
· de 200 bytes (Kbps)	588	**2048**	938
· del MTU (Mbps)	0,63	**24,86**	0,98
Latencia (µs), menos valor es mejor:			

[5] https://tekkom.ft.undip.ac.id/language/en/home/

Prueba realizada	ESP-NOW	Wi-Fi	Bluetooth
· con 100 bytes de datos	**1869**	3435	8514
· con 50 bytes de datos	**1435**	3388	6460
· con 10 bytes de datos	**1133**	3367	6200
· con 1 byte de datos	**1059**	3359	6048
Consumo (mW), menos valor es mejor:			
· solo conectado, como receptor	489	214	**141**
· solo conectado, como emisor	449	465	**212**
· transfiriendo datos, como receptor	511	477	**338**
· transfiriendo datos, como emisor	1042	538	**441**

A partir de los resultados de estas pruebas, mostrados en la Tabla 2.5, se puede ver que:

- El rango máximo aumenta con el uso de una antena externa en un 18,9 %, 4,7 % y 66,6 % en los protocolos ESP-NOW, Wi-Fi y Bluetooth, respectivamente. Además, ESP-NOW es el protocolo que soporta una mayor distancia entre dos dispositivos interconectados.
- Wi-Fi es la tecnología con mayor capacidad para transmitir datos rápidamente, tanto en la velocidad como en el tamaño de los datos.
- ESP-NOW tiene la menor latencia, 1ms para 1 byte, tres veces menos que Wi-Fi y seis veces menos que Bluetooth, siendo la mejor opción en cuanto a esta propiedad de la conexión. Debido a la baja compatibilidad de dispositivos que admiten ESP-NOW, una buena alternativa sería utilizar Wi-Fi para la transmisión de datos con poco retardo.
- En el consumo, Wi-Fi y ESP-NOW tienen un valor similar al estar conectados como transmisor, mientras que Bluetooth consume menos tanto como transmisor como receptor. En el caso de transmitir datos, ESP-NOW es el que más consume ya que necesita activar internamente Wi-Fi para funcionar. La prueba demuestra que, en el caso de tener limitada la vida útil de un dispositivo con batería, la mejor solución puede ser Bluetooth, aunque hay diversas maneras de mejorar el código que se utilice en la placa ESP32 para mejorar la eficiencia energética, como añadir paradas y suspensión.

Finalmente, tras la prueba de resistencia a barreras, se demostró que las barreras de madera y cristal no afectan gravemente a la señal en comparación con la transmisión al aire libre, que el metal afecta de manera más notoria a decenas de metros de distancia del receptor, y que el Bluetooth sin una antena externa no es capaz de atravesar muros a 10 metros o más de distancia.

Las conclusiones a las que se llega en esta publicación se pueden representar en la figura 2.5, que muestra que:

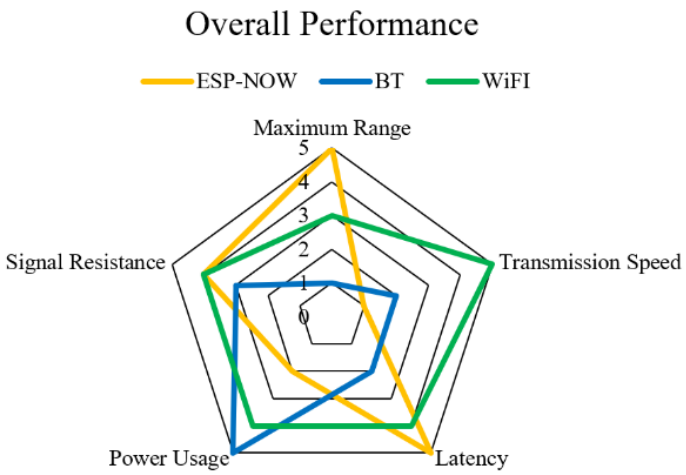

Figura 2.5: Comparación del rendimiento entre protocolos en distintos aspectos (Fuente: [26])

- La mayor ventaja de Bluetooth se encuentra en el consumo, ya que su rendimiento en el resto de los aspectos es muy deficiente.
- ESP-NOW es el mejor candidato cuando se requieren rangos elevados, una señal de comunicación resistente y mínima latencia en velocidades de datos muy pequeñas, pero su uso consume mucha energía relativamente.
- Wi-Fi es una tecnología muy equilibrada, y destaca por su velocidad.

{ https://es.wikipedia.org/wiki/Bluetooth_(especificaci %C3 %B3n) https://docs.espressif.com/projects/esp-faq/en/latest/application-solution/esp-now.html https://www.amarinfotech.com/differences-comparisons-bluetooth-5-vs-4-2.html https://www.symmetryelectronics.com/blog/bluetooth-5-versus-bluetooth-4-2-what-s-the-difference/ https://docs.arduino.cc/tutorials/nano-esp32/esp-now/ https://github.com/espressif/arduino-esp32/blob/master/libraries/Network/src/NetworkUdp.cpp https://www.electronicdesign.com/technologies/communications/artic v42-creating-faster-more-secure-power-efficient-designspart-1

```
(documento)https://ieeexplore.ieee.org/document/9573246
}
```

2.5. Ejemplo de modelo tradicional publicador-broker-suscriptor con MQTT y ESP32

Tras introducir una serie de conceptos sobre el IoT, en este apartado se detallará el ejemplo visto en el apartado de Internet de las Cosas, en la figura 2.1, que trata de un sistema de riego por aspersión.

La primera parte consiste en identificar los dispositivos a utilizar. Los sensores y actuadores deben estar conectados a un dispositivo capaz de interactuar con ellos, como puede ser una placa ESP32.

El broker utilizará una comunicación basada en colas para recibir todos los datos, integrando así una implementación del protocolo de mensajería MQTT, como puede ser Mosquitto, que únicamente se puede llevar a cabo si el broker es un ordenador, como un ordenador portátil o un single board computer. Para este ejemplo, se puede suponer que el broker será una placa Raspberry Pi.

Teniendo las placas ESP32 y la Raspberry Pi con Mosquitto, es necesario establecer la comunicación entre ambos. Una de las opciones a evaluar que sea compatible con ambas placas es Bluetooth, pero no es adecuado para una comunicación a larga distancia, ni para transmitir datos en tiempo casi real debido a su su baja velocidad, y tampoco es compatible con Mosquitto. Por lo tanto, se debe utilizar Wi-Fi para que las placas ESP32 publiquen los datos que generen y se suscriban a las órdenes y configuraciones mediante las colas adecuadas de Mosquitto. En este caso, se deberá desplegar una serie de puntos de acceso y routers para formar una red LAN o MAN que dote de Internet a todos los dispositivos.

Finalmente, se configuran las placas ESP32 para que, con la ayuda de alguna librería de código, puedan interactuar con las colas de Mosquitto, y también el broker para poder interactuar con el servidor.

La implementación de este sistema de riego presenta una serie de inconvenientes. En cuanto a inversión de dinero y tiempo, implica el despliegue de numerosos puntos de acceso y routers, así como su configuración y mantenimiento. Además, se debe asegurar una buena señal al aire libre, evaluar la zona donde se instalarán y proveer más baterías en los dispositivos que formen esta red.

Todos los dispositivos están conectados en la misma red, por lo que un tráfico alto de datos o una interrupción en el servicio puede provocar una congestión de la red y un mal funcionamiento de la misma.

Finalmente, en cuanto a seguridad, a todos los dispositivos de esta red se les otorga acceso a Internet, por lo que deben estar preparados para no sufrir un ciberataque que pueda invalidar el sistema por completo y todos los dispositivos conectados. Esto supone una mayor complejidad en el despliegue y mayor mantenimiento para evitar un riesgo significativo en la red.

Aunque la implementación de este sistema de riego se puede realizar de distintas maneras, en este ejemplo se intenta demostrar la complejidad que supone un despliegue limitado por utilizar MQTT y ESP32 en la actualidad. Pese a que, de manera independiente, ambos tienen grandes ventajas, su combinación supone una complejidad difícil de evitar.

/*TODO: añadir un nuevo esquema*/

{ https://www.prometec.net/esp32-mqtt/ (libreria de codigo) }

Capítulo 3

Herramientas y Metodología

3.1. Herramientas

En este apartado se identifican y describen las distintas herramientas hardware y software que han permitido llevar a cabo este TFG.

3.1.1. Hardware

El núcleo de este TFG se basa en la programación de una placa de desarrollo, por lo que es necesario ciertos elementos hardware para llevarlo a cabo:

Ordenador portátil: es el componente principal para el desarrollo de este TFG, en el cual se han instalado las herramientas necesarias, se ha escrito el código, se ha accedido a herramientas y recursos en línea, y ha permitido subir el código a la placa. En este caso, el ordenador personal del alumno es un Lenovo Ideapad 3 15ALC6, cuyas características destacadas para este trabajo son:

- Procesador AMD Ryzen 7 5700 U, de 64 bits y lanzado en enero de 2021, que permite ejecutar aplicaciones y servicios aprovechando sus 8 núcleos, su alta frecuencia de 1,8 GHz hasta 4,3 GHz y su bajo consumo [28].
- 12 GB de RAM DDR4, que almacenan la información temporal generada por los programas en ejecución. Su capacidad determina cuántas tareas simultáneas se pueden ejecutar y su velocidad determina la rapidez de ejecución de estas.
- Almacenamiento interno SSD de 1 TB, encargado de almacenar de manera persistente información como el sistema operativo, las herramientas instaladas y los ficheros de código utilizados en el proyecto.
- 3 puertos USB, que permiten conectar distintos dispositivos simultáneamente al ordenador, como en este trabajo, las placas para subir el código y observar la salida por terminal durante la ejecución.

- Lector de tarjetas SD, utilizado para acceder al contenido de la tarjeta SD que se conecta a una de las placas (detallado posteriormente).
- Adaptador de red Bluetooth y Wi-Fi, para poder conectar el ordenador a Internet y acceder a los recursos necesarios para el desarrollo.

Dos placas ESP32 DEVKIT V1: ambas fueron cedidas por la UCLM para el desarrollo de este TFG. este tipo de placas contienen las mismas características que el SoC ESP32 mencionado en el apartado **??** /*TODO: comprobar funcionamiento referencia*/. Concretando, utiliza el módulo ESP32-WROOM-32. Las placas están distribuidas en una placa de pruebas o breadboard, permitiendo conectar distintos elementos a los pines de las placas mediante jump wires o cables puente, sin necesidad de soldadura ni diseñar circuitos integrados y facilitando la prueba de componentes. En este caso, tienen conectados distintos módulos y sensores que permiten ampliar las funciones de estas:

- **Módulo lector de tarjeta microSD**: compuesto por un socket para insertar tarjetas microSD en un circuito impreso del cuál salen 7 pines para poder utilizar el bus SPI de las tarjetas microSD. El SPI o Serial Peripheral Interface es un estándar que se utiliza para transferir información entre circuitos integrados, como pueden ser la placa ESP32 DEVKIT y el lector de tarjetas. Los pines que tiene este módulo son los siguientes:
  - VCC: Entrada de energía, se conecta a una fuente de alimentación para alimentar el lector.
  - GND: Conexión a tierra, se conecta al terminal negativo de la fuente de alimentación.
  - Data In: También conocido como MOSI o Master Output Slave Input, se utiliza para enviar datos desde la placa ESP32 hasta la tarjeta microSD.
  - Data Out: También conocido como MISO o Master Input Slave Output, se utiliza para enviar datos desde la tarjeta microSD hasta la placa ESP32.
  - Serial Clock: Reloj SPI, se utiliza para sincronizar la transferencia de datos entre las placas.
  - Chip Select: Se utiliza para activar y desactivar la comunicación con el lector.
  - Card Detect: Se utiliza para detectar si hay una tarjeta insertada en el lector.

  Para este trabajo, el lector, cedido por la UCLM, se utiliza para leer y escribir datos en una tarjeta microSD desde la ESP32 DEVKIT V1 que actúe como broker y gestione los registros o logs y las direcciones de los dispositivos suscritos contenidos.
- **Sensor DHT11**: sensor digital capaz de medir la temperatura y la humedad, cedido por la UCLM. Alimentado por 3,3 o 5 voltios, es capaz de leer la humedad en el ambiente entre los rangos 20 y 95 % (con un 5 % de fallo) y la temperatura entre 0 y 50 ºC (con 2 ºC de fallo).
- **Potenciómetro BQ Zum Kit Advanced**: contenido originalmente en un kit junto a varios sensores y una placa controladora [17] [18], este potenciómetro de señal analógica cedido por la UCLM es capaz de devolver un valor en función a su rotación, siendo su rotación máxima 300º, al alimentarlo con 3,3 o 5 voltios [19].

Placa ESP32-2432S028R: esta placa proporcionada por el autor y popularmente conocida como Cheap-Yellow-Display o CYD para abreviar, se ha utilizado para hacer pruebas del funcionamiento de la herramienta desarrollada, y al igual que las anteriormente mencionadas placas ESP32, está potenciado por un ESP32-WROOM-32. En cuanto a componentes integra:

- Pantalla LCD de 2,8 pulgadas con resolución 320x240 píxeles, y táctil de tipo resistivo.
- LED multicolor RGB.
- Lector de tarjeta microSD.
- Sensor LDR, cuyo valor que devuelve depende de la resistencia que le otorga la luz que recibe.
- Conectores y pines adicionales para conectar un altavoz y otros módulos.

Tarjeta microSD: tarjeta proporcionada por el alumno autor y utilizada por una de las placas con lector de tarjetas para almacenar los registros o logs y las direcciones de los dispositivos suscritos al broker. Las características de la tarjeta no son deterministas para la tarjeta, ya que las especificaciones no afectan al rendimiento de la herramienta resultante de este TFG. En este caso, la tarjeta microSD utilizada es de la marca Samsung y cuenta con capacidad de almacenamiento de 1 GB.

Adaptador de tarjeta microSD a SD: proporcionada por el autor, permite utilizar una tarjeta microSD en un lector de tarjetas SD normal al adaptar su tamaño y forma. Se ha utilizado para leer, modificar y eliminar contenidos de la tarjeta microSD desde el ordenador portátil, útil para hacer probar el funcionamiento de la herramienta.

{ https://www.sparkfun.com/products/544 https://es.wikipedia.org/wiki/Serial_Peripheral_Interface https://tienda.bricogeek.com/sensores-temperatura/1574-modulo-sensor-dht11-humedad-y-temperatura.html https://github.com/witnessmenow/ESP32-Cheap-Yellow-Display/ }

3.1.2. Software

Para poder realizar el desarrollo correcto de este TFG, se han requerido las siguientes piezas de software:

Windows 10[1]: el sistema operativo es clave para poder hacer funcionar el Trabajo de Fin de Grado. En este caso, el autor ha utilizado la versión más reciente a fecha de la escritura de este documento, 22H2. Este sistema operativo desarrollado por Microsoft se encuentra instalado en el ordenador portátil mencionado en el apartado de hardware /*TODO: mencionar*/, y debido a su alta popularidad tiene una gran compatibilidad con la mayoría de aplicaciones existentes, y en este caso es beneficioso para el resto de software mencionado posteriormente. Además, integra:

- Windows Update y Windows Defender: Permite tener el sistema en la última versión y seguro gracias a los últimos parches de seguridad, una capa de protección importante para los ficheros, la información contenida y la integridad del propio sistema al estar conectado a Internet e instalar aplicaciones.

[1] https://www.microsoft.com/es-es/software-download/windows10

- Explorador de archivos: Como su nombre lo indica, permite navegar entre carpetas y acceder a los archivos, tanto de los discos duros instalados internamente como los externos (por ejemplo, pendrives, tarjetas de memoria o discos).
- Bloc de notas: Esta herramienta es ideal para crear y editar ficheros de texto plano o, en el caso de este desarrollo, ver ficheros de código de manera rápida.
- Drivers: Ofrecen compatibilidad y rendimiento con todo el hardware contenido en el ordenador, como procesador, tarjeta gráfica, teclado y ratón, puertos USB, lectores de tarjetas y otros dispositivos externos. En este caso, para poder hacer uso de las placas, se ha necesitado instalar un driver adicional denominado *CH341SER*[2].

La relevancia de usar Windows 10 es mínima, ya que es posible de adaptar el proyecto a un entorno en Linux con pocas o nulas complicaciones. El uso de Windows 10 ha sido decisión del autor por gusto.

GitHub[3]: es una plataforma en la nube propiedad de Microsoft que permite a los equipos de desarrolladores almacenar código, colaborar y realizar cambios en proyectos compartidos, alojados en esta plataforma en forma de repositorios de Git. Git es un sistema de control de versiones de código abierto creado en 2005 por Linus Torvalds, que permite a cualquier desarrollador gestionar el código fuente y el historial de cambios mediante comandos ejecutados en una terminal. Tiene la característica de ser distribuido, permitiendo usar ramas para aislar partes del código, como pueden ser nuevas funcionalidades en desarrollo, sin afectar al código final desplegado. Cada parte del equipo puede crear una rama, integrar los cambios necesarios y luego fusionarla con la rama principal para hacer efectivos estos cambios. Esto es útil para añadir varias funcionalidades simultáneamente, permitir que varias personas trabajen en los mismos archivos, comprobar las diferencias entre estos, aprobarlas y volver a una versión funcional anterior en caso de que errores. GitHub aprovecha estas funcionalidades con sus repositorios y las aplica a su interfaz web, haciendo su uso sea más accesible para usuarios con poco conocimiento técnico al evitar recordar comandos específicos.

Desde un repositorio en GitHub se pueden crear y modificar ramas, cargar ficheros, realizar *commits* (la unidad de trabajo de GitHub que representa un cambio en el repositorio), ver su histórico para hacer un seguimiento de los cambios realizados, obtener las modificaciones realizadas por otros usuarios y hacer *pull requests* (solicitudes de cambios) para integrar estos cambios en el proyecto. GitHub va un paso más allá de alojar proyectos, ya que permite la gestión de proyectos y la interacción del equipo de desarrollo a través de *issues* que retroalimentan el proyecto y ofrecen ideas, asignación de responsabilidades, hitos, etiquetas, discusiones, y gráficos y tableros estilo Kanban que permiten observar fácilmente el trabajo realizado y por hacer.

Los repositorios pueden ser públicos o privados, siendo en este último caso que el repositorio solo pueden ser vistos por los usuarios agregados y cuya funcionalidad está limitada. La plataforma permite a los usua-

[2] https://www.wch-ic.com/downloads/CH341SER_ZIP.html
[3] https://github.com/

rios interactuar con repositorios de otros usuarios, incluidos los públicos (GitHub es comúnmente utilizado para alojar proyectos de código abierto), contribuyendo o realizando una bifurcación para adaptarlo como un proyecto distinto. Otras funciones que incluye GitHub son:

- Documentación de proyectos, mediante la creación de ficheros de texto "readme" o "léeme" en lenguaje Markdown en los directorios del proyecto.
- Creación de wikis.
- Automatización de pruebas, lanzamientos y despliegues con GitHub Actions, especificando los pasos a ejecutar tras una acción específica realizada en el repositorio.
- GitHub Codespaces, un IDE online.
- Alojamiento de páginas web estáticas con GitHub Pages como parte de un repositorio, como blogs, documentación de código y libros.
- Gists o fragmentos de código compartibles o utilizables con soporte de control de versiones.

GitHub además soporta planes de pago para aumentar la funcionalidad que ofrece, proporcionar soporte personalizado a los usuarios y quitar limitaciones de repositorios privados.

En el caso de este proyecto, se ha utilizado, con el plan gratuito de GitHub, un repositorio privado durante el desarrollo para alojar el código, compartirlo fácilmente con los tutores para enviar dudas e informar del estado del proyecto, y llevar un histórico de los cambios realizados. Posteriormente, se ha modificado la visibilidad del repositorio a público para compartir el proyecto a través de la plataforma de librerías de PlatformIO (mencionado en los siguientes puntos) y aprovechar la funcionalidad de GitHub Pages para incluir una página web estática que documente las distintas funciones que componen el proyecto, ambas funciones automatizadas a través de GitHub Actions /*TODO: realizar*/. Pese a conocer alternativas a la plataforma, como GitLab, se ha decidido utilizar GitHub por la experiencia previa del alumno y la facilidad de uso que ofrece.

Otra herramienta que ofrece GitHub para no separar la funcionalidad del escritorio local es **GitHub Desktop**[4], que permite simplificar el flujo de trabajo del desarrollador y centrarse en su trabajo. Proporciona una interfaz gráfica que evita interactuar directamente con Git para clonar proyectos, hacer *commits*, cambiar de rama y ver los cambios y diferencias en los archivos.

{ https://docs.github.com/es/get-started/start-your-journey/about-github-and-git https://www.hostinger.es/tutoriales/que-es-github https://en.wikipedia.org/wiki/GitHub https://docs.github.com/en/pull-requests/committing-changes-to-your-project/creating-and-editing-commits/about-commits https://docs.github.com/es/issues/tracking-your-work-with-issues/about-issues https://desktop.github.com/ }

Visual Studio Code[5]: es un IDE ligero, potente, multiplataforma, personalizable y de código abierto creado

[4] https://desktop.github.com/
[5] https://code.visualstudio.com/

por Microsoft. El editor no se basa en un sistema de proyectos, sino que permite al usuario abrir uno o varios directorios simultáneamente, formando workspaces o espacios de trabajos compuestos por ficheros que pueden contener distintos lenguajes de programación. Por defecto, soporta los lenguajes JavaScript, TypeScript y Node.js, pero esta lista se puede ampliar mediante extensiones para utilizarlo con C++, Java, Python y otros lenguajes. Estas extensiones, creadas por Microsoft o por terceros, están disponibles en un repositorio central, y además permiten personalizar y extender las funcionalidades del editor, adaptándose a las necesidades del usuario.

Visual Studio Code integra una serie de funcionalidades que lo convierten en un editor ideal para incrementar la productividad del usuario y perfecto para su uso diario, tales como el subrayado de sintaxis y errores, sangría automática, refactorización, autocompletado, sugerencias y fragmentos de código. Ofrece la posibilidad de compilar y ejecutar fácilmente los ficheros de código y proyectos simplemente haciendo clic en el botón de "play", evitando la necesidad de aprender comandos o repetirlos constantemente y agilizando el trabajo del desarrollador. En aquellas ejecuciones de código que pueden resultar en error o son difíciles de comprender ayuda a realizar comprobaciones mediante un depurador interactivo, que permite recorrer el código fuente con puntos de interrupción, inspeccionar variables, ver pilas de llamadas y modificar líneas de código en ejecución. Además, es compatible con repositorios Git, ofreciendo al usuario la posibilidad de ver las diferencias y cambios respecto a la versión localizada en el repositorio, sin necesidad de salir de la aplicación ni usar comandos complejos en la terminal.

Este ha sido el IDE de preferencia para el desarrollo de este trabajo, y se ha complementado con las siguientes extensiones:

- PlatformIO IDE[6]: habilita el uso de Visual Studio Code como IDE para el desarrollo de software embebido para distintas plataformas y frameworks utilizando el sistema de PlatformIO (explicado posteriormente) [39].
- C/C++ Extension Pack[7]: un conjunto de tres extensiones que permiten utilizar Visual Studio Code para el desarrollo de proyectos en C/C++, incluyendo características como resaltado de sintaxis, completado de código, depuración y comprobación de errores [36] [3].
- GitHub Copilot[8]: una herramienta de programación basada en inteligencia artificial que ayuda al desarrollador a escribir código de forma rápida e inteligente [29]. Esta herramienta se puede utilizar para recibir ayuda en cualquier librería, lenguaje y framework popular, ya que el entrenamiento de esta IA se ha realizado a partir de los repositorios públicos alojados en GitHub [29]. GitHub Copilot ofrece sugerencias de código inteligentes mientras se escribe, basándose en el contexto y en los comentarios del propio código. Asimismo, proporciona un chat para realizar consultas sobre cualquier tarea del código, como pedir explicaciones, informarse acerca de conceptos de programación y guiar

[6] https://marketplace.visualstudio.com/items?itemName=platformio.platformio-ide
[7] https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools-extension-pack
[8] https://marketplace.visualstudio.com/items?itemName=GitHub.copilot

al usuario para mejorar su código o implementar nuevas funcionalidades [29] [5]. El uso de esta herramienta está limitado a los usuarios de pago o a quienes disponen de un GitHub Student Developer Pack tras asociar la cuenta de GitHub con el correo de la institución académica. También dispone de una versión de prueba [2].

- Better Comments[9]: añade diferencias de estilo y enfatizaciones a comentarios en el código destinados a alertar, hacer peticiones, indicar acciones por realizar o TODOs, o remarcar información importante, logrando así comentarios más comprensibles [16].
- TODO Highlight[10] y Todo Tree[11]: ambas extensiones se utilizan en conjunto para llevar un seguimiento de las tareas que se deben realizar en el código, evitando que el desarrollador las olvide. La primera extensión resalta las líneas que contienen el texto "TODO" y "FIXME" [33], útil para llamar la atención del usuario que recorre las líneas del código, mientras que la segunda muestra en forma de árbol todos los "TODO" y "FIXME" que se encuentren en el código [30], para poder agruparlos en un mismo lugar y acceder a ellos fácilmente.
- Code Spell Checker[12]: un corrector ortográfico que marca los errores ortográficos detectados en los comentarios y en el código, y que ayuda a solucionarlos a través de sugerencias [46].
- Doxygen Documentation Generator[13]: permite generar automáticamente en formato Doxygen los comentarios junto a parámetros como las descripciones, los parámetros y los valores retornados para su uso en la documentación. Además, ofrece soporte para autocompletado y sugerencias de comandos Doxygen [44].

{ https://code.visualstudio.com/docs/editor/whyvscode https://en.wikipedia.org/wiki/Visual_Studio_Code }

PlatformIO[14]: es una herramienta de código abierto para ingenieros y desarrolladores de software de sistemas embebidos. Permite desarrollar software desde cualquiera de los sistemas operativos modernos de manera sencilla para todo tipo de usuarios, tanto aficionados como profesionales, incluyendo las herramientas necesarias para compilar, ejecutar, subir y escribir el código. Ofrece una amplia compatibilidad, con soporte para más de mil placas embebidas diferentes, más de 40 plataformas de desarrollo y más de 20 marcos de trabajo o frameworks.

PlatformIO aloja uno de los mayores registros de librerías embebidas en PlatformIO Registry, lo que permite explorar e instalar de manera sencilla distintas bibliotecas de código, plataformas y herramientas, listadas junto a ejemplos e instrucciones de uso. Este registro se puede utilizar desde la interfaz gráfica de

[9] https://marketplace.visualstudio.com/items?itemName=aaron-bond.better-comments
[10] https://marketplace.visualstudio.com/items?itemName=wayou.vscode-todo-highlight
[11] https://marketplace.visualstudio.com/items?itemName=Gruntfuggly.todo-tree
[12] https://marketplace.visualstudio.com/items?itemName=streetsidesoftware.code-spell-checker
[13] https://marketplace.visualstudio.com/items?itemName=cschlosser.doxdocgen
[14] https://platformio.org/

la herramienta, por línea de comandos y desde su página web[15].

Una de sus características más importantes es la gestión de dependencias integrada. Es común que los proyectos aprovechen funcionalidades ofrecidas en bibliotecas, por lo que el usuario debe referenciar la biblioteca y PlatformIO se encarga de resolver las dependencias al compilar el código. Estas bibliotecas soportadas pueden estar en local (como carpetas o ficheros comprimidos), en un repositorio con control de versiones (como Git) o en el PlatformIO Registry.

Además, contiene un depurador de código, un analizador estático de código, un monitor de puerto serial y soporte para pruebas unitarias.

La manera de utilizar PlatformIO en un proyecto es sencilla, solo es necesario instalar el IDE, indicar la placa y el framework de interés, y PlatformIO se encarga de la descarga e instalación de las herramientas necesarias de forma automática. Además, ofrece cierta flexibilidad y opciones a los desarrolladores, que pueden decidir si usar la herramienta por línea de comandos o con la versión gráfica.

Esta herramienta se integra con otros IDEs o editores de texto a través de extensiones, siendo Visual Studio Code el más recomendado.

El código de este proyecto se ha creado con PlatformIO y se ha configurado para utilizar la placa ESP32 DEVKIT V1 con la plataforma Espressif32. En cuanto al framework, se ha utilizado el de Arduino, pese a ser posible también utilizar el de Espressif (ESP-IDF), debido a que cumple las necesidades del desarrollador, a la experiencia previa con placas de desarrollo Arduino por parte del autor y a la facilidad de uso. Las diferencias entre ambos son las siguientes:

- El framework ESP-IDF ofrece un soporte completo de los lenguajes C y C++, permitiendo escribir código eficiente y de alto rendimiento. Por otro lado, Arduino utiliza una implementación simplificada y adaptada a los microcontroladores, limitando la flexibilidad y funcionalidad del código.
- Las aplicaciones desarrolladas con ESP-IDF están preparadas para hacer uso de los núcleos disponibles en la placa y su estructura se basa en tareas, mientras que en Arduino por defecto solo se aprovecha de un núcleo y las aplicaciones siguen una estructura en la que se debe declarar una función `setup` y otra `loop`.
- El framework de Arduino es útil si previamente el desarrollador lo ha utilizado para desarrollar en otras placas, además de ser fácil de usar para quienes no tienen mucha experiencia, y contiene un gran rango de librerías y APIs por defecto que facilitan el desarrollo. En cambio, ESP-IDF se puede utilizar para desarrollar software que requiera controlar funciones avanzadas del hardware, como el consumo de energía y recursos, e incluye un mayor conjunto de herramientas para depurar la placa y gestionar el uso de la memoria.
- En cuanto a la comunidad, debido a la implementación de Arduino en una gran variedad de placas,

[15] https://registry.platformio.org/

es la que mayor comunidad tiene en comparación con ESP-IDF.

{ https://docs.platformio.org/en/latest/what-is-platformio.html https://marketplace.visualstudio.com/items?itemName=platformio.pl
ide https://docs.platformio.org/en/latest/platforms/index.html https://docs.platformio.org/en/latest/librarymanager/dependencies.htr
https://docs.platformio.org/en/latest/integration/ide/index.html https://registry.platformio.org/ https://www.espboards.dev/blog/esp-
idf-vs-arduino-core/ }

Trello[16]: es una aplicación web que permite elaborar listas con tareas de forma visual al estilo Kanban. Esta aplicación de Atlassian se basa en tableros personalizables donde se muestran y categorizan ideas o tareas, compuestos principalmente por:

- Listas: representadas en formato de columnas, suelen hacer referencia a las distintas fases que puede tener una tarea. Por ejemplo, en un tablero puede haber tres listas: “por hacer”, “en curso” y “hecho”.
- Tarjetas: cada tarjeta en un tablero puede ser una tarea o idea relacionada con el trabajo o proyecto, y se colocan en las columnas correspondientes según su estado o tipo. La idea de las tarjetas es moverlas según avance el desarrollo del proyecto. Se les puede además añadir un miembro asignado para reflejar qué usuario está encargado de la tarea, evitando la necesidad de preguntar personalmente; fechas de vencimiento de la tarea, con capacidad de notificar según se acerque la fecha y de marcarlas como realizadas; ficheros adjuntos para organizarlos, descripción, comentarios y checklists para dividir una tarea grande en varias tareas pequeñas.
- Miembros: cada uno con responsabilidades asignadas a las tarjetas y permisos para utilizar el tablero.

Trello se puede utilizar tanto para fines personales como empresariales, sin importar el tamaño del proyecto, utilizando los tableros para, por ejemplo, la gestión de proyectos de software, realizar anuncios escolares, planificar clases o gestionar los casos en un despacho de abogados. Gracias a la concentración de la información en un tablero, que permite observar todo de un rápido vistazo, se puede realizar un seguimiento sencillo de las tareas y sus plazos, coordinar a los miembros de un equipo y llevar a cabo reuniones productivas y motivadoras.

Trello tiene la capacidad de ampliar la funcionalidad gracias a las integraciones con otras aplicaciones (como Slack, Gmail o GitHub), a las automatizaciones de tablero sin código (por ejemplo, mover una tarjeta a una determinada lista cuando se complete) para centrarse únicamente en el trabajo, y a los *power-ups*, que actúan como extensiones de la funcionalidad básica de Trello recopiladas en un catálogo.

Para el proyecto, el alumno ha creado un tablero adaptado a sus necesidades (detallado posteriormente) usando el plan gratuito de Trello. Este tablero permite llevar un seguimiento de todas las tareas, tanto de código como de la memoria, realizadas durante el desarrollo, y ha facilitado las reuniones de seguimiento al poder mostrar a los tutores el estado de las tareas. Utiliza los siguientes *power-ups*:

[16] https://trello.com/es

- GitHub[17]: sirve para llevar un seguimiento de lo sucedido en GitHub desde el tablero, como adjuntar a tarjetas ramas, commits, incidencias y pull requests y asociar repositorios a tarjetas [4].
- Smart Fields[18]: permite crear campos personalizados en las tarjetas, como campos de texto, de número o de fecha, y soporta el uso de fórmulas para calcular el valor del campo. Estos campos se pueden mostrar desde la vista general del tablero, sin necesidad de entrar a ver los detalles de la tarjeta [9].

{ https://en.wikipedia.org/wiki/Trello https://trello.com/es https://trello.com/es/about https://trello.com/es/tour

}

Doxygen[19]: utilizada para la documentación de este trabajo, esta herramienta de código abierto, muy empleada en el desarrollo de software, permite obtener documentación a partir del código de forma sencilla. Posibilita la generación automática de documentación en distintos formatos, como HTML, PDF, Word y XML, a partir de los comentarios insertados en el código durante el desarrollo, analizando la información de las distintas clases, funciones y variables. Gracias a esta automatización, se agiliza y estandariza el proceso de documentación de proyectos, lo cual es beneficioso para entender el proyecto y el código que lo compone, además de mejorar la colaboración entre los miembros del equipo de desarrollo y el mantenimiento del propio código.

Soporta C++, C, Python, Java, PHP y otros lenguajes. La documentación se realiza a partir de comentarios, como se muestra en el Listado 3.1.

Listado 3.1: Ejemplo de uso de Doxygen

```
1  /**
2   * <Descripción corta de la función>
3   *
4   * <Descripción larga de la función>
5   *
6   * @param parametro1 <Descripción del primer parámetro de entrada>
7   * @param parametro2 <Descripción del segundo parámetro de entrada>
8   * @param <Descripción del resto de parámetros>
9   * @return <Descripción del valor o valores que retorna>
10  */
11 public int ejemploFuncion(int parametro1, bool parametro2, ...)
```

Entre las capacidades adicionales se encuentran las referencias cruzadas entre distintas partes de la documentación, soporte de Markdown en los comentarios, dibujo de diagramas para representar gráficamente

[17] https://trello.com/power-ups/55a5d916446f517774210004/github
[18] https://trello.com/power-ups/5e2212c3ba57415ef2ef9352/smart-fields
[19] https://doxygen.nl/

clases, herencias y relaciones entre partes del código, personalización de la documentación resultante y configuración mediante un fichero Doxyfile con distintos parámetros establecidos por el usuario.

En el caso de este trabajo, se ha generado documentación en formato HTML para crear una página web con toda la información de las distintas funciones que componen el proyecto, y se ha personalizado con Doxygen Awesome[20], un tema CSS aplicable a la página para disponer de una página con un aspecto moderno, limpio y compatible con la interfaz móvil.

{ https://doxygen.nl/ https://en.wikipedia.org/wiki/Doxygen https://jothepro.github.io/doxygen-awesome-css/ }

Plantilla de TFG[21]: fue desarrollada por Félix Albertos Marco, profesor del Grado de Ingeniería Informática de la sede de la UCLM de Talavera de la Reina, ofreciendo una alternativa para desarrollar la memoria del Trabajo de Fin de Grado. Esta plantilla, utilizada en este TFG, se caracteriza por emplear Markdown para redactar todos los apartados, evitando las complicaciones del formato del documento (como el interlineado, fuentes, cabeceras y pie de página y saltos de página), además de proporcionar la portabilidad del trabajo, ya que estos ficheros son de marcado ligero y no necesitan editores ni sistemas operativos específicos. El soporte de Markdown también incluye todos los elementos que este lenguaje ofrece, como tablas, listados, fragmentos de código o formateado del texto.

Esta plantilla está compuesta por una serie de ficheros y directorios donde el autor debe ir cumplimentando todo lo necesario que quiera reflejar en el documento final. Su funcionamiento es sencillo, ya que opera con una imagen Docker que, utilizando la herramienta Pandoc, convierte los ficheros a LaTeX y genera el documento PDF final, simplemente ejecutando el comando `make docker` (en el caso de tener instalado Docker) o ejecutando la herramienta en una máquina GNU/Linux. El conjunto de las características de la plantilla de Félix Albertos permite al alumno centrarse únicamente en el contenido que tiene que escribir, agilizando notablemente el desarrollo del trabajo.

{ https://www.felixalbertos.com/resources/downloads/tfg_template.html }

Teams[22]: es una aplicación multiplataforma de colaboración en equipo. Se puede utilizar de manera gratuita con el plan personal o con una licencia Microsoft 365, diferenciándose en ambos casos la versión empresarial de la personal debido a sus limitaciones.

Microsoft 365 es una suscripción que incluye las aplicaciones de Microsoft Office (Word, PowerPoint y Excel), almacenamiento en la nube, copias de seguridad y correo electrónico, que habilita el uso en la nube de éstas y permite mantener siempre la última versión con los parches de seguridad correctos mediante un pago mensual o anual. La UCLM provee a sus usuarios de cuentas de Microsoft 365 Empresa, por lo que

[20] https://jothepro.github.io/doxygen-awesome-css/
[21] https://www.felixalbertos.com/resources/downloads/tfg_template.html
[22] https://www.microsoft.com/es-es/microsoft-teams/group-chat-software/

se ha aprovechado esta suscripción para este desarrollo.

Teams permite comunicarse con personas a través de mensajes instantáneos formateables, con soporte para menciones, respuestas, adjuntos y traducciones; y a través de reuniones de dos o varios participantes, en forma de llamadas de audio o videollamadas, con capacidad de compartir pantalla, grabar la reunión, colaborar en aplicaciones (como una pizarra compartida) y traducción y transcripción en tiempo real. Tanto las traducciones como las transcripciones reducen la barrera del lenguaje que puede surgir entre personas de diferentes regiones al trabajar en conjunto. Estas opciones de comunicación evitan el uso de múltiples programas o tecnologías, centralizando así toda la comunicación.

Se basa en el concepto de equipos o *teams*, espacios de trabajo compartido para comunidades, grupos o equipos de personas. Se utilizan para compartir mensajes, contenido y herramientas entre los miembros que lo componen, y pueden ser privados o públicos, permitiendo asignar roles y responsabilidades sobre el propio equipo a los miembros. Dentro de los equipos están los canales, áreas para tener conversaciones sobre temas específico en los que es posible restringir el acceso únicamente a ciertos miembros del equipo. Tener canales dentro de los equipos permite subdividir el equipo en grupos de trabajo, agrupar las conversaciones, almacenar ficheros compartidos y reunirse con el resto de miembros.

Por otro lado, para realizar una comunicación directa o en grupos pequeños se pueden utilizar chats normales en lugar de equipos.

Teams dispone de otras funciones, como aplicaciones instalables dentro del propio software que permiten ampliar la funcionalidad, y un calendario con capacidad de programar reuniones y enviarlas a otros usuarios para que las acepten o las declinen.

Para este TFG, Microsoft Teams ha permitido mantener un contacto directo entre los tutores y el alumno a través de un equipo organizado por canales destinados a las partes de desarrollo y memoria. Estos han agilizado la consulta de dudas y de disponibilidad, y organizar archivos compartidos. Por ejemplo, en el canal destinado a la memoria se alojaron las distintas versiones del documento de la memoria, mientras que en el canal de desarrollo se encuentran enlaces, manuales y códigos de ejemplo. Además, se han programado y celebrado reuniones de seguimiento en esta aplicación, de la cual tanto el alumno como los tutores tenían experiencia de uso.

{ https://teamsdemo.office.com https://en.wikipedia.org/wiki/Microsoft_Teams https://www.microsoft.com/es-es/microsoft-teams/compare-microsoft-teams-business-options https://www.microsoft.com/es-ES/microsoft-365/buy/compare-all-microsoft-365-products }

Visio[23]: es un programa para crear diagramas y vectores. Su uso está limitado a los usuarios con licencia de Microsoft 365, servicio mencionado anteriormente y del cual la UCLM distribuye la licencia a todos los usuarios de la universidad, como es el caso del autor, quien ha destinado el uso de la aplicación al dibujo

[23] https://www.microsoft.com/es-es/microsoft-365/visio/flowchart-software/

de algunos esquemas de la memoria. Este programa, disponible tanto en línea como en una aplicación descargable, permite crear diagramas, imágenes vectoriales y objetos fácilmente, facilitando la visualización de datos e ideas de forma atractiva, lo cual es útil en los equipos y en la documentación para asegurar de manera sencilla la comprensión de los conceptos.

El programa contiene plantillas personalizables para que los usuarios no partan desde cero en la creación del diagrama, y con las extensas librerías de objetos que incluye se pueden crear diagramas profesionales, como los de flujo, red, Venn, bloques, UML, PERT, organigramas, matrices de negocio o mapas mentales. También ofrece compatibilidad con otras herramientas de Microsoft, como Teams para realizar diagramas en colaboración, y Power BI o Excel, para ofrecer una manera alternativa de visualizar los datos. Los objetos creados con este programa se pueden compartir en su propio formato VSDX, o exportarlos en otros más comunes como JPEG, PNG y PDF para hacerlos accesibles.

{ https://www.microsoft.com/es-es/microsoft-365/visio/flowchart-software#x68bca46524744e268ea489ad8cc29bbb https://en.wikipedia.org/wiki/Microsoft_Visio }

Inkscape[24]: Es una herramienta de software gratuita, multiplataforma y de código abierto que permite diseñar gráficos vectoriales. Surgió en 2003 como una bifurcación de otro editor con el mismo propósito, Sodipodi. Este programa permite generar y manipular ficheros SVG, en los cuales las imágenes no están dibujadas por píxeles ni puntos, sino por líneas y vectores, lo que permite ampliar la imagen sin pérdida de calidad. Inkscape se centra en este formato y permite a los diseñadores crear una gran variedad de gráficos, como ilustraciones, diagramas, iconos, logotipos, mapas, diseños y otras imágenes complejas, de forma sencilla. Permite crear imágenes renderizables utilizando formas vectoriales, como líneas, rectángulos, eclipses, estrellas, y texto, los cuales pueden rellenarse con colores, patrones y gradientes, además de modificar el borde de dichos objetos. En caso de que las herramientas incluidas no sean suficientes, dispone de una galería de extensiones instalables para personalizar y aumentar la funcionalidad del programa.

En este trabajo, el uso de Inkscape ha sido para casos específicos en los que se ha requerido una imagen, como puede ser para el logo de la herramienta. /*TODO: todavia no se sabe*/

{ https://inkscape.org/es/acerca-de/ https://inkscape.es/ https://en.wikipedia.org/wiki/Inkscape }

Navegador web, como Firefox[25] y Microsoft Edge[26]: utilizados en este proyecto para acceder a las herramientas web mencionadas anteriormente y realizar búsquedas de información acerca de partes del código en desarrollo y del contenido de esta memoria.

/*TODO: poner lo que vaya a utilizar para diagramas*/

[24] https://inkscape.org/es/
[25] https://www.mozilla.org/es-ES/firefox/new/
[26] https://www.microsoft.com/es-es/edge/

3.1.3. Lenguajes

C++: es un lenguaje de programación diseñado en 1979 por Bjarne Stroustrup para extender el ya existente C y añadir mecanismos de manipulación de objetos. Es un lenguaje que requiere una compilación para que el programa pueda ser ejecutado, además de ser multiparadigma, abarcando programación estructurada, orientada a objetos, genérica e imperativa (es decir, las instrucciones indican cómo realizar una tarea y se conoce el estado del programa durante la ejecución).

La orientación a objetos permite descomponer los proyectos en distintos archivos que contienen tipos de datos abstractos o clases. Estas son estructuras de tipos de datos concretos con un nombre definido. Además, permite asignar propiedades y funciones ejecutables a un objeto de esa clase y relacionar las propias clases con otras, por ejemplo, para que una clase pueda heredar de otra.

Tanto C++ como su predecesor C son lenguajes que requieren que el programador tenga claro qué hacer y cómo hacerlo, ya que permiten al programador expresar lo que quiere hacer sin restringir lo que está permitido. Son lenguajes simples, concisos y rápidos, pero, por otro lado, la compilación del código no comprueba las conversiones incorrectas de tipos, los índices erróneos de arrays ni el mal uso de punteros. Además, no cuentan con un recolector de basura que gestione la memoria automáticamente, por lo que el programador debe realizarlo manualmente. Esto convierte a C++ en un lenguaje frágil y exigente en la gestión de recursos.

C++ fue diseñado teniendo en cuenta la programación de sistemas, tanto grandes como embebidos y con recursos limitados, utilizando el rendimiento, la eficiencia y la flexibilidad como puntos clave de diseño.

{ https://www2.eii.uva.es/fund_inf/cpp/temas/1_introduccion/introduccion.html https://academia-lab.com/enciclopedia/programacion-imperativa/ http://cslibrary.stanford.edu/101/EssentialC.pdf https://aprendiendoarduino.wordpress.com/2015/03/26/lenguaje-de-programacion-c/ https://en.wikipedia.org/wiki/C%2B%2B }

En el caso de este trabajo, se utiliza el framework de Arduino, una implementación de este lenguaje que limita las funciones que C++ trae por defecto y se puede usar, e incluye algunas propias. Junto a este lenguaje, se han utilizado las siguientes librerías compatibles con la placa ESP32 y que la añaden funcionalidades y mejoran el desarrollo de código:

- FreeRTOS[27]: esta biblioteca permite utilizar FreeRTOS como un sistema operativo en el proyecto, encargándose del acceso al hardware y de la gestión de las tareas. Es un sistema operativo en tiempo real, centrado en tener un control preciso del tiempo, y está destinado a ser utilizado en dispositivos embebidos.

  FreeRTOS se basa en tareas definidas por el desarrollador, cada una con su frecuencia de ejecución.

[27] https://www.freertos.org/

Estas tareas se ejecutan dependiendo de su estado, los cuales pueden ser: disponible, en ejecución, suspendida o bloqueada. El núcleo de FreeRTOS es el scheduler o planificador, que se encarga de gestionar y ejecutar las tareas. Este distribuye el tiempo de ejecución del procesador entre las tareas y asigna las tareas a los núcleos del procesador disponibles (solo se puede ejecutar una tarea por núcleo), permitiendo la ejecución en paralelo o alternando la ejecución entre varias tareas. Además, es el responsable de cambiar los estados entre las tareas.

FreeRTOS también ofrece otras funcionalidades y herramientas, como semáforos para sincronizar tareas, colas para compartir datos entre tareas, temporizadores y un bus de notificaciones.

{ https://es.wikipedia.org/wiki/FreeRTOS https://www.luisllamas.es/como-usar-freertos-en-arduino/ }

- bblanchon/ArduinoJson[28]: es capaz de abstraer documentos JSON y las herramientas para serializarlos y deserializarlos, añadiendo una inexistente compatibilidad de C++ con JSON. El formato de texto plano JavaScript Object Notation o JSON almacena datos de manera estructurada, y es común su uso en sistemas de comunicación que intercambian información y en la operación de páginas web. Soporta objetos como texto, números, booleanos y nulos, los cuales se pueden agrupar dentro de otros objetos o en arrays; y se almacenan en formato clave-valor, facilitando el acceso al valor a través de su clave. Este proyecto utiliza ArduinoJson 7.0.4 para crear ficheros estructurados en los que almacenar propiedades de objetos, como una lista de placa suscriptoras, y almacenarlos en una tarjeta microSD, para poder recuperarlos durante el arranque de la placa ESP32.

- x385832/Elog[29]: creada para manejar eficientemente los logs o registros sin que impacte en el rendimiento de la ejecución, añade la capacidad de mostrar los registros por terminal serial, agregarlos a un fichero en una tarjeta SD y almacenarlos en la memoria flash. Admite distintos tipos de registros, dependiendo de cómo de crítico sea el mensaje, diferenciar mensajes por clases especificadas y mostrar marcas de tiempo. En el caso de este trabajo, se utiliza en la versión 1.1.5 para añadir mensajes de registro, como avisos o errores, para informar del estado de la ejecución del código, y ofrecer la posibilidad de almacenar los registros en una tarjeta microSD.

{ https://arduinojson.org/v7/faq/automatically-serialize-an-object/ https://www.luisllamas.es/en/arduino-json/ https://registry.platformio.org/libraries/x385832/Elog }

Markdown: es un lenguaje de marcado ligero utilizado en este TFG para redactar los distintos apartados de la memoria. Markdown permite escribir en documentos de texto plano, utilizando su propia sintaxis para indicar formatos especiales y el aspecto que debe tener (como negrita, cursiva o títulos), con la característica de mantener una lectura natural del documento en casos en los que no sea posible previsualizar el formato.

[28] https://registry.platformio.org/libraries/bblanchon/ArduinoJson
[29] https://registry.platformio.org/libraries/x385832/Elog

A pesar de ser un lenguaje ligero, no limita el uso a únicamente texto, ya que permite insertar imágenes, tablas, listados y otros tipos de elementos.

Otra característica notable es su portabilidad, ya que se trata de ficheros de texto que se pueden abrir con cualquier editor y en cualquier plataforma. Esta portabilidad permite comparar este lenguaje con Word, ya que este último mantiene el contenido encerrado en un formato de archivo propietario. Sin embargo, no todo lo escrito en Markdown se mantiene en ficheros .md, ya que es posible realizar conversiones a otros formatos, como HTML, para ver el contenido desde un navegador web, o PDF, para transformarlo en documento portátil, mediante scripts o aplicaciones. Estas características convierten a Markdown en un candidato ideal para todo tipo de usos, como la creación de sitios web, documentos, libros, presentaciones y mensajes de correo electrónico.

{ https://www.markdownguide.org/getting-started/ }

3.2. Metodología

En este apartado se tratan conceptos básicos de la metodología junto a la forma en la que se aplica a este TFG.

3.2.1. Metodologías tradicionales y ágiles en el desarrollo de software

Las metodologías de desarrollo de software son marcos, compuestos de técnicas y métodos, utilizados con el fin de estructurar, planear y controlar el proceso para aumentar la productividad y la calidad del software. Estas se deben elegir con precaución, ya que, si son apropiadas para el equipo y para el proyecto y su implementación se lleva a cabo correctamente, beneficiarán al proporcionar estimaciones superiores de tiempo y esfuerzo, priorizar las tareas, comprender con claridad los esfuerzos futuros y previos, posibilitar tiempo adicional suficiente para hacer ajustes, reportar al cliente el estado del proyecto y, además, entregar sistemas y productos estables y de calidad [8] [35].

Las metodologías se organizan en dos grandes bloques: las tradicionales y las ágiles.

3.2.1.1. Metodología tradicional

Las metodologías tradicionales en el desarrollo de software se caracterizan por su enfoque estructurado y secuencial en la gestión de proyectos. Enfatizan la planificación exhaustiva y la fijación de los requisitos a lograr en las primeras etapas del proyecto, y siguen de manera lineal unas fases bien definidas, abarcando el inicio del proyecto, la planificación, la implementación, la verificación y el mantenimiento, durante las cuales se genera el producto junto a la documentación clara [35] [20].

Por un lado, ofrecen unos objetivos claramente definidos y sus procesos son controlables, pero, por otro

lado, el desarrollo requiere más responsabilidad por parte de los involucrados y es poco tolerante a cambios tardíos, ya sean de requisitos o para resolver problemas, influyendo negativamente en el presupuesto y los plazos de entrega [35] [20] [47].

Entre la variedad de esta clase de metodologías se encuentran:

- Desarrollo en cascada: enfocado en la planificación y en el desarrollo lineal de etapas definidas que generan documentación, como el análisis de requisitos, diseño, implementación, pruebas y mantenimiento. Al final del desarrollo, se entrega un único producto que cumple con los requisitos especificados. Ofrece una estructura clara y un buen mantenimiento, pero supone mayores costos al tratar problemas tardíos. Es adecuado en proyectos con requisitos estables y entornos regulados por normativas [47].
- Proceso Unificado de Desarrollo: centrado en casos de uso y en la gestión de riesgos. Se desarrolla en ciclos iterativos que resultan en incrementos que se van acercando al producto deseado. Utiliza UML para modelar y documentar las partes del sistema, junto a otras buenas prácticas para gestionar los riesgos y mejorar la calidad del producto final. Su uso resulta adecuado en proyectos complejos a gran escala y que necesitan una gestión de riesgos cuidadosa, pero requiere un equipo capacitado [47].
- Desarrollo en espiral: toma aspectos del desarrollo en cascada y en iteraciones, realizando ciclos que entregan versiones del software y que están formados por fases de planificación, análisis de riesgos, ingeniería y evaluación. Permite adaptar el desarrollo a los cambios y gestionar los riesgos, pero es una metodología compleja de implementar y necesita un equipo eficaz. Es utilizada en proyectos complejos con requisitos cambiantes y un alto nivel de incertidumbre o riesgos [47].

3.2.1.2. Metodología ágil

La metodología ágil es una metodología de gestión de proyectos que impulsa el desarrollo iterativo mediante entregas incrementales, la colaboración en equipo y la planificación y el aprendizaje continuos [32] [31] [50]. Está basada en el Manifiesto Ágil, acordado en 2001 por 17 desarrolladores de software agrupados como la Agile Alliance [50], y valora [12]:

- Los individuos e interacciones sobre los procesos y las herramientas.
- El software funcionando sobre la documentación extensiva.
- La colaboración con el cliente sobre la negociación contractual.
- La respuesta ante el cambio sobre seguir un plan.

En esta metodología, el desarrollo se realiza en ciclos iterativos breves o sprints de una duración entre 1 y 4 semanas, cuyo resultado es un pequeño incremento de la funcionalidad del software. En los sprints, el equipo realiza la codificación, las pruebas y la comprobación de calidad, fases regidas por requisitos de trabajo pendiente bien definidos y priorizados. En el desarrollo existe el rol del propietario del producto,

[31] [50] es quien se encarga de representar a las partes interesadas del proyecto y se compromete a responder a las preguntas de los desarrolladores a lo largo de las iteraciones [50]. El propietario del producto se encarga de añadir, modificar y repriorizar las tareas pendientes dentro de un listado o backlog al principio de cada sprint, en función de las necesidades del cliente [31]; y al final de cada iteración es quien se reúne con las partes interesadas para revisar el progreso del desarrollo y reevaluar la priorización de las tareas, con la intención de satisfacer a los interesados [50]. Además, las tareas con las que se trabaja deben ser claras para el equipo de desarrollo, por lo que requieren una continua refinación por parte del propietario y el equipo hasta que el propio equipo decida que están listas para trabajarlas [31].

Los incrementos conllevan el beneficio de corregir constantemente errores que surjan y adaptarse a cambios, en vez de tener que enfrentarlos al final del proyecto o en etapas en las que puede resultar muy costoso [50].

Una característica fundamental del desarrollo ágil es la comunicación directa y eficaz entre los involucrados, lo que se traduce en la capacidad de comunicar de forma efectiva las necesidades del cliente y las tareas con el equipo de desarrollo. El Manifiesto Ágil sugiere una comunicación cara a cara, que permite reducir el tiempo de preguntas y respuestas para ponerse a trabajar cuanto antes. Es común apoyarse en un radiador de información que muestra el estado del proyecto, como un tablero con post-its o una pantalla grande. Esto se aplica a otra característica del desarrollo: la realización de sesiones de retroalimentación diarias para revisar el progreso, las tareas a completar en el día, los impedimentos y los riesgos. Son sesiones breves, de unos 15 minutos y sin entrar en detalle, ya que tienen el fin de reforzar la comunicación y la coordinación del equipo de desarrollo [50].

En esta metodología, es común utilizar herramientas y técnicas de integración continua y entrega continua que mejoren la calidad del proyecto y agilicen el desarrollo. Por ejemplo, automatizar la compilación, la ejecución de pruebas y el despliegue. Esto evita procesos lentos y propensos a errores, y permite demostrar un producto de calidad al final de cada iteración [31] [50].

La metodología ágil es adaptativa, es decir, se centra en adaptarse rápidamente a las necesidades cambiantes. Durante la planificación se identifican hitos, pero estos son flexibles, además de que difícilmente se describe el futuro del desarrollo. Esta característica es muy importante, ya que es la que la separa de las metodologías tradicionales predictivas. Las metodologías predictivas planifican las tareas y características previstas en detalle y tienen en cuenta los riesgos conocidos, resultados de un primer análisis exhaustivo. En el caso de no haber tenido en cuenta algún riesgo o que ocurra algún otro imprevisto, puede ser difícil realizar cambios [50]. Esta comparación no intenta demostrar que una es superior a la otra, ya que ambas son adecuadas dependiendo de las necesidades específicas del proyecto, e incluso existe la posibilidad de realizar un enfoque híbrido combinándolas [27].

Junto a los valores mencionados previamente, el Manifiesto Ágil sigue los siguientes principios [13], algu-

nos de los cuales se han tratado a lo largo de este apartado:

- La prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
- Se acepta que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
- La entrega de software funcional es frecuente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
- Los responsables de negocio y los desarrolladores trabajan juntos de forma cotidiana durante todo el proyecto.
- Los proyectos se desarrollan en torno a individuos motivados. Deben disponer del entorno y el apoyo que necesitan, y contar con la confianza hacia la ejecución del trabajo.
- El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
- El software funcionando es la medida principal de progreso.
- Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.
- La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
- A intervalos regulares, el equipo reflexiona sobre cómo ser más efectivo para, a continuación, ajustar y perfeccionar su comportamiento en consecuencia.

Dentro de este tipo de metodología se pueden encontrar, entre otros:

- Scrum: define de forma flexible roles, eventos, artefactos y reglas que los equipos deben seguir para mejorar la productividad, la calidad del trabajo y la comunicación. Los eventos principales que ocurren son los sprints iterativos e incrementales en los cuales existe un control continuo de la calidad del producto. Debe utilizarse por equipos de menos de 10 personas que sepan autoorganizar su trabajo. En grandes proyectos, el uso de esta metodología puede resultar en una pérdida de la perspectiva general durante el desarrollo. Además, aunque es compatible con todo tipo de proyectos, no es sencillo de integrar en todas las organizaciones [25].
- Kanban: se centra en la mejora del flujo de trabajo, de la productividad y de la calidad a través del uso de un tablero. El tablero muestra las tareas como tarjetas que se van desplazando entre las columnas para representar si están pendientes, en curso o concluidas. Estas columnas admiten priorización y limitación, evitando tener equipos sobrecargados. Las tareas se completan antes de comenzar otras, y ocurren reuniones regulares para obtener retroalimentación. Es una metodología fácil de integrar y que muestra los avances del proyecto de manera sencilla, pero requiere que el trabajo sea divisible en fases y que los miembros se adapten a trabajar en distintas etapas del proceso [24].

- Programación Extrema (eXtreme Programming): se basa en un entorno de comunicación constante entre desarrolladores y cliente, y en el cual existe respeto para tratar errores y críticas. Es utilizada en proyectos en los cuales el cliente no tiene una idea clara del producto final, por lo que ocurren procesos iterativos para entregar una versión y revisarla. Sin embargo, requiere una gran inversión de tiempo y disciplina para llevarla a cabo [23].

3.2.2. Scrum

Como está definido en la Guía de Scrum [45], Scrum es un marco de trabajo ligero para el desarrollo ágil de software que ayuda a personas, equipos y organizaciones a generar valor a través de soluciones adaptativas. Busca conseguir un equipo que trabaje en colaboración y obtener el mejor resultado posible de los proyectos, pero en vez de detallar instrucciones específicas a seguir, ofrece una guía de relaciones e interacciones en el equipo. Esta libertad permite usar varios procesos, técnicas y métodos que visibilicen la eficacia de la gestión, el entorno y las técnicas de trabajo para poder mejorarlas.

Se basa en el conocimiento adquirido a partir de la experiencia, en la toma de decisiones basada en observaciones, y en el pensamiento centrado en lo esencial. Utiliza un enfoque iterativo e incremental en el cual un equipo con las habilidades necesarias para el proyecto celebra ciertos eventos que previenen y controlan riesgos, inspeccionan, adaptan y desarrollan el trabajo. Estos eventos están envueltos en uno principal, el sprint.

Los sprints son etapas de desarrollo que duran entre 1 y 4 semanas de trabajo (según se establezca en el equipo). En estos se convierten las ideas en valor para el producto, llamados incrementos, los cuales se producen con cada sprint y avanzan el desarrollo hacia el objetivo del producto final. Los incrementos pasan por una revisión con los interesados del proyecto, y se ajusta lo necesario para poder comenzar el siguiente sprint, uno detrás de otro.

Tanto el sprint como los otros eventos tienen utilidad, ya que se puede visibilizar el trabajo realizado en los artefactos, inspeccionar los mismos para detectar variaciones o problemas y adaptar los procesos y los materiales en caso de que el desarrollo se desvíe fuera de los límites aceptables o si el producto resultante es inaceptable.

El proceso de llevar a cabo Scrum se define por sus eventos, roles y artefactos, resumidos en inglés en la figura 3.1 y que se detallan a lo largo de este apartado.

https://arrizabalagauriarte.com/en/10-principios-clave-metodologias-agile-scrum/

Durante los sprints y en toda la implementación de Scrum, se encuentra una pequeña unidad de personas responsables: el equipo Scrum. Este equipo es multifuncional, y sus miembros poseen las habilidades necesarias para generar valor en cada sprint. Es autogestionado, asignándose internamente las tareas y la forma de realizarlas, y está enfocado en el objetivo del producto. El equipo Scrum es responsable de llevar

Figura 3.1: Esquema de la ejecución de Scrum (Fuente: [1])

a cabo todas las actividades relacionadas con el producto (colaboración con interesados, mantenimiento, desarrollo, investigación y otras), además de crear un incremento útil y valioso en cada sprint. No tiene jerarquías ni subequipos, todos sus miembros están al mismo nivel. El tamaño ideal del equipo es de 10 personas o menos para asegurar la agilidad, facilitar la distribución de tareas entre ellos, promover una buena comunicación y mantener la productividad. Esto no implica que no pueda haber más personas, en dicho caso se organizarán varios equipos Scrum. En el equipo existen tres roles principales:

- Equipo de desarrollo: grupo de personas comprometidas a crear cualquier aspecto de un incremento de calidad en cada sprint. Se encargan de crear el Sprint Backlog con las tareas a realizar durante el sprint y de adaptar su plan de trabajo diario para cumplir con el objetivo del sprint.
- Propietario del Producto o Product Owner: única persona responsable de maximizar el valor del producto resultante. Se encarga o se responsabiliza de que el Product Backlog contenga las tareas necesarias para alcanzar el objetivo del producto, representen las necesidades de los interesados, estén ordenadas y priorizadas, y sean comprendidas. Toda la organización debe respetar sus decisiones, y cualquier posible Product Backlog debe ser justificado ante él.
- Scrum Master: responsable de asegurar que Scrum sea comprendido y ejecutado tal y como está definido en la Guía de Scrum, tanto por el equipo de desarrollo como por la organización en su conjunto, por lo que puede ser común a toda la organización. Ayuda a comprender el trabajo, facilita su realización, elimina barreras y mejora las prácticas utilizadas.

Estos miembros, junto a los interesados, tienen una idea clara del estado del desarrollo y de las tareas realizadas a partir de los siguientes artefactos, tratados a lo largo de los sprints:

- Pila del Producto o Product Backlog: lista ordenada y emergente de las tareas a realizar necesarias para mejorar el producto. Estas tareas se refinan mediante su detallado y división en partes más pequeñas y precisas, listas para ser seleccionadas en la planificación del siguiente sprint. Esta lista se basa en el cumplimiento del objetivo del producto, que es el objetivo a largo plazo.
- Pila del Sprint o Sprint Backlog: lista de tareas creada por los desarrolladores, que representa el plan de trabajo para el sprint con el fin de lograr el objetivo del sprint. No es fija y se actualiza a lo largo del sprint.
- Incremento: representa un avance hacia el objetivo del producto. Se generan uno o varios incrementos al final del sprint, los cuales se añaden a los anteriores y se verifican para garantizar su correcto funcionamiento en conjunto. Un incremento se considera completo cuando cumple con los estándares de calidad definidos en la definición de terminado.

Una característica importante de Scrum es la celebración de eventos que permiten inspeccionar y adaptar los artefactos de manera formal. Estos eventos se utilizan para aplicar regularidad a las reuniones y minimizar la necesidad de reuniones no planificadas. La Guía de Scrum recomienda que siempre se realicen en el mismo lugar y a la misma hora para reducir la complejidad. Los distintos tipos de eventos son los siguientes:

- Sprint: además de lo mencionado anteriormente, cabe detallar algunas características de este evento. El sprint es una etapa en la que se procura mantener constante la calidad del desarrollo y no se realizan cambios que puedan comprometer el objetivo del sprint. Permite modificaciones en el Product Backlog para refinar los elementos, así como aclaraciones y negociaciones con el Product Owner. La ejecución del sprint garantiza la inspección y adaptación del progreso hacia el objetivo del producto al menos una vez al mes. Su duración se justifica porque periodos más largos pueden hacer que el objetivo del sprint se vuelva inválido e incrementen la complejidad y los riesgos. Los sprints cortos limitan el riesgo de coste y esfuerzo a intervalos pequeños, comparables a proyectos de menor envergadura. Los sprints anteriores son útiles para la toma de decisiones y para pronosticar el progreso del sprint, mediante prácticas como el análisis de trabajo pendiente (burn-downs), de trabajo completado (burn-ups) y los flujos acumulativos. Existe la posibilidad de cancelar el sprint si su objetivo se vuelve obsoleto, y solo el Product Owner tiene la autoridad para hacerlo.
- Planificación del Sprint: sirve para establecer el trabajo que se realizará durante el sprint. Es el primer evento que se ejecuta antes de comenzar el sprint, en el cual el equipo Scrum colabora para crear un plan. El Product Owner debe asegurarse de que los asistentes estén preparados para discutir los elementos importantes del Product Backlog y su relación con el objetivo del producto. Los temas que se tratan son:
    - El valor que aporta el sprint y el incremento resultante al producto y a los interesados, definiendo un objetivo del sprint.
    - Las tareas a realizar durante el sprint, seleccionadas por los desarrolladores a partir del Pro-

duct Backlog. El equipo Scrum puede refinar estos elementos para aumentar su comprensión y confianza.

- El plan del trabajo necesario para crear un incremento que cumpla con la definición de terminado, elaborado por los desarrolladores.

Este evento no se limita únicamente al equipo Scrum, se pueden invitar a otras personas que asesoren al equipo.

- Scrum diario: es un evento diario de unos 15 minutos realizado durante el sprint en el que los desarrolladores inspeccionan el progreso hacia el objetivo del sprint y adaptan el Sprint Backlog según sea necesario. Los participantes informan brevemente sobre el progreso y generan un plan viable para el siguiente día laboral, identificando impedimentos, promoviendo una toma de decisiones rápida y eliminando la necesidad de reuniones no planificadas. Esto se traduce en un equipo bien comunicado, enfocado y autogestionado. El Scrum diario no tiene una estructura fija y no excluye la posibilidad de realizar discusiones más detalladas a lo largo del día.

- Revisión del Sprint: es una sesión de trabajo colaborativo que se realiza con el fin de inspeccionar los resultados al final del sprint y determinar futuras adaptaciones del proyecto. Durante este evento, el equipo Scrum presenta los resultados de su trabajo a los interesados y se discute el progreso hacia el objetivo del producto, incluyendo posibles ajustes al Product Backlog para satisfacer nuevas oportunidades.

- Retrospectiva del Sprint: es la última reunión del sprint, con el fin de planificar formas de aumentar la calidad y efectividad del desarrollo. El equipo Scrum analiza las interacciones entre personas, los procesos y herramientas, y la definición de terminado para identificar qué salió bien durante el sprint, los problemas encontrados y la forma de resolverlos (en el caso de hacerlo). Tras ello, identifican cambios útiles para mejorar, y se abordan al momento o se añaden al Sprint Backlog.

3.2.3. OpenUP

OpenUP es definido [7] como una metodología de desarrollo de software iterativa e incremental que manifiesta el proceso de construir un sistema mediante el desarrollo colaborativo de software, con un enfoque pragmático y una filosofía ágil. A su vez, permite centrar el desarrollo en los casos de uso del usuario final, en la identificación y mitigación de riesgos, y en la arquitectura del sistema desde las primeras fases. Esta metodología está concebida para ser utilizada como una base, conteniendo los elementos fundamentales para el desarrollo, y se le puede añadir o adaptar el contenido de otros procesos según las necesidades del proyecto.

OpenUP se apoya en equipos saludables, colaborativos y con conocimiento compartido acerca del proyecto, en maximizar los beneficios y cumplir con los requisitos, y en utilizar prácticas que permitan obtener

retroalimentación temprana y continua a partir de incrementos en la funcionalidad del producto.

Los proyectos que implementan OpenUP se dividen en iteraciones cortas planificadas. El uso de las iteraciones evita la generación innecesaria de documentación, diagramas e iteraciones excesivas, y facilita la detección temprana de errores, haciendo más eficiente el uso de recursos y tiempo. Durante las iteraciones, el equipo se organiza y se compromete para alcanzar los objetivos. Se define una lista de tareas detalladas que resultan en microincrementos y en la entrega de valor a las partes interesadas. Los microincrementos se producen a un ritmo constante y permiten comprender el valor que aporta la iteración al proyecto y el progreso del mismo.

OpenUP organiza las iteraciones en cuatro fases del ciclo de vida del proyecto, compuestas por objetivos y actividades que permiten gestionar eficientemente el proyecto:

- Inicio: se determina el alcance del proyecto y los objetivos a cumplir. Tras esta fase, si se considera viable, el proyecto se continúa.
- Elaboración: trata la mitigación de riesgos, la arquitectura del sistema, la validación de requisitos, el diseño, los costos y los cronogramas.
- Construcción: detalla los requisitos y su cumplimiento, el diseño, la implementación de funcionalidades y las pruebas del software.
- Transición: se centra en la transferencia del software al entorno del cliente, en la aceptación y conformidad del producto por parte de las partes interesadas, en la corrección de defectos y otros ajustes finales.

3.2.4. Implementación en este proyecto

Tras la definición de los conceptos necesarios acerca de las metodologías, esta sección desarrolla la manera en la que se aplicaron Scrum y OpenUP. El desarrollo de este proyecto se ha realizado teniendo OpenUP en mente, mientras que el esfuerzo de este proyecto se ha gestionado utilizando la metodología Scrum. Pese a conocer que la aplicación no es la más adecuada al ser un equipo más pequeño de lo normal y en el que existe un único desarrollador, la aplicación de esta metodología procura ser fiel con todos los elementos esenciales que la componen. Se ha optado por metodologías ágiles debido a que es un proyecto cuyo desarrollo comienza con unos requisitos básicos y atraviesa fases en las que se identifican fallos de rendimiento, posibles mejoras y nuevos requisitos. Una característica importante de estas metodologías es la constante retroalimentación de los interesados, que permiten redirigir el proyecto según transcurría, y la baja formalidad en cuanto a documentación permitía centrar el esfuerzo en el funcionamiento del proyecto. Por otro lado, la elección ha sido influida por la preferencia y la previa experiencia del autor con estas metodologías.

3.2.4.1. Roles y artefactos

En este proyecto han participado los tutores y el alumno con los roles distribuidos de la siguiente manera:

- Interesados: los tutores Rubén Cantarero Navarro y Ana Rubio Ruiz.
- Equipo de desarrollo: compuesto únicamente por el alumno Rubén Gómez Villegas.
- Product Owner: los tutores Rubén Cantarero Navarro y Ana Rubio Ruiz.
- Scrum Master: tanto el alumno como los tutores. Debido a que ambas partes tienen conocimiento acerca de Scrum y llegaron a un acuerdo acerca de como gestionar este proyecto, no ha sido necesaria la intervención de este rol.

Los distintos artefactos Scrum se han representado en un tablero Kanban creado con Trello para facilitar el seguimiento de la metodología. Un tablero Kanban funciona como una herramienta ágil para gestionar proyectos, ayudando a visualizar el trabajo, limitar el trabajo en curso y maximizar la eficiencia del mismo [42] [10], además de tener equipos comprometidos con una cantidad de trabajo adecuada [42]. Está compuesto por tarjetas y otras señales visuales que describen el trabajo, permitiendo que los compañeros y los interesados conozcan su estado [42]. Las tarjetas están agrupadas en columnas que representan etapas del flujo de trabajo, y se van moviendo entre columnas según avanza la tarea hasta su finalización [42] [10].

/*TODO: insertar imagen del tablero. Clonar tablero, mostrar entre sprints 4 y 1, hide empty columns*/

El tablero de Trello utilizado es una representación digital de Kanban, creado por y para el alumno y las necesidades del desarrollo, y contiene las siguientes columnas:

- *Info, templates and utils*: contiene información del tablero y una tarjeta que actúa como plantilla para representar tareas.
- *Product backlog*: contiene las tareas propias de un Product Backlog de Scrum.
- *Sprint backlog*: contiene las tareas propias de un Sprint Backlog de Scrum.
- *In progress*: lista las tareas que se están ejecutando a lo largo del sprint. Son tareas que aún no están finalizadas.
- *Review/Testing*: representa las tareas que se han terminado de desarrollar pero que deben pasar por una revisión de su correcto funcionamiento y otra del cumplimiento con las necesidades de los interesados, siendo esta última ejecutada durante la revisión del sprint.
- *Done (Sprint X)*: cuando las tareas están terminadas, acaban en esta lista, en la cual se marcan apropiadamente como finalizadas. Existe una lista por cada sprint, para poder diferenciar qué tarea se hizo cuándo, siendo “X” el número del sprint.

Como se ha mencionado anteriormente, las tarjetas se mueven dependiendo de la fase en la que se encuentren. Las tarjetas representan tareas, y se muestran en dos vistas, mostradas en la figura 3.2: la vista previa, que muestra la tarjeta resumida en una columna, y la vista detallada, la cual se abre al hacer clic en la tarjeta resumida. Las tarjetas tienen las siguientes propiedades configuradas:

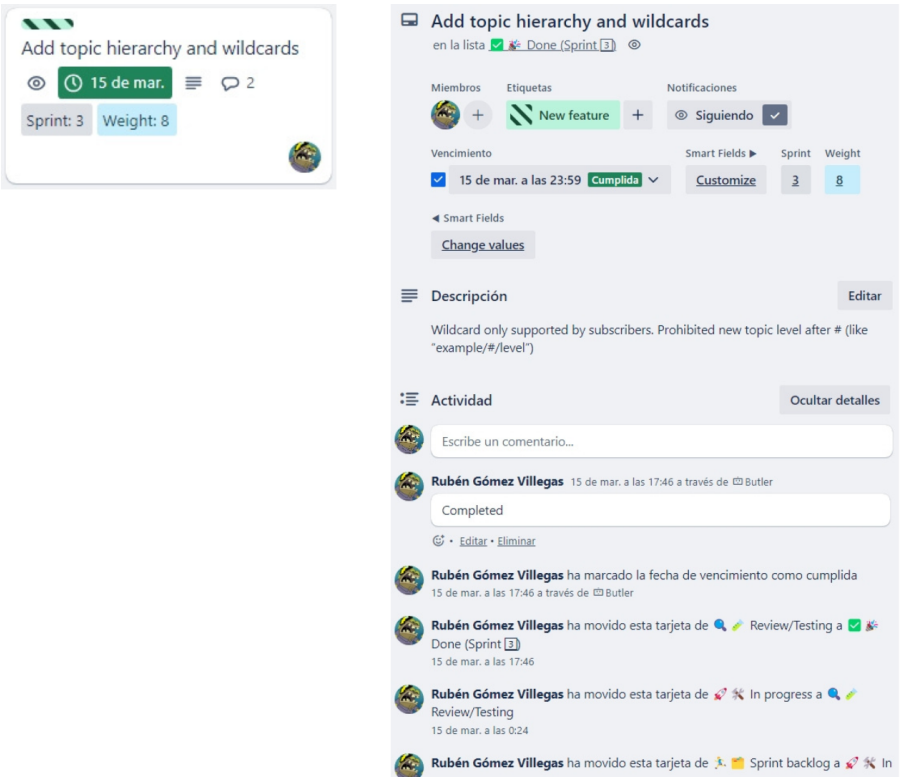

Figura 3.2: Vistas de las tarjetas del tablero: vista previa (izquierda) y detallada (derecha)

- Miembros: quienes se encargan de la tarea. Son usuarios registrados en Trello y miembros del tablero que pueden autoasignarse o ser asignados por otros.
- Etiquetas: se utilizan para clasificar de manera visual el tipo de tarjeta. Existen 6 tipos distintos, cada uno con un color y patrón (útil para las personas con daltonismo) como se puede ver en la figura 3.3, siendo posible aplicar varios simultáneamente. Son los siguientes:
  - *New feature*, verde: la tarjeta representa una tarea cuyo resultado es una nueva funcionalidad en el proyecto.
  - *Test*, amarillo: la tarjeta representa una tarea que prueba funciones y elementos ya existentes en el proyecto.
  - *Fix*, naranja: la tarjeta representa una tarea que arregla funciones y elementos ya existentes en el proyecto.
  - *Blocked*, rojo: la tarjeta representa una tarea pausada debido a un impedimento u obstáculo, como una dependencia de otra tarea.
  - *Details card*, azul: la tarjeta muestra información, no representa ninguna tarea.
  - *Report*, rosa: la tarjeta representa una tarea relacionada con esta memoria del proyecto.
- Vencimiento: fecha límite de realización de la tarea.

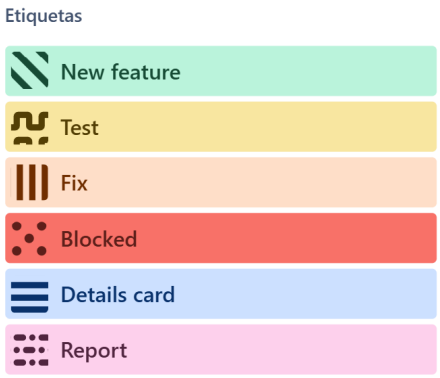

Figura 3.3: Etiquetas disponibles en el tablero

- Sprint: un campo personalizado que representa el número del sprint de la tarea.
- Weight: un campo personalizado numérico que representa el número del sprint de la tarea. El weight o peso es un número subjetivo que representa la dificultad, el tiempo a dedicar y la experiencia o falta de esta por parte del desarrollador respecto a una tarea.
- Card count: un campo personalizado numérico utilizado para contabilizar la cantidad de tarjetas en una lista. Siempre tiene el valor 1 y está oculto.
- Descripción: en el caso de necesitar detallar o añadir información adicional a las tareas, existe este campo.
- Actividad: muestra las acciones que han ocurrido (como el paso de una lista a otra o la asignación de miembros) y permite a los miembros del tablero hacer comentarios.

En este tablero se pueden identificar dos tipos de tarjetas: las tarjetas de información y las de tareas. Las de tareas se crean a partir de una plantilla y se mueven entre las listas durante los sprints. En cambio, las de información son fijas y sirven para detallar el uso de una lista y resumir su contenido. Éstas se encuentran en la primera posición de las listas, tienen la etiqueta azul y dos campos que se pueden observar desde la vista previa:

- Sum of Weight: calcula la suma de los pesos de una lista de tareas. Útil para calcular el peso total de un sprint.
- Sum of Card count: calcula la cantidad de tareas en una lista. Debido a que Trello por defecto no proporciona una forma de realizar esto, el campo es útil en las listas en las que no caben todas las tarjetas en pantalla, además que sirve para comparar entre sprints la cantidad de tareas hechas.

Además, las tarjetas de información en los sprints indican la fecha de inicio y fin del sprint. En el caso de ser un sprint finalizado, se le adjunta el pull request realizado en GitHub para conocer fácilmente el incremento del producto al que dio lugar.

En el tablero se ha habilitado la automatización que ofrece Trello a través de la selección de varios triggers o disparadores y de acciones consecuentes relacionadas con las propiedades de las tarjetas y las acciones realizables sobre ellas. En este proyecto, las reglas activadas son las siguientes, mostradas en la figura 3.4:

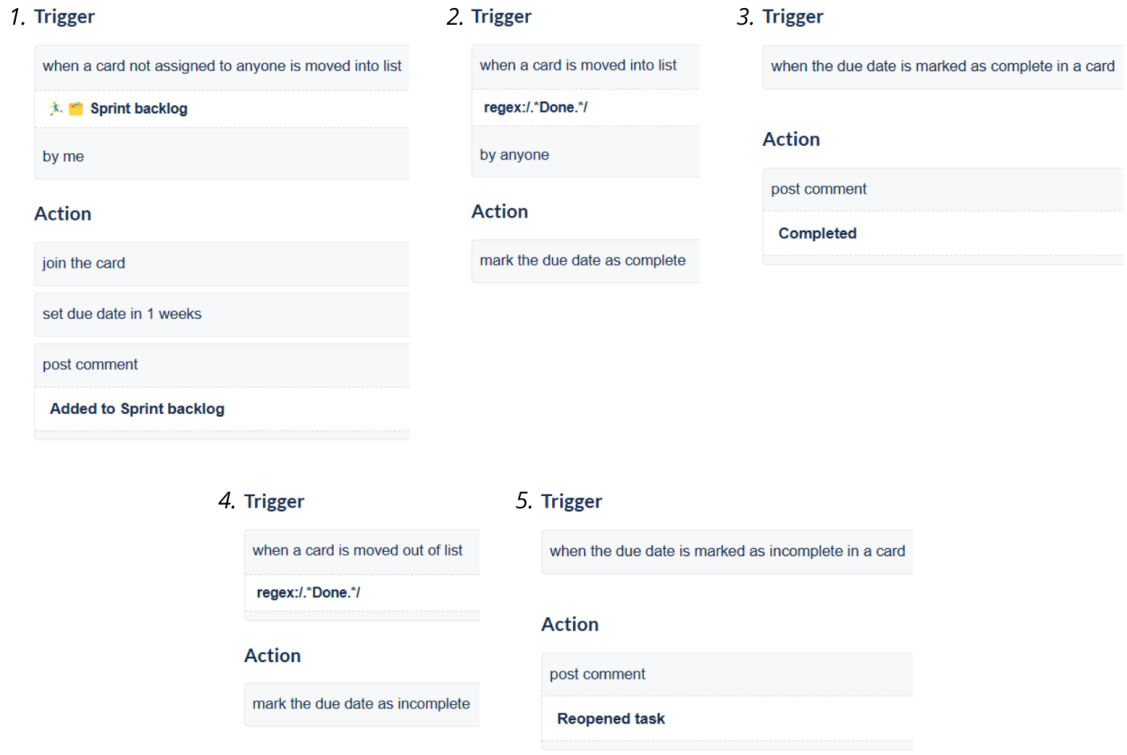

Figura 3.4: Reglas de automatización utilizadas en el tablero

1. *Cuando una tarjeta sin miembro asignado se mueve al Sprint backlog por un miembro X, el miembro X se le asigna, se establece la fecha de vencimiento a 1 semana y se añade el comentario "Añadido al Sprint backlog"*: automatiza la selección de una tarea para ejecutarla durante el sprint.
2. *Cuando una tarjeta se mueve a cualquier lista "Done" por cualquiera, marcar la fecha de vencimiento como completa*: automatiza el completado de una tarea.
3. *Cuando la fecha de vencimiento de una tarjeta se marca como completa, se añade el comentario "Completado"*: automatiza el registro del completado de una tarea.
4. *Cuando una tarjeta se mueve fuera de cualquier lista "Done" por cualquiera, marcar la fecha de vencimiento como incompleta*: automatiza la reversión del completado de una tarea.
5. *Cuando la fecha de vencimiento de una tarjeta se marca como incompleta, se añade el comentario "Tarea reabierta"*: automatiza el registro de la reapertura de una tarea.

Cabe recalcar que este tablero lo ofrece el alumno para su uso en cualquier desarrollo de software, y que en este caso su contenido está en inglés debido a que su uso ha sido en gran parte para gestionar la parte codificada, también en inglés, de este TFG.

3.2.4.2. Ejecución de los eventos

Los eventos definidos en Scrum se han realizado mediante Teams, donde se programan las reuniones para crear recordatorios y llevarlas a cabo, evitando el consumo de tiempo que supone para el alumno y los tutores el desplazamiento, y facilitando la manera de mostrar código y otros aspectos del proyecto. Se han ejecutado de la siguiente forma:

Planificación del sprint: antes de iniciar el sprint, el desarrollador y los Product Owners comparan los objetivos del proyecto y las necesidades con las tareas ya realizadas, y debaten sobre qué tareas nuevas añadir en forma de tarjetas al product backlog para realizar a lo largo del desarrollo. Estas tareas también se pueden basar en ideas surgidas tanto por el desarrollador como por los propietarios. Asimismo, se planifica el fin del sprint, limitando el tiempo a 1 semana si el incremento se basa en el código del producto, o a 2 semanas si el incremento se basa en la documentación y la memoria de este proyecto; y se consideran las tareas para añadirlas al sprint backlog y así realizarlas en el sprint entrante, con el suficiente refinamiento y que resulten en uno o varios incrementos previamente debatidos. Estas tareas son añadidas a la lista correspondiente del tablero y el desarrollador rellena los campos de manera apropiada, incluyendo el peso de la tarea, siendo este un valor útil para conocer si es demasiada tarea a realizar en un sprint. Normalmente, si el equipo de desarrollo fuera de mayor tamaño habría que llevar a cabo procesos como el *planning poker* para estimar un peso con el que el equipo esté de acuerdo. Pero como no es el caso, el desarrollador elige un valor de la secuencia modificada de Fibonacci: 0.5, 1, 2, 3, 5, 8, 13, 20, 40 o 100. Como parte final del sprint, se programan las demás reuniones y la próxima planificación del sprint. Finalmente, en el tablero Kanban se prepara una nueva lista *Done* con el número del sprint.

Sprint: a lo largo del sprint, el desarrollador ejecuta las tareas establecidas en el sprint backlog haciendo uso de las herramientas correspondientes. En cuanto comienza una tarea, la tarjeta es añadida a la lista *In progress* y se moverá hacia la lista *Review/Testing* cuando se finalice. Dentro de esta fase, la tarea puede pasar por pruebas realizadas dentro del sprint, y si se reconsidera como no completada, se mueve de vuelta a *In progress*. Dichas tareas se cumplen siguiendo la calidad definida en los siguientes apartados, y durante su progreso se irán realizando *commits* o publicaciones con los cambios al repositorio, concretando más, en la rama `develop` (se explica acerca de esta rama en posteriores apartados). A modo de añadir métricas de tiempo invertido en el desarrollo, los sprints se han realizado a lo largo de los días de la semana, con una jornada no fija de entre 1 y 8 horas diarias.

Scrum diario: no se ha realizado debido a que el equipo de desarrolladores está formado únicamente por el autor y a que los otros roles de Scrum ya conocen la tarea a realizar a lo largo del sprint por la reunión de planificación. Por otro lado, los tutores han mostrado la predisposición de atender lo antes posible cualquier consulta, ya sea a través de llamada rápida o mensaje de Teams, siendo esto un buen sustitutivo del Scrum diario.

Revisión del sprint: el desarrollador y los product owners revisan el estado final de las tareas establecidas en el sprint, además de probar la nueva funcionalidad del incremento. Tras una posible modificación rápida, las tarjetas de tareas del tablero Kanban son movidas a la columna *Done* del correspondiente sprint, marcándose así como completadas, y se realiza un pull request desde la rama `develop` a `main`, aumentando la versión del producto (ramas y versionado explicados en posteriores apartados). En el caso de ser una versión superior a la 1.0.0, se ejecuta el proceso de crear un release en el repositorio, subir la nueva versión del producto al registro de PlatformIO y generar la página web de documentación.

Retrospectiva del sprint: utilizadas para comentar el rendimiento de los sprints, la idoneidad de la cantidad de tareas, mejoras en la gestión del proyecto y en las propias reuniones. Por ejemplo, en una reunión de este tipo se debatió el paso de sprints de 1 semana a 2 semanas cuando las tareas están relacionadas con la redacción y no con el código, debido a que las primeras consumían más tiempo. No existía un procedimiento para el desarrollo de esta reunión, simplemente el tema se exponía y se resolvía en consecuencia.

3.2.5. Control de versiones

En este apartado se detalla la forma en la que se ha implementado el control de versiones, una característica beneficiosa en el desarrollo de software, ya que permite a los equipos de desarrollo trabajar en paralelo evitando conflictos en el código. Además, facilita el rastreo y la gestión de los cambios realizados a lo largo del desarrollo, y permite comparar el estado actual del código fuente con versiones previas para resolver errores [15]. Para este TFG, se han aprovechado las herramientas que ofrece GitHub, como las ramas, los pull requests y las liberaciones o releases.

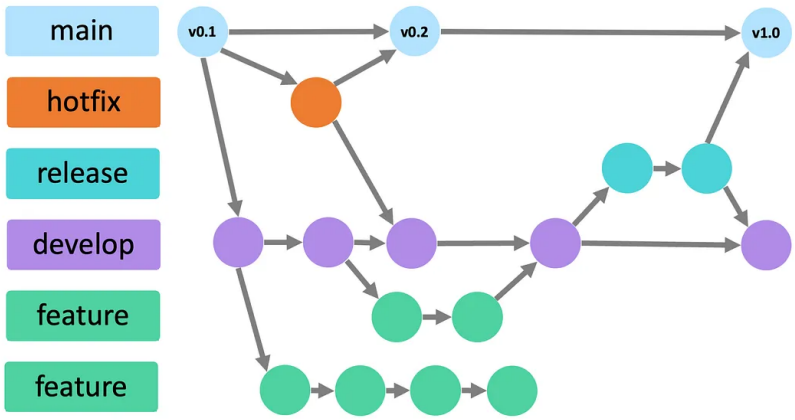

Figura 3.5: Ilustración de Gitflow (Fuente: [49])

En cuanto a la ramificación del repositorio, se utiliza una adaptación del flujo de trabajo Gitflow, un modelo de creación de ramas publicado en 2010 por Vincent Driessen, útil para proyectos de entrega continua [14]. Según la publicación [21], y para detallar la figura 3.5, existen dos ramas principales que perduran a lo

largo del tiempo: `master` o `main`, que registra el historial de publicación oficial del proyecto, y `develop`, en la cual se van integrando las funciones para la siguiente liberación del código. Cuando el código de la rama `develop` se encuentra en un punto estable y listo para la liberación, todos los cambios se fusionan hacia la rama `master`, lo que se traduce en una nueva versión del producto que se etiqueta. Junto a estas ramas, se utilizan otras con un tiempo de vida limitado y que en algún momento se eliminarán:

- Ramas de características o *feature*: se utilizan para desarrollar nuevas características para una futura versión. Se crean a partir de `develop`, existen mientras la característica está en desarrollo, y finalmente se fusionan de nuevo hacia `develop` añadiendo la característica, o se descartan.
- Ramas de lanzamiento o *release*: nombradas `release-<next-version>`, se utilizan para preparar una nueva versión de producción. Se crea una a partir de `develop` cuando este último refleja las características destinadas a la versión a construir, se le asigna un número de versión, se mantiene la rama para añadir cambios ligeros y metadatos, hasta que finalmente está lista para el lanzamiento y se fusiona en `master`, representando una nueva versión. También se fusiona hacia `develop`, para mantener esos pequeños cambios realizados a esta rama.
- Ramas de parche rápido o *hotfix*: nombradas `hotfix-<next-version>`, se utilizan para resolver un fallo crítico en una versión de producción. Se crea a partir de `master`, se solventa el problema y finalmente se fusiona en `master`, aumentando la versión del producto, y en `develop`.

El uso de Gitflow por parte del alumno proviene de la previa experiencia utilizándolo y la compatibilidad que tiene con las metodologías del proyecto, pero en este caso difiere respecto a lo definido anteriormente:

- No se utilizan ramas de características. Al ser un equipo de desarrollo de un solo miembro y al no haberse planeado el desarrollo de distintas características en paralelo (la complejidad del proyecto no lo requiere), se ha considerado innecesaria la creación de este tipo de ramas, por lo que todos los cambios se van publicando a la rama `develop`.
- No se utilizan ramas de lanzamiento. Para simplificar el desarrollo, la única rama que contiene los lanzamientos es `main`, y en caso de realizar un ligero cambio en el código se crean ramas *hotfix*. `develop` se fusiona directamente hacia `main`, y si luego ocurre un *hotfix* en `main`, `main` se fusiona hacia `develop`.

Para etiquetar las versiones se ha utilizado un versionado semántico, que sirve para lanzar versiones que puedan indicar a los sistemas dependientes si dicha versión puede romper el funcionamiento del sistema. En su definición [40] se trata el formato de la versión, que es `X.Y.Z`, formado por números enteros no negativos ni precedidos por ceros, y el incremento de versiones está definido de la siguiente forma:

- `X`, versión mayor: se han realizado cambios incompatibles. Cada incremento de `X` provoca establecer `Y` y `Z` a 0.
- `Y`, versión menor: se ha añadido funcionalidad compatible con versiones anteriores. Cada incremento

de `Y` provoca establecer `Z` a 0.

- `Z`, versión parche: repara errores compatibles con versiones anteriores.

La primera versión lanzada al público se define como `1.0.0`, y es posible agregar identificadores y metadatos a versiones prelanzamiento. Por ejemplo, `1.0.0-alpha+001`.

Se ha preferido el versionado semántico sobre otros, como el versionado de calendario (usado por Ubuntu, cuya versión 24.04 muestra que salió en abril de 2024). Por ejemplo, ante el de calendario, es preferible este uso para que la versión no muestre una sensación de obsolescencia al utilizar una versión lanzada hace años ni una falta de madurez al utilizar una versión lanzada recientemente. De manera general, su uso se debe a la facilidad, la comprensión y a la previa experiencia del alumno.

/TODO: hablar de automatización (RELEASES AUTOMATICAS, GENERACION DE LA WEB, CUANDO SE IMPLEMENTE)/

3.2.6. Criterios de calidad y estilo del código

A lo largo del desarrollo del código, se han seguido unos criterios de calidad establecidos por el autor con el fin de obtener un código comprensible tanto para desarrolladores como para usuarios, fácil de mantener y retomar en el futuro. Además, se ha procurado mantener un estilo homogéneo del código, lo cual ofrece una sensación de calidad y que también se define en este apartado. Los criterios son los siguientes:

- El código está escrito en inglés, incluyendo variables, funciones, comentarios y mensajes de registro. Al utilizar el idioma más hablado en el mundo [6], la base de usuarios puede ser mucho más amplia que si se utilizara el idioma local.
- El código funciona como se espera. A partir de una tarea se crea el código, y dicho código cumple con la tarea asignada.
- El código es autoexplicativo. En caso de ser complejo de comprender, hay comentarios que facilitan su entendimiento.
- Gran parte de las funciones están documentadas utilizando Doxygen, especificando una descripción, parámetros (en caso de existir) y valores de retorno (en caso de tenerlos). Al comentar, tienen prioridad las funciones disponibles para ser utilizadas por los usuarios finales. También se documentan otras partes del código, como las enumeraciones y los valores configurables.
- Las importaciones, enumeraciones, definiciones de valores, documentación y otros elementos considerados necesarios se muestran en archivos de cabecera (`.h`), evitando sobrecargar los archivos de código fuente (`.cpp`) con elementos no funcionales.
- El funcionamiento del código es eficiente, aprovechando las capacidades de la placa de desarrollo y del lenguaje de programación. Para ello, se realizan pruebas y consultas con los tutores y con herramientas de inteligencia artificial.

- Las líneas de comentarios no superan los 80 caracteres de longitud y las de código no superan los 120. En caso de necesitar más espacio, se introducen saltos de línea. Estos valores se originan a partir del primer almacenamiento digital de la historia de la informática, las tarjetas perforadas de 80x12 agujeros, y en la actualidad se limitan los caracteres por fila para mejorar la legibilidad del código sin necesidad de hacer scroll horizontal [38].
- El estilo de programación utilizado es el *Kernighan & Ritchie*, que evita líneas sin sentido y el consumo innecesario de espacio vertical en la sintaxis. Junto a este estilo, se emplea la indentación mediante tabulaciones y la convención de nombrado *lowerCamelCase*, útil para identificar distintas palabras al nombrar elementos del código sin necesidad de ampliar horizontalmente el código. El Listado 3.2 muestra un ejemplo de este estilo.

Listado 3.2: Ejemplo de estilo del código Kernighan & Ritchie con lowerCamelCase

```
1  public int main(int argc, char *argv[]) {
2      while (x == y) {
3          doSomething();
4          doSomethingElse();
5          if (error)
6              fixError();
7          else
8              continueAsNormal();
9      }
10 }
```

- La ejecución muestra registros o *logs* si el usuario lo desea, los cuales se distribuyen por el código en puntos clave. Estos registros permiten realizar un seguimiento de lo que realiza la ejecución del código del proyecto e identificar fácilmente los fallos.

/*TODO: mencionar test si al final se hacen TENER EN CUENTA LAS MENCIONES A TESTS A LO LARGO DEL APARTADO*/

Capítulo 4

Resultados

En este capítulo se describe el avance del desarrollo del proyecto a través de iteraciones o sprints, mencionando los logros y problemas identificados en cada uno. Además, se muestra, mediante diagramas y listados, el funcionamiento de la herramienta desarrollada en su última versión a fecha de redacción de esta memoria. Este proyecto se puede encontrar en el repositorio /*TODO: insertar repo*/

4.1. Iteración 0

Periodo: 28/09/2023 - 18/02/2024 (20,43 semanas). Peso total: 143.

El objetivo principal de este sprint fue la preparación previa al desarrollo y la obtención de conocimientos necesarios.

Tareas:

- Preparar el entorno de desarrollo (peso: 3). Consiste en la instalación de las herramientas y la creación del repositorio.
- Familiarizarse con ESP32 y PlatformIO (peso: 3). Aprender a utilizar PlatformIO para desarrollar programas y subirlos a la placa de desarrollo.
- Escribir el anteproyecto del TFG (peso: 5).
- Comunicar placas ESP32 a través de ESP-NOW (peso: 2). Aprender a enlazar las placas a través de sus direcciones MAC y comprender el uso de las funciones para formar y emitir un mensaje (procurando no sobrepasar el tamaño máximo del payload) y para extraer los datos al recibirlo.
- Entender y realizar ejemplos de FreeRTOS (peso: 100). Leer la documentación de FreeRTOS y comprender los distintos conceptos que abarca, como las tareas (con sus estados y prioridades), las colas y los semáforos.
- Realizar ejemplos emisor-receptor con ESP-NOW y FreeRTOS (peso: 5). Aplicar el conocimiento

de ambos aspectos para crear tareas que envíen y reciban datos aprovechando los hilos de ejecución disponibles.

- Realizar ejemplos publicador-suscriptor con MQTT (peso: 5). Desplegar un broker Mosquitto en Docker y crear un código para que la placa ESP32 publique mensajes, mientras que otro permite obtenerlos. Esto permite conocer los tipos y parámetros al trabajar con mensajes y otras capacidades en MQTT.
- Preparar tablero Kanban (peso: 20). Crear el tablero, las columnas y las plantillas, junto a la configuración necesaria para la gestión del proyecto a lo largo de todo el desarrollo.

4.2. Iteración 1

Periodo: 19/02/2024 - 25/02/2024 (1 semana). Peso total: 26.

El objetivo principal de este sprint fue realizar una primera versión funcional centrada en los roles de broker y suscriptor, sin tener en cuenta los topics.

Tareas:

- Crear el esqueleto de la librería (peso: 5). Preparar los ficheros necesarios para definir una librería en PlatformIO. En esta iteración, la librería contenía el fichero de configuración y el de código para definir un broker vacío.
- Definir la estructura de los mensajes (peso: 8). La primera definición de los mensajes se realizó fuera de la librería, en ficheros `.h` que formaban parte de un ejemplo. En esta definición se distinguen tres tipos de mensajes: suscripción, desuscripción y publicación, que eran contenidos en un mensaje genérico con un campo de byte `msgType` para identificar el tipo de mensaje. Tanto los mensajes de suscripción como los de publicación contenían únicamente un campo `topic`, definido con capacidad para 10 caracteres, y los de publicación además contenían un campo `content` con capacidad para 20 bytes. El código de la definición se muestra en el Listado 4.1.

Listado 4.1: Primera definición de los mensajes

```
1 #define MSGTYPE_SUBSCRIBE 1
2 #define MSGTYPE_UNSUBSCRIBE 2
3 #define MSGTYPE_PUBLISH 3
4
5 typedef struct {
6     char topic[10];
7 } SubscribeAnnouncement;
8
9 typedef struct {
```

```
10      char topic[10];
11  } UnsubscribeAnnouncement;
12
13  typedef struct {
14      char topic[10];
15      uint8_t content[20];
16  } PublishContent;
17
18  typedef union {
19      SubscribeAnnouncement subscribeAnnouncement;
20      UnsubscribeAnnouncement unsubscribeAnnouncement;
21      PublishContent publish;
22  } PayloadUnion;
23
24  typedef struct {
25      uint8_t msgType;
26      PayloadUnion payload;
27  } Message;
```

- Crear un ejemplo de suscriptor (peso: 13). Sin utilizar la librería, se crearon dos códigos de ejemplo para los roles broker y suscriptor a partir de la definición de mensajes anterior. El broker utilizaba FreeRTOS para crear dos tareas que se ejecutan en paralelo: la primera generadora de mensajes de prueba en una cola de mensajes, y la otra para enviarlos a las placas que se hayan suscrito al broker (sin tener en cuenta los topics en esta iteración). Esta suscripción es controlada al añadir las direcciones de los suscriptores a un vector, y por cada nuevo suscriptor se comprueba si ya existía. El suscriptor, por su parte, crea mensajes de suscripción y se los envía al broker cada 2 segundos, además de controlar los mensajes de suscripción recibidos. El pseudocódigo de estos ejemplos se muestra en los Listados 4.2 y 4.3.

Listado 4.2: Pseudocódigo de la primera definición del broker

```
1  struct PayloadStruct: //Estructura de mensajes enviados
2      int number
3
4  vector<uint8_t> subscribers //Contiene las direcciones MAC de las placas
       ↪ suscriptoras
5  QueueHandle_t messagesQueue //Cola de mensajes para enviar
6
```

```
//Cada vez que se reciba un mensaje, se llama a esta función
Función OnDataRecv(uint8_t *mac, uint8_t *incomingData):
    Message recvMessage = Copia de los datos recibidos
    Si recvMessage.msgType es un mensaje de suscripción:
        Si mac no estaba en subscribers: //Si es un suscriptor nuevo
            Imprimir mensaje de aviso de suscripción con topic
            Agregar mac a subscribers

//Tarea encargada de gestionar la cola de mensajes
Función DispatchMessagesTask():
    Message message
    Por siempre:
        Recibir de la cola de mensajes y copiar en message
        Por cada subscriber en subscribers:
            Registrar el subscriber como par
            Enviar message a subscriber

//Tarea encargada de crear mensajes de prueba en la cola de mensajes
Función ProduceMessageTask():
    Por siempre:
        PayloadStruct payload
        payload.number = Número aleatorio
        Message sendMessage //Nuevo mensaje
        sendMessage.msgType = Definir mensaje como publicación
        sendMessage.payload.subscribeAnnouncement.topic = "mock" //Asignar topic
            ↪ al payload SubscribeAnnouncement
        sendMessage.payload.content = payload
        Enviar sendMessage a la cola messagesQueue
        Esperar 1 segundo

Función setup():
    Inicializar ESP-NOW
    Registrar la función OnDataRecv como callback al recibir un mensaje
    Inicializar la cola messagesQueue
    Crear tarea DispatchMessagesTask
    Crear tarea ProduceMessageTask
```

Listado 4.3: Pseudocódigo de la primera definición del suscriptor

```
struct PayloadStruct: //Estructura de mensajes recibidos
    int number

uint8_t destBoardAddr[] //Contiene la dirección MAC de la placa broker

//Cada vez que se reciba un mensaje del broker, se llama a esta función
Función OnDataRecv(uint8_t *mac, uint8_t *incomingData):
    Message recvMessage = Copia de los datos recibidos
    PayloadStruct messagePayload = recvMessage.payload.publish.content //
        ↪ Contenido recibido
    Imprimir messagePayload.number

Función subscribe(char* topic):
    Message subMsg //Nuevo mensaje
    subMsg.msgType = Definir mensaje como suscripción
    subMsg.payload.subscribeAnnouncement.topic = topic //Asignar topic al
        ↪ payload SubscribeAnnouncement
    Enviar subMsg al broker

Función setup():
    Inicializar ESP-NOW
    Registrar la función OnDataRecv como callback al recibir un mensaje
    Registrar el broker como par

Función loop():
    subscribe("topic1")
    Esperar 2 segundos
```

/*TODO: preguntar si mencionar los ejemplos que utilicé*/

4.3. Iteración 2

Periodo: 26/02/2024 - 08/03/2024 (1,71 semanas). Peso total: 49.

El objetivo principal de este sprint fue el soporte agregado a la publicación de mensajes y al uso de los temas.

Tareas:

- Probar la subida de la librería a PlatformIO (peso: 5).
- Crear ejemplo publicador (peso: 8). En adición al ejemplo del suscriptor realizado en la iteración 1, teniendo en común la desatención del topic y el desarrollo fuera de la librería, el publicador crea mensajes que publica en el broker. El Listado 4.4 muestra el pseudocódigo de este ejemplo. Asimismo, se añadió al broker la capacidad de gestionar estos mensajes, modificando la función `OnRecv` como se observa en el Listado 4.5.

Listado 4.4: Pseudocódigo de la primera definición del publicador

```
1  struct PayloadStruct: //Estructura de mensajes enviados
2      int number
3
4  uint8_t destBoardAddr[] //Contiene la dirección MAC de la placa broker
5
6  Función generateAndPublish(char* topic):
7      PayloadStruct payload
8      payload.number = Número aleatorio
9      Message sendMessage //Nuevo mensaje
10     sendMessage.msgType = Definir mensaje como publicación
11     sendMessage.payload.publish.topic = topic //Asignar topic al payload
           ↪ PublishContent
12     sendMessage.payload.content = payload
13     Enviar sendMessage al broker
14
15 Función setup():
16     Inicializar ESP-NOW
17     Registrar el broker como par
18
19 Función loop():
20     generateAndPublish("topic1")
21     Esperar 5 segundos
```

Listado 4.5: Pseudocódigo de la modificación del broker del Listado \ref{resultados:broker1} para admitir publicaciones

```
1  //Cada vez que se reciba un mensaje, se llama a esta función
2  Función OnDataRecv(uint8_t *mac, uint8_t *incomingData):
```

```
    Message recvMessage = Copia de los datos recibidos
    Si recvMessage.msgType es un mensaje de suscripción:
        Si mac no estaba en subscribers: //Si es un suscriptor nuevo
            Imprimir mensaje de aviso de suscripción con topic
            Agregar mac a subscribers
    Sino si recvMessage.msgType es un mensaje de publicación:
        PublishContent pubMsg = recvMessage.payload.publish
        PayloadStruct pubContentPayload = pubMsg.content
        Imprimir pubContentPayload
        Enviar recvMessage a la cola messagesQueue
```

- Mejorar la estructura de los mensajes (peso: 8). La estructura original de la iteración 1 tenía un problema, y es que al utilizar un `typedef union` (utilizado para crear un mensaje genérico), todos los mensajes tenían el mismo tamaño, independientemente de su contenido, provocando un consumo de memoria innecesario. Para solucionarlo, se cambió la estructura de los mensajes, utilizando el primer byte para identificar el tipo de mensaje, y dependiendo del tipo se conoce la cantidad de bytes que habrá que leer del mensaje recibido. Este cambio se puede ver en el Listado 4.6. A su vez, se realizaron ligeros cambios en el publicador, suscriptor y broker para gestionar los mensajes, añadiendo el mencionado método para identificarlos.

Listado 4.6: Segunda definición de los mensajes

```
typedef enum {
    MSGTYPE_SUBSCRIBE = 0x00,
    MSGTYPE_UNSUBSCRIBE,
    MSGTYPE_PUBLISH
} MessageType;

typedef struct {
    MessageType type;
} MessageBase;

typedef struct : public MessageBase {
    char topic[10];
} SubscribeAnnouncement;

typedef struct : public MessageBase {
    char topic[10];
```

```
} UnsubscribeAnnouncement;

typedef struct : public MessageBase {
    char topic[10];
    uint8_t content[20];
} PublishContent;
```

- Hacer ejemplos con tarjeta SD (peso: 8). Primer acercamiento a este tipo de memoria externa para aprender a trabajar con ficheros y carpetas.
- Implementar el uso de más topics (peso: 20). Para añadir esta posibilidad, se decidió crear una clase `BrokerTopic`, siendo cada una un topic o tema con un listado de suscriptores y una cola de mensajes, y las funciones apropiadas para gestionar la lista y la cola. Se crea una clase `BrokerTopic` por cada mensaje (de publicación o suscripción) recibido con un nuevo topic en el broker, añadiendo soporte a temas. El Listado 4.7 muestra el pseudocódigo de la primera versión de esta clase, y 4.8 muestra la implementación de `BrokerTopic`.

Listado 4.7: Pseudocódigo de la primera definición de la clase BrokerTopic

```
Clase BrokerTopic:
Privado:
    char topic[10]
    vector<uint8_t[6]> subscribers //Direcciones MAC de los suscriptores
    QueueHandle_t messagesQueue //Cola que almacena mensajes para enviar

Público:
    Función constructora
    Función getTopic()
    Función getSubscribersAmount() //Cuenta la cantidad de suscriptores
    Función getSubscribers()
    Función subscribe(uint8_t *mac):
        Si no isSubscribed(mac):
            Registrar mac como par
            Agregar mac a subscribers
    Función isSubscribed(uint8_t *mac): //Comprueba si la dirección MAC ya
        ↪ estaba suscrita
        Por cada subscriber en subscribers:
            Si mac es igual que subscriber:
                Retornar true //Está suscrito
```

```
        Retornar false //No está suscrito
    Función sendToQueue(PublishContent *pubContent) //Enviar mensaje a la cola
        ↪ messagesQueue
    Función dispatchMessages(): //Enviar todos los mensajes de la cola
        ↪ messagesQueue a los suscriptores
        PublishContent message = Recibir una copia del primer mensaje de
            ↪ messagesQueue
        Por cada subscriber en subscribers:
            Registrar subscriber como par
            Enviar message a subscriber
```

Listado 4.8: Pseudocódigo de la segunda definición del broker

```
struct PayloadStruct: //Estructura de mensajes enviados
    int number

vector<BrokerTopic> topicsVector //Contiene los topics

//Cada vez que se reciba un mensaje, se llama a esta función
Función OnDataRecv(uint8_t *mac, uint8_t *incomingData):
    Message msgType = Primer byte de incomingData
    Según msgType:
        Si es un mensaje de suscripción:
            SubscribeAnnouncement *subAnnounce = Copia de los datos recibidos
            bool subscribed = false //Si la dirección MAC se suscribió en un topic
                ↪  existente
            Por cada topicObject en topicsVector:
                Si el topic de subAnnounce es el mismo que el de topicObject:
                    Si mac no está suscrita en el topicObject:
                        topicObject.subscribe(mac)
                    subscribed = true

            Si subscribed es false:
                BrokerTopic newTopic = Nuevo topic creado con el mismo topic que
                    ↪ el de subAnnounce
                newTopic.subscribe(mac)
                Agregar newTopic a topicsVector
```

```
23
24        Si es un mensaje de publicación:
25            PublishContent *pubContent = Copia de los datos recibidos
26            bool sent = false //Si la publicación se envió en un topic existente
27            Por cada topicObject en topicsVector:
28                Si el topic de pubContent es el mismo que el de topicObject:
29                    topicObject.sendToQueue(pubContent)
30                    sent = true
31
32            Si sent es false:
33                BrokerTopic newTopic = Nuevo topic creado con el mismo topic que
                      ↪ el de subAnnounce
34                topicObject.sendToQueue(pubContent)
35                Agregar newTopic a topicsVector
36
37  //Tarea encargada de enviar cada mensaje almacenado en los topics
38  Función DispatchMessagesTask():
39      Por siempre:
40          Por cada topicObject en topicsVector:
41              topicObject.dispatchMessage()
42          Esperar 1 segundo
43
44  //Tarea encargada de crear mensajes de prueba
45  Función ProduceMessageTask():
46      Por siempre:
47          PayloadStruct payload
48          payload.number = Número aleatorio
49          Message sendMessage //Nuevo mensaje
50          sendMessage.msgType = Definir mensaje como publicación
51          sendMessage.payload.subscribeAnnouncement.topic = "mock" //Asignar topic
                ↪ al payload SubscribeAnnouncement
52          sendMessage.payload.content = payload
53
54          bool queued = false //Si se pudo encolar el mensaje en un topic existente
55          Por cada topicObject en topicsVector:
56              Si el topic de sendMessage es el mismo que el de topicObject
```

```
            Enviar sendMessage a la cola de topicObject
            queued = true

    Si queued es false:
        BrokerTopic newTopic = Nuevo topic creado con el mismo topic que el de
            ↪ sendMessage
        Enviar sendMessage a la cola de topicObject
        Agregar newTopic a topicsVector

    Esperar 1 segundo

Función setup():
    Inicializar ESP-NOW
    Registrar la función OnDataRecv como callback al recibir un mensaje
    Inicializar la cola messagesQueue
    Crear tarea DispatchMessagesTask
    Crear tarea ProduceMessageTask
```

4.4. Iteración 3

Periodo: 09/03/2024 - 15/03/2024 (1 semana). Peso total: 57.

El objetivo principal de este sprint fue la implementación de paralelismo al procesar los tipos de mensajes y la compatibilidad con caracteres comodín en los topics.

Tareas:

- Mover funciones de publicar/suscribir de los ejemplos al código de la librería (peso: 8). Las funciones desarrolladas como ejemplos en las iteraciones 1 y 2 fueron movidas tras considerarse versiones funcionales adecuadas, quedando declaradas como en el Listado 4.9. Los códigos anteriores fueron modificados para hacer llamadas a la librería.

Listado 4.9: Pseudocódigo de la primera versión de los parámetros de funciones de publicar y suscribir definidos en la librería

```
Función publish(uint8_t *mac, char *topic, void *payload)
Función subscribe(uint8_t *mac, char *topic)
```

- Modificar callback del broker para crear tareas (peso: 20). Para aprovechar la capacidad del paralelismo, se implementaron tareas capaces de gestionar los mensajes de suscripción y publicación, que recorren los objetos `BrokerTopic` para determinar en cuál es más adecuado publicar o suscribir. En este primer acercamiento, cada vez que el broker recibe uno de estos mensajes, se crea una tarea. La modificación se llevó a cabo como se muestra en el Listado 4.10. Junto a esto, se eliminó la tarea `DispatchMessagesTask` del broker y se creó la función `publish(PublishContent pubContent)` en `BrokerTopic`, que envía directamente el mensaje de publicación a todos los suscriptores del objeto. Esto se debe a la nueva solución, en la que la carga del procesamiento de los suscriptores se realiza en cada tarea `PublishTask` creada.

Listado 4.10: Pseudocódigo de la modificación del broker para crear tareas según el mensaje recibido

```
1  struct SubscribeTaskParams: //Estructura para los parámetros de la tarea de
       ↪ suscripción
2      SubscribeAnnouncement *subAnnounce
3      uint8_t *mac
4
5  struct PublishTaskParams: //Estructura para los parámetros de la tarea de
       ↪ publicación
6      PublishContent *pubContent
7      uint8_t *mac
8
9  Función SubscribeTask(void *parameter):
10     SubscribeTaskParams params = parameter //Extraer los parámetros
11     SubscribeAnnouncement subAnnounce = params.subAnnounce
12     uint8_t mac = params.mac;
13
14     bool subscribed = false //Si la dirección MAC se suscribió en un topic
           ↪ existente
15     Por cada topicObject en topicsVector:
16         Si el topic de subAnnounce es el mismo que el de topicObject:
17             Si mac no está suscrita en el topicObject:
18                 topicObject.subscribe(mac)
19             subscribed = true
20
21     Si subscribed es false:
22         BrokerTopic newTopic = Nuevo topic creado con el mismo topic que el de
               ↪ subAnnounce
```

```
23        newTopic.subscribe(mac)
24        Agregar newTopic a topicsVector
25
26  Función PublishTask(void *parameter):
27     PublishTaskParams params = parameter //Extraer los parámetros
28     PublishContent pubContent = params.pubContent
29     uint8_t mac = params.mac;
30
31     bool sent = false //Si la publicación se envió en un topic existente
32     Por cada topicObject en topicsVector:
33        Si el topic de pubContent es el mismo que el de topicObject:
34           topicObject.sendToQueue(pubContent)
35           sent = true
36
37     Si sent es false:
38        BrokerTopic newTopic = Nuevo topic creado con el mismo topic que el de
             ↪ subAnnounce
39        topicObject.sendToQueue(pubContent)
40        Agregar newTopic a topicsVector
41
42  //Cada vez que se reciba un mensaje, se llama a esta función
43  Función OnDataRecv(uint8_t *mac, uint8_t *incomingData):
44     Message msgType = Primer byte de incomingData
45     Según msgType:
46        Si es un mensaje de suscripción:
47           SubscribeTaskParams subTaskParams
48           subTaskParams.subAnnounce = Copia de los datos recibidos
49           subTaskParams.mac = mac
50           Crear tarea SubscribeTask con parámetro subTaskParams
51        Si es un mensaje de publicación:
52           PublishTaskParams pubTaskParams
53           pubTaskParams.pubContent = Copia de los datos recibidos
54           pubTaskParams.mac = mac
55           Crear tarea PublishTask con parámetro pubTaskParams
```

- Añadir comprobador, agregador y eliminador de pares (peso: 13). ESP-NOW tiene un límite de

pares que se pueden agregar, por lo que es necesario implementar un comprobador de pares para no agregar pares repetidos. Con este objetivo, se creó la función `addPeer(uint8_t *mac)` en `BrokerTopic`, que es llamada antes de enviar un mensaje a los suscriptores. Además, se creó la función `removePeer(uint8_t *mac)` para eliminar un par registrado en la futura implementación de la desuscripción.

- Hacer ejemplos con formatos CSV y JSON en tarjeta SD (peso: 8). Crear ejemplos para serializar y deserializar objetos en ficheros contenidos en una tarjeta SD con el fin de aprender las capacidades de las librerías que lo permiten para implementar la persistencia de topics en futuras iteraciones.

- Añadir soporte a jerarquía de topics y caracteres comodín (peso: 8). La librería que se desarrolla en este proyecto se basa en MQTT, y está influenciada por sus capacidades, como la jerarquía y los caracteres comodín o wildcards `+` y `#`. Esto implica tener en cuenta el topic que inserta el usuario al publicar o suscribirse mediante dos funciones creadas en la librería. Definidas en el Listado 4.11, `pubTopicCheck` y `subTopicCheck` ambas arreglan el mal uso de los caracteres `/`, comprueban que el tamaño del topic no sea mayor del permitido y recorren todos los caracteres comprobando que sean ASCII y estén bien utilizados, como por ejemplo, la prohibición del publicador para usar wildcards. Son funciones llamadas antes del envío de los mensajes, es decir, desde los roles publicador y suscriptor/desuscriptor.

  En cuanto a la jerarquía de suscripción, se refiere a la suscripción a un tema como `cocina/+`, que significa la suscripción a publicaciones con temas como `cocina/nevera` o `cocina/grifo`. Esto se ha resuelto implementando una nueva estrategia para crear `BrokerTopics`. Primero, se sigue creando un `BrokerTopic` por cada suscripción, incluso si el topic tiene un wildcard. Para cada publicación, en cambio, se elimina esta capacidad, evitando un consumo de memoria por publicaciones sin receptores, que en este caso se descartarán. Una publicación a un topic pasa por la comprobación del topic con los `BrokerTopics` ya creados, y si es compatible o "publicable", envía mensajes a los suscriptores. Por ejemplo, una publicación con topic `cocina/grifo` es "publicable" para los suscriptores de `cocina/+`, pero, por el contrario, una publicación `baño/grifo` no lo es, y si no existen `BrokerTopics` compatibles se descarta el mensaje. La implementación se ha llevado a cabo agregando la función para comprobar si es "publicable" junto a un atributo que facilita dicha comprobación, como se muestra en el Listado 4.12.

Listado 4.11: Pseudocódigo de las funciones añadidas para comprobar los topics

```
1 #define MAXTOPICLENGTH 10 //Declarada la longitud máxima del topic
2
3 Función fixTopicAndCheckLength(char *topic):
4   Si topic es null:
5     Retornar error
```

```
  Elimina caracteres / añadidos al principio o al final
  Si topic es demasiado grande:
    Retornar error
    Retornar éxito

Función pubTopicCheck(char *topic):
  Si fixTopicAndCheckLength(topic) retorna error:
    Retornar error

  Por cada caracter en topic:
    Si caracter es '+' o es '#' o no es ASCII:
        Retornar error

  Retornar éxito

Función subTopicCheck(char *topic):
  Si fixTopicAndCheckLength(topic) retorna error:
    Retornar error

  Por cada caracter en topic:
    Si no es ASCII:
        Retornar error

    Si es '+' o es '#':
      Si está en una posición incorrecta:
        Retornar error

  Retornar éxito
```

Listado 4.12: Pseudocódigo de la comprobación de compatibilidad en BrokerTopic

```
Clase BrokerTopic:
Privado:
    bool hasWildcards //Si el topic contiene wildcards
    char topic[MAXTOPICLENGTH]
    ...

```

```
Público:
    Función constructora
    ...
    Función publish(PublishContent pubContent)
    Función isPublishable(char *publishTopic):
        Si publishTopic es igual a topic:
            Retornar true
        Sino si hasWildcards es false:
            Retornar false
        Sino:
            Recorre publishTopic y topic comprobando que los wildcards en topic
                ↪ sean compatibles con los niveles de publishTopic
            Retornar true si son compatibles, false si no
```

4.5. Iteración 4

Periodo: 16/03/2024 - 21/03/2024 (1 semana). Peso total: 39.

El objetivo principal de este sprint fue añadir soporte para la desuscripción y arreglar varios errores y vulnerabilidades.

Tareas:

- Probar publicador/suscriptor MQTT con Python (peso: 5). Al igual que en la iteración 0, pero esta vez con Python, trata de desarrollar un código para explorar las capacidades de MQTT. En concreto, en esta iteración se comprobó que si un usuario se suscribe a dos temas como `cocina/nevera` ↪ y `cocina/+`, y se publica un mensaje en `cocina/nevera`, cuántas veces recibe el mensaje el suscriptor. El resultado fue 1.
- Realizar pruebas de estrés o bullying a la solución actual (peso: 5). La solución desarrollada en la iteración 3 presentó una vulnerabilidad revisada al final del sprint anterior. Anteriormente, se creaban tareas cada vez que se recibía un mensaje, lo que significaba que un nodo malicioso que publicara o se suscribiera constantemente al broker podría realizar ataques de denegación de servicio, provocando sobrecargas en la memoria o tareas mal ejecutadas. La tarea consistió en desarrollar un publicador malicioso que enviara mensajes constantemente a un broker desplegado y modificado para que las tareas demoren más tiempo, generando tres tipos de errores:
    - El broker no podía crear tareas, posiblemente debido a que la memoria de la placa estaba saturada con demasiadas tareas encoladas.

- Error de “núcleo entró en pánico” y “carga prohibida”.
- Los mensajes no se trataban correctamente, el contenido era interpretado con caracteres erróneos.

- Arreglar vulnerabilidad DDOS (peso: 8). A partir de la prueba anterior, se vio necesario idear una nueva solución para gestionar los mensajes. En esta, el broker ejecuta constantemente una tarea por cada tipo de mensaje: una para publicar, otra para suscribir y otra para desuscribir. Estas tareas leen constantemente de una cola de mensajes, en las cuales los mensajes se insertan tras su recepción, como se muestra en el Listado 4.13.

Listado 4.13: Pseudocódigo de la modificación del broker para insertar mensajes recibidos en colas

```
1  //Cantidad de tareas ejecutadas por tipo de mensaje
2  #define SUBSCRIBETASKS 1
3  #define UNSUBSCRIBETASKS 1
4  #define PUBLISHTASKS 1
5
6  struct SubscribeTaskParams: //Estructura para los parámetros de la tarea de
       ↪ suscripción
7      SubscribeAnnouncement *subAnnounce
8      uint8_t *mac
9
10 struct PublishTaskParams: //Estructura para los parámetros de la tarea de
       ↪ publicación
11     PublishContent *pubContent
12     uint8_t *mac
13
14 vector<BrokerTopic> topicsVector //Contiene los topics
15
16 //Colas de mensajes
17 QueueHandle_t subMsgQueue;
18 QueueHandle_t unsubMsgQueue;
19 QueueHandle_t pubMsgQueue;
20
21 //Tarea encargada de procesar mensajes de suscripción
22 Función SubscribeTask(void *parameter):
23     SubscribeTaskParams params
24     Por siempre:
```

```
25        Recibir de la cola subMsgQueue y copiar en params
26        //Extraer los parámetros
27        SubscribeAnnouncement subAnnounce = params.subAnnounce
28        uint8_t mac = params.mac;
29
30        bool subscribed = false //Si la dirección MAC se suscribió en un topic
              ↪ existente
31        Por cada topicObject en topicsVector:
32            Si el topic de subAnnounce es el mismo que el de topicObject:
33                Si mac no está suscrita en el topicObject:
34                    topicObject.subscribe(mac)
35                subscribed = true
36                Salir del bucle
37
38        Si subscribed es false:
39            BrokerTopic newTopic = Nuevo topic creado con el mismo topic que el de
                  ↪ subAnnounce
40            newTopic.subscribe(mac)
41            Agregar newTopic a topicsVector
42
43        Esperar 1 segundo
44
45  //Tarea encargada de procesar mensajes de publicación
46  Función PublishTask(void *parameter):
47      PublishTaskParams params = parameter
48      Por siempre:
49          Recibir de la cola pubMsgQueue y copiar en params
50          //Extraer los parámetros
51          PublishContent pubContent = params.pubContent
52          uint8_t mac = params.mac;
53
54          Por cada topicObject en topicsVector:
55              Si topicObject.isPublishable(pubContent.topic): //Si el topic de la
                    ↪ publicación es compatible con el del BrokerTopic
56                  topicObject.publish(pubContent)
57
```

```
58        Esperar 1 segundo
59
60  //Cada vez que se reciba un mensaje, se llama a esta función
61  Función OnDataRecv(uint8_t *mac, uint8_t *incomingData):
62     Message msgType = Primer byte de incomingData
63     Según msgType:
64        Si es un mensaje de suscripción:
65           SubscribeTaskParams subTaskParams
66           subTaskParams.subAnnounce = Copia de los datos recibidos
67           subTaskParams.mac = mac
68           Enviar subTaskParams a la cola subMsgQueue
69        Si es un mensaje de publicación:
70           PublishTaskParams pubTaskParams
71           pubTaskParams.pubContent = Copia de los datos recibidos
72           pubTaskParams.mac = mac
73           Enviar pubTaskParams a la cola pubMsgQueue
74
75  Función setup():
76     Inicializar ESP-NOW
77     Registrar la función OnDataRecv como callback al recibir un mensaje
78     Inicializar la cola subMsgQueue
79     Inicializar la cola unsubMsgQueue
80     Inicializar la cola pubMsgQueue
81     Crear tareas SubscribeTask según SUBSCRIBETASKS
82     Crear tareas PublishTask según PUBLISHTASKS
```

- Implementar la desuscripción (peso: 13). Crear las funciones de desuscripción en la librería y en `BrokerTopic`, y la tarea y cola necesarias en el broker para gestionar este tipo de mensajes recibidos. La desuscripción elimina la dirección MAC de la lista de suscriptores de un topic, en este caso se realiza aprovechando las funciones integradas en C++ para buscar en un vector y eliminar el elemento, junto con la función `removePeer` creada anteriormente. En adición a eliminar de la lista de suscriptores, para evitar malgasto de memoria, si al desuscribir el `BrokerTopic` se queda sin suscriptores, se elimina. La implementación se muestra en los Listados 4.14, 4.15 y 4.16.

Listado 4.14: Pseudocódigo de la función de desuscribir definida en la librería

```
1  Función unsubscribe(uint8_t *mac, char *topic):
2    Inicializar ESP-NOW
```

```
  Registrar el broker como par
  Si subTopicCheck(topic) retorna error:
    Retornar error

  UnsubscribeAnnouncement unsubMsg //Nuevo mensaje
  unsubMsg.type = Definir mensaje como desuscripción
  unsubMsg.topic = topic
  Enviar unsubMsg al broker
```

Listado 4.15: Pseudocódigo de la adición de la desuscripción al broker

```
//Cantidad de tareas ejecutadas por tipo de mensaje
#define SUBSCRIBETASKS 1
#define UNSUBSCRIBETASKS 1
#define PUBLISHTASKS 1

struct UnsubscribeTaskParams: //Estructura para los parámetros de la tarea de
    ↪ desuscripción
    UnsubscribeAnnouncement *unsubAnnounce
    uint8_t *mac

vector<BrokerTopic> topicsVector //Contiene los topics

//Colas de mensajes
QueueHandle_t unsubMsgQueue;

//Tarea encargada de procesar mensajes de desuscripción
Función UnsubscribeTask(void *parameter):
    UnsubscribeTaskParams params
    Por siempre:
        Recibir de la cola unsubMsgQueue y copiar en params
        //Extraer los parámetros
        UnsubscribeAnnouncement unsubAnnounce = params.subAnnounce
        uint8_t mac = params.mac;

        Por cada it en topicsVector:
            Si el topic de unsubAnnounce es el mismo que el de it:
```

```
            Si mac está suscrita en el it:
                it.unsubscribe(mac)
                Si it tiene 0 suscriptores:
                    Eliminar it de topicsVector
            Salir del bucle

        Esperar 1 segundo

//Cada vez que se reciba un mensaje, se llama a esta función
Función OnDataRecv(uint8_t *mac, uint8_t *incomingData):
    Message msgType = Primer byte de incomingData
    Según msgType:
        ...
        Si es un mensaje de desuscripción:
            UnsubscribeTaskParams unsubTaskParams
            unsubTaskParams.unsubAnnounce = Copia de los datos recibidos
            unsubTaskParams.mac = mac
            Enviar unsubTaskParams a la cola unsubMsgQueue

Función setup():
    Inicializar ESP-NOW
    Registrar la función OnDataRecv como callback al recibir un mensaje
    Inicializar la cola subMsgQueue
    Inicializar la cola unsubMsgQueue
    Inicializar la cola pubMsgQueue
    Crear tareas SubscribeTask según SUBSCRIBETASKS
    Crear tareas UnsubscribeTask según UNSUBSCRIBETASKS
    Crear tareas PublishTask según PUBLISHTASKS
```

Listado 4.16: Pseudocódigo de la adición de la desuscripción a BrokerTopic

```
Clase BrokerTopic:
Privado:
    ...
    char topic[MAXTOPICLENGTH]
  vector<uint8_t[6]> subscribers //Direcciones MAC de los suscriptores

```

```
Público:
   Función constructora
   ...
  Función removePeer(uint8_t *mac)
  Función isSubscribed(uint8_t *mac)
  Función unsubscribe(uint8_t *mac):
    Recorrer subscribers con un valor igual a mac
    Si lo encuentra:
      Borrar de subscribers
      removePeer(mac)
      Retornar true
    Retornar false //Si no se encontró
```

- Arreglar el envío de mensajes duplicados al suscribirse múltiples veces al mismo topic (peso: 8). Añadir un comprobador de los destinatarios a quienes ya se les envió el mensaje. Esto se realizó creando, al recibir un mensaje de publicación, un vector en el broker, `alreadySentMacs`, el cual es compartido por todos los `BrokerTopics` compatibles. Este contiene todos los destinatarios. Primero se comprueba si un suscriptor ya está en esta lista y, si no es el caso, se le envía el mensaje y se añade a la lista. El uso de este vector desde `BrokerTopic` se observa en el Listado 4.17.

Listado 4.17: Pseudocódigo de la nueva comprobación para evitar envíos duplicados en BrokerTopic

```
Clase BrokerTopic:
Privado:
  ...
  vector<uint8_t[6]> subscribers //Direcciones MAC de los suscriptores

Público:
   Función constructora
   ...
  Función addPeer(uint8_t *mac)
  Función publish(PublishContent pubContent, vector<uint8_t[6]> alreadySentMacs)
      ↪ :
    Por cada subscriber en subscribers:
      bool alreadySent = false //Si ya se envió el mensaje
      Por cada sentMac en alreadySentMacs:
        Si sentMac es igual a subscriber: //Comprueba si está en la lista de "ya
            ↪ enviados"
```

```
        alreadySent = true
        Salir del bucle

    Si alreadySent es false:
      addPeer(subscriber)
      Enviar pubContent a subscriber
      Agregar subscriber a alreadySentMacs
```

4.6. Iteración 5

Periodo: 16/03/2024 - 21/03/2024 (1 semana). Peso total: 57.

El objetivo principal de este sprint fue la mejora de la calidad del código, la corrección de errores y el primer paso para redactar la memoria.

Tareas:

- Encontrar (peso: 8) y arreglar (peso: 20) la recepción de mensajes corruptos o duplicados. Tras la revisión del sprint anterior, se encontró el fallo de que los mensajes de suscripción y desuscripción se duplicaban en las colas. El primer paso para resolverlo fue aislar el error, creando un broker dedicado únicamente a atender peticiones de suscripción. Luego de realizar pruebas modificando este código, se descubrió que el fallo se encontraba en la mala gestión de la memoria con las variables que almacenan los parámetros de los mensajes y se insertan en las colas, por lo que se solucionó haciendo un mejor uso de los punteros y de las liberaciones de memoria.
- Clonar la plantilla de la memoria del TFG (peso: 5).
- Renombrar variables, añadir comentarios y arreglar el formato del código (peso: 8). Junto con eliminar código sin usar, comentarios innecesarios e implementar EditorConfig[1] y cumplir con los criterios de calidad, con el fin de mejorar la calidad del código.
- Crear ideas para nombrar la herramienta (peso: 8). En gran parte del desarrollo, la librería no ha tenido un nombre definido, siendo este paso es esencial para poder publicarla y darle una identidad.
- Probar y familiarizarse con la plantilla de la memoria (peso: 8). Aprender a utilizar la plantilla para empezar a redactar en posteriores iteraciones.

4.7. Resultados del TFG

Este apartado debe explicar cómo el empleo de la metodología permite satisfacer tanto el objetivo principal como los específicos planteados en el TFG así como los requisitos exigidos (según exposición en capítulo

[1] https://editorconfig.org/

Objetivos).

Capítulo 5

Conclusiones

En este capítulo se debe incluir el juicio crítico y discusión sobre los resultados obtenidos. Si es pertinente deberá incluir información sobre trabajos derivados como publicaciones o ponencias, así como trabajos futuros, solo si estos están planificados en el momento en que se redacta el texto. Incluirá obligatoriamente la justificación de las competencias de la tecnología específica cursada por el estudiante que se han adquirido durante el desarrollo del TFG

5.1. Revisión de los Objetivos

En esta sección se deberá revisar en qué grado se han completado los objetivos fijados al principio del proyecto. Se deberá también indicar las posibles desviaciones de los objetivos fijados, así como de la planificación, y tratar de justificar tales desviaciones.

5.2. Presupuesto

Si el TFG consiste en el desarrollo e implementación de un prototipo, la memoria debe incluir el coste del prototipo considerando tanto el hardware como los recursos humanos necesarios para su desarrollo.

Cuando se tiene en cuenta la puesta en marcha de un proyecto de ingeniería, la planificación y presupuesto que se realizan de modo previo a su ejecución son críticos para gestionar los recursos que permitan alcanzar los objetivos de calidad, temporales y económicos previstos para el proyecto. Es muy importante que todas las justificaciones aportadas se sustenten no solo en juicios de valor sino en evidencias tangibles como: historiales de actividad, repositorios de código y documentación, porciones de código, trazas de ejecución, capturas de pantalla, demos, etc.

5.3. Competencias Específicas de Intensificación Adquiridas y/o Reforzadas

Se deberán listar aquellas competencias de la intensificación que hayan sido adquiridas y/o reforzadas con el desarrollo de este TFG, incluyendo su justificación.

Bibliografía

[1] Agile scrum for Web Development | Neon Rain Interactive. https://www.neonrain.com/blog/agile-scrum-for-web-development/.

[2] Build software better, together. https://education.github.com/discount_requests/application.

[3] 2024. C++ programming with Visual Studio Code. https://code.visualstudio.com/docs/languages/cpp.

[4] GitHub | Trello Power-Ups. https://trello.com/power-ups/55a5d916446f517774210004/github.

[5] 2024. GitHub Copilot overview. https://code.visualstudio.com/docs/copilot/overview.

[6] 2023. Los idiomas más hablados en el mundo en 2023 | Statista. https://es.statista.com/estadisticas/635631/los-idiomas-mas-hablados-en-el-mundo/.

[7] OpenUP - UTM. https://www.utm.mx/~caff/doc/OpenUPWeb/.

[8] ¿Qué es y para qué sirve una metodología de desarrollo de software? - desarrollodesoftware. https://desarrollodesoftware.dev/metodologia.

[9] Smart Fields | Trello Power-Ups. https://trello.com/power-ups/5e2212c3ba57415ef2ef9352/smart-fields.

[10] Tablero Kanban: Qué es, cómo hacerlo y ejemplos | Miro. https://miro.com/es/agile/que-es-tablero-kanban/.

[11] Tecnología LORA y LoRAWAN - CatSensors. https://www.catsensors.com/es/lorawan/tecnologia-lora-y-lorawan.

[12] Agile Alliance Manifiesto por el Desarrollo Ágil de Software. https://agilemanifesto.org/iso/es/manifesto.html.

[13] Agile Alliance Principios del Manifiesto Ágil. https://agilemanifesto.org/iso/es/principles.html.

[14] Atlassian Flujo de trabajo de Gitflow | Atlassian Git Tutorial. https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow.

[15] Atlassian Qué es el control de versiones | Atlassian Git Tutorial. https://www.atlassian.com/es/git/tutorials/what-is-version-control.

[16] Bond, A. Better Comments - Visual Studio marketplace. https://marketplace.visualstudio.com/items?itemName=aaron-bond.better-comments.

[17] BQ Educación Componentes zum Kit Advanced. https://tienda.bq.com/products/componentes-zum-kit-advanced?variant=37589957935292.

[18] BQ Educación Zum Kit Advanced: kit de robótica para niños | BQ Educación. https://educacion.bq.com/zum-kit-advanced/.

[19] Centro Gallego de Robótica Educativa Potenciómetro para proyectos. - Centro Gallego de Robótica Educativa. https://centroderobotica.com/producto/potenciometro-para-proyectos/.

[20] De Negocios Feda, E. 2019. GESTIÓN ÁGIL vs GESTIÓN TRADICIONAL DE PROYECTOS ¿CÓMO ELEGIR? - Escuela de Negocios FEDA. https://www.escueladenegociosfeda.com/blog/50-la-huella-de-nuestros-docentes/471-gestion-agil-vs-gestion-tradicional-de-proyectos-como-elegir.

[21] Driessen, V. 2010. A successful Git branching model. https://nvie.com/posts/a-successful-git-branching-model/.

[22] Editorial Team - everything RF 2022. What is Thread? - everything RF. https://www.everythingrf.com/community/what-is-thread.

[23] Equipo editorial de IONOS 2019. Extreme Programming: desarrollo ágil llevado al extremo. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/extreme-programming/.

[24] Equipo editorial de IONOS 2020. Kanban. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-kanban/.

[25] Equipo editorial de IONOS 2019. Scrum: gestión de proyectos con un enfoque flexible. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/scrum/.

[26] Eridani, D. et al. 2021. Comparative Performance Study of ESP-NOW, Wi-Fi, Bluetooth Protocols based on Range, Transmission Speed, Latency, Energy Usage and Barrier Resistance. *2021 International Seminar on Application for Technology of Information and Communication (iSemantic)* (Sep. 2021), 322–328.

[27] Esei 2023. Agile Methodology vs. Traditional Project Management. https://www.eseibusinessschool.com/es/agile-vs-traditional-project-management/.

[28] Expósito, D.B. AMD Ryzen 7 5700U: características, especificaciones y precios. https://www.geektopia.es/es/product/amd/ryzen-7-5700u/.

[29] GitHub GitHub Copilot - Visual Studio Marketplace. https://marketplace.visualstudio.com/items?itemName=GitHub.copilot.

[30] Gruntfuggly Todo Tree - Visual Studio Marketplace. https://marketplace.visualstudio.com/items?itemName=Gruntfuggly.todo-tree.

[31] Jacobs, M. et al. 2022. What is Agile development? - Azure DevOps. https://learn.microsoft.com/en-us/devops/plan/what-is-agile-development.

[32] Jacobs, M. et al. 2022. What is Agile? - Azure DevOps. https://learn.microsoft.com/en-us/devops/plan/what-is-agile.

[33] Liu, W. TODO Highlight - Visual Studio Marketplace. https://marketplace.visualstudio.com/items?itemName=wayou.vscode-todo-highlight.

[34] Manole, L. 2023. Edge, Fog, and Cloud Computing: What's the Difference? https://blog.isa.org/edge-fog-and-cloud-computing-whats-the-difference.

[35] Meno, A. 2021. Metodologías de desarrollo de software (tradicionales vs ágiles) – Tecnitium. https://tecnitium.com/metodologias-de-desarrollo-de-software/.

[36] Microsoft C/C++ Extension Pack - Visual Studio Marketplace. https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools-extension-pack.

[37] Mishra, B. 2018. TMCAS: An MQTT based Collision Avoidance System for Railway networks. (Jul. 2018), 1–6.

[38] Nabheetscn 2022. The Magic 80 Char limit which used to exist for developers. https://community.sap.com/t5/technology-blogs-by-members/the-magic-80-char-limit-which-used-to-exist-for-developers/ba-p/13530175.

[39] PlatformIO PlatformIO IDE - Visual Studio marketplace. https://marketplace.visualstudio.com/items?itemName=platformio.platformio-ide.

[40] Preston-Werner, T. Semantic Versioning 2.0.0. https://semver.org/spec/v2.0.0.html.

[41] Rcarrillo 2023. Qué es LoRa, cómo funciona y características principales. https://www.vencoel.com/que-es-lora-como-funciona-y-caracteristicas-principales/.

[42] Rehkopf, M. ¿Qué es un tablero kanban? | Atlassian. https://www.atlassian.com/es/agile/kanban/boards.

[43] Rodal, E. 2021. Diferencias entre fog computing, edge computing y cloud computing. https://www.podcastindustria40.com/fog-computing/.

[44] Schlosser, C. Doxygen Documentation Generator - Visual Studio Marketplace. https://marketplace.visualstudio.com/items?itemName=cschlosser.doxdocgen.

[45] Schwaber, K. and Sutherland, J. 2020. *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game*. https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf.

[46] Street Side Software Code Spell Checker - Visual Studio Marketplace. https://marketplace.visualstudio.com/items?itemName=streetsidesoftware.code-spell-checker.

[47] TecnoDigital 2024. Metodologías de desarrollo de software clásicas: enfoques tradicionales. https://informatecdigital.com/software/metodologias-de-desarrollo-de-software-clasicas/.

[48] Urazayev, D. et al. 2023. Indoor Performance Evaluation of ESP-NOW. (May 2023), 1–6.

[49] Wen, R. 2023. From Git-flow to GitHub-flow. https://blog.kinto-technologies.com/posts/2023-03-07-From-Git-flow-to-GitHub-flow/.

[50] Wikipedia contributors 2024. Agile software development. https://en.wikipedia.org/wiki/Agile_software_development.

[51] Wikipedia contributors 2024. Thread (network protocol). https://en.wikipedia.org/wiki/Thread_(network_protocol).

[52] Wikipedia contributors 2024. Zigbee. https://en.wikipedia.org/wiki/Zigbee.

Anexo A. Uso de la Plantilla

A continuación se presenta la estructura y utilización de esta plantilla.

A.1. Configuración

La configuración de la plantilla se realiza a través del fichero *config.yaml*. Este es un fichero *yaml*, cuyos campos se describen en la Tabla A.1.

Tabla A.1: Valores del fichero *config.yaml*

clave	valor
Cite	Citas bibliográficas a incluir en la bibliografía no referenciadas en el texto
Cotutor	Nombre y apellidos del co-tutor académico
Csl	Fichero csl con el formato de las referencias
Department	Departamento del tutor académico
Language	Lenguaje de la plantilla [english|spanish]
Month	Mes de defensa del TFG
Name	Nombre del autor del TFG
Technology	Tecnología específica
Title	Título del TFG
Tutor	Nombre del tutor académico
Year	Año de defensa del TFG

Un ejemplo de configuración de este fichero se muestra en el Listado A.1.

Listado A.1: Ejemplo de fichero de configuración config.yaml

```
1 Cite: @Gutwin2010GoneBut, @Rekimoto1997PickDrop
2 Cotutor: John Deere
```

```
3  Csl: input/resources/csl/acm-sig-proceedings.csl
4  Department: Ciencias de la Computación
5  Language: spanish
6  Month: Agosto
7  Name: Johny Anston
8  Technology: Sistemas de Información
9  Title: Una Aplicación para Resolver Problemas Genéricos
10 Tutor: Adam Smith
11 Year: 2023
```

A.2. Estructura de Directorios

La plantilla tiene la estructura mostrada en la Figura A.1.

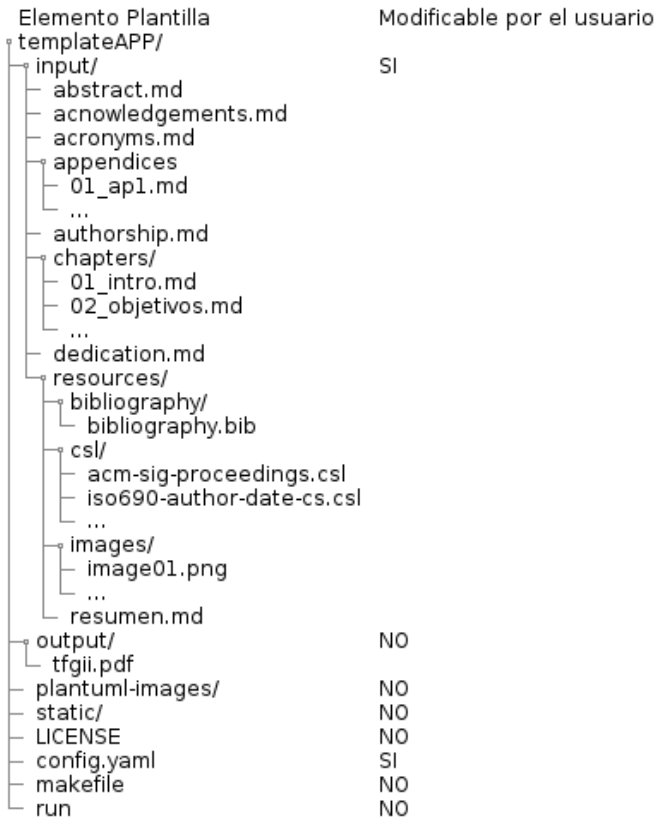

Figura A.1: Estructura de directorios de la plantilla del TFG

El documento generado por la plantilla es **output/tfgii.pdf**. Este documento se sobreescribe cada vez que se compila la plantilla. Es este documento, compilar se refiere a procesar los elementos de la plantilla para

generar el documento *pdf*.

Las carpetas y ficheros modificables por el usuario y que contienen la información propia del TFG son las siguientes (marcadas como SI en la figura A.1):

- input
- config.yaml

El resto de ficheros y directorios no deben de ser modificados por el usuario para el correcto funcionamiento de la plantilla.

A.2.1. Carpeta input

La carpeta *input* contiene los ficheros que corresponden con las partes del TFG.

En primer lugar están los ficheros correspondientes al **abstract**, **acknowledgements**, **acronyms**, **authorship** y **dedication**. En estos, a excepción del **acronyms**, sólo hay que introducir el texto correspondiente.

El fichero **acronyms** tiene un formato yaml. En el, hay que introducir los pares *acrónimo* y *valor* separados por ":" y un espacio. Un ejemplo sería el del Listado A.2.

Listado A.2: Ejemplo de definición de acrónimos

```
1 HTML: Lenguaje de marcas de hypertexto
2 HCI: Human Computer Interaction
```

También se encuentran en el directorio *input* los directorios **appendices** y **chapters**. Contienen, respectivamente, los apéndices y los capítulos del *TFG*, que corresponden con ficheros en *markdown*. En estos ficheros hay que tener en cuenta lo siguiente:

- El fichero debe comenzar con un encabezado de primer nivel
- Los ficheros se ordenan en el documento según su orden alfabético
- Sólo se tienen en cuenta los ficheros que comienzan con dos dígitos numéricos, p.ej. 01
- Si un fichero no comienza con dos dígitos numéricos, es ignorado y no se incluye a la hora de generar el documento

A.2.2. Carpeta resources

El directorio *resources* contiene recursos de utilidad para la generación del documento, como imágenes o bibliografía.

La bibliografía se recoge en el fichero **bibliography/bibliography.bib**. Es un fichero en formato *bibtex* [**bibtex?**] que contiene los elementos bibliográficos. Por ejemplo, en el Listado A.3 se encuentra definido

el elementos bibliográfico *bibtex*.

Listado A.3: Ejemplo de definición de elemento bibliográfico

```
@misc{bibtex,
  title = {BibTeX Format Description},
  howpublished = {\url{https://www.bibtex.org/Format/}},
  note = {Accessed: 2023-09-28}
}
```

El directorio **csl** contiene ficheros con diferentes formatos de bibliografía. Finalmente, el directorio **images** contiene las imágenes a utilizar en el documento.

A.3. Elementos del Documento

Si bien la plantilla permite utilizar cualquier elemento de *Markdown* y *pandoc*, hay una serie de elementos que pueden ser de especial utilidad de cara a la realización de un TFG:

- Referencias bibliográficas
- Figuras
- Tablas
- Listados de código
- Notas al pie de página
- Acrónimos
- PlantUML
- Referencias dentro del Documento

A.3.1. Citas Bibliográficas

La plantilla utiliza la bibliografía que se recoge en el fichero correspondiente. Para incluir una cita a un elemento bibliográfico, hay que añadir en el documento su referencia precedida del símbolo @. Por ejemplo, para añadir la cita al elemento biliográfico *bibtex* se haría de la forma indicada en el Listado A.4.

Listado A.4: Ejemplo de cita de elemento bibliográfico

```
Es un fichero en formato *bibtex* @bibtex.
```

A.3.2. Figuras

Para incluir figuras lo hacemos añadiendo una imagen en la cual especificamos el elemento *label*, que sirve para referenciarla. Por ejemplo, el código del Listaqdo A.5 produce como resultado la Figura A.2.

Listado A.5: Ejemplo de definición de una figura

```
1 ![Logo de HTML5\label{anexo:ejemploFiguraResultado}](HTML5_sticker.png){width
    ↪ =50%}
```

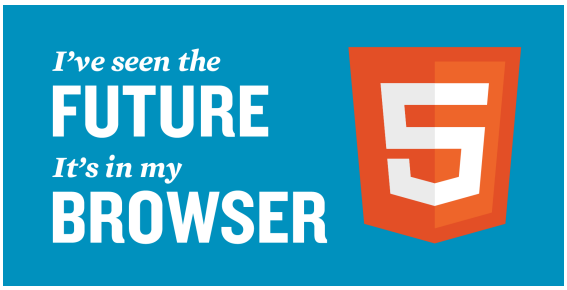

Figura A.2: Logo de HTML5

A.3.3. Tablas

Para definir una tabla se utiliza la sintaxis mostrada en el Listado de Código A.6, cuyo resultado se muestra en la Tabla A.2.

Listado A.6: Ejemplo de definición de una tabla

```
1 | Right | Left | Default | Center |
2 |------:|:-----|---------|:------:|
3 | 12 | 12 | 12 | 12 |
4 | 123 | 123 | 123 | 123 |
5 | 1 | 1 | 1 | 1 |
```

Tabla A.2: Tabla de ejemplo

Right	Left	Default	Center
12	12	12	12
123	123	123	123
1	1	1	1

A.3.4. Notas al Pie de Página

Para la definición de notas al pie de página se utiliza el código mostrado en el Listado A.7, cuyo resultado se muestra al pie de esta página[1].

[1] Ejemplo de nota al pie de página.

Listado A.7: Ejemplo de definición de una nota a pie de página

```
cuyo resultado se muestra al pie de esta página[^anexo:ejemploNotaPie].

[^anexo:ejemploNotaPie]: Ejemplo de nota al pie de página.
```

A.3.5. Acrónimos

Para hacer referencia a un acrónimo, por ejemplo HTML, hay que utilizar el código mostrado en el Listado A.8. Previamente, el acrónimo debe de haber sido definido, tal y como se muestra en el Listado A.2.

Listado A.8: Ejemplo de definición de referencia a un acrónimo

```
Para hacer referencia a un acrónimo, por ejemplo [HTML](#HTML), hay ...
```

A.3.6. PlantUML

Esta plantilla permite la inserción de diagramas en *PlantUML* [**plantuml?**]. En el Listado A.9 se muestra un ejemplo básico de un diagrama de objetos en *PlantUML*, cuyo resultado se muestra en la Figura A.3.

Listado A.9: Ejemplo de código plantuml básico

```
@startuml
skinparam dpi 300
object Object01
object Object02
object Object03
object Object04
object Object05
object Object06
object Object07
object Object08

Object01 <|-- Object02
Object03 *-- Object04
Object05 o-- "4" Object06
Object07 .. Object08 : some labels
@enduml
```

También se puede dar formato, colores y formas, a los diagramas generados con PlantUML, tal y como se muestra en el Listado A.10. El resultado se muestra en la Figura A.4.

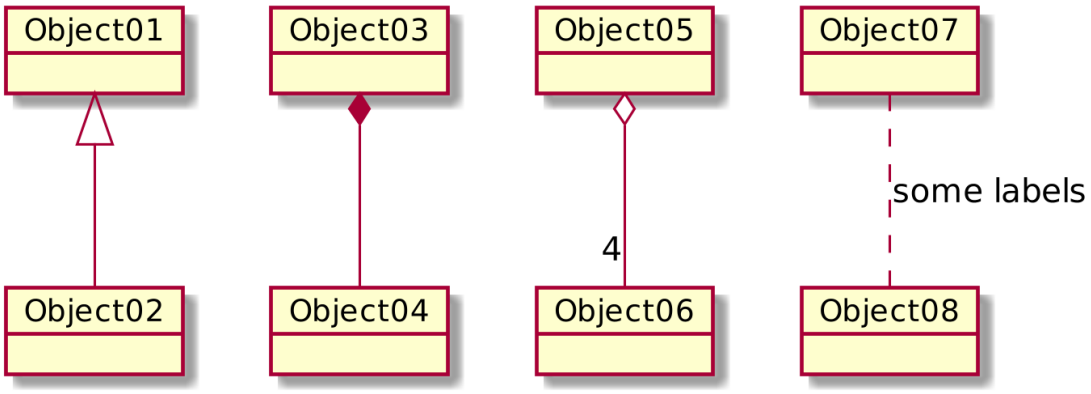

Figura A.3: Ejemplo de plantuml básico

Listado A.10: Ejemplo de código plantuml con formato

```
@startmindmap
skinparam dpi 300
<style>
mindmapDiagram {
  node {
    BackgroundColor white
    FontColor #B30033
  }
  :depth(1) {
    BackgroundColor white
    FontColor #B30033
  }
  .uclm {
    BackgroundColor white
    FontColor #B30033
  }
  .active {
    BackgroundColor #B30033
    FontColor white
  }
}
</style>
* Requisitos
** Requisitos de usuario <<active>>
```

```
*** Requisitos de dominio <<active>>
*** Requisitos de negocio <<active>>
*** Requisitos de usuario final <<active>>
** Requisitos de sistema
** Requisitos de SW/HW

@endmindmap
```

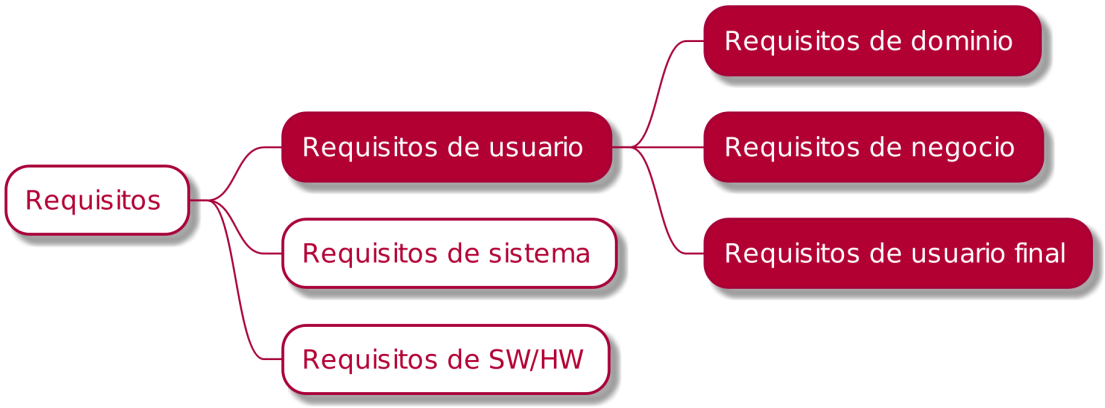

Figura A.4: Ejemplo de plantuml con formato

A.3.7. Referencias Dentro del Documento

Para hacer una referencia a un capítulo dentro del documento se indica entre llaves el texto del enlace, seguido entre paréntesis del nombre del capítulo precedido del carácter *'#'*. El nombre del capítulo se indica en minúsculas y se sustituyen los espacios por el carácter *'-'*. Por ejemplo, si se quiere referenciar al capítulo *"Estructura de Directorios"* se utilizará el siguiente código mostrado en la Figura A.11.

Listado A.11: Referencia dentro del documento

```
Referencia a la [estructura de directorios](#estructura-de-directorios)
```

A.4. Creación del Documento

La creación del documento se puede realizar utilizando un contenedor en *Docker* o de forma nativa en *Linux*[2]. La Figura A.5 muestra la estructura del directorio raíz de la plantilla.

A continuación, se explica cómo funciona cada una de estas aproximaciones, así como sus requisitos.

[2]Se ha probado que funciona correctamente en la distribución *Ubuntu 22.04.1*.

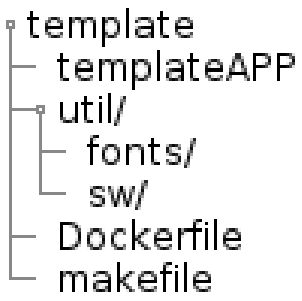

Figura A.5: Estructura de directorios raíz de la plantilla

A.4.1. Ubuntu

Ejecutar 'make' en el directorio raíz de la plantilla.

Hay que tener instalado el siguiente software[3]:

- texlive-latex-base
- texlive-lang-spanish
- texlive-xetex
- make
- pandoc-plantuml-filter

También hay que instalar las fuentes adecuadas. Para ello crear el directorio (si no existe), copiar las fuentes y actualizarlas según el código de la Figura A.12.

Listado A.12: Instalación de las fuentes para el documento

```
$ sudo mkdir -p /usr/share/fonts/truetype/msttcorefonts/
$ sudo cp util/fonts/* /usr/share/fonts/truetype/msttcorefonts/
$ sudo fc-cache -f -v
$ sudo texhash
```

Finalmente, hay que instalar pandoc, pero en su versión 2.19.2-1. Para ello, ejecutar la orden de la Figura A.13 en el directorio raíz de la plantilla. Si se utiliza cualquier otra versión es muy probable que se tengan problemas, por ejemplo a la hora de generar imágenes con *plantuml*.

Listado A.13: Instalación de Pandoc en su versión 2.19.2-1

```
$ sudo dpkg -i util/sw/pandoc-2.19.2-1-amd64.deb
```

[3]Para instalar el software en Ubuntu hay que ejecutar *apt install nombre-paquete*

A.4.2. Docker

Ejecutar ‘make docker’ en el directorio raíz de la plantilla.

En el caso de utilizar Docker, el único prerequisito es tenerlo instalado[4].

[4] https://docs.docker.com/engine/install/ubuntu/

Anexo B. Título del Anexo II

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh

sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

B.1. Una sección

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetuer a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consectetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a nulla. Curabitur tristique arcu

eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus.