UNIVERSITAT POLITÈCNICA DE VALÈNCIA

TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

DESARROLLO DE UNA APLICACIÓN IoT PARA EL ENVÍO DE IMÁGENES MEDIANTE EL PROTOCOLO MQTT

Francisco Mahedero Biot

Tutor: Antonio León Fernández

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2019-20

Valencia, 2 de Julio de 2020

Resumen

Actualmente, el concepto de IoT (Internet de las Cosas) cobra una gran importancia debido a la sociedad de la información en la que vivimos. El uso de los distintos dispositivos inteligentes que existen actualmente, permiten crear aplicaciones de utilidad en nuestra vida.

En este trabajo se utiliza el protocolo MQTT para la supervisión de un entorno como puede ser un campus, edificio etc. Se utilizan varios ESP32-CAM como cámaras y se envían las imágenes mediante el protocolo mencionado de forma segura. En esta solución se ofrece al usuario una aplicación web como interfaz gráfica para la visualización de las imágenes.

Resum

Actualment, el concepte de IoT (Internet de les coses) cobra una gran importància degut a la societat de la informació en la que vivim. L'ús dels distints dispositius intel·ligents que existeixen actualment, permeten crear aplicacions d'utilitat en la nostra vida.

En aquest treball s'utilitza el protocol MQTT per a la supervisió d'un entorn com poden ser un campus, edifici etc. S'utilitzen diversos ESP32-CAM com a càmeres i s'envíen les imatges mitjançant el protocol esmenat de forma segura. En aquesta solució s'oferix a l'usuari una aplicació web com interfície gràfica per a la visualització de les imatges.

Abstract

Nowadays, the IoT concept (Internet of things) have a great importance due to the information society where we live. The use of the different intelligent devices that exist nowadays, let to create utility applications in our lives.

In this project, the protocol MQTT is used to monitor an environment such as a campus, a building etc. Various ESP32-CAM are used as cameras and the pictures are sent by the protocol named above in a secure way. In this solution an application web is offered to the user as a graphic interface to visualize the pictures.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

Actualmente vivimos en una sociedad completamentamente conectada, donde los datos, la información, instantaneidad y otros factores se convierten en algo esencial en nuestro día a día. Esta forma de vida es posible gracias a los grandes avances que se han hecho en el campo de las TIC y que ha permitido el gran avance de IoT.

IoT se podría definir como la red de dispositivos agrupados e interconectados en la que son visibles y tienen la posibilidad de interactuar entre sí. Así pues, los objetos físicos pueden compartir y recopilar datos sin la necesidad de la intervención humana, es decir, es una conexión M2M.

Las aplicaciones y posibilidades son muy extensas, las cuales nos permiten mejorar tanto la vida cotidiana como los entornos empresariales. Entre los diferentes campos de aplicación de IoT encontramos el escenario de la domótica, donde se pueden ver ya dispositivos controlados por voz a los que se solicitan que reproduzcan canciones, encender luces o un sinfín de posibilidades. También se usa en aplicaciones industriales donde los dispositivos IoT permiten recabar información sobre los diferentes procesos que se están llevando a cabo. Otro ejemplo de aplicación es el ligado con el sector ganadero dónde la monitorización biométrica y la geolocalización ayudan para controlar la salud y localización de los animales. No nos podemos olvidar de las crecientes *Smart Cities* y *Smart Buildings* donde el IoT se usa para tareas como el control del tráfico o control de los suministros de agua y calefacción.[1]

Haciendo uso de la definición de domótica podríamos decir que un sistema domótico es capaz de recoger información proveniente de unos sensores, procesarla y remitir la información a unos actuadores o salidas. Así pues, la domótica nos permite controlar diferentes acciones de nuestra casa, entre las cuales se encuentra la seguridad.

Gracias a los avances en la tecnología Wifi han surgido diferentes protocolos que nos permiten mayor escalabilidad, asicronía y menor acoplamiento. Entre los protocolos que nos permiten esto, el elegido es MQTT puesto que se trata de un protocolo simple, liviano y flexible perfecto para su aplicación IoT.

La seguridad del hogar es actualmente una preocupación de la población y se ha convertido en un negocio. Mediante el uso de IoT, podemos visualizar las imágenes de un habitáculo de forma remota para tener en todo momento nuestra casa segura, a un bajo precio y con un sistema simple.

Es por esto que con toda la información explicada, se va a desarrollar un sistema para la monitorización de la seguridad de diferentes partes de un campus utilizando una cámara y visualizando las imágenes en una aplicación web.

1.2. Objetivos

El objetivo principal de este proyecto es la realización de una aplicación IoT que permitirá la obtención de imágenes de múltiples ubicaciones de un campus universitario. En el objetivo principal se puede incluir la aplicación del protocolo MQTT en un entorno real y el uso de los conceptos explicados anteriormente sobre IoT. El entorno del campus sería un ejemplo práctico, pero este proyecto se podría utilizar en muchas otras situaciones entre las que se encuentra la vigilancia de un entorno natural mediante el cambio de conexiones Wifi por conexiones LPWAN tipo Lora. Se quiere construir una red de comunicación entre diversos dispositivos como son diversas cámaras y un servidor web mediante el uso de un nodo central que sirva de control principal. La aplicación web que reside en el servidor web nos permitirá la visualización de las imágenes capturadas.

Este objetivo principal se ha plasmado en los siguientes subobjetivos:

- Instalación y prueba del servidor MQTT en un ordenador monoplaca como es la Raspberry pi.
- Programación de la placa ESP32-CAM con la finalidad de tomar imágenes y enviarlas al servidor MQTT.
- Programación de un script que permita crear una imagen a partir de la información recibida por el servidor MQTT y su posterior guardado en el servidor web.
- Programación del servidor web que permita mostrar las imágenes en tiempo real.

1.3. Estructura de la memoria

Esta memoria está compuesta por 5 capítulos:

- *Capítulo 1*: En el capítulo 1 se explica el concepto de IoT y sus aplicaciones. Por otra parte se presentan los objetivos a alcanzar, la estructura del proyecto y la metodología a seguir.
- *Capítulo 2*: En el capítulo 2 se introduce una visión general del proyecto y de su funcionamiento.
- *Capítulo 3*: En el capítulo 3 se presentan los fundamentos teóricos sobre MQTT, NodeJS y tecnología usada.
- *Capítulo 4*: En el capítulo 4 se explica como se lleva a cabo el procedimiento del envío de las imágenes, así como las configuraciones de los elementos que tienen lugar en este procedimiento.
- *Capítulo 5*: En el capítulo 5 se detalla la segunda parte del proyecto que consiste en el servidor web y la visualización de las imágenes a través de una aplicación web.

- *Capítulo 6*: En el capítulo 6 se redacta una conclusión y las posibles líneas futuras que tendrá el proyecto.

1.4. Metodología de trabajo

Para definir la metodología del trabajo, partimos de una idea principal como es el desarrollo de una aplicación de Internet de las Cosas con almacenado y procesado de imágenes. A partir de esta idea, se establece una serie de tareas ordenadas cronológicamente. Más concretamente, se han realizado las siguientes actividades:

1. Investigación acerca de Internet de las Cosas y el protocolo MQTT (5 horas)
2. Elección y estudio de cuales son las mejores herramientas a emplear: entornos de programación, hardware, software y lenguajes de programación.(32 horas)
3. Estudio de las herramientas elegidas y del protocolo MQTT.(80 horas)
4. Desarrollo del proyecto (80 horas)
5. Pruebas del proyecto y depuración del código.(53 horas)
6. Redacción de la memoria (50 horas)

Como puede observarse, gran parte del tiempo se ha empleado en la redacción de la memoria y en la realización de los distintos objetivos a realizar . Además, algunas de las actividades han sido transversales a la realización del trabajo, como el desarrollo del proyecto y las respectivas pruebas.

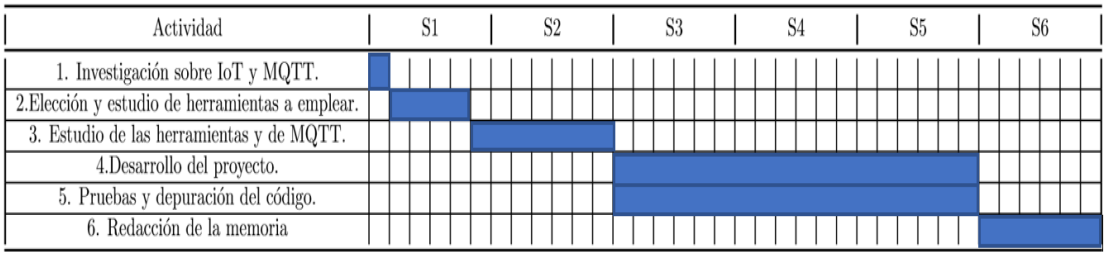

Figura 1.1: Diagrama temporal de Gantt

Capítulo 2

Visión general del trabajo

2.0.1. Explicación funcionamiento

La finalidad principal del trabajo, se basa en la realización de un sistema mediante el cual podamos visualizar en tiempo real las imágenes de diversas partes de un campus universitario, haciendo uso del protocolo MQTT y con la previa conexión Wifi. Está conformado por tres partes importantes que son:

- La cámara situada en cada parte del campus
- El servidor MQTT encargado de la distribución de imágenes
- El servidor web con el que visualizar las imágenes.

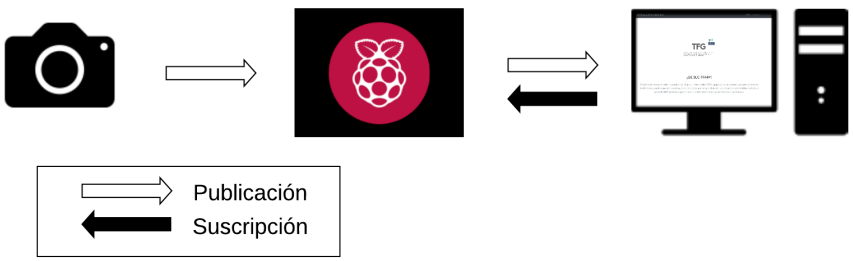

Figura 2.1: Conexión ESP-32 CAM a puerto USB

La cámara, se trata de una placa ESP32-CAM, que es la encargada del envío de las imágenes mediante el protocolo MQTT. En este trabajo, específicamente, se saca por pantalla las imágenes que pertenecerían a la ETSIT, al Ágora y a la Casa del Alumno. Así pues los topics estan nombrados con estos nombres, para facilitar su distinción.

Las imágenes, llegan a la Raspberry pi (broker) por medio de la conexión Wifi y usando el protocolo MQTT. Esta se encarga de leer cuales son los topics y de distribuir las imágenes a aquellos clientes que estén suscritos a estos topics respectivamente.

El servidor Web, recibe las imágenes y las almacena en el backend. Las imágenes se van re-escribiendo y al mismo tiempo se encarga de mostrarlas en el frontend para que el usuario las visualice.

En el servidor web se utiliza un template engine llamado Handlebars que nos permitirá realizar el frontend, es decir la parte visual de nuestra aplicación web. Así pues, mediante la petición del navegador a nuestro servidor web, obtenemos el front end y desde este se realizan peticiones Ajax al backend para refrescar las imágenes en tiempo real.

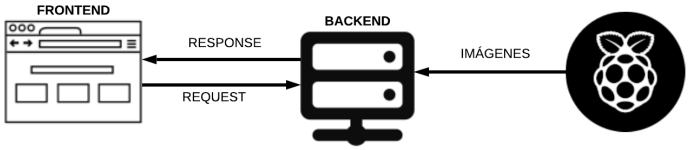

Figura 2.2: Funcionamiento general del servidor web

Una vez en el navegador, se puede visualizar una primera página con información sobre el TFG y acceso a las 3 pestañas donde visualizar cada una de las diferentes páginas con las imágenes que corresponden a cada uno de los lugares que se estan vigilando.

Figura 2.3: Página de inicio de la página web

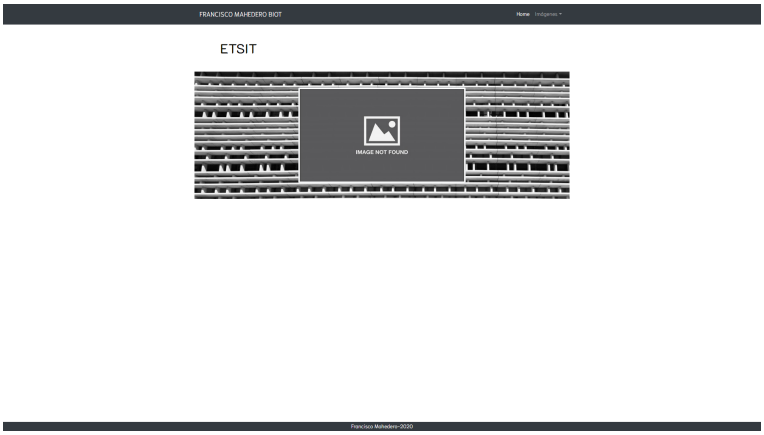

Figura 2.4: Página donde ver imágnes de la ETSIT de la página web

Este sistema permite entre otros, su uso como cámaras de videovigilancia. Este sistema tiene bastantes ventajas en aquellas situaciones donde se necesita un dispositivo que no consuma mucho y que su coste sea reducido. El coste debe de ser asequible para que pueda llegar a un mayor público. Así pues, únicamente con la compra de un dispositivo tan sencillo como es el usado en este sistema y con la conexión al servidor correspondiente, se puede monitorizar en todo momento el habitáculo que desees.

2.0.2. Arquitectura de red

La conexión entre dispositivos es fundamental para poder enviar la información mediante el protocolo MQTT. Para que esto sea posible, se conectan a la red Wifi local, permitiéndose así el intercambio de paquetes entre los distintos dispositivos que forman el proyecto.

Se debe distinguir dos tipos de IP para poder entender el funcionamiento de la red. Estos tipos de IP son dos:

- *IP local*: es la dirección IP asignada localmente a cada uno de los dispositivos que se conectan a esta red. Solo pueden ser vistas por aquellos dispositivos que se encuentren conectados a esta red.

- *IP pública*:es la dirección IP asignada por tu operador a tu router para poder ser accedido desde cualquier parte de internet.

Así pues en el primer escenario, en el que todos los elementos del sistema se encuentran en la red local, a todos los dispositivos se les asigna una IP local por parte del router. De este modo, aquellos dispositivos que quieran conectarse al broker lo podrán hacer siempre y cuando se encuentren en la misma red local.

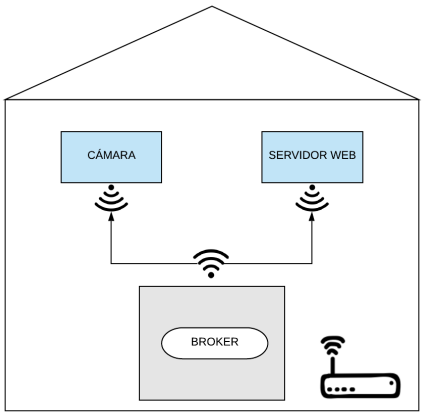

Figura 2.5: Arquitectura red local

En el segundo escenario en el que el broker se encuentre en la red local pero la cámara envíe información desde fuera de esta o que el servidor web se encuentre alojado en un hosting, es necesario la utilización de la IP pública para poder acceder a nuestra red local desde el exterior.

Por lo tanto para acceder al broker desde la fuera de la red local, se utiliza lo que se llama NAT, es decir una traducción de la dirección red pública a una local. El protocolo MQTT utiliza el puerto 1883 o el 8883 si la conexión es por medio de SSL. Así pues, cuando se quiera acceder al puerto 8883 para la conexión con el broker, se redireccionará a la dirección local del broker. Para ello es necesario modificar la configuración en el router y asignar la redirección de los puertos.

Figura 2.6: Configuración del router

Así pues, esas son las dos formas de acceso a la red que existen. De esta última forma el proyecto tiene más funcionabilidad que si únicamente se accediese al sistema desde dentro de una red local.

Capítulo 3

Fundamentos teóricos

En este capítulo se van a explicar los fundamentos teóricos del protocolo MQTT, de Node.js y de las herramientas utilizadas. En primer lugar, una explicación del protocolo MQTT y seguidamente Node.js y toda la información sobre los fundamentos de la tecnología utilizada en la parte de desarrollo web.

3.1. Protocolo MQTT

3.1.1. Introducción

El protocolo MQTT (Message Queue Telemetry Transport) se trata de un protolo de mensajería asíncrona, usado en el ámbito de las comunicaciones machine-to-machine (M2M), en el campo de IoT(Internet de las cosas). Su funcionamiento se basa en el intercambio de mensajes mediante el modelo de publicación y suscripción.

Este protocolo liviano permite su implementación en redes con un ancho de banda limitado y con alta latencia. Por otra parte, su flexibilidad, permite su aplicación en distintos dispositivos y servicios de IoT.

3.1.2. Modelo publicación/suscripción

El modelo publicación y suscripción fuciona a través de la conexión a un nodo central, llamado *broker*, por parte de los clientes. Tanto para el envío de los mensajes, como para su posterior recepción, ambos se suscriben a un *topic* donde se disponen los mensajes, convirtiendo así este modelo en un modelo no-bloqueante.

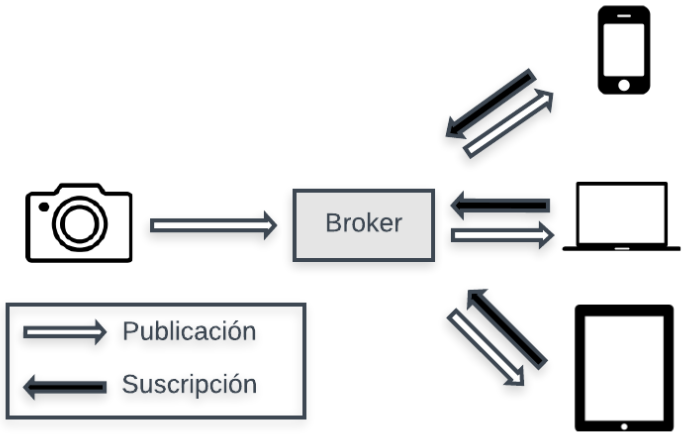

Figura 3.1: Ejemplo modelo suscripción/publicación

Este descople viene dado en tres dimensiones:

- Tiempo: ambos clientes no tienen porque estar funcionando al mismo tiempo.
- Espacio: ambos clientes no tienen porque conocerse entre sí.
- Sincronización: no se produce ningún efecto de interrupción en los clientes al publicar o recibir un mensaje de un *topic*.

3.1.3. Cliente MQTT

Los suscriptores y los publicadores, son cada uno de ellos un cliente MQTT. Un cliente MQTT puede ser tanto suscriptor como publicador, o ambos a la vez.

La ventaja que tienen los clientes MQTT es la facilidad de implementación, y al ser un protocolo ligero, el hardware no es un problema, ya que un cliente puede ser desde un ordenador hasta cualquier otro dispositivo como un microcontrolador.

En este trabajo hemos utilizado como Cliente MQTT una placa **ESP32-CAM** y un **ordenador**.

3.1.4. Broker MQTT

El broker MQTT se trata del servidor encargado de la distribución de los mensajes a los receptores. El broker recibe los mensajes, hace un chequeo del topic al que están suscritos y los encamina hacía los clientes suscritos a este topic. El broker nos brinda la posibilidad de que los mensajes sean persistentes, es decir, que se guarden los mensajes hasta que el cliente al que va dirigido se conecte.

Otra de las funciones del broker es la de autentificar la identidad de los clientes como medida de seguridad. En este trabajo utilizaremos como hardware del broker un ordenador de placa simple

llamado *Raspberry Pi* con el software Raspbian, versión adaptada de Debian. Por otra parte, el software instalado para llevar a cabo la función de broker será *Mosquitto*, escrito en C.[2]

3.1.5. Topic

El topic es el tema del mensaje, es decir, a quien va dirigido ese mensaje o de quien queremos recibir tal mensaje. El topic sirve como identificador al *broker* para poder encaminar cada uno de los mensajes recibidos hacia los clientes adecuados que también se hayan suscrito al *topic*. Los *topics* son *case sensitive*, es decir, disitinguen entre mayúsculas y minúsculas. Estos nos brindan las posibilidad de poder jerarquizarlos. Por ejemplo:

- casa/salon/temperatura
- casa/salon/humedad
- casa/terraza/temperatura

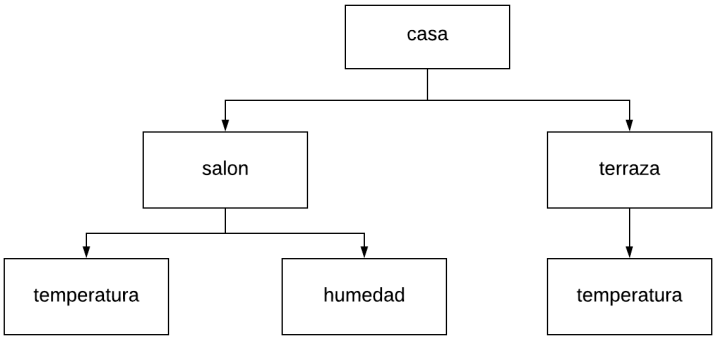

Figura 3.2: Ejemplo jerarquía de los topics

También podemos utilzar los comodines # y + para referirnos a cualquier nivel de la jerarquía. Ejemplo:

- casa/+/humedad: sustituye a casa/salon/humedad y casa/cocina/humedad.
- casa/#: engloba todo lo que se encuentre en casa (únicamente se puede utilizar al final).

3.1.6. QoS (Quality of Service)

MQTT dispone de un mecanismo para gestionar el envío de mensajes al cliente ante fallos. Los diferentes niveles que se aprecian son:

- **QoS 0**: El mensaje se envía una sola vez y no se recibe confirmación de entrega.
- **QoS 1**: El mensaje se envía hasta que el receptor recibe la entrega. Se pueden producir mensajes duplicados.

- **QoS 2** : Se garantiza la entrega del mensaje y solo una vez. Un punto en contra es que esto aumenta la sobrecarga y disminuye el rendimiento del sistema.

3.1.7. Intercambio de mensajes MQTT

MQTT está basado en TCP/IP y los mensajes del protocolo MQTT se dividen en tres partes:

- Cabecera fija: Consta de 2 a 5 bytes. Estos forman la cabecera de control y la longitud del mensaje. Esta longitud está codificada de 1 a 4 bytes, en los que se usan los 7 primeros bits y el restante se trata de un bit de continuidad.

- Cabecera variable: Esta cabecera es opcional.

- Payload: Esta formado por el contenido real del mensaje. Su máximo es de 256 MB aunque en las implementaciones reales el máximo se encuentra entre 2 y 4 kB.

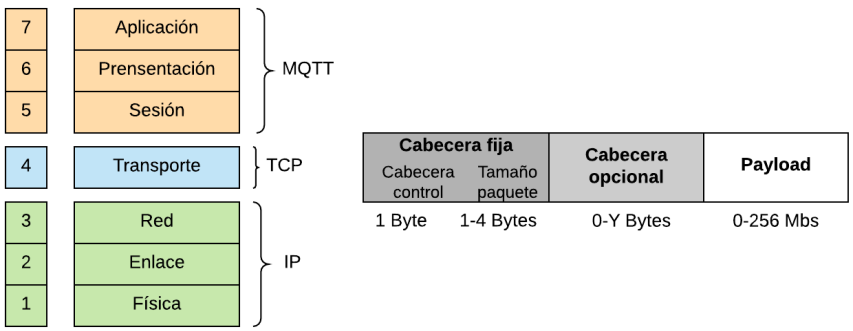

Figura 3.3: Mensaje MQTT

La conexión entre los clientes y el broker se hace por el puerto 1883 y con el 8883 si la conexión funciona sobre TLS. TLS nos proporciona autentificación y privacidad de la información entre extremos. Los tipos de mensajes junto con sus códigos son los siguientes:

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

Tabla 3.1: Mensajes intercambiados en MQTT.

Las peticiones PING se envían de forma regular al broker desde el cliente. El broker responde con un PING response, método que permite a ambas partes determinar si una aún sigue disponible.[3]

3.2. Tecnología Back end

3.2.1. Introducción Node.js

Node se trata de un entorno de ejecución de JavaScript orientado a eventos asíncronos. JavaScript y Node.js se ejecutan utilizando el motor de tiempo de ejecución JavaScript V8, siendo V8 el nombre del motor de JavaScript que alimenta Google Chrome.[4]

Figura 3.4: Logo de Node.js

3.2.2. El uso de Node.js

Node.js se usa principalmente en la creación de aplicaciones web ya que permite el manejo de una gran cantidad de conexiones simultáneas con alto nivel de rendimiento, es decir tiene gran capacidad de escalabilidad.

Esto es posible ya que node.js se basa en un modelo de solitud y respuesta sin bloqueo controlado por eventos. Es decir da la posibilidad de ser liviano y efeciente frente a las aplicaciones en tiempo real que se ejecutan en los dispositivos.

3.2.3. Funcionamiento de Node.js

Node.js a diferencia de otras técnicas utilizadas anteriormente en las cuales cada conexión crea un subproceso ocupando la RAM del sistema, node.js crea un solo subproceso y con el anteriormente nombrado modelo de entrada y salida, sin bloqueo de la salida, permite que se puedan soportar miles de conexiones al mismo tiempo mantenidas en el bucle de eventos.

Por lo tanto, se podría decir que el servidor consta de un subproceso que procesa un evento tras otro, así pues, al no tener que crear más subprocesos ni cambiar entre subprocesos, significa que tiene muy poca sobrecarga.

3.2.4. Express

Express.js se trata de un framework rápido y flexible para Node.js que nos permite la creacion de APIs y aplicaciones web fácilmente. Express.js nos da la posibilidad del manejo de rutas, archivos estáticos, uso de motor de plantillas como *HBS*, integración de base de datos, middlewares y manejo de errores.

Figura 3.5: Logo de Express

3.3. Tecnología Front end

3.3.1. HTML

HTML es un lenguaje de marcado utilizado en el desarrollo de páginas web. Su significado es Hypertext Markup Language que puede ser traducido como Lenguaje de Formato de Documentos para Hipertexto. Este estándar surgió a cargo de W3C, la cual es la organización que se encarga de la estandarización de muchas de las tecnologías relacionadas con la web.

Figura 3.6: Logo de HTML

HTML se encarga de indicar una descripción de sobre como los textos y la estructura de una página web aparecen. Al tratarse de un estándar, busca ser un lenguaje que dé la posibilidad de que cualquier página web sin importar la versión pueda ser interpretada por igual por cualquier navegador web.

La última recomendación ha sido *HTML5* en el que se añaden etiquetas, atributos y comportamientos, así como una gran variedad de funcionalidades que permiten a los sitios web y aplicaciones tener mayor alcance.

3.3.2. Template engines

Un template engine es una implementación de software que permite que un documento, es decir, una plantilla contenga parámetros, identificados con una sintaxis especial para que luego se puedan reemplazar por argumentos enviados a esa plantilla por un sistema de procesamiento.

Figura 3.7: Logo de Handlebars

3.3.3. JavaScript

JavaScript es un lenguaje, pequeño y liviano, de scripting multiplataforma y orientado a objetos.

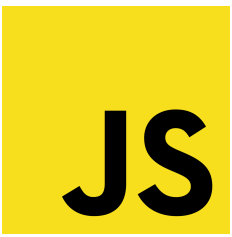

Figura 3.8: Logo de JavaScript

El núcleo de JavaScript puede extenderse a diversos propósitos, los cuales son:

- *Client-side JavaScript*: proporciona objetos para controlar un navegador y su modelo de objetos. Esto permite entre otras acciones, responder a los eventos de un usuario. Se utiliza en nuestro proyecto para incluir las imágenes en las respectivas páginas en el front end.
- *Server-side JavaScript*: proporciona objetos relevantes a la ejecución de JavaScript en un servidor. Permite que una aplicación efectúe la manipulación de archivos en el servidor. Se usa en el proyecto en la realización del back end.

La última versión que se publicó fue ECMAScript 6.

3.3.4. Peticiones AJAX

Ajax se puede definir como la técnica que permite al servidor y navegador intercambiar información de forma asíncrona. Se pueden hacer peticiones de texto HTML, XML o JSON. Su uso permite que una página web que ya ha sido cargada, solicite nueva información sin la necesidad de recargar toda la página web por completo, por lo tanto, estas aplicaciones se ejecutan en el cliente (navegador del usuario) mientras que se produce la comunicación asíncrona con el servidor en segundo plano.

Figura 3.9: Logo de Ajax

Ajax ofrece mejor experiencia para el usuario, mejorando la interactividad, la usabilidad y la velocidad de las aplicaciones.

3.3.5. JQuery

Se puede definir JQuery como una biblioteca de JavaScript. Permite el desplazamiento y manipulación de documentos HTML, manejo de eventos, la animación y Ajax sean mucho más simples con una API fácil de usar en diversos navegadores. Sin el uso de JQuery, las funcionalidades basadas en JavaScript que ofrece requerirían mucho más código, por lo tanto se pueden lograr grandes resultados en menos tiempo y espacio.

Figura 3.10: Logo de la librería JQuery

3.3.6. CSS

CSS es el lenguaje utilizado para presentar de forma estructurada en lenguaje de marcado como es HTML o XML. CSS1 se encuentra obsoleto, CSS2 funciona como recomendación y actualmente CSS3 es el lenguaje que se encuentra en camino de ser el estándar.

Figura 3.11: Logo de CSS

CSS nos permite separar el contenido de la forma de presentación del documento, así pues, se

logra que los documentos HTML adquieran una cierta apariencia utilizando hojas de estilo con la extensión *.css*. Gracias a la separación entre contenido y forma de presentación del documento, se pueden crear distintos estilos según el método de renderizado.

3.3.7. BootStrap

Bootstrap es un framework CSS que fue desarrollado en una primera estancia por twitter, esta es una herramienta utilizada con el fin de poder crear interfaces de usuario que sean capaces de adaptarse a cualquier dispositivo y pantalla. Bootstrap nos permite crear cualquier tipo de sitio web mediante el uso de estilos y elementos de sus librerías CSS.

En sus librerías se incluye botones con características avanzadas, etiquetas, diferentes formatos para los mensajes de alerta, barras de progreso y la posibilidad de insertar imágenes responsive.

Figura 3.12: Logo de BootStrap

Los componentes JavaScript que utiliza BootStrap usan la librería JQuery. Proporciona elementos adicionales como son diálogos, sliders y tooltips.

3.4. Hardware

Como ya se ha explicado, el hardware a utilizar está compuesto por estos tres elementos: Raspberry Pi, ESP-32 CAM y un ordenador.

3.4.1. Raspberry Pi 4

La Raspberry Pi 4 se trata del último modeo de este mini ordenador de bajo coste. Este modelo se caracteriza por diferentes características:

- Puede decodificar vídeo en 4k, con un almacenamiento más rápido a través de USB 3.0 y con conexiones más rápidas mediante Ethernet Gigabit real ya que no comparte bus en la conexión ethernet.
- Admite la conexión de dos pantallas a la vez con una resolución de hasta 4K.
- Ofrece un rendimiento de dos a tres veces mayor que el anterior modelo.

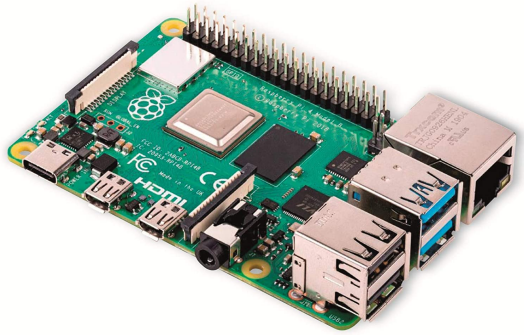

Figura 3.13: Figura de la Raspberry Pi 4

La Raspberry Pi 4 incorpora el procesador Broadcom BCM2711 de 64 bits con cuatro núcleosARM A72 y una velocidad del procesador de 1.5GHz.

La memoria RAM es de tipo LPDDR4 a una velocidad de 2400MHz con tres capacidades de RAM: 1GB, 2GB y 4GB.

En esta Raspberry Pi 4 se aprecia que soporta PoE para alimentar la propia Raspberry mediante el cable de red Ethernet. La Raspberry Pi se puede alimentar por tres vías:

- *Puerto USB tipo C*: es necesario disponer de una fuente de alimentación que proporcione una tensión de 5V o 5,1V y una intensidad de corriente de 3A.
- *PoE HAT*: se puede alimentar mediante el cable de red proporcionándole alimentación vía switch PoE o inyector PoE.
- *GPIO*: es necesario que se proporcione un mínimo de 3A de intensidad.

La Raspberry Pi, en cuanto a la conectividad inalámbrica respecta dispone de Wifi con los estándares 802.11b/g/n/ac y dispone de bluetooth 5.0. A parte, entre otros puertos que dispone la Raspberry Pi 4, se dispone de dos puertos micro HDMI 2.0 jack 3,5mm para la salida de audio y vídeo y se dispone también de un slot para tarjetas micro SD.

En cuanto al análisis externo, para poder utilizar la Raspberry Pi es necesario el uso de un cable HDMI y una fuente de alimentación de 5,1V y 3A.

3.4.2. ESP-32 CAM

La ESP32-CAM tiene un pequeño tamaño de camara muy competitivo que puede operar de forma independiente con un sistema mínimo y con una corriente *deep sleep* de hasta 6mA, permitiendo un gasto de energía menor. Esta placa es un chip combinado de 2,4 GHz con Wi-Fi y Bluetooth diseñado con la tecnología de TSMC (empresa de semiconductores) de ultra baja potencia de 40nm.

Figura 3.14: Figura de la ESP32-CAM

Esta está diseñada para lograr el mejor rendimiento de potencia y radiofrecuencia, siendo así versátil, robusta y fiable en una amplia variedad de aplicaciones y diferentes escenarios.

Su utilización es muy extendida en cuanto al campo de IoT. Presenta características que permite que sea idónea en este escenario com los son:

- Se despierta periódicamente, solo cuando se detecta una condición específica.
- Su ciclo de trabajo bajo se utiliza para minimizar la cantidad de energía que gasta el chip.
- La salida del amplificador de potencia es ajustable, lo cual permite que haya una buena compensación entre velocidad de datos, consumo de energía y el rango de comunicación.

Entre las caracteríticas más importantes por las que se ha elegido este dispositivo es debido a la posibilidad de su conexión WiFi. Entre las características que soporta respecto al Wifi se encuentra:

- 802.11 b/g/n
- 802.11 n (2.4GHz), más de 150 Mbps
- Monitoreado automático de los Beacon
- 4 interfaces Wifi virtuales
- Defragmentación
- Bloqueo de ACK inmediato

Todas estas caracteríticas permiten que la placa ESP32-CAM sea idónea para un proyecto donde IoT es el tema principal.[5]

3.4.3. Ordenador

El dispositivo utilizado como tercer elemento necesario, ha sido un ordenador portátil, en este caso un Lenovo Legion del cual se disponía. Este dispositivo es el que nos permite llevar a cabo el servidor web.

Actualmente en el mercado hay una gran variedad de modelos de ordenadores tanto de torre como portátiles, clasificables por su tamañoo, precio etc.

Las caracteríticas de este dispositivo no son muy importantes debido a que su función se podría haber realizado en cualquier otro dispositivo ya que no demanda tener unas características específicas para poder llevar a cabo la acción.

Capítulo 4

Procedimiento para envío de imágenes

4.1. Introducción

En este capítulo, se explica como se ha llevado a cabo la configuración de la primera parte del trabajo, es decir, la configuración del *Mosquitto*[2] junto con la explicación del script para el envío de las imágenes.

4.2. Configuración de Mosquitto

La configuración de *Mosquitto* en la Raspberry Pi se lleva a cabo mediante la línea de comandos con *apt-get*, una herramienta para gestionar paquetes instalables. Así pues, se debe tener conexión a internet y abrir una terminal en la Raspberry Pi. Los comandos a utilizar son los siguientes:

Primero descargar la clave de firma:

```
$ sudo wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
```

Se añade la clave a una lista para permitir la autenticación de un paquete que descargaremos posteriormente:

```
$ sudo apt-key add mosquitto-repo.gpg.key
```

En tercer lugar accedemos a esta carpeta

```
$ cd /etc/apt/sources.list.d/
```

Una vez hemos accedido a esta carpeta , el siguiente paso es la descarga de la lista de repositorios de Mosquitto. En este caso, la versión de Raspbian es *Stretch*.

```
$ sudo wget http://repo.mosquitto.org/debian/mosquitto-stretch.list
```

En siguiente lugar, se actualiza la lista de paquetes disponibles

```
$ sudo apt-get update
```

Para finalizar, procedemos a la instalación del mosquitto en nuestra Raspberry Pi

```
$ sudo apt-get install mosquitto
```

4.2.1. Prueba del funcionamiento del broker

Para poder llevar a cabo la prueba del funcionamiento del broker, es necesario la implementación de un cliente. Para ello, se introduce el siguiente comando:

```
$ sudo apt-get install mosquitto-clients
```

Por último, escribiendo estos dos comandos se consigue conectarte al broker como publicador o suscriptor.

```
$ mosquitto_sub -h BROKER -t TOPIC
```

```
$ mosquitto_pub -h BROKER -t TOPIC -m MENSAJE
```

Donde:

- **-h**: a continuación se introduce la IP o nombre de la máquina donde se encuentre instalado el mosquitto. Para hacer esto más sencillo, a la Raspberry Pi, se le ha asignado una IP estática, para evitar tener que averiguar la IP cada vez que se haga la prueba, siendo esta 192.168.1.132.
- **-t**: a continuación se escribe el topic al cual se quiere suscribir o publicar.

Así pues, la prueba de que el funcionamiento del broker es el adecuado será posible.

4.2.2. Configuración de usuario y contraseña

Para poder añadir un nivel de seguridad al sistema, es necesario el uso de de usuarios y contraseñas[2]. Los pasos a seguir son:

La creación de un usuario y una contraseña mediante el siguiente comando.

```
$ sudo mosquitto_passwd -c /etc/mosquitto/passwd usuarioprueba
Password:
Reenter password:
```

Para que este usuario y esta constraseña sean usados se debe modificar el archivo *default.conf*.

```
$ sudo nano /etc/mosquitto/conf.d/default.conf
```

Y escribir en este archivo:

```
password_file /etc/mosquitto/passwd
allow_anonymous false
```

Así ya sería posible la autenticación. Para finalizar, solo hace falta reiniciar el Mosquitto:

```
sudo systemctl restart mosquitto
```

4.2.3. Configuración TLS/SSL

Como se ha comentado previamente, las conexiones se llevan a cabo de forma segura utilizando seguridad TLS. Para que esto sea posible es necesario crear una conexión encriptada entre el Broker MQTT y el cliente MQTT.

Para la creación de los certificados y llaves usamos **openssl**[6] software. Solamente necesitamos un certificado del servidor de confianza, es decir, no necesitamos crear un certificado de cliente ni claves. Se ha de tener en cuenta que **openssl** ya no provee un certificado de autentificación, por lo que debemos crear nuestro propio certificado CA autofirmado.

Por lo tanto, lo que al final obtenemos son:

- **Cliente**: Certificado CA (Autoridad certificadora).
- **Servidor**: Certificado CA que firma el certificado del servidor, el certificado del servidor firmado por el certificado CA y clave privada del servidor para la desencriptación.

Así pues, tendríamos creado todo lo necesario para poder llevar a cabo una conexión segura entre los clientes y el Mosquitto.

4.3. Configuración ESP-32 CAM

En esta sección se explicará como se ha conectado la placa ESP-32 CAM al puerto USB del ordenador, los pasos previos a la programación del script en el IDE de Arduino y se analizará las distintas partes que componen el programa.

4.3.1. Proceso de conexión

El proceso de conexión entre la placa y el puerto USB se llevó a cabo haciendo uso del módulo CP2102 USB a TTL y de 4 cables para la respectiva conexión. Las conexiones del módulo a la placa ESP-32 CAM, respectivamente, son las siguientes:

- +5V - 5V
- TXD - U0R
- RXD - U0T
- GND - GND

Finalmente también es necesario la conexión puente en la placa de los pin GND y IO0, los cuales solo sirven para habilitar la carga del código de programa.

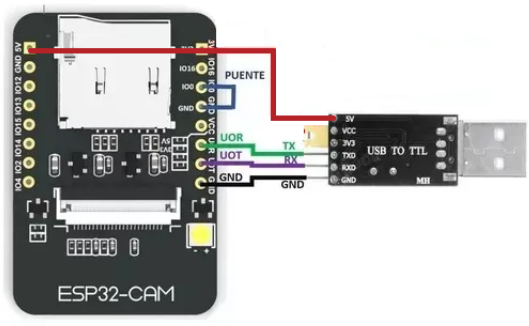

Figura 4.1: Conexión ESP-32 CAM a puerto USB

4.3.2. Pasos previos en el IDE de Arduino

Para trabajar con la placa ESP-32 CAM, se necesita tener instalada la librería correspondiente. Para ello utilizaremos Github y la librería relacionada con esta placa en la página de *espressif* en Github. Así pues, los pasos a seguir son los siguientes:

1. Instalación del IDE de Arduino y de Github.
2. Abrimos una ventana de Github en el directorio *[ARDUINOSKETCHBOOKDIR]/hardware/espressif/esp32*, donde clonar el repositorio y mediante este comando lo clonamos.

```
$ git clone https://github.com/espressif/arduino-esp32.git
```

3. Seguidamente se accede a *[ARDUINOSKETCHBOOKDIR]/hardware/espressif/esp32* donde abriremos una ventana de Github y se ejecutará el siguiente comando.

```
$ git submodule update --init --recursive
```

4. Seguidamente en este directorio ejecutaremos el archivo *get.exe*.

5. Conectamos la placa y esperamos a que se instalen los drivers.

6. Seleccionamos en Herramientas la placa ESP32 Dev Module y el puerto COM donde se ha conectado la placa.

7. Cargamos el programa en la placa (necesario presionar el botón de reset antes de hacerlo) y para empezar con su funcionamiento, se debe desconectar el puente creado (entre los pin GND y IO0) y presionar el botón reset de nuevo.

Así pues, ya estará lista la placa para funcionar.

4.3.3. Cliente MQTT Cámara

La cámara de fotos es el dispositivo encargado de la captura y publicación de las imágenes en este sistema. Su función es capturar las imágenes cada cierto tiempo y enviarlas al broker a través de una conexión TLS y suscribiéndose a un topic.

Para poder llevarse a cabo tal tarea, únicamente es necesario el acceso a una red Wifi y el establecimiento de la conexión al Broker MQTT. También es necesario el uso de distintas librerías para poder realizar la conexión WiFi, así como, realizar la conexión MQTT con el broker. La ventaja de utilizar MQTT en este sistema, es que la cámara no necesita de la interacción de otros dispositivos para poder llevar a cabo su tarea, es decir, esta envía las fotos pero desconoce si otros dispositivos lo van a recibir.

El procedimiento de la cámara se basa en tres partes principales que son:

- Conexión a la red wifi.
- Conexión con el broker.
- Captura de la imagen.
- Envío de la imagen.

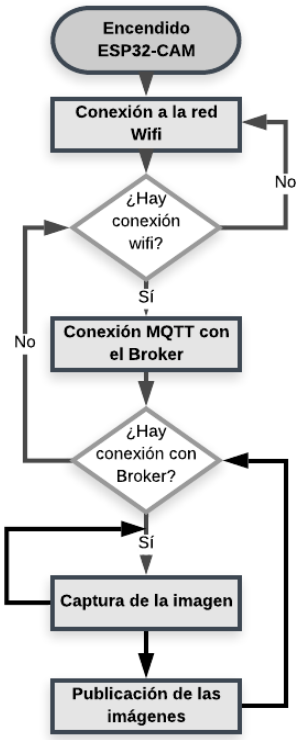

Figura 4.2: Diagrama de flujo del funcionamiento de la cámara

Lo que se muestra en este diagrama es el procedimineto que lleva a cabo la cámara para el envío y captura de las imágenes.

En primer lugar, la cámara se conecta a internet a través de una red Wifi, siendo la red Wifi de casa, es decir utiliza la IP local que se le ha asignado. La cámara vendrá con una configuracón de credenciales predefinida en el código, ya que no se dispone de una interfaz visual para poder hacer esta elección configurable. En el caso de ser necesario el cambio de la red Wifi, se debe acceder al script y cambiarlo manualmente. La conexión a la red Wifi, se intenta de forma constante, hasta que es posible tal conexión.

En segundo lugar, se lleva a cabo la conexión al servidor central, es decir, al Broker MQTT. Para poder realizar esta conexión, son necesarias las credenciales de autentificación (compuestas por un usuario y una contraseña) y el certificado para la conexión TLS/SSL que se ha explicado con anterioridad. Así pues la cámara intenta la conexión, si esta no es posible, se espera 5 segundos y se vuelve a intentar la reconexión. Una vez se ha realizado la conexión satisfactoriamente, la cámara ya está preparada para poder enviar las imágenes.

Cabe destacar, que si en cualquier momento y en cualquiera de los procesos, se va la conexión en la red Wifi o con el broker, la cámara intentará reconectarse de forma automática, sin necesidad de volver a cargar el programa.

En tercer lugar, la cámara se encarga de capturar la imagen. Previamente a la captura de las imágenes, se inicializa la cámara. La cámara captura la imagen y si se ha hecho de forma satisfactoria

ya procede al envío de la imagen, sinó, intenta la capturar de nuevo.

Para finalizar, se envían las imágenes a los topics *ETSIT*, *agora* y *casaAlumno*.

Para poder seguir todo el procedimiento y que es lo que ocurre en cada momento, se imprime en el monitor serial cada uno de los pasos que se llevan a cabo. Así pues, en el caso de no saber cual es el error en el funcionamiento del sistema, únicamente debemos mirar que ocurre en el monitor serial.

4.4. Escalabilidad del sistema

En todo sistema actualmente que conlleve el tratamiento de información y en una sociedad con un creciente uso de la tecnología sobre todo en IoT, es necesario y imprescindible que todo sistema sea capaz de poder adaptarse a las necesidades del volumen de procesamiento de información.

Este sistema, donde el número de clientes puede aumentar exponencialmente,es en todo momento escalable, es decir, que mediante ciertas modificaciones es posible el aumento del tráfico de información entre los clientes MQTT y el broker. Entre las diferentes maneras que hay de aumento de la escalabilidad se encuentran:

- *Potencia individual*: Aumento de la capacidad del broker por medio del reemplazo de la máquina actual por una más potente.
- *Número de brokers*: Es posible la utilización de más de un broker y incluso la interconexión entre ellos para así acabar formando un entramado que permita un aumento considerable de la capacidad de procesar información de distintos clientes MQTT.

Así pues, mediante esta capacidad y el uso de Node js que como ya hemos indicado funciona mediante eventos y que le aporta también esta capacidad de ser escalable, hace este proyecto escalable en todos los elementos que lo conforman.

Capítulo 5

Servidor Web

5.1. Introducción

En este capítulo, se lleva a cabo la explicación sobre el procedimiento para la visualización de las imágenes en el servidor web que se ha programado.

5.2. Funcionamiento general

El servidor web, se puede definir como el software utilizado para la distribución de contenido web mediante la realización de conexiones unidireccionales o bidireccionales, síncronas o asíncronas con el cliente. El cliente es el navegador web, el cual mediante una petición al servidor web, el navegador recibe un documento estático o un documento generado de forma dinámica, es decir, que el servidor tiene que ejecutar un código de programa antes de tramitar la respuesta.

Para que esto sea posible, se introduce en el cliente una URL que por medio de una transmisión HTTP o HTTPS, protocolo el cual se basa en los protocolos de red IP y TCP, el servidor entrega los contenidos a los ordenadores que hayan realizado la petición.

El servidor web programado en NodeJS se ha realizado mediante el framework Express que ofrece muchas facilidades a la hora de realizar APIs y aplicaciones web. Existen dos tipos de peticiones a un servidor web:

- GET: El método GET está dirigido a la recuperación de datos, es decir, obtener información del servidor. Así pues, por medio de la respuesta del GET obtenemos la información como podría ser un documento HTML. La ventaja de las llamadas por el método GET es que estas pueden ser cacheadas, es decir, ser guardadas en el historial navegador y indexadas por los buscadores.
- POST: El método POST está dirigido al envío de información por parte del cliente al servidor para que se actualice o se cargue. Esto puede ser utlizado, por ejemplo, para el envío de información en un formulario.

Así pues, el funcionamiento de este proyecto se basa en la petición del navegador al servidor web para que le facilite los archivos del front-end que es la parte visual de nuestra aplicación y

donde se visualizan las imágenes. Por otra parte, existen unas peticiones al servidor web constantes, cada cierto tiempo, para poder refrescar las imágenes en tiempo real. Nuestra aplicación web, únicamente utiliza peticiones GET, tanto para la obtención de cada una de las páginas como la obtención de las imágenes. En el proyecto se pueden diferenciar dos partes que son: front end y back end.

5.3. Back end

Se puede definir el back end como la parte del desarrollo web que se encarga de que toda la lógica de la página web funcione, es decir, que los procesos para que funcione la página web se lleven a cabo de forma correcta.

Para la realización del back end se ha utilizado el framework Express de Node.js como ya he comentado anteriormente. Así pues, el proyecto esta basado en la siguiente jerarquía:

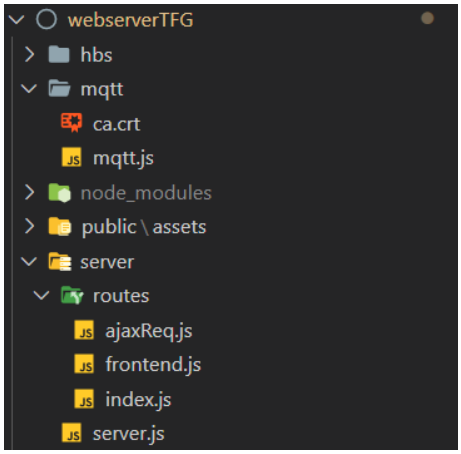

Figura 5.1: Jerarquía de archivos del backend

El archivo js principal es el server.js. Este archivo es el encargado de llevar a cabo el proceso de lanzar el servidor web y todas sus funciones complementarias. En cada uno de los archivos que constituyen el backend es necesario el requerimiento de express. En el archivo server.js se dispone de las distintas rutas y archivos para poder acceder a los diferentes recursos del servidor:

- Script del cliente MQTT.
- Ruta a los archivos pertenecientes al front end.
- Ruta a los archivos parciales del front end.
- Ruta a la ruta donde se encuentra la lógica del envío de las páginas web requeridas en cada momento y de las imágenes.

Cabe destacar que en este archivo también se encuentra la función encargada de escuchar en todo momento las peticiones de los usuarios que se quieran conectar al servidor web. En el proyecto se ha utilizado el puerto 3000 para su uso local, pero para poder desplegar el proyecto en un entorno

real, ha sido necesario crear una constante que según el entorno en el que se encuentre la aplicación web, seleccione un puerto, siendo estos el 3000 para entorno local o el puerto asignado en el hosting.

5.3.1. Cliente MQTT

El cliente MQTT de la parte del servidor es el encargado de recibir la información de las imágenes y guardarlas en el formato apropiado en una carpeta del servidor. Para poder llevar a cabo estas funciones es necesario la instalación del paquete MQTT para Node.js y la conexión del cliente a la red Wifi y al broker. La conexión a la red Wifi se realiza por medio de la conexión del hardware donde se encuentra alojado el software del servidor web, en este caso se trata de un ordendor portátil.

Para la conexión al broker MQTT, es necesario la configuración de las distintas opciones, entre las que se encuentran: el puerto de conexión 8883, que es el correspondiente con la conexión segura TLS/SSL, el usuario y contraseña para la autenticación con el broker así como indicar el certificado para la conexión TLS y la opción de que no se tenga en cuenta el nombre del dominio en el certificado TLS. Esta opción última es necesaria debido a un problema que existe con el nombre del servidor en el certificado obtenido mediante openssl.

El cliente MQTT que se pone en funcionamiento al inciar el servidor web, inicia en primera estancia la conexión con el *broker* de la forma que se ha explicado anteriormente en el apartado de arquitectura de red, es decir mediante la IP asignada en la NAT o mediante la IP pública del router y una dirección de puerto asignada si se trata de una conexión desde fuera de nuestra red local. Hasta que la conexión no se ha realizado, el cliente MQTT persiste en el intento de conexión hasta que esta sea posible.[7]

Seguidamente se produce la suscripción a cada uno de los topics correspondientes, siendo estos los ya mencionados *ETSIT*, *casaAlumno* y *agora*. En el momento en que se empiezan a recibir los distintos mensajes provenientes del broker, se guarda esta información con el nombre del topic que pertoque y con la extensión *.jpg* para que así se guarde con formato de imagen y sea posible la visualización de las mismas. Las imágenes son guardadas en la carpeta de imágenes del servidor. Como todas las imágenes adquieren el mismo nombre, estas se van sobreescribiendo y así se evita una sobrecarga de imágenes y la necesidad de borrar parte de ellas cada cierto tiempo. A parte, permite que el cliente siempre tenga una imagen que visualizar al acceder a la aplicación web.

Cabe destacar que si en cualquier momento hay algún error en cualquiera de los procesos, se produce una alerta que nos indica cual es el error que está ocurriendo.

5.3.2. Routing

Cuando se habla de routing, se refiere a cómo, las solicitudes de los clientes son respondidas por los puntos finales de una aplicación. La aplicación se encarga de escuchar las solicitudes que coinciden con las rutas especificadas desde el front end y los métodos especificados y cuando se detecta una coincidencia se llama a la función encargada de la devolución del recurso requerido.

En la carpeta del servidor Routes, se pueden encontrar de forma ordenada, los distintos archivos javascript que contienen las funciones encargadas de enviar las respuestas al front-end tras una petición GET de este mismo.

Desde el archivo principal que es el server.js se accede a los archivos con las rutas. Así pues podemos encontrar dos archivos que contienen las rutas necesarias para llevar a cabo las funciones de la aplicación web.

El primero y el inicial al correr el servidor es el archivo *frontend.js*. Este archivo contiene las rutas del inicio de la página web y las rutas de las distintas pestañas con las imágenes correspondientes. Así pues las rutas se corresponderan con:

- /: proporciona el archivo correspondiente con la página de inicio.
- */Etsit*: proporciona el archivo correspondiente con la página de visualización de las imágenes de Etsit.
- */casaAlum*: proporciona el archivo correspondiente con la página de visualización de las imágenes de la casa del Alumno.
- */Agora*: proporciona el archivo correspondiente con la página de visualización de las imágenes del Ágora.

En segundo lugar, se encuentra el archivo contenedor de las rutas encargadas de servir las imágenes tras las peticiones ajax que provienen del front end. Su funcionamiento consiste en tres rutas que dependiendo de que sección del front end lo pida, se le servirá una imagen u otra. Así pues, una vez se recibe la petición con la ruta, el back end se encarga de verificar si la imagen que se está pidiendo está o no disponible. En el caso de que sí que esté disponible, se enviará una respuesta con dicha imagen, sinó la respuesta que se enviará será la de una imagen con la información de imagen no encontrada. La ruta para cada una de estas imágenes son:

- */etsitFoto*: devuelve la imagen ETSIT.jpg.
- */casaAlumno*: devuelve la imagen casaAlumno.jpg.
- */agoraFoto*: devuelve la imagen agora.jpg.

Cada una de las imágenes disponibles se enviarán con una marca de tiempo, esto se hace así para poder evitar que el caché del navegador no utilice siempre la imagen que tenga almacenada, ya que esta imagen siempre tiene el mismo nombre. Así pues cada una de las imágenes enviadas tienen un distintivo de la anterior y se asegura que sea distinto mediante el *timestamp*.

5.4. Front end

El front end puede describirse como la parte del desarrollo web que se encarga de interactuar con los usuarios, es decir, la parte visual con la que el usuario interactúa. Cuando el usuario introduce el nombre de dominio correspondiente (en este caso localhost:3000), el navegador se encarga de hacer las peticiones al servidor web correspondiente y la cual, en primera estancia, es la petición para obtener el home. En la jerarquía de carpetas se distinguen la carpeta public que es aquella la cual contiene todos los archivos que pueden ser públicos, es decir, visibles al usuario y la carpeta views que contiene los archivos hbs de cada uno de las páginas de la aplicación web.

Para la realización de la parte visual, se ha utilizado un template engine llamado Handlebars (HBS) que ya se ha mencionado. Mediante el uso de HTML y CSS se ha dado forma a esta parte visual y con el uso de javascript se ha podido crear un sitio web con el cual interactuar.

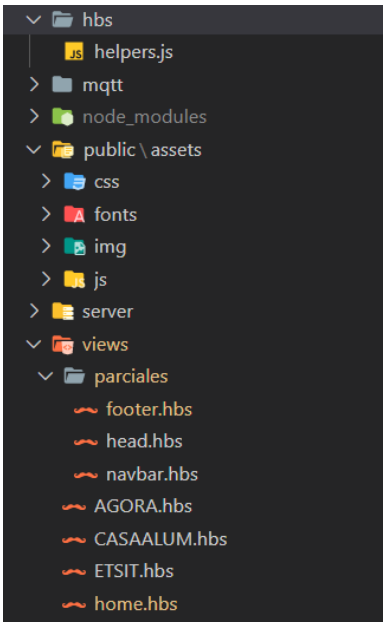

Figura 5.2: Jerarquía de archivos del frontend

Los archivos *hbs* se pueden dividir en dos partes: aquellos los cuales se van a repetir en cada una de las páginas como sería el header y el footer y que son llamados parciales, y por otra parte estarían cada una de estas páginas con una función distinta ,que son el home y las páginas de visualización de las imágenes. Para la escritura de ambos se ha usado HTML y CSS, siendo BootStrap la librería utilizada de CSS.

Los archivos JavaScript del cliente también se dividen en dos funciones. Por una parte está el archivo encargado de la creación de la imagen en cada una de las páginas y que se comparte entre todos los archivos con este fin. Este archivo es llamado *crearImagen.js*. Por otra parte se encuentra el encargado de llevar a cabo las peticiones ajax al back end y que ha sido programado utilizando la librería JQuery de JavaScript.

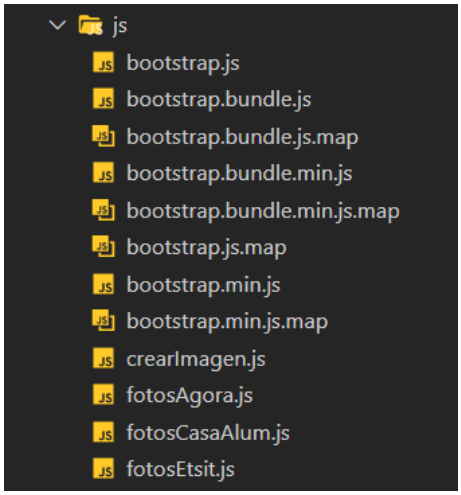

Figura 5.3: Archivos JavaScript del frontend

5.4.1. Archivos HBS

Los archivos HBS son los encargados de darle forma a cada una de las páginas en la aplicación web, mediante el uso de HTML y CSS. El uso de HBS permite entre otros, la facilidad de poder reutilizar parte del código que sea necesario, código el cual se llaman parciales.

5.4.1.1. Views Parciales

Los parciales son los archivos los cuales contienen aquello que en toda página web se repite en su estructura: head, footer y navbar.

Empezando por el head, este está compuesto por todos aquellos links que contienen la información sobre los archivos css que dan forma a cada una de las páginas, conjuntamente con el título, que aparece en la pestaña del navegador.

En segundo lugar, se tiene el navbar, donde se estructura y da forma al menú. El navbar está constituido por la pestaña de Home, la cual accede a la pantalla inicial y otra pestaña desplegable llamada Imágenes que contiene el acceso a cada una de las tres páginas con las imágenes correspondientes. El título que se ha utilizado también da acceso a la pantalla de home.

En tercer lugar, está el footer, donde se indica el nombre del autor y la fecha actual. Para que se pueda mostrar el año de forma correcta, se lleva a cabo mediante el uso de helpers que proporcionan la posibilidad de crear una función donde devolver la información que interesa. Para poder incluirlo en el código, únicamente es necesario el uso de {} donde se incluye el nombre de la función.

Cada uno de los parciales se incluyen en las demás páginas mediante el uso de {{>}}, lo que permite la reutilización de los parciales en cada una de las páginas. Un ejemplo sería {{>footer}}, que incluye el footer en todas aquellas páginas donde se haya incluido esta sintaxis.

```
{{> head}}
{{> navbar}}
```

Figura 5.4: Inclusión de los parciales

5.4.1.2. Views principales

Las páginas principales son el home y las páginas donde se visualizan las imágenes. El home y las páginas de las imágenes están compuestas por una distinta organización en su estructura escrita en HTML. Así pues tenemos dos organizacion distintas:

- *Home*: Este view principal está compuesto por 3 partes distinguidas, separadas en forma de *div*. La primera parte se trata de una imagen que simula el logo de una empresa con el título del TFG. La segunda parte se puede apreciar un resumen explicativo del TFG. Para la realización de este resumen se han utilizado dos etiquetas distintas:*h3* para el título y *p* para el texto que compone este div. Para finalizar, el último div consta de 3 tarjetas, compuestas por una imagen y por un texto descriptivo de la información que se obtiene al navegar a través de la tarjeta y el respectivo botón para acceder a la página donde se muestra la imagen respectivamente.

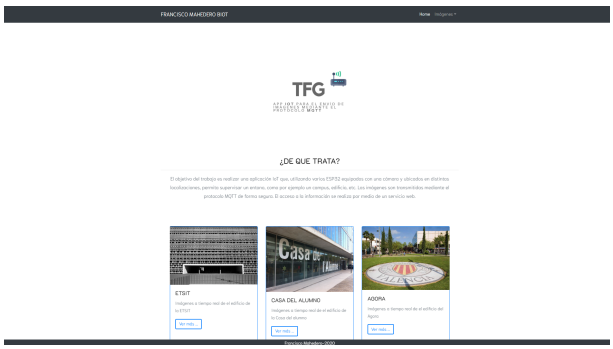

Figura 5.5: Página de inicio de la página web

- *ETSIT, AGORA, CASAALUM*: Estos views principales estan divididos en 2 div que se agrupan en uno principal. El primer div contiene un *h1* para el título principal de la página y el otro div contiene la imagen que se va refrescando conforme se van obteniendo. Detrás de esta imagen, hay otra imagen de fondo que corresponde al fondo del segundo div.

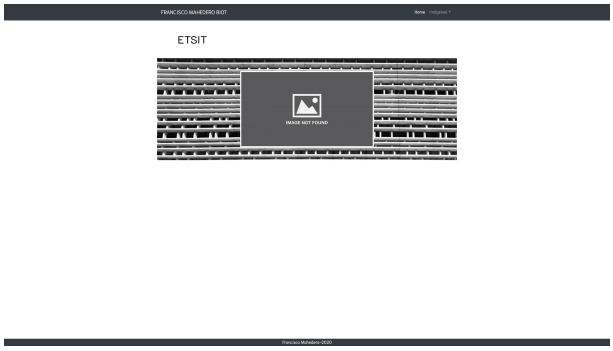

Figura 5.6: Página de las imágenes de la página web

Cada uno de los *div* y elementos que conforman las páginas contienen unas clases y id que mediante la hoja de estilos css dan forma visual a la página. En este caso se ha usado BootStrap, que permite dar una forma visual a la aplicación web de forma rápida, con sus clases y id ya predeterminados, con lo cual se ahorra tiempo de escritura.

5.4.2. JavaScript

En el front end se pueden apreciar dos tipos de archivos javascript: el encargado de crear la imagen en el div que le corresponde y el encargado de llevar a cabo las peticiones al backend para recibir las imágenes cada cierto tiempo mediante las peticiones Ajax. A la hora de realizar las peticiones Ajax se ha utilizado la librería JQuery de JavaScript.

5.4.2.1. crearImagen.js

Este archivo es el encargado de generar el elemento en HTML que permite visualizar las imágenes ya guardadas en el backend. La característica de la función de este archivo es la capacidad de poder ir reemplazando este elemento cada vez que se obtiene la URL de la imagen nueva.

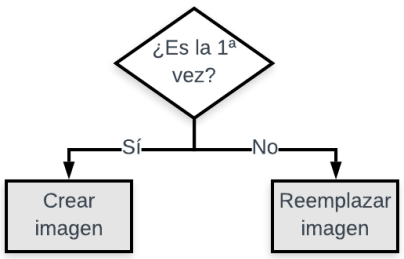

Figura 5.7: Algoritmo del funcionamiento de crearImagen.js

Esta función contiene un algortimo en el que si la función detecta que es la primera vez, se crea la imagen de forma normal. Si la función detecta que no es la primera vez, la función reemplaza este elemento HTML anterior por lo tanto no se crea una imagen nueva. El elemento HTML que se crea, es un elemento *img* con la clase de BootStrap *img-thumbnail*.

Si se crease una imagen nueva en todo momento, se crearía una sucesión de imágenes interminable y sería necesario reiniciar el proceso cada cierto tiempo.

5.4.2.2. Archivos Ajax

Los archivos encargados de las peticiones Ajax son: fotosAgora.js, fotosCasaAlum.js y fotosEtsit.js.

Cada uno de estos archivos estan escritos mediante la librería JQuery, que como ya se ha descrito en los fundamentos teóricos, es una librería de JavaScript. Estos archivos están constituidos por una petición Ajax que se repite cada cierto intervalo. El intervalo marcado es de 6 segundos.

Las peticiones Ajax son peticiones GET al back end. Cada petición GET se hace a la URL correspondiente. Como ejemplo, si la petición proviene de fotosAgora.js la petición se hará a la URL */agoraFoto*. La respuesta puede ser de dos maneras: satisfactoria si se ha podido llevar a cabo la función alojada en el archivo *crearImagen.js* o insatisfactoria y por lo tanto no se ha podido obtener la información. Cabe destacar que cuando no hay ninguna imagen, no se recibe un a respuesta de error sino que se recibe la imagen que indica que no se ha podido encontrar ninguna imagen.

El hecho de haber usado JQuery, permite escribir el código de forma más limpia que si se hace sin el uso de esta librería. Además, jQuery permite que la longitud del código se reduzca sustancialmente, lo cual aporta facilidad al leer el código en caso de encontrar un error en tal código.[8]

El código se puede consultar en Github, en la dirección: `https://github.com/FMBIoT/WEBSERVER-MQTT`.

Capítulo 6

Conclusiones y líneas futuras

Como se ha podido ver, se ha conseguido desarrollar un sistema para la compartición de imágenes dentro del paradigma IoT, lo cual permite al cliente supervisar en todo momento lo que ocurre en un lugar determinado. Este sistema ofrece la posibilidad de videovigilancia mediante imágenes en tiempo real haciendo uso del protocolo MQTT para poder coordinar de forma asíncrona el funcionamiento de varios dispositivos, en este caso cámaras.

El proyecto se ha centrado en esta función de videovigilancia pero se debe destacar, como ya se ha hecho mención en los objetivos, que este proyecto puede definirse como un producto con algunas modificaciones para utilizarlo con otros fines como podría ser videovigilancia o sensores de calor en bosques y el uso de una conexión LPWAN mediante LoRa. Este sistema es utilizable a nivel exterior de la red local, lo que permite que tenga una función real, es por eso que se lleva a cabo el redireccionamiento NAT.

Otro punto a destacar es el planteamiento inicial que permite que el proyecto en todo momento sea escalable en el tiempo, como la utilización de MQTT con un broker centralizado y el uso de Node js. En el futuro se seguirá trabajando en el proyecto para poder aportar valor de uso al cliente en una variante del sistema realizado.

Respecto al protocolo MQTT se ha podido observar que ha cumplido satisfactoriamente todas las necesidad que eran necesarias para llevar a cabo el sistema planteado en el campo de IoT. El protocolo ha permitido una comunicación fluida entre todos los elementos del sistema.

Para poder llevar a cabo el proyecto se han utilizado diferentes tecnologías y lenguajes de programación en los que se brindaban librerías que permitían llevar a cabo un cliente MQTT con facilidad. Esto demuestra que el protocolo MQTT es muy versátil y que se puede usar en muchos escenarios.

Finalmente, se ha usado como interfaz de usuario una aplicación web, la cual detrás de una mera presentación de cara al cliente (front end) contiene una función en segundo plano (back end) que permite que todo funcione correctamente. El desarrollo de esta aplicación permite valorar el trabajo que hay y entender como funcionan.

Respecto a las líneas futuras del proyecto, la optimización de la experiencia como usuario es vital. Empezando por el elemento inicial que es la cámara, esta podría ser sustituida por una de mejor calidad y por lo tanto mayor nitidez.También sería necesario el diseño de una envolvente para la cámara que permita dar cierta serguridad al hardware. En cuanto al software, crear una

apliación que permita cambiar configuración de la cámara, para dar flexibilidad en su uso, como la selección de la red Wifi o la calidad de la imagen a enviar.

Respecto al servidor Web, el desarrollo de un front end más elaborado para hacerlo atractivo al usuario también sería un buen punto sobre que el que trabajar, para poder así dar una buena experiencia de interacción con la aplicación web.

Como ya se ha nombrado anteriormente, una de las líneas futuras podría ser la utilización de LoRa y LPWAN para la transmisión de imágenes comprimidas. De este modo, en lugares donde fuese complicada la conexión directa con una red Wifi, la cual es cuestión de metros, se puede hacer mediatne RF con Lora, la cual se conecta a un gateway con conexión a internet a una distancia máxima de hasta 20km. De esta forma sería posible llevar a cabo ciertas tareas mínimas, debido a las limitaciones que LoRa tiene.

Otro aspecto en el futuro que se podría modificar es el broker, por dos motivos:

- Capacidad: aumento de la capacidad de procesamiento del broker cambiándolo por otro dispositivo más potente o la utilización de diversos broker que permita llevar a cabo un entramado que aumente esta capacidad.
- Eliminación: eliminar el broker como nodo central y llevar a cabo la comunicación entre los distintos dispositivos. Esto permitiría disminuir los fallos que se puedan producir en un proyecto centralizado.

Mediante estas mejoras, se busca pulir un producto que permita ser comercializado y que se pueda vender al usuario final. Así pues, se busca satisfacer al cliente en función a unas necesidad que el mercado demanda, siguiendo siempre la línea de los avances tecnológicos.

Bibliografía

[1] Deloitte. *¿Qué es IoT (Internet Of Things)? - Deloitte España.* [En línea]. Disponible en: https://www2.deloitte.com/es/es/pages/technology/articles/IoT-internet-of-things.html.

[2] *Eclipse Mosquitto™, An open source MQTT broker*. [En línea]. Disponible en: https://mosquitto.org/.

[3] IBM. *Conozca MQTT, ¿Por qué MQTT es uno de los mejores protocolos de red para Internet de las Cosas?* [En línea]. Disponible en: https://developer.ibm.com/es/articles/iot-mqtt-why-good-for-iot/.

[4] *About Node.js*. [En línea]. Disponible en: https://nodejs.org/en/about/.

[5] Spressif Systems. *ESP32 SERIES DATASHEET*. [En línea]. Disponible en: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[6] OpenSSL. *OpenSSL, Cryptography and SSL/TLS Toolkit*. [En línea]. Disponible en: https://www.openssl.org/docs/.

[7] *MQTT.js, API Documentation*. [En línea]. Disponible en: https://www.npmjs.com/package/mqtt.

[8] *JQuery.Ajax. API Documentation*. [En línea]. Disponible en: https://api.jquery.com/jquery.ajax/.