FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

CARRERA DE ESPECIALIZACIÓN EN
INTERNET DE LAS COSAS

MEMORIA DEL TRABAJO FINAL

Sistema de gestión remota de dispositivo conversor Modbus a MQTT.

Autor:
Ing. Domenje Carlos Ruben

Director:
Ing. Fernando Lichtschein (FIUBA)

Jurados:
Esp. Ing. Sebastian Guarino (FIUBA)
Mg. Ing. Matías Álvarez (FIUBA)
Mg. Ing. Lucas Dórdolo (FIUBA)

Este trabajo fue realizado en la ciudad de Avellaneda, Santa Fe,
entre mayo de 2020 y abril de 2021.

Resumen

La presente memoria describe el diseño de una plataforma web de gestión para un dispositivo conversor de protocolo Modbus a MQTT, el cual permite el envío de datos en tiempo real a un servidor, almacenamiento en una base de datos y generación de eventos configurables por el usuario. Este desarrollo se realizó teniendo en cuenta la importancia de poder visualizar datos de forma remota y en cualquier dispositivo móvil o PC que pueda ejecutar un navegador web.

Para realizar este trabajo se aplicaron conceptos de gestión de proyectos y arquitectura de protocolos de internet. En cuanto al desarrollo de la base de datos, servidor y página web se utilizaron técnicas de arquitectura y gestión de grandes volúmenes de datos, desarrollo de aplicaciones multiplataforma, ciberseguridad y testing de sistemas de internet de las cosas.

Agradecimientos

A mis padres, que me dieron la posibilidad de estudiar y me han brindado su apoyo a lo largo de mi vida.

A Maria Inés, que me ha acompañado durante el cursado, por su apoyo, ayuda y comprensión.

A mis compañeros y profesores de la Especialización en Internet de las Cosas por compartir sus conocimientos y por su acompañamiento a lo largo del año.

Al Ing. Fernando Lichtschein, por su orientación, seguimiento, supervisión y aportes durante la realización del presente trabajo.

A todos ellos, muchas gracias.

Índice general

Índice de figuras

Índice de tablas

Dedicado a mis socios, Ivan y Luis de D&T

Capítulo 1

Introducción general

En este capítulo se realiza una introducción al protocolo Modbus y la vinculación con la internet de las cosas. Asimismo, se mencionan algunos sistemas en el mercado, y por último se explica la motivación, alcance y objetivos del presente trabajo.

1.1. Comunicación Modbus y supervisión SCADA

Gracias al desarrollo e innovación de nuevas tecnología y a la automatización de procesos industriales a través del tiempo, se ha dado lugar a avances significativos que le han permitido a industrias implementar procesos de producción eficientes, seguros y competitivos.

Actualmente en el sector industrial es muy utilizado el PLC (*Programmable Logic Controller*) [1] para controlar máquinas, lineas de producción y en general todo lo relacionado con el proceso propio de la industria y, para que esto sea posible, los equipos, sensores y actuadores deben comunicarse entre si. Con este fin se utiliza, entre otros, el protocolo de comunicacion Modbus [2], introducio por la empresa Modicon en 1979.

Modbus permite conectar un dispositivo servidor con varios dispositivos clientes. Existen dos versiones de implementación:

- Interfaz serie (RS-232 y RS-485) llamada Modbus serie.
- Interfaz Ethernet llamada Modbus TCP.

Modbus es un protocolo de mensajería de capa de aplicación, posicionado en el nivel 7 del modelo OSI [3]. La comunicación está basada en un protocolo de solicitud / respuesta, donde es siempre iniciada por el servidor y, por lo tanto, los clientes nunca transmitirán datos sin una solicitud previa.

Las solicitudes desde el nodo servidor a los clientes pueden ser realizados en dos modos:

- Modo unicast: en este modo, el servidor direcciona a un esclavo.
- Modo broadcast: en este modo, el servidor envía una solicitud a todos los esclavos simultáneamente, sin recibir respuesta.

Este protocolo permite comunicarse de manera sencilla con todo tipo de arquitecturas de red como puede observarse en la figura 1.1, y además, con el uso de *gateways* se puede interactuar entre varios tipos de buses o redes.

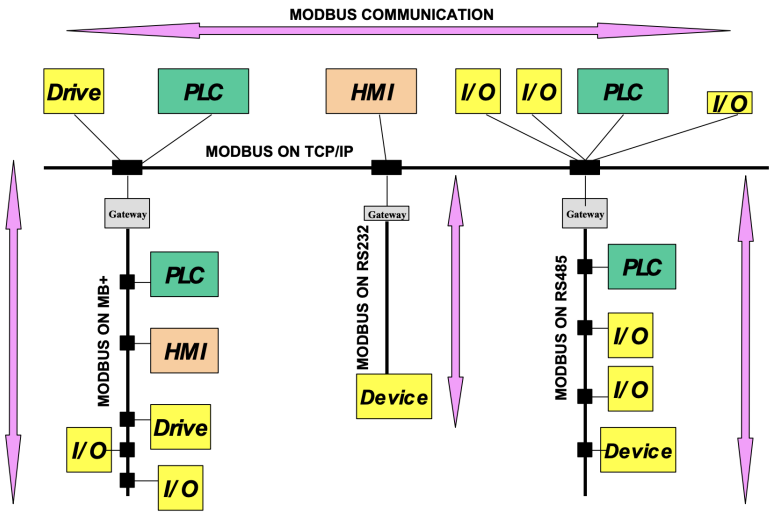

FIGURA 1.1. Ilustración de diferentes arquitecturas de red utilizando el protocolo Modbus[1].

Para la supervisión y control de un proceso industrial que utiliza Modbus como protocolo de comunicación, se utilizan sistemas HMI (*Human Machine Interface*) y hacen referencia a la manera que interactúa el humano con las diferentes máquinas que componen el sistema.

Se trata de un sencillo panel que transmite órdenes, visualiza resultados de manera gráfica y obtiene visualización del estado del proceso o la máquina en tiempo real.

A modo de ejemplo se pueden observar en la figura 1.2 diferentes modelos de HMI de la empresa Siemens.

FIGURA 1.2. Diferentes modelos de HMI de la empresa Siemens[2].

En un nivel superior de gestión del proceso industrial antes mencionado, se encuentra el sistema SCADA, cuya palabra es un acrónimo que deriva del inglés *Supervisory Control And Data Acquisition* y su significado en español se traduce como Control Supervisor y Adquisición de Datos.

Los sistemas SCADA son programas de software que se utilizan para gestionar y controlar sistemas remotos o locales mediante el uso de una interfaz gráfica que comunica al usuario con el programa. Cuentan con una estructura que parte de

[1] https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf

[2] https://new.siemens.com/global/en/products/automation/simatic-hmi/panels/basic-panels.html

sus controladores lógicos programables (PLC) o unidades de terminal remotas (RTU)[4], es decir, de microordenadores que se comunican con múltiples objetos, ya sean máquinas, dispositivos, sensores o HMI. Estos microordenadores después de comunicar, envían la información desde estos objetos a los ordenadores con el software SCADA.

Por lo tanto, el sistema SCADA procesa, distribuye y muestra los datos, permitiendo a los operadores y otros trabajadores realizar un análisis para facilitar la toma de decisiones. Entre sus principales características, podemos destacar:

- Supervisar remotamente las instalaciones y equipos.
- Monitorizar y controlar las operaciones en tiempo real.
- Procesar datos que hagan más fácil la toma de decisiones.
- Mostrar a través de imágenes dinámicas el comportamiento de los procesos.
- Arrojar señales de alarma (visuales o sonoras) frente a imprevistos.

En la figura 1.3 se puede observar la implementación de un sistema SCADA realizado con el software WINCC [5] de la empresa Siemens para el monitoreo y gestión de energía eléctrica.

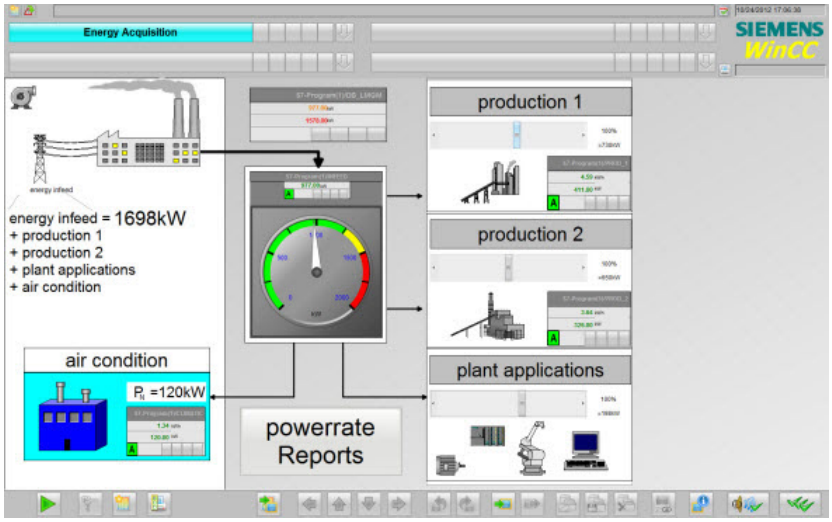

FIGURA 1.3. Ejemplo de implementación de un sistema SCADA con el software WINCC de la empresa Siemens para monitoreo y gestión de energía eléctrica[3].

1.2. Internet de las cosas

Por lo general, el término Internet de las cosas (IoT) se refiere a escenarios en los que la conectividad de red y la capacidad de cómputo se extienden a objetos, sensores y artículos de uso diario que habitualmente no se consideran computadoras, permitiendo que estos dispositivos generen, intercambien y consuman datos con una mínima intervención humana.

[3] https://support.industry.siemens.com/cs/document/65384955/gestión-de-energía

El concepto de combinar computadoras, sensores y redes para monitorear y controlar diferentes dispositivos ha existido durante décadas. Sin embargo, la reciente confluencia de diferentes tendencias del mercado tecnológico está permitiendo que la internet de las cosas esté cada vez más cerca de ser una realidad generalizada.

Estas tendencias incluyen la conectividad omnipresente, la adopción generalizada de redes basadas en el protocolo IP [6], la economía en la capacidad de cómputo, la minimización, los avances en el análisis de datos y el surgimiento de la computación en la nube.

Los beneficios que internet de las cosas ofrece tanto a empresas como a personas individuales son numerosos. Desde un punto de vista económico, permite a las empresas gestionar y controlar mejor sus procesos de forma remota y en tiempo real, aumentando su eficiencia y posibilitando la toma de mejores decisiones y acciones preventivas como el mantenimiento de la maquinaria o sistemas instalados. En definitiva, esto conlleva un ahorro de costos y tiempo para la organización. En el caso de individuos particulares, los beneficios se traducen en mejoras de su calidad de vida y comodidad en el día a día mediante automatizaciones en su entorno cotidiano.

Las implementaciones de la IoT utilizan diferentes modelos de conectividad, cada uno de los cuales tiene sus propias características. Los cuatro modelos de conectividad descritos por la Junta de Arquitectura de Internet (IAB)[7] se mencionan en la siguiente lista:

- *Device-to-Device* (dispositivo a dispositivo).
- *Device-to-Gateway* (dispositivo a puerta de enlace).
- *Device-to-Cloud* (dispositivo a la nube).
- *Back-End Data-Sharing* (intercambio de datos a través del backend).

Estos modelos destacan la flexibilidad en las formas en que los dispositivos de la IoT pueden conectarse y proporcionar un valor para el usuario.

La irrupción de la Internet de las cosas en la industria, tiene como objetivo conseguir una apertura total en la conectividad, es decir, la interconexión de todos los agentes que intervienen en la cadena del proceso productivo. Los dispositivos IoT serán capaces de capturar y analizar los datos obtenidos para tomar decisiones en tiempo real o enviarlos a la nube para almacenarlos y analizarlos mediante técnicas de *big data* e inteligencia artificial. Los empleos mas comunes en la industria pueden ser:

- Obtención de valores de parámetros físicos y actuación sobre máquinas.
- Toma de datos para mantenimiento predictivo.
- Control de la eficiencia energética mediante sensores.
- Automatización de procesos manuales y obtención de datos relevantes a través de sistemas embebidos.
- Comunicación Hombre – Máquina, notificaciones de alarmas, visualización de datos en tiempo real mediante *wearables*.

La importancia de la internet de las cosas en el mundo de las empresas y las personas queda reflejado en el crecimiento exponencial de dispositivos IoT conectados en el mundo, como se visualiza en la figura 1.4. Para este año, se prevé que cerca de 36 mil millones de dispositivos IoT se encuentren en funcionamiento en el mundo, cifra que se extiende hasta los más de 75 mil millones para el año 2025.

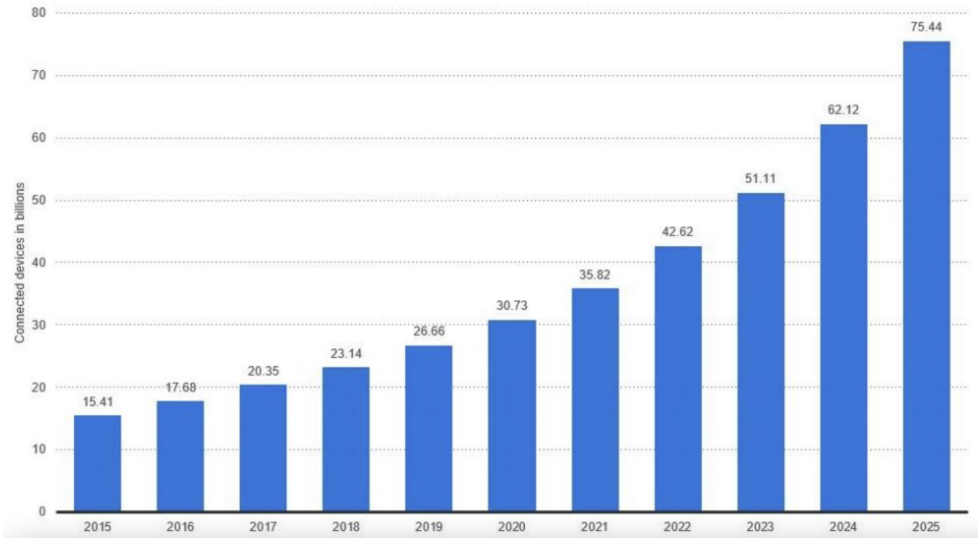

FIGURA 1.4. Crecimiento exponencial de dispositivos conectados en el mundo[4].

1.3. Estado del arte

En la actualidad existen varias plataformas cloud que brindan un conjunto de servicios de computación en la nube para hacer frente a las necesidades del negocio en lo que respecta al desarrollo de aplicaciones, almacenamiento y cómputo.

Algunas de las que se encuentran consolidadas a nivel mundial son: Google, Microsoft, Amazon, Samsung, entre otras [8]. Estos sistemas, si bien tienen toda la infraestructura realizada para conectar dispositivos IoT, requieren de desarrollo web que implica la programación de toda la estructura de la información que se requiere enviar a la nube. Por otro lado, no todas las empresas implementan como protocolo de aplicación a MQTT [9]. Se resumen algunos servicios ofrecidos por estas empresas en la tabla 1.1.

TABLA 1.1. Comparación de servicios cloud en el mercado.

Empresa	**Servicios Cloud**	**Protocolo de aplicación**	**Sistema operativo**
Google	Google Cloud	Weave	Linux
Amazon	AWS IoT	MQTT	Linux
Microsoft	Azure IoT	AMQP	Windows IoT
Samsung	SmartThings	MQTT	Linux

[4] https://www.iotworldonline.es/las-grandes-estadísticas-del-internet-de-las-cosas-iot/

En cuanto a plataformas que ofrecen la etapa de procesamiento, análisis y visualización de datos, se pueden mencionar algunas como ThingBoard, Ubidots, Node-Red, entre otras. Estas plataformas asumen que el cliente cuenta con el hardware necesario para enviarles la información relevante, y ellas se encargan de analizarla y presentarla de manera conveniente mediante tableros o dashboards, que le permiten al usuario ver el estado e históricos de sensores, actuadores y nodos.

Las principales desventajas de estas plataformas son la fuerte dependencia que generan y la poca adaptabilidad al hardware que se requiere gestionar, además la mayoría de ellas brindan servicios con costos monetarios elevados.

1.4. Motivación

En la actualidad existen múltiples dispositivos conversores de protocolos que permiten enviar datos a la nube, pero la mayoría no posee un sistema de gestión a través de una plataforma web. La metodología de conexión es a través de la vinculación con servicios web como los mencionados en la sección 1.3 donde el usuario deberá encargarse de realizar todo el desarrollo de la visualización de datos en forma clara, el almacenamiento de datos en una base de datos y la generación de eventos que puedan serle útil.

Empresas multinacionales como ser ABB, Scheider Electric, Honeywell o Siemens poseen sus propias plataformas web de gestión y protocolos de comunicación, lo que genera un sistema cerrado y dependiente de un determinado fabricante de dispositivos. Por lo antes mencionado, se hace inviable la conexión de conversores de otras marcas.

Por estos motivos, surgió la posibilidad de ofrecer un servicio de gestión web. Que contenga todos los componentes necesarios para que el usuario solo requiera de una configuración básica de conexión al servidor. Que pueda visualizar los datos requeridos de forma remota, pudiendo generar reportes y almacenamiento de datos en breve periodo de tiempo de utilización.

1.5. Alcances y objetivos

1.5.1. Objetivos

El propósito de este trabajo fué el desarrollo de un sistema que contenga un servidor MQTT, una base de datos para alojar información de reportes de dispositivos y una plataforma web que permita visualizar de forma remota datos enviados de diferentes sensores que contengan instalado un conversor de protocolo Modbus a MQTT. Esto permite maximizar la eficiencia del proceso, ya que puede ser monitoreado en tiempo real desde cualquier parte del mundo, contando con avisos de alarmas, reportes históricos y almacenamiento de datos.

1.5.2. Alcance

Para la realización de este trabajo se propuso desarrollar una plataforma web operativa de un sistema de gestión para conversores de protocolo Modbus a MQTT que se conectan a sensores de temperatura fabricados por la empresa D&T [10]. El presente desarrollo incluye los siguientes aspectos:

- Implementación de servidor MQTT.
- Implementación de servidor para gestión de usuarios y dispositivos.
- Implementación de base de datos no relacional.
- Visualización de datos en tiempo real en una plataforma web.
- Lectura de datos provenientes de sensores de temperatura que contenga un conversor de protocolo Modbus a MQTT.

Capítulo 2

Introducción específica

2.1. Funcionamiento general del sistema

En este capitulo se exponen las caracteristicas del software utilizado para la programacion del sistema, partiendo del backend hasta el frontend. Tambien se identifican los modulos de funcionamiento del dispositivo conversor Modbus a MQTT para utilizar en el trabajo. Finalmente se describe la utilzacion Nginx como servidor de datos para la integracion del sistema.

2.2. Dispositivo conversor Modbus a MQTT

El conversor consta de un circuito electronico que actúa de interfaz de conexión entre aparatos o dispositivos que se encuentran en una red Modbus y posibilita la conversion de datos al protocolo MQTT a traves de una conexion .

En la figura 2.1 se puede observar el diagrama de bloques con las partes mas importantes que componen al dispositivo.

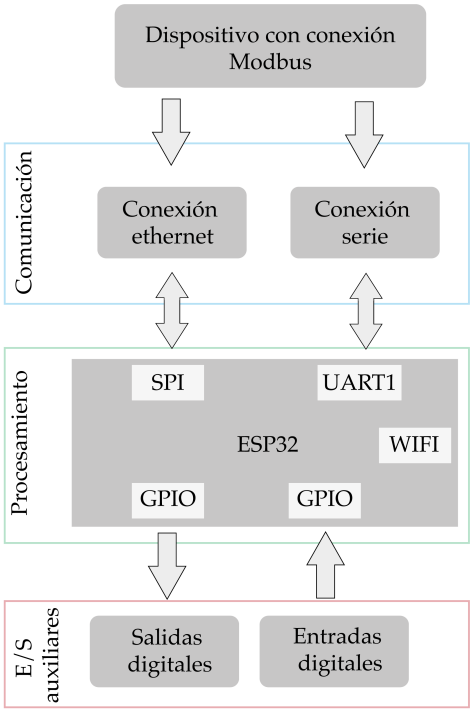

FIGURA 2.1. Diagrama de bloques de dispositivo conversor Modbus a MQTT

Al sistema se lo puede separar en tres grupos:

- Comunicación: este bloque está compuesto por un circuito de conexión ethernet para poder recibir mensajes de dispositivos que están conectados a través de Modbus TCP.

  Por otro lado el hardware contiene un circuito de conexión para dispositivos que se conectan por Modbus serie.

- Procesamiento: es el bloque principal donde se gestionan todos los datos recibidos por parte del módulo de comunicación y convierte los mensajes Modbus para luego poder ser enviados por MQTT a traves de wifi.

  Este módulo está constituido por un microcontrolador ESP32 [11] de la empresa Espressif [12] quien se encarga de controlar el modulo de comunicacion y el modulo de entradas y salidas auxiliares.

  Los datos provenientes del módulo de comunicación son reconvertidos a un formato de datos del tipo JSON [13] para ser enviados via wifi por protocolo MQTT.

- Entradas y salidas auxiliares: este bloque permite conectar dispositivos externos al conversor.

  Como ejemplo se pueden mencionar sensores para las entradas y actuadores para las salidas.

2.3. Protocolos de comunicación

En el caso de la comunicación con un servidor remoto, se requiere tener acceso a Internet y por lo tanto es necesario conectarse también con un módem, que a su vez debe estar conectado a un proveedor de servicios de Internet (ISP, por sus siglas en inglés correspondientes a *Internet Service Provider*).

Necesariamente para poder conectarse a Internet, el conversor de protocolo Modbus a MQTT debe implementar el protocolo TCP/IP[14] o UDP/IP,[15] que determinan las características de las capas de transporte y red del estándar OSI [16] respectivamente. La capa de aplicación, ubicada en la parte superior del modelo, es la encargada de ofrecer a las aplicaciones de usuario la posibilidad de comunicarse con otros dispositivos a través de los servicios brindados por las demás capas. Entre los protocolos de aplicación, existen múltiples como AMQP, CoAP, DDS, STOMP, MQTT y HTTP, siendo estos dos últimos los utilizados en el trabajo realizado.

Ambos protocolos son ampliamente utilizados para aplicaciones en la internet de las cosas. MQTT se basa en un modelo de publicaciones y suscripciones, en el que un cliente publica mensajes en un tema o tópico, y todos aquellos nodos que se encuentran suscriptos a ese tema reciben el mensaje publicado. MQTT es ideal para aplicaciones de IoT, debido principalmente a que requiere un muy bajo ancho de banda, tiene un menor consumo de potencia que otras alternativas, y además es sencillo y ligero de implementar.

Por otro lado, HTTP [17] (*Hypertext Transfer Protocol*) nos permite realizar una petición de datos y recursos, como pueden ser documentos HTML [18] y es la base de cualquier intercambio de datos en la web. La estructura esta basada en

cliente y servidor, esto quiere decir que una petición de datos es iniciada por el elemento que recibirá los datos (el cliente), normalmente un navegador web. Así, una página web completa resulta de la unión de distintos sub documentos recibidos, como ser un documento que especifique el estilo de maquetación de la página web, el texto, las imágenes, vídeos, scripts, entre otros.

Clientes y servidores se comunican intercambiando mensajes individuales. Los mensajes que envía el cliente, se llaman peticiones, y los mensajes enviados por el servidor se llaman respuestas.

2.3.1. Protocolo MQTT

MQTT o *Message Queue Telemetry Transport*, es un protocolo de transporte liviano de mensajes basado en un modelo de publicaciones y suscripciones, como se ilustra en la figura 2.2. El protocolo MQTT funciona sobre TCP/IP o sobre otros protocolos de red con soporte bidireccional y sin pérdidas de datos.

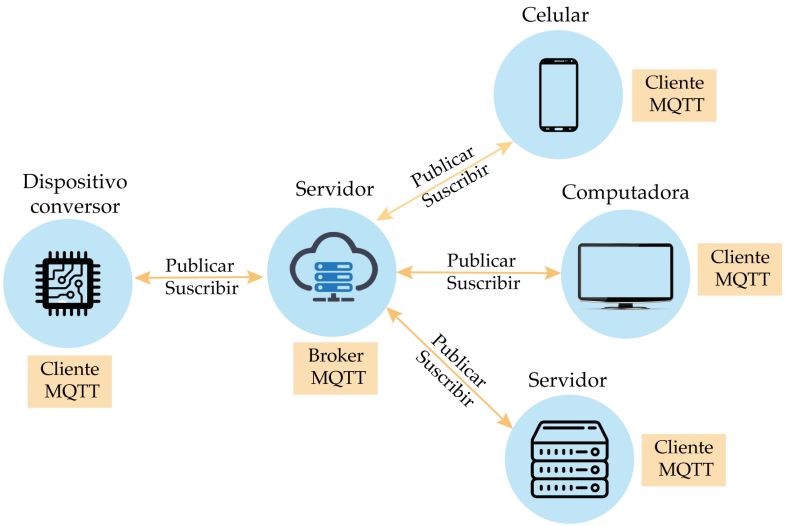

FIGURA 2.2. Modelo de funcionamiento de protocolo MQTT.

En modelos del tipo publicación y suscripción, un dispositivo puede publicar un mensaje en un tema o tópico y/o suscribirse a un tópico particular para la recepción de mensajes. A diferencia de un modelo cliente/servidor típico donde ambas partes se comunican directamente, en este esquema se desacopla tanto el cliente que envía un mensaje, como él o los clientes que están suscriptos a dicho tópico y reciben ese mensaje.

La conexión entre ambas partes es manejada por una tercera parte llamada broker MQTT. El broker MQTT es el responsable de recibir todos los mensajes, filtrarlos y distribuirlos según corresponda, es decir determinar qué cliente está suscripto a cada tópico y enviar los mensajes publicados en ellos a estos suscriptores.

La conexión en MQTT es siempre entre un cliente y el broker, como se ilustra en la figura 2.3 es decir que los clientes nunca se conectan entre ellos directamente.

Para iniciar una conexión, el cliente envía un mensaje CONNECT al broker y este responde con un mensaje CONNACK y un código de estado. Una vez que la conexión queda establecida, el broker la mantiene abierta hasta que el cliente envía

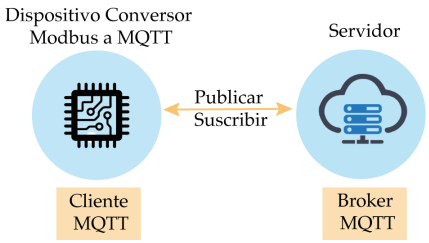

FIGURA 2.3. Esquema de conexión cliente - broker MQTT.

un comando de desconexión o la conexión se pierde por algun motivo. Los puertos estándar son 1883 para comunicación no encriptada y 8883 para comunicación encriptada usando SSL/TLS [19].

Durante el handshake SSL/TLS, el cliente valida el certificado del servidor para autenticarlo. El cliente puede también proveer un certificado al broker durante el handshake, que el broker utilizará para autenticar al cliente. Si bien no es parte de la especificación, se ha vuelto habitual que los brokers admitan la autenticación de clientes con certificados SSL/TLS. Debido a que el protocolo MQTT apunta a ser un protocolo para dispositivos de IoT con recursos limitados, SSL/TLS puede no ser siempre una opción y, en algunos casos, puede que no sea deseable. En tales casos, la autenticación se presenta como un nombre de usuario y una contraseña de texto simple que el cliente envía al servidor como parte de la secuencia de paquetes CONNECT/CONNACK. En el caso del presente trabajo, se optó utilizar autenticación mediante nombre de usuario y contraseña y certificado SSL para el cliente web.

Los mensajes son la información que se quiere intercambiar entre los dispositivos, ya sean comandos o datos. En MQTT la palabra tópico refiere a una cadena de caracteres que el broker utiliza para filtrar mensajes para cada cliente conectado. Los tópicos consisten en uno o más niveles. Cada nivel de tópico está separado por una barra. Existen algunos caracteres reservados que son utilizados como comodines o *wildcards* que permiten la suscripción a múltiples tópicos, como el carácter '+' y el carácter '#', que representan el *wildcards* de nivel único y de nivel múltiple, respectivamente. En cualquier caso, no es necesario que los clientes creen previamente el tópico antes de publicar o suscribirse a él. El broker acepta cada tópico válido sin previa inicialización.

Ejemplos de tópicos y uso de los wildcards son provistos en la figura 2.4:

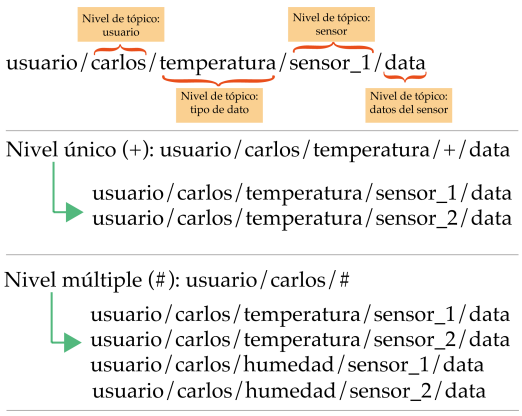

FIGURA 2.4. Ejemplo de tópicos y uso de *wildcards*.

2.3.2. Protocolo HTTP

Diseñado a principios de la década de 1990, HTTP es un protocolo ampliable, que ha ido evolucionando con el tiempo. Es lo que se conoce como un protocolo de la capa de aplicación, y se transmite sobre el protocolo TCP, o el protocolo encriptado TLS, aunque teóricamente podría usarse cualquier otro protocolo fiable. Gracias a que es un protocolo capaz de ampliarse, se usa no solo para transmitir documentos de hipertexto (HTML), si no que además se usa para transmitir imágenes o vídeos, o enviar datos o contenido a los servidores, como en el caso de los formularios de datos. HTTP puede incluso ser utilizado para transmitir partes de documentos, y actualizar páginas web en el acto.

Es un protocolo basado en el principio de cliente-servidor donde las peticiones son enviadas por una entidad llamada agente de usuario (del ingles *user agent*). La mayoría de las veces el agente de usuario o cliente, es un navegador web. Cada petición individual se envía a un servidor, el cuál la gestiona y responde. Entre cada petición y respuesta, hay varios intermediarios, normalmente denominados proxies [20] , los cuales realizan distintas funciones, como *gateways* o caches. En la figura 2.5 se puede visualizar un caso típico de aplicacion de utilizacion de proxy como *gateway*.

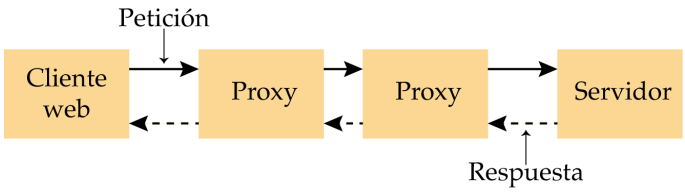

FIGURA 2.5. Diagrama de bloques ilustrando la utilización de proxy como *gateway*.

El navegador es siempre el que inicia una comunicación (petición), y el servidor nunca la comienza, por lo que para poder mostrar una página Web, el navegador envía una petición de documento HTML al servidor. Entonces procesa este documento y envía más peticiones para solicitar scripts, hojas de estilo y otros datos que necesite. El navegador, une todos estos documentos y datos y como resultado se obtiene la pagina web.

Al otro lado del canal de comunicación, está el servidor el cual "sirve"los datos que ha pedido el cliente. Un servidor conceptualmente es una unica entidad, aunque puede estar formado por varios elementos que se reparten la carga de peticiones (*load balancing*), u otros programas que gestionan otras computadoras como cache, bases de datos, servidores de correo electrónico, entre otros y que generan parte o todo el documento que ha sido pedido.

Una peticion de HTTP esta formada por los siguientes campos:

- un método HTTP: normalmente pueden ser un verbo, como: GET, POST o un nombre como OPTIONS o HEAD que defina la operación que el cliente quiera realizar. El objetivo de un cliente, suele ser una petición de recursos, usando GET, o presentar un valor de un formulario HTML, usando POST, aunque en otras ocasiones puede hacer otros tipos de peticiones.
- La dirección del recurso pedido: la URL del recurso.
- La versión del protocolo HTTP.

- Cabeceras HTTP opcionales que pueden aportar información adicional a los servidores.
- O un cuerpo de mensaje, en algún método como puede ser POST, en el cual envía la información para el servidor.

A modo de ejemplo, en la figura 2.6 se puede observar una peticion GET al servidor utilizado en este trabajo.

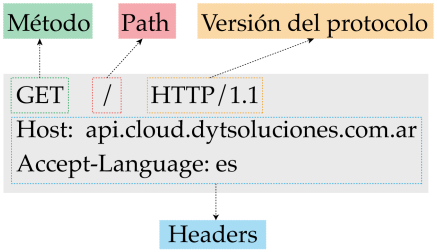

FIGURA 2.6. Ejemplo de petición GET al servidor utilizado para realizar este trabajo.

Por otro lado, las respuestas estan formadas por los siguientes campos:

- La versión del protocolo HTTP que están usando.
- Un código de estado, indicando si la petición ha sido exitosa, o no, y debido a que.
- Un mensaje de estado, una breve descripción del código de estado.
- Cabeceras HTTP, como las de las peticiones.
- Opcionalmente, el recurso que se ha pedido.

En la figura 2.7 se observa un modelo de respuesta para ejemplificar los campos mas importantes:

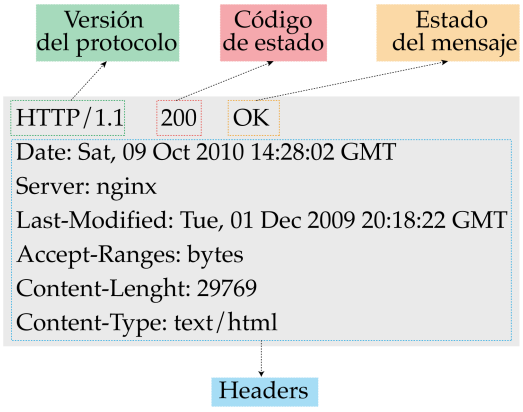

FIGURA 2.7. Ejemplo de modelo de respuesta HTTP del servidor.

2.4. Infraestructura del backend

En informática, la expresíon backend hace referencia a la parte de la infraestructura de datos de un sistema que se encuentran generalmente en el servidor y se encarga de procesar y almacenar informacion. Las funciones mas importantes del backend son las siguientes:

- Acceder a la información que se pide, a través de la plataforma web: cuando se usa una aplicacion web, se solicita informacion de manera continua, esto implica que el sistema tiene que ser capaz de encontrar y acceder a lo solicitado a traves de funciones de consulta.
- Combinar la información encontrada y transformarla: una vez encontrada, el backend combina la información para que resulte útil al usuario.
- Devolver la información al usuario: finalmente, el backend envía la información relevada de vuelta al usuario.

En este trabajo el backend consiste en un servidor, una aplicacion y una base de datos y debe cumplir con los siguientes criterios:

- Escalabilidad: hace referencia a la flexibilidad del mismo para integrar nuevas estructuras y códigos si el futuro la aplicacion lo requiere.
- Seguridad: debido a la constante interacción del backend con la base de datos, se debe hacer uso de conexiones seguras como HTTPS.
- Robustez: es la capacidad para funcionar en cualquier contexto ante situaciones inesperadas.

Para el desarrollo del backend del trabajo, se utilizó el *framework* Node.js [21] , el cual fue ideado como un entorno de ejecucion de JavaScript orientado a eventos asíncronos. Está diseñado para crear aplicaciones de red escalables.

Además de la alta velocidad de ejecución, Node.js dispone del bucle de eventos (*Event Loop*), que permitirá gestionar enormes cantidades de clientes de forma asíncrona. Tradicionalmente para trabajar de forma asíncrona las aplicaciones se valían de la programación basada en hilos (*programming threaded applications*), pero esto supone la utilización de un espacio de memoria que va escalando a medida que la cantidad de clientes conectados a la aplicación aumenta, por lo tanto si se necesita gestionar grandes cantidades de conexiones habrá que ampliar el número de servidores.

Para el almacenamiento de los datos se utilizó una MongoDB [22] como base datos. MongoDB es una base de datos de documentos que ofrece una gran escalabilidad y flexibilidad, y un modelo de consultas e indexación avanzado. El modelo de documentos de MongoDB resulta muy facil de aprender y usar, y proporciona todas las funcionalidades que se necesitan para satisfacer los requisitos mas complejos a cualquier escala. Almacena datos en documentos flexibles similares a JSON, por lo que los campos pueden variar entre documentos y la estructura de datos puede cambiarse en el tiempo.

2.4.1. Node.js

Node.js, surge en 2009 como respuesta a algunas necesidades encontradas a la hora de desarrollar sitios web, específicamente el caso de la concurrencia y la velocidad. Es un *framework* para implementar operaciones de entrada y salida basado en eventos, *streams* y construido por encima del motor de Javascript V8[23], que es con el que funciona el Javascript de Google Chrome. Este motor utiliza el código JavaScript y lo convierte en un código de máquina más rápido. El código de máquina es un código de nivel más bajo que la computadora puede ejecutar

sin necesidad de interpretarlo primero, ignorando la compilación y por lo tanto aumentando su velocidad.

Node.js utiliza un modelo de solicitudes y respuestas sin bloqueo controlado por eventos que lo hace ligero y eficiente. Puede referirse a cualquier operación, desde leer o escribir archivos de cualquier tipo hasta hacer una solicitud HTTP.

La finalidad de Node.js no tiene su objetivo en operaciones intensivas del procesador sino en la creación de aplicaciones de red rápidas, ya que es capaz de manejar una gran cantidad de conexiones simultáneas con un alto nivel de rendimiento, lo que equivale a una alta escalabilidad.

En comparación con las técnicas tradicionales de servicio web donde cada conexión genera un nuevo subproceso, ocupando la RAM del sistema y regularmente maximizando la cantidad de RAM disponible, Node.js opera en un solo subproceso, utilizando el modelo entrada y entrada sin bloqueo de la salida, lo que le permite soportar decenas de miles de conexiones al mismo tiempo mantenidas en el bucle de eventos.

Cuando hay una nueva solicitud se genera un tipo de evento. El servidor empieza a procesarlo y, cuando hay una operación de bloqueo de solicitud y respuesta, no espera hasta que se complete y en su lugar crea una función de devolución de llamada o *callback function*. El servidor comienza en el acto a procesar otro evento y cuando finaliza la operación de solicitud y respuesta, continuará trabajando en la solicitud ejecutando la devolución de llamada tan pronto como tenga tiempo.

Para comprender el funcionamiento general de Node.js se puede observar la figura 2.8 el proceso de ejecución que realiza el servidor.

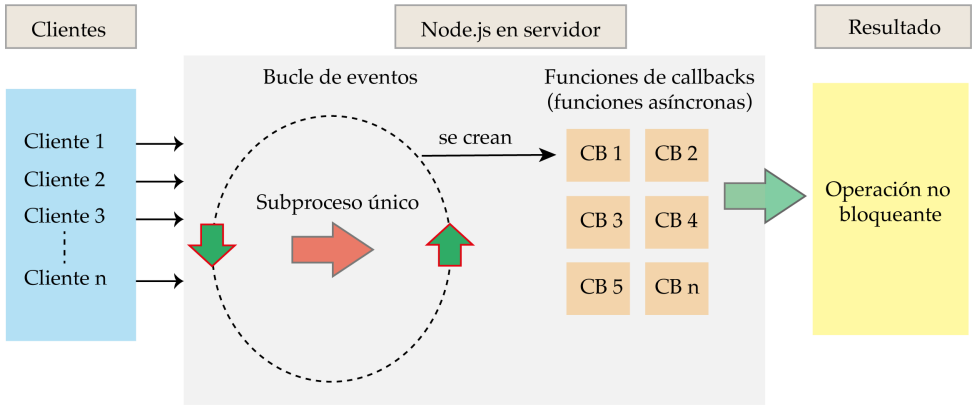

FIGURA 2.8. Ilustración del proceso de ejecucion que realiza Node.js en un servidor

Cuando un sistema síncrono ejecuta una llamada, las instrucciones posteriores a esa llamada no se ejecutan hasta que esta ha sido completada. Node.js es un sistema asíncrono. Esto significa que las llamadas y métodos son ejecutados de forma secuencial, pero sin esperar a que la anterior llamada haya finalizado como se muestra en la figura 2.9. Como conclusión podemos observar que se reduce considerablemente el tiempo de ejecución.

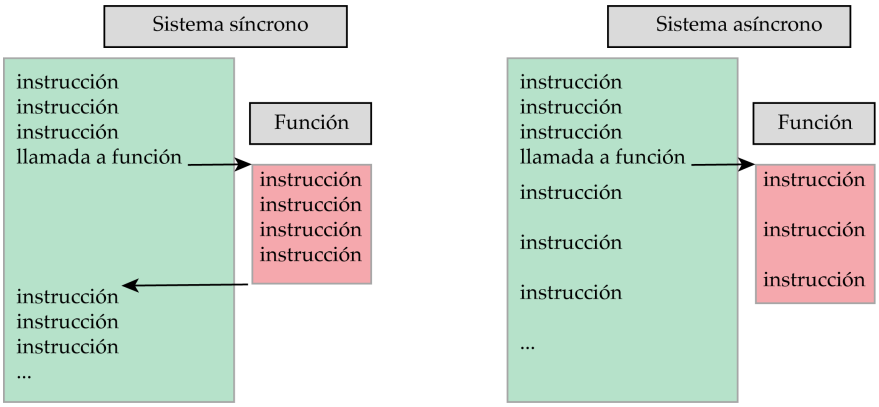

FIGURA 2.9. Ilustración de un sistema síncrono y un sistema asíncrono

2.4.2. Base de datos MongoDB

MongoDB es una base de datos noSQL[24], donde se puede agregar informacion en forma de documentos en vez de registros como es el caso de las bases de datos SQL[25]. La principal ventaja de esta base de datos noSQL es la velocidad de consulta, el motivo de esto es porque la información se la almacena en archivos de formato BSON que son versiones modificadas de JSON.

Las caracteristicas principales de MongoDB son:

- Escalabilidad horizontal: MongoDB está diseñado para escalar de manera ilimitada a lo que otras bases de datos no podrían, a través de su replicación WEBSITE:24 y su *sharding*[26] se puede almacenar información de varios equipos conectados entre sí.
- Consultas Ad hoc: permite la consulta de información a través de búsquedas de campos, expresiones regulares con comandos de manera rápida y eficaz.
- Indexación: MongoDB utiliza indexación para aumentar la eficiencia en la búsqueda de información.
- Replicación: MongoDB puede realizar replicación maestro – esclavo. El maestro ejecuta comandos para la lectura y escritura de los datos mientras que el esclavo realiza solo lectura de datos pero no los puede modificar.
- Distribución de carga: MongoDB permite que pueda ejecutarse en distintos servidores a la vez, haciendo más fácil su distribución en la carga de usuarios que deseen conectarse así como acceder a la información.

Algunos comandos para el manejo de MongoDB se pueden observar en la tabla 2.1. Normalmente en las bases de datos, se mencionan operaciones CRUD, del acronimo de Çrear, Leer, Actualizar y Borrar"(del original en ingles *Create*, *Read*, *Insert* y *Delete*) y se usa para referirse a las funciones basicas para el manejo de la información. Para el caso de MongoDB, estas palabras se reemplazan por *Insert*, *Find*, *Update* y *Remove* respectivamente.

MongoDB almacena los documentos creados en colecciones que son el análogo a tablas en base de datos relacionales. En la figura 2.10 se ejemplifica la analogía entre tablas en base de datos relacionales y colecciones en MongoDB.

TABLA 2.1. Comandos mas utilizados para el manejo de base de datos que utiliza MongoDB.

Comando	Descripción
use	Crear una nueva base de dato o utilizar una existente.
insertOne	Crear un documento en una coleccion determinada.
count	Cuenta la cantidad de documentos que hay en una colección.
drop	Elimina una coleccion de la base de datos.
find	Se busca un documento en una colección
remove	Elimina un documento de una colección
update	Modifica un documento de una coleccion

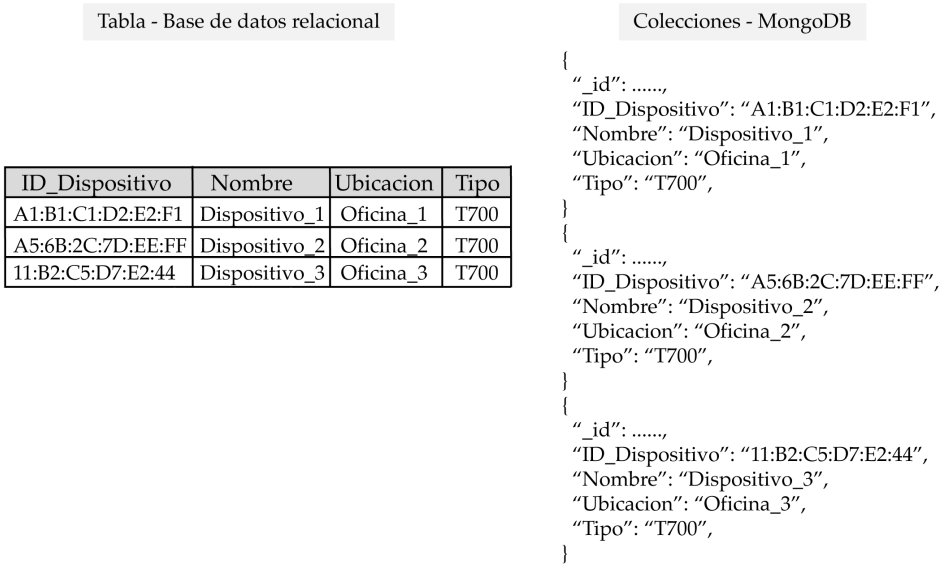

FIGURA 2.10. Ilustración de analogia entre tablas en base de datos relacionales y colecciones en MongoDB.

2.5. Infraestructura del frontend

Frontend es la parte de un programa o dispositivo a la que un usuario puede acceder directamente. Son todas las tecnologías de diseño y desarrollo web que se ejecutan en el navegador y que se encargan de la interactividad con los usuarios. Se desarrolla, principalmente, a través de tres lenguajes: HTML (*HyperText Markup Language*), CSS (*Cascading Style Sheets*)[27] y JS (Javascript)[28] . Cada uno de estos lenguajes se usa para desarrollar diferentes partes del frontend. Estos lenguajes de programacion se dividen en tres tareas basicas del desarrollo:

- Arquitectura: HTML es el componente mas importante de cualquier proceso de desarrollo de sitios web y proporciona un marco general de cómo se verá el mismo. Es un lenguaje de marcado que nos permite indicar la estructura de nuestro documento mediante etiquetas. Este lenguaje nos ofrece una gran adaptabilidad, una estructuración lógica y es fácil de interpretar tanto por humanos como por máquinas.
- Apariencia : CSS (en español Hoja de Estilos en Cascada) controla el aspecto de presentación del sitio, una vez que este ya está construido con HTML. Es un lenguaje de diseño gráfico para definir y crear la presentación de un

documento estructurado escrito en un lenguaje de marcado. Está diseñado principalmente para marcar la separación del contenido del documento y la forma de presentación de este.

- Interacción: JS. Es un lenguaje de programación basado en eventos que se utiliza para transformar una página estática en una interfaz dinámica interactuando con el usuario, el navegador y el servidor. JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript[29]. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Para el desarrollo del frontend de este trabajo se utilizó Angular[30] que es un *framework* de diseño eficiente y sofisticado de plataformas web.

2.5.1. Angular

Angular es una plataforma de desarrollo, construida sobre *Typescript*[31]. Como plataforma, Angular incluye:

- Un marco basado en componentes para crear aplicaciones web escalables.
- Una colección de bibliotecas bien integradas que cubren una amplia variedad de características, que incluyen enrutamiento, administración de formularios, comunicación cliente-servidor, entre otras.
- Un conjunto de herramientas para desarrolladores que layudan a desarrollar, compilar, probar y actualizar el código.

Para crear aplicaciones con Angular, se generan *templates* con HTML y se controlan estos mismos con lógica creada en los componentes, que serán exportados como clases. Así mismo, se agrega lógica en servicios para manejar datos que la aplicación tendrá y finalmente se .[en]capsulan"los componentes y servicios en módulos o *NgModules*.

Cuando se inicia la aplicación, lo hace desde el *root module*. Angular toma el control y muestra el contenido en el explorador web, reaccionando a la interacción de los usuarios que utilicen la app de acuerdo a las instrucciones que contiene la lógica programada.

Un módulo o *NgModule* declara un contexto de compilación para un conjunto de componentes. Puede asociar sus componentes con servicios, para formar unidades funcionales.

Cada aplicación generada con Angular cuenta con un *root module* o modulo de raíz llamado convencionalmente *AppModule*, el cual provee el mecanismo de arranque que inicia la aplicación. Una aplicación contiene varios módulos funcionales, además un módulo puede importar funcionalidades de otros módulos, y exportar sus propias funcionalidades.

Cada aplicación de Angular tiene al menos un componente. Al igual que el *root module*, existe el *root component* que conecta una jerarquía de componentes con el DOM (*Document Object Model*)[32]. Cada componente define una clase que contiene datos y lógica, y esta vinculada con el archivo HTML.

El binding a propiedades, o *property binding* en inglés, sirve para asignar un valor a una propiedad de un elemento de un *template*. Esa asignación podrá ser un valor

literal, escrito tal cual en el *template*, pero generalmente se tratará de un valor obtenido a través de una propiedad del componente, de modo que si el estado del componente cambia, también cambie la propiedad del elemento asignada en el template.

Por otro lado, cuando un usuario interactúa con un enlace, pulsa un botón, selecciona un evento de una lista desplegable o manipula el texto, el binding de eventos o *event binding* permite a una aplicacion de Angular ejecutar codigo y acciones cuando se produce un evento.

Todos los datos o lógica que no está asociada directamente a una vista y que requiera ser utilizada en diferentes partes de la aplicacion y entre diferentes componentes, puede ser escrita en un servicio. Al igual que un componente, los servicios son exportados como clases. Los servicios cuentan con el decorador @Injectable() que provee metadata que permite que los servicios sean inyectados en componentes como dependencias. *Dependency injection* o Inyección de Dependencias permite manejar las clases de los componentes de forma ligera y eficiente.

El esquema de la figura 2.11 ejemplifica en forma de bloques, el funcionamiento general de Angular.

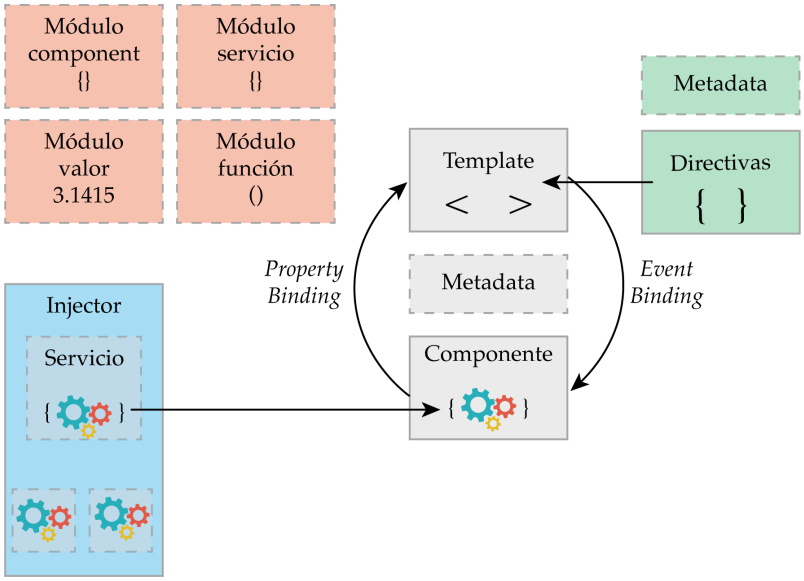

FIGURA 2.11. Ilustración de aquitectura de funcionamiento de Angular[1].

2.6. Servidor Nginx

Nginx [33] es un servidor web de código abierto que también es usado como proxy inverso, cache de HTTP, y balanceador de carga. Es un software modular, lo que significa que las diferentes características son presentadas en forma de módulos y, como administrador, pueden ser activadas o desactivadas. Como consecuencia, el usuario goza de las siguientes características:

- *Application Acceleration* (acelerador de aplicaciones), agiliza la entrega de contenidos.

[1]Imagen acondicionada de https://angular.io/guide/architecture

- Servidor proxy inverso para la aceleración web (HTTP, TCP, UPD) o como proxy de correo electrónico (IMAP, POP3, SMTP).
- Cifrado TLS para una transferencia de datos segura.
- Gestión de ancho de banda para un mejor rendimiento.
- Balanceo de carga con reorientación de solicitudes para disminuir la carga del servidor.

Nginx trabaja enfocado a eventos. Como consecuencia, puede procesar solicitudes de forma asíncrona, ahorrando memoria y espacio. Este software de servidor es soportado por una gran variedad de sistemas operativos, incluyendo variantes de UNIX / Linux, Mac OS o Windows, fue concebido inicialmente como una respuesta al problema C10K [34], que se refiere al problema de rendimiento de manejar 10,000 conexiones concurrentes.

Logra excelentes resultados en el procesamiento de un gran número de solicitudes de los clientes y aprovecha eficazmente los recursos.

Nginx es un servidor asíncrono construido buscando solucionar los problemas de concurrencia que experimentaban ciertos sitios. El algoritmo desarrollado para este servidor es mucho más eficiente y consume menos recursos.

Nginx genera procesos *worker*, cada uno de los cuales puede manejar muchas conexiones. Se puede lograr esto debido a la implementación de un mecanismo de bucle rápido que busca y procesa eventos continuamente. Cada conexión manejada por el *worker* es dispuesta dentro del bucle de eventos, donde vive con otras conexiones. Los eventos dentro de este bucle se procesan de forma asíncrona, permitiendo que el trabajo sea manejado de forma no bloqueante. Cuando una conexión se cierra, se elimina del bucle. Esta disposición asíncrona y monohilos ayuda a que,incluso con recursos limitados, Nginx sea bastante rápido en cuanto al procesamiento de conexiones. Por consiguiente, el uso de CPU y memoria tiende a bajar debido a la eficiencia en el manejo de conexiones. En la figura **??** se ejemplifica el proceso principal de Nginx con la creacion de n *workers* por cada consulta / respuesta.

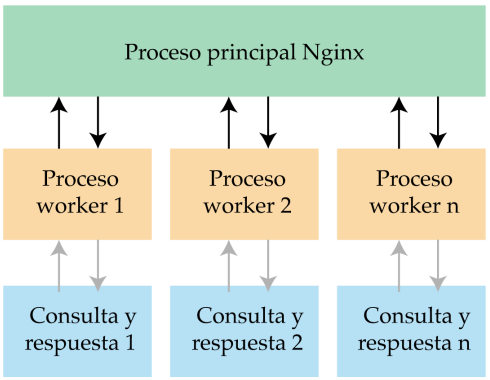

FIGURA 2.12. Ilustración de funcionamiento principal de Nginx con n consultas / respuestas

Para este trabajo se utilizo Nginx como proxy inverso. Se conoce como proxy inverso cuando un servidor acepta todo el tráfico y lo reenvía a un recurso especifico, por ejemplo a una consulta al backend o bien acceder al frontend. El motivo para utilizar un servidor proxy inverso es el agregado de seguridad al servidor

principal, restringir el acceso a rutas definidas y evitar ataques. En la figura 2.12 se observa la configuración típica de ngix como proxy inverso, donde se destaca que cada uno de los clientes tendrán un solo punto de acceso a los recursos del servidor.

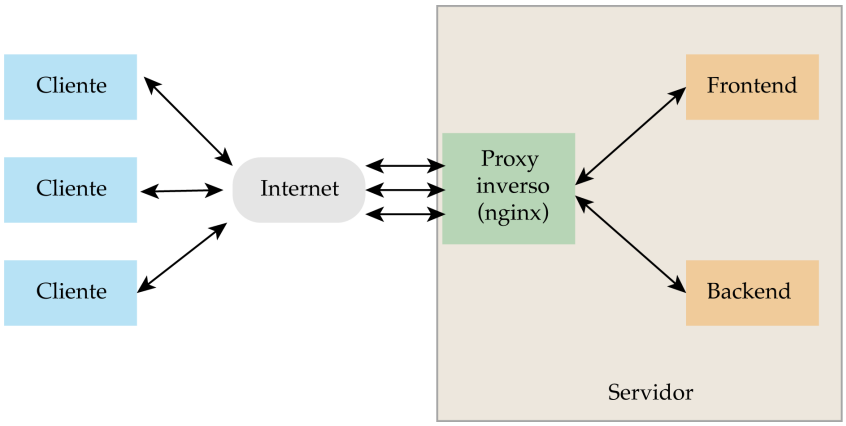

FIGURA 2.13. Ilustración de configuración de ngix como proxy inverso.

Capítulo 3

Diseño e implementación

3.1. Diseño de la estructura general del sistema

3.2. Implementación del backend

3.2.1. Software implementado en el servidor con NodeJS

3.2.2. Configuración del broker MQTT

3.2.3. Software implementado en la base de datos

3.2.4. Seguridad en el servidor

3.3. Implementación del frontend

3.3.1. Diseño de plataforma web con Angular

3.3.2. Seguridad en el frontend

3.4. Implementación y configuración de Nginx

Capítulo 4

Ensayos y Resultados

4.1. Pruebas unitarias

4.1.1. Postman para comprobación de funciones del backend

4.1.2. Cypress para pruebas unitarias

4.2. Ensayo de integración y sistema

4.2.1. Pruebas en la creación de dispositivos

4.2.2. Pruebas de verificación de conectividad con el dispositivo conversor

4.2.3. Pruebas de verificación de almacenamiento de datos en la base de datos

Capítulo 5

Conclusiones

En este capítulo se presentan los aspectos más relevantes del trabajo realizado y se mencionan los pasos a seguir.

5.1. Trabajo realizado

Se desarrolló e implementó un sistema de gestión de dispositivos conversores de protocolo Modbus a MQTT conectados a sensores de temperatura. La plataforma web demostró ser útil para el análisis del comportamiento de los sensores frente a las variaciones diarias de uso y aplicación. A continuación se listan los logros destacados del trabajo final:

- Programación e implementación de software en el servidor de datos para la vinculación de dispositivos conversores.
- Implementación de certificados SSL para dotar de seguridad a todo el sistema.
- Desarrollo de base de datos para el almacenamiento histórico de datos enviados para su posterior análisis.
- Implementación de aplicación web para visualización, análisis y control de datos enviados por diferentes dispositivos conversores.
- Integración en la nube del sistema de gestión.

El grado de cumplimiento de los requerimientos fué como se tenia previsto durante la planificación ya que se pudo lograr integrar el sistema e instalarlo en un servidor remoto para realizar pruebas con clientes. Estos últimos se encuentran ensayando los dispositivos conversores de protocolo Modbus a MQTT conectados a sensores de temperatura para el monitoreo del funcionamiento de cámaras frigoríficas.

Fue necesario contratar un servicio de servidor en la nube y un servicio de hosting web para poder realizar las pruebas de forma remota y que diferentes personas puedan probar el sistema. Esto llevó a un pequeño atraso en el desarrollo ya que se debió estudiar nuevos conceptos de programación y configuración de estos servicios.

Durante el desarrollo de este trabajo final se aplicaron conocimiento adquiridos a lo largo de todo el año de la Especialización en Internet de las Cosas. Todas las asignaturas cursadas aportaron conocimientos necesarios y experiencia para la práctica profesional en el área del desarrollo web. Sin embargo, se resaltan a continuación aquellas materias de mayor relevancia para este trabajo:

- Gestión de Proyectos: la elaboración de un plan de proyecto para organizar el trabajo final, facilito la realización del mismo y evitó demoras innecesarias.
- Protocolos de Internet: se aplicaron conceptos aprendidos para la programación del servidor con los protocolos MQTT y HTTP.
- Desarrollo de aplicaciones multiplataforma: se adquirieron conocimientos de programación para la plataforma web adaptable a cualquier dispositivo que pueda ejecutar un navegador web.
- Arquitectura de datos: se desarrolló la base de datos teniendo en cuenta las especificaciones y técnicas aprendidas.
- Ciberseguridad en IoT: se utilizaron técnicas de seguridad para proteger al sistema frente a posible ataques cibernéticos.
- Testing de Sistemas de Internet de las Cosas: se aplicaron los conocimientos adquiridos durante la materia, sobre todo en las áreas de testing unitarios y ensayos de integración del sistema.

5.2. Próximos pasos

Resulta imprescindible identificar el trabajo futuro, para dar continuidad al esfuerzo realizado hasta el momento y poder realizar un sistema comercialmente atractivo. A continuación se listan las líneas de trabajo más trascendentes:

- Desarrollo de notificaciones y alertas al usuario ante posible eventos configurables.
- Lectura de dispositivos que no pertenezcan a la medición de temperatura.
- Agregar un nivel de gestión de organizaciones para permitir a un usuario ser parte devarias de ellas.
- Contratación de un servicio de servidor remoto de produccion para asegurar la disponibilidad, integridad y utilización de recursos del software implementado.

Bibliografía

[1] https://www.se.com/. *Controladores PLC*. https://www.se.com/mx/es/product-category/3900-controladores-plc-y-pac. (Visitado 12-03-2021).

[2] https://modbus.org/. *Modbus Specifications and Implementation Guides.* https://modbus.org/specs.php. (Visitado 12-03-2021).

[3] https://standards.iso.org/. *ISO/IEC 7498-1*. https://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip. (Visitado 12-03-2021).

[4] Deon Reynders Gordon Clarke y Edwin Wright. *Practical Modern SCADA Protocols*. Newnes, 2003.

[5] https://siemens.com/. *The scalable and open SCADA system for maximum plant transparency and productivity*. https://new.siemens.com/global/en/products/automation/industry-software/automation-software/scada/simatic-wincc-v7.html. (Visitado 14-03-2021).

[6] https://www.rfc es.org/. *Internet Protocol*. https://www.rfc-es.org/rfc/rfc0791-es.txt. (Visitado 14-03-2021).

[7] https://www.iab.org//. *Internet Architecture Board*. https://www.iab.org/. (Visitado 13-03-2021).

[8] https://kinsta.com/. *Cuota de mercado de la nube – una mirada al ecosistema de la nube en 2021*. https://kinsta.com/es/blog/cuota-de-mercado-de-la-nube/. (Visitado 13-03-2021).

[9] https://mqtt.org/. *MQTT Specifications*. https://mqtt.org/mqtt-specification/. (Visitado 14-03-2021).

[10] https://www.dytsoluciones.com.ar. *D&T - Desarrollos para IoT*. https://www.dytsoluciones.com.ar/es/. (Visitado 14-03-2021).

[11] Espressif. *ESP32*. https://www.espressif.com/en/products/socs/esp32. (Visitado 16-03-2021).

[12] Espressif. *Pagina principal de Espressif*. https://www.espressif.com/. (Visitado 16-03-2021).

[13] json.org. *Introducción a JSON*. http://www.json.org/json-es.html. (Visitado 16-03-2021).

[14] https://www.ietf.org/. *SPECIFICATION OF INTERNET TRANSMISSION CONTROL PROGRAM*. https://tools.ietf.org/html/rfc675. (Visitado 17-03-2021).

[15] https://www.rfc es.org/. *PROTOCOLO DE DATAGRAMAS DE USUARIO*. "https://www.rfc-es.org/rfc/rfc0768-es.txt. (Visitado 17-03-2021).

[16] Hubert Zimmermann. *OSI Reference Model*. https://www.cs.cmu.edu/~srini/15-744/F02/readings/Zim80.pdf. (Visitado 16-03-2021).

[17] https://www.ietf.org/. *Hypertext Transfer Protocol – HTTP/1.1*. https://www.ietf.org/rfc/rfc2616.txt. (Visitado 16-03-2021).
[18] Sergio Luján-Mora. *Programación en Internet: clientes web*. Editorial Club Universitario, 2001.
[19] https://tools.ietf.org. *The Transport Layer Security (TLS) Protocol*. "https://tools.ietf.org/html/rfc5246. (Visitado 17-03-2021).
[20] https://www.welivesecurity.com. *Que es un proxy y para que sirve*. "https://www.welivesecurity.com/la-es/2020/01/02/que-es-proxy-para-que-sirve/. (Visitado 17-03-2021).
[21] https://nodejs.org. *Documentacion Node.js*. https://nodejs.org/es/docs/. (Visitado 18-03-2021).
[22] https://www.mongodb.com. *¿Qué es MongoDB?* https://www.mongodb.com/es/what-is-mongodb. (Visitado 18-03-2021).
[23] https://v8.dev/. *What is V8?* https://v8.dev/. (Visitado 18-03-2021).
[24] AKOB MATTSSON ADAM LITH. *Investigating storage solutions for large data*. https://publications.lib.chalmers.se/records/fulltext/123839.pdf. (Visitado 18-03-2021).
[25] Mike Chapple. *The Fundamentals of SQL*. https://www.lifewire.com/sql-fundamentals-1019780. (Visitado 18-03-2021).
[26] mongodb.org. *Sharding*. https://docs.mongodb.com/manual/sharding/. (Visitado 19-03-2021).
[27] Hakon W Lie. *Cascading HTML style sheets - a proposal*. https://www.w3.org/People/howcome/p/cascade.html. (Visitado 20-03-2021).
[28] https://developer.mozilla.org/. *About JavaScript*. https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript. (Visitado 20-03-2021).
[29] https://www.ecma international.org. *ECMA-262*. https://www.ecma-international.org/publications-and-standards/standards/ecma-262/. (Visitado 20-03-2021).
[30] https://angular.io/. *Angular*. https://angular.io/. (Visitado 20-03-2021).
[31] https://www.typescriptlang.org/. *TypeScript es JavaScript Escalable.* https://www.typescriptlang.org/. (Visitado 20-03-2021).
[32] https://lenguajejs.com/. *Que es el DOM?* https://lenguajejs.com/javascript/dom/que-es/. (Visitado 20-03-2021).
[33] https://nginx.com. *NGINX Documentation*. https://docs.nginx.com/. (Visitado 20-03-2021).
[34] http://www.kegel.com. *The C10K problem*. http://www.kegel.com/c10k.html. (Visitado 20-03-2021).