



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 María 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

Resumen	I
1. Introducción general	1
1.1. Comunicación Modbus y supervisión SCADA	1
1.2. Internet de las cosas	3
1.3. Estado del arte	5
1.4. Motivación	6
1.5. Alcances y objetivos	6
1.5.1. Objetivos	6
1.5.2. Alcance	7
2. Introducción específica	9
2.1. Dispositivo conversor Modbus a MQTT	9
2.2. Protocolos de comunicación	10
2.2.1. Protocolo MQTT	11
2.2.2. Protocolo HTTP	13
2.3. Infraestructura del backend	14
2.3.1. Node.js	15
2.3.2. Base de datos MongoDB	17
2.4. Infraestructura del frontend	19
2.4.1. Angular	19
2.5. Servidor Nginx	21
3. Diseño e implementación	25
3.1. Diseño de la estructura general del sistema	25
3.2. Implementación del backend	26
3.2.1. Modelos utilizados en la base de datos	29
3.2.2. Implementación CRUD para el manejo de usuarios	30
3.2.3. Configuración del broker MQTT	32
3.2.4. Seguridad en el servidor	32
3.3. Implementación del frontend	33
3.3.1. Diseño de plataforma web con Angular	34
3.4. Implementación y configuración de Nginx	48
4. Ensayos y Resultados	51
4.1. Pruebas unitarias	51
4.1.1. Pruebas de integridad del broker MQTT	51
4.1.2. Pruebas unitarias de funciones del backend	53
4.2. Ensayos de integración	57
4.2.1. Pruebas de integridad en login de usuarios	58
4.2.2. Pruebas de integridad en dispositivos conectados	62
4.2.3. Ejecución automática de tests realizados	66

5. Conclusiones	69
5.1. Trabajo realizado	69
5.2. Próximos pasos	70
Bibliografía	71

Índice de figuras

1.1.	Arquitectura de red Modbus	2
1.2.	Diferentes modelos de HMI de la empresa Siemens	2
1.3.	Implementación de sistema SCADA con WINCC	3
1.4.	Crecimiento de dispositivos IoT	5
2.1.	Diagrama de bloques de dispositivo conversor	9
2.2.	Esquema de funcionamiento MQTT	11
2.3.	Esquema de conexión cliente - broker MQTT.	12
2.4.	Ejemplo de tópicos y uso de <i>wildcards</i>	12
2.5.	Utilización de proxy como <i>gateway</i>	13
2.6.	Petición GET al servidor	14
2.7.	Modelo de respuesta HTTP	14
2.8.	Ejecución de Node.js en servidor	16
2.9.	Funciones síncronas y asíncronas	17
2.10.	Comparación tabla - colecciones en MongoDB	18
2.11.	Arquitectura de funcionamiento de Angular	21
2.12.	Proceso principal Nginx	22
2.13.	Configuración Nginx como proxy inverso	23
3.1.	Diagrama de bloques de sistema implementado	25
3.2.	Conexión entre Nginx - API clientes y base de datos	26
3.3.	Descripción bloque API clientes	27
3.4.	Descripción bloque cliente MQTT	28
3.5.	Modelos de datos utilizados en Mongoose	31
3.6.	Pantalla de login de usuario	34
3.7.	Pantalla de registro de usuario	35
3.8.	Estructura de componentes de login y registro	36
3.9.	Rutas del sistema	40
3.10.	Pantalla principal - <i>dashboard</i>	41
3.11.	Sensor de temperatura T700	42
3.12.	Pantalla de edición de dispositivos	43
3.13.	Componente gráfico de echarts	44
3.14.	Opciones de configuración de echarts	44
3.15.	Componente gráfico de echarts	45
3.16.	Pantalla adaptada a pantallas para PC	46
3.17.	Pantalla adaptada a tablets	46
3.18.	Pantalla adaptada a celulares	47
4.1.	Testing broker MQTT	53
4.2.	Carpetas de funciones para testing en Postman	54
4.3.	Petición HTTP a función de login de usuarios en Postman	55
4.4.	Respuesta a petición de login desde Postman	56
4.5.	Respuesta a petición de dispositivos con token inválido	56

4.6. Respuesta a petición de dispositivos con token válido	57
4.7. Archivos de instalación de Cypress	58
4.8. Resultado de test de login de la plataforma web	62
4.9. Resultado de test de dispositivos de la plataforma web	65
4.10. Estados de conexión de componente de dispositivo	66
4.11. Comando en Angular para ejecución de test de Cypress	66
4.12. Resultados de los test ejecutados en Cypress	67

Índice de tablas

1.1. Comparación de servicios cloud en el mercado.	6
2.1. Comandos utilizados en MongoDB	19
4.1. Comandos utilizados en MongoDB	53
4.2. Test de login utilizando Cypress	61
4.3. Test de login utilizando Cypress	64

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 tecnologías 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, líneas 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 sí. Con este fin se utiliza, entre otros, el protocolo de comunicación Modbus [2], introducido 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]. Proporciona comunicación cliente / servidor entre dispositivos conectados en diferentes tipos de buses o redes. Es un protocolo de solicitud / respuesta en el cuál el servidor realiza consultas de datos a un cliente de la red.

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.

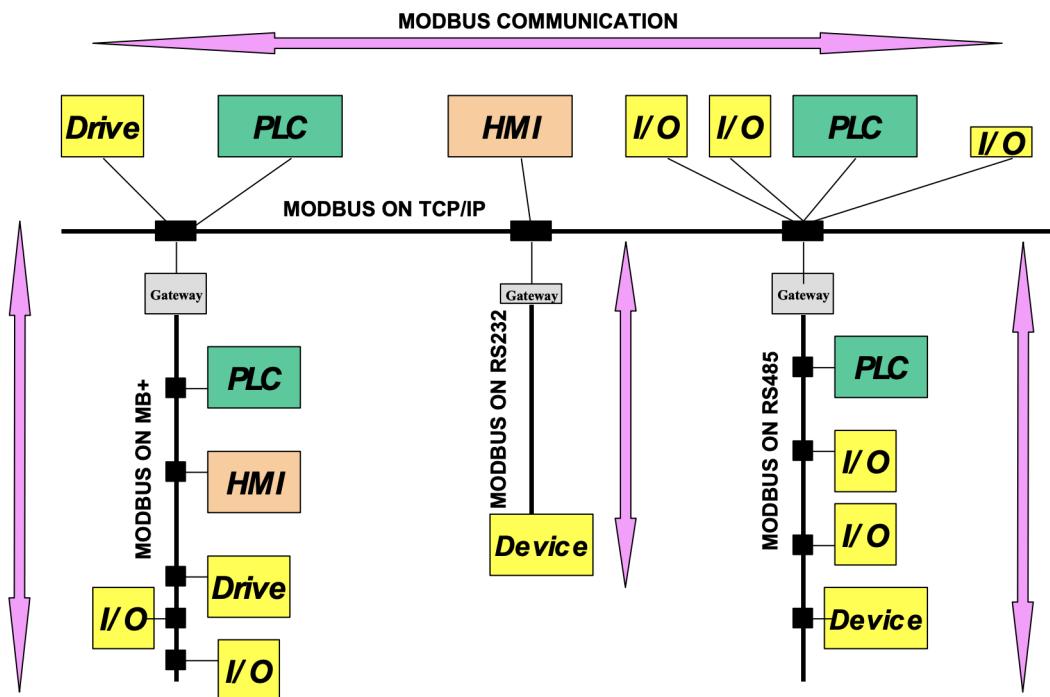


FIGURA 1.1. Ilustración de diferentes arquitecturas de red utilizando el protocolo Modbus¹.

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 Siemens².

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

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

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

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 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.

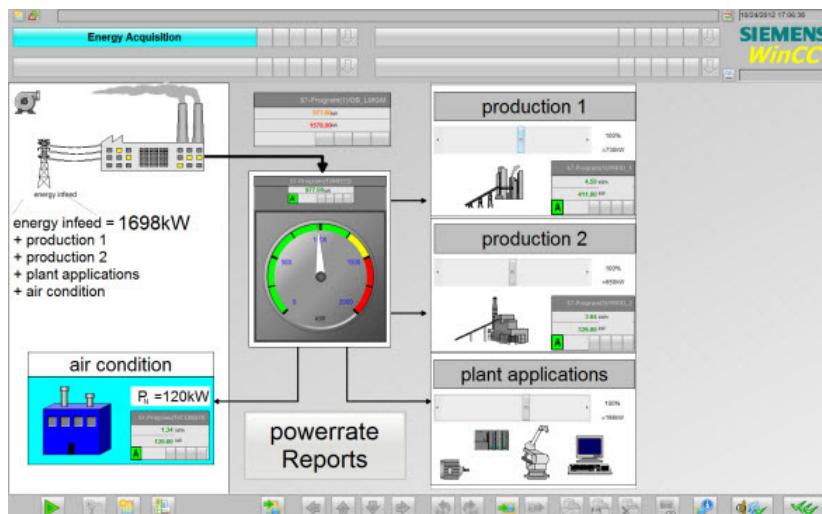


FIGURA 1.3. Ejemplo de implementación de un sistema SCADA con el software WINCC de la empresa Siemens para monitoreo y gestión de energía eléctrica³.

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,

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

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.

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 más 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 reflejada 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.

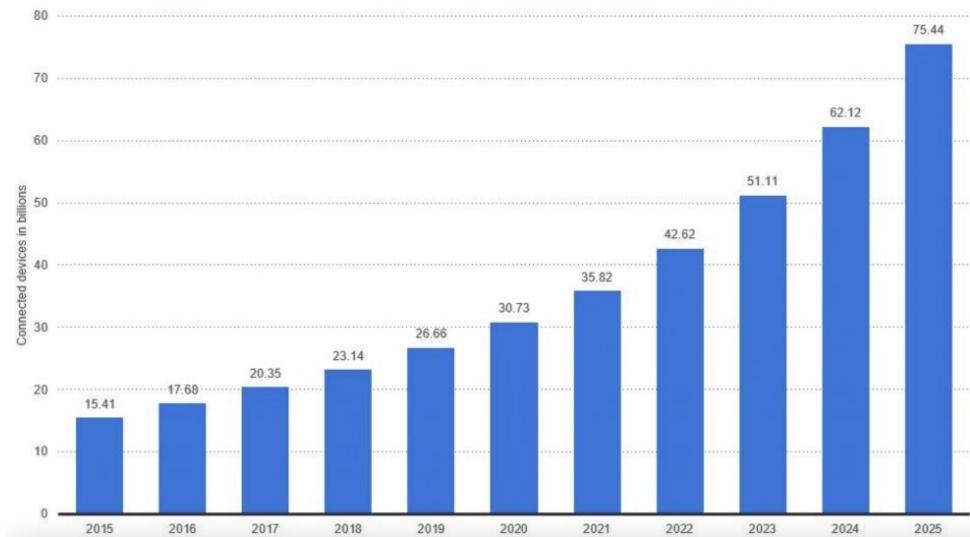


FIGURA 1.4. Crecimiento exponencial de dispositivos conectados en el mundo⁴.

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.

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.

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

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

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 breves períodos 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

En este capítulo se exponen las características del software utilizado para la programación del sistema, partiendo del backend hasta el frontend. También se identifican los módulos de funcionamiento del dispositivo conversor Modbus a MQTT utilizado en el trabajo. Finalmente se describe la utilidad de Nginx como servidor de datos para integrar el sistema.

2.1. Dispositivo conversor Modbus a MQTT

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

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

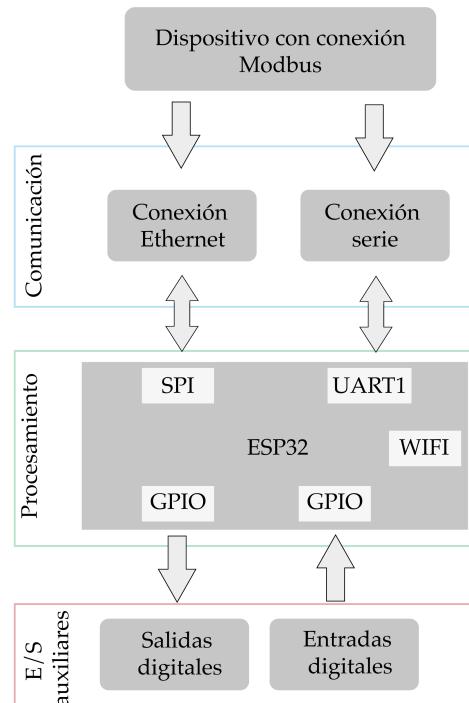


FIGURA 2.1. Diagrama de bloques de dispositivo conversor Modbus a MQTT.

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 través de Wi-Fi

Este módulo está constituido por un microcontrolador ESP32 [11] de la empresa Espressif [12] quien se encarga de controlar el módulo de comunicación y el módulo 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 vía Wi-Fi 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.

2.2. 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]. 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 suscritos 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 está 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.

Clients 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.

2.2.1. 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.

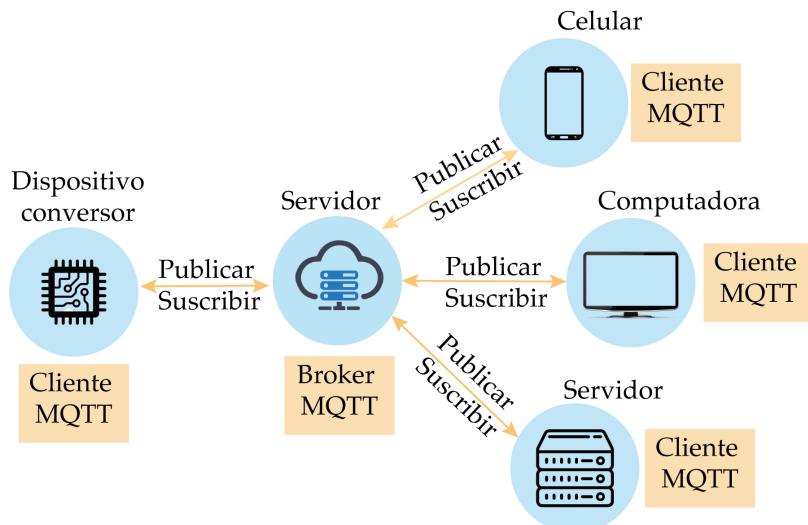


FIGURA 2.2. Modelo de funcionamiento de protocolo MQTT.

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 el o los clientes que están suscritos 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 suscrito 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 un comando de desconexión o la conexión se pierde por algún motivo. Los puertos estándar son 1883 para comunicación no encriptada y 8883 para comunicación encriptada usando SSL/TLS [19].

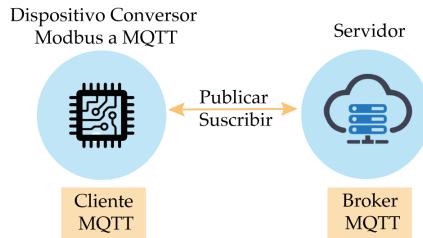


FIGURA 2.3. Esquema de conexión cliente - broker MQTT.

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 *wildcard* 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:

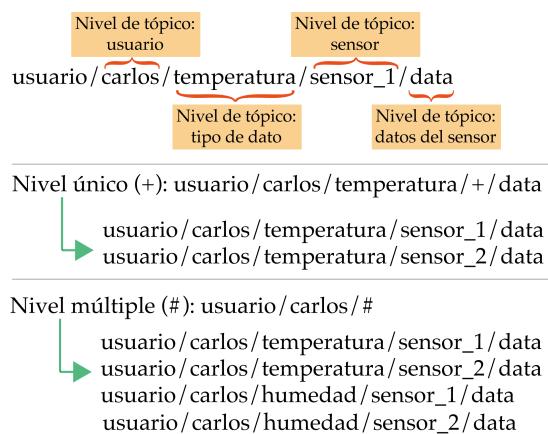


FIGURA 2.4. Ejemplo de tópicos y uso de *wildcards*.

2.2.2. 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 inglés *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 aplicación de utilización de proxy como *gateway*.

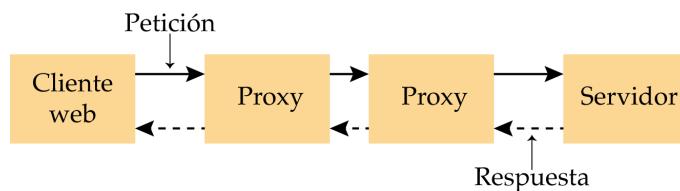


FIGURA 2.5. Diagrama de bloques ilustrando la utilización de proxy como *gateway*.

El navegador es siempre el que inicia una comunicación (petición), por lo que para poder mostrar una página web, 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 única 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 petición de HTTP está 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 petición GET al servidor utilizado en este trabajo.

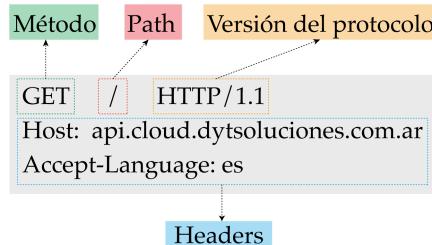


FIGURA 2.6. Ejemplo de petición GET al servidor utilizado para realizar este trabajo.

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

- La versión del protocolo HTTP que se usa.
- Un código de estado, indicando si la petición ha sido exitosa.
- 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 más importantes:

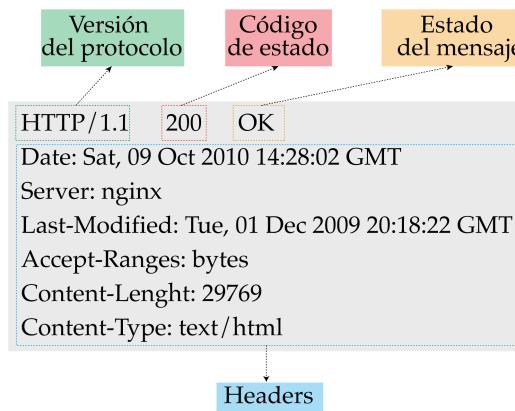


FIGURA 2.7. Ejemplo de modelo de respuesta HTTP del servidor.

2.3. Infraestructura del backend

En informática, la expresión 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 información. Las funciones más importantes del backend son las siguientes:

- Acceder a la información que se pide a través de la plataforma web: cuando se usa una aplicación web, se solicita información de manera continua. Esto implica que el sistema tiene que ser capaz de encontrar y acceder a lo solicitado a través 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 aplicación web 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 en el futuro la aplicación 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 ejecución 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 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 fácil de aprender y usar, y proporciona todas las funcionalidades que se necesitan para satisfacer los requisitos más 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.

2.3.1. 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 subprocesso, ocupando la RAM del sistema y regularmente maximizando la cantidad de RAM disponible, Node.js opera en un solo subprocesso, utilizando el modelo entrada y salida 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 en la figura 2.8 el proceso de ejecución que realiza el servidor.

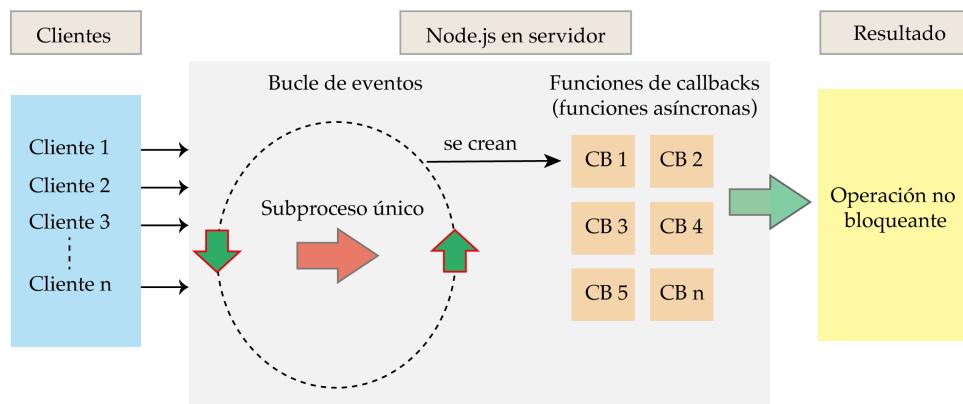


FIGURA 2.8. Ilustración del proceso de ejecución que realiza Node.js en un servidor.

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.

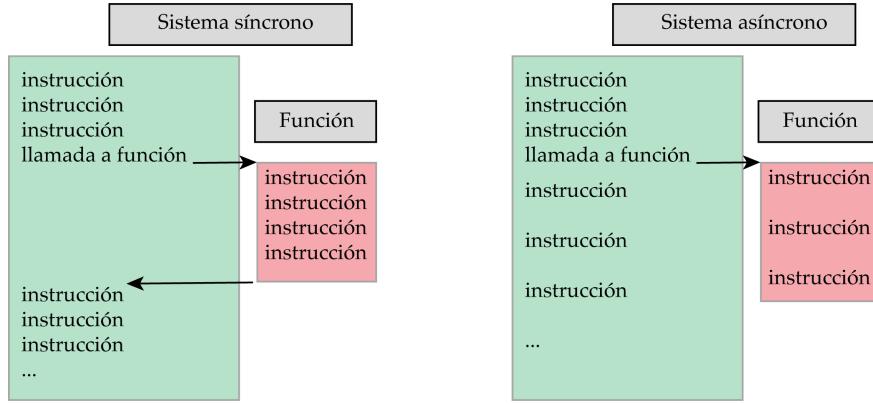


FIGURA 2.9. Ilustración de un sistema síncrono y un sistema asíncrono.

2.3.2. Base de datos MongoDB

MongoDB es una base de datos noSQL [24], donde se puede agregar información 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 características 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 [26] y su *sharding* [27] 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 acrónimo de Crear, Leer, Actualizar y Borrar (del original en inglés *Create, Read, Insert* y *Delete*) y se usa para referirse a las funciones básicas 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 - Base de datos relacional			
ID_Dispositivo	Nombre	Ubicacion	Tipo
A1:B1:C1:D2:E2:F1	Dispositivo_1	Oficina_1	T700
A5:6B:2C:7D:EE:FF	Dispositivo_2	Oficina_2	T700
11:B2:C5:D7:E2:44	Dispositivo_3	Oficina_3	T700

Colecciones - MongoDB
{
"_id":,
"ID_Dispositivo": "A1:B1:C1:D2:E2:F1",
"Nombre": "Dispositivo_1",
"Ubicacion": "Oficina_1",
"Tipo": "T700",
},
{
"_id":,
"ID_Dispositivo": "A5:6B:2C:7D:EE:FF",
"Nombre": "Dispositivo_2",
"Ubicacion": "Oficina_2",
"Tipo": "T700",
},
{
"_id":,
"ID_Dispositivo": "11:B2:C5:D7:E2:44",
"Nombre": "Dispositivo_3",
"Ubicacion": "Oficina_3",
"Tipo": "T700",
}

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

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

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

2.4. 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 interacción con los usuarios. Se desarrolla, principalmente, a través de tres lenguajes: HTML (*HyperText Markup Language*), CSS (*Cascading Style Sheets*) [28] y JS (Javascript) [29]. Cada uno de estos lenguajes se usa para desarrollar diferentes partes del frontend. Estos lenguajes de programación se dividen en tres tareas básicas del desarrollo:

- Arquitectura: HTML es el componente más 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 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 interacciéndola con el usuario, el navegador y el servidor. JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript [30]. 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 [31] que es un framework de diseño eficiente y sofisticado de plataformas web.

2.4.1. Angular

Angular es una plataforma de desarrollo construida sobre *TypeScript* [32]. 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 ayudan 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 encapsulan 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 aplicación 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 módulo 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*) [33]. Cada componente define una clase que contiene datos y lógica, y está 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*. Esta 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 cambia 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 aplicación de Angular ejecutar código 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 aplicación 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.

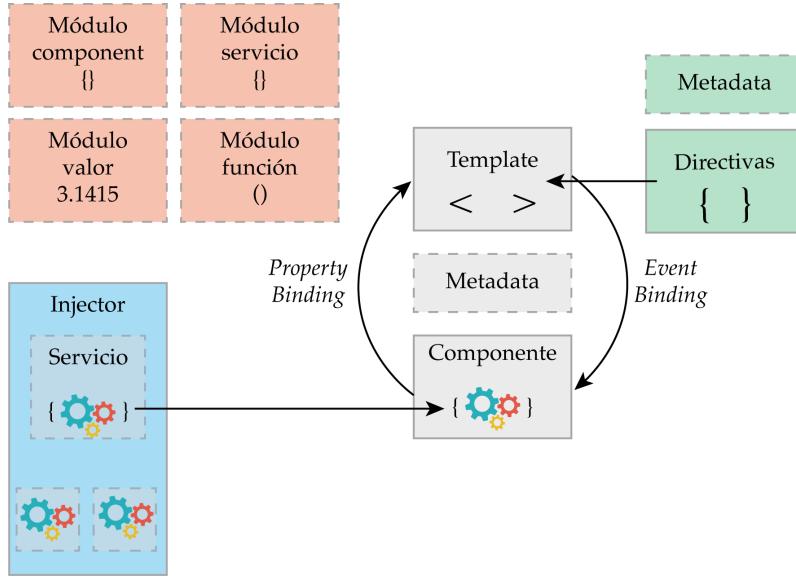


FIGURA 2.11. Ilustración de arquitectura de funcionamiento de Angular¹.

2.5. Servidor Nginx

Nginx [34] 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.
- Servidor proxy inverso para la aceleración web (HTTP, TCP, UDP) 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 [35], 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.

¹Imagen acondicionada de <https://angular.io/guide/architecture>

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 2.12 se ejemplifica el proceso principal de Nginx con la creación de n *workers* por cada consulta / respuesta.

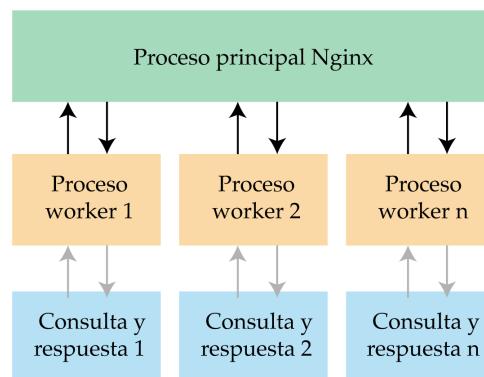


FIGURA 2.12. Ilustración de funcionamiento principal de Nginx con n consultas / respuestas.

Para este trabajo se utilizó Nginx como proxy inverso. Se conoce como proxy inverso cuando un servidor acepta todo el tráfico y lo reenvía a un recurso específico, 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 2.13 se observa la configuración típica de Nginx como proxy inverso, donde se destaca que cada uno de los clientes tendrá un solo punto de acceso a los recursos del servidor.

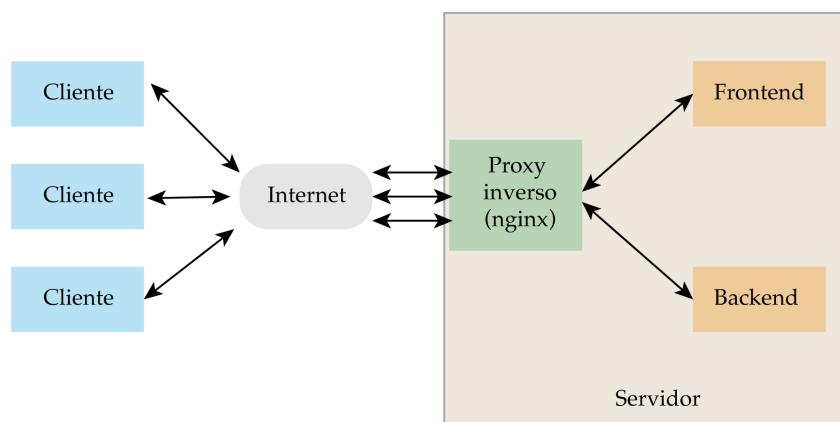


FIGURA 2.13. Ilustración de configuración de nginx como proxy inverso.

Capítulo 3

Diseño e implementación

En este capítulo se detalla el diseño de la arquitectura del sistema en todos sus componentes. Se menciona el motivo de elección en cada caso y la implementación correspondiente.

3.1. Diseño de la estructura general del sistema

La estructura del sistema implementado puede observarse en la figura 3.1. El sistema consta de clientes que serán navegadores web para acceder al frontend y a las API [36] que contiene el backend. Por otro lado, del lado del cliente, los dispositivos conversores Modbus a MQTT se conectan al broker que también forma parte del servidor.

Las funciones implementadas en el backend se conectan a una base de datos remota para almacenar los datos de clientes web o bien mediciones de dispositivos conectados.

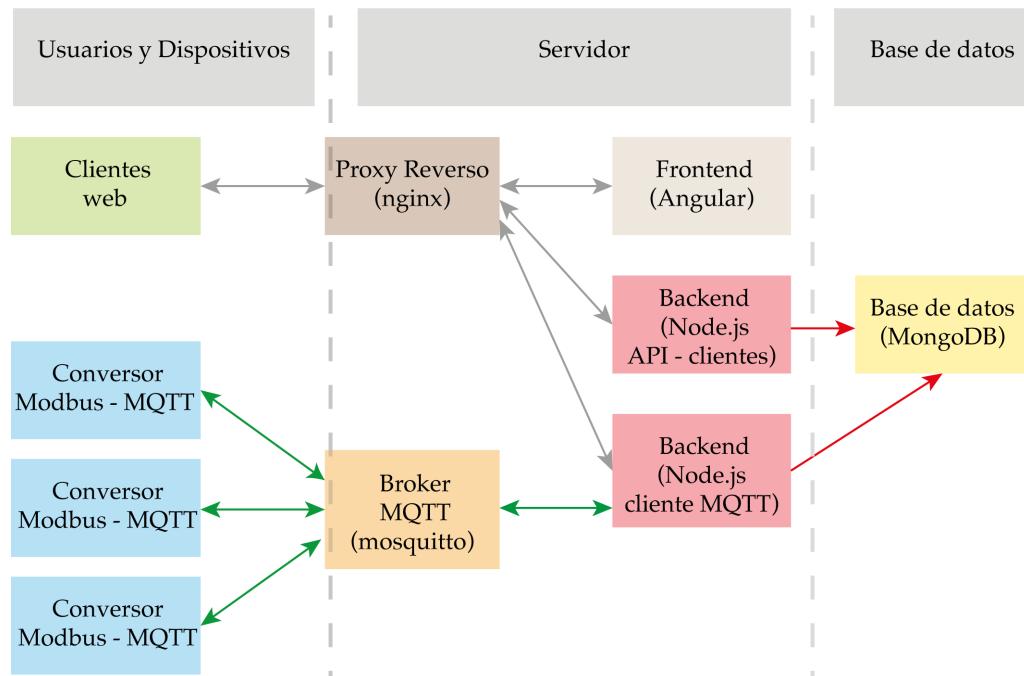


FIGURA 3.1. Diagrama de bloques de sistema de gestión implementado.

En el servidor se desarrolló el frontend en Angular mientras que el backend consta de dos módulos. Por un lado el modulo de API de clientes que fue desarrollado en Node.js y contiene la implementación de funciones para interactuar con la base de datos y la creación de usuarios, dispositivos y visualización de configuración. Además, el backend consta de un módulo cliente MQTT que se encarga de interactuar entre el broker y los dispositivos conectados a él. Es el encargado de gestionar la información que envían los dispositivos hacia la base de datos para luego ser utilizados por el frontend.

Todo el servidor es gestionado por Nginx, el cual actúa como proxy inverso lo que permite un único canal de comunicación entre usuarios y sistema. Por último, la base de datos se encuentra en un servidor remoto y se accede a ella mediante credenciales de conexión.

3.2. Implementación del backend

El backend del trabajo realizado está dividido en diferentes bloques de funcionamiento.

El bloque de backend de funciones API de clientes que se observa en la figura 3.2 donde su conexión con el punto de entrada al servidor gestionado por Nginx es el puerto 8000. Por otro lado, este bloque se conecta con la base de datos remota MongoDB Atlas [37] para almacenar datos de nuevos usuarios, dispositivos, configuración de dispositivos, organizaciones y relaciones entre usuarios y organizaciones. Los clientes que acceden a cualquier función de este bloque lo harán a través de un puerto seguro, dotado de certificados SSL que es manejado por Nginx y mapeado al puerto 8000.

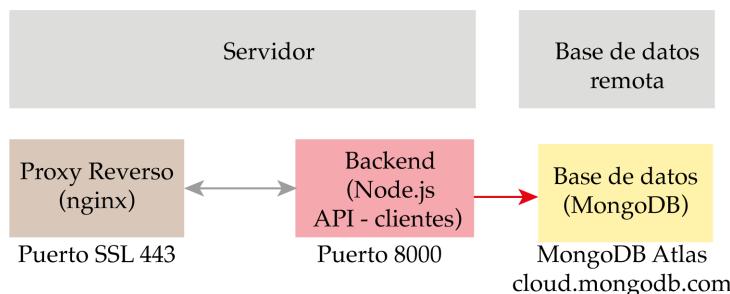


FIGURA 3.2. Diagrama de conexión entre Nginx, modulo de API de clientes y base de datos en MongoDB Atlas.

La estructura interna de este bloque consta de rutas y un bloque de funciones para el manejo antes mencionado. En la figura 3.3 puede observarse un desglose general de este bloque.

Se utilizó la librería de Node.js llamada Express.js [38] la cual proporciona mecanismos para:

- Escritura de manejadores de peticiones con diferentes verbos HTTP en diferentes rutas.
- Integración con motores de renderización de vistas para generar respuestas mediante la introducción de datos en plantillas.

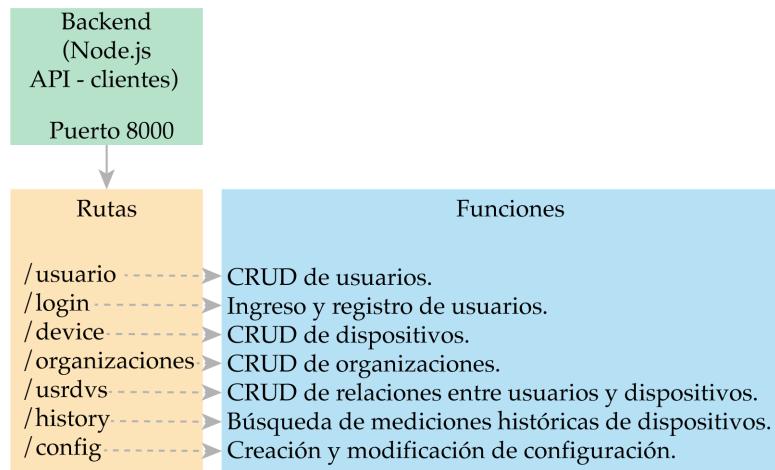


FIGURA 3.3. Descripción de rutas y funciones principales de bloques que API clientes.

- Establecer ajustes de aplicaciones web como qué puerto usar para conectar, y la localización de las plantillas que se utilizan para renderizar la respuesta.
- Añadir procesamiento de peticiones *middleware* adicional en cualquier punto dentro de la tubería de manejo de la petición.

En el código 3.1 se muestra la forma adoptada para crear las rutas para las funciones de gestión.

```

1 // Inclusion del modulo express
2
3 var express = require('express');
4
5 var app = express();
6
7 //Rutas
8 app.use('/usuario', require('./routes/usuario'));
9
10 app.use('/login', require('./routes/auth'));
11
12 app.use('/device', require('./routes/devices'));
13
14 app.use('/organizaciones', require('./routes/organizations'));
15
16 app.use('/usrdvs', require('./routes/usrs_dev'));
17
18 app.use('/history', require('./routes/historical'));
19
20 app.use('/config', require('./routes/sensors-config'));
21
22
23 // Escuchar peticiones
24 app.listen(process.env.PORT, () => {
25   console.log(`server: \x1b[32m% \x1b[0m', 'running');
26 });
27

```

CÓDIGO 3.1. Utilización de Express para crear rutas en el servidor.

Otra biblioteca importante que se utiliza en este bloque es Mongoose [39], que se utiliza para escribir consultas para una base de datos de MongoDB, con características como validaciones, construcción de *queries*, *middlewares*, conversión de tipos entre otras, que enriquecen la funcionalidad de la base de datos.

La parte central del uso de Mongoose está en la definición de un esquema donde se indica la configuración de los documentos para una colección de MongoDB. En el código 3.2 se muestra su inicialización y la conexión remota.

```

1 // Inclusion de mongoose.
2 var mongoose = require('mongoose');
3
4 // Conexion a la base de dato remota
5 try {
6     await mongoose.connect( process.env.DB_CNN , {
7         useNewUrlParser: true ,
8         useUnifiedTopology: true ,
9         useCreateIndex: true
10    });
11
12    console.log('DB Online');
13
14 } catch (error) {
15     console.log(error);
16     throw new Error('Error a la hora de iniciar la BD');
17 }
18
19 }
```

CÓDIGO 3.2. Utilización de Mongoose para el manejo de MongoDB.

Además, dentro del backend del trabajo realizado, se encuentra el bloque para el manejo de mensajes MQTT que se publican en el broker y provienen de los dispositivos conectados a él. En la figura 3.4 se puede ver su conexión con el broker y además con la base de datos MongoDB.

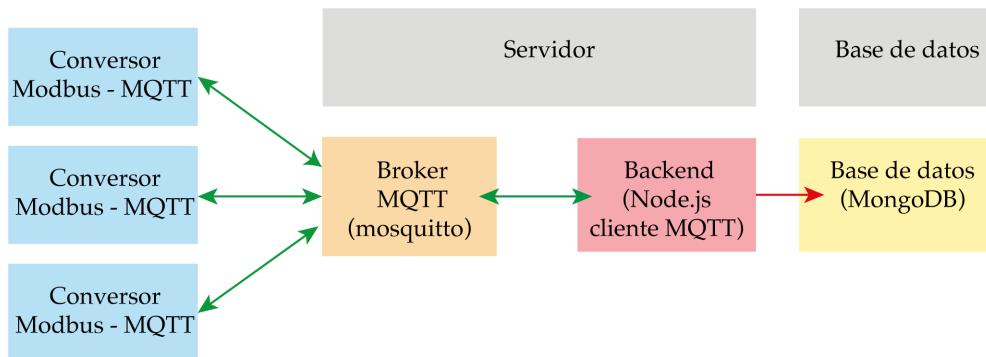


FIGURA 3.4. Descripción de cliente MQTT implementado en el servidor.

Para crear un cliente MQTT en Node.js se utiliza la librería MQTT.js [39] con la cual se pueden crear clientes que se conecten al broker del sistema. En el código 3.3 se muestra la implementación en el backend del servidor.

```

1 // Incluimos la libreria mqtt
2 var mqtt = require('mqtt');
3
4
```

```

5 // Definicion de variables de conexion.
6 var options= {
7     port: +process.env.MB_PORT,
8     username: process.env.MB_USERNAME,
9     password: process.env.MB_PASSWORD,
10 };
11
12 // Conexion al broker MQTT
13 var client = mqtt.connect(process.env.MB_URL, options);
14
15 /* Evento de conexion con el broker */
16 client.on('connect', function() {
17     logger._log('info', 'MQTT broker conectado');
18
19         /* Subcripcion a los topicos */
20     MQTTsubscriber.subscribe(client);
21 });
22
23 /* Evento de error en la conexion con el broker */
24 client.on('error', (error) => {
25     logger._log('error', 'Error: ' + error.message);
26 });
27
28 /* Evento de reconexion del broker */
29 client.on('reconnect', () => {
30     logger._log('warn', 'Reconectando al Broker MQTT');
31 });
32
33
34 /* Evento de conexion cerrada con el broker */
35 client.on('close', () => {
36     logger._log('warn', 'Conexion con el broker cerrada');
37 });
38
39 /* Evento de broker Offline */
40 client.on('offline', () => {
41     logger._log('info', 'Conexion con el broker Offline');
42 });
43
44 /* Evento de broker finalizado */
45 client.on('end', () => {
46     logger._log('info', 'Conexion con el broker Finalizada');
47 });
48
49 /* Evento de mensaje del broker donde se recibe
50 * el topico y el mensaje
51 */
52 client.on('message', function(topic, message, packet) {
53     console.log(topic);
54     console.log(message);
55     console.log(packet);
56
57     MQTTsubscriber.observe(topic, message, packet);
58 });

```

CÓDIGO 3.3. Cliente MQTT en el servidor utilizando librería MQTT.js.

3.2.1. Modelos utilizados en la base de datos

Para el manejo de la base de datos, Mongoose requiere que se creen *Schemas* de cada uno de los modelos que intervienen en este trabajo.

Para la implementación del modelo de usuario se realizó un estudio de los campos que serán utilizados por el cliente para el registro y acceso al sistema. El código 3.4 muestra esta definición en conjunto con sentencias para validar información del modelo.

```

1 // Inclusion de librerias utilizadas
2 var mongoose = require('mongoose');
3 var uniqueValidator = require('mongoose-unique-validator');
4
5 // Definicion de Schema
6 var Schema = mongoose.Schema;
7
8
9 // Variable con roles permitidos cuando se crea un usuario.
10 var rolesValidos = {
11     values: ['CREATOR_ROLE', 'ADMIN_ROLE', 'USER_ROLE'],
12     default: 'USER_ROLE',
13     message: '{VALUE} no es un rol permitido'
14 };
15
16 //Creo el schema para el modelo de Usuario
17 var usuarioSchema = new Schema({
18
19     nombre: { type: String, required: [true, 'El nombre es requerido'] },
20     email: { type: String, unique: true, required: [true, 'El correo es
necesario'] },
21
22     role: { type: String,
23             required: true,
24             default: 'USER_ROLE',
25             enum: rolesValidos },
26     created_by: { type: Schema.Types.ObjectId, ref: 'Usuario', require:[
true, 'debe crearlo un usuario'] },
27     created: { type: String, default: '' },
28     updated: { type: String, default: '' },
29     deleted: { type: String, default: '' },
30 });
31
32 // Plugin que me permite enviar un mensaje de error cuando se hacen las
validaciones.
33 usuarioSchema.plugin(uniqueValidator, { message: '{PATH} debe ser unico' });
34
35 // Exporto el modelo para que se pueda utilizar en las funciones del
programa.
36 module.exports = mongoose.model('Usuario', usuarioSchema);

```

CÓDIGO 3.4. Definición de Schema para el modelo de usuario.

Para el resto de los modelos utilizados en el trabajo, el procedimiento adoptado es similar al del código 3.4. En la figura 3.5 se muestran los modelos más utilizados en este trabajo para realizar operaciones con la base de datos.

3.2.2. Implementación CRUD para el manejo de usuarios

Como se mencionó en la sección 3.2, en el bloque del backend encargado de manejar las funciones APIs para crear nuevos usuarios y dispositivos, entre otras funcionalidades, se implementaron funciones para crear, modificar, leer y borrar cada uno de ellos. Además, se implementaron criterios de búsqueda de dispositivos y usuarios teniendo en cuenta parámetros que provienen de las rutas.

Dispositivos	Organizaciones	Sensor T700
creado: time modificado: time borrado: time usuario: Usuario_id nombre: string mac: string org: Org_id ubicacion: string	creado: time modificado: time borrado: time usuario: Usuario_id nombre: string	temperature: number valid: boolean timestamp: time model: string mac: string

FIGURA 3.5. Descripción modelos de datos utilizados para crear los Schemas para realizar operaciones en Mongoose.

```

1  /* Crear un nuevo usuario*/
2
3
4  const crearUsuario = async (req, res = response) => {
5    const { nombre, email, password, role, created_by } = req.body;
6  /* Completo los campos del modelo*/
7    var usuario = new Usuario({
8      nombre: nombre,
9      email: email,
10     role: role,
11     created_by: created_by,
12     created: new Date().toISOString(),
13   });
14 /* Guardo el nuevo usuario en la base de datos*/
15   await usuario.save();
16   /* Envio la respuesta*/
17   res.json({
18     ok: true,
19     usuario
20   });
21 }
22
23 /* Borrar un usuario*/
24 const borrarUsuario = async (req, res = response) => {
25   const id = req.params.id;
26   await Usuario.findByIdAndDelete( id );
27   res.json({
28     ok: true,
29     msg: 'Usuario Eliminado'
30   })
31 }
32
33 /* Leer un usuario*/
34 const getUsuarioById = async (req, res = response) => {
35   var id = req.params.id;
36   const user = await Usuario.findById(id)
37     .populate('nombre email role password')
38   res.status(200).json({
39     ok: true,
40     usuario: user
41   });
42 }
43
44 /* Editar un usuario*/
45 const actualizarUsuario = async (req, res = response) => {
46   var id = req.params.id;
47   const { password, email, role, ...campos} = req.body;
48

```

```

49     campos.updated = new Date().toISOString();
50     const usuarioActualizado = await Usuario.findByIdAndUpdate( id,
51         campos, {new: true});
52     res.json({
53         ok: true,
54         usuario: usuarioActualizado
55     })

```

CÓDIGO 3.5. Implementación CRUD para usuarios que se registran.

Para la implementación del CRUD de organizaciones y dispositivos, se empleó el mismo concepto de programación, utilizando los modelos correspondientes.

3.2.3. Configuración del broker MQTT

El broker MQTT utilizado para este trabajo es Mosquitto [40]. Es un servidor de mensajes de código abierto (con licencia EPL/EDL) que implementa las versiones 3.1 y 3.1.1 del protocolo MQTT. Es ampliamente utilizado debido a su rapidez, lo que permite emplearlo en gran número de ambientes, incluso si éstos son de pocos recursos. En la sección 2.2.1 se explicó en detalle el uso del protocolo MQTT y se especificó el funcionamiento del broker.

La configuración de Mosquitto se realiza a través de un archivo interno que se encuentra entre los archivos de instalación del broker con el nombre de *mosquitto.conf*.

En el código 3.6 se muestra el uso y aplicación de cada parámetro configurado para este trabajo.

```

1 /* Comando para no permitir conexiones de usuarios anonimos*/
2 allow_anonymous false
3
4
5 /* Defino que el password se guardara en la carpeta passwd*/
6 password_file /etc/mosquitto/passwd
7
8 /* Conexion en el puerto 1883 para los dispositivos - sin seguridad */
9 listener 1883
10
11 /* Conexion en el puerto 884 con aplicacion de seguridad SSL */
12 listener 884
13
14 /* Utilizacion de protocolo MQTT por websockets para este puerto*/
15 protocol websockets
16
17 /* Utilizacion de protocolo MQTT por websockets para este puerto*/
18 certfile /
19 cafile /
20 keyfile /

```

CÓDIGO 3.6. Configuración utilizada en Mosquitto como broker MQTT.

3.2.4. Seguridad en el servidor

Para dotar de seguridad al servidor, se instalaron certificados SSL que es un estándar de seguridad global que permite la transferencia de datos cifrados entre un navegador y un servidor web.

Para establecer esta conexión segura, se instala en un servidor web un certificado SSL (también llamado certificado digital) que cumple dos funciones:

- Autenticar la identidad del sitio web, garantizando a los visitantes que no están en un sitio falso.
- Cifrar la información transmitida.

Para dotar de seguridad al sistema se utilizó una herramienta llamada *certbot* [41] y en el código 3.7 se muestran los pasos realizados para instalar los certificados en Nginx.

```

1 // Habilitar https a traves del firewall
2 sudo ufw allow 'Nginx Full'
3 sudo ufw delete allow 'Nginx HTTP'
4
5 /* Obtener un certificado SSL para el dominio utilizado en este trabajo */
6
7 sudo certbot --nginx -d cloud.dytsoluciones.com.ar -d www.cloud.
8 dytsoluciones.com.ar
9
10 /* Se completan los pasos requeridos por Certbot y para verificar la
11 instalacion se ejecuta el comando*/
12 sudo systemctl status certbot.timer
13
14 /* Ademas, para testea el proceso de renovacion se ejecuta*/
15 sudo certbot renew --dry-run

```

CÓDIGO 3.7. Procedimiento realizado para instalar certificados SSL en Nginx.

Una vez realizados estos pasos, el acceso al servidor sera por HTTPS y el sitio estará protegido.

3.3. Implementación del frontend

Para el diseño del frontend, se tuvo en cuenta los siguientes requerimientos:

- Debe ser un cliente del broker MQTT para poder observar datos en tiempo real de los dispositivos conectados.
- Debe tener funciones de consultas hacia la base de datos a través del backend.
- Debe contar con un menú de configuración para vinculación de nuevos dispositivos.
- Debe adaptarse a cualquier tamaño de pantalla como ser celulares, tabletas y monitores.
- Debe tener acceso protegido con usuario y contraseña para cada usuario.

Para la implementación se utilizó Angular como *framework* de programación.

3.3.1. Diseño de plataforma web con Angular

La estructura general de la plataforma consta de una pantalla de inicio donde el usuario puede iniciar sesión, o bien realizar un registro al sistema para luego poder ingresar al sistema.

En la figura 3.6 se puede observar la pantalla de login, mientras que la figura 3.7 muestra la pantalla de registro de usuario.

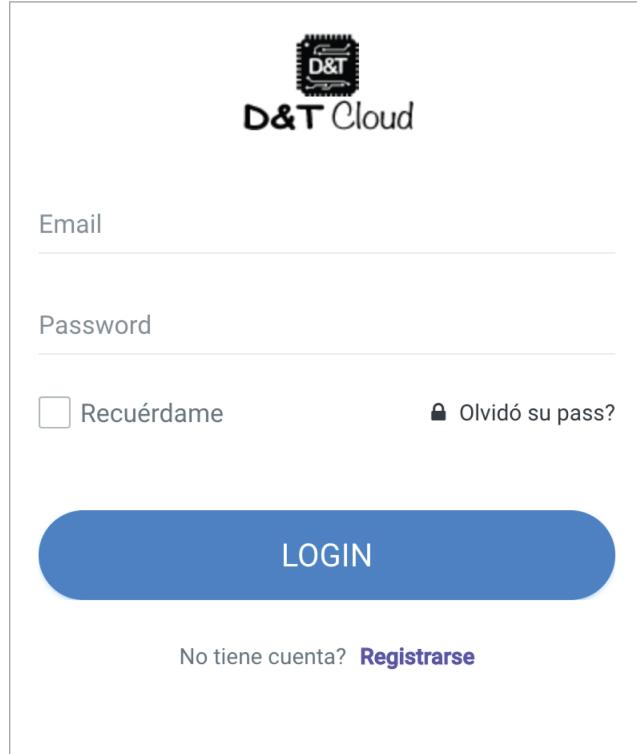


FIGURA 3.6. Ilustración de pantalla de login de usuario en plataforma web.

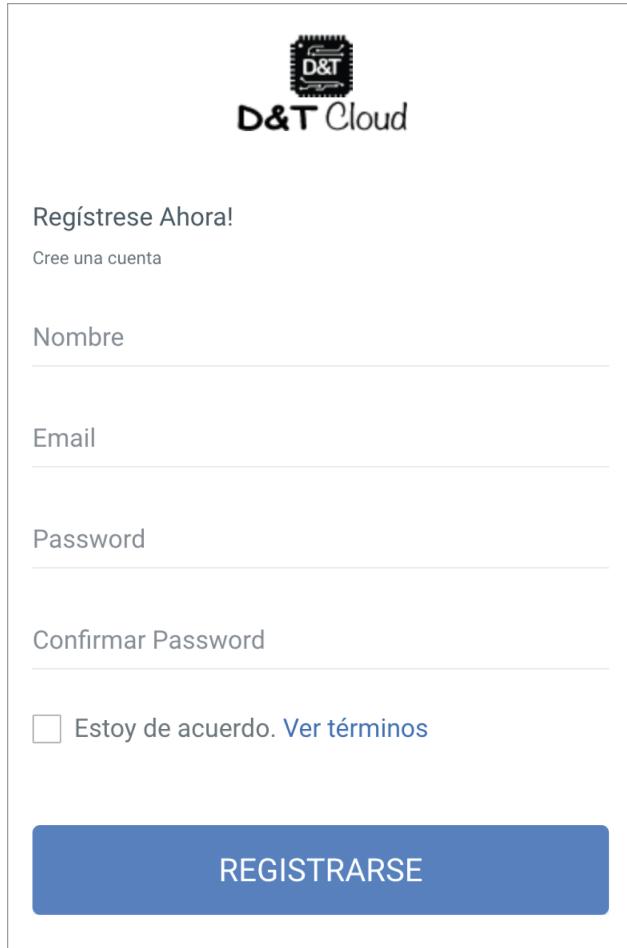


FIGURA 3.7. Ilustración de pantalla de registro de usuarios en plataforma web.

Para la implementación en Angular, dentro de la carpeta /src se creo un componente para la pantalla de login y un componente para la pantalla de registro. En la figura 3.8 puede observarse la estructura y archivos creados para el desarrollo de las pantallas antes mencionadas.

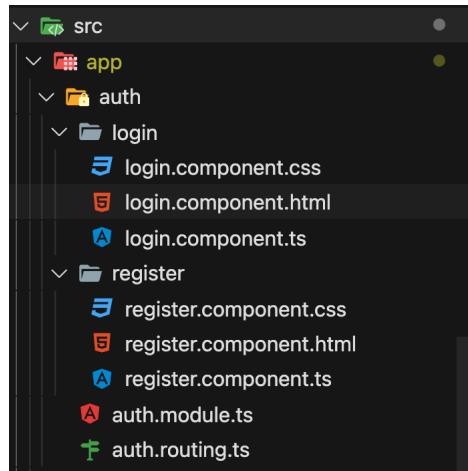


FIGURA 3.8. Ilustración de estructura creada en el desarrollo de la pantalla de login y registro.

Los comandos de angular utilizados para crear estos componentes se pueden observar en el código 3.8.

```

1 // Comando para crear el componente login en la carpeta auth
2 ng create component auth/login
3 // Comando para crear el componente register en la carpeta auth
4 ng create component auth/register

```

CÓDIGO 3.8. Comandos de Angular para crear componentes de login y registro.

Para la programación del componente login, se utilizó una plantilla HTML para formar la imagen deseada y utilizar dos campos de texto para introducir el usuario y contraseña. En el código 3.9. se puede observar la implementación de los aspectos principales de la pantalla teniendo en cuenta que además se utilizaron clases CSS para asignarle el estilo deseado.

```

1 <div class="card-body">
2
3     <form (submit)="login()" 
4         class="form-horizontal form-material"
5         id="loginform"
6         autocomplete="off"
7         [FormGroup] = "loginForm">
8
9         <div class="form-group m-t-40">
10            <div class="col-xs-12">
11                <input class="form-control"
12                    type="email"
13                    placeholder="Email"
14                    formControlName='email'
15                    >
16            </div>
17        </div>
18    </div>
19    <div class="form-group">
20        <div class="col-xs-12">
21            <input class="form-control"
22                type="password"
23                placeholder="Password"
24                formControlName='password'>

```

```
25             </div>
26         </div>
27
28         <div class="form-group text-center m-t-20">
29             <div class="col-xs-12">
30                 <button class="btn btn-info btn-lg btn-block
text-uppercase btn-rounded"
31                     type="submit"
32                     [class.spinner]="isLoading" [disabled]=""[disabled]="
33 isLoading">Log In</button>
34             </div>
35         </div>
36
37     </form>
</div>
```

CÓDIGO 3.9. Desarrollo de código HTML para la pantalla de login teniendo en cuenta estilos de diseño CSS.

Otro aspecto importante en el desarrollo de la pantalla es la programación del comportamiento de cada campo en el que el usuario interactúa con la página. El código 3.10 muestra las funciones principales que se programaron en el archivo *TypeScript*.

```
1 // Definicion de clase LoginComponent
2 export class LoginComponent {
3
4     public formSubmitted = false;
5     public isLoading = false;
6
7
8     public loginForm = this.fb.group({
9         email:[localStorage.getItem('email') || '', [Validators.required,
10            Validators.email]],
11         password: ['', Validators.required],
12         remember: [false]
13     });
14
15 // Constructor de la clase donde se incluyen servicios para utilizar
16 constructor(private router:Router,
17             private fb: FormBuilder,
18             private usuarioService: UsuarioService,
19             private sweetAlert:SweetAlertService
20             ) { }
21
22 // Metodo login() , cuando el usuario presiona el boton
23 login(){
24     this.formSubmitted = true;
25     this.isLoading = true;
26     if(this.loginForm.invalid){
27         this.isLoading = false;
28         return;
29     }
30     // Validacion del formulario
31     if (this.loginForm.get('remember').value){
32         localStorage.setItem('email',this.loginForm.get('email').value)
33     } else{
34         localStorage.removeItem('email');
35     }
36     // Utilizacion de servicio injectado en el constructor.
37     this.usuarioService.loginUsuario(this.loginForm.value)
38         .subscribe(resp => {
39             console.log('login correcto', resp);
```

```

39     this.isLoading = false;
40 // Si el login es correcto, navega a la pagina principal.
41     this.router.navigateByUrl('/');
42   },
43   (err) => {
44     console.log(err);
45     this.isLoading = false;
46     this.sweetAlert.showAlert(
47       'Error',
48       err.error.msg,
49       'error',
50       'Ok'
51     );
52   });
53 }
54 }
55 }
56 }
```

CÓDIGO 3.10. Fragmentos de código más relevantes utilizado en el archivo *TypeScript* del componente login.

Otro aspecto importante del frontend es la programación de un servicio que permita conectarse con el backend y realizar peticiones HTTP. Para el caso de la pantalla de login, se requiere acceder a la base de datos para verificar si el email y password utilizado por el usuario son correctos y así poder acceder a la página principal.

El código 3.11 muestra la implementación de un servicio que es utilizado por el componente de login de usuario. En él puede observarse un método llamado loginUsuario el cual realiza una petición POST al backend. Luego espera una respuesta por parte de la petición para verificar si la misma es correcta. Por otro lado, se programó un método crearUsuario el cual crea un nuevo usuario en la base de datos, utilizando las funciones CRUD programadas en el backend.

```

1
2 export class UsuarioService {
3
4   public usuario: UserModel;
5
6   constructor(private http: HttpClient,
7             private router: Router) { }
8
9   crearUsuario( formData: RegisterForm){
10     return this.http.post(`${base_url}/usuario`, formData );
11   }
12
13   loginUsuario( formData: LoginForm){
14     return this.http.post(`${base_url}/login`, formData )
15     .pipe(
16       tap( (resp:any) =>{
17         localStorage.setItem('token', resp.token);
18         localStorage.setItem('menu', JSON.stringify(resp.menu));
19         //sessionStorage.setItem('user_id', resp.id._id);
20       })
21     );
22   }
23 }
```

CÓDIGO 3.11. Fragmentos de código más relevantes utilizados en el servicio de usuario.

En general, toda la implementación del frontend está basada en estas tres implementaciones de código. La metodología adoptada fue crear componentes por cada una de las funcionalidades donde el usuario interactúa con la plataforma y a su vez, la utilidad de los servicios para interactuar entre el backend y el frontend teniendo en cuenta las consultas a la base de datos.

Por otro lado, para observar los datos en tiempo real provenientes de los dispositivos conectados al broker MQTT, es necesario que el frontend pueda acceder y suscribirse a los tópicos a los que estos dispositivos están publicando.

Para ello se utiliza la librería `ngx-mqtt` [42] que permite crear un cliente MQTT a través del uso de websockets [43]. El código 3.12 muestra la forma de realizar la conexión con el broker MQTT utilizando el puerto seguro 884 descrito en la sección 3.2.3. Luego en cada componente que se utiliza la conexión se implementan los métodos de suscripción y publicación.

```

1 // Definicion de parametros de conexion
2
3 mqtt_b: {
4   connectOnCreate: true ,
5   protocol: "wss",
6   host: "host.al.broker",
7   port: 884,
8   path: "",
9   username: "user",
10  password: "pass",
11  keepalive: 60,
12  reconnectPeriod: 1000,
13  test: false ,
14 },
15
16 // Configuracion del servicio de la libreria MQTT
17 const MQTT_SERVICE_OPTIONS: IMqttServiceOptions = {
18   clientId: 'mqtt_dy',
19   connectOnCreate: environment.mqtt_b.connectOnCreate ,
20   protocol: (environment.mqtt_b.protocol === "wss") ? "wss" : "ws",
21   hostname: environment.mqtt_b.host ,
22   port: environment.mqtt_b.port ,
23   path: environment.mqtt_b.path ,
24   username: environment.mqtt_b.username ,
25   password: environment.mqtt_b.password ,
26   keepalive: environment.mqtt_b.keepalive ,
27   reconnectPeriod: environment.mqtt_b.reconnectPeriod ,
28 };

```

CÓDIGO 3.12. Implementación de cliente MQTT en Angular utilizando la librería `ngx-mqtt`.

Una vez que el usuario realiza el login de forma exitosa, puede acceder a las diferentes pantallas de la plataforma. En la figura 3.9 se puede observar un diagrama de todas las rutas implementadas en el trabajo a las que un usuario puede acceder.

Cuando un usuario se registra a la plataforma, por defecto es un usuario administrador. Este rol le permite acceder y crear nuevos usuarios donde puede asignar roles de operador.

El rol operador, solo permite a los usuarios visualizar información referida a los dispositivos conectados, invalidando la posibilidad de editar, borrar o agregar dispositivos y usuarios.

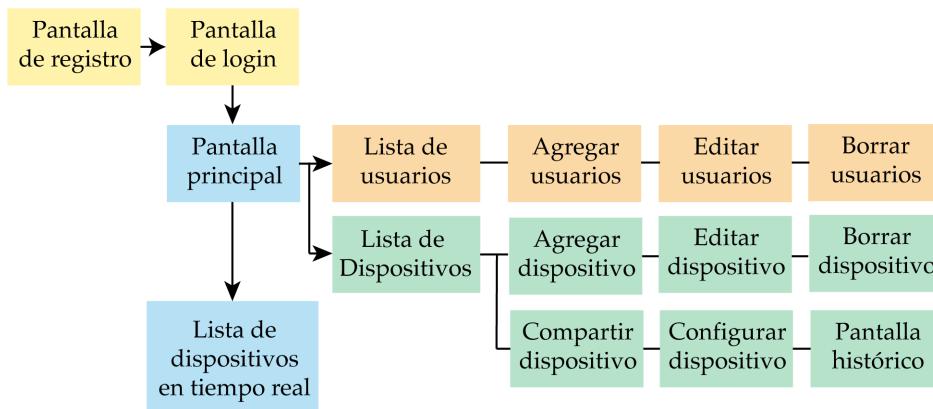


FIGURA 3.9. Ilustración de bloques con rutas implementadas en el sistema.

En la pantalla principal o también llamada *dashboard*, se muestran dispositivos que se encuentran agregados al sistema por parte de un usuario administrador. Estos, son leídos desde la base de datos cuando el usuario ingresa a la plataforma y se adaptan a un modelo de sensor cargado en el sistema. En la figura 3.10 se muestran dos dispositivos que hacen referencia a sensores de temperatura fabricados por la empresa D&T que utilizan el modulo conversor Modbus a MQTT.

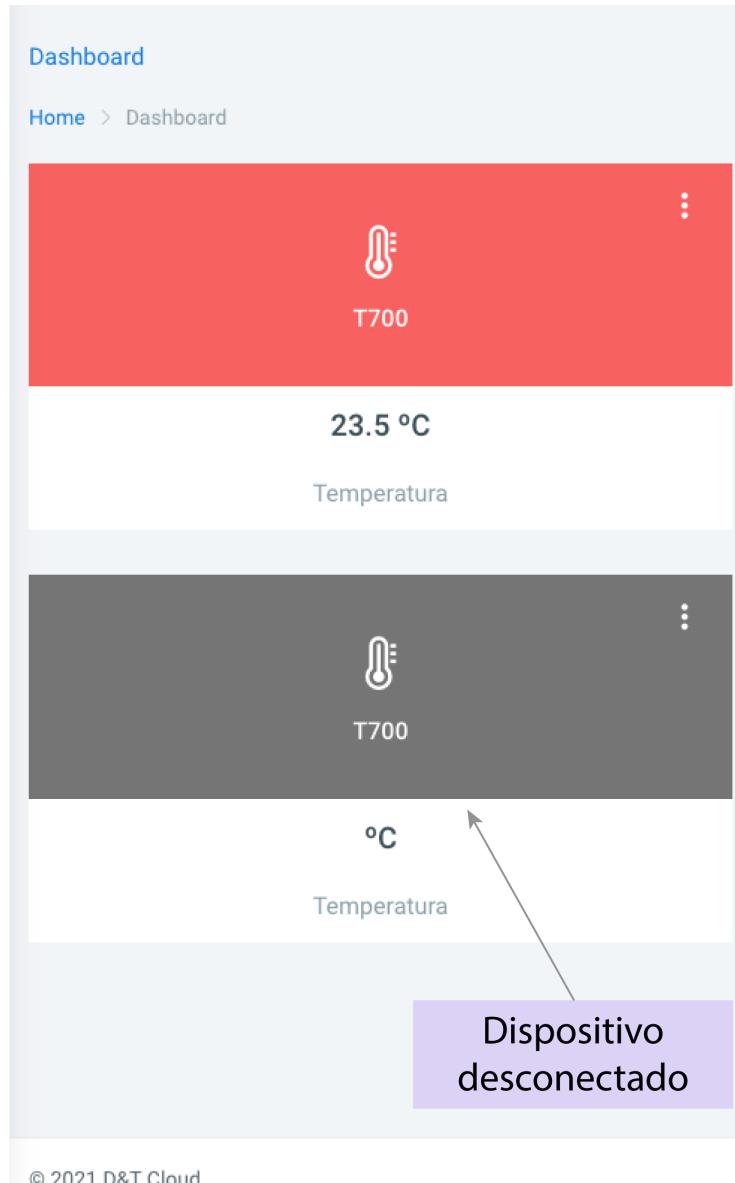


FIGURA 3.10. Ilustración de lista de sensores de temperatura T700 conectados a conversores Modbus a MQTT.

Cada bloque de sensor corresponde a un componente de Angular, el cual tiene un servicio asociado para el manejo de datos en tiempo real y además la posibilidad de acceder a sus datos históricos. En la figura 3.11 se observa en detalle las opciones de acceso implementadas en el componente.



FIGURA 3.11. Ilustración de componente utilizado para modelar el sensor T700 y sus opciones de acceso.

Si el usuario requiere realizar cambios en el dispositivo y editar parámetros como se muestra en la figura 3.12, puede ingresar directamente desde el componente y sus accesos directos, o bien buscarlo en el listado de dispositivos y presionando en el botón de edición.

The screenshot shows a user interface for editing device configurations. At the top, a blue header bar displays the word 'Edición'. Below this, the main content area has a light gray background. The title 'Configuración de Dispositivo' is centered at the top of the content area. Below the title, a sub-instruction 'Edite los campos que requiera' is displayed. The form consists of several input fields:

- Nombre:** A text input field containing the value 'T700'.
- ID:** A text input field containing the value '30:ae:a4:15:55:08'.
- Organización:** A dropdown menu currently set to 'org-1'.
- Ubicación:** A text input field containing the value 'D&T - Office'.

At the bottom of the form is a green rectangular button labeled 'Confirmar'.

FIGURA 3.12. Ilustración de pantalla de edición de dispositivos.

Para la visualización de datos históricos de dispositivos se utilizó la librería para Angular echarts [44]. Esta amplia librería permite la utilización de gráficos para mostrar datos históricos de mediciones de los dispositivos que almacenan datos en MongoDB. En la figura 3.13 puede observarse las características principales que ofrece el componente para realizar un gráfico y las opciones correspondientes en la figura 3.14.

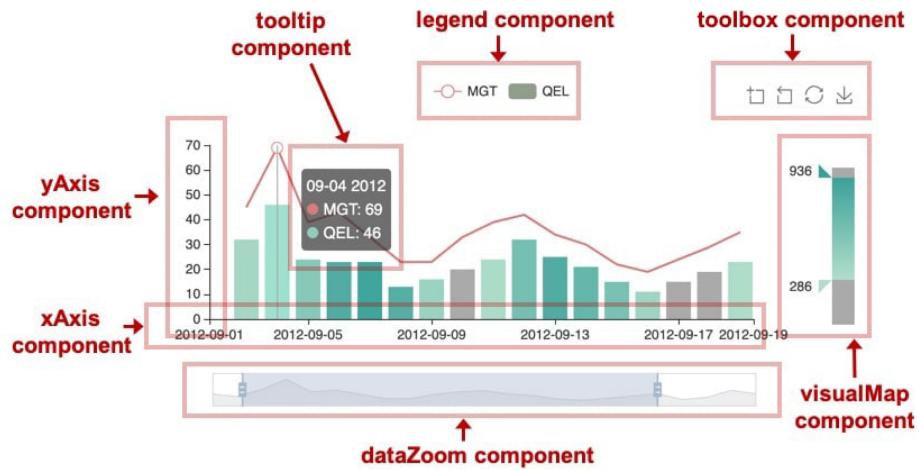


FIGURA 3.13. Ilustración de componente gráfico utilizando la librería echarts¹.

```
var option = {
    legend: {...},
    toolbox: {...},
    tooltip: {...},
    dataZoom: [..., ...],
    visualMap: {...},
    xAxis: [...],
    yAxis: [...],
    grid: [...],
    dataset: {
        source: [...]
    },
    series: [
        {
            type: 'line',
            ...
        },
        {
            type: 'bar',
            ...
        }
    ];
};
```

FIGURA 3.14. Ilustración de opciones de configuración de gráfico utilizando echarts².

¹Fragmento de imagen tomada de <https://echarts.apache.org/en/tutorial.html>

²Fragmento de imagen tomada de <https://echarts.apache.org/en/tutorial.html>

La figura 3.15 muestra el gráfico de mediciones históricas que corresponden a un sensor de temperatura que utiliza un dispositivo conversor de datos Modbus a MQTT.

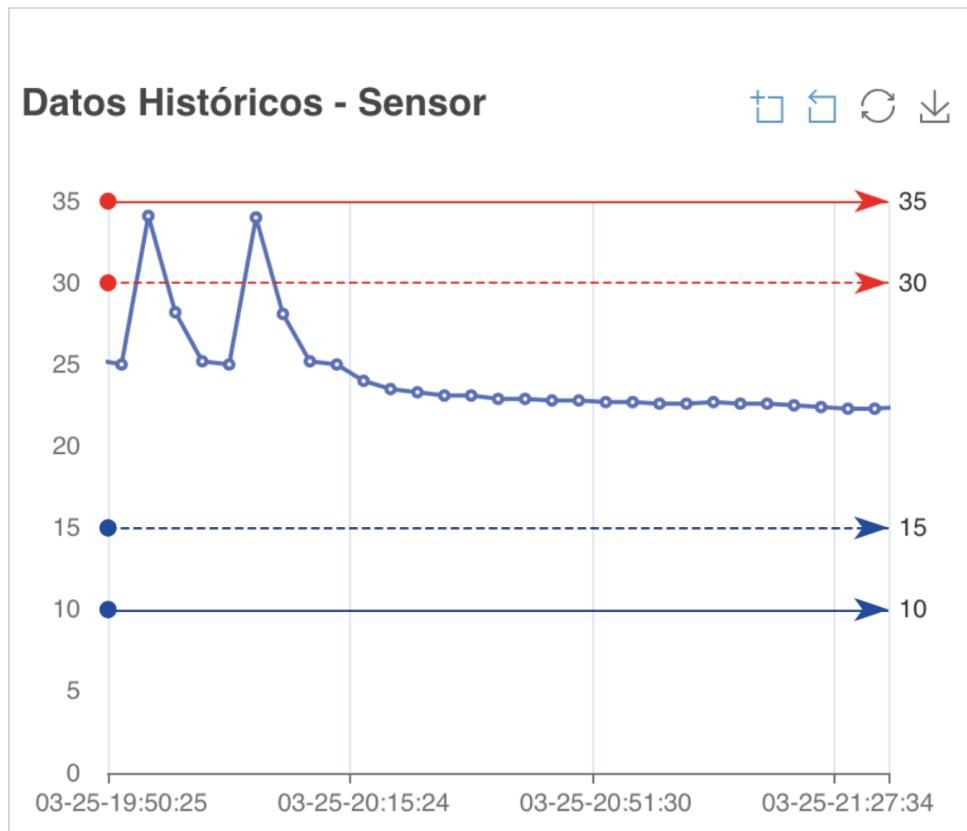


FIGURA 3.15. Ilustración de datos históricos de sensor de temperatura conectado a conversor de datos Modbus a MQTT.

El diseño web *responsive* o adaptativo es una técnica de diseño web que busca la correcta visualización de una misma página en distintos dispositivos. Desde computadoras de escritorio a tablets y móviles. Se trata de dimensionar y colocar los elementos de la web de forma que se adapten al ancho de cada dispositivo, permitiendo una correcta visualización y una mejor experiencia de usuario. Para el diseño de la plataforma se utilizaron componentes con estilos que se adaptan a diferentes tamaños de pantallas.

En la figura 3.16. se observa la visualización de la pantalla principal adaptada para pantallas que son utilizadas en PC de escritorio, se debe notar que el usuario puede visualizar el contenido completo, teniendo en cuenta la barra lateral desplegada con las opciones disponibles.

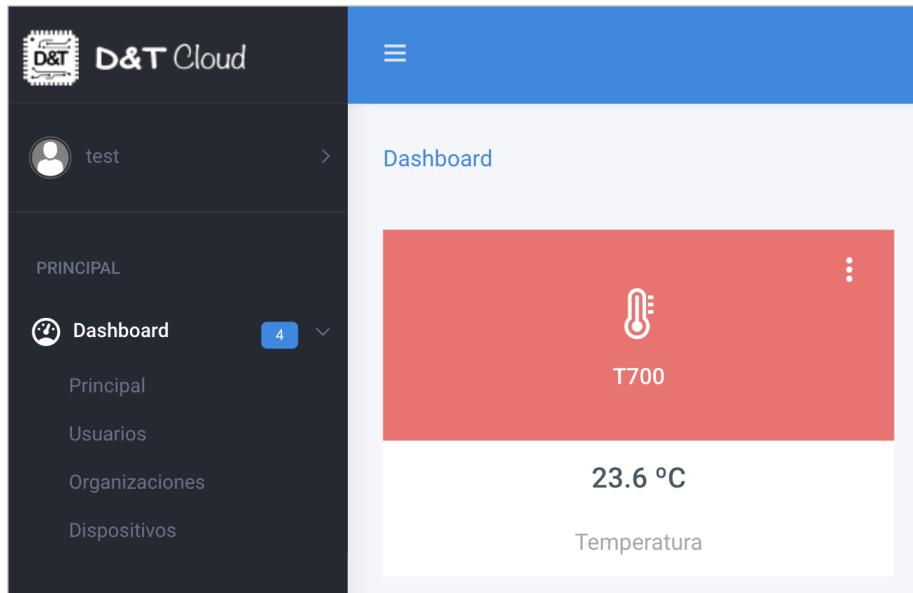


FIGURA 3.16. Ilustración de pantalla principal adaptada a pantallas de PC.

Por otro lado la figura 3.17 muestra la pantalla adaptada a tablets y que se encuentran de forma horizontal, donde se aprecia que la barra lateral se oculta y si el usuario necesita algún ítem deberá presionar el botón correspondiente para poder desplegarla.

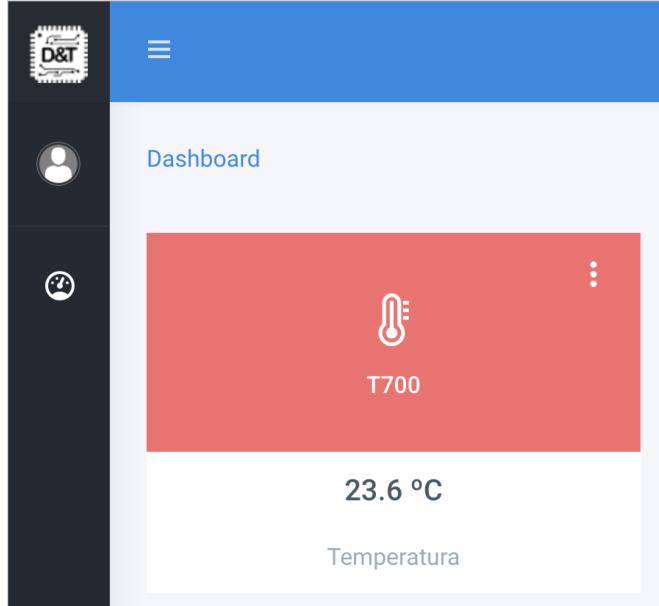


FIGURA 3.17. Ilustración de pantalla principal adaptada a tablets.

En última instancia, la figura 3.18 muestra la pantalla adaptada para celulares y se aprecia el listado de dispositivos en una sola columna y la barra lateral oculta para poder desplegarla a través de un botón.

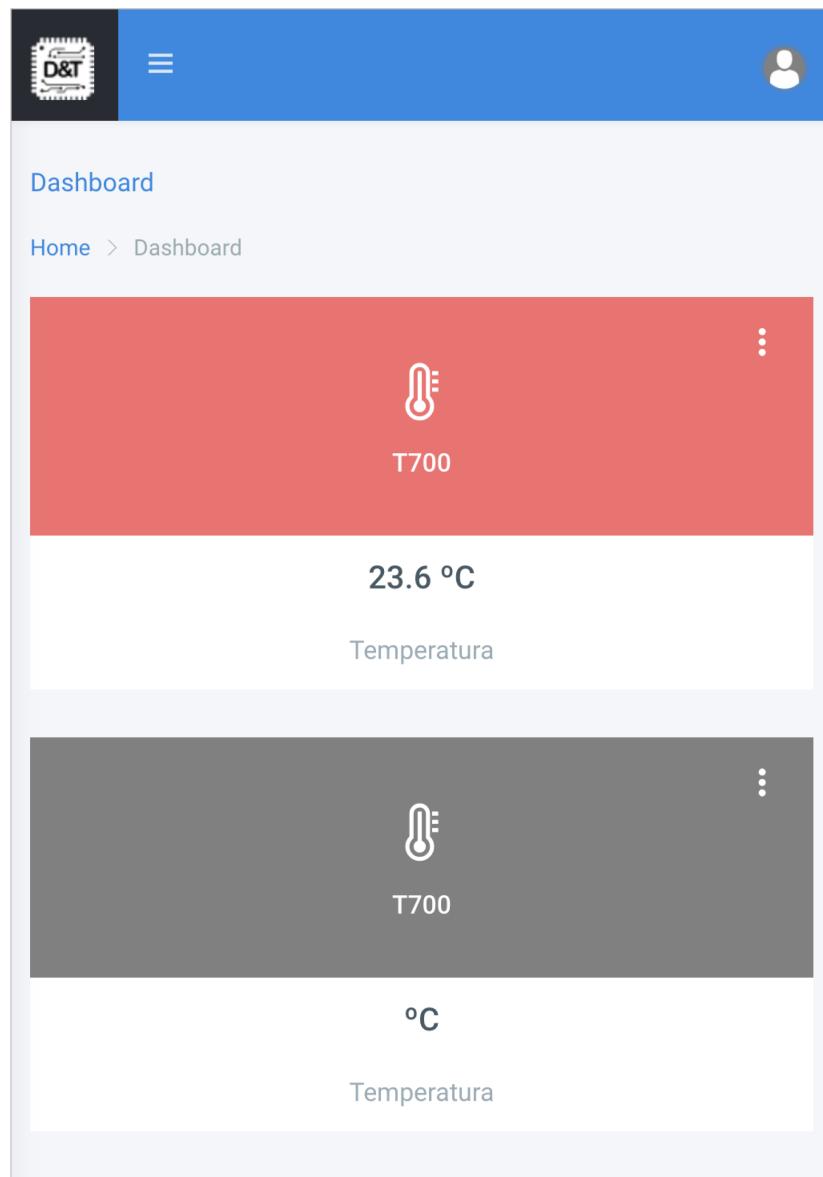


FIGURA 3.18. Ilustración de pantalla principal adaptada a celulares.

3.4. Implementación y configuración de Nginx

Para implementar Nginx en un servidor con sistema operativo Linux como el de este trabajo, se debe instalarlo con el comando que se muestra en el código 3.13.

```

1 // Instalacion de Nginx
2 sudo apt install nginx
3
4 // Aplicar ajustes al firewall
5
6 sudo ufw app list
7
8 sudo ufw allow 'Nginx HTTP'
9
10 // Comprobar que Nginx funcione en el servidor
11 systemctl status nginx
12

```

CÓDIGO 3.13. Instalación de Nginx en servidor con sistema operativo Linux.

Para la configuración de Nginx en el servidor se deben tener en cuenta los siguientes archivos y directorios y como cada uno influye en el funcionamiento:

- /etc/nginx: directorio de configuración de Nginx. En él se encuentran todos los archivos de configuración de Nginx.
- /etc/nginx/nginx.conf: archivo de configuración principal de Nginx. Esto se puede modificar para realizar cambios en la configuración general de Nginx.
- /etc/nginx/sites-available/: directorio en el que se pueden guardar bloques de servidor por sitio. Nginx no utilizará los archivos de configuración de este directorio a menos que estén vinculados al directorio sites-enabled. Normalmente, toda la configuración del bloque de servidor se realiza en este directorio y luego se habilita estableciendo un vínculo con el otro directorio.
- /etc/nginx/sites-enabled/: directorio en el que se almacenan los bloques de servidor habilitados por sitio. Normalmente, estos se crean estableciendo vínculos con los archivos de configuración del directorio sites-available.
- /etc/nginx/snippets: este directorio contiene fragmentos de configuración que pueden incluirse en otras partes de la configuración de Nginx. Los segmentos de configuración potencialmente repetibles reúnen las condiciones para la conversión a fragmentos.

Los dominios utilizados para este trabajo son:

- frontend: <https://cloud.dytsoluciones.com.ar>.
- backend: <https://api.cloud.dytsoluciones.com.ar>.

La configuración para el frontend se realizó siguiendo los pasos del código 3.14.

```

1 // Se crea el directorio con el nombre de dominio utilizado
2 sudo mkdir -p /var/www/cloud.dytsoluciones.com.ar/html
3
4 //Se asigna la propiedad del directorio con la variable de entorno $USER
5 sudo chown -R $USER:$USER /var/www/cloud.dytsoluciones.com.ar/html
6

```

```
7  
8 // Para que Nginx pueda utilizar las directivas correctas se crea un  
9 // archivo de configuración predeterminado.  
10 sudo nano /etc/nginx/sites-available/cloud.dytsoluciones.com.ar  
11  
12 // Se crea el enlace entre el archivo de configuración y el directorio  
13 sudo ln -s /etc/nginx/sites-available/cloud.dytsoluciones.com.ar/ /etc/  
14 nginx/cloud.dytsoluciones.com.ar/  
15 //Una vez finalizada la configuración se reinicia el servicio  
16 sudo systemctl restart nginx
```

CÓDIGO 3.14. Configuración de Nginx en servidor con sistema operativo Linux.

Capítulo 4

Ensayos y Resultados

En este capítulo se detallan los resultados esperados y obtenidos sobre etapas puntuales en el trabajo. También se indican las herramientas y metodología empleadas en cada caso. Finalmente se expone el caso de uso completo integrando todos los componentes que lo componen.

4.1. Pruebas unitarias

Las pruebas unitarias o *unit testing* son una forma de comprobar que un fragmento de código funciona correctamente. Para este trabajo se programó un *script* utilizando shUnit2 [45], para automatizar diferentes pruebas en la conexión con el broker MQTT.

Las pruebas unitarias en el backend fueron realizadas teniendo en cuenta las etapas de desarrollo del proyecto y así poder asegurar que en la implementación del frontend no haya problemas en las respuestas a las diferentes consultas.

Por otro lado, se utilizó la herramienta de *testing* que se incluye en Angular para realizar pruebas de las partes más importantes del frontend.

4.1.1. Pruebas de integridad del broker MQTT

Para hacer pruebas de integridad en el broker MQTT, se instaló shUnit2 y en un *script* se programaron las diferentes pruebas para verificar si la conexión de clientes con el broker MQTT es segura o bien tiene fallas de seguridad. Estas pruebas automatizadas se realizaron mientras se cursaba la materia Ciberseguridad en Internet de las Cosas.

En el código 4.1 se muestran las diferentes funciones de testeo que se incluyeron en el script. La función de cada una es lograr publicar un mensaje en el broker teniendo en cuenta diferentes etapas de autenticación.

```

1  #! /bin/sh
2
3  #####
4  # Test para saber si un cliente sin credenciales
5  # puede conectarse - Envio de mensaje prueba_test
6  #####
7
8  testClienteAnonimo() {
9    VALUE=$(mosquitto_pub -p 8883 -m prueba_test -t /test -d | grep -o '
10   PUBLISH')
11  assertFalse "$VALUE" "PUBLISH"
12 }
```

```

12 #####
13 # Test para saber si un cliente con solo usuario
14 # puede conectarse
15 #####
16 testClienteConUsuario(){
17     VALUE=$(mosquitto_pub -p 8883 -h localhost -u carlos -m hello -t /test
18         -d | grep -o 'PUBLISH')
19     assertFalse "$VALUE" "PUBLISH"
20 }
21
22 #####
23 # Test para saber si un cliente con solo usuario
24 # y password puede conectarse
25 #####
26 testClienteConUserYPass(){
27     VALUE=$(mosquitto_pub -p 8883 -h localhost -u carlos -P carlos -m hello
28         -t /test -d | grep -o 'PUBLISH')
29     assertFalse "$VALUE" "PUBLISH"
30 }
31
32 #####
33 # Test para saber si un cliente puede conectarse
34 # con certificados validos
35 #####
36 testClienteTLS(){
37     VALUE=$(mosquitto_pub -p 8883 --cafile ..//ca/ca.crt --cert ..//client/
38         client.crt --key ..//client/client.key -h localhost -u carlos -P cars
39         -m hello -t /test -d | grep -o 'PUBLISH')
40     assertEquals "$VALUE" "PUBLISH"

```

CÓDIGO 4.1. Script para testing de integridad y seguridad de broker MQTT.

En la tabla 4.1 se describen los tests realizados.

TABLA 4.1. Comandos más utilizados para el manejo de base de datos desarrollada en MongoDB.

Test	Descripción	Resultado esperado
testClienteAnonimo()	Verificar si un cliente sin credenciales puede conectarse al broker.	Error de conexión
testClienteConUsuario()	Verificar si un cliente con nombre de usuario y sin password puede conectarse al broker.	Error de conexión
testClienteConUserYPass()	Verificar si un cliente con nombre de usuario y password puede conectarse al broker.	Error de conexión
testClienteTLS()	Verificar si un cliente con nombre de usuario y password y certificados puede conectarse al broker.	Publicación exitosa

Los resultados obtenidos pueden observarse en la figura 4.1.

```
(base) carlos@carlos-D15D:~/iot_dev/certs/shunit2$ shunit2 ./test_prueba.sh
testClienteAnonimo
Error: The connection was lost.
testClienteConUsuario
Error: The connection was lost.
testClienteConUserYPass
Error: The connection was lost.
testClienteTLS

Ran 4 tests.

OK
```

FIGURA 4.1. Test de integridad y seguridad del broker MQTT.

4.1.2. Pruebas unitarias de funciones del backend

Para las pruebas unitarias de las funciones del backend se utilizó el software Postman [46], el cual permitió realizar las pruebas de todas las funciones definidas en el backend desde un único sitio.

En la figura 4.2 pueden observarse las carpetas de cada sección de funciones utilizadas en el backend.

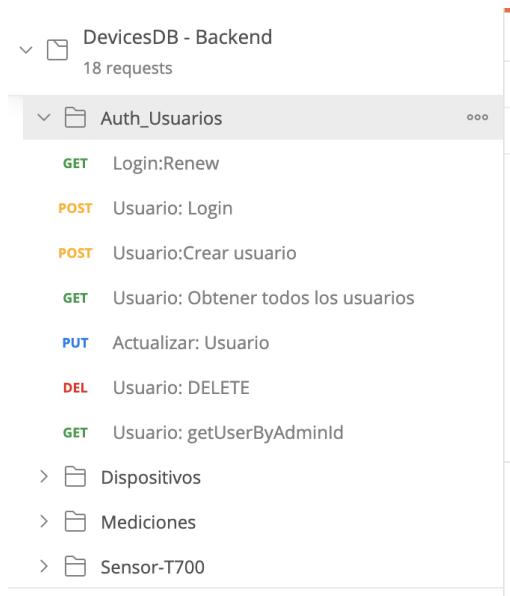


FIGURA 4.2. Carpeta con peticiones http para testing de backend en Postman.

Para hacer una petición http al backend, se seleccionó la opción de login de usuario donde se realizó una petición POST pasando como parámetros en el cuerpo del mensaje el email y password como se observa en la figura 4.3.

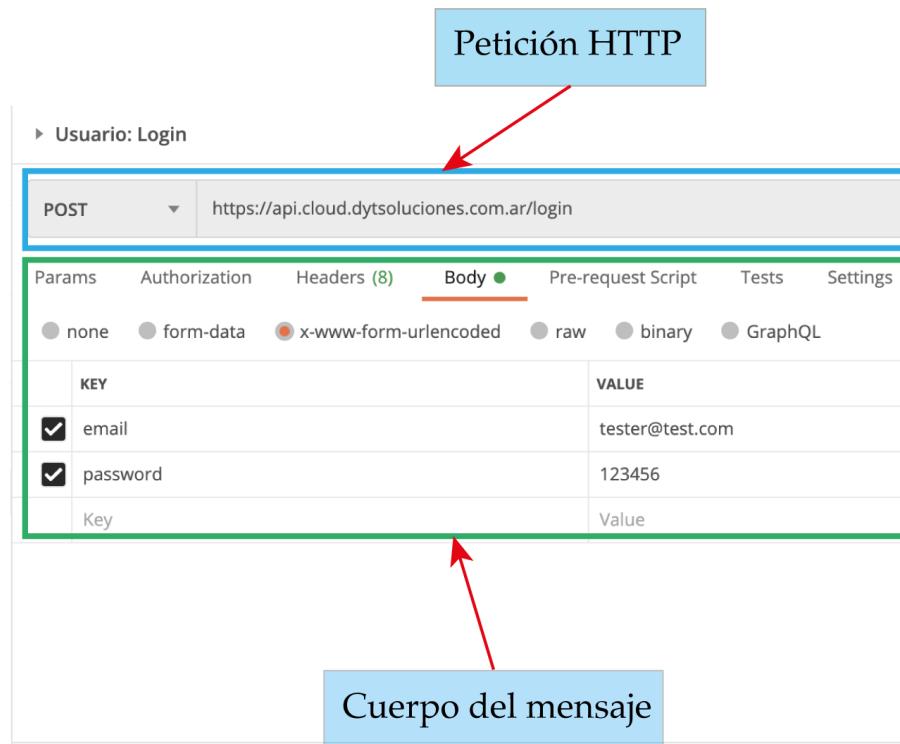


FIGURA 4.3. Ilustración de entorno Postman para hacer una petición POST a la función login de usuario del backend.

La respuesta a esta petición espera un mensaje en formato JSON con el estado. En caso de ser correcta, se espera un código 200 y además un *token* de acceso para las funciones que lo requieren. En la figura 4.4 se puede observar la respuesta a la petición POST para el login de usuario.



FIGURA 4.4. Ilustración de respuesta del backend a la petición POST de login de usuario en Postman.

Luego del test de login de usuario y de obtener un token, se realizó un test de acceso a funciones del backend que tienen como condición que el usuario envíe un token válido para poder acceder a las respuestas de las mismas.

El test consistió en realizar la petición con un token inválido esperando un estado de error en su respuesta como se muestra en la figura 4.5.

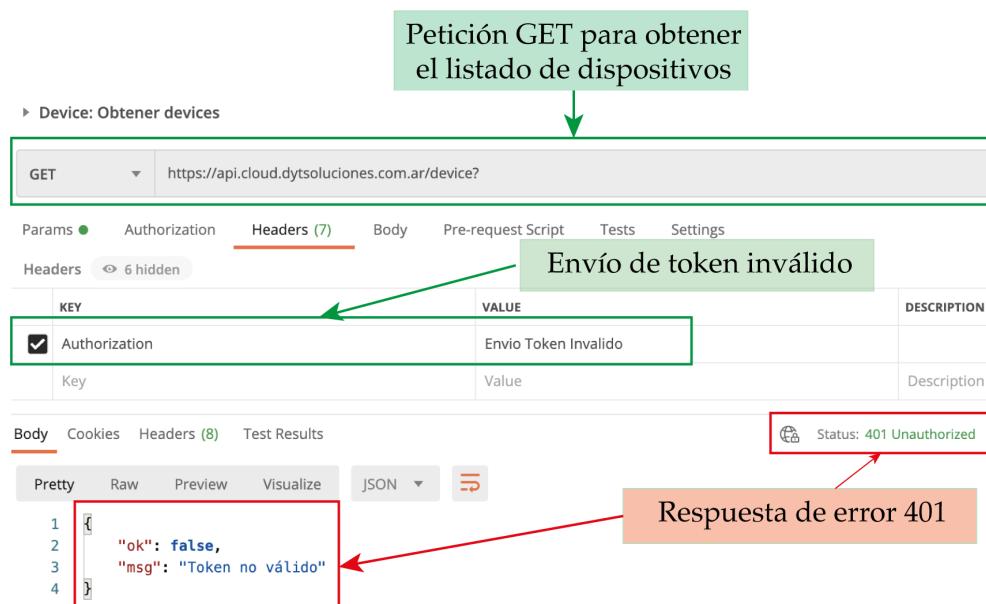


FIGURA 4.5. Ilustración de respuesta del backend a la petición GET de dispositivos con token inválido.

Por otro lado, en la figura 4.6 se realizó la petición de dispositivos al servidor utilizando el token válido adquirido en el login de usuario.

Petición GET para obtener el listado de dispositivos

Device: Obtener devices

GET https://api.cloud.dytsoluciones.com.ar/device?

Headers (7)

Envío de token valido

KEY	VALUE	DESCRIPTION
Authorization	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlaWQiOiI1ZmRhO... Key Value Description	

Status: 200 OK

Body (Pretty, Raw, Preview, Visualize, JSON)

Respuesta de válida

```

1 {
2   "ok": true,
3   "devices": [
4     {
5       "created": "2021-02-19T14:39:59.347Z",
6       "updated": null,
7       "deleted": null,
8       "_id": "602fcdbf4a2aa90984dacbdd",
9       "usuario": {
10         "_id": "5fb526876cd78b07b8be980d",
11         "nombre": "test",
12         "email": "test@test.com"
13       },
14       "nombre": "T700",
15       "mac": "30:ae:a4:15:55:08",
16       "org": {
17         "_id": "5fd7fde5eea8cc0c6743a23d",
18         "nombre": "org-1"
19       },
20       "ubicacion": "D&T - Office",
21       "__v": 0
22     }
  
```

FIGURA 4.6. Ilustración de respuesta del backend a la petición GET de dispositivos con token válido.

4.2. Ensayos de integración

El objetivo de las pruebas de integración es verificar el correcto funcionamiento entre los distintos componentes del sistema una vez que han sido probados unitariamente. El fin es comprobar que interactúan correctamente a través de sus interfaces, tanto internas como externas, cubren las funcionalidades establecidas y se ajustan a los requisitos no funcionales especificados en las verificaciones correspondientes.

Para las pruebas de todo el sistema se utilizó la herramienta Cypress [47] que es un framework que incluye librerías de aserciones, mocks y pruebas e2e [48] automáticas. Esta herramienta está diseñada especialmente para manejar frameworks de Javascript como Angular, Vue, ReactNative, entre otros.

Para la instalación de Cypress sobre Angular, se utilizó npm [49] y en el código 4.2 se puede observar el comando utilizado.

```
2 npm install cypress --save-dev
```

CÓDIGO 4.2. Comando de instalación de Cypress en Angular.

En la figura 4.7 se observa la integración de Cypress al archivo de dependencias del desarrollo de Angular y además la creación de dos archivos para el testeo del sistema en la carpeta de integración.

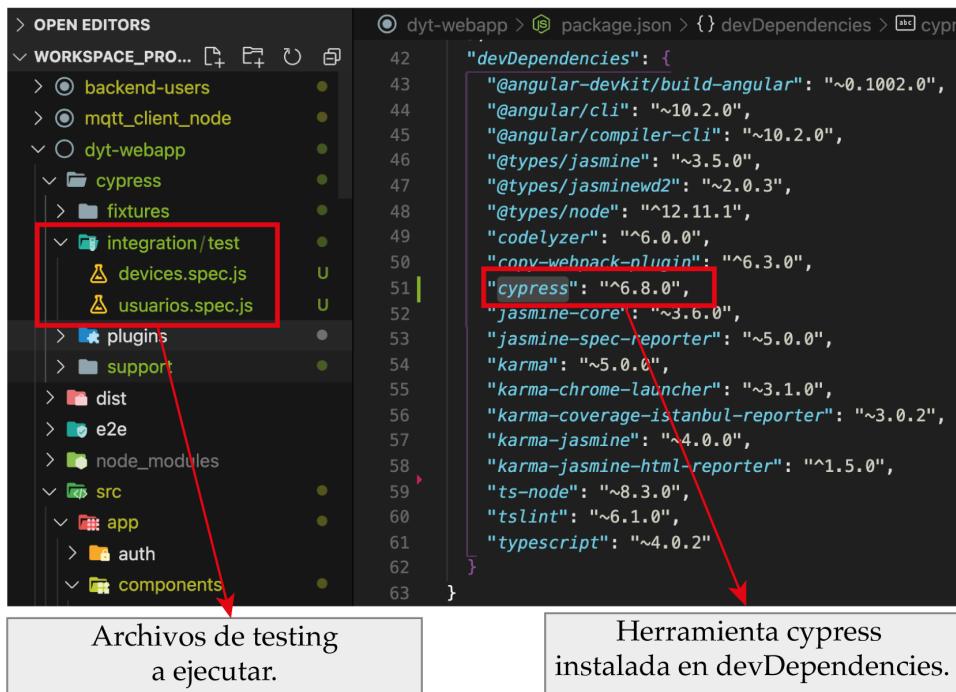


FIGURA 4.7. Ilustración archivos de testing generados en carpeta *integration/test* y herramienta instalada en dependencias de desarrollo.

4.2.1. Pruebas de integridad en login de usuarios

Esta prueba consiste en automatizar los test unitarios que se realizaron en la sección 4.1.2. Con el uso de Cypress se pudo simular el comportamiento de la pantalla de login de la plataforma web con el usuario, pudiendo observar para cada test, la respuesta que esta pantalla informa al usuario.

En el código 4.3 se muestra la implementación para los tests que se realizaron en la pantalla de login.

```
1 //<reference types="cypress" />
2
3 context('Test de pagina de login', () => {
4     // Antes de cada test, debe visitar la pantalla de login.
5     beforeEach(() => {
6         cy.visit('http://localhost:4200');
7     })
8
9
10 it('Si el campo email del formulario esta vacio, debe avisar al
11   usuario.', ()=>{
12     cy.get('.m-t-40 > .col-xs-12 > .form-control')
13       .type('none@test.com').should('have.value', 'none@test.com')
```

```
14
15     cy.get(':nth-child(3) > .col-xs-12 > .form-control')
16         .type('123456').should('have.value', '123456')
17
18     cy.get('#loginform')
19         .submit()
20         .next()
21
22     cy.get('#swal2-content')
23         .contains('Email no encontrado')
24
25 });
26
27 it('Si el email ingresado no pertenece a un usuario registrado debe
28   dar un aviso.', ()=>{
29     cy.get('.m-t-40 > .col-xs-12 > .form-control')
30         .type('test@test.com').should('have.value', 'test@test.com')
31
32     cy.get(':nth-child(3) > .col-xs-12 > .form-control')
33         .type('malPass').should('have.value', 'malPass')
34
35     cy.get('#loginform')
36         .submit()
37         .next()
38
39     cy.get('#swal2-content')
40         .contains('Credenciales incorrectas')
41
42 });
43
44 it('Si el email ingresado tiene un formato incorrecto, debe avisar al
45   usuario.', ()=>{
46     cy.get('.m-t-40 > .col-xs-12 > .form-control')
47         .type('none! test.com').should('have.value', 'none! test.com')
48
49     cy.get(':nth-child(3) > .col-xs-12 > .form-control')
50         .type('123456').should('have.value', '123456')
51
52     cy.get('#loginform')
53         .submit()
54         .next()
55
56     cy.get('.col > p')
57         .contains('Debe especificar un email valido')
58
59 });
60
61 it('Si el campo del password del formulario esta vacio, debe avisar al
62   usuario.', ()=>{
63     cy.get('.m-t-40 > .col-xs-12 > .form-control')
64         .type('none@test.com').should('have.value', 'none@test.com')
65
66     cy.get('#loginform')
67         .submit()
68         .next()
69
70     cy.get('.col > p')
71         .contains('Debe especificar un password')
72
73 });
74
75 it('Al ingresar un email y password registrado en el sistema, la
76   pagina debe navegar hacia la ruta /dashboard.', ()=>{
```

```
73     cy.get('.m-t-40 > .col-xs-12 > .form-control')
74       .type('test@test.com').should('have.value', 'test@test.com')
75
76     cy.get(':nth-child(3) > .col-xs-12 > .form-control')
77       .type('123456').should('have.value', '123456')
78
79     cy.get('#loginform')
80       .submit()
81       .next()
82
83     cy.contains('Dashboard')
84
85   });
86
87 it('Al realizar el login de un usuario registrado, se debe almacenar
88   el token en el localStorage.', ()=>{
89   cy.get('.m-t-40 > .col-xs-12 > .form-control')
90     .type('test@test.com').should('have.value', 'test@test.com')
91
92   cy.get(':nth-child(3) > .col-xs-12 > .form-control')
93     .type('123456').should('have.value', '123456')
94
95   cy.get('#loginform')
96     .submit()
97     .next()
98
99   cy.window()
100    .its("localStorage")
101      .invoke("getItem", "token")
102        .should("exist");
103
104 });
105});
```

CÓDIGO 4.3. Código de implementación para tests realizados en la pantalla de login de la plataforma web.

La descripción y resultados de los tests realizados fueron los que se detallan en la tabla 4.2.

TABLA 4.2. Diferentes test automáticos en pantalla login de usuario de la plataforma web utilizando Cypress.

Test Cypress	Descripción	Resultado
Si el campo email del formulario está vacío, debe avisar al usuario.	Presionar el botón login sin completar el campo email del formulario.	Mensaje informando al usuario que debe especificar un email válido.
Si el email ingresado no pertenece a un usuario registrado debe dar un aviso.	Realizar el login con un email no registrado en el sistema.	Mensaje de error informando al usuario que el email no fue encontrado.
Si el email ingresado tiene un formato incorrecto, debe avisar al usuario.	Realizar el login con un texto sin el símbolo en el campo email del formulario.	Mensaje informando al usuario que debe especificar un email válido.
Si el campo del password del formulario está vacío, debe avisar al usuario.	Presionar el botón login sin completar el campo password del formulario.	Mensaje informando al usuario que las credenciales son incorrectas.
Al ingresar un email y password registrado en el sistema, la página debe navegar hacia la ruta /dashboard.	Realizar el login con un email y password previamente registrado en el sistema.	Visualización de la pantalla dashboard en la plataforma web.
Al realizar el login de un usuario registrado, se debe almacenar el token en el localStorage.	Realizar el login con un email y password previamente registrado en el sistema y verificar si existe el item token en el localStorage.	El token existe en el localStorage del navegador web.

Es importante mencionar que no solamente se están automatizando los tests unitarios sino que además se están verificando los componentes HTML de la plataforma así como los datos alojados en la base de datos. Este tipo de tests también se conocen como e2e (*end-to-end*) y replican el comportamiento de los usuarios con el software en un entorno de aplicación completo. Estas pruebas verifican que los flujos que sigue un usuario trabajen como se espera. En la figura 4.8 se pueden observar desde la interfaz de Cypress los resultados de los tests realizados.

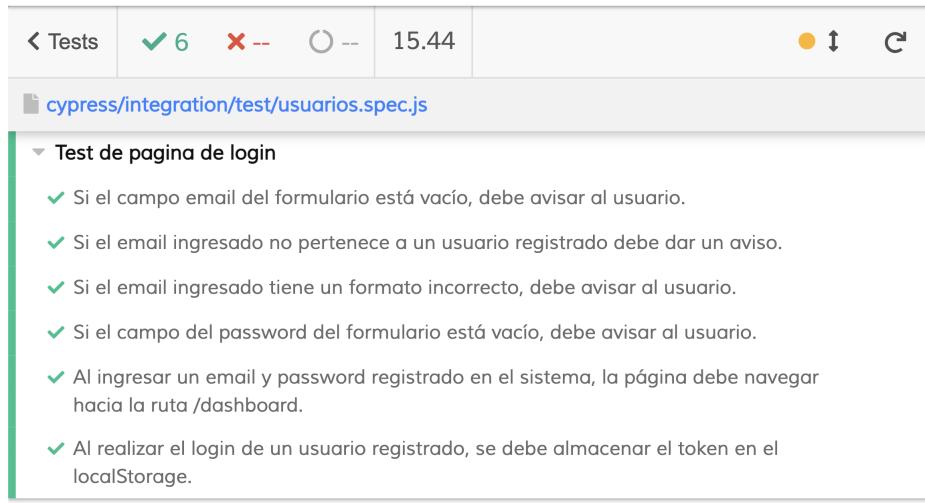


FIGURA 4.8. Ilustración del entorno web de Cypress con los resultados de los test realizados y verificados.

4.2.2. Pruebas de integridad en dispositivos conectados

Al igual que la sección 4.2.1 mediante Cypress se realizaron los tests de integración para los dispositivos del sistema. Una condición a tener en cuenta cuando se realizaron estos tests fue que el usuario llamado test tiene al menos dos dispositivos en su cuenta.

El código 4.4 muestra la implementación de cada una de las pruebas realizadas.

```

1  //> <reference types="cypress" />
2
3
4
5 context('Test de Dispositivos', () => {
6
7     Cypress.Commands.add('login', () => {
8         cy.request({
9             method: 'POST',
10            url: 'https://api.cloud.dytsoluciones.com.ar/login',
11            body: {
12                email: 'test@test.com',
13                password: '123456',
14            }
15        })
16        .then((resp) => {
17
18            window.localStorage.setItem('token', resp.body.token)
19
20        })
21    })
22
23    // Antes de cada test, debe visitar la pantalla de login.
24    beforeEach(() => {
25        cy.login()
26    })
27
28    it('En la pantalla del dashboard, deben mostrarse los dispositivos
29    asociados al usuario que realizo el login.', ()=>{
30        cy.visit('http://localhost:4200/dashboard');
31        cy.contains('Dashboard');
```

```
32     cy.get(':nth-child(1) > app-t700-sensor > .card > .row > :nth-
33         child(1) > .social-widget > .soc-header')
34             .should('exist');
35     });
36
37     it('Al navegar hacia la pantalla de dispositivos asignados al
38         usuario, debe existir un listado de estos.', ()=>{
39         cy.visit('http://localhost:4200/dashboard/devices');
40         cy.contains('Dispositivos')
41         cy.get('tbody > tr > td').eq(0)
42             .should('contain', 1)
43     });
44
45     it('En el dashboard si un dispositivo esta conectado, el color de su
46         componente no debe ser gris.', ()=>{
47         cy.visit('http://localhost:4200/dashboard');
48         cy.contains('Dashboard')
49         cy.wait(10000)
50         cy.get(':nth-child(1) > app-t700-sensor > .card > .row > :nth-
51             child(1) > .social-widget > .soc-header', {timeout: 10000})
52             .should('have.css', 'background-color', 'rgb(248, 108, 107')
53     });
54 }
```

CÓDIGO 4.4. Código de implementación para tests realizados en la interacción de usuarios con dispositivos que se encuentran en el sistema.

Por otro lado en la tabla 4.3 se puede observar la descripción de cada prueba y los resultados obtenidos pueden observarse en la figura 4.9.

TABLA 4.3. Diferentes test automáticos en pantalla login de usuario de la plataforma web utilizando Cypress.

Test Cypress	Descripción	Resultado
En la pantalla del <i>dashboard</i> , deben mostrarse los dispositivos asociados al usuario que realizó el login.	Al hacer un login de usuario, la página navega hacia el <i>dashboard</i> y se verifica que exista al menos un dispositivo.	Listado de dispositivos vinculados al usuario que realizó el login.
Al navegar hacia la pantalla de dispositivos asignados al usuario, debe existir un listado de estos.	Navegar hacia la pantalla /dashboard/devices, verificar que se listen los dispositivos asociados al usuario que realizó el login	El primer numero de la lista debe ser igual a uno, indicando que se carga el primer dispositivo.
En el <i>dashboard</i> si un dispositivo está conectado, el color de su componente no debe ser gris.	Cargar la pagina del <i>dashboard</i> , esperar diez segundos para testear si el color del componente del dispositivo cambió a un color diferente del gris indicando que está conectado.	Se visualiza el cambio de color del componente que identifica a un dispositivo conectado al sistema.

```
cypress/integration/test/devices.spec.js

Test de Dispositivos
  ✓ Debe loguearse con credenciales correctas y dirigirse a /dashboard
  ✓ Al dirigirse a dispositivos debe aparecer un listado de ellos
  ✓ Si un sensor esta conectado no debe ser de color gris

  ▼ BEFORE EACH
    1 request ● POST 200 https://api.cloud.dytsoluciones...

  ▼ TEST BODY
    1 visit http://localhost:4200/dashboard
      (xhr) ● GET 200 /login/renew
    2 contains Dashboard
      (xhr) ● GET 200 /sockjs-node/info?t=1616644132835
      (xhr) ● GET 200 /login/renew
      (xhr) ● GET 200 /usrdvs/shared/test@test.com
    3 wait 10000
    4 get :nth-child(1) > app-t700-sensor > .card >
      .row > :nth-child(1) > .social-widget >
      .soc-header
    5 - assert expected <div.soc-header> to have CSS property
```

FIGURA 4.9. Ilustración del entorno web de Cypress con los resultados de los test realizados a dispositivos asociados a un usuario de la plataforma.

Es importante mencionar que se realizó el test de cambio de color del componente de un dispositivo para poder determinar de manera indirecta que el dispositivo está recibiendo datos del broker MQTT.

En la figura 4.10 puede observarse los dos estados posibles que pueden tener los dispositivos que se muestran en la pantalla del *dashboard*.

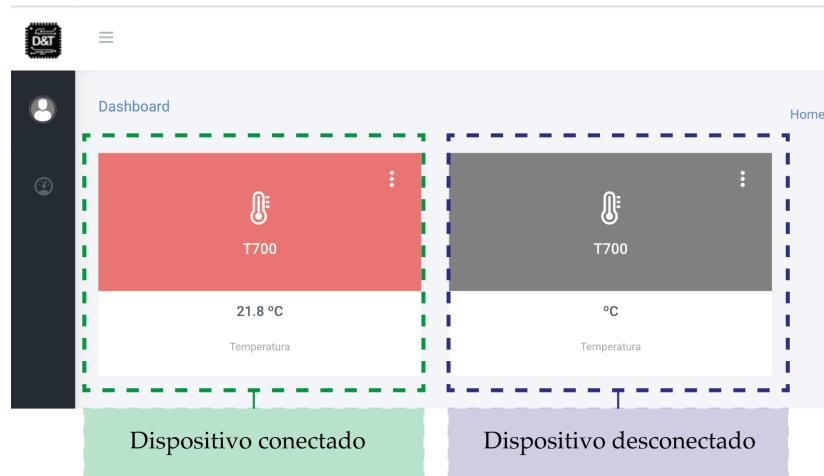


FIGURA 4.10. Ilustración de los dos estados de conexión de dispositivos en un dashboard. El dispositivo conectado (izquierda) muestra una temperatura de 21.8 °C, mientras que el dispositivo desconectado (derecha) muestra una temperatura de 0°C.

4.2.3. Ejecución automática de tests realizados

A fin de automatizar aún más la secuencia de tests de la sección 4.2.2 y la sección 4.2.1, se agregó en los comandos de ejecución de Angular un comando de Cypress para que realice todas las pruebas que se encuentran contenidas en la carpeta creada para tal fin. La figura 4.11 muestra la forma en que Angular ejecutará el comando asignado a la carpeta de test programados.

```
package.json ×
dty-webapp > package.json > {} scripts > e2e
  > Debug
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "cypress run --spec 'cypress/integration/**/*' --browser chrome"
  },
  11 | }
```

FIGURA 4.11. Ilustración de comando de ejecución de test e2e de Cypress en el navegador Chrome.

Para ejecutar el test completo, se ingresó el comando que se visualiza en el código 4.5.

¹ npm run e2e

CÓDIGO 4.5. Comando ejecutado para comienzo de tests programados en cypress.

El resultado obtenido se puede observar en la figura 4.12 y se concluye que el resultado de todos los tests fueron los esperados.

Spec	Tests	Passing	Failing	Pending	Skipped
✓ test/devices.spec.js	00:19	3	3	-	-
✓ test/usuarios.spec.js	00:11	6	6	-	-
✓ All specs passed!	00:30	9	9	-	-

FIGURA 4.12. Ilustración de respuesta de los test ejecutados en Cypress.

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 tenía 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, facilitó 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 trascendentales:

- 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 de varias de ellas.
- Contratación de un servicio de servidor remoto de producción 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.](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/.](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. *Página 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](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. Replication](https://docs.mongodb.com/manual/replication/). <https://docs.mongodb.com/manual/replication/>. (Visitado 19-03-2021).
- [27] [mongodb.org. Sharding](https://docs.mongodb.com/manual/sharding/). <https://docs.mongodb.com/manual/sharding/>. (Visitado 19-03-2021).
- [28] Hakon W Lie. *Cascading HTML style sheets - a proposal*. <https://www.w3.org/People/howcome/p/cascade.html>. (Visitado 20-03-2021).
- [29] <https://developer.mozilla.org/.About JavaScript>. https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript. (Visitado 20-03-2021).
- [30] <https://www.ecma-international.org/.ECMA-262>. <https://www.ecma-international.org/publications-and-standards/standards/ecma-262/>. (Visitado 20-03-2021).
- [31] <https://angular.io/.Angular>. <https://angular.io/>. (Visitado 20-03-2021).
- [32] <https://www.typescriptlang.org/.TypeScript es JavaScript Escalable>. <https://www.typescriptlang.org/>. (Visitado 20-03-2021).
- [33] <https://lenguajejs.com/.Que es el DOM?> <https://lenguajejs.com/javascript/dom/que-es/>. (Visitado 20-03-2021).
- [34] <https://nginx.com/.NGINX Documentation>. <https://docs.nginx.com/>. (Visitado 20-03-2021).
- [35] <http://www.kegel.com/.The C10K problem>. <http://www.kegel.com/c10k.html>. (Visitado 20-03-2021).
- [36] <http://www.redhat.com/.Qué son las API y para qué sirven>. <https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces>. (Visitado 21-03-2021).
- [37] <https://www.mongodb.com/.MongoDB Atlas>. <https://www.mongodb.com/es/cloud/atlas>. (Visitado 21-03-2021).
- [38] <https://expressjs.com/.Express. Infraestructura web rápida, minimalista y flexible para Node.js>. <https://expressjs.com/es/>. (Visitado 21-03-2021).
- [39] <https://www.npmjs.com/.MQTT.js>. <https://www.npmjs.com/package/mqtt>. (Visitado 21-03-2021).

- [40] Eclipse Foundation. *Eclipse Mosquitto™ An open source MQTT broker.* <https://mosquitto.org/>. (Visitado 21-03-2021).
- [41] https://certbot.eff.org/. *Get your site on Lock https://* <https://certbot.eff.org/>. (Visitado 21-03-2021).
- [42] sclausen. *nginx-mqtt.* <https://sclausen.github.io/nginx-mqtt/>. (Visitado 21-03-2021).
- [43] https://tools.ietf.org/. *The WebSocket protocol.* <https://tools.ietf.org/html/draft-ietf-hybi-thewebsoketprotocol-17>. (Visitado 21-03-2021).
- [44] Apache Software Foundation/. *Apache ECharts - An Open Source JavaScript Visualization Library.* <https://echarts.apache.org/en/index.html>. (Visitado 22-03-2021).
- [45] Kate Ward. *shUnit2.* <https://github.com/kward/shunit2>. (Visitado 24-03-2021).
- [46] Postman. *The Collaboration Platform for API Development.* <https://www.postman.com/>. (Visitado 24-03-2021).
- [47] Cypress. *The web has evolved. Finally, testing has too.* <https://www.cypress.io/>. (Visitado 24-03-2021).
- [48] irontec. *Introducción a la automatización de tests E2E con Cypress.io.* <https://blog.irontec.com/introduccion-automatizacion-tests-e2e-cypress-io/>. (Visitado 26-03-2021).
- [49] npm. *Build amazing things.* <https://www.npmjs.com/>. (Visitado 24-03-2021).