UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Diseño y desarrollo de una plataforma web para la monitorización a tiempo real de datos industriales

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Gómez Carbó, Joan

Tutor/a: Fons Cors, Joan Josep

CURSO ACADÉMICO: 2022/2023

Resumen

El proyecto propuesto tiene como objetivo principal rediseñar el servicio de visualización del entorno operativo de las máquinas de producción en ámbitos industriales, aprovechando las capacidades del Internet de las Cosas (IoT). Actualmente, la empresa utiliza dispositivos para controlar las máquinas de producción, los cuales se conectan a una red SCADA para supervisar y visualizar los datos. Sin embargo, la visualización actual es limitada en términos de flexibilidad, interacción y captura de datos.

El proyecto busca migrar a una tecnología más actual mediante el uso de interfaces de visualización más cómodas, estéticas y con una interacción más potente. Además, se pretende que esta solución permita la visualización del estado de los procesos en cualquier lugar a través de Internet. Se busca también facilitar la escalabilidad y personalización de las soluciones para cada cliente.

Para lograr estos objetivos, se utilizarán tecnologías como Python para el backend, Vue.js para el frontend, Mosquitto MQTT y WAMP Crossbar.io para las comunicaciones entre los PLCs y el frontend. Además, se implementará una capa de servicios REST para interconectar los componentes con el servidor. Como resultado del proyecto, se proporcionará una monitorización realista a tiempo real del proceso industrial propuesto.

Palabras clave: Internet de las Cosas, interfaces, SCADA, servicios REST, MQTT, WAMP, monitorización, tiempo real

Resum

El projecte proposat té com a objectiu principal redissenyar el servei de visualització de l'entorn operatiu de les màquines de producció en àmbits industrials, aprofitant les capacitats de l'Internet de les Coses (IoT). Actualment, l'empresa utilitza dispositius per controlar les màquines de producció, els quals es connecten a una xarxa SCADA per supervisar i visualitzar les dades. No obstant això, la visualització actual és limitada en termes de flexibilitat, interacció i captura de dades.

El projecte busca migrar cap a una tecnologia més actual mitjançant l'ús d'interfícies de visualització més còmodes, estètiques i amb una interacció més potent. A més, es pretén que aquesta solució permeti la visualització de l'estat dels processos en qualsevol lloc a través d'Internet. També es busca facilitar l'escalabilitat i personalització de les solucions per a cada client.

Per aconseguir aquests objectius, s'utilitzaran tecnologies com Python per al backend, Vue.js per al frontend, Mosquitto MQTT i WAMP Crossbar.io per a les comunicacions entre els PLCs i el frontend. A més, s'implementarà una capa de serveis REST per interconnectar els components amb el servidor. Com a resultat del projecte, es proporcionarà una monitorització realista en temps real del procés industrial proposat.

Paraules clau: Internet de les Coses, interfícies, SCADA, serveis REST, MQTT, WAMP, monitorització, temps real

Abstract

The proposed project aims to redesign the visualization service for the operational environment of production machines in industrial settings, leveraging the capabilities of the Internet of Things (IoT). Currently, the company uses devices to control the production machines, which are connected to a SCADA network for data supervision and visualization. However, the current visualization is limited in terms of flexibility, interaction, and data capture.

The project aims to migrate to a more up-to-date technology by utilizing more comfortable, aesthetically pleasing interfaces with enhanced interaction. Additionally, the solution is intended to enable process status visualization from any location through the Internet. The objective is also to facilitate scalability and customization of the solutions for each client.

To achieve these goals, technologies such as Python for the backend, Vue.js for the frontend, Mosquitto MQTT, and WAMP Crossbar.io for communication between the PLCs and the front-end will be employed. Furthermore, a REST service layer will be implemented to interconnect the components with the server. The project will provide a realistic real-time monitoring of the proposed industrial process as an outcome.

Key words: Internet of Things, interfaces, SCADA, REST services, MQTT, WAMP, monitoring, real-time

Índice general

Índice general v
Índice de figuras vii
Índice de tablas viii

1 Introducción **1**
1.1 Motivación . . . 1
1.2 Objetivos . . . 2
1.3 Estructura de la memoria . . . 3
1.4 Metodología de trabajo . . . 4
2 Estado del arte **7**
2.1 Sistemas SCADA . . . 8
2.1.1 Principales características del sistema SCADA . . . 9
2.1.2 Componentes Hardware de un sistema SCADA . . . 9
2.1.3 Desventajas de un sistema SCADA . . . 11
2.2 Frameworks IoT . . . 11
2.2.1 Descripción y ejemplos . . . 11
2.2.2 Desventajas de los frameworks IoT . . . 12
2.3 Propuesta del trabajo . . . 12
3 Análisis del problema **15**
3.1 Contexto del problema . . . 15
3.2 Requisitos específicos del sistema . . . 16
3.2.1 Técnicas de extracción de requisitos utilizadas . . . 16
3.2.2 Lista de requisitos extraídos . . . 20
3.2.3 Plan de trabajo . . . 21
3.3 Casos de uso . . . 22
4 Diseño **25**
4.1 Arquitectura y componentes . . . 25
4.1.1 Componentes del sistema . . . 25
4.1.2 Arquitectura del sistema . . . 31
4.2 Comunicaciones entre componentes . . . 32
4.2.1 Comunicaciones con el backend . . . 32
4.2.2 Comunicaciones con el frontend . . . 32
4.2.3 Formato de datos canónico . . . 33
5 Desarrollo **35**
5.1 Entorno de desarrollo . . . 35
5.2 Desarrollo de los componentes . . . 35
5.2.1 Broker MQTT - Mosquitto . . . 35
5.2.2 Módulo Principal - Python . . . 37
5.2.3 Broker WAMP - Crossbar.io . . . 38

5.2.4 API Rest - Python 39
5.2.5 Base de datos - PostgreSQL 41
5.2.6 Proyecto Vue.js 43

6 Implantación **47**
6.1 Sistema Ubuntu Server 47
6.2 Servidor Apache 2.0 47
6.3 Implantación del proyecto en el servidor 49

7 Pruebas **51**
7.1 Diseño de las pruebas 51
7.2 Resultados 52
7.2.1 Primera prueba 52
7.2.2 Segunda prueba 53
7.2.3 Tercera prueba 53

8 Conclusiones **55**
8.1 Tiempos estimados y tiempos reales 56
8.2 Relación con los estudios cursados 57

9 Trabajos futuros **59**

Bibliografía **61**

Apéndice

A Objetivos de Desarrollo Sostenible **63**

Índice de figuras

2.1 Pantalla de un sistema SCADA 9
2.2 Niveles de comunicación de un sistema SCADA 10
2.3 Esquema de la solución propuesta 13

3.1 Primer prototipo: indicadores a tiempo real 17
3.2 Primer prototipo: datos históricos 18
3.3 Primer prototipo: gráfica de datos 18
3.4 Primer prototipo: imagen del proceso 19
3.5 Segundo prototipo 20
3.6 Primer caso de uso: comunicación a tiempo real 22
3.7 Segundo caso de uso: comunicación de datos almacenados 22

4.1 Primer componente del sistema 26
4.2 Segundo componente del sistema 27
4.3 Tercer componente del sistema 27
4.4 Componente de almacenamiento de datos en el sistema 28
4.5 Diagrama de la base de datos 28
4.6 Pantalla de inicio 29
4.7 Pantalla de login 29
4.8 Pantalla de visualización 30
4.9 Componente API Rest 30
4.10 Diagrama completo de los componentes del sistema 31
4.11 Formato de datos canónicos que recibe el sistema 33
4.12 Formato de datos canónicos que envía el sistema 34

5.1 Estado del broker Mosquitto 36
5.2 Diagrama del módulo principal 37
5.3 Inicialización de contenedor Crossbar.io 39
5.4 Puesta en marcha de la API 40
5.5 Diagrama de tablas generado por DBeaver 42
5.6 Ejemplo de tabla de un dispositivo 43
5.7 Interfaz de usuario de la plataforma 44
5.8 Filtro de variables en la gráfica 44

6.1 Estructura de directorios de Apache en un sistema Ubuntu 48
6.2 Carpeta "dist"del proyecto Vue.js 49

7.1 Registros de la terminal 52
7.2 Fallo en el inicio de sesión 52
7.3 Filtro temporal de la gráfica 53
7.4 Gráfica con datos históricos 53
7.5 Tabla creada de forma dinámica 54

A.1 Objetivos de Desarrollo Sostenible 63

Índice de tablas

3.1 Plan de trabajo 21

5.1 API Rest - Endpoint usuarios 40
5.2 API Rest - Endpoint número dashboards 40
5.3 API Rest - Endpoint dashboards 40
5.4 API Rest - Endpoint dispositivo 41
5.5 API Rest - Endpoint token de sesión. 41

8.1 Plan de trabajo con tiempos reales 56

CAPÍTULO 1
Introducción

Hoy en día millones de dispositivos han adquirido la capacidad de conectarse a Internet, recopilar datos y comunicarse con otros dispositivos a través de la red. Todas estas tecnologías las clasificamos en el campo de Internet de las Cosas (IoT). Dichos dispositivos tienen una amplia capacidad de aplicación, por lo cual uno de los sectores donde se popularizaron más rápidamente fue la industria.

Sin embargo los dispositivos IoT no suelen tener grandes capacidades de procesamiento de los datos. Suelen emplearse como activadores o recolectores de datos, para posteriormente retransmitirlo a otras tecnologías más potentes.

En sus inicios, los dispositivos interconectados mediante una red se solían utilizar en las industrias a puerta cerrada, es decir, sin contacto con otras redes como Internet, con el objetivo de automatizar procesos.

Pero este paradigma de comunicación ha crecido en la sociedad los últimos años debido a los avances en la tecnología y al aumento del número de dispositivos electrónicos domésticos que posee el usuario medio, así como el desarrollo de coches inteligentes, dispositivos "wearables" (relojes inteligentes, pulseras, etc.), domótica, etc.

En el presente trabajo cooperaremos con una industria para migrar su implementación actual de sus dispositivos inteligentes al nuevo paradigma mencionado anteriormente. Diseñaremos una solución rentable que resuelva los problemas que poseen actualmente y se adapte a las necesidades del cliente.

1.1 Motivación

En este apartado se fundamenta la motivación detrás de todos los aspectos relacionados con el proyecto que se llevará a cabo. Para ello se especifican las motivaciones personales que explican el contenido y la orientación del trabajo, las motivaciones técnicas, los objetivos de aprendizaje y posibles propósitos profesionales posteriores. De esta manera, se exponen los motivos que han impulsado la realización de este proyecto, los cuales son los siguientes:

- Aprender una tecnología para el desarrollo de frontend. Hablo del framework de JavaScript de código abierto conocido como Vue.js. Se trata de uno

de los frameworks más flexibles del mercado y con una documentación muy completa que permiten su rápido aprendizaje.

- Aplicar el máximo número de conocimientos posibles aprendidos durante la carrera. No aplico únicamente los conceptos de la rama de Tecnologías de la Información, sino que rememoro todo lo dado y aplico cada conocimiento en el momento que corresponde. Desde bases de datos y servidores, hasta integración de diferentes tecnologías, programación, etc.

- Mi principal interés en diseñar y desarrollar un proyecto yo solo desde el inicio. Es mi primer proyecto con un posible alcance fuera del educativo. Además, considero que se trata de un proyecto complejo ya que, debo conocer tanto tecnologías de desarrollo de frontend como de backend.

- Crear una infraestructura de código abierto a partir de tecnologías de código abierto. Quisiera aportar mi grano de arena al desarrollo libre. Además, la plataforma podría usarse para ayudar en la comunicación entre puestos de trabajo y la cooperación.

- Aprendizaje de Python, uno de los lenguajes de programación más utilizados actualmente en el desarrollo. Por desgracia no hemos podido verlo en la carrera, pero personalmente me parece muy potente y fácil de aprender. Creo que es un buen recurso para utilizar durante mi crecimiento profesional.

- Conocer las diferentes formas de comunicación entre dispositivos, los protocolos que pueden tener, los formatos de datos que pueden enviar. He trabajado con JSON, pero ahora sé que hay muchos otros que pueden ser útiles según la necesidad.

- Poder encontrar una solución a un problema. Tener un pensamiento crítico frente a un acontecimiento y poder resolverlo de forma eficiente. Poder tomar decisiones y asegurarse de que son las correctas o retractarse y barajar distintas opciones alternativas. Esta motivación nace del conocimiento que la universidad me ha aportado a lo largo de la carrera.

Estas motivaciones dejan claro por qué llevar a cabo este proyecto resulta de mi interés. Además sirve como punto de partida de los objetivos que se describen a continuación.

1.2 Objetivos

El objetivo principal de este trabajo de fin de grado consiste en el desarrollo de una infraestructura que permita la monitorización de datos industriales, migrando de la tecnología existente a una más actual. Además del desarrollo de una solución, donde los usuarios registrados puedan acceder a sus pantallas de visualización. Para poder cumplir con este objetivo se tienen que llevar a cabo los siguientes objetivos secundarios:

- Establecer comunicaciones entre los distintos dispositivos que participan en el sistema.
- Hacer uso de formatos de datos canónicos y definir su explotación en el sistema.
- Elaborar un mecanismo de persistencia de los datos en el sistema.
- Tener en cuenta la flexibilidad y escalabilidad de la solución en todo el proceso, tanto por la parte del backend con los dispositivos, como por la parte del frontend con los usuarios que se conecten a la plataforma.
- Diseñar una solución que permita al sistema la transmisión de datos en tiempo real.
- Diseñar un mecanismo complementario que permita a los clientes visualizar datos históricos.
- Aprender a gestionar correctamente el tiempo del proyecto y cumplir con las fechas de entrega.
- Aportar un pensamiento crítico a cada decisión del proyecto.

Estos objetivos serán la guía que orientará el desarrollo y la implementación de la solución propuesta. Son indispensables para poder especificar aquello que se desea conseguir con este trabajo y el propósito que servirá como punto de partida de todo el proyecto.

1.3 Estructura de la memoria

Esta memoria está compuesta por diez capítulos los cuales siguen el siguiente orden:

Capítulo 1 – Introducción: se introduce la idea general del proyecto. También se expone la motivación principal para el desarrollo de este, los objetivos a alcanzar, la metodología empleada y la estructura del documento.

Capítulo 2 – Estado del arte: se exponen otras tecnologías existentes en el mercado que realizan funcionalidades similares al proyecto propuesto. Además, se muestra la propuesta y la diferencia del producto desarrollado con los ya existentes.

Capítulo 3 – Análisis del problema: se especifican los requisitos. Teniendo en cuenta la metodología utilizada para este proyecto, esta fase es la más importante. Los requisitos extraídos no pueden ser ambiguos, deben de estar bien especificados. Además, se desarrollan casos de uso que usaremos más adelante en el apartado de pruebas.

Capítulo 4 – Diseño: se presenta la arquitectura del sistema desarrollado, el diseño detallado de la parte software y de la parte hardware, además del razonamiento de por qué se escoge dicha tecnología.

Capítulo 5 – Desarrollo: en este capítulo se muestra el desarrollo del proyecto. Se explica cada componente software detalladamente y se mencionan los problemas y soluciones que han surgido a lo largo de esta etapa.

Capítulo 6 – Implantación: se presentan los pasos a realizar en la instalación de la solución desarrollada a un servidor local y la configuración que se debería realizar de este.

Capítulo 7 – Pruebas: se exponen las pruebas realizadas para la comprobación del correcto funcionamiento. En este apartado se utilizan los casos de uso descritos en la fase de análisis.

Capítulo 8 – Conclusiones: se comentan las conclusiones a las que se han llegado tras la realización del TFG. Además se relacionan las asignaturas cursadas con las tareas de este proyecto.

Capítulo 9 – Trabajos futuros: en este capítulo se presentan los trabajos futuros que se podrían realizar a partir de este proyecto.

Capítulo 10 – Referencias: se comparten los enlaces de las referencias utilizadas para la documentación y realización del proyecto.

1.4 Metodología de trabajo

La metodología que hemos utilizado para la realización de este trabajo de fin de grado ha sido la técnica en cascada. Esta técnica, también conocida como desarrollo en cascada, se trata de un enfoque tradicional y lineal para el desarrollo de software. Se basa principalmente en la idea de que el desarrollo puede dividirse en una secuencia de fases distintas y secuenciales. Cada fase tiene su base en la finalización de la fase anterior.

El desarrollo en cascada sigue una estructura rígida y planificada, con un enfoque en la documentación exhaustiva y una planificación detallada antes de comenzar la implementación. La diferencia más destacada entre esta metodología y las vertientes ágiles es que, las metodologías ágiles se centran menos en la planificación inicial. En cambio utilizando este tipo de metodología en cascada los requisitos iniciales deben estar claramente definidos.

Generalmente, todo proyecto que utilice esta metodología se puede dividir en las siguientes fases:

- Análisis de requisitos: esta fase es la más importante. Se deben reunir y especificar mediante las técnicas convenientes todos los requisitos del proyecto a implementar en el desarrollo. Posteriormente, se deciden qué tareas habrá que completar para llegar al resultado final y se establece el plan de proyecto con los costes.
- Diseño: una vez establecidos los requisitos, se procede al diseño del sistema. Se crea una representación visual de la estructura y el flujo del software. Esto incluye el diseño de la arquitectura, la interfaz de usuario, la base de datos y otros componentes.

- Implementación: en esta fase, los programadores traducen el diseño del software en código real. Se desarrollan los diferentes módulos y componentes del sistema de acuerdo con las especificaciones y el diseño establecido previamente.
- Pruebas: ulteriormente a la implementación, se somete el proyecto a una fase de pruebas con el fin de verificar el correcto comportamiento del software. Se realizan pruebas exhaustivas para identificar los errores, fallos o comportamientos inesperados. Estas pruebas pueden incluir pruebas unitarias, pruebas de integración y pruebas de aceptación.
- Despliegue: Una vez completadas las pruebas y se hayan corregido los posibles errores, el software se implanta en el entorno de producción.
- Mantenimiento: después del despliegue en producción, el software se encuentra en funcionamiento. En esta etapa se realizan actualizaciones, correcciones de errores y mejoras si son necesarias. El mantenimiento es una fase esencial para garantizar el rendimiento y la funcionalidad del software a lo largo del tiempo.

Si bien la metodología en cascada tiene algunas desventajas en comparación con las metodologías ágiles, también tiene ventajas que pueden ser relevantes en ciertos contextos:

- El modelo es simple y fácil de usar.
- Es fácil de administrar ya que cada fase consta de entregables específicos.
- El proceso es predecible, es decir, se tiene una idea con anterioridad de cómo evolucionará el proyecto.
- Las fases no se superponen. Se ejecuta secuencialmente una detrás de la otra.
- La documentación exhaustiva puede ser beneficiosa en proyectos con trazabilidad rigurosa.

En resumen, hemos decidido recurrir a este tipo de metodología ya que en este proyecto encontramos una visión clara del producto final, los clientes no pueden alterar los requisitos en mitad del proyecto, y por lo tanto no hay requisitos ambiguos.

CAPÍTULO 2
Estado del arte

En los últimos años, hemos experimentado un notable crecimiento de uso de las tecnologías pertenecientes al sector conocido como Internet de las Cosas. Este concepto es definido en la Referencia [5] como el concepto de conectar múltiples dispositivos a través de una red, por la que pueden intercambiar información.

Si aplicamos este concepto al entorno industrial, obtenemos un nuevo concepto apodado Industrial IoT o IIoT, referido a todos los dispositivos que intervienen y se comunican en procesos industriales. Según el arículo "The Industrial Internet of Things" [2], el IIoT constituye un mundo cuya visión es aquella en la que los activos conectados operan como parte de un sistema que conforman la empresa inteligente. Esta tecnología tiene la reputación de haber creado la cuarta revolución industrial, siendo una tendencia que numerosas corporaciones adoptan en sus negocios de mercado. IIoT es el requisito fundamental para la llamada Industria 4.0, correspondiente a una industria compuesta por sensores, cámaras, sistemas... interconectados y los cuales captan y transmiten información.

Sin embargo, la Industria 4.0 no se queda aislada en un recinto, si no que comprende tanto los dispositivos físicos como los sistemas que intervengan en las comunicaciones, las plataformas, las aplicaciones, las tecnologías de comunicación e interacción, tanto dentro como fuera de la industria.

Partiendo de esta premisa, en esta nueva industria la tecnología suele emplearse para la monitorización de los procesos industriales a tiempo real con el objetivo de obtener una mayor eficiencia y calidad del producto. Además de comprobar el estado de los procesos, se pueden identificar problemas y ayudar a los operarios en la toma de decisiones.

Apoyándonos de nuevo en el libro "Industrial Internet of Things"[5], podemos observar que las soluciones y herramientas emergentes con Inteligencia Artificial o AI, también han sido aplicadas a la Industria 4.0. Este software es utilizado para el análisis y procesado de grandes cantidades de información. Puede hacer uso de la disciplina denominada "Machine Learning" para identificar ciertos patrones y elaborar predicciones con la finalidad de proporcionar información de mayor calidad al operario o evitar fallas en los procesos sin necesidad de intervención humana. Cabe mencionar que en este trabajo no se aplicará ninguna solución relacionada con AI, pero puede ser interesante para próximos pasos en el proyecto.

La mayoría de las soluciones actuales de monitorización de datos industriales en tiempo real utilizan sistemas SCADA (Supervisory Control and Data Acquisition) para recopilar y analizar los datos de los sensores en tiempo real. Estos sistemas han demostrado ser eficaces para supervisar y controlar los procesos, pero también tienen limitaciones, como ya se comentó en la introducción, en términos de escalabilidad, flexibilidad y seguridad.

También existen distintas herramientas para el desarrollo de soluciones IoT que ofrecen conectividad con el exterior de la industria. Se hablará de estos frameworks IoT más adelante y las razones por las que no se ha optado por su uso en este trabajo.

2.1 Sistemas SCADA

Como ya se ha mencionado anteriormente, la automatización industrial consiste en gobernar la actividad y la evolución de los procesos de forma automatizada, sin necesidad de la intervención humana. Para ello se ha desarrollado un sistema denominado SCADA, cuyas siglas en inglés significan Control de Supervisión y Adquisición de Datos. A través de este sistema se pueden supervisar y controlar todas la variables que participan en un proceso o planta. A si mismo, se deben utilizar diversos periféricos, software de aplicación, sistemas de comunicación, etc., que permiten al operador de planta tener control y acceso a través de una pantalla de computador.

Un sistema SCADA se puede definir como una aplicación o conjunto de aplicaciones software especialmente diseñadas para trabajar sobre ordenadores de control de producción, con acceso directo a la industria y una interfaz gráfica de alto nivel para su visualización.

Exiten sistemas, tales como autómatas o PLC, que permiten controlar la integración y comunicación entre dispositivos a través de una red ethernet junto con el sistema SCADA. Esta tecnología suele emplearse en el control de oleoductos, sistemas de transmisión de energía eléctrica, yacimientos de gas y petróleo, redes de gas natural, entre otras. En la Figura 2.1 vemos un ejemplo de visualización de un sistema SCADA a través de un monitor.

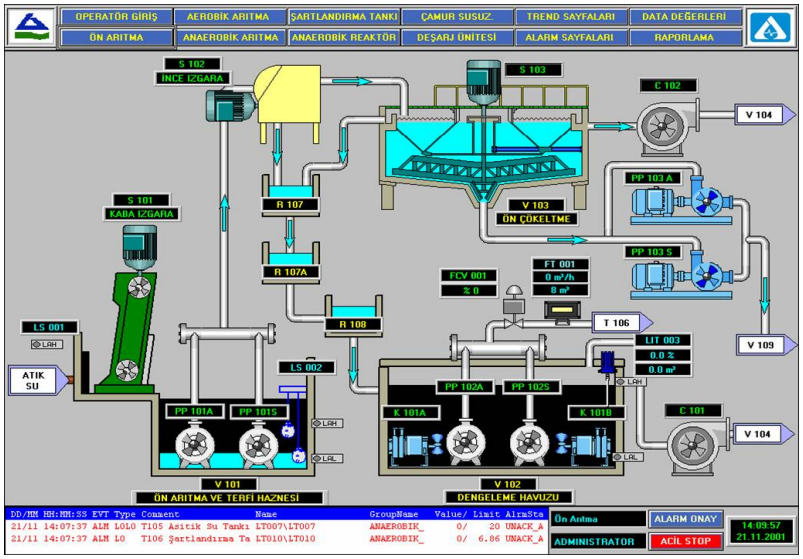

Figura 2.1: Pantalla de un sistema SCADA

2.1.1. Principales características del sistema SCADA

Tal y como se comenta en la Referencia [1], las principales características de los sistemas SCADA son:

- Adquisición y almacenado de datos para recoger, procesar y almacenar la información recibida.
- Representación gráfica y animada de variables de proceso.
- Monitorización por medio de alarmas.
- Ejecutar acciones de control para modificar la evolución del proceso.
- Arquitectura abierta y flexible con capacidad de ampliación y adaptación.
- Conectividad con otras aplicaciones y bases de datos.
- Supervisión a través de un monitor.
- Explotación de los datos adquiridos para una gestión de calidad.

2.1.2. Componentes Hardware de un sistema SCADA

Un sistema SCADA necesita de componentes inherentes de hardware para poder tratar la información captada. Algunos de los componentes principales son los siguientes:

- Ordenador Centrar o MTU. Se trata de un ordenador principal. Este supervisa y recoge información del resto de las subestaciones.

- Ordenadores remotos o RTU. Son ordenadores que se encuentran situados en nodos estratégicos. Estos reciben las señales de los sensores de campo y transmiten la información a la MTU. No tienen porque ser un PC. Una tendencia actual es dotar a los controladores lógicos programables (PLC) con capacidad de funcionar como RTU. Esta estrategia es sobre la que se trabaja en este proyecto.

- Red de comunicación. Hoy en día, gracias a la estandarización de las comunicaciones de campo, es posible implementar un sistema SCADA sobre prácticamente cualquier tipo de BUS. Todos estos tipos se comunican mediante el protocolo TCP/IP. Asumiremos en nuestro proyecto una conexión ethernet cableada, pero puede ser posible conexiones Bluetooth, microondas o incluso satélite.

- Instrumentos de Campo. Todos aquellos que realizan la automatización y captan información del sistema para comunicarlo a las capas superiores. Su programación e implementación está fuera del alcance de este proyecto.

En la Figura 2.2 se observa un diagrama de niveles sobre las partes hardware mencionadas. La MTU sería el nivel más alto, pues toda información acaba llegando a este ordenador central. Por debajo se encuentran los RTU o en este caso PLCs. Y como último nivel, todo tipo de sensores, actuadores, hardware que capte la información del entorno y la transmita a los niveles superiores. La red de comunicación es bidireccional, es decir, que la información fluye tanto hacia los niveles superiores como hacia los niveles inferiores.

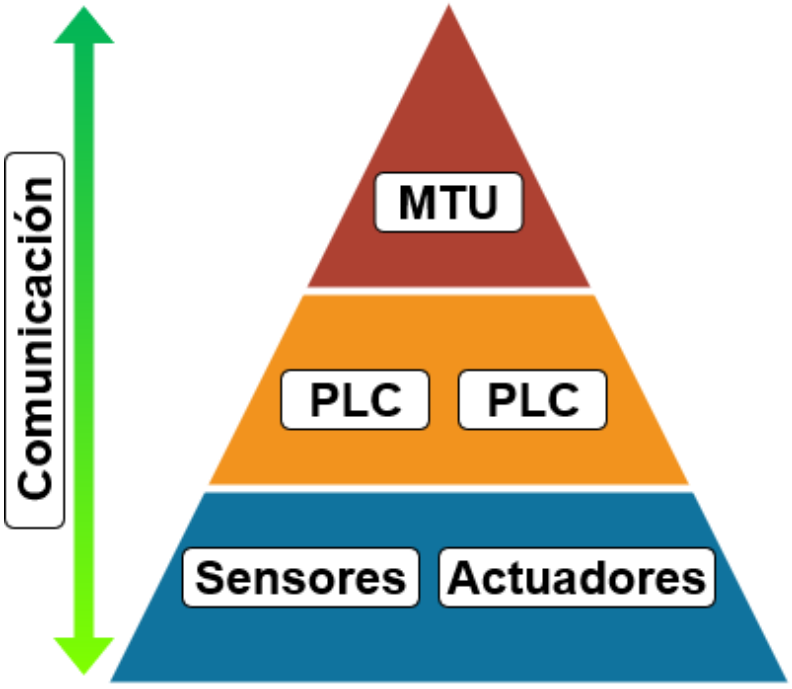

Figura 2.2: Niveles de comunicación de un sistema SCADA

2.1.3. Desventajas de un sistema SCADA

Hasta este punto, se han expuesto todas las ventajas y facilidades que aporta el sistema SCADA a la automatización. No obstante, también presenta desventajas. Algunas de estas son:

1. Vulnerabilidades de seguridad. Los sistemas SCADA son vulnerables a ataques de redes de comunicación. Mayormente están conectados a redes de área local, aislados de Internet, es por ello que muchas empresas optan por no invertir demasiados recursos en mejorar la seguridad.

2. Riesgo de interrupción en la comunicación. Si la red de comunicaciones entre los dispositivos y el sistema central se interrumpe, puede haber pérdida de datos o falta de control a tiempo real. Esto puede originar retardos en la producción o dificultades a la hora de reaccionar a eventos críticos.

3. Costos de implementación y mantenimiento. La implementación de este tipo de sistemas puede ser costosa, ya que implica la adquisición de hardware y software especializado. A su vez, exige un mantenimiento continuo y actualizaciones periódicas, que puede implicar una subida significativa en costes a largo plazo.

4. Complejidad en la configuración y personalización. Configurar y personalizar un sistema SCADA con el objetivo de satisfacer las necesidades de una industria requiere conocimientos técnicos avanzados. Aumenta el tiempo de implementación y los recursos invertidos en él.

5. Limitaciones en la escalabilidad. Los sistemas SCADA tienen dificultades para escalar y adaptarse a cambios en los requisitos o expansiones futuras.

2.2 Frameworks IoT

2.2.1. Descripción y ejemplos

Los frameworks de IoT son plataformas que proporcionan herramientas para desarrollar soluciones IoT de manera eficiente. Estos frameworks ofrecen funcionalidades de conectividad, procesamiento de datos y gestión de dispositivos.

El funcionamiento habitual de estos frameworks es el uso de APIs de comunicación desde el backend y la personalización de un frontend a través de una plataforma privada. En este caso, los dispositivos envían a través de Internet mensajes a una API ya definida por la herramienta, y los clientes se conectan a la plataforma específica y visualizan la información.

Actualmente existen varios frameworks de IoT en el mercado al alcance de todo tipo de empresas. Algunos de los más famosos son: AWS IoT, Azure IoT, Google Cloud IoT, Ubidots o IBM Watson IoT. En un principio se planteó la posibilidad de utilizar uno de estos frameworks para avanzar en el desarrollo de la solución. Sin embargo, se encontraron desventajas que dificultaban el cumplimiento de los requisitos del proyecto.

2.2.2. Desventajas de los frameworks IoT

Las deventajas que se encontraron a estos frameworks de IoT respecto a la realización de este proyecto fueron las siguientes:

- Los costes: los servicios que ofrecen no son de código abierto. Ponen tarifas altamente costosas dependiendo del volumen de datos que se recibe y del número de dispositivos que se conectan.

- Integración limitada: como ya se ha mencionado, dependiendo de la suscripción que se tuviese en dicha plataforma, permitía el uso de más o menos dispositivos.

- Personalización limitada: uno de los principales problemas que se detectaron fue la poca personalización tanto de la comunicación entre dispositivos como de las pantallas del frontend. Por parte de las comunicaciones, el formato de los datos estaba muy restringido y no admitía ciertos formatos que manejan los autómatas. Por parte del frontend, el diseño de la pantalla estaba limitado a los visores que ofrece cada plataforma, sin posibilidad de crear visores específicos.

2.3 Propuesta del trabajo

La solución que se propone en este proyecto no consiste en reemplazar la tecnología ya existente, al contrario, complementar esta tecnología para poder paliar algunos de los inconvenientes del sistema SCADA y sin tener las limitaciones de los frameworks IoT.

La propuesta es instalar un servidor el cual se comunique directamente con los PLCs, implementando un middleware. De esta forma, si la comunicación directa con el SCADA fallase, los datos podrían persistir dado que también se comunican al servidor. El servidor a su vez, procesa los datos y los envía al servicio web que los requiera. Este flujo de comunicaciones lo podemos observar en la Figura 2.3

En cuanto a los problemas de seguridad, interesa utilizar conexiones seguras. Se debería trabajar con estructuras de datos canónicas definidas en la etapa de diseño, de forma que la complejidad de configuración se ve reducida. A través de un servicio web se ofrece una vista personalizada a cada cliente de su planta y los procesos que se están produciendo con datos a tiempo real.

Por otro lado, las limitaciones de escalabilidad desaparecen. Se utilizarán comunicaciones con un modelo publicador-suscriptor, por lo que a la hora de añadir un nuevo PLC solamente habrá de comunicarse correctamente con el servidor.

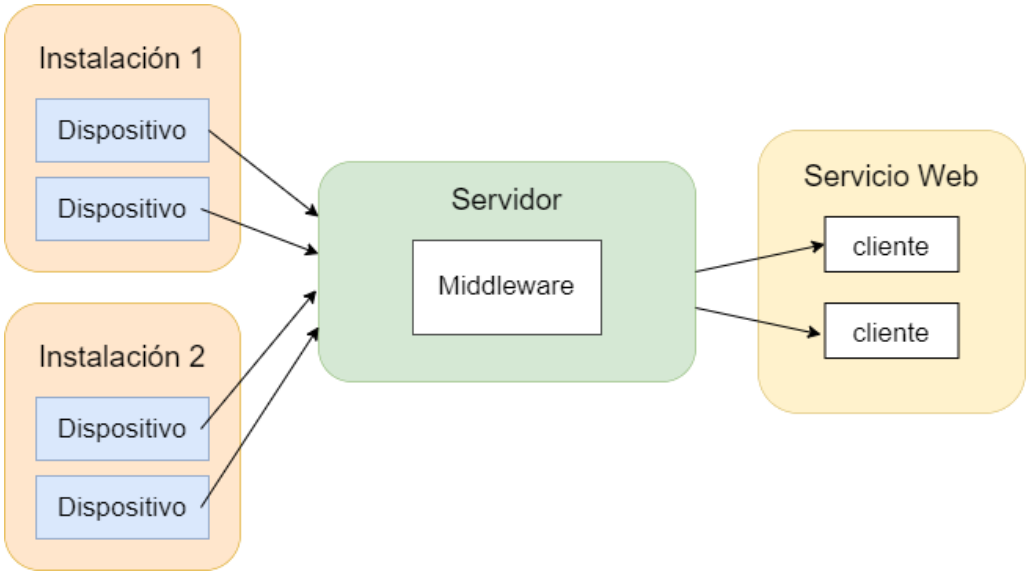

Figura 2.3: Esquema de la solución propuesta

Cabe mencionar que en el servidor persisten los datos, y en caso de que el cliente quiera ver un registro de datos anteriores, el servicio web se los pedirá al servidor y este se los proporcionará.

Por último, eliminamos la necesidad de tener que encontrarse delante de la pantalla del ordenador central para poder visualizar los datos. De esta forma, los operarios pueden realizar distintas tareas situadas geográficamente en lugares distintos y poder ver a la vez qué datos se están reportando en tiempo real.

CAPÍTULO 3
Análisis del problema

En este capítulo se extraerán las tareas y los requisitos que deberá cumplir la solución que se propone en el trabajo. En primer lugar se explicará el contexto del que nace la problemática. Posteriormente, se recogerán los requisitos específicos que se necesitarán para resolver el problema. Por último, se definirán varios casos de uso partiendo de los requisitos dados.

3.1 Contexto del problema

El problema nace en el ámbito industrial, concretamente de una planta depuradora de agua. En dicha planta, encontramos varias tareas industriales controladas, administradas y monitorizadas por autómatas, sensores y activadores.

Actualmente los dispositivos mencionados son controlados por un operario, el cual tiene a su disposición un panel SCADA, situado en una sala de la planta. Cada sensor y activador se encuentra conectado con un PLC, el cual envía y recibe datos del panel SCADA.

Los datos que llegan al panel SCADA son explotados y mostrados al operario, y este puede tomar decisiones basándose en dichos datos. Todas las conexiones que se realizan entre los dispositivos que intervienen en la red SCADA son conexiones Ethernet. Los PLCs instalados en la planta poseen la capacidad de transmitir la información por diferentes protocolos de comunicación utilizados en la industria.

Pese a que esta infraestructura que posee la planta se encuentra funcionando correctamente, el encargado de la planta tiene varias necesidades las cuales no son cubiertas con la tecnología actual. Por un lado, se comentan las limitaciones de acceso a los datos, es decir, para poder visualizar los datos el operario debe encontrarse directamente en el panel SCADA. En caso de que el encargado no se encuentre de guardia en ese momento, no podría ver los datos registrados a no ser que fuese a la planta. Por otro lado, la visualización que ofrece el panel SCADA le parece antiestética, y prefiere un nuevo diseño más colorido pero sin perder la eficiencia de los datos.

Al tratarse de una planta automatizada, los datos se envían constantemente y se deben poder ver lo antes posible, es por ellos que estamos hablando de una implementación a tiempo real. A su vez, el sistema SCADA que ya poseen retiene

los datos para poder inspeccionarlos en otro momento. Por lo tanto, se deberá diseñar una solución que no sólo ofrezca datos a tiempo real, sino que pueda acceder a datos históricos anteriores los cuales hayan sido persistidos en algún lugar.

3.2 Requisitos específicos del sistema

3.2.1. Técnicas de extracción de requisitos utilizadas

Para poder realizar la extracción de requisitos del sistema, se ha recurrido a la metodología centrada en el usuario, la cual se caracteriza por situar al usuario como eje principal de todo desarrollo del sistema. Existen una serie de técnicas y métodos para la extracción de estos, pero únicamente se han utilizado aquellos que proporcionan más información y de mejor calidad. Estas técnicas han sido:

- Observación de los usuarios: se ha observado al operario que interactúa con el sistema SCADA, en su entorno de trabajo real. Esta técnica proporciona información sobre las actividades y tareas que realiza la persona encargada del panel principal. La principal característica extraída es la monitorización a tiempo real. Por lo que esta característica debe ser una necesidad crítica en nuestro sistema. Otra de las conclusiones que se ha sacado es la importancia de la organización de los datos. La visualización debería tener una estructura coherente y de sencilla comprensión.
- Entrevistas: se ha realizado una entrevista individual al encargado de la planta, ya que él es quien ha planteado el problema y el usuario final de la solución. En esta técnica, es útil formular preguntas abiertas, ya que ayuda a una comprensión más profunda de los requisitos. A continuación se muestran las preguntas realizadas durante la entrevista y la conclusión recibida de la respuesta del usuario entrevistado:
  - A la pregunta "¿Cuáles son las principales actividades que realizas en tu trabajo relacionadas con el sistema?" se extrae la conclusión de que, el usuario interactúa con el sistema de forma continua. La principal actividad es monitorizar y tomar decisiones según los datos que reciba el sistema.
  - A la pregunta "¿Qué aspectos del sistema actual te resultan más útiles y satisfactorios? ¿Por qué?" se extrae la conclusión de que, el sistema actual responde rápido a las órdenes del operario y transmite los datos al instante. Además, persiste los datos durante un tiempo para poder revisarlos y extraer conclusiones de momentos anteriores.
  - A la pregunta "¿Qué información o datos consideras que son esenciales para tu trabajo?" se extrae la conclusión de que, todos los datos que participen en los procesos automatizados son importantes. Además, se debe indicar en qué momento se han transmitido esos datos. Esta variable se encuentra codificada actualmente por el sistema SCADA, no es transmitida por los PLCs.

- A la pregunta "¿Qué funcionalidades adicionales te gustaría ver en el nuevo sistema?" se extrae la conclusión de que, el usuario desearía poder ver una gráfica de los datos.

- A la pregunta "¿Te gustaría interactuar con el sistema haciendo uso de diferentes dispositivos? (ordenador, smartphone, tableta, ...)" se extrae la conclusión de que, para el usuario es necesaria esta función.

- A la pregunta "¿Cuáles son las mayores dificultades que encuentras al utilizar el sistema actual?" se extrae la conclusión de que, para el usuario la estética actual no es convincente, y la accesibilidad se encuentra muy limitada ya que le gustaría acceder a los datos sin necesidad de estar en la industria.

- Prototipado: El desarrollo de prototipos de baja y alta fidelidad del sistema permiten obtener retroalimentación de los usuarios antes de la implementación final. Estos, junto con las pruebas de usabilidad ayudan a identificar problemas de diseño, errores y requisitos. En este caso, se han utilizado prototipos de baja fidelidad, prototipos digitales básicos, desarrollados con la herramienta Draw.io, para el diseño de la pantalla de visualización de los datos en el frontend. Los mockups desarrollados han sido los siguientes:

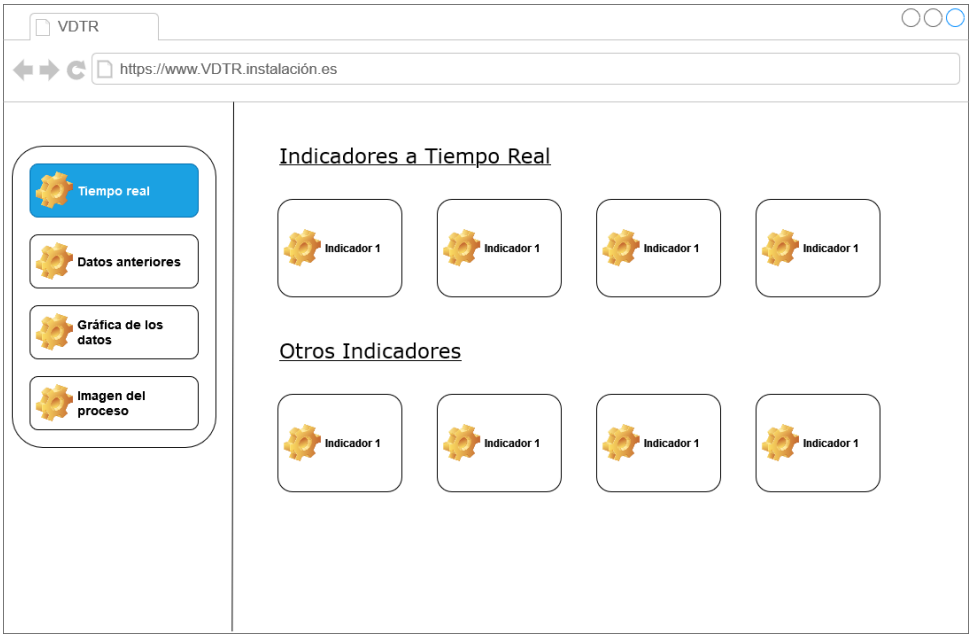

Figura 3.1: Primer prototipo: indicadores a tiempo real

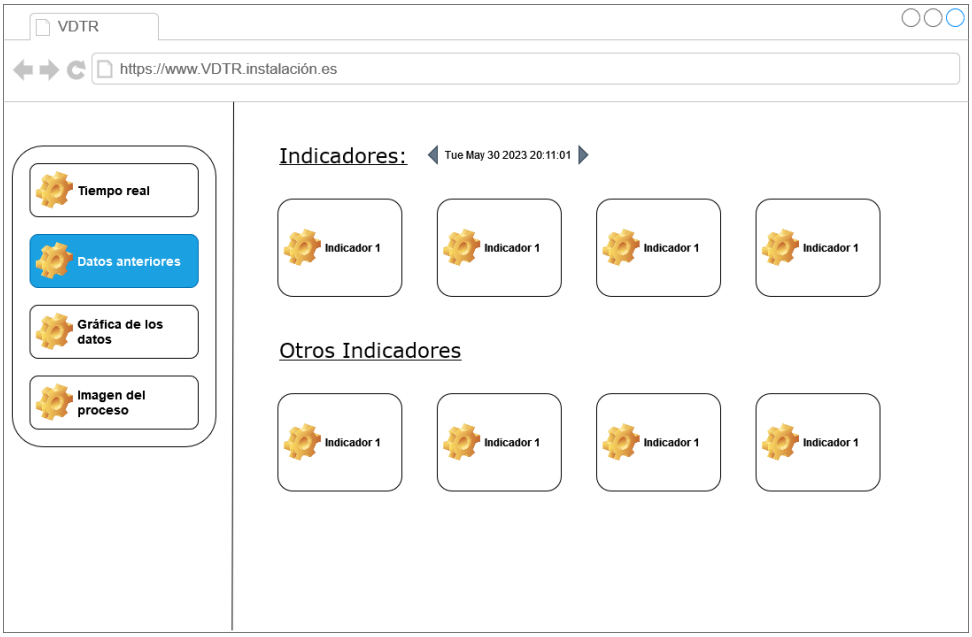

Figura 3.2: Primer prototipo: datos históricos

En el primer prototipo que se hizo, se utilizaba un menú lateral para saltar de una pantalla a otra. Cada pantalla poseía una de las utilidades que más interesaban al usuario. Las Figuras 3.1, 3.2, 3.3, 3.4 hacen referencia a este primer prototipo. La pestaña azul de cada una de las imágenes nos indica donde está situado el usuario. Al presentar el prototipo digital al cliente, este no quedó convencido. Una de las réplicas más relevantes que puso a este modelo fue la navegación entre ventanas. El cliente prefiere tener todos los datos en una sola pantalla con la finalidad de poder ver todas las funciones de un solo vistazo. Por lo tanto, se debe idear un nuevo prototipo con este requisito.

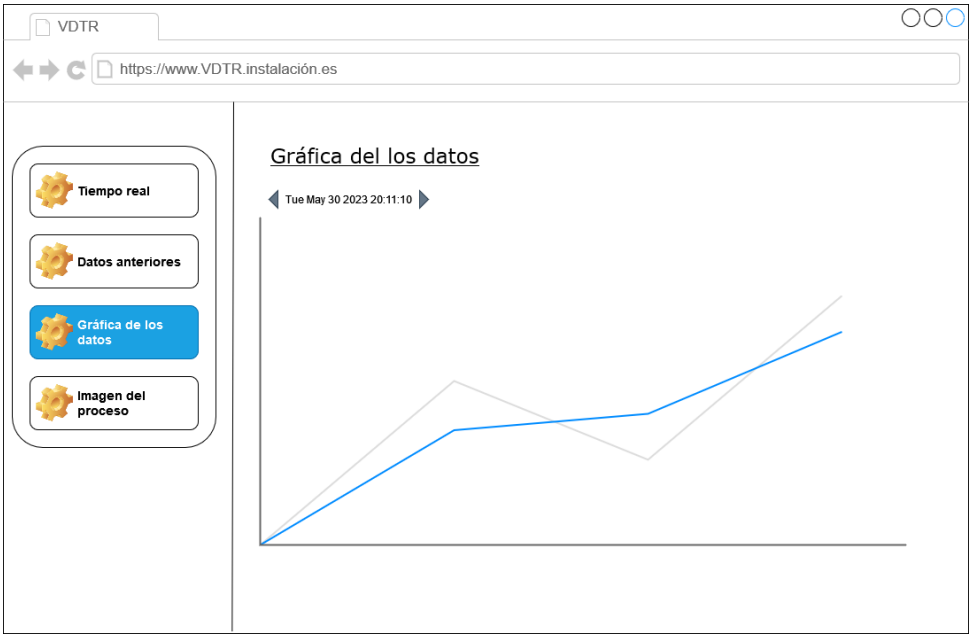

Figura 3.3: Primer prototipo: gráfica de datos

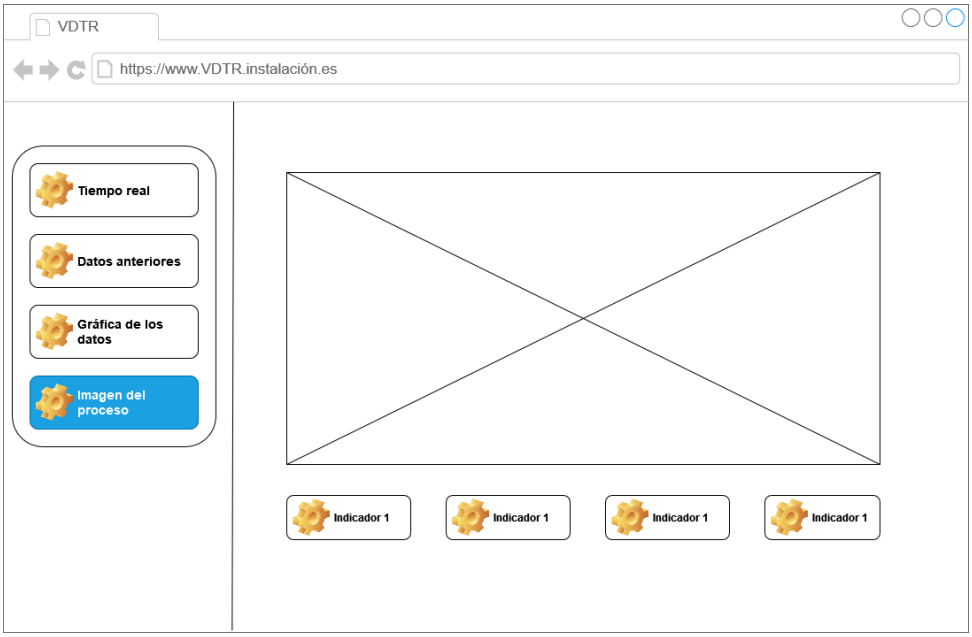

Figura 3.4: Primer prototipo: imagen del proceso

A partir de los comentarios del cliente sobre el primer prototipo se diseñó otro prototipo digital de baja fidelidad que logró convencer al cliente. Este prototipo no es definitivo, pero marca las bases del diseño. Este diseño de una sola pantalla se puede observar en la Figura 3.5. Consta de los elementos ya creados en el prototipo anterior pero sin ningún tipo de navegación. Los elementos empleados son genéricos, pues en este momento no se sabe el número exacto de indicadores, ni la imagen del proceso de la planta.

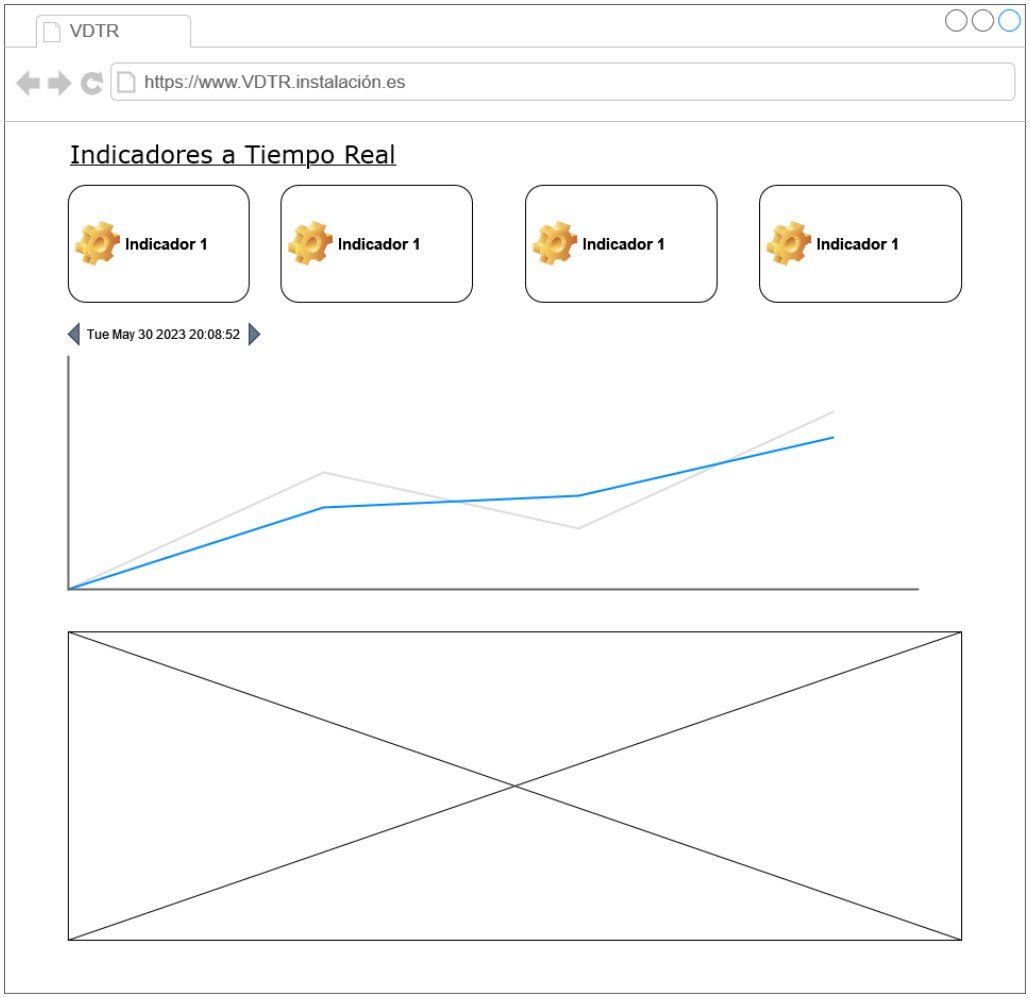

Figura 3.5: Segundo prototipo

3.2.2. Lista de requisitos extraídos

Como resultado de aplicar las técnicas comentadas anteriormente, se obtiene la siguiente lista de requisitos:

1. Monitorización a tiempo real. El sistema debe proporcionar una función de monitorización continua y en tiempo real de los datos.
2. Organización coherente de los datos. La visualización de los datos debe tener una estructura clara.
3. Transmisión instantánea de datos. El sistema debe ser capaz de transmitir los datos de forma inmediata.
4. Alta escalabilidad del sistema. El sistema debe ser escalable para la rápida incorporación de nuevos dispositivos sin afectar a la accesibilidad de este.
5. Persistencia de datos. El sistema debe almacenar los datos para su posterior explotación si se requiere.
6. Indicación de los instantes de transmisión de los datos. El sistema debe relacionar los datos con el momento en que estos han sido recibidos.

7. Monitorización de datos históricos. El sistema debe poder proporcionar datos de momentos anteriores.

8. Gráfica de datos. El sistema debe permitir la visualización de los datos en forma de gráfica para facilitar la interpretación y el análisis de tendencias.

9. Interacción con diferentes dispositivos. El sistema de ser accesible y permitir la interacción con diferentes dispositivos, como ordenadores, smartphones, etc., para brindar flexibilidad al usuario. Por lo que el se deberá desarrollar un diseño "responsive" de la pantalla de visualización.

10. Accesibilidad remota de los datos. El sistema debe permitir el acceso a los datos de forma remota.

11. Visualización en una sola pantalla. Requisito extraído de la fase de prototipado de las pantallas del frontend. La visualización debe ser sencilla y rápida.

3.2.3. Plan de trabajo

En este apartado se van a extraer las tareas de alto nivel a realizar en el presente proyecto, creadas a partir de los requisitos anteriores. A su vez se realiza una estimación temporal de cada tarea con la finalidad de crear una planificación a seguir durante el desarrollo de este.

Tabla 3.1: Plan de trabajo

Tarea a desarrollar	**Tiempo estimado (horas)**
Comunicación con los dispositivos ya instalados	14
Explotación de los datos recibidos del backend	10
Almacenamiento de los datos recibidos	10
Comunicación instantánea con el frontend	18
Comunicación bajo demanda con el frontend	14
Diseño y desarrollo del frontend	28
Recepción y explotación de los datos en el frontend	10
Implantación de la solución en un servidor	14
Pruebas	8
Total horas estimadas	126

La estimación realizada se ha hecho en base a los conocimientos que se tienen en el momento en el que se empieza a desarrollar el proyecto. Este tiempo puede aumentar o disminuir en función de las dificultades que se vayan encontrando durante el propio desarrollo.

3.3 Casos de uso

Continuando con la fase de análisis, se van a desarrollar distintos casos de uso partiendo de los requisitos del Punto 3.2.2. Cada caso de uso representa una interacción específica entre un usuario, o sistema externo, y el sistema en desarrollo.

Usualmente, los diagramas de casos de uso se conforman por los siguientes componentes para llevar a cabo la modelización:

- Los actores: representan todos los elementos que pueden interactuar con el sistema. Estos elementos pueden ser usuarios u otros sistemas externos.
- Los escenarios de uso: consisten en las funcionalidades completas que ofrece el sistema.
- Las relaciones: se utilizan para asociar los actores con los escenarios de uso o para especificar relaciones entre actores.
- El sistema software: se extraen los requisitos.

El primer caso de uso, reflejado en la Figura 3.6, cumple con la funcionalidad de prestar un servicio de monitorización a tiempo real al usuario. Participan dos actores, el usuario cliente y el sistema externo PLC.

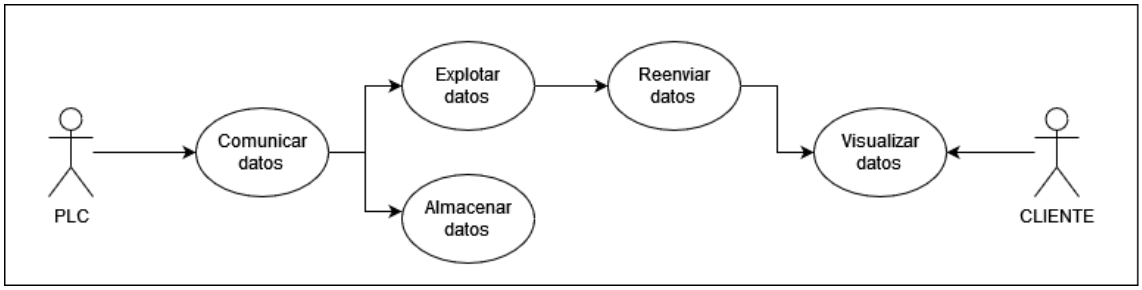

Figura 3.6: Primer caso de uso: comunicación a tiempo real

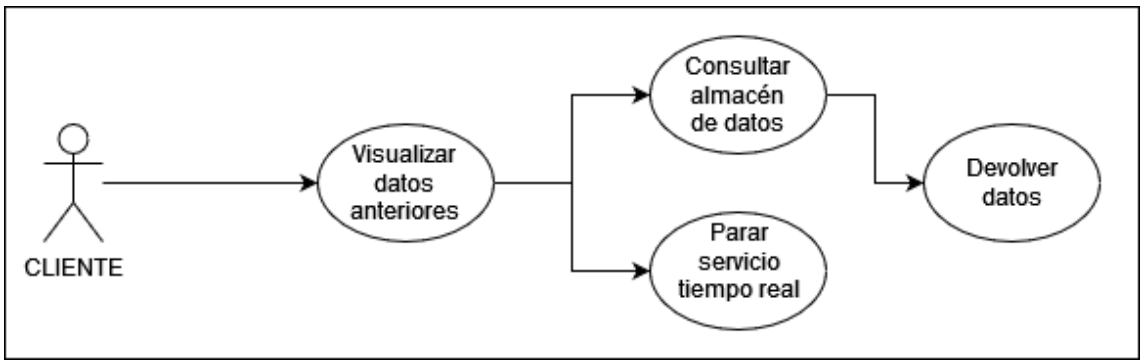

Figura 3.7: Segundo caso de uso: comunicación de datos almacenados

En el segundo caso de uso, Figura 3.7, se muestra la funcionalidad de proporcionar datos históricos al cliente, siempre y cuando este los solicite.

Estos casos de uso desarrollados han sido extraídos a partir de características y comportamientos que se han acotado previamente, y servirán para guiar el diseño y el desarrollo del proyecto. Posteriormente, también serán útiles para la definición de pruebas del sistema.

CAPÍTULO 4
Diseño

En este capítulo se define el diseño del proyecto, etapa previa al desarrollo, en la que se concreta qué componentes va a necesitar la solución propuesta y cómo se van a relacionar entre ellos.

En esta etapa se realizará el acuerdo de las estructuras de datos, la arquitectura software, las representaciones de interfaz y definición de las comunicaciones. El objetivo del diseño es producir un modelo o representación de la entidad que se construirá posteriormente.

En primer lugar se especificarán los componentes que participarán en el sistema final. Partiendo de esta tarea, se podrá construir una arquitectura del sistema que englobe los componentes definidos anteriormente. La comunicación entre todos estos componentes es fundamental para el correcto funcionamiento del sistema, y será lo siguiente que se diseñe. Por último, partiendo del prototipo aceptado en la fase de análisis, se podrá plantear una interfaz gráfica sobre la que trabajar cuando se empiece su desarrollo.

4.1 Arquitectura y componentes

Para poder definir los componentes del sistema, se deben tener en cuenta los requisitos de la fase de análisis y el plan de trabajo especificado. Cada componente del diseño debe cumplir al menos una de las funciones especificadas del sistema. Posteriormente se planteará una arquitectura para el sistema.

4.1.1. Componentes del sistema

En primer lugar, el sistema necesitará un componente que pueda comunicarse con los dispositivos ya instalados en la industria. Como ya se mencionó anteriormente estos dispositivos son PLCs, autómatas que recogen datos y los envían a través de la red. Estos autómatas tienen la capacidad de emplear varios protocolos de envío, aspecto interesante a la hora de definir las comunicaciones con el sistema. Este primer componente, Figura 4.1, deberá ajustarse al protocolo que se defina para la comunicación, recibir los datos, y transmitirlos al siguiente componente.

No hay un número exacto de dispositivos que intervengan en la comunicación, de hecho, debe plantearse un diseño que permita la rápida incorporación de un nuevo dispositivo, es decir, un diseño escalable. Para poder aportar esta escalabilidad al diseño, se ha decidido que las comunicaciones sean comunicaciones basadas en colas con un patrón publicador-suscriptor. De esta forma, los dispositivos serían publicadores en una cola y el componente sería suscriptor de dicha cola. En el ámbito industrial, existen varios protocolos de comunicación basados en colas.

En la siguiente sección de este mismo capítulo se hablará sobre los protocolos de comunicación que se han escogido para la implementación de las comunicaciones del sistema.

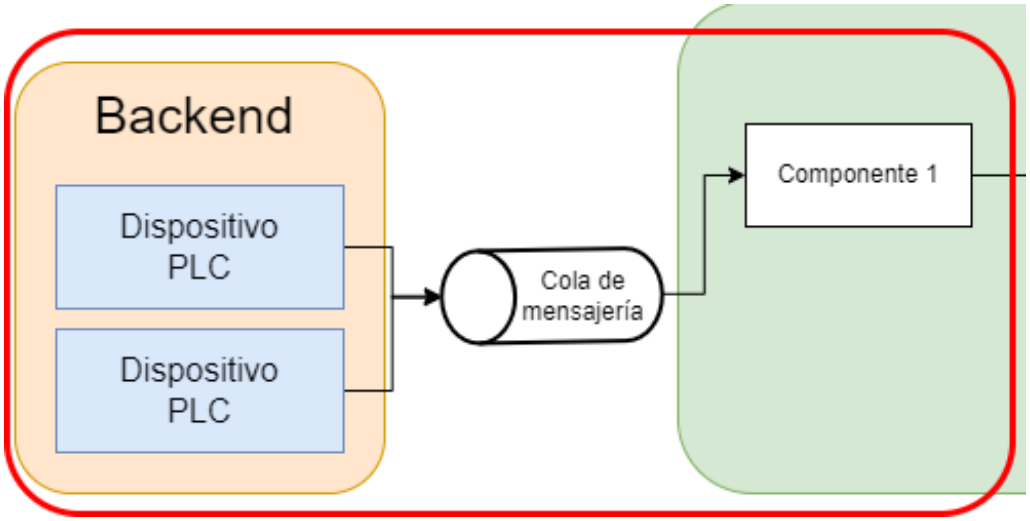

Figura 4.1: Primer componente del sistema

El sistema debe cumplir otra función importante, que consiste en la explotación, orquestación y modificación de los datos. Para llevar a cabo esta tarea, se desarrollará un componente adicional.

Este segundo componente recibirá los datos provenientes del componente anterior y los reconstruirá en un formato canónico que se definirá más adelante. Además, se agregará un nuevo campo a los datos recibidos, que indicará el momento en el que se recibió la información. Este nuevo dato resulta útil para la interacción en tiempo real y el filtrado por fecha de los datos históricos.

Una vez que se hayan completado todas las operaciones necesarias, el componente deberá reenviar los datos a dos componentes adicionales. Uno de ellos se encargará de enviar la información al frontend, mientras que el otro se encargará de almacenar los datos. Es importante destacar que estas comunicaciones deben realizarse de manera ágil, sin tiempos de espera.

En la Figura 4.2 se ilustra este segundo componente y sus interacciones con los demás componentes del sistema. Las flechas representan la dirección del flujo de información.

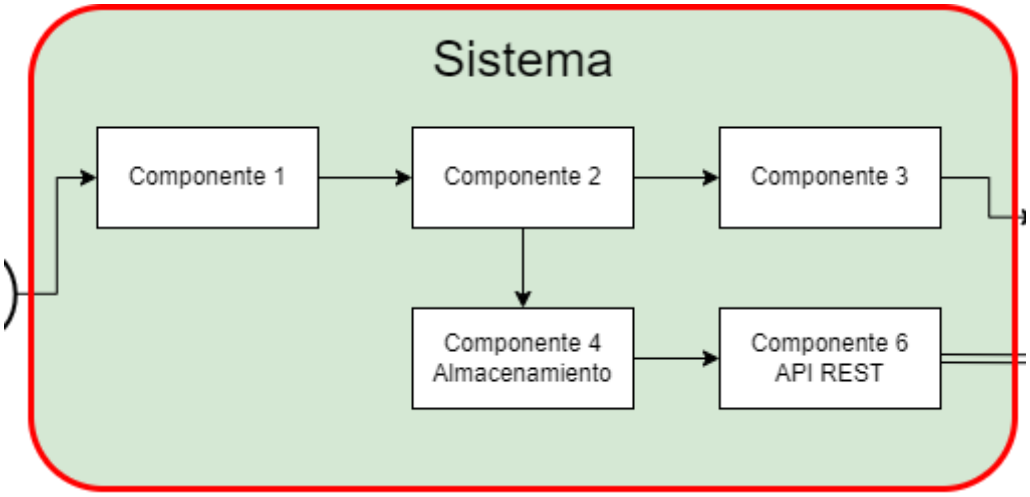

Figura 4.2: Segundo componente del sistema

La función siguiente que se abordará se refiere a la comunicación con el frontend. Con este fin, se ha desarrollado un tercer componente que recibe los datos provenientes del componente anterior y los retransmite en tiempo real. Es fundamental que este componente sea diseñado de manera escalable, ya que debe ser capaz de atender a un número indeterminado de clientes sin comprometer la eficiencia del sistema.

Para lograr esta característica, se ha optado nuevamente por emplear un patrón de diseño publicador-suscriptor. En este enfoque, el componente actúa como publicador, enviando los datos a un tema específico, mientras que los clientes se conectan como suscriptores a dichos temas. En futuras especificaciones se detallarán los formatos utilizados en la comunicación entre ellos.

En la Figura 4.3 se presenta este tercer componente, así como sus interacciones con los demás componentes del sistema.

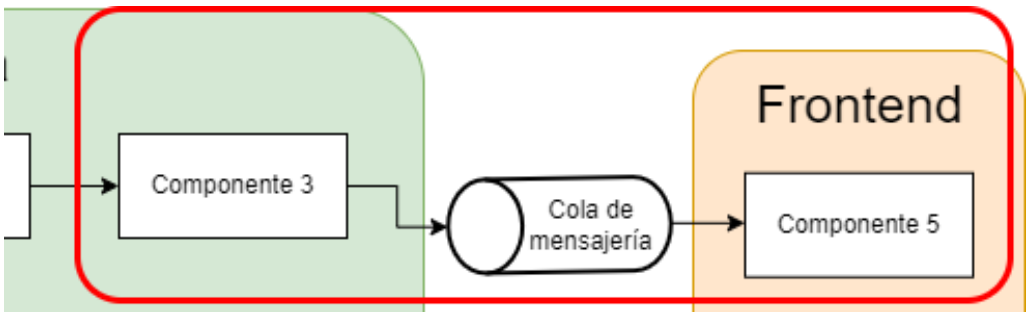

Figura 4.3: Tercer componente del sistema

Como ya se ha avanzado anteriormente, otra de las funciones que necesita el sistema es un elemento de almacenamiento de los datos. En este caso se ha optado por utilizar una base de datos.

Para diseñar la base de datos se ha elaborado un diagrama, Figura 4.5, sobre las tablas que se van a utilizar, los atributos necesarios y la relación entre ellas:

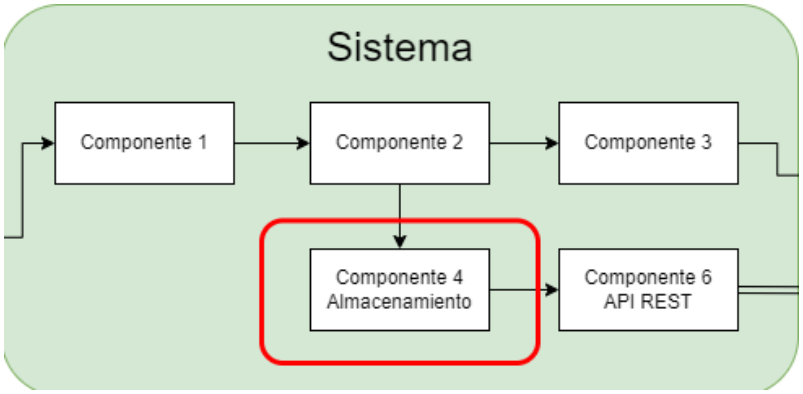

Figura 4.4: Componente de almacenamiento de datos en el sistema

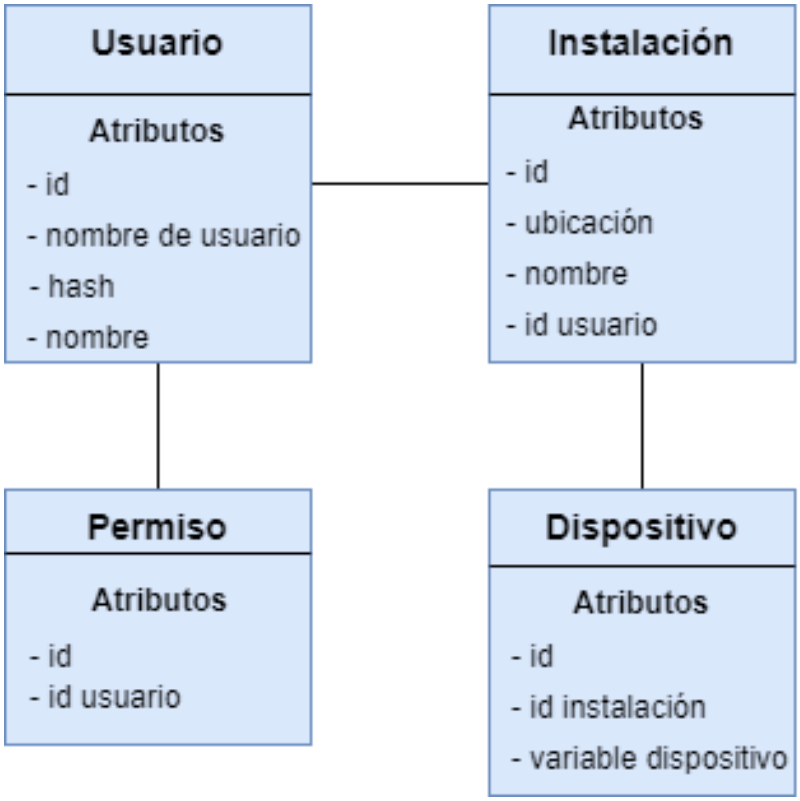

Figura 4.5: Diagrama de la base de datos

En la Figura 4.5 se aprecia una tabla llamada dispositivo_X. Estas tablas son creadas dinámicamente por el sistema al recibir datos de un nuevo dispositivo. La ”X” del nombre de la tabla hace referencia al identificador del dispositivo.

Para proporcionar el servicio de visualización, se ha desarrollado un pequeño proyecto frontend. El diseño de la visualización se basa en el prototipo presentado en el capítulo de análisis. Este componente puede ser desplegado posteriormente en un servidor para que sea accesible a través de Internet.

Pero antes de desarrollar código, se han diseñado las interfaces de usuario finales haciendo uso de la misma herramienta que se utilizó para el prototipo en la fase de análisis. El usuario accederá a la plataforma a través de Internet y se encontrará con la pantalla de la Figura 4.6. De está pantalla accederá al formulario de identificación de usuario, Figura 4.7. Una vez se verifique que el usuario

exista, la aplicación dirigirá al usuario a la pantalla de visualización de la industria Figura 4.8, la cual será una página dinámica que irá actualizando los datos a medida que esta los reciba.

Figura 4.6: Pantalla de inicio

Figura 4.7: Pantalla de login

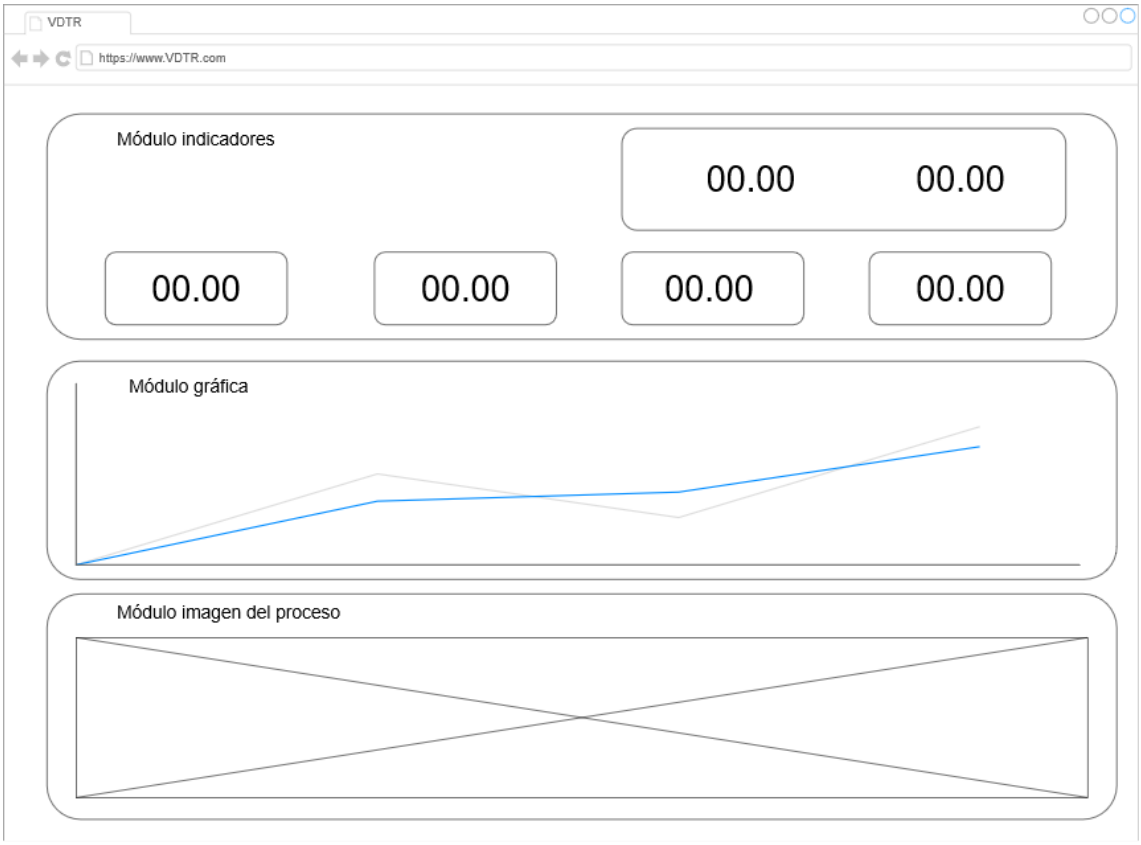

Figura 4.8: Pantalla de visualización

Desde el cliente frontend existe la posibilidad de solicitar datos de momentos anteriores. Para aportar esta funcionalidad al sistema se ha diseñado un nuevo componente que corresponde a una API REST, Figura 4.9.

Esta API tiene acceso a la base de datos que almacena los datos que se reciben en el sistema, por lo que sirve de puente entre base de datos y frontend. La API además realiza ciertas comprobaciones de seguridad con el fin de evitar ataques relacionados con SQL Injection.

Por otra parte, este componente no es utilizado únicamente para recuperar datos históricos, también se recuperan cuentas de usuario comprobando la autenticación que se realiza en la Figura 4.7. Además, otorga tokens de sesión temporales a los usuarios, de esta forma el usuario puede cerrar sesión si lo desea, o cerrar el navegador evitando que la sesión se quede abierta por alguna cookie almacenada.

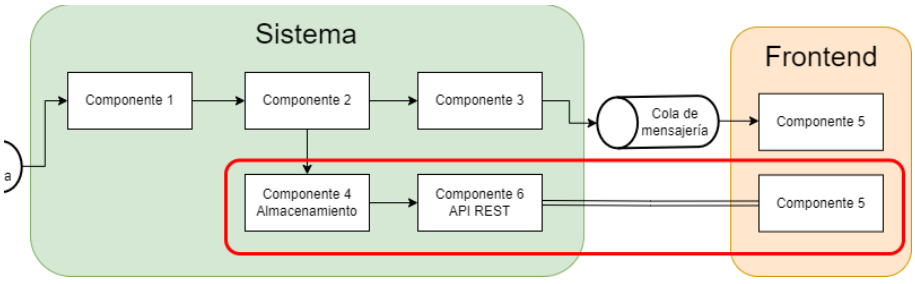

Figura 4.9: Componente API Rest

Por el momento ya se han diseñado todos los componentes necesarios para cubrir las necesidades del sistema. La interacción entre todos estos componentes la podemos ver reflejada en la Figura 4.10. En esta Figura se pueden diferenciar tres regiones. La región izquierda, compuesta por los dispositivos industriales, ajena al alcance de este proyecto. La región central, correspondiente a los componentes del sistema relacionados con el backend y las comunicaciones. Y por último, la región de la derecha, la cual también forma parte de la solución, que corresponde al proyecto del frontend.

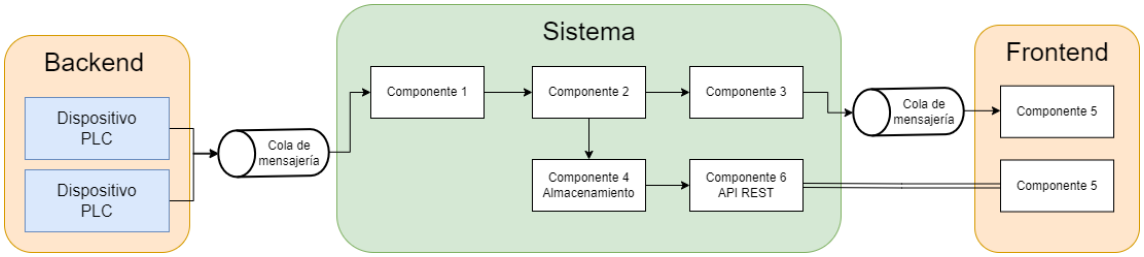

Figura 4.10: Diagrama completo de los componentes del sistema

4.1.2. Arquitectura del sistema

Una vez definidos todos los componentes del sistema, se procede a especificar su arquitectura. El sistema exhibe una arquitectura híbrida, en la cual predomina la arquitectura orientada a eventos, pero también se pueden identificar elementos de otras arquitecturas, como la orientada a servicios (SOA) y la cliente-servidor.

Como se muestra en la Figura 4.10, las comunicaciones entre los dispositivos del backend y las pantallas del frontend siguen un patrón publicador-suscriptor, como se mencionó anteriormente. Este patrón corresponde a la arquitectura orientada a eventos, que se encuentra presente en la mayoría del sistema.

Sin embargo, el servicio proporcionado por el frontend se realiza bajo demanda de los clientes. Estos clientes solicitan un recurso al sistema, y el sistema recibe la solicitud y proporciona dicho recurso. En este caso, se está hablando de una arquitectura cliente-servidor para el frontend.

Es importante destacar que uno de los componentes del sistema es una API que brinda servicios al frontend. Esta API ofrece funcionalidades como inicio de sesión, consulta de datos, entre otros. Aunque no se considera una arquitectura orientada a servicios en su totalidad, sí se beneficia de algunos principios SOA, como la modularidad, la reutilización y la capacidad de escalar asociadas a esta arquitectura. El hecho de que no se considere arquitectura SOA es debido a que en el sistema no existen múltiples servicios independientes, si no, un único servicio. En caso de que en un futuro se decida integrar varias bases de datos, con varias APIs de acceso a ellas, ya se asemejaría más a esta arquitectura.

4.2 Comunicaciones entre componentes

En esta sección se especifican las comunicaciones entre los componentes del sistema mencionados anteriormente, detallando los protocolos y los formatos que se usarán.

4.2.1. Comunicaciones con el backend

Respecto al primer componente, a la hora de comunicarse con el backend ya se mencionó que se realiza mediante colas de mensajería. Los autómatas ya disponen de protocolos basados en colas de mensajería. Algunos de los protocolos más utilizados con un patrón publicador-suscriptor son:

- MQTT (Message Queuing Telemetry Transport): es un protocolo de mensajería ligero y de bajo consumo diseñado para redes de sensores y dispositivos con ancho de banda limitado.
- AMQP (Advanced Message Queuing Protocol): es un protocolo de mensajería estándar y orientado a empresas. Ofrece varios patrones de comunicación, uno de ellos siendo publicador-suscriptor.
- JMS (Java Message Service) : es una API de Java que proporciona un estándar para la mensajería asíncrona. Admite varios el patrón publicador-suscriptor entre otros.

De entre todos los protocolos vistos, se ha decidido utilizar el protocolo MQTT. Se ha elegido este protocolo por la eficiencia y bajo consumo de recursos de este, su tolerancia a conexiones intermitentes, su alta escalabilidad y su facilidad de uso. Además, según la Referencia [3], este protocolo es ampliamente utilizado en la conectividad de dispositivos industriales de IoT, demostrando su relevancia en ese ámbito específico.

Además, las conexiones que se realizan entre el componente y los dispositivos se deben cifrar, asegurando de esta forma la confidencialidad de la comunicación, la integridad de los datos y la autenticidad del dispositivo. La opción escogida para asegurar estas conexiones es la de uso de certificados SSL/TLS, ya que los PLCs soportan las comunciaciones con esta opción.

4.2.2. Comunicaciones con el frontend

Para las conexiones entre el sistema y el frontend desarrollado se han diseñado comunicaciones a través de websockets, utilizando el protocolo WAMP. El Protocolo de Aplicación de Mensajes Web (WAMP) es un protocolo de comunicación basado en el modelo cliente-servidor que permite la comunicación en tiempo real. Este protocolo proporciona varios patrones de mensajería, entre ellos el patrón publicador-suscriptor que se utiliza en este proyecto. Además, WAMP proporciona mecanismos para autenticar y autorizar clientes y servidores, asegurando las conexiones a los recursos.

Sin embargo, este protocolo será utilizado exclusivamente para la interacción en tiempo real. La comunicación con la API se realiza mediante llamadas HTTP tradicionales, como por ejemplo GET, POST, PUT, DELETE, entre otras, utilizando el protocolo HTTP. Es importante que el frontend procese estas llamadas a la API de manera eficiente, por lo que sería conveniente utilizarlas de forma asíncrona en la parte del cliente.

4.2.3. Formato de datos canónico

En los proyectos donde intervienen varios componentes es interesante manejar un tipo de formato común o canónico. Definir bien estos formatos otorga ventajas como la interoperabilidad, la flexibilidad, la reutilización y la escalabilidad. Además facilita la construcción y el mantenimiento de sistemas integrados más robustos y eficiente.

En este proyecto se ha optado por utilizar el formato de datos JSON (JavaScript Object Notation). Un formato ligero y ampliamente utilizado en el intercambio de datos. Se basa en una estructura clave-valor y nos ofrece distintas ventajas como por ejemplo la flexibilidad del lenguaje, la compatibilidad con la web y la interoperabilidad entre sistemas. Por ello es por lo que se ha elegido este formato.

Tanto para las comunicaciones con el frontend, como con el backend, debe haber una forma de identificar el dispositivo y la industria.

En la Figura 4.11 se observa el formato de datos que se espera recibir a través de las colas de mensajería con el backend. Se trata de un objeto JSON con claves de identificación del dispositivo y de la instalación, y un campo correspondiente a un objeto JSON anidado con todos los datos que el PLC envíe.

```
{
    "id_instalacion": "intalación1",
    "id_dispositivo": "1234",
    "datos":{
        ...
    }
}
```

Figura 4.11: Formato de datos canónicos que recibe el sistema

Este formato de datos es muy parecido al que envía posteriormente el sistema al frontend, pero añadiéndole un campo extra. Este campo extra es el momento en el que se reciben los datos en el sistema o "timestamp", Figura 4.12. Este campo se utiliza en la base de datos para ordenar los datos, y en el frontend para ordenar la información. En concreto, corresponde a una marca de tiempo en milisegundos.

El campo ”datos” de los formatos presentados es determinado por el PLC. En este caso, en la depuradora de donde se extraen los datos, algunos de los campos que se presentan son: caudalímetro, nivel de PH, oxígeno del agua, porcentaje de la soplante, etc.

```
{
    "id_instalacion": "intalación1",
    "id_dispositivo": "1234",
    "datos":{
        ...
    },
    "timestamp": 1657049101000
}
```

Figura 4.12: Formato de datos canónicos que envía el sistema

CAPÍTULO 5
Desarrollo

En este capítulo se muestra el proceso de desarrollo de todos las partes del proyecto. Se explica detalladamente cada componente descrito en el capítulo anterior, qué problemas han habido en su desarrollo y cuáles han sido las soluciones para resolverlos.

5.1 Entorno de desarrollo

Dado que uno de los objetivos del proyecto es abaratar costes en la solución, todas las herramientas de desarrollo, así como el entorno, son de código abierto.

El proyecto se desarrolla en una máquina virtual con sistema operativo Ubuntu Desktop. El editor de texto utilizado ha sido Visual Studio Code con los plugins necesarios para un desarrollo más cómodo.

Para el control de versiones del proyecto software se utiliza Git, creando un repositorio en local. Para el almacenamiento de este repositorio se utiliza la plataforma GitHub. Por último, con ayuda de GitHub Desktop se realizan las actualizaciones del repositorio.

5.2 Desarrollo de los componentes

5.2.1. Broker MQTT - Mosquitto

A la hora de implementar las comunicaciones con el backend mediante colas de mensajería utilizando el protocolo MQTT, se ha instalado un broker el cual pueda manejar dichas colas y no suponga una gran carga para el sistema.

El broker de mensajería elegido ha sido Mosquitto, que como observamos en la Referencia [11], se trata de una implementación de código abierto y ligero que utiliza el protocolo MQTT. Es una tecnología sencilla, eficiente y con capacidad de escalar. Estas son las razones por las que hemos utilizado este broker de mensajería en el proyecto. A su vez, ofrece características de seguridad, como autenticación y encriptación.

La instalación de este broker en la máquina virtual es muy sencilla. Para realizar la instalación y posterior configuración, se ha consultado tanto la Referencia anterior [11], como la Referencia [6]. En este caso, se ha configurado el componente para poder securizar las conexiones mediante certificados autofirmados. Solamente se procesarán mensajes de dispositivos que tengan un certificado firmado por el sistema.

Los comandos de instalación del broker en la máquina virtual Ubuntu han sido los siguientes:

```
# Montaje del servicio Mosquitto
$ sudo apt-get update
$ sudo apt-add-repository ppa:mosquitto-dev/mosquitto-ppa
$ sudo systemctl enable mosquitto.service

# Reinicio del servicio Mosquitto
$ sudo systemctl restart mosquitto

# Estado del servicio Mosquitto
$ sudo systemctl status mosquitto
```

La configuración del broker Mosquitto se encuentra por defecto en el archivo */etc/mosquitto/mosquitto.conf*. Modificando este archivo se activa la autenticación de acceso al broker, el uso de certificados, el puerto de escucha del broker, etc.

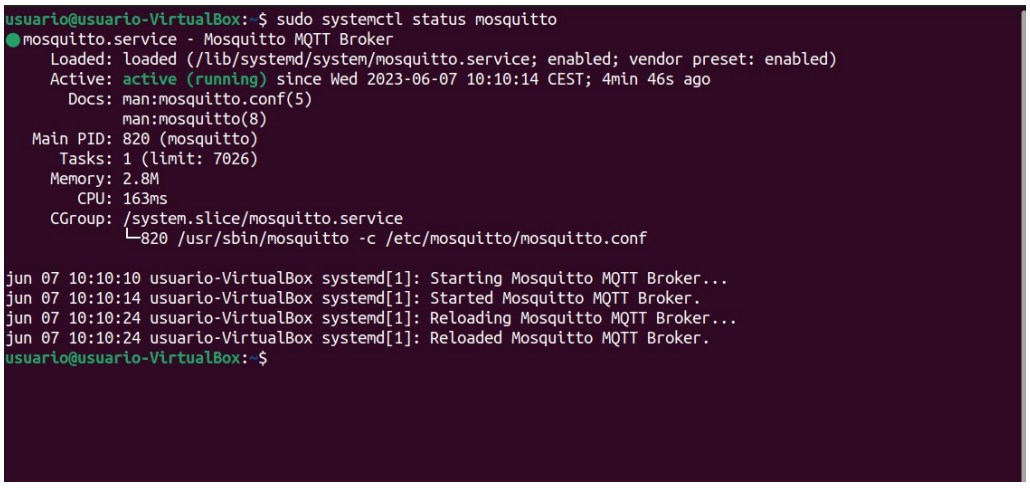

Figura 5.1: Estado del broker Mosquitto

En este apartado no se ha identificado ningún problema adicional más allá de la curva de aprendizaje de la tecnología y la investigación para asegurar las conexiones entre el broker y los dispositivos externos. Según el Plan de trabajo 3.1, esta tarea corresponde a la *Comunicación con los dispositivos ya instalados* y se estimó inicialmente en 14 horas. Sin embargo, debido a la necesidad de implementar conexiones seguras, el tiempo total requerido para esta tarea se ha incrementado a 18 horas.

5.2.2. Módulo Principal - Python

El módulo que se presenta a continuación engloba funcionalidades de los componentes uno, dos y tres definidos en el capítulo anterior, Figura 4.10. Se comenzó a desarrollar por separado pero finalmente se optó por desarrollar un solo proceso que hiciese las tres funciones. Se ha elegido Python como lenguaje de programación para este componente debido a varias ventajas que ofrece, como una sintaxis clara y legible, así como una amplia disponibilidad de bibliotecas y frameworks. Para el aprendizaje de este lenguaje se ha utilizado documentación oficial de Python y la Referencia [9].

A la hora de cumplir con la funcionalidad de conectarse a las colas MQTT como suscriptor, se ha desarrollado un proceso el cual utiliza la librería *paho-mqtt* para esta función. Este proceso recoge todos los mensajes dirigidos a la cola con el topic *VDTR/#* , es decir, a todos los mensajes que se envíen a partir en dirección al sistema. Un dispositivo publicará en el topic *VDTR/id432/1234* y este componente puede extrae el id de la instalación y el id del dispositivo del propio topic.

La siguiente funcionalidad que cumple el módulo es el procesamiento de los datos, añadiendo al formato el campo "timestamp", campo ya definido en el formato de la Figura 4.12.

En tercer lugar, el módulo comunica los datos que le llegan a la base de datos para que esta los almacene. Se ha desarrollado una función que crea la tabla dinámicamente de un nuevo dispositivo en caso de que el id de este no exista previamente. De esta forma aumentamos la escalabilidad y la eficiencia del sistema, evitando crear tablas a mano cada vez que se añada un nuevo dispositivo.

Por último, el módulo se conecta como publicador al broker Crossbar.io, broker de comunicación con el frontend, y reenvía la información modificada a través del broker con un topic que identifica el id de la instalación y el id del dispositivo, forma similar a como se hace en el backend. Esta función hace posible al comunicación en tiempo real. Si en un momento dado no existen suscriptores a ese topic el mensaje se pierde, no existe persistencias en estas colas.

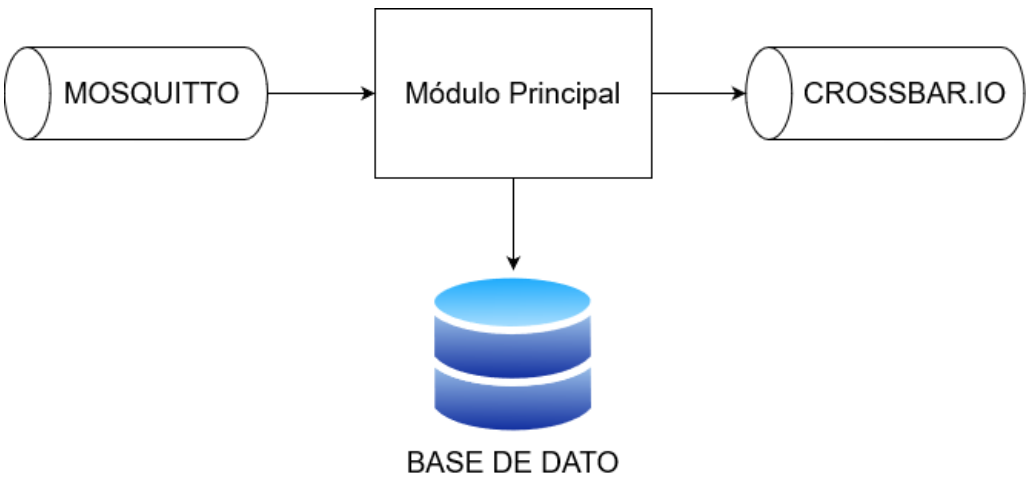

Figura 5.2: Diagrama del módulo principal

El problema a destacar en este desarrollo ha sido la ejecución del programa. Ya que en un principio el mismo hilo de ejecución se conectaba tanto a un broker como al otro. La solución fue utilizar hilos de ejecución concurrentes y asíncronos para cada broker, de forma que el mismo programa pudiese conectarse a las dos colas. Para realizar la comunicación de datos entre estos dos hilos independientes, se utiliza una pila interna con persistencia. En el momento en el que el hilo conectado al broker MQTT deposita en la pila un mensaje listo para reenviar, el otro hilo obtiene el mensaje y lo reenvía por el broker WAMP. En caso de no haber mensaje en la pila, el segundo hilo se mantiene a la espera.

Este desarrollo pertenece, en cierta parte, a las tareas *Comunicación con los dispositivos ya instalados, Explotación de los datos recibidos del backend, Almacenamiento de los datos recibidos y Comunicación instantánea con el frontend* del plan de trabajo reflejado en la Tabla 3.1. Como vemos, engloba parcialmente varias tareas en un mismo desarrollo, es por ello que no podemos comparar el tiempo real de realización de la tarea con un tiempo estimado anteriormente. El tiempo total de esta tarea ha sido de 24 horas.

5.2.3. Broker WAMP - Crossbar.io

Para realizar la comunicación con el frontend mediante el protocolo WAMP se ha instalado un broker de código abierto llamado Crossbar.io. Este es usado en aplicaciones web en tiempo real ya tiene algunas características clave: enrutamiento automático, control de acceso, escalabilidad e integración con varios lenguajes.

Para la implementación de este broker en el sistema se ha ejecutado una imagen docker. Para ello debemos instalar docker y ejecutar el siguiente comando para arrancar el broker:

```
1 # Arrancar broker Crossbar.io
2 $ sudo docker run -it -p 8080:8080 crossbario/crossbar
```

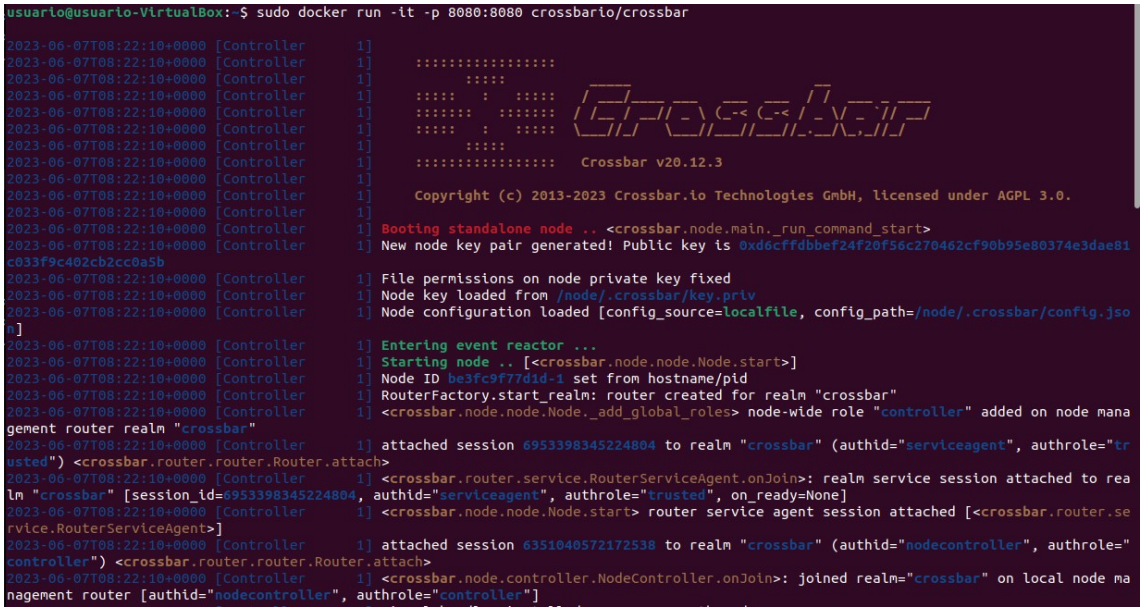

Figura 5.3: Inicialización de contenedor Crossbar.io

En este apartado se presentó un problema que tuvo una solución sencilla. Inicialmente se consideró instalar el broker Crossbar.io a partir de su código fuente en el sistema. No obstante, esta opción requería una instalación manual de las configuraciones y dependencias necesarias para su correcto funcionamiento. Además, se busca aumentar la portabilidad, ya que aún no se ha decidido en qué tipo de sistema se ejecutará la solución en el futuro. Por esta razón, se exploraron otras alternativas y se decidió utilizar un contenedor Docker. El uso de contenedores no solo brinda portabilidad, sino que también proporciona aislamiento, seguridad y escalabilidad.

Esta implementación se relaciona con la tarea *Comunicación instantánea con el frontend* del Plan de trabajo 3.1 con una estimación de 18 horas. Finalmente, debido a la curva de aprendizaje del broker y al problema mencionado anteriormente, el tiempo de desarrollo ha sido de 22 horas.

5.2.4. API Rest - Python

La implementación del componente escogido para comunicar los datos históricos al frontend es una API REST. Una API (Application Programming Interface) es un conjunto de reglas y protocolos que permiten la interacción entre distintos componentes. En este caso, una API Rest es un conjunto de reglas y convenciones que define cómo los servicios web se pueden comunicar y realizar operaciones sobre recursos a través de la web utilizando HTTP y JSON. Es una solución escalable para exponer funcionalidades y datos a través de interfaces web.

En este proyecto, se ha desarrollado una pequeña API que cubre las necesidades del frontend. Ha sido implementada con Python, utilizando la librería ”fastapi” y se despliega en el propio sistema a través de un pequeño servidor ASGI (Asynchronous Server Gateway Interface) especializado en aplicaciones de Python llamado ”Uvicorn”. Es importante utilizar conexiones asíncronas para evitar los tiempos de espera de respuesta a los diferentes clientes que accedan

simultáneamente a la API. En la siguiente Figura 5.4 se observa el comando de puesta en marcha de la API, especificando el puerto en el que se desea que esta escuche las peticiones.

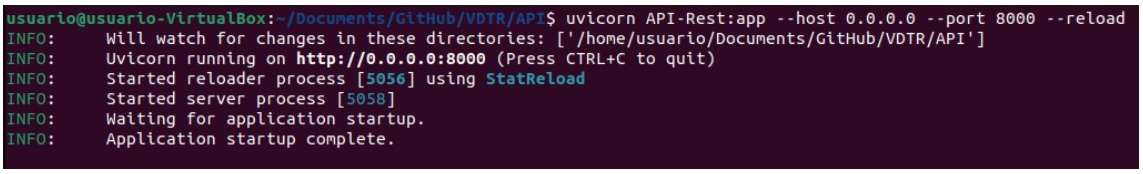

Figura 5.4: Puesta en marcha de la API

La API ha sido instalada en el propio sistema, pero no se descarta poder desacoplarla de este en un futuro. Los endpoints que presenta esta API son los siguientes:

Endpoint	/usuarios
Método HTTP	GET
Headers	token
Uriparams	-
Queryparams	-
Descripción	Este endpoint devuelve información acerca del usuario relacionado con el token proporcionado en la llamada.

Tabla 5.1: API Rest - Endpoint usuarios

Endpoint	/numeroDashboards
Método HTTP	GET
Headers	token
Uriparams	nombre_usuario
Queryparams	-
Descripción	Este endpoint devuelve el número de pantallas de visualización que posee el usuario indicado.

Tabla 5.2: API Rest - Endpoint número dashboards

Endpoint	/dashboards
Método HTTP	GET
Headers	token
Uriparams	nombre_usuario
Queryparams	-
Descripción	Este endpoint devuelve nombre de las pantallas de visualización que posee el usuario.

Tabla 5.3: API Rest - Endpoint dashboards

Endpoint	/dispositivo
Método HTTP	GET
Headers	token
Uriparams	id_dispositivo
Queryparams	-
Descripción	Este endpoint devuelve datos propios sobre el dispositivo que se indica, así como los datos recogidos por este.

Tabla 5.4: API Rest - Endpoint dispositivo

Endpoint	/token
Método HTTP	GET
Headers	token
Uriparams	-
Queryparams	nombre_usuario, contrasenya_usuario
Descripción	Este endpoint devuelve un token único de inicio de sesión. Se comprueba si los datos de inicio de sesión son correcto y en caso de serlo se le otorga un token al frontend para que pueda realizar futuras consultas a la API

Tabla 5.5: API Rest - Endpoint token de sesión.

Este componente recibe una de las peticiones anteriores, realiza una serie de comprobaciones (token de seguridad, correcto formato de los datos, etc.), accede a la base de datos utilizando la información proporcionada en la llamada y devuelve el recurso demandado en cada caso.

El desarrollo de este componente no ha supuesto una gran dificutad, sin embargo, se ha invertido mucho tiempo en comprender la generación y funcionamiento de tokens de sesión para poder securizar las comunicaciones entre la API y el frontend.

Este componente se encuentra estrechamente relacionado con la tarea de *Comunicación bajo demanda con el frontend*, cuya estimación de horas fue de 14. En este caso se ha retrasado cuatro horas la implementación, sumando en total 18 horas a la finalización de la API.

5.2.5. Base de datos - PostgreSQL

Para cumplir con la funcionalidad de almacenamiento de datos, se ha instalado y configurado una base de datos PostgreSQL en el sistema siguiendo las indicaciones de la Referencia [8]. Se ha decidido utilizar esta tecnología por sus ventajas como la alta disponibilidad, la tolerancia a fallos, la escalabilidad y el buen rendimiento de una base de datos PostgreSQL. Para facilitar la administración de la base de datos, se utiliza DBeaver, un Sistema de Gestión de Bases de Datos (SGBD) que proporciona una interfaz gráfica intuitiva.

DBeaver permite estructurar la base de datos de manera eficiente y realizar consultas SQL de forma sencilla. Además, se ha implementado un código en SQL para crear la base de datos y las tablas presentes en el diagrama de la Figura 4.5. Este código se ejecutó exitosamente en la base de datos denominada VDTR a través de DBeaver, facilitando el proceso de implementación y gestión de la base de datos. DBeaver proporciona un diagrama con las tablas de la base de datos, Figura 5.5.

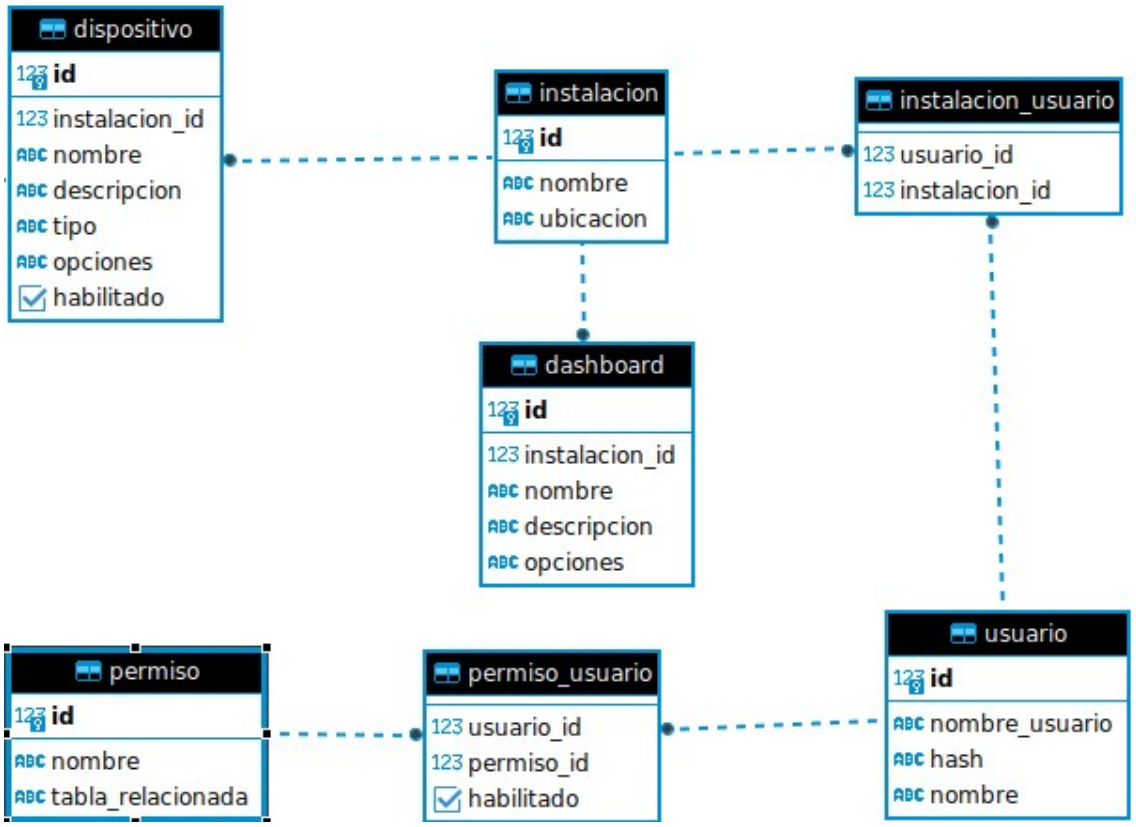

Figura 5.5: Diagrama de tablas generado por DBeaver

En el diagrama mostrado en la Figura 5.5, no se incluye la tabla correspondiente a los dispositivos implementada al final del proyecto. La creación dinámica de las tablas de dispositivos da lugar a una estructura similar a la de la Figura 5.6. Como se mencionó anteriormente, todas las tablas relacionadas con los dispositivos se generan de manera dinámica desde el módulo principal.

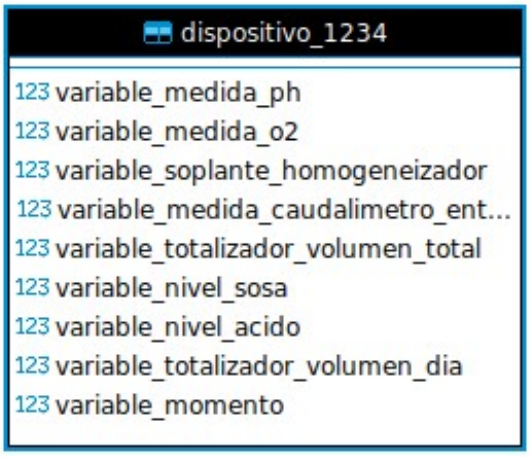

Figura 5.6: Ejemplo de tabla de un dispositivo

Respecto al plan de trabajo reflejado en la Tabla 3.1, esta implementación se relaciona con la tarea *Almacenamiento de los datos recibidos* la cual se estimaba en 10 horas. No ha ocurrido ningún problema durante este proceso, de modo que se ha finalizado antes de lo previsto, tan solo se han necesitado 8 horas.

5.2.6. Proyecto Vue.js

Por último, con el objetivo de permitir que el usuario acceda a la información, se ha desarrollado una plataforma sencilla utilizando el framework Vue.js. Este framework emplea componentes reutilizables. En otras palabras, la interfaz de usuario está compuesta por partes que pueden ser utilizadas en múltiples interfaces, siguiendo el principio DRY (Don't Repeat Yourself) de la programación. Esto no solo facilita la creación de nuevas interfaces, sino que también agiliza el proceso. Para el aprendizaje de Vue.js se ha consultado la web y documentación oficial, y como material de apoyo la Referencia [10].

En la siguiente imagen, Figura 5.7, podemos observar una interfaz creada para un usuario, en este caso el cliente de la industria depuradora, con indicadores como el caudal, el oxígeno, el PH, etc. Se han remarcado en distintos colores los componentes que forman la pantalla.

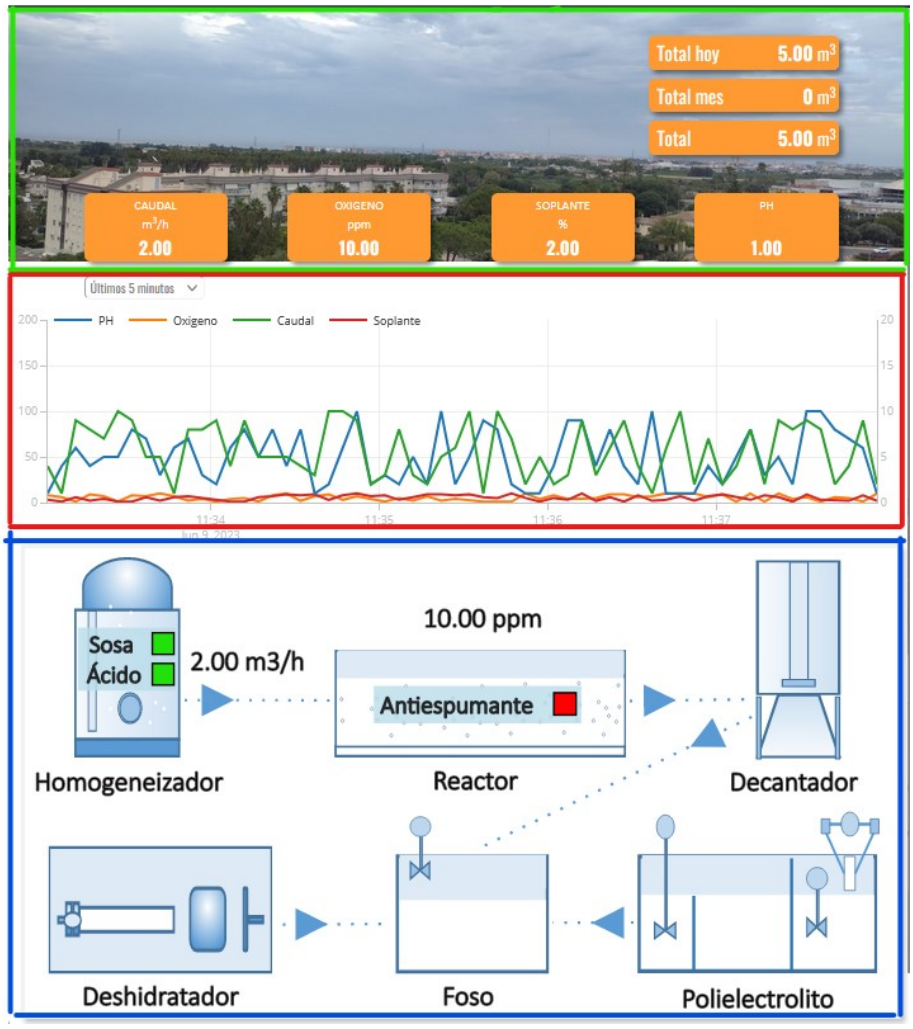

Figura 5.7: Interfaz de usuario de la plataforma

El primer componente, remarcado en verde, corresponde a los indicadores en tiempo real de la industria. Estos indicadores siempre muestran el último dato recibido en el frontend correspondiente al campo al que están asociados.

El segundo componente, remarcado en rojo, corresponde a la gráfica. Esta gráfica se actualiza dinámicamente a medida que el frontend recibe datos nuevos, mostrando así un efecto de movimiento. Como se observa en la Figura 5.8, la leyenda de la propia gráfica sirve como filtro para visualizar únicamente las variables que se encuentran seleccionadas en ese instante.

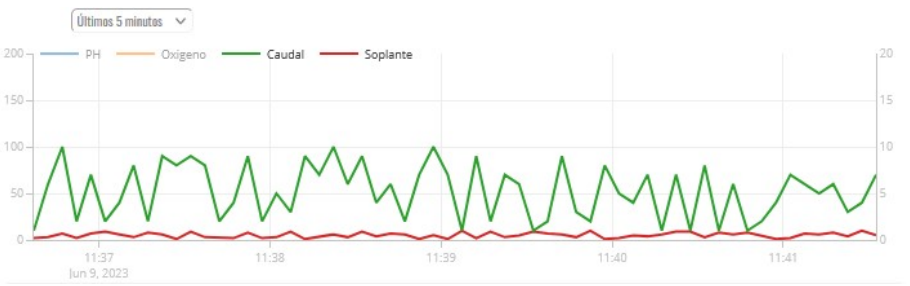

Figura 5.8: Filtro de variables en la gráfica

Por último, remarcado en azul, se ha desarrollado una imagen SVG que representa el proceso que se produce en la industria donde se implementará la solución. En esta imagen se encuentran varios indicadores. Los indicadores numéricos cambian a medida que reciben los datos a tiempo real. Por otro lado, los indicadores correspondientes a activadores, cambian al color verde en caso de encontrarse activados y a rojo en caso contrario.

La dificultad de la tarea ha recaído en el tiempo de aprendizaje del framework y del uso de sus librerías de conexión. Además, el diseño del frontend se ha implementado desde cero utilizando css.

Esta implementación se encuentra relacionada con las tareas de *Diseño y desarrollo del frontend y Recepción y explotación de los datos en el frontend* del Plan de trabajo, Tabla 3.1. Ambas tareas suman un total de 38 horas, pero se han utilizado aproximadamente 68 horas, ya que se ha invertido mucho tiempo en el aprendizaje.

CAPÍTULO 6
Implantación

En el presente capítulo se presentan los pasos realizados para la instalación de la solución desarrollada anteriormente en un servidor local al cual poder acceder mediante una IP local.

El sistema donde se ha implantado ha sido una máquina virtual con sistema operativo Ubuntu Server 22.04, y el servidor web utilizado ha sido Apache. Como se observa, se continúa con la dinámica de utilizar tecnologías de código abierto, para proporcionar la solución más asequible.

6.1 Sistema Ubuntu Server

Pero el hecho de que la tecnología sea de código abierto no es la única razón por la que se ha escogido esta en concreto. Algunas de las razones por las que se ha optado por usar Ubuntu Server como sistema host son:

- Proporciona estabilidad y seguridad, ya que este ofrece soporte a largo plazo, recibe actualizaciones de seguridad y correcciones de errores.
- Existe una amplia documentación sobre estos sistemas, además de una gran comunidad detrás de ellos.
- Además, este sistema ofrece compatibilidad y versatilidad con múltiples hardwares y softwares.

6.2 Servidor Apache 2.0

Por otro lado, en la propia máquina Ubuntu Server se ha instalado Apache 2.0 para poder hacer accesible el servicio de la plataforma frontend desarrollada anteriormente. Se ha optado por esta tecnología por las siguientes razones:

- Es el servidor web HTTP más popular, por lo que existe una gran comunidad de usuarios que brindan soporte. A su vez, existe gran documentación sobre el servidor que facilita su instalación y resolución de problemas.

- Posee gran estabilidad y rendimiento, ya que es capaz de manejar cargas de trabajo de alto tráfico.
- Apache ofrece módulos según las necesidades del escenario.
- Tiene una amplia gama de módulos y herramientas para reforzar la seguridad del servidor.

Para la instalación y configuración de este servicio, se han seguido los pasos de los capítulos uno y dos de la Referencia [7]. En este libro se encuentra una guía completa de instalación, configuración y mantenimiento del servidor Apache en varios sistemas operativos.

```
# Instalar servidor Apache 2.0 en un sistema Ubuntu
$ sudo apt-get install apache2 apache2-utils
```

Apache se encuentra estructurado por directorios, tal y como se observa en la Figura 6.1.

Figura 6.1: Estructura de directorios de Apache en un sistema Ubuntu

Entre ellos encontramos el archivo de configuración del servidor, el directorio de módulos instalados en servidor, el directorio de los sitios web accesibles en el servidor, etc.

6.3 Implantación del proyecto en el servidor

Hasta el momento, se ha trabajado con una máquina virtual Ubuntu Desktop, donde se encontraban todos los servicios corriendo (base de datos PostgreSQL, broker Mosquitto, broker Crossbar.io, ...). En este caso, se van a trasladar todos esos servicios a la máquina actual Ubuntu Server. Sin embargo, dado que se conectan entre ellos utilizando la dirección IP, podría instalarse cada servicio en un host distinto, formando de este modo un sistema distribuido.

De forma que la implantación en la mayoría de los casos es repetir las instalaciones que se han realizado en la etapa de desarrollo, cambiando las direcciones IP para que apunten al servidor nuevo (o si corren en la misma máquina pueden comunicarse mediante la dirección "localhost"). A su vez, debe instalarse Python y mover los componentes del desarrollo a la máquina Ubuntu Server.

La única excepción donde cambia la implantación es en el proyecto Vue.js del frontend. Para poder realizar un despliegue en producción de un proyecto Vue.js se debe ejecutar el comando *npm run build* . Este comando construye una carpeta con el proyecto limpio y con todos los recursos necesarios, Figura 6.2.

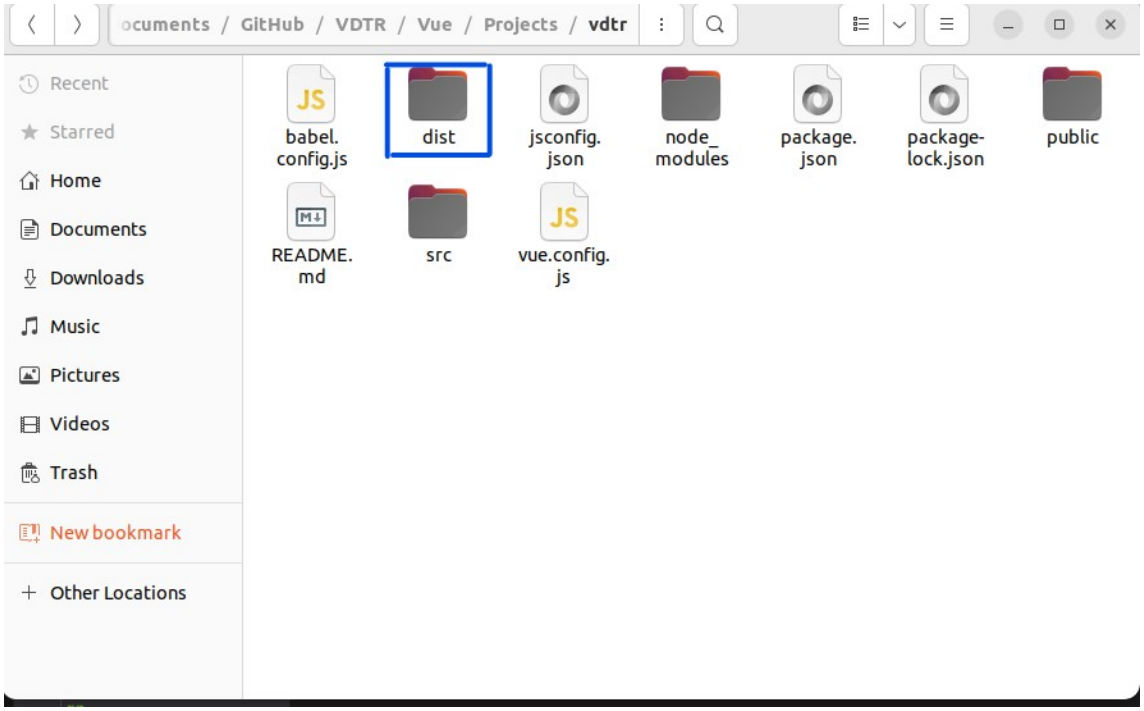

Figura 6.2: Carpeta "dist"del proyecto Vue.js

Como el servidor Apache solamente correrá este servicio, se colocan todos los archivos que se encuentran dentro de la carpeta ”dist” en el directorio *”var/www/html”* de apache. Una vez hecho este paso, ya se puede acceder a la plataforma a través de la IP del servidor.

Durante la realización de esta tarea, ha surgido un problema relacionado con la reescritura de las URLs en el servidor. De forma que no daba la posibilidad de cambiar de interfaz entre unas ventanas y otras del proyecto. La solución a este problema ha sido crear un archivo ”htaccess” utilizando el módulo de Apa-

che ”mod_rewrite.c” y escribiendo una configuración para poder habilitar esta función. Este archivo se incluye junto con los demás, en el mismo directorio y, finalmente funciona según lo esperado.

Esta implantación se relaciona con la tarea *Implantación de la solución en un servidor* del plan de trabajo, Tabla 3.1. En este caso se estimaron 14 horas para esta tarea, pero la realidad ha aumentado hasta 16 horas.

CAPÍTULO 7
Pruebas

Una vez finalizada la fase de implantación comienza la fase de pruebas, donde se somete al sistema a un conjunto de pruebas con las que comprobar que el proyecto sigue el funcionamiento esperado.

Dado que no se dispone de un PLC para la realización de estas pruebas, se han programado unos simuladores en Python que envían información aleatoria al sistema, de la misma manera que lo haría un PLC real.

7.1 Diseño de las pruebas

Las pruebas que se realizan en el sistema se basan en los casos de uso representados en las Figuras 3.6 y 3.7.

En la primera prueba, que se basa en la Figura 3.6, se verifica mediante registros en la terminal si la información llega correctamente al sistema. Una vez confirmado esto, se procede a iniciar sesión desde el frontend como si fuera un cliente y se verifica que la información recibida coincida con la enviada desde el PLC simulado.

La segunda prueba, basada en la Figura 3.7, consiste en iniciar sesión en la plataforma como si fuera el cliente y solicitar al sistema datos de momentos anteriores. El sistema debe proporcionar esos datos y mostrarlos en forma de gráfica en la pantalla.

La tercera prueba se ha diseñado con el propósito de evaluar la escalabilidad del sistema al agregar un nuevo dispositivo backend. El objetivo es comprobar si el sistema es capaz de manejar de manera eficiente la recepción y procesamiento de la información proveniente de este dispositivo adicional. En caso de ser la primera vez que se conecta, el sistema deberá ser capaz de escalar automáticamente y crear una tabla correspondiente a dicho dispositivo en la base de datos. Esta función se ha implementado en el proceso Python principal del sistema, el cual comprueba mediante llamadas a la base de datos si la tabla correspondiente al id del dispositivo ya existe.

7.2 Resultados

Estos son los resultados obtenidos en las distintas pruebas realizadas.

7.2.1. Primera prueba

Tal y como se observa en la Figura 7.1, los datos se reciben de forma correcta en el sistema, siguiendo el formato definido en la Figura 4.11. con un intervalo de tiempo regular especificado por el PLC simulado. Por otra parte, se ha creado en la base de datos un usuario con una contraseña de prueba para comprobar el correcto funcionamiento del inicio de sesión. Como se observa en la Figura 7.2, si el cliente falla sus credenciales de acceso, la plataforma se lo comunica con un mensaje en rojo. Una vez ha iniciado sesión, se ha comprobado que los datos que recibe el frontend cambian al mismo tiempo que se imprimen por la terminal del sistema. Por lo tanto esta prueba ha resultado exitosa.

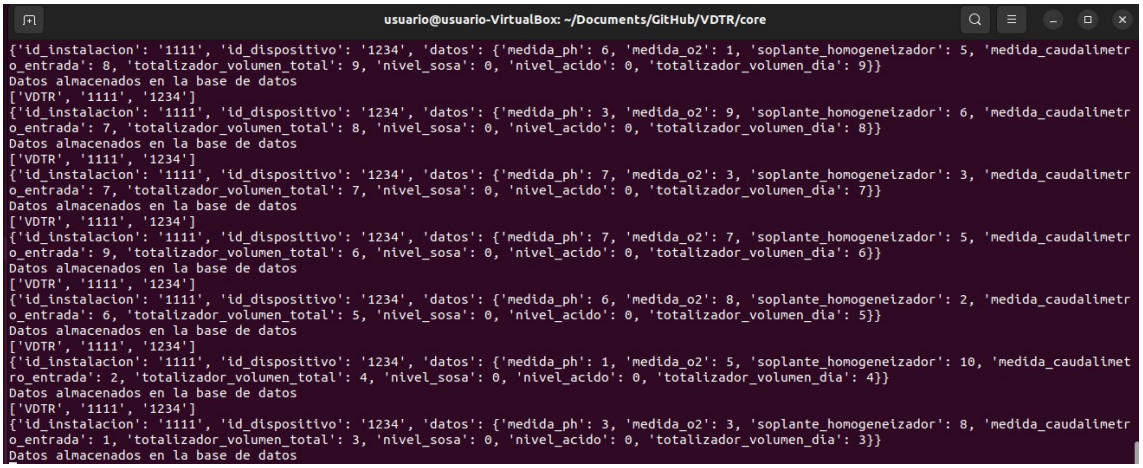

Figura 7.1: Registros de la terminal

Figura 7.2: Fallo en el inicio de sesión

7.2.2. Segunda prueba

Para la realización de la segunda prueba se ha hecho uso del mismo usuario de prueba que en el anterior apartado. El cliente ha iniciado sesión correctamente y ha visualizado sus datos en tiempo real. A continuación ha pulsado en el filtro de tiempo de la gráfica y ha desplegado las distintas opciones que este ofrece, Figura 7.3. De entre todas las opciones, escoge la última "Filtrar Fecha", y selecciona en una ventana desplegable un rango de fecha y hora.

Figura 7.3: Filtro temporal de la gráfica

Entonces la gráfica detiene el movimiento de actualización automático y recupera del sistema los datos comprendidos en el intervalo especificado para mostrarlos posteriormente en la pantalla, Figura 7.4. El resultado de esta prueba también resulta exitoso.

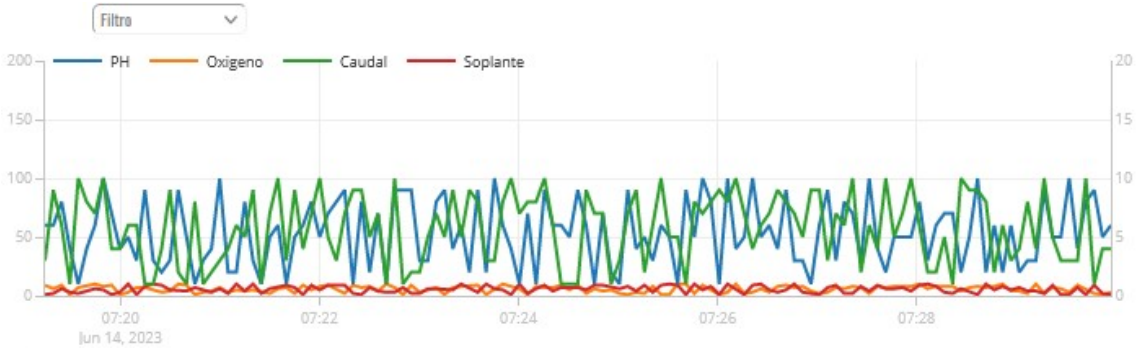

Figura 7.4: Gráfica con datos históricos

7.2.3. Tercera prueba

Para la última prueba, puesto que el PLC es simulado, se ha cambiado de valor la variable "*id_dispositivo*", de forma que el sistema no reconozca al mismo simulador como el mismo PLC, si no como uno diferente. Según la implementación, el sistema creará una nueva tabla para este dispositivo siguiendo la nomenclatu-

ra "*dispositivo_{id_dispositivo}*". En este caso, el identificador del dispositivo es "5678".

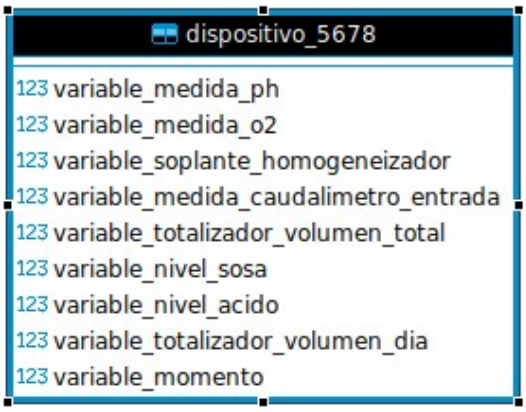

Figura 7.5: Tabla creada de forma dinámica

Como se observa en la Figura 7.5, se ha creado de forma dinámica y automática una nueva tabla en la base de datos. Los atributos de esta tabla son las variables que se envían desde el PLC. Además, podemos notar que el nombre de la tabla sigue la nomenclatura mencionada anteriormente. Por lo tanto, podemos concluir que el resultado de la prueba ha sido exitoso.

En este capítulo, hemos concluido con éxito todas las pruebas diseñadas para este proyecto. Con respecto al Plan de trabajo, como se muestra en la Tabla 3.1, esta tarea corresponde a las pruebas. Se estimaron 4 horas para completarla, sin embargo, se utilizaron solo 3 horas para verificar el correcto funcionamiento.

CAPÍTULO 8
Conclusiones

Durante el desarrollo de mi TFG me propuse alcanzar una serie de objetivos, presentes en el capítulo de Introducción. A continuación, reflexionaré sobre si se han logrado dichos objetivos y compartiré lo que he aprendido a nivel profesional y personal durante este proyecto.

En primer lugar, pude establecer comunicaciones efectivas entre los distintos dispositivos que participan en el sistema. Implementé mecanismos de comunicación confiables y eficientes que permitieron la transmisión de datos entre los dispositivos de manera fluida. Esto aseguró la interoperabilidad y el intercambio de información clave para el funcionamiento del sistema.

Además, logré hacer uso de formatos de datos canónicos y definir su explotación en el sistema. Establecí un conjunto de formatos estándar para representar y transmitir los datos, lo que facilitó la integración de distintos dispositivos y la interpretación consistente de la información en todo el sistema.

Otro objetivo alcanzado fue el diseño de un mecanismo de persistencia de los datos en el sistema. Implementé una base de datos en PostgreSQL que permitió guardar de forma segura y confiable los datos recibidos. Esto facilitó su posterior acceso y análisis, garantizando la disponibilidad de información histórica.

Asimismo, tuve en cuenta la flexibilidad y escalabilidad de la solución durante todo el proceso. Diseñé el backend de manera modular y escalable, lo que permitió la incorporación de nuevos dispositivos de manera sencilla y sin afectar el funcionamiento global del sistema. Además, el frontend desarrollado con Vue.js fue diseñado de forma modular y adaptable, brindando una experiencia de usuario flexible y permitiendo su expansión según las necesidades futuras.

En cuanto a la transmisión de datos en tiempo real, diseñé una solución que cumplió con este objetivo. Implementé tecnologías y protocolos que permitieron una comunicación en tiempo real entre los distintos componentes del sistema, asegurando que los datos fueran actualizados y disponibles de forma inmediata.

También diseñé un mecanismo complementario que permitió a los clientes visualizar datos históricos. Implementé funcionalidades, a través de una API REST, que permitieron acceder y visualizar datos almacenados previamente en el sistema, lo que brindó a los usuarios una visión histórica y permitió un análisis retrospectivo de la información.

En cuanto a la gestión del tiempo del proyecto y el cumplimiento de las fechas de entrega, debo reconocer que no logré completar este objetivo satisfactoriamente. Aunque hice esfuerzos por establecer una planificación adecuada, me encontré con desafíos inesperados y dificultades técnicas que retrasaron la finalización de algunas tareas. Sin embargo, esta experiencia me ha enseñado la importancia de una gestión efectiva del tiempo y la necesidad de adaptarse y reevaluar los plazos a lo largo del proyecto.

Por último, a lo largo de este proyecto he aprendido a aportar un pensamiento crítico a cada decisión tomada. Tuve que evaluar diferentes alternativas, sopesar sus ventajas y desventajas, y tomar decisiones fundamentadas en función de los objetivos y requisitos del sistema. Este enfoque crítico me permitió tomar decisiones informadas y optimizar el diseño y funcionamiento del sistema.

En resumen, a pesar de no haber logrado cumplir completamente con el objetivo de gestión del tiempo del proyecto, he alcanzado la mayoría de los objetivos establecidos inicialmente. Este proyecto me ha brindado valiosas lecciones y aprendizajes a nivel profesional y personal. He adquirido habilidades técnicas en la integración de aplicaciones y el desarrollo de soluciones escalables, así como habilidades de pensamiento crítico y toma de decisiones. Estoy seguro de que esta experiencia me será de gran utilidad en mi futuro profesional.

Haciendo referencia a la industria depuradora de donde se extrae la problemática, se han cubierto todas las necesidades comentadas en la fase de requisitos. Es por ello que podemos decir que el proyecto ha resultado exitoso. El siguiente paso para con la industria sería, instalar el sistema en un servidor, local o remoto, y crear usuarios para todos los cargos que vayan a hacer uso del servicio.

8.1 Tiempos estimados y tiempos reales

Como se puede observar en la actualización del plan de trabajo, Tabla 8.1, se han registrado los tiempos reales invertidos en cada tarea del proyecto. Al compararlos con el plan de trabajo anterior, representado en la Tabla 3.1, se puede notar que en la mayoría de las tareas se ha requerido más tiempo del estimado inicialmente, acumulando un total de 45.5 horas adicionales al proyecto. Afortunadamente, dado que se inició el proyecto con suficiente antelación, este retraso no afectará a la entrega final. No obstante, este resultado nos brinda una valiosa lección sobre la duración real de cada tarea, conocimiento que aplicaremos en futuros proyectos.

Tabla 8.1: Plan de trabajo con tiempos reales

Tarea desarrollada	**Tiempo real invertido (horas)**
Comunicación con los dispositivos ya instalados	18 + 3.5 = 21.5
Explotación de los datos recibidos del backend	10
Almacenamiento de los datos recibidos	8 + 3.5 = 11.5
Comunicación instantánea con el frontend	22 + 3.5 = 25.5
Comunicación bajo demanda con el frontend	18

Tabla 8.1: Tareas

Tarea a desarrollar	**Tiempo real invertido (horas)**
Diseño y desarrollo del frontend	48
Recepción y explotación de los datos en el frontend	20
Implantación de la solución en un servidor	16
Pruebas	3
Total horas realizadas	171.5

8.2 Relación con los estudios cursados

La realización de este proyecto ha sido una experiencia enriquecedora que me ha permitido aplicar y profundizar los conocimientos adquiridos a lo largo de mis estudios en la universidad. La relación con los estudios cursados ha sido fundamental en varias áreas del proyecto. A continuación se detallan las principales conexiones con este trabajo:

- Para cumplir con la función de persistencia de los datos, se han aplicado conceptos de las asignaturas de Bases de Datos y Sistemas de Información, así como Tecnología de Bases de Datos. Además, se ha utilizado el lenguaje SQL aprendido en estas asignaturas para la definición de la base de datos, sus tablas y las consultas.
- En la implementación del proyecto, se utilizó un servidor Apache que fue estudiado y configurado en la asignatura de Sistemas y Servicios en Red. Por lo tanto, la tarea relacionada con el servidor no presentó un gran desafío.
- En cuanto a las conexiones seguras del proyecto, se gestionaron mediante certificados autofirmados. Este tipo de gestión se estudió en la asignatura de Redes Corporativas, donde se dedicó un tema a los certificados y las entidades que intervienen en el proceso real de firma y emisión de los mismos.
- Para el proceso de análisis y extracción de requisitos, se utilizaron técnicas aprendidas en la asignatura de Desarrollo Centrado en el Usuario. Aunque no se empleó la metodología DCU en su totalidad debido a que el proceso no fue tan iterativo con el usuario, se extrajeron técnicas e ideas que resultaron útiles en el proyecto.
- Al desarrollar las pantallas de visualización, se aplicaron conocimientos relacionados con las asignaturas de Interfaces Persona-Computador y Desarrollo Centrado en el Usuario.
- Por último, este trabajo se puede relacionar directamente con la asignatura de Integración de Aplicaciones. En el proyecto se abordó un problema real de integración, aplicando soluciones estudiadas en dicha asignatura, como colas de mensajería, APIs y formatos canónicos.

CAPÍTULO 9
Trabajos futuros

En cuanto al futuro del proyecto desarrollado, existen varias áreas que podrían ser exploradas para ampliar y mejorar el proyecto. A continuación, se presentan algunas sugerencias que han surgido a lo largo de este:

- Sistema distribuido: aunque el proyecto ha sido implementado de forma centralizada en una sola máquina, los componentes se comunican entre sí utilizando mecanismos que permiten su ejecución en hosts diferentes. Convertir este proyecto en un sistema distribuido podría brindar beneficios significativos en términos de eficiencia, escalabilidad y disponibilidad del sistema, evitando así puntos únicos de fallo.

- API-Led connectivity: se trata de una forma de conectar datos y aplicaciones a través de APIs reutilizables y funcionalmente completas. Sería beneficioso para el proyecto contar con este tipo de enfoque, ya que en un futuro puede que exista más de un solo sistema al que conectarse, más de una base de datos, y aplicando este enfoque, la integración de los nuevos sistemas sería transparente al cliente.

- IA: como se comentó en la introducción de este proyecto, ya se están utilizando mecanismos de inteligencia artificial con la finalidad de analizar los datos recogidos por dispositivos IoT y poder tomar decisiones sin necesidad de intervención humana. Esta podría ser una tarea beneficiosa para el proyecto, pudiendo analizar los datos recibidos y detallando un informe más completo, o incluso predictivo, al cliente.

- Deplegar en Cloud: actualmente el servidor en el que se encuentra el proyecto es un servidor local. Pero la finalidad sería poder acceder a él desde cualquier parte. Una forma de hacerlo sería desplegando dicho servidor en algún Cloud, pagando las tasas requeridas.

- Variedad de conectores: por el momento, el sistema únicamente admite el formato JSON de entrada en el backend. Una tarea interesante podría ser el desarrollo de conectores con otros formatos (CSV, XML, etc.), anticipándonos al caso en el que un dispositivo no tenga la capacidad de enviar la información en JSON.

- Auditar las conexiones y evitar los posibles ataques: en este trabajo se ha mencionado el uso de conexiones seguras y el uso de certificados, sin embargo, tal y como se puede consultar en la Referencia [4], existen muchos otros tipos de ataques que pueden afectar al sistema. Convendría realizar un estudio de estos ataques contra el sistema y la forma de evitarlos. El único tratado en este trabajo fue el ataque de SQL Injection, el cual se ha frenado haciendo reconocimientos de código en la API.

Bibliografía

[1] Pérez-López, Esteban. Los sistemas SCADA en la automatización industrial. *Revista Tecnología en Marcha*, 2015, vol. 28, no 4, pág 3-14.

[2] Boyes, Hugh and Hallaq, Bil and Cunningham, Joe and Watson, Tim. The industrial internet of things (IIoT): An analysis framework. *Elsevier*, 2018, vol. 101, pág 1-12.

[3] Semle, Aron Protocolos IIOT para considerar. *Aadeca Revista*, 2016, vol. 34.

[4] Panchal, Abhijeet C and Khadse, Vijay M and Mahalle, Parikshit N. Security Issues in IIoT: A Comprehensive Survey of Attacks on IIoT and Its Countermeasures. *IEEE.org*, 2018, pág. 124–130.

[5] R. Anandan, Suseendran Gopalakrishnan, Souvik Pal, Noor Zaman. *Industrial Internet of Things (IIOT)*. Wiley-Scrivener, USA, 2022.

[6] Gastón C. Hillar. *MQTT Essentials - A Lightweight IoT Protocol*. Packt Publishing, UK, 2017.

[7] Darren James Harkness. *Apache Essentials: Install, Configure, Maintain*. Apress, USA, 2nd edition, 2022.

[8] Luca Ferrari, Enrico Pirozzi. *Learn PostgreSQL*. Packt Publishing, UK, October 2020.

[9] Luciano Ramalho. *Fluent Python*. O'Reilly Media Inc. , 2nd edition, 2022.

[10] Maya Shavin. *Learning Vue*. O'Reilly Media Inc. , December 2022.

[11] Gaston C. Hillar. *Hands-On MQTT Programming with Python*. Packt Publishing, May 2018.

APÉNDICE A
Objetivos de Desarrollo Sostenible

Los Objetivos de Desarrollo Sostenible (ODS) representan una colección de 17 metas globales establecidas por las Naciones Unidas en 2015, como parte de la Agenda 2030 para el desarrollo sostenible. Estos objetivos abarcan una amplia gama de temas interrelacionados, como la erradicación de la pobreza, el acceso a una educación de calidad, la igualdad de género, la promoción de fuentes de energía limpia, entre otros. Los ODS fueron concebidos con el propósito de abordar los desafíos globales y fomentar un desarrollo sostenible en los ámbitos social, económico y ambiental. En la Figura A.1 se presentan los objetivos propuestos por las Naciones Unidas.

Figura A.1: Objetivos de Desarrollo Sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	Alto	Medio	Bajo	No procede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades.**				X
ODS 11. **Ciudades y comunidades sostenibles.**		X		
ODS 12. **Producción y consumo responsables.**		X		
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados. Con el desarrollo de este trabajo, se contribuye en mayor o menor medida con los siguientes objetivos:

- ODS 9: Industria, innovación e infraestructura: Este objetivo busca promover el desarrollo de infraestructuras sostenibles, fomentar la innovación tecnológica y apoyar el crecimiento económico sostenible. El proyecto se encuentra directamente ligado a este objetivo, ya que el contexto del problema que se trata ocurre en una industria y la solución a este consiste, precisamente, en innovar una nueva forma de transportar los datos desde los autómatas a los operarios. Con la solución se aumentaría la economía de la empresa al evitar fallos no controlados en la industria.

- ODS 11: Ciudades y comunidades sostenibles: Este objetivo se enfoca en hacer que las ciudades y los asentamientos humanos sean inclusivos, seguros, resilientes y sostenibles. La implantación de este proyecto podría contribuir a mejorar la eficiencia y la sostenibilidad de las industrias dentro de las ciudades. Además, otorgaría seguridad en la empresa al ofrecer el sistema de monitorización. Así pues, podría evitarse fallos catastróficos que afectasen a la ciudades donde se ubican dichas empresas.

- ODS 12: Producción y consumo responsable. Este objetivo promueve un consumo y una producción sostenibles. La creación de una plataforma de monitorización podría ayudar a identificar patrones de consumo y producción ineficientes, permitiendo tomar decisiones medidas para reducir el desperdicio y mejorar la eficiencia energética en las industrias.

Estos son los principales objetivos de desarrollo sostenible con los que se relaciona este proyecto. Como vemos, se relaciona estrechamente con el ODS 9, pero afecta de manera indirecta a otros objetivos.