Grado en Ingeniería Informática

Trabajo de Final de Grado

Implementación de un sistema de monitorización

Autor:
Pau Centelles Ortells

Supervisor:
José Luis Martínez
Tutor académico:
Jorge Sales Gil

Fecha de lectura: 27 de Abril de 2023
Curso académico 2022/2023

Resumen

Este documento presenta la memoria del trabajo final de grado, que consiste en el desarrollo de una aplicación web para la monitorización de la plataforma de recogida de datos *SmartCity,* de la empresa *IoTsens*. En este trabajo se detalla el proceso de planificación, análisis e implementación del proyecto.

La aplicación web desarrollada tiene como objetivo analizar los datos emitidos por los sistemas de soporte de la plataforma *SmartCity*. En particular, se realiza un análisis de rendimiento de los sistemas de *Dockers*, sistemas informáticos y el servicio de mensajería de la empresa. Y además, se proporciona un sistema de alarmas configurables para detectar de forma automática caídas del servicio de las distintas partes de la plataforma *SmartCity*.

Para el desarrollo de la aplicación, se siguió una metodología ágil de tipo Scrum y se utilizaron lenguajes de programación como Java para el desarrollo del *backend*, *JavaScript* para el desarrollo del *frontend* y los lenguajes InfluxDB y PHP para las peticiones a bases de datos. El *framework* de Spring Boot fue utilizado para el desarrollo de la aplicación.

Palabras clave

Panel de monitorización, Java, Spring Boot, aplicación web, administración de sistemas.

Abstract

This document presents the report of the final degree work, which consists of the development of a web application for monitoring the data collection platform Smart-City of the company *IoTsens*. This work details the process of planning, analysis and implementation of the project.

The web application developed aims to analyze the data emitted by the support systems of the *SmartCity* platform. In particular, a performance analysis is carried out for the Docker systems, computer systems, and the company's messaging service. In addition, a configurable alarm system is provided to detect automatically service failures of the different parts of the *SmartCity* platform.

For the development of the application, a Scrum-type agile methodology was followed, and programming languages such as Java for the backend development, JavaScript for frontend development, and InfluxDB and PHP for database requests were used. The Spring Boot framework was used for the development of the backend.

Keywords

Monitoring panel, Java, Spring Boot, web application, system administration.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

1.1.1. Descripción de la empresa

El Grupo Gimeno [1] fue fundado hace 150 años, con la creación de Facsa, una empresa especializada en el transporte y distribución de agua potable. A lo largo de los años, el grupo ha evolucionado y ha adaptado su actividad a las nuevas necesidades, hasta conformar un conjunto de empresas y divisiones especializadas en proporcionar servicios a los ciudadanos (ver figura 1.1).

En el año 2013, se fundó IoTsens como un departamento dentro de la división de Servicios, con el objetivo de ofrecer soluciones de *software* a las otras empresas del Grupo Gimeno. Un año después, en 2014, IoTsens inició su expansión para proporcionar sus servicios a empresas externas y ayuntamientos. Actualmente, es una empresa pionera a nivel nacional en el desarrollo de ciudades inteligentes.

El principal cometido de IoTsens es el desarrollo de soluciones verticales enfocadas en el Internet de las Cosas[1] (IoT), capaces de comunicarse con dispositivos y sensores para su monitorización y configuración remota. Estas soluciones se caracterizan por estar construidas siguiendo las especificaciones de un sector o empresa en particular, para poder así satisfacer unas necesidades concretas que no se encuentran en un *software* genérico [1].

IoTsens proporciona una arquitectura flexible que permite el desarrollo de soluciones verticales para todo tipo de escenarios. Este sistema permite una estructura dinámica y adaptable a todo tipo de sistemas, a través de los cuales se realiza una comunicación bidireccional, recibiendo los datos de sistemas que posteriormente serán analizados y explotados para ajustar el propio sistema de acuerdo con los resultados obtenidos.

[1]Internet de las cosas (IdC) o, en inglés, *Internet of things (IoT)*, describe objetos físicos (o grupos de estos) con sensores, capacidad de procesamiento, software y otras tecnologías que se conectan e intercambian datos con otros dispositivos y sistemas a través de internet u otras redes de comunicación [2].

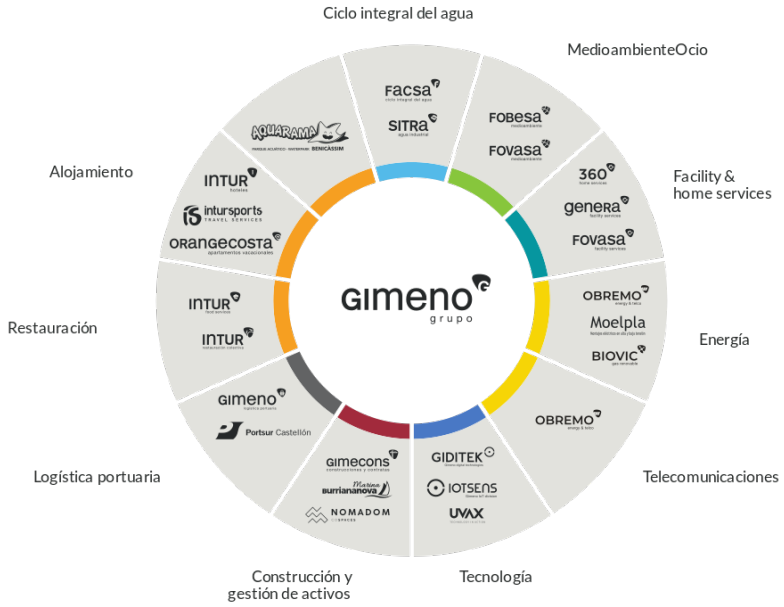

Figura 1.1: Estructura empresarial del Grupo Gimeno.

Entre las soluciones verticales que ofrece IoTsens se encuentran:

- **SmartCity**: una solución que permite la recopilación y monitorización remota de datos producidos por redes de sensores ubicados en una ciudad, para su posterior análisis.
- **SmartWater**: un servicio que digitaliza el proceso de lectura de contadores de agua, para empresas o particulares.
- **SmartBuilding**: una solución que posibilita la gestión y control remoto de edificios, para mejorar su seguridad, eficiencia y accesibilidad.
- **SmartIrrigation**: mediante sensores, permite monitorizar y automatizar los sistemas de riego de una instalación, y así conseguir una mayor eficiencia de gestión.

1.1.2. Descripción del proyecto

Este proyecto surge como solución a la necesidad de monitorizar los diversos sistemas que conforman SmartCity, como servidores, el sistema de *pods*[2] y el servicio de mensajería, con el fin de detectar posibles caídas de servicio o errores.

IoTsens cuenta con una plataforma de control de dispositivos, conocida internamente como *Control Platform*. Esta plataforma es una aplicación web que proporciona las herramientas para gestionar los dispositivos y sensores monitorizados por IoTsens. El proyecto desarrollado

[2]Pod: Es un grupo de micro-servicios comparten los recursos de almacenamiento y red.

se integrará en la arquitectura existente, proporcionando nuevas vistas para monitorizar los nuevos recursos, y una nueva herramienta, que permita la configuración de alarmas para los recursos monitorizados.

Es importante destacar que los sistemas a monitorizar ya estaban implementados y funcionado correctamente antes del inicio de la estancia de prácticas. Además, la plataforma de control de la empresa ya se encontraba desplegada en un servidor. El proyecto pretende ampliar la funcionalidad de esta plataforma de control, mediante la inclusión de nuevas vistas que permitan visualizar las métricas de los sistemas, y herramientas para detectar y notificar posibles fallos en los sistemas.

La arquitectura de IoTsens consta de cuatro capas diferenciadas:

1. **Dispositivos y sensores:** Capa formada por todos los dispositivos de IoT que monitorizan recursos y emiten señales con la información recopilada.
2. **Red y conexiones:** Capa que contiene los dispositivos que forman las redes de comunicación para el intercambio y envío de datos entre sistemas.
3. **Almacenamiento y procesamiento de datos:** Capa encargada de recopilar y almacenar información producida por los sensores y aplicaciones de la empresa. Para la comunicación e intercambio de mensajes, se utilizan servidores de RabbitMQ[3]. Las métricas de sistemas se almacenan en la base de datos de InfluxDB[4], donde se registran automáticamente las últimas medidas disponibles. Esas medidas se registran mediante el uso de la herramienta *Telegraf*[5] que lee el estado del sistema y almacena un registro en la base de datos. También existen otras bases de datos de apoyo que sirven para almacenar información de configuración y registros del sistema.
4. **Aplicación:** La última capa donde se alojan todas las aplicaciones que forman parte de IoTsens. El proyecto desarrollado se aloja en esta capa, y se comunicará con la capa de almacenamiento de datos para su funcionamiento.

La aplicación a desarrollar debe ofrecer una interfaz para la monitorización en tiempo real de los sistemas de mensajería alojados en la capa tres, así como los sistemas y servidores que forman parte de IoTsens. El sistema de alertas de la aplicación debe permitir la configuración para cualquier fuente y tener la capacidad de responder a cualquier valor monitorizado que se encuentre fuera de los parámetros especificados.

Durante el desarrollo del proyecto, he colaborado con el departamento de *software* para crear un programa que recoja la información de la base de datos, la procese y muestre en pantalla los datos relevantes para facilitar su análisis. Además, el programa debe lanzar alertas de forma automática en caso de que se cumplan las condiciones especificadas por un usuario. Esas condiciones se almacenan en una base de datos de MariaDB[6], que también guarda un

[3]RabbitMQ: Sistema de almacenamiento temporal y envío de mensajes que funciona como intermediario entre los clientes y las aplicaciones.

[4]InfluxDB: base de datos especializada en operaciones de lectura y escritura en tiempo real.

[5]Telegraf es un servicio que recopila y envía métricas y eventos de bases de datos, sistemas y sensores.

[6]MariaDB: sistema de gestión de bases de datos derivado de MySQL con licencia GPL (General Public License).

registro a corto plazo de las alertas emitidas por sensores, y los mensajes de advertencia y errores producidos por sistemas y aplicaciones.

Para garantizar la escalabilidad y la facilidad de implementación de nuevos sistemas a monitorizar en un futuro, la aplicación debe seguir una estructura modular. La aplicación debe ser diseñada pensando en que se podría expandir su funcionalidad en el futuro para monitorizar nuevos sistemas. Actualmente, los sistemas que se busca monitorizar son:

- Las colas de mensajes de RabbitMQ. Se debe poder consultar los mensajes acumulados en una cola, la tasa de consumo, y la tasa de producción de mensajes. La presencia de un gran número de mensajes en una cola podría indicar que el consumidor ha dejado de funcionar o que no está funcionando de forma eficiente.

- La monitorización del rendimiento y estado de los servidores de IoTsens. Se mostrarán los niveles de CPU, memoria y disco de cada uno de los sistemas conectados a la empresa.

- El estado de los *pods* de la empresa. Todos los *pods* están configurados para que se reinicien automáticamente en caso de error o parada del servicio. Por lo tanto, se monitorizará el número total de veces que se han reiniciado los dispositivos, ya que una cifra elevada indicaría que el contenedor no se está ejecutando correctamente y está constantemente reiniciándose.

1.2. Objetivo y alcance del proyecto

Los datos relacionados con los servidores de la empresa pueden ser visualizados a través de la aplicación *Grafana*[7], mediante el uso de consultas a la base de datos. Para acceder a la información de los *pods*, es necesario conectarse a la API[8] (Application Programming Interface) del servidor de Kubernetes[9], mientras que para visualizar el estado de las colas de RabbitMQ, se utiliza una aplicación web proporcionada por el mismo software. Actualmente, el administrador de sistemas debe utilizar tres aplicaciones distintas para comprobar si hay algún problema en los sistemas de la empresa. La motivación del proyecto a desarrollar es ofrecer una solución que solvente este problema.

El objetivo del proyecto es desarrollar una aplicación web sencilla y escalable que sea capaz de mostrar el estado actual de los sistemas de la plataforma *SmartCity*. Además, deberá ofrecer una interfaz donde se puedan configurar alarmas para los recursos monitorizados, que se activen cuando se cumplan las condiciones establecidas y generen excepciones, que contengan un mensaje con la información del motivo por el que se ha lanzado la alarma.

Para alcanzar el objetivo del proyecto, se pueden dividir las distintas tareas en las partes que forman el alcance del proyecto:

[7]Grafana: Aplicación web de código abierto que se conecta a bases de datos de InfluxDB para la visualización y formato de datos.

[8]API: Interfaz de programación de aplicaciones. Son funciones que permiten la comunicación entre programas distintos.

[9]Kubernetes: Plataforma que permite la automatización y despliegue de servicios mediante el uso de *pods*.

- Crear un micro-servicio que lea el estado de las tres fuentes a monitorizar (servicio de mensajería, servidores y *pods*) de forma eficiente, y que se ejecute periódicamente para actualizar esa información.
- Desarrollar una aplicación de Spring Boot[10] que utilice el micro-servicio anterior para responder peticiones HTTP[11] (Hypertext Transfer Protocol) entrantes con mensajes que contengan las métricas de los recursos monitorizados.
- Implementar un sistema que permita crear y configurar alarmas, que analice los recursos monitorizados en busca de valores que hagan saltar las alarmas activas, y que envíen un mensaje de error a la cola de mensajería correspondiente.
- Crear una interfaz sencilla y reutilizable que permita mostrar en pantalla los datos en tiempo real de los recursos, y que permita crear alarmas, configurarlas y ver un registro de las excepciones generadas por las alarmas creadas.

El alcance del proyecto abarca todo el desarrollo de la aplicación, el *backend* y la interfaz web que mostrará los datos. Sin embargo, no se incluye la configuración de las bases de datos de InfluxDB, ni la configuración de los servidores y *pods*, ya que han sido diseñados y configurados previamente por la empresa.

1.3. Objetivo y alcance del producto

El objetivo del producto actual es facilitar el trabajo del departamento de administración de sistemas, proporcionando una interfaz clara y sencilla que muestre el estado de los sistemas de la empresa. Además, debe proporcionar las herramientas para la creación y configuración de alarmas para detectar irregularidades de funcionamiento y notificar al equipo de administración de sistemas, con el fin de responder rápida y eficazmente ante ellas.

El alcance del producto se puede dividir en tres puntos de vista, según el área en el que se apliquen. Estos son el alcance funcional, organizativo e informático.

1.3.1. Funcional

El alcance funcional del producto define que debe ser capaz de obtener y presentar al usuario el estado actual de los sistemas de la empresa de manera clara y comprensible. Por otro lado, debe proporcionar una interfaz que permita la configuración de alarmas que actúen sobre un conjunto de sistemas, y mostrar una lista de las últimas excepciones lanzadas por las alarmas, con información relevante sobre su causa. La plataforma de la empresa requiere inicio de sesión para acceder a la aplicación de monitorización, el acceso a la aplicación de monitorización está restringido a usuarios autorizados. Por lo tanto, se distinguen tres tipos usuarios:

[10] Spring Boot es un *framework* que proporciona un modelo basado en Java para el desarrollo de aplicaciones empresariales, automatiza tareas de gestión de datos.

[11] HTTP: Protocolo de internet para la transferencia de información.

- **Superadministrador:** Es un tipo de usuario que cuenta con todos los permisos disponibles y tiene acceso a todas las funciones del portal web. Este tipo de usuario está destinado a usarse únicamente durante el desarrollo de la aplicación y durante la corrección de errores de funcionamiento.

- **Administrador de sistemas:** Este tipo de usuario puede realizar cualquier tipo de tarea dentro de la aplicación de monitorización. Al contrario que el superadministrador, sus privilegios son limitados fuera de esta. Tanto el superadministrador como el administrador de sistemas pueden usar los servicios de monitorización y pueden crear alarmas de los distintos sistemas monitorizados.

- **Usuario observador:** Este tipo de usuario solo tiene permisos para leer la información de los recursos monitorizados, no cuenta con más privilegios. Su funcionalidad está restringida a ser un observador de los datos.

1.3.2. Organizativo

La aplicación web será de ayuda para el departamento de administración de sistemas de la empresa, que harán uso de las herramientas de monitorización para simplificar sus tareas de mantenimiento, y usarán el sistema de alarmas para poder detectar problemas de forma automática.

1.3.3. Informático

Respecto al aspecto técnico, la aplicación depende de varios subsistemas de la empresa para su correcto funcionamiento. Entre ellos:

- **RabbitMQ:** Es el servicio de mensajería de la empresa. Los mensajes se escriben en colas, y se consumen por aplicaciones para que los procesen. Es uno de los tres sistemas a monitorizar, pero no se obtendrá el estado de sus colas a través de RabbitMQ, sino mediante Telegraf, que se explicará a continuación. El sistema de mensajería de RabbitMQ se utilizará como almacenamiento a corto plazo de los mensajes de error generados por el servicio de alarmas.

- **InfluxDB:** Es una base de datos especializada en la lectura y escritura de medidas en tiempo real. La empresa utiliza Telegraf, un agente de servidor basado en complementos, que recoge las métricas de los sistemas de mensajería y de los servidores, y los almacena en la base de datos de InfluxDB. Nuestra aplicación obtendrá los datos de los servidores y del sistema de mensajería a partir de esta base de datos.

- **Kubernetes:** Plataforma de automatización y gestión de contenedores, para la ejecución de servicios y aplicaciones. La aplicación del proyecto debe comunicarse con el servidor de Kubernetes para obtener sus métricas.

1.4. Descripción detallada del desarrollo del proyecto

El objetivo inicial del proyecto era desarrollar una aplicación web que mostrara información en tiempo real de los distintos sistemas que forman la plataforma de SmartCity. Una vez se completó la interfaz web para monitorizar los datos, y se verificó su correcto funcionamiento, se comenzó a implementar el sistema de alarmas. Sin embargo, durante su implementación surgieron problemas de rendimiento y funcionamiento que resultaron difíciles de solucionar, porque el enfoque inicial estaba planeado para ser usado por un solo usuario. Así que se realizó una reescritura significativa del servicio de obtención de datos para adaptarlo a un sistema más eficiente y escalable. Después de finalizar el sistema de alarmas, se iniciaron las tareas de integración de la aplicación en la plataforma de control de la empresa. Para ello, se debían implementar las funciones correspondientes en el *backend* del Control Platform para permitir la comunicación con la aplicación de monitorización, así como implementar la nueva interfaz y sistema de usuarios, junto con otras mejoras en el proyecto.

1.4.1. Tecnologías usadas

La funcionalidad principal del proyecto se implementó utilizando el *framework* de Spring Boot, que está basado en el lenguaje de programación de Java. Este *framework* se utiliza para crear los micro-servicios que proveerán el sistema de monitorización y alarmas a la aplicación principal. Se escogió para el proyecto, ya que es un *framework* de código abierto, muy utilizado y con abundante documentación. Existen numerosas alternativas, pero no cuentan con todas las ventajas de Spring Boot, por lo que solo deberían considerarse si el proyecto necesita unas funcionalidades muy concretas[3]. También se han utilizado las siguientes tecnologías:

- **Angular**: Es un *framework* basado en el lenguaje de programación TypeScript que es utilizado para realizar las vistas de la aplicación web. Se escogió porque se especializa en la creación de páginas web a partir de componentes reutilizables. Eso permite añadir nuevas funcionalidades a proyectos ya existentes de forma sencilla y escalable.
- **InfluxDB**: Sistema de gestión de bases de datos de métricas, eficaz en lecturas y escrituras en tiempo real. Se utiliza para realizar consultas y obtener el estado actual de los servidores y colas de mensajes de RabbitMQ. La ventaja respecto a otras bases de datos tradicionales, es que esta ha sido diseñada específicamente para el almacenamiento de métricas. Está diseñado para ser escalable y está optimizado para la ejecución de consultas de lectura y escritura en tiempo real, para la monitorización de sistemas.
- **RabbitMQ**: Sistema de almacenamiento temporal y envío de mensajes. Es una de las tecnologías más usadas por la empresa. Su función es actuar de intermediario entre dispositivos que producen información, denominados emisores, y los sistemas que obtienen esa información, los consumidores. RabbitMQ proporciona las herramientas para la construcción de un sistema de mensajería asíncrona entre servicios. El intercambio de mensajes se realiza mediante el protocolo MQTT[12] (Message Queuing Telemetry Transport). La empresa sigue una arquitectura basada en micro-servicios; esta arquitectura se beneficia

[12]MQTT: Protocolo de mensajería para comunicaciones entre sistemas y dispositivos. Usado principalmente por sensores y dispositivos de IoT para la transmisión de datos.

del uso de RabbitMQ para la comunicación entre sistemas y micro-servicios, ya que su sistema de colas asíncronas permite que los servicios pueden conectarse a una cola para almacenar mensajes, o para consumirlos (ver Figura 1.2).

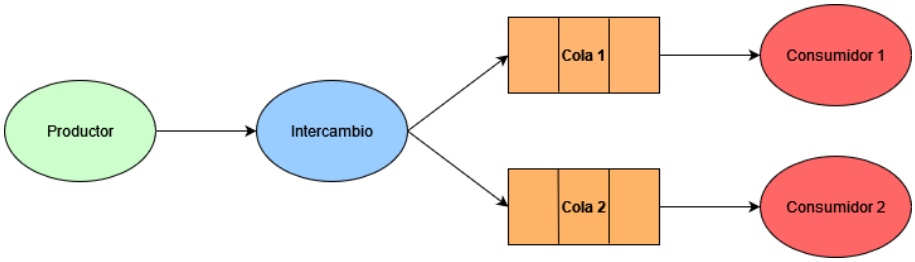

Figura 1.2: Esquema de un sistema de RabbitMQ.

El productor suele ser un sistema o sensor que genera mensajes de forma periódica. Estos mensajes se publican en un "intercambio" definido. Los intercambios reciben mensajes generados por emisores, y se encargan de enviarlos a todas las colas suscritas al intercambio, que actúan como un almacenamiento temporal de los mensajes. Un sistema que actúa como consumidor, se conectará a la cola y recibirá los mensajes acumulados. Si no hay mensajes almacenados, el consumidor pasa a un modo de espera, hasta que reciba nuevos datos y reanude su ejecución [4].

- **Postman**: Es una herramienta para la construcción y la comunicación con API. Se ha utilizado durante el desarrollo del proyecto, para comprobar que la aplicación de monitorización funcionara correctamente. Se utiliza la herramienta para construir peticiones HTTP, que se envían a la aplicación desarrollada durante las prácticas; y se recibe su respuesta, con los datos solicitados. De esta forma no es necesaria la implementación de una interfaz web para comprobar su correcto funcionamiento.

- **Kubernetes**: Plataforma de código abierto que automatiza la gestión y despliegue de contenedores. En este contexto, los contenedores son una tecnología proporcionada por Docker[13] que permite la ejecución de aplicaciones en un entorno aislado del sistema operativo. Por su parte, un *pod* es un conjunto de uno o varios contenedores que se ejecutan en un entorno compartido. Este entorno se encarga de alojar, ejecutar y supervisar los contenedores. La aplicación obtendrá las métricas de los *pods* desplegados, realizando consultas *HTTP* a la *API* de Kubernetes, que devolverá una lista con todos los contenedores desplegados y su estado actual en un mensaje JSON[14] (JavaScript Object Notation), que deberá ser interpretado por la aplicación.

- **MariaDB**: Sistema de gestión de bases de datos derivado de MySQL. Se utiliza para almacenar los datos de las alarmas configuradas y guardar un registro de todas las alarmas que se han activado, con información del motivo. Se escogió por su facilidad de implementación en el proyecto, y porque ofrece altos tiempos de respuesta ante grandes cantidades de datos.

- **JPA** (Java Persistance API): Es una especificación que permite la recuperación de datos de un sistema de almacenamiento no volátil y mantener los datos en el sistema. Se utiliza

[13]Dockers: también conocidos como contenedores, es una máquina virtual ligera que contiene todo lo necesario para ejecutar un software en específico.
[14]JSON: Formato de texto sencillo para la transmisión de datos.

en el proyecto junto con Spring Boot para crear repositorios que generen el código para realizar operaciones en la base de datos de forma automática, durante su ejecución.

- **Node**: Es un entorno de ejecución basado en JavaScript diseñado para la creación de aplicaciones web escalables y asíncronas. Se encarga de cargar y ejecutar las vistas de Angular y todas sus dependencias.

Cabe destacar que Angular y Spring Boot se escogieron porque son las tecnologías usadas en la plataforma de control de la empresa. El proyecto de las prácticas se integrará en la plataforma, así que para facilitar este proceso, se han usado las mismas tecnologías. Para el desarrollo de la aplicación, se ha utilizado IntelliJ IDEA como entorno de desarrollo y GitLab para el control de versiones del proyecto y administración de repositorios

1.5. Estructura de la memoria

En este apartado se detallará el contenido del resto de la memoria, en el que se especifican las distintas etapas del desarrollo del proyecto de prácticas. Los contenidos de los siguientes capítulos son:

- **Capítulo 2, planificación del proyecto:** Este capítulo contiene información sobre la organización del proyecto, metodologías usadas, su evolución a lo largo del tiempo y contratiempos.
- **Capítulo 3, análisis y diseño del sistema:** En este capítulo se detallará el funcionamiento del sistema, la elección del diseño y se analizará su utilidad respecto a otras alternativas.
- **Capítulo 4, implementación y pruebas:** En este capítulo se explicará cómo se ha implementado la aplicación, y qué pruebas se han realizado para comprobar su correcto funcionamiento.
- **Capítulo 5, conclusiones:** En este capítulo se comentará a qué conclusiones se han llegado durante la realización de este proyecto y qué conocimientos se han adquirido durante la estancia en prácticas.

Capítulo 2

Planificación del proyecto

En el capítulo actual se detallará la etapa de planificación del proyecto, se explicará la metodología escogida para su desarrollo, y cómo ha afectado a la asignación y desglose de tareas y seguimiento del progreso. También se mostrarán los cálculos realizados para el análisis de riesgos y costes relacionados con el desarrollo del proyecto.

Además, se hará un análisis de la eficacia de la metodología escogida en el proyecto actual, y cómo contribuye a dividir el proyecto en tareas más manejables, que facilitarán los cálculos de estimaciones y seguimiento del avance durante el desarrollo del proyecto.

2.1. Metodología

Dada la naturaleza del proyecto actual, se ha escogido una metodología Ágil, concretamente una metodología de tipo Scrum. Esta metodología busca dividir el proyecto final en tareas más pequeñas, llamadas esprints. Como el proyecto actual se puede dividir fácilmente en etapas más pequeñas e independientes, se ordenan por etapas que cubran cada una de las funcionalidades necesarias [5]. Cuando se encontraba algún imprevisto que impedía el avance de las tareas relevantes, se trabajaba en objetivos secundarios hasta que se solucionara el problema, así se mantenía un flujo de trabajo constante. No se siguió la metodología Scrum de forma estricta, algunos de los cambios fueron:

- **Reuniones diarias:** Para comprobar el progreso del proyecto, el estado de las tareas asignadas y asegurar que los objetivos se van cumpliendo a un ritmo adecuado, la empresa realiza reuniones diarias de 15 minutos de duración con todas las partes implicadas. Con estas reuniones se pueden detectar problemas en las implementaciones de algunas funcionalidades, reaccionar si alguna parte del proyecto no se está implementado al ritmo adecuado, o cambiar los requisitos del proyecto sin que provoque cambios radicales en las funcionalidades ya implementadas.
- **Reuniones puntuales:** Cada dos semanas se realizaba una reunión individual con el supervisor del proyecto, para informar del progreso realizado, problemas encontrados y

discutir posibles soluciones.

- **Sistema de tickets:** Para organizar el trabajo, la empresa utiliza la herramienta Redmine, que facilita la gestión y seguimiento de proyectos mediante el uso de tickets para dividir y asignar los objetivos entre todas las partes involucradas.

2.2. Planificación temporal del proyecto

Como se ha explicado en el apartado anterior, el proyecto se ha desarrollado siguiendo una metodología ágil. Así que para poder realizar la planificación del proyecto se deben concretar las tareas necesarias que el propietario del producto cree que son necesarias.

2.2.1. Pila del producto

Una pila de producto es una lista ordenada del trabajo que el cliente cree que se necesita desarrollar para desarrollar el producto. En ella se especifican las funcionalidades, mejoras, tecnología y corrección de errores necesarios. La pila de trabajo nunca se termina, está en constante crecimiento y evolución. Se modifica durante el desarrollo del proyecto, adaptándose a los imprevistos y nuevas necesidades [6].

En este apartado se explicará la estructura de la pila inicial del producto, que consistía de 4 esprints. Se dedicó la primera semana a formación sobre las herramientas utilizadas en la empresa, y los últimos 7 días se reservaron como precaución, en caso de que se encontraran problemas que retrasaran el desarrollo de los esprints. Durante el desarrollo, las tareas de los esprints se terminaron antes de lo esperado, por lo que se incluyeron nuevos requisitos que se agruparían en un quinto esprint.

Las pilas de producto se han dividido en tareas, ya que es parecido al sistema de tickets que se utiliza en la empresa. Como no es un proyecto desarrollado para un cliente, no se han basado los esprints en los requisitos del usuario, sino en subtareas. Cada tarea tendrá asignada una puntuación según la importancia y complejidad a la hora de realizarlas.

T01 - Formación en las herramientas usadas para el proyecto

Tareas a realizar: Aprender a utilizar Angular y Spring Boot mediante el uso de tutoriales y el desarrollo de una pequeña aplicación de prueba. Familiarizarse con el lenguaje de programación *InfluxQL*, que se utilizará para realizar las consultas a la base de datos de InfluxDB.

Requisitos: Obtener los conocimientos necesarios para ser capaz de empezar a desarrollar el software.

T02 - Creación del código para leer la base de datos de InfluxDB

Tareas a realizar: Crear un programa en Java que sea capaz de conectarse a la base de datos de InfluxDB, para realizar consultas que permitan recibir y procesar información a un

formato de Java válido para su posterior análisis y envío a la interfaz.

Requisitos: Poder enviar consultas a la base de datos, recibir la respuesta, y almacenarla en un POJO[1] (Plain Old Java Object).

T03 - Diseñar un servicio para obtener el estado de los pods

Tareas a realizar: Añadir al *backend* un servicio que se comunique con la API de Kubernetes para obtener los datos de los *pods* de la empresa. Crear un controlador que, a partir de los datos recogidos, sea capaz de responder a peticiones HTTP para devolver esos datos en formato JSON.

Requisitos:

- El servicio implementado debe ser capaz de comunicarse con la API de Kubernetes para recibir el estado de los *pods*, y transformarlo en una estructura de datos de Java.
- El controlador debe responder ante peticiones HTTP para devolver el estado actual de los *pods* en formato JSON.

T04 - Diseñar un servicio para obtener el estado actual de colas de RabbitMQ

Tareas a realizar: Implementar en el proyecto un servicio que realice las consultas a la base de datos de InfluxDB, para obtener el estado de las colas del servicio de mensajería de RabbitMQ y las transforme en un formato de datos compatible con Java. Se deben obtener el número de mensajes preparados, la velocidad a la que se consumen, y la velocidad a la que se generan. Programar un controlador que utilice el servicio anterior para enviarlos por HTTP cuando reciba una petición.

Requisitos:

- Un servicio que pida datos de las colas de RabbitMQ a la base de datos de InfluxDB y procese la información recibida para almacenarse en un formato compatible con las estructuras de datos de Java.
- El controlador debe responder ante peticiones HTTP para devolver el estado actual de las colas en formato JSON.

T05 - Diseñar el servicio que obtiene las medidas almacenadas en InfluxDB sobre los servidores

Tareas a realizar: Se debe crear un servicio que realice tres consultas a la base de datos de InfluxDB, para obtener las medidas más recientes de CPU, memoria RAM y de la memoria secundaria de los servidores de la empresa. Como InfluxDB solo permite juntar dos tablas por el tiempo, se realizarán las tres consultas por separado para posteriormente unirlas manualmente.

[1] POJO: Identifica las clases que no dependen de *frameworks* especiales.

A continuación se procesan los datos y se ponen a disposición de un controlador, que responderá las peticiones HTTP devolviendo los datos.

Requisitos:

- Un servicio que realice las tres consultas a InfluxDB y procese la información recibida en un único objeto.
- El controlador debe utilizar el servicio anterior para devolver la información de los Servidores en un mensaje HTTP.

T06 - Diseñar las interfaces web para mostrar la información de los sistemas de la empresa

Tareas a realizar: Mediante las herramientas proporcionadas por Angular, se debe crear una aplicación web que se comunique con los servicios anteriores para recibir el estado de los servidores, sistema de RabbitMQ y *pods* de la empresa y mostrar por pantalla su estado actual.

Requisitos:

- Disponer de una interfaz para cada sistema que se pretende monitorizar.
- La página web debe refrescarse automáticamente con los datos más recientes cuando estén disponibles.

T07 - Diseñar el sistema de alarmas

Tareas a realizar: Implementar la funcionalidad de la aplicación web que permita crear, configurar y borrar alarmas. Estas alarmas escanearán los datos recibidos por los servicios y si encuentran algún dispositivo que se encuentra fuera de los parámetros especificados, se activará. Para mantener la aplicación escalable y eficiente, se cambiaron los controladores para que automáticamente consulten cada minuto a los sistemas monitorizados, y almacenen el resultado en una caché. Así se consigue mantener constante el número de accesos a la base de datos sin importar el número de clientes conectados. Los datos recibidos cada minuto, se escanean primero y se envían después a la caché. Este proceso no ralentizará la petición del cliente, ya que siempre tendrá disponibles los datos de la caché.

Requisitos:

- Servicio, controlador e interfaz que permitan crear alarmas, modificarlas, borraras y cambiar si están operativas.
- Una tarea programada debe analizar cada alarma activa, y comprobar si se cumple su condición especificada.
- Se debe disponer de un sistema de caché que se actualice automáticamente, para reducir la carga sobre los servidores.

T08 - Diseñar un sistema que permita mantener un registro de las alarmas activadas

Tareas a realizar: se debe crear un servicio que se ejecute cuando se active una alarma. Ese servicio recogerá toda la información disponible sobre la causa y el sistema, y construirá un mensaje. Ese mensaje se enviará a una cola de mensajes de RabbitMQ, para ser consumida por otro programa que se encargará de manejar los mensajes de alerta; y a una base de datos de MariaDB, que servirá como almacenamiento a largo plazo de los mensajes generados por las alarmas.

Requisitos:

- Se debe configurar una cola de *RabbitMQ* que reciba todos los mensajes de alerta enviados por el servicio anterior.
- Configurar una base de datos de MariaDB para que se almacenen los mensajes de alerta generados por el servicio de alarmas.

T09 - Descargar el proyecto *iotsens-control-platform*, familiarizarse con el entorno y herramientas

Tareas a realizar: Antes de empezar la integración de las funcionalidades del proyecto realizado, se debe aprender como funciona la aplicación web de la empresa, ya que también utiliza Angular, pero tiene una estructura más compleja y robusta. Este entorno contiene funcionalidades como un sistema de sesiones y soporte para múltiples idiomas, por lo que la aplicación actual deberá ser adaptada para utilizarlas.

Requisitos: Obtener los conocimientos necesarios para poder implementar el sistema de monitorización dentro de la plataforma de IoTsens.

T10 - Integración de los servicios de monitorización dentro de la plataforma de control

Tareas a realizar: Se deberá trasladar toda la funcionalidad de aplicación web dentro de la plataforma de control. Para ello se deben implementar las interfaces, métodos y servicios que se conectarán con el micro-servicio de la aplicación original para recoger los datos más recientes emitidos por los sistemas monitorizados y mostrarlos por la aplicación de control.

Requisitos:

- Adaptar el proyecto para que se ejecute de forma paralela al *iotsens-control-platform*.
- Implementar los controladores, modelos y servicios para los servidores, *RabbitMQ* y *pods*. Estos deben ser capaces de comunicarse con los controladores de la aplicación original y poner a disposición los datos recibidos a la vista de la plataforma de la empresa.
- Los datos solo pueden ser accedidos por los usuarios con los permisos necesarios.

- La interfaz debe estar disponible tanto en inglés como español, para eso se debe utilizar la librería de Transloco, disponible para Angular.

T11 - Diseño de las vistas de los sistemas

Tareas a realizar: Se deben crear las vistas correspondientes en la plataforma de IoTsens para mostrar los datos obtenidos por medio de los controladores implementados en la tarea T10. Requisitos:

- Poder crear, borrar y listar alarmas desde la interfaz de *control-platform*.
- Poder activar y desactivar el funcionamiento de una alarma directamente desde la lista de alarmas.
- Poder crear una alarma específica para un componente desde la lista de su recurso.

T12 - Integrar el sistema de alarmas en el *control-platform*.

Tareas a realizar: Se debe integrar la funcionalidad completa de la gestión de alarmas en la plataforma de *control-platform*, se deben poder crear y borrar alarmas sobre los distintos sistemas que se configuraron para monitorizar anteriormente. También se aprovechará para añadir nuevas funcionalidades, como controlar el acceso a las herramientas por medio de inicio de sesión, poder crear alarmas sobre componentes específicos desde la lista de su recurso y poder activar y desactivar las alarmas sin que sea necesario recargar la página.

Requisitos:

- Poder crear, borrar y listar alarmas a través de la interfaz de *control-platform*.
- Poder activar y desactivar el funcionamiento de una alarma directamente desde la lista de alarmas.
- Poder crear una alarma específica para un componente desde la lista de su recurso.

2.2.2. Primer esprint

Para el primer esprint, se quería desarrollar una mínima aplicación funcional. Es decir, desarrollar un programa capaz de conectarse a la base de datos de InfluxDB y que muestre los datos de un sistema en tiempo real. Como la mayoría de datos de los sistemas de la empresa se almacenan en esa base de datos, conseguir el funcionamiento de esa aplicación supondría un gran avance en el proyecto. Por lo tanto, el primer esprint se dividió en las siguientes tareas:

- T01 - Formación en las herramientas usadas para el proyecto.
- T02 - Creación del código para leer la base de datos de InfluxDB.

- T04 - Diseñar un servicio para obtener el estado actual de colas de RabbitMQ.
- T06 - Diseñar las interfaces web para mostrar la información de los sistemas de la empresa.

A continuación se pueden ver las subtareas que forman el primer esprint, y la duración aproximada que se esperaba de cada una:

- Preparar el IDE[2] (Integrated Development Environment) para que incluya las herramientas necesarias para realizar el proyecto. 2 horas
- Realizar el curso de formación de Angular, estudiar el *framework* de Spring Boot que se utilizará en el proyecto. 8 horas
- Configurar las herramientas de *InfluxDB* y *Grafana* para el proyecto: 2 horas
- Estudiar el lenguaje de *InfluxQL* para realizar consultas a la base de datos: 5 horas
- Implementar un programa en Java que pueda realizar consultas a InfluxDB y recoger la información: 12 horas
- Implementar las clases, modelos y métodos necesarios para el funcionamiento de la aplicación de Spring Boot: 8 horas
- Desarrollo de la interfaz del *frontend* y los servicios que se comuniquen con el *backend*. 5 horas

No se incluyeron otras tareas de formación y configuración de las herramientas utilizadas. Para no conectarse directamente al servidor de la empresa durante el primer sprint, se configuró un servicio de *Telegraf* local.

2.3. Seguimiento del proyecto

2.3.1. Esprints

Como se ha seguido una metodología Ágil, se explicará a continuación el desarrollo del proyecto a través de los esprints definidos en la planificación. Se definirán las tareas planificadas, sus subtareas y su implementación. En caso de que hubiera algún contratiempo, también se comentará.

Esprint 1

Las tareas asignadas al esprint actual ya se han mencionado en el apartado anterior, por lo que no se van a incluir en este apartado.

[2]Es una aplicación de software que contiene las herramientas necesarias para el desarrollo de un proyecto de software.

El objetivo del primer esprint era el desarrollo de una aplicación que cumpliera la mínima funcionalidad del proyecto, para utilizarse de base e ir expandiéndola conforme avance el desarrollo del proyecto. Por lo tanto, en este esprint se esperaba desarrollar una aplicación web que mostrara las métricas de las colas de RabbitMQ en tiempo real. Para ello se debe disponer de una aplicación web realizada con Angular, que contiene un servicio encargado de pedir la información al *backend*, que dispondrá de controladores que responderán a las peticiones de los servicios, y se conectarán a la base de datos para realizar consultas y procesar los datos obtenidos.

Las primeras subtareas del esprint avanzaron con lentitud, ya que se tenía que configurar el entorno de trabajo, e instalar todas las herramientas necesarias para su desarrollo. Fue necesaria la formación en los *frameworks* de *Angular* y *Spring Boot*, de los que se siguieron unos cursos de formación en línea. Y también se tuvo que aprender a escribir consultas de *InfluxQL*, y el lenguaje de TypeScript. También fue necesario estudiar la documentación de configuración de los servidores de InfluxDB y *Telegraf* para configurar un servicio local de monitorización.

Se encontraron varios problemas durante el desarrollo. Por temas de compatibilidad, la empresa utiliza una versión desactualizada de InfluxDB, por lo que gran parte de la documentación disponible en línea estaba obsoleta o no era relevante. Fue bastante complicado encontrar una manera de obtener los datos por medio de un programa en Java, ya que la base de datos no proporcionaba soporte para ser utilizada con ese lenguaje. El segundo problema relacionado con InfluxDB es que no funciona como un sistema gestor de base de datos convencional, se especializa en almacenar valores de un dispositivo a lo largo del tiempo, por lo que costó bastante tiempo entender el lenguaje de *InfluxQL* y encontrar la consulta que devolviera el último valor registrado de todos los elementos.

Por último, durante el desarrollo se descubrió que las consultas entre la vista y el programa estaban siendo bloqueadas. Todos los métodos estaban correctamente implementados, pero no se conseguía realizar el intercambio de datos. La causa era que el filtro CORS[3] (*Cross-Origin Resource Sharing*) no estaba configurado. Por motivos de seguridad, se bloquean todas las conexiones AJAX[4] (Asyncronous JavaScript And XML) fuera del origen actual, por lo que se debe configurar manualmente el filtro CORS para especificar las conexiones que se quieren permitir.

Al terminar el esprint, se había conseguido implementar una simple aplicación web que mostrara una lista de colas del servicio de mensajería de la empresa, y el estado de los mensajes acumulados, entrantes y salientes.

Esprint 2

Tareas del esprint:

- T03 - Diseñar un servicio para obtener el estado de los *pods*.
- T05 - Diseñar el servicio que obtiene las medidas almacenadas en InfluxDB sobre los servidores.

[3]CORS: Es una especificación W3C que permite filtrar las conexiones externas.

[4]AJAX: Es una tecnología que permite actualizar el contenido de páginas web sin tener que recargar todo su contenido.

En el segundo esprint se pretendía terminar toda la parte del proyecto relacionada con la recogida de métricas de los dispositivos de la empresa.

Subtareas del esprint:

- Construir un programa capaz de comunicarse con la API de Kubernetes.
- Implementar un servicio que obtenga los datos de InfluxDB sobre los servidores.
- Programar los controladores, modelos y servicios necesarios.
- Expandir la aplicación web con las vistas de los *pods* y los servidores.

Como *Telegraf* no recoge los datos de *pods* de la empresa, se empezó el segundo esprint buscando una manera alternativa de acceder a las métricas. Los administradores podían consultar los datos utilizando Putty[5], pero realizar peticiones SSH[6] (*Secure Shell*) desde un programa de Java resultaba costoso y poco eficaz. No se pudo avanzar en la tarea hasta que se descubrió que Kubernetes dispone de una API a la que se le pueden enviar peticiones HTTP. De la misma forma que los servicios de la aplicación web y la aplicación de monitorización se comunican, también se podían obtener los datos de los *pods*. Aun así, no se conectó la aplicación web directamente a la API de Kubernetes, ya que sería necesario que pasaran por la aplicación de monitorización para cuando se implemente el sistema de alarmas. Se desarrolló la interfaz y los servicios en la aplicación web, y los controladores y métodos encargados de recoger y procesar los datos recibidos de la API. En el esprint anterior se implementó una aplicación básica que recogía los datos almacenados en InfluxDB, por lo que se podía reutilizar gran parte del código, y facilitó su implementación. El único contratiempo significativo fue que la implementación hecha anteriormente solo sirve para una única fuente, y los datos de los servidores se encuentran en tres tablas distintas (Memoria, CPU y Disco). *InfluxQL* no permite realizar consultas sobre varias tablas, por lo que para obtener los datos de los servidores, se necesitaron tres consultas distintas, y luego se procesarían los datos para juntarlas en un único objeto de Java. Aparte de ese problema, el resto de la implementación es idéntica a lo realizado anteriormente, así que se terminó este esprint más rápido de lo esperado.

Al terminar este esprint, ya estaba implementada una aplicación que permitía consultar el estado actual de los servidores, *pods* y colas de *RabbitMQ* de la empresa.

Esprint 3

Tareas del esprint:

- T07 - Diseñar el sistema de alarmas
- T08 - Diseñar un sistema que permita mantener un registro de las alarmas activadas

El objetivo del tercer esprint es implementar un sistema que escanee los datos recogidos por la aplicación de monitorización implementada en el esprint anterior, para poder reaccionar ante

[5] *Putty*: Cliente SSH para la ejecución de comandos en una máquina de forma remota.
[6] SSH: Protocolo de red utilizado en la conexión a máquinas por medio de consola de comandos.

parámetros que se encuentran fuera de los valores esperados especificados en unas alarmas. Esas alarmas deberán ser creadas mediante el uso de la aplicación web.

Subtareas del esprint:

- Crear las tablas necesarias para las alarmas.
- Implementar métodos de Java que permitan realizar operaciones sobre la base de datos.
- Crear las interfaces para gestionar y crear alarmas.
- Programar el servicio que escanea datos y lanza alarmas.
- Crear servicio que envíe registros de alarmas al sistema de mensajería *RabbitMQ* y a una base de datos.

En el esprint actual se pretende terminar la funcionalidad de la aplicación. El objetivo es crear un sistema de alarmas configurables que envíen mensajes de alerta cuando se detecten valores en los sistemas monitorizados fuera de los parámetros establecidos.

Se había planteado implementar el sistema de alarmas como un proceso que funciona de forma paralela al sistema de monitorización, pero pronto se encontraron problemas a este enfoque. De la forma en la que estaba programada la aplicación, por cada cliente conectado, se hacía una petición al servidor, y el sistema de alarmas actuaba como otro cliente más, haciendo peticiones de los datos a la aplicación para escanearlos. Esta forma no era nada escalable u óptima, y aparte, se encontraron problemas para implementarla, ya que el *framework* Spring Boot no está diseñado para ejecutarse en un subproceso. Por lo tanto, se tuvo que rediseñar la estructura de la aplicación y plantear un nuevo enfoque.

La alternativa propuesta fue implementar el sistema de alarmas durante el proceso de petición de datos por parte del cliente, la intención era detectar si hay parámetros que activan alarmas antes de enviarlos a la vista, entre la capa del servicio y del controlador. Esta solución facilita la implementación, pero no soluciona los problemas de escalabilidad y rendimiento cuando hay un gran número de usuarios conectados. Este problema se solucionó mediante la implementación de un sistema de Caché, que se explicará con más detalle en el apartado 5.3.2.

Para proseguir con la nueva implementación, se tuvieron que reescribir métodos ya implementados en otros esprints. Pero aun así, las subtareas se implementaron sin muchos más problemas y se terminaron en el tiempo previsto, sin ningún otro contratiempo.

Esprint 4

Tareas del esprint:

- T09 - Descargar el proyecto *iotsens-control-platform*, familiarizarse con el entorno y herramientas
- T10 - Integración de los servicios de monitorización dentro de la plataforma de control

- T11 - Diseño de las vistas de los sistemas
- T12 - Integrar el sistema de alarmas en el control-platform

Subtareas del esprint:

- Descargar el proyecto de *iotsens-control-platform* y estudiar su estructura.
- Desarrollar las interfaces y los servicios del *frontend* de la plataforma.
- Implementar los métodos de controladores y servicios que se conectarán a la aplicación desarrollada.
- Adaptar el proyecto para que funcione con los métodos de la plataforma de *IoTsens*.
- Realizar varias mejoras.

Este esprint se desarrolló sin ninguna complicación notable. Se dedicó bastante tiempo a entender como funcionaba la plataforma y como añadir la funcionalidad de mi aplicación web dentro de la plataforma web. La implementación de una interfaz funcional para monitorizar los *pods* fue lenta, pero el resto de interfaces y servicios correspondientes se implementaron más rápidamente, ya que el código seguía la misma estructura. No se pudo reutilizar la interfaz ya desarrollada en la aplicación web porque había que mantener un estilo coherente con el resto de la plataforma de control y adherirse a las nuevas reglas de estilo especificadas. También se añadieron funcionalidades extra que requerían de modificar el código original del programa de monitorización, como poder crear alarmas sobre un elemento específico, poder activarlas o desactivarlas desde la lista de alarmas o que se mostrara la fecha de creación de una alarma.

2.3.2. Pila de producto final

El tiempo asignado fue más que suficiente para el desarrollo de la aplicación, por lo que se propusieron tareas adicionales que no estaban previstas en la planificación inicial, para mejorar la funcionalidad del proyecto. Incluso con la inclusión de las mejoras, se finalizó el desarrollo del proyecto mucho antes de lo esperado, y no se tuvo que recurrir a los días de margen que se asignaron durante la planificación.

El tiempo restante de la estancia en prácticas se dedicó a un nuevo proyecto. Este consistía en el desarrollo de un sistema de decodificación para el protocolo de mensajes Modbus. Este decodificador recibe los mensajes emitidos por un sensor, y los transforma a un formato que pueda ser interpretado por la plataforma de control. Esta implementación se explica con más detalle en el Anexo A.

2.3.3. Diagrama BurnDown

El proyecto se desarrolló dentro del tiempo previsto. Se había asignado tiempo suficiente para lidiar con posibles imprevistos. Se comprobaba la correcta funcionalidad de la aplicación

a lo largo del desarrollo del proyecto. Cuando se terminó la implementación, no se encontraron problemas graves de funcionamiento. Algunas etapas del desarrollo del proyecto trascurrieron de forma más lenta a la planificada, sobre todo las tareas de formación y las implementaciones iniciales. Mientras que las herramientas proporcionadas por el *framework* de Spring Boot facilitaron la implementación y redujeron el coste del desarrollo.

El tiempo restante de la estancia de prácticas se utilizó para implementar el decodificador del protocolo Modbus. Se entra en más detalle sobre su implementación en el Anexo A.

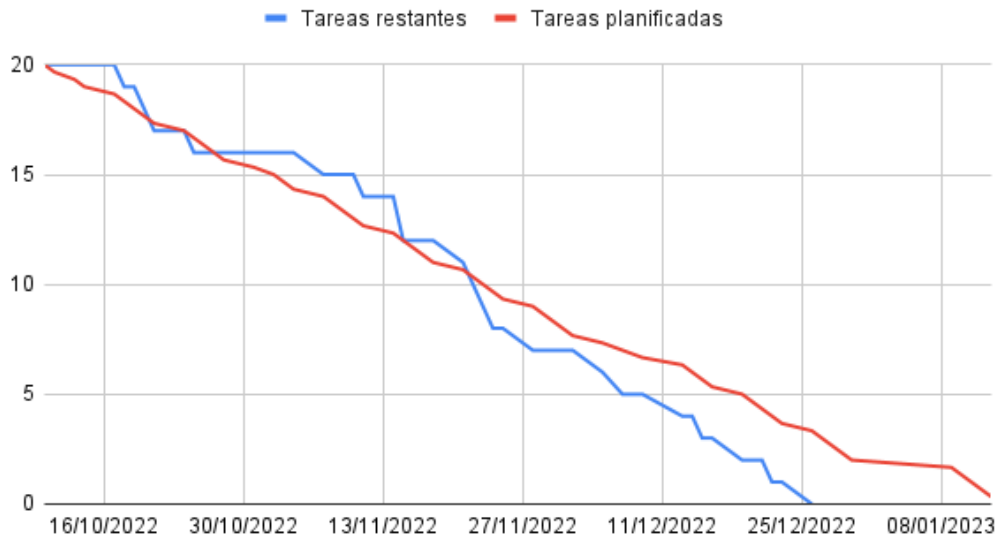

Figura 2.1: Representación de las tareas planificadas y realizadas.

2.3.4. Costes temporales

A continuación se mostrarán los costes temporales asociados al proyecto. En la Tabla 2.2 se muestran los costes temporales asociados inicialmente al desarrollo del proyecto. Se pueden ver los costes de documentación y presentación del proyecto en la Tabla 2.3.

2.3.5. Calendario

Para alcanzar las trescientas horas asignadas al desarrollo del proyecto, se dedicaron cinco horas cada día por parte del alumno, para su desarrollo. Se empezó la estancia de prácticas el día diez de octubre, y se completó el trece de enero. A continuación se muestra un resumen de la distribución de las horas agrupado por semanas.

Octubre: 90 horas

- Semana del 10 al 14: 20 horas
- Semana del 17 al 21: 25 horas

Tareas	Tareas predecesoras	Tiempo (h)
1. Trabajo Final de Grado		**300**
1.1 Propuesta técnica		**5**
1.1.1 Documentación acerca del ámbito del proyecto		1
1.1.2 Definición del proyecto	1.1.1	2
1.1.3 Definición del alcance y objetivos	1.1.2	2
1.2 Formación		**30**
1.2.1 Formación de las herramientas a utilizar	1.1	15
1.2.2 Formación de las tecnologías a utilizar	1.2.1	15
1.3 Desarrollo e implementación		**265**
1.3.1 Implementación		257
1.3.1.1 Lectura de la API de InfluxDB para acceder al estado de colas	1.2	15
1.3.1.2 Implementación de métodos auxiliares de Spring Boot	1.3.1.1	72
1.3.1.3 Visualización de los datos en formato tabla	1.3.1.2	10
1.3.1.4 Lectura de la API de Kubernetes para acceder al estado de Pods	1.3.1.3	10
1.3.1.5 Lectura de la API de InfluxDB para acceder al estado de servidores	1.3.1.4	15
1.3.1.6 Implementación del sistema de alarmas	1.3.1.5	30
1.3.1.7 Interfaces para gestionar alarmas	1.3.1.6	15
1.3.1.8 Registro de excepciones	1.3.1.7	15
1.3.1.9 Integraciones en la plataforma de control	1.3.1.8	75
1.3.2 Verificación y validación		8
1.3.2.1 Pruebas de funcionamiento	1.3.1	4
1.3.2.2 Pruebas de integración sobre la plataforma de control	1.3.2.1	4
1.4 Puesta en marcha	**0**	**0**
1.4.1 Entrega del producto	0	0

Figura 2.2: Definición de las tareas del proyecto y su coste temporal (en horas).

Tareas	Tareas predecesoras	Tiempo (h)
1.5 Redacción y presentación del TFG		**135**
1.5.1 Redacción de informes quincenales		12
1.5.2 Tutorias		15
1.5.3 Redacción de la memoria técnica	1.5.1	92
1.5.4 Preparación de la presentación	1.5.3	15
1.5.5 Presentación del proyecto	1.5.4	1

Figura 2.3: Tareas de documentación y presentación (en horas).

- Semana del 24 al 28: 25 horas
- Semana del 31 al 04: 20 horas

Noviembre: 100 horas

- Semana del 07 al 11: 25 horas
- Semana del 14 al 18: 25 horas
- Semana del 21 al 25: 25 horas

- Semana del 28 al 02: 25 horas

Diciembre: 85 horas

- Semana del 05 al 09: 15 horas
- Semana del 13 al 16: 20 horas
- Semana del 19 al 23: 25 horas
- Semana del 26 al 30: 25 horas

Enero: 25 horas

- Semana del 09 al 13: 25 horas

2.4. Estimación de los recursos y costes

En este apartado se describirán los costes aproximados relacionados con el desarrollo del proyecto y los recursos utilizados.

2.4.1. Recursos tecnológicos

En cuanto los costes relacionados con el software, la gran mayoría era gratuito y de código abierto, exceptuando las licencia de IntelliJ y PrimeNG. En cuanto a los costes relacionados con el hardware, todos los dispositivos han sido proporcionados por la empresa. Se proporcionó un ordenador de oficina, con los periféricos necesarios, y también se utilizaron los servidores que se deben monitorizar, que alojan las bases de datos de la empresa, los servicios de mensajería, y almacenan los repositorios y proyectos de la empresa.

2.4.2. Recursos humanos

Para desarrollar el proyecto, se formó un equipo compuesto por un programador junior, y un supervisor de la empresa, que también ejercía las funciones de jefe de proyecto. No se ha requerido a nadie más en el proyecto, ya que el proyecto utiliza en su gran mayoría herramientas que ya estaban implementadas antes del comienzo del desarrollo del proyecto, por lo que no se incluirán.

2.4.3. Costes totales

Los costes totales del proyecto se calculan como la suma de los costes de recursos tecnológicos y los recursos humanos que se han comentado previamente. En cuanto a los recursos humanos,

se deben calcular los costes del sueldo del programador júnior y del jefe del proyecto. En la Tabla 2.1 se muestran los valores totales:

Rol	Horas diarias	Sueldo/hora (€)	Coste total (€)
Programador júnior	5	11,45	3.435,00
Jefe de proyecto	0,5	17,49	524,70

Tabla 2.1: Costes relacionados con recursos humanos.

Los salarios se han obtenido consultando la página de Indeed, una plataforma que permite la búsqueda y comparación de ofertas de trabajo, y ofrece estadísticas de los salarios medios para un puesto y una ubicación especificados [7]. Los costes de recursos humanos han sido calculados basándose en las horas diarias dedicadas durante el proyecto.

Para calcular los costes de recursos tecnológicos, se deben calcular los gastos relacionados con hardware y software. Aunque la empresa ya disponía de estos recursos de antemano y no hizo falta ninguna inversión para la realización del proyecto, se van a realizar los cálculos como si no hubiera sido así (ver Tabla 2.2).

Hardware/Software	Coste total (€)
HP PRODESK 400 G5 SFF 5ZS17EA	399,00
HP ProDisplay P222va x2	89x2 = 178,00
Servidor	1200
IntelliJ Idea Ultimate	499

Tabla 2.2: Costes de los recursos de hardware y software utilizados.

El coste total de los recursos se puede ver a continuación, en la Tabla 2.3. A los costes humanos se le deben añadir los costes de contratación y seguridad social, que supone aproximadamente un 20 % del coste original.

Recurso	Coste (€)
Recursos humanos	3.959,70
Recursos tecnológicos	2.276,00
Total	6.235,70

Tabla 2.3: Costes totales de los recursos humanos y tecnológicos.

2.5. Riesgos

En esta sección se describirán los posibles riesgos que pueden aparecer durante el desarrollo del proyecto. Al ser un equipo pequeño que solo contiene un programador, los riesgos del proyecto dependen principalmente del rendimiento de este. También significa que no habrá riesgos relacionados con la asignación de tareas o por factores externos, ya que todos los sistemas a monitorizar funcionaban correctamente al empezar el proyecto, y la implementación del proyecto depende principalmente de una única persona.

Para realizar el análisis de riesgos, se han listado todos los riesgos identificados y su tipo (ver Tabla 2.4). Los generales pueden afectar al proyecto en cualquier momento de su desarrollo, y los específicos se asocian a problemas puntuales. En la Tabla 2.5 se identifican cuáles son las causas que podrían desencadenar los riesgos mencionados anteriormente, y en la Tabla 2.6 se describen qué medidas de prevención y contingencia se han adoptado para los riesgos mencionados en las tablas anteriores.

Código	**Descripción**	**Tipo de riesgo**
R01	Falta de experiencia para utilizar las tecnologías del proyecto	General
R02	Retrasos en el desarrollo de las tareas	General
R03	Requisitos del proyecto no cumplidos	General
R04	Lectura de datos monitorizados obsoletos o incorrectos	Específica
R05	Incorrecta activación de las alarmas	Específica

Tabla 2.4: Lista de los riesgos identificados.

R01 - Falta de experiencia
Magnitud: Aumenta su relevancia conforme avanza el proyecto. **Descripción:** El desarrollador carece de la experiencia necesaria para continuar con el desarrollo del proyecto. **Impacto:** Puede provocar retrasos en el desarrollo del proyecto. **Indicadores:** Lentitud en el desarrollo de tareas, incumplimiento de la planificación.
R02 - Retrasos en el desarrollo de las tareas
Magnitud: Aumenta su relevancia conforme avanza el proyecto. **Descripción:** No se completan las tareas de los esprints en el tiempo asignado. **Impacto:** Puede provocar retrasos en el desarrollo del proyecto y modificaciones en la planificación. **Indicadores:** Se dedica más tiempo de lo esperado a las tareas, las tareas son más complejas de lo previsto, o no se terminan en el tiempo planificado.
R03 - Requisitos del proyecto no cumplidos
Magnitud: Alta, la aplicación funciona incorrectamente. **Descripción:** Se activan alarmas en situaciones para las que no han sido configuradas. **Impacto:** Puede provocar retrasos en el desarrollo, o sobrecargar el sistema de mensajería y gestión de alarmas. **Indicadores:** Las alarmas se activan aunque no se cumpla su condición especificada, o no se activan para el caso en el que han sido programadas.
R04 - Lectura de datos monitorizados obsoletos o incorrectos
Magnitud: Alta, la aplicación funciona incorrectamente. **Descripción:** La aplicación muestra información por pantalla que no es representativa del estado actual de los sistemas. **Impacto:** Puede provocar retrasos en el desarrollo. **Indicadores:** Se muestran datos que no tienen sentido, o se activan alarmas aunque no haya ningún problema con los sistemas.
R05 - Incorrecta activación de las alarmas
Magnitud: Alta, la aplicación funciona incorrectamente. **Descripción:** Se activan alarmas en situaciones para las que no han sido configuradas. **Impacto:** Puede provocar retrasos en el desarrollo, o sobrecargar el sistema de mensajería y gestión de alarmas. **Indicadores:** Las alarmas se activan aunque no se cumpla su condición especificada, o no se activan para el caso en el que han sido programadas.

Tabla 2.5: Análisis de riesgos.

Código	Plan de prevención	Plan de contingencia
R01	Dedicar tiempo a formar al programador en las herramientas utilizadas.	Dedicar tiempo a formación hasta que el programador sea más capaz, y asignar más tiempo a las primeras tareas.
R02	Añadir tiempo de margen a las tareas durante la planificación.	Utilizar el tiempo de margen asignado para las tareas que no se completen a tiempo. Y si fuera necesario, eliminar tareas secundarias o mejoras que no formen parte de los requisitos del producto.
R03	Acordar con el cliente los requisitos de la aplicación.	Dedicar más tiempo al desarrollo de la aplicación o volver a negociar los requisitos.
R04	Realización de tests y pruebas automáticas para verificar la veracidad de los datos.	Encontrar la causa, arreglarla, y asegurarse que no se pueda repetir.
R05	Realización de tests en un entorno de pruebas, simulando todos los casos y escenarios.	Encontrar el origen del problema, y solucionarlo.

Tabla 2.6: Planes de prevención y contingencia.

Capítulo 3

Análisis del sistema

En este capítulo se explicará en detalle una de las primeras fases del desarrollo de software, la definición de los requisitos. En esta fase se realiza un análisis detallado de los requisitos del sistema a desarrollar. Estos requisitos forman una parte fundamental del desarrollo, ya que afectarán a su planificación e implementación. Los requisitos crean un compromiso entre el cliente y la empresa, cuando se hayan cumplido todos los requisitos especificados, se considerará que el proyecto ha terminado.

Los requisitos describen la funcionalidad que debe proporcionar el proyecto final. Como se ha seguido una metodología Ágil, se han definido agrupándolos en conjuntos de funcionalidades independientes, para facilitar la planificación y desarrollo.

3.1. Definición de requisitos

En la planificación temporal del proyecto se detallaron las tareas y requisitos que se deben satisfacer en cada una de las etapas para considerar el proyecto como terminado.

3.1.1. Historias de usuario

En esta sección se especificarán los requisitos únicamente desde el punto de vista del usuario final. Como el proyecto ha utilizado una metodología Ágil para su desarrollo, se utilizarán historias de usuario (HU) para especificar los requisitos finales de proyecto.

Las historias de usuario no se han tenido en cuenta en la definición de tareas de la planificación temporal, ya que la interacción con el usuario final ha sido mínima, y no deberían afectar al desarrollo de las distintas funcionalidades del proyecto, sino que debería ser simplemente orientativo para los componentes del proyecto con los que no interactúa el usuario final. A continuación se mostrarán todas las historias de usuario que se deben implementar para completar la funcionalidad requerida del proyecto.

Épica 1

- HU01: Como usuario del producto, quiero poder consultar el estado actual de las colas de mensajería.
- HU02: Como usuario del producto, quiero poder ver una lista de los *pods* de la empresa, con su estado actual.
- HU03: Como usuario del producto, quiero poder consultar el estado actual de los servidores de la empresa.
- HU04: Como usuario del producto, quiero disponer de una plataforma donde poder consultar la información anterior de forma cómoda y rápida.

Épica 2

- HU05: Como usuario del producto, quiero poder crear, borrar, activar y desactivar alarmas sobre los elementos monitorizados.
- HU06: Como usuario del producto, quiero poder configurar las alarmas, para que se activen cuando se cumplan los parámetros especificados.
- HU07: Como usuario del producto, quiero disponer de un registro de todas las veces que las alarmas han lanzado una excepción, y su motivo.
- HU08: Como usuario del producto, quiero que el servicio de alarmas lance excepciones cuando se cumplan las condiciones especificadas.

Épica 3

- HU09: Como usuario del producto, quiero disponer de la funcionalidad en la plataforma de control de la empresa.
- HU10: Como usuario del producto, quiero tener la opción de aplicar alarmas a sistemas generales o específicos.
- HU11: Como usuario del producto, quiero restringir el acceso al sistema de alarmas para ciertos usuarios.

3.1.2. Diagrama de casos de uso

El diagrama de casos de uso es una herramienta utilizada para representar procesos y sistemas, desde la que se puede ver sus relaciones y como interactúan entre ellos (ver Figura 3.1).

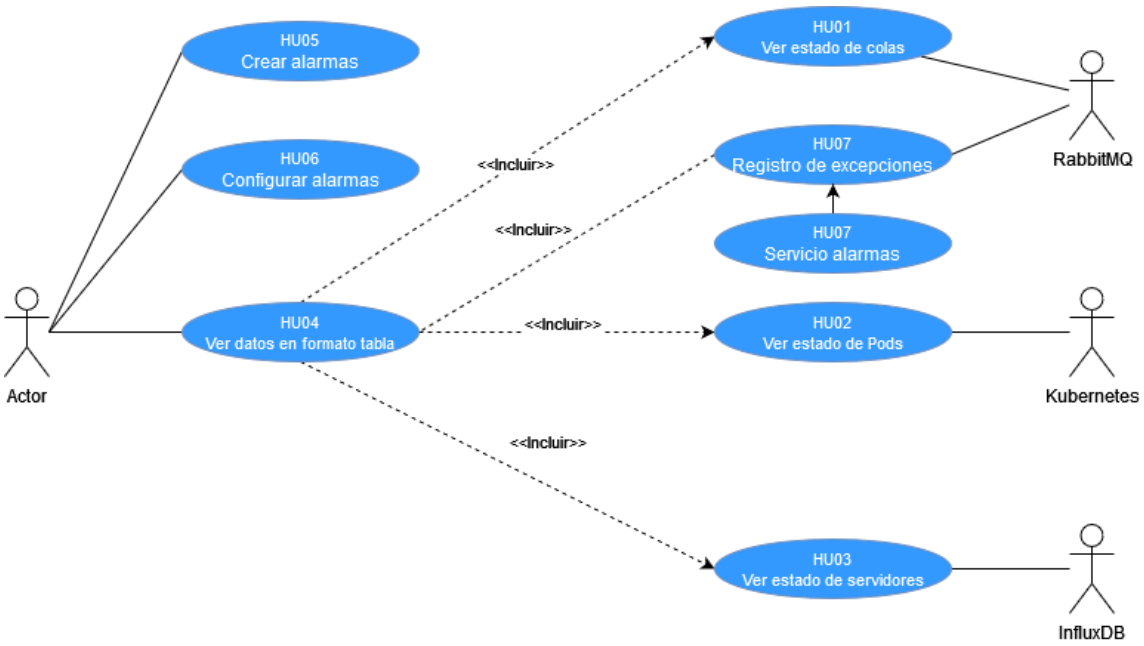

Figura 3.1: Diagrama de casos de uso.

3.2. Análisis de requisitos

3.2.1. Componentes del sistema

En el apartado actual se van a explicar las funciones de cada uno de los componentes principales del sistema. No se van a analizar todos los componentes que forman parte del proyecto, sino los más relevantes para cumplir con los requisitos especificados por el cliente.

El producto debe cumplir unos requisitos específicos. Debe ser una aplicación web que sea capaz de comunicarse con los sistemas de la empresa para obtener su estado, mostrar las métricas de los sistemas al usuario final en formato de tablas o listas, y además poder configurar alarmas sobre las métricas para detectar posibles errores de funcionamiento automáticamente. Por último, la aplicación se incluirá en la plataforma de control de IoTsens, por lo que se integrará su funcionalidad en la plataforma.

Para su funcionamiento, la aplicación utiliza una base de datos propia, que se usará para almacenar los datos de las alarmas creadas, y un registro a largo plazo de las excepciones que se han generado por las alarmas activas. Esta ha sido implementada utilizado MariaDB, por su velocidad y escalabilidad. Para realizar las labores de monitorización, la aplicación también necesita obtener datos de fuentes externas, estos datos son:

- Sistemas y servidores: Se utiliza Telegraf para recoger las métricas en los sistemas, y se almacenan en una base de datos de InfluxDB. Para obtener esos datos se ha utilizado la librería oficial de InfluxDB para Java, que permite crear una conexión a la base de datos y realizar consultas sobre ella. Debido a que solo se ha utilizado para la ejecución de las consultas, y no se ha programado su funcionamiento, no se entrará en mucho detalle sobre

la librería.

- Pods: El servidor de Kubernetes dispone de una API desde la que se pueden realizar peticiones por HTTP, y obtener los datos de los *pods* del sistema. Para ello se ha desarrollado un programa que pueda enviar esas peticiones, recibir la respuesta en formato JSON, y procesar su contenido para obtener la información relevante.
- RabbitMQ: El servicio de mensajería de RabbitMQ dispone de una API, pero su funcionalidad es limitada. Su principal uso es el envío y obtención mensajes. Y aunque se puede usar para obtener el estado de una sola cola, utilizar la API para obtener el estado de todas las colas del sistema suponía una implementación más compleja. Las métricas de RabbitMQ también se encuentran en la base de datos de InfluxDB. Se podía reutilizar gran parte del código ya desarrollado, y además ofrece mayor libertad a la hora de elegir qué datos recibir. Se eligió esta implementación, ya que también ofrece la posibilidad de modificar la consulta en un futuro, y ajustarla a nuevos requisitos.

El proyecto consta de dos partes. La aplicación de Spring Boot actúa como backend. Realiza las peticiones y consultas para recibir los datos de sistemas mencionados anteriormente, se procesan para poder almacenarse en objetos de Java, y se ponen a disposición de controladores, que responden a peticiones HTTP para enviar la información recibida de los sistemas. Por otro lado, la aplicación web desarrollada con Angular, es la encargada de proporcionar la interfaz en la que mostrar los datos recibidos de forma gráfica, con el uso de tablas y listas, y de realizar las peticiones al backend, la aplicación de Spring Boot. Cuando se termine el desarrollo de las funcionalidades del proyecto, se empezará el proceso de integración de los componentes de la aplicación web dentro de la plataforma de control de la empresa.

En el diagrama de la Figura 3.2 se puede visualizar la interacción entre los componentes de las dos aplicaciones. El flujo de información es bidireccional en todas las operaciones, ya que se utiliza un sistema de peticiones y respuesta. Un detalle del diagrama es que se puede apreciar como la aplicación de Control Platform se conecta a la de monitorización mediante el backend, y no se encuentra la aplicación realizada con Angular. La razón es porque en el proceso de integración, se reemplazó la aplicación web de angular por la plataforma de control. Esa plataforma ya consta de un backend, que recibe los datos enviados por la aplicación del proyecto, y los procesa en un formato compatible para la aplicación web. Actúa como intermediario, filtrando la información y permitiendo la funcionalidad de filtrar la información, o especificar el orden de los datos entrantes.

3.2.2. Requisitos de datos

En el siguiente apartado se describirán las estructuras de datos implementadas en el sistema, que se usan en el proyecto final. En la tabla 3.1, se pueden ver los requisitos de datos junto con su función, estructura, y el autor que realizó la propuesta.

Sistema	**Autor**: Programador júnior. **Parámetros**: Nombre del sistema, fecha de la medida, nivel de consumo de CPU por parte del usuario y sistema, nivel de memoria RAM disponible, y memoria secundaria disponible. **Descripción**: Es la información que se quiere monitorizar sobre los servidores y sistemas de la empresa. Se utilizarán estos datos para almacenar la medida más reciente del sistema y mostrarla por la aplicación web.
Pod	**Autor**: Programador júnior. **Parámetros**: Nombre, estado, reinicios, dirección IP, nombre del nodo al que pertenece. **Descripción**: Contiene la información que se quiere monitorizar de los *pods* de Kubernetes, la mayoría de parámetros son de información. Los *pods* están configurados por defecto para que se reinicien si se ha interrumpido su ejecución, por lo que los parámetros más relevantes para la monitorización son el número de reinicios totales, y el estado del *pod*.
Rabbit	**Autor**: Programador júnior. **Parámetros**: Fecha de la última medida, nombre de la cola, número de mensajes almacenados, tasa de consumo de mensajes, tasa de llegada de mensajes. **Descripción**: Almacena los datos de la última métrica disponible de las colas del servicio de mensajería. Es necesario poder consultar los mensajes acumulados, y la tasa de consumo y producción de estos, para poder detectar problemas de funcionamiento.
Alarma	**Autor**: Jefe de proyecto. **Parámetros**: Identificador único, sobre qué sistema será eficaz la alarma, el recurso específico de un sistema sobre el que se quiere aplicar la alarma, la medida del recurso que se quiere monitorizar, la condición, el valor que servirá para decidir si lanzar la excepción, y un indicador que muestra si la alarma está activada. **Descripción**: Almacenará los datos de las alarmas configuradas, el campo de recurso estará vacío si es una alarma que se aplica sobre todos los recursos, o contendrá el nombre del recurso en específico que la alarma debe monitorizar.
Registro de alarma	**Autor**: Programador júnior. **Parámetros**: Fecha en la que se generó la excepción, identificador de la alarma que ha creado la excepción, origen, nombre del dispositivo, la condición que se había configurado, y el valor de la medida del dispositivo que ha causado que la alarma se active. **Descripción**: Cuando se active la alarma, se generará un informe con toda la información correspondiente, esos datos se almacenarán en una base de datos, y se podrán consultar desde la aplicación web.

Tabla 3.1: Tabla de requisitos de datos.

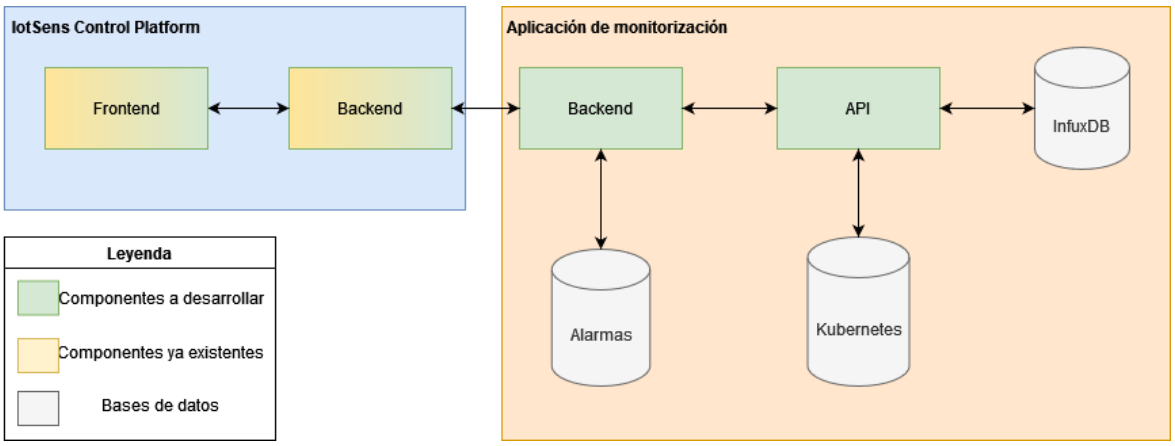

Figura 3.2: Esquema del flujo de información del proyecto.

3.2.3. Diagrama de clases

En el diagrama de clases se pueden ver las relaciones entre las entidades y atributos de los requisitos de datos del proyecto explicados en el apartado anterior (ver Figura 3.3).

Es un modelo simple, las clases que almacenan los datos de sistemas, colas de RabbitMQ y de *pods* son independientes, solo se utilizan para la transferencia de datos entre las vistas y el programa. Una alarma (Trigger) puede existir por sí sola, pero una excepción (TriggerLog) debe haber sido lanzada por una alarma. Y los recursos que monitoriza la alarma y la condición, que puede cambiar en un futuro, se definen mediante las clases de "Origin" y "Cond", que son de tipo enumeración.

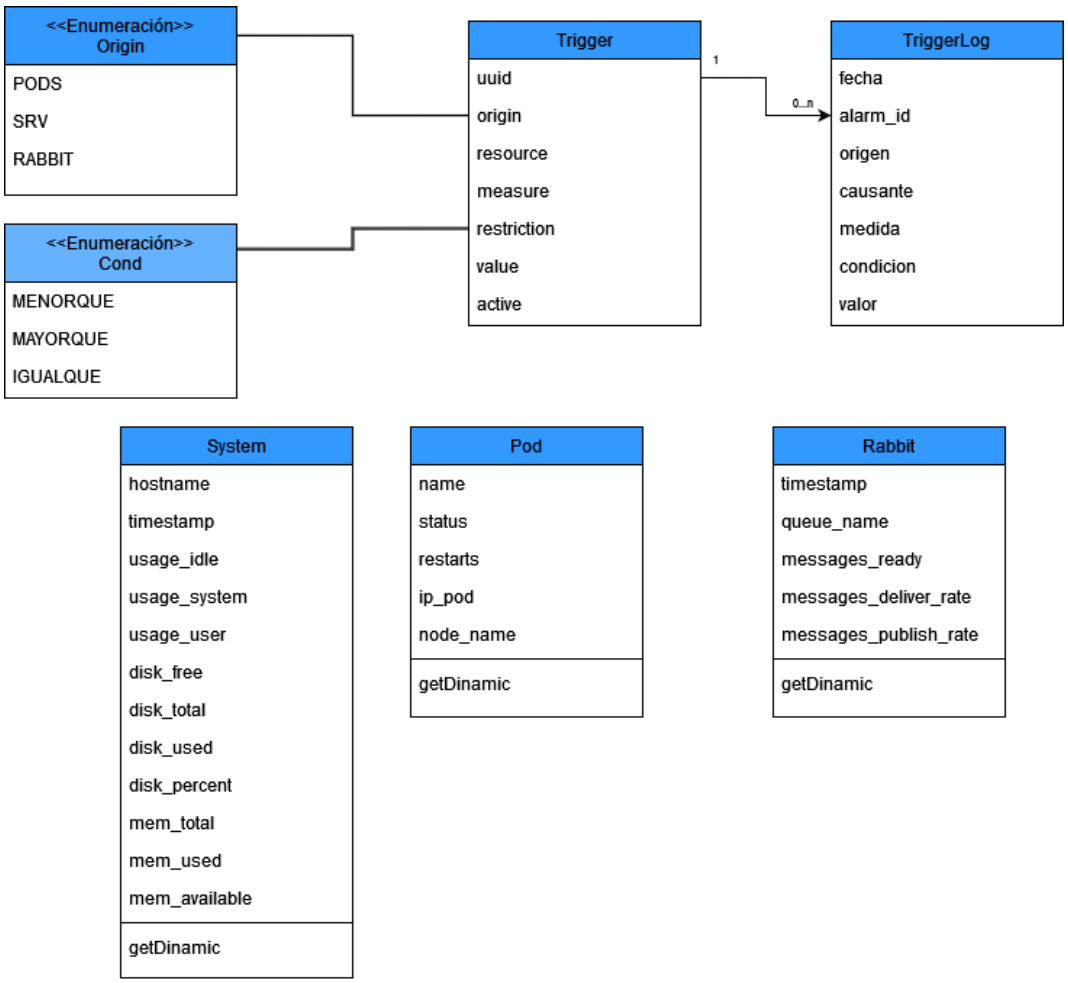

Figura 3.3: Diagrama de clases y atributos.

Capítulo 4

Diseño del sistema

4.1. Diseño de la base de datos

Respecto al diseño de la base de datos, el sistema de monitorización de Kubernetes y la base de datos de InfluxDB ya estaban implementadas previo al inicio de la estancia de prácticas, por lo que solo se utilizaron para realizar consultas de lectura de los datos sobre los sistemas a monitorizar. Para almacenar los datos de las alarmas configuradas y el registro de excepciones generadas, se implementó una base de datos en MariaDB. Como se implementó el patrón de diseño de DAO (Data Access Object), se pueden utilizar las clases de Repository para obtener los datos de la base de datos sin necesidad de escribir consultas. La estructura de las tablas de la base de datos de MariaDB se puede visualizar en la figura 4.1.

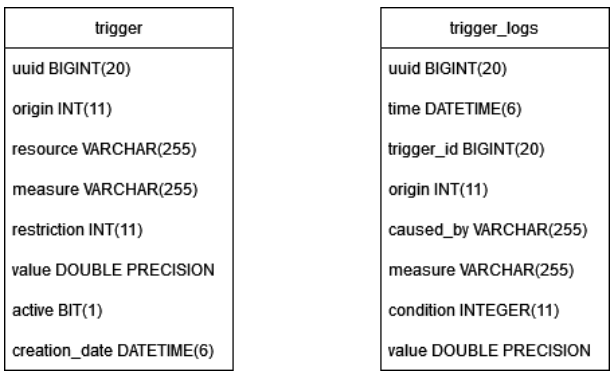

Figura 4.1: Estructura de la base de datos.

4.2. Diseño de software

Los patrones de diseño describen como solucionar problemas comunes de software. Los patrones de software proporcionan estándares que permiten escribir un código robusto y reutilizable, y estructurado [8]. Algunos de sus beneficios son:

- Proporcionan soluciones estandarizadas a problemas recurrentes. Con ello, se consigue reducir el tiempo dedicado a la implementación.
- Proporcionan reusabilidad. Los patrones de diseño se suelen centrar en el desarrollo de componentes independientes y escalables. Esta propiedad también facilita su mantenimiento y mejora su robustez.
- Como los patrones de diseño ya están definidos de antemano, facilita la comprensión y mantenimiento del código. Una persona que no haya trabajado en el proyecto, pero sea familiar con el protocolo, podrá entender el código con mayor facilidad.

Durante el desarrollo del proyecto se han utilizado múltiples patrones y estándares. A continuación se explicarán los patrones de diseño que tuvieron más relevancia durante el desarrollo del proyecto.

4.2.1. Patrón MVC

El patrón de diseño Modelo Vista Controlador (MVC) especifica que una aplicación debe estar formada por un modelo de datos, una vista para presentar información, y el código de control para modificar los datos, y que cada una de las capas esté separada en componentes distintos [9] (ver Figura 4.2). MVC es un patrón de arquitectura, pero no define todas las capas de la aplicación. Se define en detalle cómo se deben implementar los componentes de la capa de interacción/interfaz, pero el patrón no detalla cómo se debe implementar la capa de negocio o la capa de acceso a datos [10].

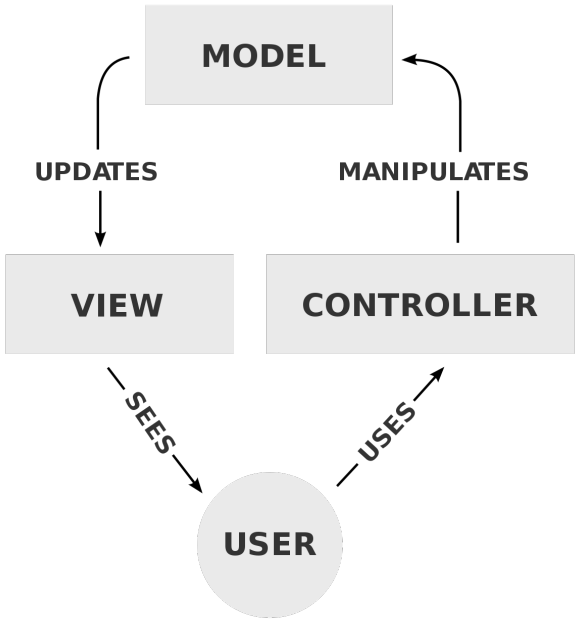

Figura 4.2: Diagrama de interacciones del patrón MVC.

El patrón MVC ofrece una estructura para la arquitectura de la aplicación, pero no para todas sus capas. Se centra en la capa de interfaz e interacción con el usuario. No proporciona

lógica de negocio, servicio o control de acceso a datos. El patrón MVC está dividido en tres componentes:

- Modelo: Especifica el contenido y formato de los datos, el estado.
- Vista: Se encarga de recoger la información y mostrarla al usuario, pero no puede transformarla ni manipularla.
- Controlador: Actúa de intermediario entre el modelo y la vista. Gestiona la lógica de la aplicación. Escucha eventos generados por la vista (u otra fuente externa) y ejecuta la acción correspondiente. Suele utilizarse para llamar a métodos del modelo con el objetivo de obtener unos datos.

La aplicación de monitorización desarrollada durante la estancia de prácticas utiliza este protocolo. El *framework* de Spring Boot facilita la implementación de este patrón. Proporciona etiquetas para métodos que especifican su comportamiento.

La capa de la vista ha sido reemplazada por una aplicación de Angular separada, que se comunica con el controlador. El controlador se ha configurado para que responda ante eventos en forma de peticiones HTTP, que cuando se activen devolverán los datos correspondientes en el formato especificado en el Modelo.

4.2.2. Patrón DAO

El patrón de diseño DAO (Data Access Object) se encarga de gestionar la persistencia de los objetos, es decir, la capacidad de almacenar y recuperar datos de un sistema de almacenamiento no volátil. Este patrón permite separar la capa de negocio de la capa de aplicación, mediante el uso de una API abstracta. De esta forma, se pueden desarrollar ambas capas de forma paralela e independizar la aplicación del mecanismo de persistencia utilizado.

Los métodos proporcionados por un objeto DAO depende de las necesidades de cada aplicación. Al ser un protocolo cercano a la capa de almacenamiento, los métodos de los DAO se corresponden con los elementos de la base de datos a la que referencia. Normalmente, se utiliza para definir los métodos que permiten realizar operaciones sobre el medio de almacenamiento, definiendo métodos para recuperar objetos mediante un identificador, almacenar valores nuevos, sin necesidad de definir consultas al sistema de almacenamiento.

En el proyecto se ha utilizado el patrón DAO junto con el patrón Repository, proporcionado por Spring Boot. Ambos patrones proporcionan una funcionalidad similar.

El Repository también se encarga de proporcionar las herramientas para la obtención, almacenamiento y búsqueda de datos en medios de almacenamiento, sin uso de consultas. la principal diferencia se encuentra en que los DAO se consideran más cercanos a la capa de datos y los Repository actúan más cerca de la capa de lógica de la aplicación, actuando sobre objetos específicos. En las aplicaciones que implementan el *framework* de Spring Boot, el Repository funciona como una extensión del DAO, simplificando el acceso a los datos [11].

Cuando se crea un nuevo objeto Repository, simplemente hay que definir una interfaz que extienda la clase JpaRepository proporcionada por Spring Boot. Sin definir ningún método dentro de la interfaz, al crear un objeto de ese tipo, se proporcionarán automáticamente métodos básicos para realizar operaciones, como poder buscar por identificador, obtener todos los elementos, o borrar un elemento específico.

Mediante el uso de Repository, se pueden escribir nuevos métodos para realizar operaciones más complejas, si se definen en la interfaz del repositorio. Únicamente es necesario especificar el tipo de dato que va a devolver, los parámetros necesarios, y el nombre del método. Si se sigue la sintaxis especificada por Spring Boot, el Repository será capaz de interpretar el nombre del método, construir una consulta al sistema de almacenamiento con el uso del DAO, y devolver los datos que resulten de la ejecución de la operación; sin necesidad por parte del programador de implementar el método o definir la consulta.

La implementación de un patrón de diseño DAO proporciona una serie de ventajas a los proyectos que lo incluyan:

- **Desacoplar la lógica de negocio de la capa de la aplicación**: como se ha comentado anteriormente, se puede mantener una estructura de datos independiente de la lógica del programa, por lo que se puede modificar la estructura de los datos de sistemas de almacenamiento persistentes o viceversa. La separación de las capas permite que la capa DAO se encargue de realizar las operaciones y conversiones necesarias sobre los datos del sistema de almacenamiento, para enviar los datos a la aplicación mediante la estructura definida en la capa de negocio. Además, se puede modificar la capa DAO para que realice operaciones de optimización sin interrumpir el funcionamiento de la aplicación.
- **Independencia sobre la capa de almacenamiento**: el patrón DAO permite un cambio del sistema de persistencia de datos utilizado independiente de la aplicación. Se puede cambiar el sistema de almacenamiento de una aplicación (una base de datos, almacenamiento en la nube, etc.) sin modificar el funcionamiento de la aplicación, solo se necesitaría volver a implementar los métodos DAO definidos. Este sistema también permite la configuración de distintos sistemas de persistencia que se intercambien según las necesidades.
- **Gestión de la persistencia unificada**: se simplifican las tareas de mantenimiento de código, ya que todos los objetos que definan métodos DAO se encuentran concentrados en una única ubicación. Además, al no ser necesaria la implementación de las consultas y métodos definidos, se reduce el tiempo de desarrollo y la probabilidad de que ocurran errores en la implementación.

4.3. Diseño de la arquitectura del sistema

La arquitectura del proyecto es un modelo cliente-servidor. El cliente realiza las peticiones a un servidor, que realiza los cálculos correspondientes y devuelve al cliente el resultado de la operación. Para implementar esta arquitectura se ha seguido un modelo de programación por capas. El objetivo es desacoplar el proyecto en partes independientes. De esta forma se puede trabajar en paralelo, y facilita la modificación de partes de la aplicación sin que afecte al resto del proyecto. La programación por capas sigue la siguiente estructura:

- **Capa de presentación**: el usuario final interactúa con esta capa, su función es presentar la información y recibir los datos que procesa el usuario. También se conoce como interfaz gráfica y debe ser entendible e intuitiva. Se comunica únicamente con la capa de negocio.
- **Capa de negocio**: el código de la aplicación reside en esta capa, se reciben las peticiones del usuario y se devuelve el resultado.
- **Capa de datos**: representa el medio de almacenamiento no volátil y las herramientas para acceder a los datos. Se comunica únicamente con la capa de negocio y responde a sus solicitudes.

A continuación se va a describir el diseño del proyecto a nivel de producción. Para ello, se describirán las clases que han sido desarrolladas durante la estancia de prácticas, que forman la estructura del proyecto (ver Figura 4.3).

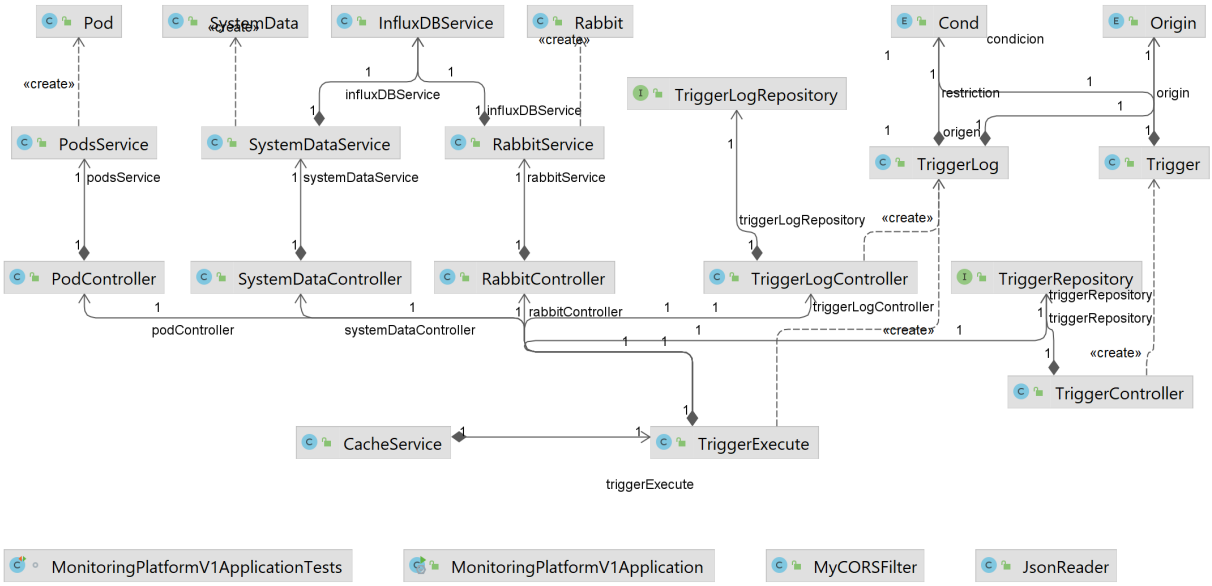

Figura 4.3: Diagrama de clases del proyecto de monitorización.

Como se utilizó el patrón de Modelo-Vista-Controlador, las clases se pueden agrupar en categorías. Algunas clases han sido importadas mediante librerías, o ya estaban desarrolladas previamente a la estancia de prácticas, por lo que no se van a mencionar. Otras clases cumplen funciones muy parecidas, por lo que se van a agrupar y explicar conjuntamente.

Modelos: Contiene las clases que representan los datos de los sistemas. Se definen sus parámetros y las funciones para realizar operaciones u obtener los datos de la clase. Las clases que pertenecen al modelo son:

- **Pod**: Almacena la información relacionada con un *pod* de Kubernetes, como su nombre, IP, estado y número de veces que se ha reiniciado. Se utiliza en otras clases del proyecto para almacenar de forma temporal los datos recuperados por el sistema de monitorización. Las clases de Rabbit y SystemData tienen un uso y estructura idénticos, almacenando los datos de colas de RabbitMQ y de Servidores de la empresa, respectivamente.

- **Trigger y TrigerLog**: Tienen la misma función que los métodos explicados en el apartado superior, almacenan los datos de las Alarmas configuradas y de los registros de errores, para luego poder procesarlos y mostrarlos por pantalla. La principal diferencia es que las clases utilizan anotaciones especiales de la API de persistencia de Java, para facilitar el almacenamiento de los datos en bases de datos.

Controlador: Las clases del controlador actúan como intermediarios entre el modelo y la vista. Se encargan de escuchar las peticiones de la vista, acceder a los datos almacenados en el medio de almacenamiento no volátil, realizar las operaciones necesarias sobre los datos, y los devuelve a la vista. Los métodos que forman parte del controlador son:

- **PodController, RabbitController, SystemDataController**: Como estos tres controladores representan los sistemas a monitorizar, estas clases solo tienen dos métodos. Contienen el método que responde a las peticiones HTTP emitidas por la vista, utilizando las herramientas proporcionadas por Spring Boot; y también contiene el método que conecta con el servicio que se encarga de recoger los datos.
- **TriggerController, TriggerLogController**: Su función es idéntica a los métodos del apartado anterior, se encargan de contestar a peticiones de la vista. Se distinguen de los anteriores en que responden no solo operaciones de lectura, sino también de escritura y edición.

Vista: Las clases de la vista no se han programado en el proyecto de monitorización, sino en el de la aplicación web, desarrollado con Angular. Todas las vistas del proyecto han seguido una misma estructura. Las vistas de alarmas, registros, *pods*, servidores y colas de *RabbitMQ* contienen:

- Un fichero HTML, que define la estructura y contenido de la vista, y un archivo CSS que aplica el estilo.
- Un componente, que define el comportamiento de la vista, y permite cambiar su contenido sin necesidad de recargar la página. Permiten la implementación de páginas dinámicas, y están programados en el lenguaje TypeScript.
- Un servicio que incluye los métodos que se conectan con el controlador correspondiente de la aplicación de monitorización, para recibir los datos que mostrar a la vista. Para realizar esa función, realiza peticiones HTTP a la aplicación de monitorización, y devuelve los datos recibidos al componente de la vista.

4.4. Diseño de las interfaces

En este apartado se hablará sobre el proceso de diseño de la interfaz gráfica. El proyecto a desarrollar es una aplicación web, por lo que la implementación de la interfaz debe realizarse utilizando el lenguaje de marcas HTML[1] para definir el contenido, el lenguaje de diseño CSS[2]

[1]HTML: HyperText Markup Language, es el lenguaje utilizado para estructurar y desplegar páginas web.
[2]CSS: Cascading Style Sheets, permite modificar la apariencia de los elementos definidos en un fichero HTML.

para personalizar la apariencia, y el lenguaje TypeScript para definir la funcionalidad y la interacción con el usuario.

4.4.1. Diseño de las vistas

La interfaz de una aplicación es el punto principal por el que interactuará el usuario. Una aplicación poco intuitiva puede provocar que se deje de usar la aplicación, o que se creen nuevas exigencias y modificaciones por parte del usuario, que no se tuvieron en cuenta inicialmente. Para su implementación, se han seguido unos principios de diseño que se centran en facilitar el uso de la aplicación al usuario. [12]

- **Conocer al usuario**: Se debe construir un perfil de usuario con la información del usuario objetivo (formación, experiencia, destreza, etc.). De esta forma, se pueden tomar decisiones de diseño específicas enfocadas a facilitar el uso de la aplicación al usuario objetivo. Un usuario puede cometer errores, por lo tanto, se debe diseñar la aplicación con esta mentalidad. El desarrollador debe tomar las medidas necesarias para la detección y prevención, mediante la restricción de las acciones y validación de datos. Un ejemplo de esto son los formularios con verificación de datos, que muestren mensajes con información para que el usuario sea capaz de rectificar su error.
- **Minimizar la memorización**: La aplicación no debe depender de la memoria de los usuarios, sino que debe diseñar con el objetivo de facilitar la información al usuario. Las acciones deben poder realizarse en pocos pasos, para evitar que el usuario tenga que memorizar pasos complejos para utilizar la aplicación. Ejemplos de este principio son el uso de comentarios informativos o accesos directos en la interfaz.
- **Optimizar las operaciones**: El objetivo es evitar que la aplicación entorpezca la interacción con el usuario. Se debe diseñar la interfaz para que sea intuitiva, rápida y fácil de usar. Debe ser claro para el usuario en todo momento cuál es el siguiente paso. También se tienen que tener en cuenta los tiempos de operaciones y secuencia de interacciones que debe realizar el usuario para cumplir su objetivo.
- **Diseñar para futuro**: Se debe tener en mente que tanto diseñadores de *software* como usuarios pueden cometer errores. Las interfaces web deben ser adaptables, estructuradas y consistentes. Deben de ser capaces de adaptarse a cualquier dispositivo y pantalla. También se recomienda la separación de la interfaz y los datos que alberga, teniendo una interfaz que funcione independientemente de los datos recibidos. De esta forma se consigue mantener la consistencia y facilitar la actualización de su contenido. Ejemplos de este principio son los menús desplegables, que ofrecen respuestas obtenidas por una consulta, o las tablas con datos que se construyen a partir de una información externa.

Se han seguido estos principios durante el diseño de las interfaces. La interfaz sigue un estilo minimalista, buscando un aspecto visual simple y relajado, para evitar saturar al usuario con información. Para ello, se ha usado una paleta de colores muy reducida, donde predomina el uso de tonos de grises. La funcionalidad de la aplicación se divide en páginas separadas, a las que se puede acceder mediante el uso de un menú de navegación. Y la estructura de cada página se divide en una serie de "cartas", que contienen y separan los elementos de la vista.

4.4.2. Creación de las vistas

Para la creación de las vistas, se han seguido los principios de diseño anteriores, y se ha buscado la simplicidad y cohesión con el resto de la aplicación. A la hora de integrar la aplicación dentro de la plataforma de control, se ha intentado imitar el estilo de la plataforma, para mantener la cohesión y no destaque respecto al resto de páginas del ecosistema.

Las interfaces de monitorización de datos son muy similares, las vistas de colas de *RabbitMQ* y *pods* de Kubernetes siguen una estructura muy parecida, por lo que solo se mostrará una de las dos. La lista de colas de RabbitMQ permite ordenar los elementos por los valores de sus columnas, o aplicar filtros para mostrar solamente los elementos que cumplan una condición especificada (ver Figura 4.4). Para la interfaz que contiene las métricas de los servidores, se muestra una visualización de datos por medio de tablas, se puede apreciar el uso de "cartas" para diferenciar los elementos que forman la web, se distribuye la información a lo largo de la pantalla, dejando espacio para mejorar la lectura de los datos, y el uso de distintos colores de fondo para cada fila, con el fin de mejorar la lectura (ver Figura 4.5).

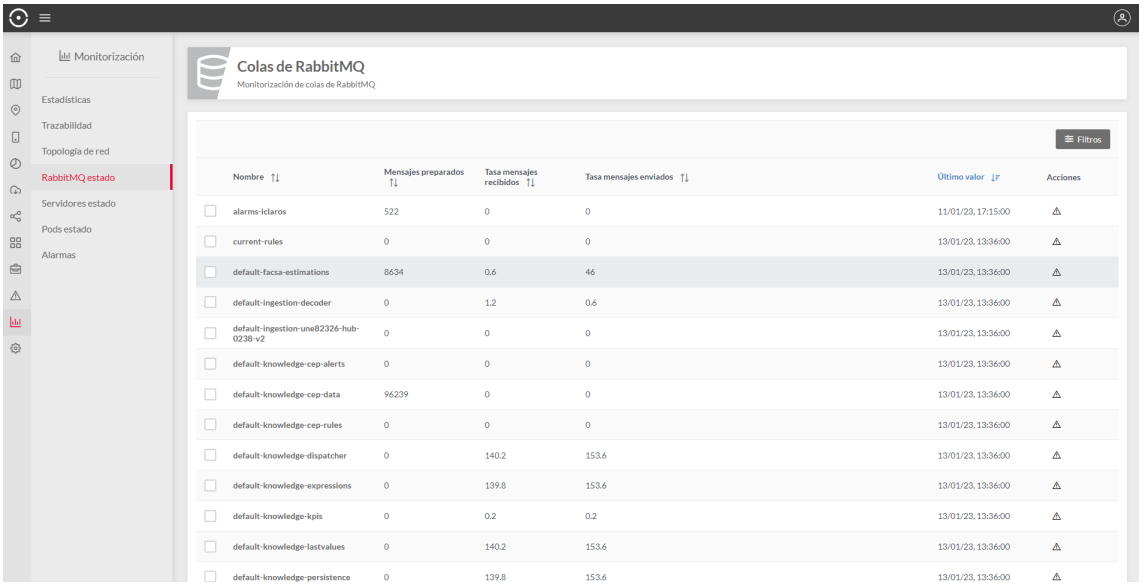

Figura 4.4: Vista de un listado de colas de RabbitMQ.

Para la interfaz encargada de visualizar las alarmas configuradas, la aplicación permite aplicar filtros sobre las alarmas, u ordenarlas según los valores de las columnas (ver Figura 4.6). La interfaz incluye un botón que abre el formulario de creación de una nueva alarma, y permite seleccionar y borrar alarmas de forma individual o de forma múltiple. Cada fila tiene una columna de acciones con iconos que permiten aplicar ciertas acciones sobre cada alarma listada, como poder activarlas o desactivarlas directamente desde la lista, o crear una suscripción a la alarma para recibir un correo cada vez que se active. Por último, se diseñaron ventanas de confirmación que se activan antes de operaciones que no se pueden deshacer, como el borrado de alarmas. Y se implementó un sistema de notificaciones que informa al usuario cuando una acción se ha realizado correctamente, o si se ha encontrado algún problema.

El formulario para registrar una nueva alarma, ha sido diseñado con el principio del diseño para errores en mente, mencionado en el apartado anterior. Se implementó un validador que no

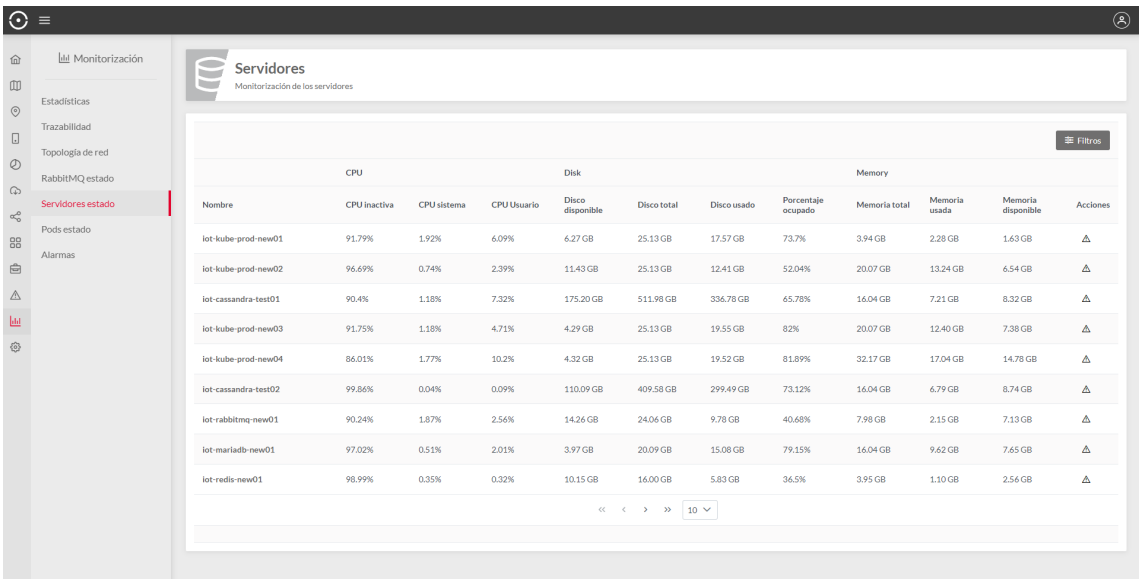

Figura 4.5: Vista con una tabla que contiene información de los servidores.

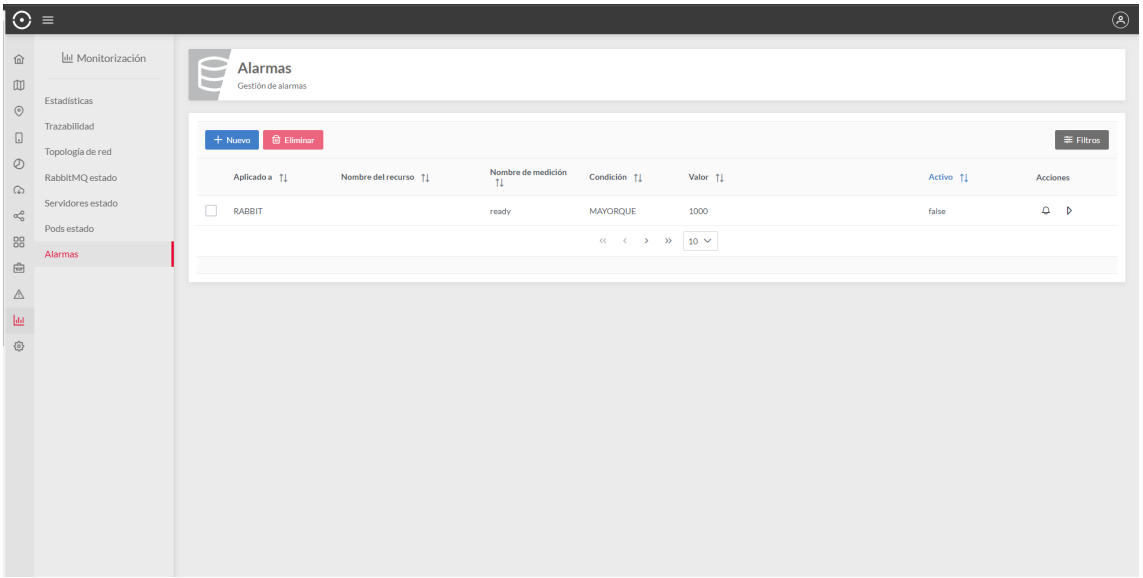

Figura 4.6: Vista de la lista de alarmas registradas.

permite enviar los datos si hay algún parámetro incorrecto, se restringió la entrada de valores para que solo acepte valores numéricos, y se incluyeron desplegables para simplificar y controlar los datos de la condición y el sistema a memorizar, en los que se profundizará más en el apartado de mejoras (ver Figura 4.7).

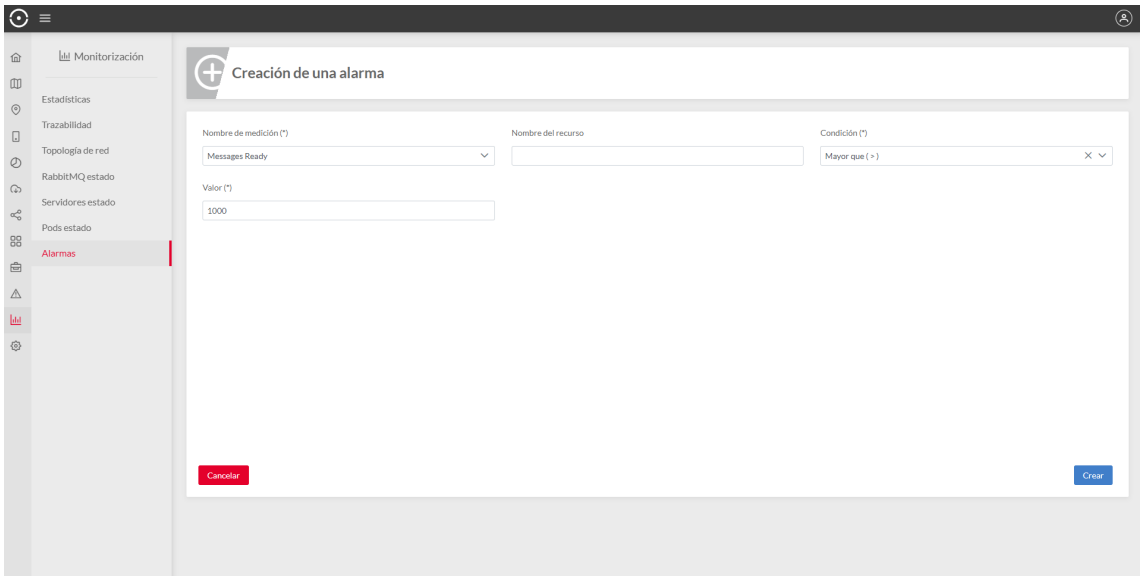

Figura 4.7: Vista del formulario para registrar nuevas alarmas.

Capítulo 5

Implementación y pruebas

5.1. Estructura del código

El proyecto ha sido dividido en dos partes, el servicio de monitorización, encargada de la lectura de los datos y gestión de alarmas y excepciones; y la aplicación web, que se conecta al servicio anterior y proporciona la interfaz al usuario para consultar la información y la consulta de alarmas. Se tomó esta decisión por la tarea de integración en la plataforma de control. Como el servicio es independiente de la interfaz, la plataforma de control puede realizar una conexión directa al servicio, sin tener que modificar la aplicación de monitorización. A su vez, como Angular se caracteriza por el uso de componentes reutilizables, la interfaz desarrollada serviría como base para las vistas que se implementarían en la plataforma de control.

A continuación se explicará como se han estructurado las clases que componen las dos aplicaciones del proyecto.

5.1.1. Aplicación de monitorización

La aplicación de monitorización ha sido desarrollada en el lenguaje de Java, utilizando en su totalidad el patrón MVC, por lo que las clases se organizan siguiendo la misma estructura, agrupando en paquetes[1] las clases con funcionalidad parecida. Los paquetes definidos son:

- **Controladores:** Agrupa las clases que definen los puntos finales de API[2], responden ante llamadas HTTP y se comunican con el servicio correspondiente para obtener los datos correspondientes con el formato especificado en el Modelo.
- **Modelos**: Son clases creadas para almacenar los datos que se procesarán en el controla-

[1]Recurso de Java para agrupar clases con funcionalidades parecidas. Funcionan como carpetas, separando y categorizando el contenido.

[2]Puntos finales de API (también conocidos como *endpoint*): proporcionan el medio para que aplicaciones externas puedan interactuar e intercambiar datos con la aplicación actual. La forma más común es mediante mensajes HTTP.

dor. Proporcionan una estructura común entre los elementos del mismo tipo, e incluyen métodos para obtener y procesar los datos. También se incluyen dentro las clases especiales de tipo enumeración, que limitan los valores de algunos parámetros de las clases del modelo.

- **Repositorios:** Incluye las clases que utilizan la abstracción proporcionada por Spring Boot que permite acceder a datos almacenados en medios como bases de datos sin programar los métodos o definir las consultas. Al compilar el código, se interpreta el nombre del método definido en el repositorio, y se crea de forma automática la consulta que permite obtener los datos.

- **Servicios:** Incluye las clases que añaden funcionalidad a la aplicación. Se incluyen los métodos que manejan la caché, que obtienen las métricas de InfluxDB y Kubernetes, métodos de configuración, y métodos de procesamiento de datos. Spring Boot permite etiquetar estas clases para indicar que proporcionan un servicio en la capa de negocio.

5.1.2. Aplicación web

La aplicación web fue desarrollada en Angular, que se caracteriza por el uso de componentes modulares y reutilizables. Esta propiedad ha afectado a la estructura de la aplicación web, que ha sido dividida en:

- **Componentes:** El elemento más básico de Angular, todos los elementos dentro de la interfaz están formados por ellos. Los componentes son un conjunto de elementos que definen una parte concreta de la interfaz. Están formados por un fichero HTML, que define la estructura del componente en la interfaz; un fichero CSS, que define las propiedades y estilos que se aplicarán al HTML; y un fichero escrito en TypeScript, que definirá funcionalidades y comportamientos, y se comunicará con el servicio para obtener los datos del *backend*. El proyecto incluye un componente por cada elemento de la interfaz que se quería implementar, como la lista de servidores, o el formulario de creación de alarmas.

- **Modelos:** Al igual que con la aplicación de monitorización, son clases que se utilizan para generar objetos de un tipo. Permite almacenar información sobre tipos de objetos, e implementar métodos que permitan transformar los datos almacenados. Se ha creado uno para cada recurso que se maneja en la aplicación, como las alarmas o los datos de colas de RabbitMQ.

- **Tuberías:** En informática, una tubería es una herramienta que permite transferir datos para ejecutar varias instrucciones de forma secuencial. Angular proporciona las herramientas para la creación de tuberías personalizadas, que reciben un valor de entrada, y lo transforman para mostrarlos en la vista utilizando el formato especificado en su implementación. La única tubería que se definió en el proyecto transforma un número de bytes a Gibibytes. Fue necesaria su implementación, ya que las métricas de almacenamiento almacenadas en InfluxDB se devuelven en bytes, y su procesamiento a otra unidad facilita su lectura.

- **Servicios:** Los servicios abarcan cualquier clase que proporcione unos datos, funciones o herramientas que una aplicación necesita. Se han utilizado principalmente para recoger los

datos de la aplicación de monitorización, para pasarlos a la vista. Esos servicios construyen una petición HTTP, la envían al controlador correspondiente, y esperan su respuesta, que se interpretará y procesará para pasar el resultado a la vista. También se ha creado un servicio de validación de formularios, que recibe los valores introducidos y comprueba que tengan un formato válido.

5.2. Descripción técnica de la implementación

La implementación de la aplicación de monitorización ha sido realizada utilizando el *framework* de desarrollo Spring Boot. Proporciona las herramientas necesarias para implementar aplicaciones web y micro-servicios de forma rápida y fácil. [13]

La principal ventaja de Spring Boot es la función de inyección de dependencias. En las clases del proyecto se definen constructores, que contienen todas las dependencias necesarias para su funcionamiento (ver Figura 5.1).

```
@Autowired
public TriggerExecute(TriggerRepository triggerRepository, PodController podController,
                      RabbitController rabbitController, SystemDataController systemDataController,
                      TriggerLogController triggerLogController, RabbitTemplate rabbitTemplate,
                      ObjectMapper objectMapper) {
    this.triggerRepository = triggerRepository;
    this.podController = podController;
    this.rabbitController = rabbitController;
    this.systemDataController = systemDataController;
    this.triggerLogController = triggerLogController;
    this.rabbitTemplate = rabbitTemplate;
    this.objectWriter = objectMapper.writerFor(TriggerLog.class);
}
```

Figura 5.1: Ejemplo de un constructor de Spring Boot.

Estos constructores especiales se definen utilizando la etiqueta específica `@Autowired`, que se proporciona por Spring Boot. De esta forma, cuando se cree un objeto de la clase anterior, no se tendrán que proporcionar todas las dependencias especificadas en el constructor. Spring Boot se encarga automáticamente de inicializarlas e importarlas. Este sistema favorece la creación de aplicaciones modulares, con clases que realizan tareas específicas. Y como se puede ver en la sección 5.1 “Estructura del código”, esta propiedad ha influenciado el desarrollo, se han separado las clases por la funcionalidad que proporcionan.

5.2.1. Implementación del sistema de obtención de datos

Para que la aplicación de monitorización obtenga los datos de los sistemas de la empresa, se han desarrollado tres servicios separados. Los datos de RabbitMQ se podían obtener mediante consultas a la API de RabbitMQ, o mediante consultas a la base de datos de InfluxDB. Se optó por la segunda opción, ya que los datos de servidores solo se pueden obtener mediante la base de datos, por lo que se podría reusar código para ambos servicios.

La obtención de los datos de InfluxDB se realiza mediante el uso de una librería externa, que devuelve los datos en una lista de objetos sin formato, que se procesan en objetos POJO que siguen la estructura definida en el Modelo. A continuación, se ponen a disposición del controlador.

La principal diferencia entre el servicio de RabbitMQ y de Servidores es que son necesarias tres consultas separadas para obtener todos los datos de los servidores, ya que las métricas de la CPU, memoria secundaria y RAM se almacenan en tablas distintas, y en las consultas de InfluxDB no permiten obtenerlos con una única consulta, ya que la opción de "JOIN" usada en otros lenguajes, solo funciona para unir tablas por la fecha de la medida. Por lo tanto, se realizan tres consultas separadas, y en un método separado se procesan y almacenan en una lista de POJOs.

Para la obtención de los datos de Kubernetes, se realiza una petición HTTP a su API, que permite obtener una lista del estado de los *pods* en formato JSON. Para proteger el acceso a los datos, es necesario agregar un token de autenticación a la petición HTTP. Para generarlo, se realiza una conexión directa al servidor mediante SSH, y se genera el token que servirá para autorizar las peticiones enviadas por el programa.

En la implementación antigua del sistema, se llamaban automáticamente a los métodos anteriores cada vez que un cliente accedía a la aplicación web. Ese sistema se mejoró creando una clase que utiliza la etiqueta de Spring Boot `@Scheduled`, que permite crear un servicio que se ejecuta repetidamente en intervalos de tiempo especificados. Ese servicio llama de forma periódica a las clases anteriores, y actualiza los datos en una caché, reduciendo así el tiempo de respuesta para usuarios. Esta implementación se explicará con más detalle en el apartado de mejoras.

5.2.2. Implementación de la aplicación web

Para la implementación de la aplicación web, se desarrolló una aplicación de Angular básica. No se encontraron problemas durante el desarrollo, ya que la documentación de este framework es abundante. Angular proporciona numerosas herramientas para facilitar las tareas de programación. Las vistas eras principalmente listas o tablas, solamente se necesita definir las columnas y la fuente de información, y Angular se encarga de crear las filas correspondientes con la información (ver Figura 5.2).

Todos los elementos de una aplicación hecha con Angular están formados por elementos modulares y reutilizables. Un componente puede contener y cargar otros componentes. El HTML principal, la raíz de la aplicación, no contiene casi elementos. Contiene las etiquetas básicas de funcionamiento, y en el cuerpo de la página, una referencia al componente de enrutación de Angular. Este componente es capaz de identificar el recurso que se está pidiendo mediante la URL (Uniform Resource Locator), y cargar el componente relacionado (ver Figura 5.3). La tabla de datos de la figura 5.2 se muestra en la vista cuando este componente carga el recurso.

Finalmente, se implementaron los servicios que realizan las peticiones HTTP a la aplicación de monitorización, y proporcionan los datos a los componentes. La integración con la plataforma de control se realizó sin problemas, al ser un proyecto ya existente, se pudo reutilizar parte del

```
<table mat-table [dataSource]="dataSource" class="mat-elevation-z8 demo-table">
  <ng-container matColumnDef="timestamp">
    <th mat-header-cell *matHeaderCellDef> última medida</th>
    <td mat-cell *matCellDef="let rabbit">  {{rabbit.timestamp | date:'yyyy-MM-dd HH:mm:ss'}}</td>
  </ng-container>

  <ng-container matColumnDef="queue_name">
    <th mat-header-cell *matHeaderCellDef> Nombre </th>
    <td mat-cell *matCellDef="let rabbit"> {{rabbit.queue_name}} </td>
  </ng-container>

  <ng-container matColumnDef="messages_ready">
    <th mat-header-cell *matHeaderCellDef> Messages Ready</th>
    <td mat-cell *matCellDef="let rabbit"> <span
      [style.color]="rabbit.messages_ready > 1000 ? 'red' : null">
      {{rabbit.messages_ready}}
  </span> </td>
  </ng-container>

  <ng-container matColumnDef="messages_publish_rate">
    <th mat-header-cell *matHeaderCellDef> Incoming rate</th>
    <td mat-cell *matCellDef="let rabbit"> {{rabbit.messages_publish_rate}}/s </td>
  </ng-container>

  <ng-container matColumnDef="messages_deliver_rate">
    <th mat-header-cell *matHeaderCellDef> Deliver rate</th>
    <td mat-cell *matCellDef="let rabbit"> {{rabbit.messages_deliver_rate}}/s </td>
  </ng-container>

  <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
  <tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
</table>
```

Figura 5.2: Ejemplo de una tabla desarrollada con Angular.

código existente para las nuevas funcionalidades.

5.3. Mejoras

En el apartado de mejoras se discutirán las modificaciones más relevantes que se han realizado al proyecto original, con el objetivo de mejorar algún aspecto de su funcionalidad. Estas modificaciones no se planificaron y surgieron durante el desarrollo del proyecto.

5.3.1. Simplificación del formulario de alarmas

Durante el desarrollo de las interfaces, se detectó un problema de la lógica del sistema de alarmas. En el formulario se incluyeron los campos para el entorno a monitorizar (Kubernetes, RabbitMQ y servidores) y otro campo en el que el usuario debía incluir qué parámetro de ese entorno se quería monitorizar. El problema radica en que las opciones disponibles del segundo

```
const routes: Routes = [
  { path: '', redirectTo: 'containers', pathMatch: 'full'},
  { path: 'triggerTable/:id', component: TriggerDetailsComponent},
  { path: 'triggerTable', component: TriggerTableComponent},
  { path: 'systemList', component: SystemTableComponent},
  { path: 'rabbitList', component: RabbitTableComponent},
  { path: 'logList', component: TriggerLogTableComponent},
  { path: 'podList', component: PodTableComponent},
  { path: 'triggerAdd', component: TriggerAddComponent},
  { path: 'gridster', component: GridsterTestComponent}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Figura 5.3: Fichero de enrutación de Angular.

campo, dependen del primero. Si se utilizaba un cuadro de texto para ese parámetro, el programa dependía de la memoria del usuario para recordar qué parámetros son los disponibles. Además, habría que implementar un sistema de validación y normalización de las entradas para recibir los datos correctamente.

Otra alternativa era utilizar un menú desplegable, pero se tendrían que incluir las opciones de todos los entornos, y se crearía la posibilidad de configurar alarmas con parámetros incorrectos (por ejemplo, se podría crear una alarma que monitoriza las veces que se ha reiniciado un servidor, cuando ese campo no existe en la tabla de servidores). Y se seguiría necesitando un sistema de validación.

La solución fue eliminar el campo de entorno del formulario, y mostrar un menú desplegable con todas las opciones disponibles. A partir del parámetro escogido en el desplegable, se puede implementar un método que calcule el entorno monitorizado (ver Figura 5.4).

5.3.2. Optimización del coste de acceso a datos

Esta fue la mejora más significativa del proyecto, y también la más necesaria. Como se mencionó brevemente en el esprint 3 del apartado de seguimiento del proyecto, se tuvo que reescribir parte del proyecto para programar el sistema de alarmas. Durante su desarrollo, se encontró un problema con la carga de los datos. Cada vez que un cliente cargaba la interfaz web, se enviaba una petición para cada servicio del que se quería obtener información (ver Figura 5.5).

Para obtener los datos de cada recurso se deben realizar un total de cinco consultas independientes: una para obtener los *pods*, otra para obtener las colas de RabbitMQ, y tres consultas a InfluxDB para obtener la CPU, memoria y espacio en disco de los servidores. La interfaz se programó para que se enviaran nuevas peticiones de datos cada minuto con el objetivo de

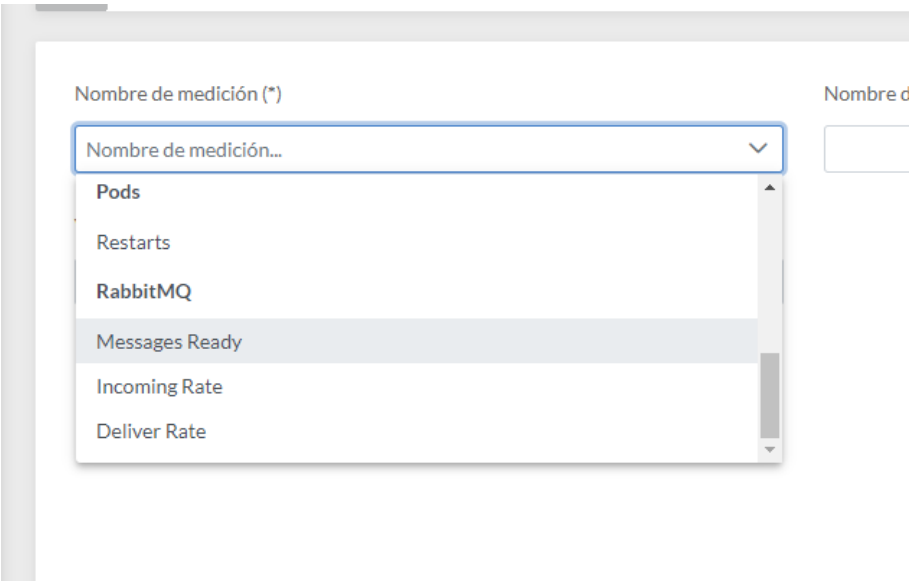

Figura 5.4: Mejora en el formulario para reducir errores.

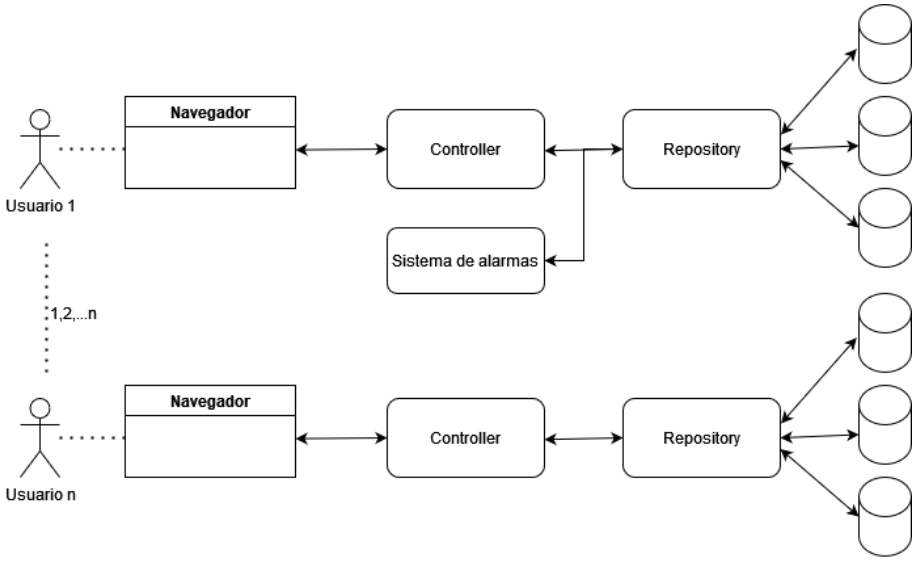

Figura 5.5: Estructura de las peticiones original del proyecto.

mostrar al usuario la información más reciente.

Si la aplicación es usada por solo un cliente, no presenta ningún problema, ya que la petición del cliente genera las peticiones a la base de datos correspondientes. El problema se encuentra cuando múltiples clientes utilizan la aplicación web simultáneamente. Para cada cliente, se realizan cinco peticiones, provocando que el tiempo de respuesta aumente casi exponencialmente. Además, la información que reciban dos clientes de sus peticiones será prácticamente idéntica, y habrá provocado un número innecesario de consultas a la base de datos.

La solución adoptada ha sido implementar un sistema de Caché, que almacene las métricas de los sistemas de la empresa. El contenido de la Caché se actualizaría automáticamente cada

minuto, mediante llamadas a los servicios que obtienen los datos de los sistemas. De esta forma, no importa la cantidad de usuarios conectados, ya que las consultas se mantendrán constantes.

Otra ventaja del nuevo sistema es el tiempo de respuesta. Antes de aplicar la mejora, el cliente debía esperar a que se realizara la consulta para obtener los datos. Utilizando el sistema de caché, el usuario tiene disponible los datos inmediatamente, ya que la consulta se ha realizado previamente. Además, se facilita la implementación del servicio de alarmas, que escaneará las métricas de los sistemas antes de reemplazar los datos en la caché. De esta forma, aunque el proceso de las alarmas tenga un coste temporal elevado, los usuarios siguen teniendo acceso a las métricas de sistemas (ver Figura 5.6).

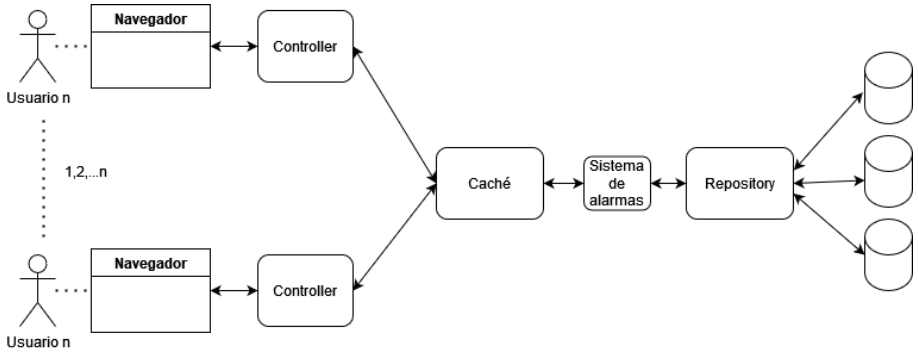

Figura 5.6: Estructura de las peticiones original del proyecto.

Se ha implementado el sistema de Caché mediante la abstracción proporcionada por Spring. Se proporcionan etiquetas que se pueden incluir en los métodos para definir la funcionalidad de caché a aplicar. Cuando se va a ejecutar un método etiquetado con "Cacheable", se busca si ya existe un registro en la caché asociado a ese método, y se devolverá su valor, evitando volver a ejecutar el código. También se incluyen etiquetas para vaciar o reemplazar etiquetas.

Se puede observar el crecimiento casi exponencial en la implementación sin caché. Si se realizan varias peticiones de usuarios al mismo tiempo, la aplicación debe responder en orden a cada una de ellas. Mientras que con la implementación con caché, el coste se mantiene lineal, ya que el acceso a la información es inmediato y no se realizan peticiones a la base de datos (ver Tabla 5.1).

Número de usuarios conectados	Tiempo de respuesta (s)	
	Sin caché	Con caché
1	1,32	1,32
2	2,63 + 0,1	1,32
3	4,4 + 0,2	1,32
n	1,31 * n + 0,1 * (n-1)	1,32

Tabla 5.1: Comparación de costes antes y después de la implementación de Caché.

5.4. Verificación y validación

La validación y verificación es una de las últimas fases de todo desarrollo de software. Su objetivo es comprobar que se cumplen los requisitos especificados por el cliente. Estos procedimientos también se utilizan para comprobar que el producto final cumple con los niveles de calidad típicos de un producto de software.

A continuación se describirán las pruebas realizadas para validar el correcto funcionamiento de la aplicación. Estas pruebas no representan todas las pruebas que se realizarán sobre el producto final, ya que al terminar la integración con la plataforma de control, se realizarán pruebas más exhaustivas utilizando la herramienta Cucumber[3]. Durante el desarrollo del proyecto, se han realizado pruebas para verificar la efectividad de la implementación escogida y que los datos recibidos de las fuentes monitorizadas son correctos. Se han utilizado pruebas unitarias, tests de aceptación y pruebas de integración.

5.4.1. Pruebas unitarias

Las pruebas unitarias son la unidad más pequeña de pruebas que se pueden realizar en el proceso de desarrollo de software. Se comprueba el correcto funcionamiento de componentes concretos y aislados.

El proyecto desarrollado se caracteriza por su modularidad, las distintas funciones del software están separadas mediante el uso de clases reutilizables e independientes. Por lo tanto, la realización de tests que permitan probar el funcionamiento de los módulos de forma independiente resulta ser un proceso sencillo. Solo es necesario inicializar las variables necesarias y llamar al método que se encarga de realizar la función.

La implementación de los tests se ha realizado definiendo métodos dentro del proyecto, mediante el uso de la herramienta JUnit. La herramienta es una librería de código abierto que permite la ejecución de tests de forma automática en la JVM[4] (Java Virtual Machine). También ofrece integración con los entornos de desarrollo de software, para facilitar su uso. Mediante el uso de anotaciones especiales en un código, se definen los métodos especiales que se utilizarán para pruebas. Cuando se compila un proyecto de Java, los tests definidos se ejecutan automáticamente para comprobar que todos los procesos funcionen correctamente.

En los tests realizados en el proyecto, se puede apreciar el uso de la anotación `@Test`, proporcionada por JUnit (ver Figura 5.7). Esa etiqueta le indica al IDE que es un método que se puede ejecutar para validar una operación. En los métodos de tests se especifica qué resultado se espera obtener, mediante el uso del método `assertThat()`, que compara el resultado de la operación del método con el esperado. Si coinciden, se considera que se ha superado el test. La etiqueta de `@Before` se utiliza para marcar el orden de ejecución. En el código mostrado se usa la etiqueta para inicializar el servicio de RabbitMQ antes de la ejecución de los tests.

[3]Cucumber: Herramienta para la validación de código que permite especificar los comportamientos esperados del software en un lenguaje lógico que los clientes puedan entender.

[4]JVM: Permite interpretar y ejecutar instrucciones escritas en el lenguaje Java.

```
no usages
@Before
public void setUp(){
    rabbitService = new RabbitService();
}

no usages
@Test
public void fetchRabbitMQData(){
    String queueName = "test-modbus";

    Rabbit receivedQueue = rabbitService.getRabbitByName(queueName);

    assertEquals(receivedQueue.getMessagesReady(), (Double) 8.0);
    assertEquals(receivedQueue.getMessagesDeliverRate(),(Double) 0.0);
    assertEquals(receivedQueue.getMessagesPublishRate(),(Double) 0.0);

}
```

Figura 5.7: Estructura de las peticiones original del proyecto.

Como la aplicación ha seguido un diseño modular, se puede verificar fácilmente el correcto funcionamiento de las distintas partes de nuestro sistema. No se han realizado tests sobre todas las funcionalidades de la aplicación, sino sobre los componentes que es difícil predecir su comportamiento y son más propensos a generar errores, o que son parte de una parte indispensable del funcionamiento de la aplicación, como el sistema de generación de excepciones.

5.4.2. Pruebas de integración

Las pruebas de integración forman parte de la fase de pruebas del desarrollo de software. En esta fase se comprueba que los módulos de software individuales desarrollados pueden funcionar de forma conjunta.

Para realizar las pruebas de integración, la empresa utiliza un servidor que proporciona el servicio de Jenkins. Este sistema ofrece un sistema de integración continua, que permite construir, desplegar y automatizar los proyectos de la empresa. Se utiliza para gestionar las versiones de un proyecto, automatizar tareas repetitivas, y realizar pruebas de funcionamiento de forma automática.

La herramienta de Jenkins permite configurar su comportamiento mediante el uso de extensiones. Estos se han configurado para que actúen cuando se despliegan nuevas funcionalidades en el sistema. Esta herramienta comprueba que se ha realizado la integración con el sistema actual correctamente, y detecta posibles errores de ejecución. En el caso de que no se despliegue la aplicación correctamente, se recopilará toda la información necesaria y enviarán correos a los responsables con la información del problema, de forma automática y sin interacción humana.

5.4.3. Pruebas de aceptación

Las pruebas de aceptación son una descripción formal del comportamiento del producto de software desarrollado. Se utilizan como comprobación del correcto funcionamiento y cumplimiento de los requisitos especificados por el cliente.

Como se ha utilizado una metodología Ágil de tipo Scrum, las historias de usuario del apartado 3.1.1 se utilizarán para definir las condiciones que especifican cuando una tarea está lista para su revisión. En la revisión de la tarea se comprobará su correcto funcionamiento y su implementación.

Las pruebas de aceptación se muestran a continuación, agrupadas por las distintas secciones que forman la aplicación:

- Tablas de consulta de métricas de los sistemas (HU01, HU02, HU03, HU04)
  - Se lista la tabla con las métricas de los servidores de la empresa, las colas de RabbitMQ y los *pods* de Kubernetes.
  - Se pueden filtrar y ordenar las filas por sus distintos parámetros.
  - Las tablas de métricas se actualizan automáticamente con la información más reciente.
- Gestión y activación de alarmas (HU05, HU06, HU07, HU08)
  - Se proporciona una interfaz desde la que poder configurar nuevas alarmas.
  - Se pueden listar y gestionar las alarmas desde un único lugar. Permitiendo su activación y desactivación.
  - Se dispone de un historial de las últimas excepciones lanzadas por el sistema de monitorización.
  - El sistema de alarmas genera y registra excepciones automáticamente cuando se cumplen los parámetros especificados.
- Integración con la plataforma de control (HU09, HU10, HU11)
  - Se restringe el acceso al sistema de alarmas a los usuarios sin los permisos correspondientes.
  - Las herramientas de gestión de alarmas están disponibles dentro de la plataforma de control.
  - Pulsar sobre un recurso monitorizado debe dar la opción de crear una alarma sobre ese recurso en específico.

Capítulo 6

Conclusiones y mejoras

6.1. Conclusiones

El objetivo del proyecto ha sido diseñar e implementar una aplicación de monitorización en tiempo real, de los sistemas de la empresa de Grupo Gimeno. La aplicación desarrollada cumplió con los requisitos especificados, y se implementaron dentro del plazo establecido.

Trabajar en este proyecto ha sido una experiencia muy interesante. Desde un punto de vista técnico, se ha puesto a prueba mi capacidad para diseñar y desarrollar un sistema robusto y escalable, capaz de recopilar y analizar grandes cantidades de datos en tiempo real, con un tiempo de implementación limitado y unas especificaciones concretas. También he aprendido sobre tecnologías emergentes en el campo del internet de las cosas y la analítica de datos, lo que me ha permitido integrar estas capacidades en la aplicación.

En cuanto a la gestión del proyecto, la metodología ágil aplicada y las herramientas de colaboración en equipo han garantizado una comunicación fluida y una entrega de los resultados acorde a los plazos establecidos. En las primeras semanas de estancia en prácticas, el progreso era lento y la cantidad de nuevas tecnologías en las que formarse era abrumadora. Aun así, se progresaba todos los días, y gracias a las reuniones diarias, se detectaban problemas de implementación y se evitaban retrasos importantes. Con la ayuda del supervisor y de los compañeros del departamento de software, se consiguieron resolver los apartados de la implementación más complicados. La reserva de horas de margen en caso de contratiempos resultó ser una opción acertada, ya que se aprovechó ese tiempo en implementar mejoras propuestas por el usuario.

En cuanto a la utilidad de los estudios, los conocimientos adquiridos a lo largo de la carrera y del itinerario de Tecnologías de la Información han resultado ser muy útiles. Algunas de las herramientas, lenguajes de programación y patrones de diseño que se usaron, ya se habían estudiado en la universidad, o seguían una estructura muy similar, por lo que adaptarse a las nuevas tecnologías fue relativamente fácil.

A nivel personal, esta experiencia ha sido desafiante, pero también gratificante. He aprendido a trabajar en equipo en un entorno profesional, a adaptarme a las situaciones cambiantes y

a enfrentar problemas complejos con creatividad y determinación. He podido desarrollar mis aptitudes de trabajo en equipo y potenciar mis habilidades de programación para el desarrollo de proyectos.

Por último, destacar que la experiencia de trabajar en el desarrollo de un proyecto y en la empresa ha sido muy positiva, en parte gracias a mis compañeros de trabajo, que me asistían en los problemas que surgían durante el desarrollo, y en parte por la empresa, que me ha permitido trabajar en un proyecto con utilidad en el mundo real. El desarrollo del proyecto me ha permitido ganar conocimientos en tecnologías relevantes para el desarrollo de software y aplicaciones web, que me abrirán muchas puertas en el futuro.

6.2. Trabajo futuro

Aunque los requisitos funcionales definidos para la aplicación de monitorización se completaron, se propusieron una serie de mejoras que no se pudieron completar en el tiempo establecido. Estas mejoras estaban pensadas para proporcionar mayor funcionalidad y facilitar su uso al usuario.

- Implementación de un Dashboard[1] que muestre desde una vista un resumen del estado actual de los sistemas de la empresa. Para ello se incluirán gráficas de rendimiento y consumo de recurso, los valores de los KPI[2] (Key Performance Indicator).
- Añadir soporte para la herramienta de *Transloco*, que permite definir traducciones para el contenido de la aplicación web, y cambiar el idioma en tiempo de ejecución. Esta mejora se implementó de forma parcial, ya que no se completaron todas las traducciones a tiempo.
- Implementar un sistema de subscripción a alarmas. Ese sistema debería permitir a los usuarios suscribirse a una alarma, y cuando genere una excepción, se notificará por un servicio de mensajería con la información necesaria.

[1] Dashboard: También conocido como Cuadro de mandos, representa de manera gráfica métricas y estadísticas.
[2] KPI: Es una métrica que indica el desempeño de un sistema.

Bibliografía

[1] Grupo Gimeno. Nuestro grupo. https://www.grupogimeno.com/grupo-gimeno/. [Consulta: 14 de Febrero de 2023].

[2] IoT. Internet de las cosas (IoT). https://es.wikipedia.org/wiki/Internet_de_las_cosas, 2023. [Consulta: 30 de Enero de 2023].

[3] Pavel Fol. Spring vs the World: Comparing SpringBoot Alternatives. https://www.jrebel.com/blog/spring-boot-alternatives. [Consulta: 18 de Marzo de 2023].

[4] RabbitMQ.com. RabbitMQ tutorial - Publish/Subscribe. https://www.rabbitmq.com/tutorials/tutorial-three-python.html. [Consulta: 9 de Marzo de 2023].

[5] unir.net. Metodologías de gestión de proyectos. https://www.unir.net/empresa/revista/metodologias-gestion-proyectos/. [Consulta: 30 de Enero de 2023].

[6] scrummanager.com. Pila del producto. https://www.scrummanager.com/bok/index.php/Pila_del_producto/. [Consulta: 15 de Febrero de 2023].

[7] Indeed.com. ¿Cuánto se gana como uno Programador/a junior en España? https://es.indeed.com/career/programador-junior/salaries. [Consulta: 21 de Febrero de 2023].

[8] Pankaj. Java Design Patterns. https://www.digitalocean.com/community/tutorials/java-design-patterns-example-tutorial/. [Consulta: 15 de Febrero de 2023].

[9] MVC. Patrón de diseño Modelo-Vista-Controlador (Model-View-Controller). https://en.wikipedia.org/wiki/Model-view-controller. [Consulta: 15 de Febrero de 2023].

[10] geeksforgeeks.org. MVC Design Pattern. https://www.geeksforgeeks.org/mvc-design-pattern/. [Consulta: 15 de Febrero de 2023].

[11] Anshul Bansal. DAO vs Repository Patterns. https://www.baeldung.com/java-dao-vs-repository. [Consulta: 15 de Febrero de 2023].

[12] uxpin.com. The basic principles of user interface design. https://www.uxpin.com/studio/blog/ui-design-principles/#toc_4--Aim-for-Simplicity. [Consulta: 9 de Marzo de 2023].

[13] IBM. ¿Qué es Java Spring Boot? https://www.ibm.com/ar-es/topics/java-spring-boot. [Consulta: 27 de Febrero de 2023].

[14] Oracle. Primitive data types. https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html. [Consulta: 21 de Marzo de 2023].

Anexo A

Desarrollo del decodificador de protocolo Modbus

A.1. Introducción

Como se terminó el proyecto de prácticas antes de lo planificado, se desarrolló un proyecto adicional para la empresa. Este proyecto consiste en el desarrollo de una librería que proporcione los métodos para la decodificación de mensajes que utilicen protocolo de mensajes Modbus. Este decodificador recibe los mensajes emitidos por un sensor, y los transforma a un formato que pueda ser interpretado por la plataforma de control. De esta forma, todos los dispositivos de la empresa que utilicen este protocolo, pueden ser monitorizados en tiempo real desde la aplicación web.

Aparte del decodificador, también se desarrolló una librería que funciona como codificador. Proporciona soporte para la construcción automática de mensajes del protocolo Modbus. Este funcionará de forma paralela al decodificador. El codificador construye y envía las peticiones de datos al sensor, mientras que el decodificador recibirá la respuesta del sensor, y la transformará en un formato compatible con la plataforma de control.

A.1.1. El protocolo Modbus

Modbus es un protocolo de mensajería implementado utilizando una comunicación que ocurre entre parejas de dispositivos. Un dispositivo envía una petición de datos, y espera la respuesta del receptor. Normalmente, el emisor suele ser un programa o un sistema de monitorización, mientras que el receptor suele ser un sensor, que está monitorizando un recurso. Los mensajes emitidos por el sensor siguen siempre el mismo formato, que se puede consultar en la Tabla A.1.

Los mensajes que utilizan el protocolo Modbus sigue una estructura simple para la transmisión de información:

Dirección	Función	Datos	Comprobador CRC
8 bits	8 bits	N x 8 bits	16 bits

Tabla A.1: Estructura de un mensaje del protocolo Modbus.

- **Dirección:** Varios sensores pueden estar conectados a una misma red. La dirección es un número de dos cifras que se utiliza como identificador único.
- **Función:** Modbus permite enviar operaciones de lectura y escritura. En el campo de función se indica el tipo de operación. Hay cuatro operaciones de lectura, para la lectura del relé, entradas digitales o analógicas del sensor. Y también hay cuatro operaciones de escritura, que se aplican sobre los mismos tipos de entradas que en las operaciones de lectura.
- **Datos:** Contiene los datos a escribir o leer del sensor. Dependiendo de la función y de la cantidad de datos de la petición, el tamaño de este campo puede variar. Este campo siempre tendrá un número de bytes múltiplo de 8, y en caso de enviar datos de menor tamaño, se usa el relleno de bytes.
- **Comprobador CRC:** Es una medida de seguridad, se genera un código a partir del contenido del resto de campos. El receptor del mensaje vuelve a generar el código con la información recibida. Si no coincide con el CRC recibido, significa que el mensaje ha sido alterado durante su envío.

La empresa tenía configurado un sensor que emitía mensajes en formato Modbus de forma periódica. Los datos se reciben como una lista de bytes, que se pueden transformar a hexadecimal para obtener su valor. En la Tabla A.2 se puede visualizar un ejemplo de uno de los mensajes recibidos.

Dirección	Función	Datos	Comprobador CRC
0101 0101	0000 0010	0000 0001 0000 0000	1011 0001 1011 1000
55	02	0100	B1B8

Tabla A.2: Ejemplo de uno de los mensajes emitidos por el sensor.

El mensaje anterior indica que se envió por el sensor de la dirección 55, la función que ha generado el mensaje ha sido de tipo 2, que según la documentación, es una operación de lectura de una corriente eléctrica, que en este caso ha devuelto un valor de 100. Se puede comprobar la integridad del mensaje mediante el CRC. Los códigos CRC disponen de una propiedad única, si se utiliza el mensaje entero, incluyendo el CRC, para generar un nuevo CRC, devolverá una lista de ocho bits de valor 0, si el mensaje es correcto.

A.2. Estructura del proyecto

Con el conocimiento anterior sobre el esquema de un mensaje de Modbus, se puede proceder a explicar la implementación. Para el correcto funcionamiento del sistema de monitorización del sensor, se tuvieron que implementar distintos sistemas interconectados (ver Figura A.1).

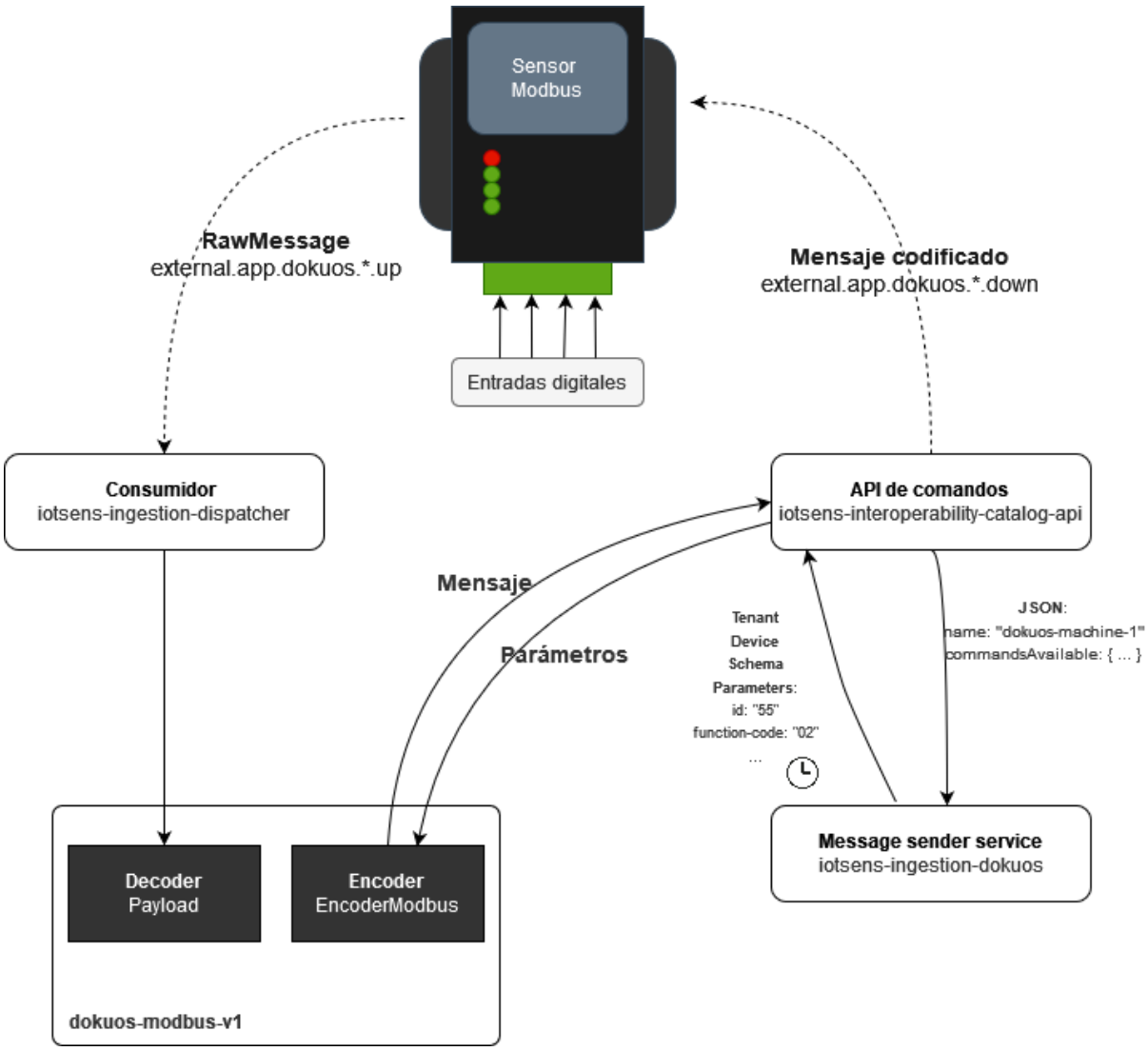

Figura A.1: Interacción entre los distintos sistemas para la comunicación con el sensor.

A continuación se explicarán los distintos sistemas y su función en el proyecto.

- **dokuos-modbus-v1:** Este proyecto proporciona las librerías de codificación y decodificación de los mensajes de Modbus. Contiene las funciones y métodos para la construcción de mensajes y para su traducción.
- **iotsens-ingestion-dispatcher:** Recibe los mensajes emitidos por el sensor, mediante una conexión a una cola de RabbitMQ. Este sistema se encarga de consumir los mensajes del sensor, traducirlos, y enviar los datos a la plataforma de control. Se utilizan los métodos de decodificación proporcionados por *dokuos-modbus-v1*.
- **iotsens-ingestion-dokuos:** Es un servicio que repite su ejecución en un intervalo establecido. El servicio recibe de la API de comandos una lista de todos los comandos disponibles que acepta el sensor. A continuación, envía a la API de comandos una lista de todos los sensores registrados y los parámetros necesarios para construir el mensaje de

petición de datos al sensor.

- **iotsens-interoperability-catalog-api:** También conocida como la API de comandos, se encarga de proporcionar información sobre comandos disponibles a otros servicios, y del envío de mensajes a sensores. Una vez recibidos los parámetros de *iotsens-ingestion-dokuos*, utiliza el codificador de la librería de *dokuos-modbus-v1*, para construir los mensajes en un formato comprensible por el sensor. Y a continuación se publican en una cola de RabbitMQ, para que puedan ser consumidos por el sensor.

A.3. Implementación del proyecto

A excepción de las librerías de codificación y decodificación, que se tuvieron que implementar desde cero, el resto de servicios ya existían previo a empezar el desarrollo en este proyecto. En esos servicios se implementaron las funciones adicionales para que pudieran utilizar los mensajes del protocolo Modbus.

El decodificador recibe los datos como una lista de bytes. Esos datos se transforman a hexadecimal, y se extrae el tipo de función del mensaje, ya que es necesario para saber como decodificar la información. A continuación se crea una lista de medidas, que contendrán los datos recibidos, y serán enviados a la plataforma de control.

El codificador recibe los parámetros, y genera el mensaje en formato hexadecimal. El servicio de mensajería de RabbitMQ que se utiliza para la conexión con el servidor solo permite el envío de mensajes de texto, y el sensor solo acepta una lista de bytes. Así que se necesita transformar el mensaje de hexadecimal a una palabra en codificación ASCII. Para codificar el mensaje siguiendo el protocolo, se crea el mensaje en formato hexadecimal, a partir de los parámetros recibidos por *iotsens-ingestion-dokuos*, y se transforman a una cadena en formato válido para RabbitMQ (ver Figura A.2).

```
private String read0XSimple(Map<String, String> parameters){
    StringBuilder message = new StringBuilder();
    message.append(paddedValue(parameters.get(MODBUS_SLAVE_ID), width: 2));
    message.append(paddedValue(parameters.get(MODBUS_FUNCT_CODE), width: 2));
    message.append(paddedValue(parameters.get("MODBUS_REGS_OFFS"), width: 4));
    message.append(paddedValue(parameters.get("MODBUS_REGS_NUM"), width: 4));
    message.append(getCRCfromBytes(Task.toBytes(message.toString())));

    return hexToAscii(message.toString());
}
```

Figura A.2: Fragmento del código del codificador de mensajes.

A.4. Problemas en la implementación

La implementación del proyecto era relativamente fácil, ya que la empresa ya tiene diseñado el sistema para la comunicación con sensores. Solamente había que expandir la estructura existente para permitir la comunicación con sensores de protocolo Modbus.

El único problema en la implementación ocurrió con la transformación de datos entre ASCII y binario. El sensor enviaba una lista de datos en binario, que al entrar en RabbitMQ, se transformaban a una palabra en ASCII. Esa palabra se tenía que transformar otra vez a una lista de bytes. La codificación en ASCII asocia un carácter distinto a un valor que varía entre 0 y 254. Mientras que Java almacena los bytes usando un valor que varía entre -128 y 127 [14]. La conversión entre estos dos formatos no se realizaba correctamente, y se perdía información. Para solucionarlo se tuvo que implementar un sistema que tratara con este caso específico, pero eso afectaba al resto del sistema de mensajería ya implementado. Así que se creó un caso específico para manejar mensajes de tipo Modbus, separado del resto de mensajes de la empresa.