



UNIVERSIDAD DE GRANADA

TRABAJO FIN DE MÁSTER

Servicio de planificación de tareas de monitorización en nodos IoT en tiempo real

Autor

Luis Miguel Aguilar González (estudiante)

Director/es

Juan Antonio Holgado Terriza (tutor)



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

Granada, 13 de julio de 2024

Servicio de planificación de tareas de monitorización en nodos IoT en tiempo real

Autor

Luis Miguel Aguilar González (estudiante)

Director/es

Juan Antonio Holgado Terriza (tutor)

Servicio de planificación de tareas de monitorización en nodos IoT en tiempo real

Luis Miguel Aguilar González (estudiante)

Palabras clave: monitoriza, IoT, sensores, métricas, HTML, CSS, JavaScript, Angular, Java, Spring Boot, MongoDB

Resumen

El objetivo de este trabajo consiste en el desarrollo de un nuevo servicio de planificación de tareas de tiempo real que se ejecutan en dispositivos IoT. El sistema permite la creación y configuración de tareas, así como la puesta en marcha de dichas tareas en un dispositivo sobre un sistema operativo de tiempo real. Por otra parte, como un sistema IoT, monitoriza métricas de cada dispositivo en función de los sensores que se conecten a las tareas de tiempo real. Este proyecto se ha desarrollado con las últimas tecnologías frontend (HTML, CSS y JavaScript) dentro del framework de Angular y en el backend con Java y Spring Boot, además de una base de datos no relacional en MongoDB.

Real-Time Task Scheduling Monitoring Service on IoT Nodes

Luis Miguel Aguilar González (estudiante)

Keywords: monitors, IoT, sensors, metrics, HTML, CSS, JavaScript, Angular, Java, Spring Boot, MongoDB

Abstract

The aim of this work is to develop a new service for scheduling real-time tasks running on IoT devices. The system allows the creation and configuration of tasks, as well as the launching of these tasks on a device over a real-time operating system. Moreover, as an IoT system, it monitors metrics of each device based on the sensors that are connected to the real-time tasks. This project has been developed with the latest frontend technologies (HTML, CSS and JavaScript) within the Angular framework and in the backend with Java and Spring Boot, as well as a non-relational database in MongoDB.

Yo, **Luis Miguel Aguilar González (estudiante)**, alumno del **MÁSTER DE DESARROLLO DEL SOFTWARE**, con DNI **20887213Q**, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Máster en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Además, el trabajo está publicado en github en la URL <https://github.com/laguilarg99/RITA> con licencia Apache License 2.0 para que cualquiera se sienta con libertad de usarlo, estudiarlo, compartirlo y modificarlo.

Fdo: *Luis Miguel Aguilar González (estudiante)*

Granada a 13 de julio de 2024.

D. **Juan Antonio Holgado Terriza (tutor)**, Profesor del Área de Lenguajes y Sistemas Informáticos del Departamento Lenguajes y Sistemas Informáticos de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado *Servicio de planificación de tareas de monitorización en nodos IoT en tiempo real*, ha sido realizado bajo su supervisión por **Luis Miguel Aguilar González (estudiante)**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 13 de julio de 2024.

El director:

Juan Antonio Holgado Terriza (tutor)

Agradecimientos

Quiero dedicar este trabajo y todo el esfuerzo que le he puesto, a mi tutor Juan Antonio Holgado Terriza por su labor como guía y docente durante todo su desarrollo. También a mis padres y a mi pareja, Elena, que me apoyan cada día para ser una mejor persona y luchar por todos mis sueños.

Índice general

1. Introducción	1
1.1. Introducción	1
1.2. Motivación	2
1.3. Objetivos	4
1.4. Contexto de la investigación	5
1.5. Estructura del trabajo	7
2. Conceptos y estado del arte	9
2.1. Los sistemas de tiempo real	9
2.2. Definición, características y tipos	10
2.2.1. Definición de sistemas de tiempo real y su clasificación	11
2.2.2. Tipos de tareas de tiempo real	12
2.2.3. Parámetros para la definición del comportamiento temporal del sistema	13
2.3. Planificación de sistemas de tiempo real	14
2.3.1. Estudio de planificabilidad de un conjunto de tareas: Test de garantía	14
2.3.2. Algoritmos de planificación	15
Rate Monotonic (RMS)	16
Deadline Monotonic (DM)	16
Earliest Deadline First (EDF)	16
Least Laxity First (LLF)	17
2.3.3. Algoritmo de planificación EDF: Estudio en profundidad	17
2.4. La nube: Definición y proveedores	20
2.4.1. Proveedores	20
Amazon Web Services (AWS)	20
Azure Cloud	20
Google Cloud (GCP)	20
2.4.2. IaC	21
3. Servicio administrador de tareas IoT de tiempo real: RITA	23
3.1. Requerimientos del sistema	23
3.1.1. Requisitos funcionales	24
3.1.2. Requisitos no funcionales	29

3.1.3. Restricciones semánticas	31
3.2. Modelo conceptual del sistema	34
3.2.1. Devices	36
3.2.2. Metrics	37
3.2.3. Organizations	38
3.2.4. Sensors	38
3.2.5. Tasks	38
3.2.6. Users	39
3.3. Arquitectura del sistema	40
3.4. Diseño de la interfaz de usuario	43
3.5. Implementación	49
3.5.1. Capa de Visualización	49
Justificación de la tecnología usada	49
Componentes del Frontend	50
Descripción del funcionamiento de RITA	50
Seguridad, Privacidad y Gestión de los modelos	52
3.5.2. Capa de Procesamiento	53
Justificación de las tecnologías usadas	53
Estructura, comunicación, Seguridad y Modelos.	53
3.5.3. Capa de Dispositivos	57
3.6. Pruebas	62
3.7. Despliegue y puesta en marcha	63
3.8. Resultados	65
4. Conclusiones	69
4.1. Revisión de los objetivos iniciales y conclusiones obtenidas	69
4.2. Trabajos Futuros	70
Bibliografía	72
A. Planificación y gestión del proyecto	73
A.1. Metodología/Ciclo de vida	74
A.2. Herramientas utilizadas	75
A.2.1. MongoDB	75
A.2.2. Java con Spring Boot	76
A.2.3. Angular	77
A.3. Gestión del proyecto. Planificación temporal. Diagrama de Gantt	78
A.3.1. Iteraciones del desarrollo	78
A.3.2. Planificación temporal	93
Sprint Review 1 - 15 nov 2023	93
Sprint Review 2 - 29 nov 2023	93
Sprint Review 3 - 13 dic 2023	93
Sprint Review 4 - 27 dic 2023	93
Sprint Review 5 - 12 ene 2024	94

Sprint Review 6 - 29 ene 2024	94
Sprint Review 7 - 12 feb 2024	94
Sprint Review 8 - 28 mar 2024	95
Sprint Review 9 - 11 abr 2024	95
Sprint Review 10 - 25 abr 2024	95
Sprint Review 11 - 27 may 2024	95
Sprint Review 12 - 14 jun 2024	95
A.3.3. Lista de tareas	96
A.3.4. Diagrama de Gantt	97
B. Innovación	101
B.1. Estudio del Mercado	101
B.1.1. Segmentos de Clientes	101
B.1.2. Competidores	102
AWS IoT	102
Azure IoT	102
B.1.3. Análisis DAFO - FODA	103
Debilidades de la empresa	103
Amenazas	103
Fortalezas	103
Oportunidades	104
B.1.4. Propuesta de Valor	104
B.1.5. Recursos Clave	104
B.1.6. Actividades Clave	105
B.1.7. Socios y Alianzas Clave	105
B.1.8. Estructura de Costes	105
Costes de la solución actual	105
Costes del producto final	106
B.1.9. Vías de Ingresos	106
B.1.10. Canales de Distribución	106
B.1.11. Relación con los Clientes	106
B.1.12. Descripción del Producto Mínimo Viable	106
B.1.13. Roadmap de Desarrollo	107

Índice de figuras

2.1. Imagen de los parámetros del sistema	13
2.2. Ejemplo de planificación EDF	19
3.1. Diagrama de casos de uso	28
3.2. Modelo conceptual con un modelo entidad-relación	35
3.3. Conceptos principales del sistema	36
3.4. Diagrama UML del modelo conceptual	40
3.5. Arquitectura del sistema	41
3.6. Interfaz de usuario dashboard	43
3.7. Interfaz de usuario para crear dispositivos/organizaciones	44
3.8. Login	45
3.9. Página principal	46
3.10. Creación de dispositivos	46
3.11. Formulario de creación de tareas	47
3.12. Gráfico de métricas	48
3.13. Ejemplo de dispositivo	55
3.14. Ejemplo de métrica	55
3.15. Ejemplo de organización	55
3.16. Ejemplo de sensor	55
3.17. Ejemplo de tarea	56
3.18. Ejemplo de usuario	56
3.19. Comunicación entre los módulos del dispositivo	61
3.20. Logs del dispositivo	65
3.21. Gráfica Resultado	66
3.22. Formulario de la tarea	67
A.1. Interfaz de usuario de MongoDB	76
A.2. Gráficas de rendimiento en MongoDB	77
A.3. Código de la aplicación de ejemplo	78
A.4. Ejecución de la aplicación de ejemplo	79
A.5. Ejecución de la aplicación de ejemplo avanzada	79
A.6. Detalle de la ejecución de la aplicación de ejemplo avanzada	80
A.7. Detalle de la instalación de MongoDB	81

A.8. Nueva conexión de MongoDB	81
A.9. Nuevo proyecto en Angular	82
A.10.Creando nuevo componente en Angular	83
A.11.RabbitMQ Manager	91
A.12.RabbitMQ Manager página principal	92
A.13.Diagrama de Gantt para el proyecto desde noviembre de 2023 hasta junio de 2024.	98
A.14.Diagrama de Gantt desde Jira	99
B.1. Diagrama de Devsecops	107

Índice de tablas

2.1. Parámetros del sistema	13
2.2. Tabla de valores	18
3.1. RF1 - Registrar usuario	24
3.2. RF2 - Modificar usuario	24
3.3. RF3 - Eliminar usuario	24
3.4. RF4 - Añadir organización	25
3.5. RF5 - Modificar organización	25
3.6. RF6 - Eliminar organización	25
3.7. RF7 - Añadir dispositivo	25
3.8. RF8 - Modificar dispositivo	25
3.9. RF9 - Eliminar dispositivo	26
3.10. RF10 - Añadir tarea	26
3.11. RF11 - Modificar tarea	26
3.12. RF12 - Eliminar tarea	26
3.13. RF13 - Crear sensor	27
3.14. RF14 - Modificar sensor	27
3.15. RF15 - Eliminar sensor	27
3.16. RF16 - Visualizar métricas	27
3.17. RF17 - Actualizar los datos	27
3.18. RNF1 - Rendimiento del sistema para crear los distintos elementos	29
3.19. RNF2 - Rendimiento del sistema para crear las tareas	29
3.20. RNF3 - El sistema debe aguantar la existencia de usuarios enviando pe- ticiones a la aplicación a la vez.	29
3.21. RNF4 - El sistema debe estar siempre disponible al tratarse de sistemas críticos para el usuario.	30
3.22. RNF5 - El sistema debe ser seguro en cuanto a los datos sensibles del usuario y del sistema.	30
3.23. RNF6 - El sistema debe ser seguro en cuanto a los datos.	30
3.24. RNF7 - El sistema debe ser escalable horizontal y verticalmente.	30
3.25. RNF8 - Rendimiento del sistema para actualizar los distintos elementos.	30
3.26. RNF9 - Rendimiento del sistema para eliminar los distintos elementos.	30
3.27. RS1 - Es necesario que no exista otro usuario idéntico	31

3.28. RS2 - Es necesario que no exista otra organización igual	31
3.29. RS3 - Es necesario que no exista otro dispositivo igual dentro de una misma organización	31
3.30. RS4 - La tarea debe ser planificable para poder ser añadida a un dispositivo	32
3.31. RS5 - Para poder eliminar un dispositivo no puede tener tareas previamente	32
3.32. RS6 - Para poder eliminar una organización hay que eliminar todos los dispositivos previamente	32
3.33. RS7 - Es necesario que no exista otra tarea idéntica	32
3.34. RS8 - Es necesario que exista un usuario para crear una organización . . .	32
3.35. RS9 - Es necesario que exista una organización para crear un dispositivo .	33
3.36. RS10 - Es necesario que exista un dispositivo para crear un sensor	33
3.37. RS11 - Es necesario que exista un sensor y un dispositivo para crear una tarea	33

Capítulo 1

Introducción

En este capítulo se incluye una descripción del trabajo, su motivación, objetivos que se persiguen, la estructura que tendrá el mismo, así como una planificación y estimación de costes.

1.1. Introducción

Vivimos cada vez en un mundo más interconectado, hasta el punto de que llevar un ordenador de gran potencia en la mano con conexión a internet se ha convertido en un evento cotidiano. Y no solo los dispositivos móviles que llevamos con nosotros a todas partes, sino que los electrodomésticos que usamos en casa son cada vez más inteligentes y cuentan con distintas conexiones que nos posibilitan manejarlos a distancia.

En este entorno en el que nos vemos envueltos por la conectividad, nace un concepto revolucionario conocido como, el Internet de las cosas o como nos referiremos en adelante, IoT (Internet of Things) [1], concepto que pretende elevar aún más los límites de esta interconexión y automatización. El término IoT sienta un precedente no solo para mejorar la comunicación entre nuestros dispositivos o aumentar las posibilidades de automatización sino que además aumenta drásticamente la recopilación y análisis de datos a gran escala. En este trabajo se pretende aprovechar el potencial de este concepto, en auge y en continua evolución, comunicando múltiples dispositivos con una aplicación en la nube.

La nube, en el contexto del IoT, tiene el objetivo de centralizar el almacenamiento y el procesamiento de los datos que están disponibles a través de la red que IoT proporciona. En lugar de almacenar los datos y de procesarlos en los distintos dispositivos que normalmente tienen una cantidad de recursos reducidos, se utilizan estos servidores remotos que ofrecen las principales soluciones Cloud; Amazon Web Services (AWS), Microsoft Azure o Google Cloud Platform (GCP), aunque en ocasiones se pueden utilizar servidores dedicados propios. Es sorprendente como todos estos servicios reconocen el potencial del internet de las cosas y es por ello por lo que ofrecen servicios especializados para este tipo de implementaciones como pueden ser en el caso de AWS con AWS IoT Analytics para la parte del análisis de datos y AWS Greengrass para el Edge Computing. Microsoft, por su parte, ofrece, Azure IoT Central y Google, Google Cloud IoT Core. Estos servicios permiten a las empresas desarrollar, desplegar y escalar sus soluciones IoT de la manera más ágil posible permitiendo la implementación de sistemas de grandísima complejidad con un gran número de dispositivos. Además, estos servicios no solo ofrecen esta facilidad en el escalado y la integración de nuestra aplicación, sino que ofrecen otros múltiples servicios de análisis de datos y computación que permiten al cliente (en nuestro caso a nosotros), no tener que preocuparse por la gestión de la infraestructura y centrarse en el desarrollo de la aplicación.

1.2. Motivación

Este proyecto presenta distintos aspectos que hacen interesante su desarrollo, tanto por el valor que puede aportar a la comunidad científica como por otros motivos que aquí expreso:

En primer lugar, a nivel profesional, trabajo en un proyecto donde el IoT cumple un papel fundamental, conocer más de cerca cómo funciona este paradigma de la informática me permitiría continuar de una manera más activa en el éxito del mismo, ampliando mi experiencia y habilidades en un área que como ya hemos comentado está en continua evolución, de esta manera puedo adquirir conocimientos relacionados a como se conectan y comunican entre sí los dispositivos y que soluciones y posibilidades pueden ofrecer. Lo que hace este proyecto aún más interesante para mi crecimiento profesional es la componente de tiempo real, pues la combinación de IoT y los sistemas de tiempo real dan la posibilidad de ofrecer soluciones de altísima calidad para sistemas críticos donde la respuesta procede, de sistemas IoT caracterizados, en la mayoría de los casos, por tener bajos recursos.

En segundo lugar, nuestro proyecto pretende desarrollar una aplicación para la gestión de dispositivos de tiempo Real en el marco del paradigma IoT y las tecnologías Cloud, como tal, no es algo novedoso por la parte del IoT, gestionar este tipo dispositivos ya es algo que implementan la mayoría de los fabricantes de teléfonos que también cuentan con un entorno de dispositivos inteligentes en los que se pueden asignar distintas tareas como reproducir contenido multimedia o encender la lavadora, sin embargo, este tipo de sistemas no funcionan en tiempo real, en ellos las peticiones que se realizan son totalmente asíncronas y aunque se intenta que la respuesta sea lo más rápida posible, no se vigila el hecho de que estas tengan un tiempo límite para su finalización, de hecho es común con algunos asistentes inteligentes de estos ecosistemas que a veces respondan con un mensaje como el siguiente: “no he podido encontrar resultados para lo que me preguntas”, esto es inconcebible según el tipo de sistema de tiempo real del que estemos hablando como se detallará en el capítulo 2.

Otra de las razones por las que este trabajo me parece interesante es por el desafío técnico que puede suponer la gestión de los dispositivos con bajos recursos en los que habrá que tener en cuenta probablemente, desde el consumo energético hasta el uso de la CPU y además hacer uso de estos datos para estudiar la planificabilidad de las tareas si se espera añadir alguna más. Abordar todo esto me brindará también una serie de conocimientos teóricos desde el un entorno práctico que incluirá el desarrollo de una interfaz de usuario con algunos de los principales frameworks de desarrollo web, el despliegue de la propia aplicación y la comunicación que se realizará desde esta con los distintos dispositivos además del desarrollo de un backend que deberá tener distintos endpoints para la planificabilidad del sistema, la gestión y control de dispositivos entre otros.

Por último, este trabajo, también nos permitiría abordar temas como la seguridad y la privacidad en IoT, en general, ser capaces de mantener toda la robustez de los sistemas habituales, pero teniendo en cuenta las limitaciones que tenemos y que ya he mencionado. Es un gran reto asegurar la seguridad de los datos que se están enviando, y la velocidad en su transmisión, así como cumplir con las limitaciones temporales de las tareas ejecutadas en los dispositivos si se interrumpen o en general, el rendimiento, mermado por dicha comunicación.

En conclusión, esta sección el proyecto, presenta una gran oportunidad para mi crecimiento personal y profesional. Además, el aporte a la comunidad científica es significativo pues permite evaluar las fronteras entre los sistemas de tiempo real y el paradigma de IoT junto con las tecnologías en la nube.

1.3. Objetivos

El objetivo principal de este proyecto se centra en **el desarrollo de un nuevo servicio de planificación de tareas de monitorización que se ejecutan en dispositivos IoT en tiempo real**. Esto permitirá crear, gestionar y supervisar las tareas de tiempo real que se ejecutan en dichos dispositivos.

Para conseguir dicho objetivo necesitamos cumplir con los siguientes objetivos específicos:

1. Explorar soluciones académicas o del mercado que faciliten la gestión de tareas de tiempo real en dispositivos IoT
2. Comprender el funcionamiento de los sistemas de tiempo real y los aspectos más significativos de la planificación de tareas en sistemas operativos.
3. Seleccionar la infraestructura más adecuada para la plataforma de gestión de tareas de tiempo real en dispositivo IoT
4. Desarrollar e implementar el servicio de planificación de tareas de tiempo real como un servicio cloud que se conecta a los dispositivos IoT
5. Determinar qué tipo de dispositivos pueden actuar como nodos IoT del servicio de planificación.
6. Aplicar buenas prácticas de ingeniería del software en el desarrollo de la plataforma.
7. Realizar un piloto para que un grupo de usuarios puedan probar la validez de la nueva plataforma desarrollada

1.4. Contexto de la investigación

Aunque el internet de las cosas [2] [3] pueda parecer un concepto abstracto y relativamente futurista, fue ya por 1999 cuando el profesor del MIT, Kevin Ashton, mencionó por primera vez este concepto en una conferencia. Sin embargo, no fue hasta 2009 que se dio a conocer ampliamente como IoT. Principalmente, la implementación del IoT ha contado con 2 grandes aliados:

- Las redes inalámbricas. Cada vez tienen más capacidad de transmisión de datos.
- La aparición de tecnologías como el 5G. Aunque no se ha implementado de manera real a gran escala, ha demostrado ser una solución viable para comunicaciones a altísima velocidad disminuyendo el que puede ser uno de nuestros problemas, la latencias en la comunicación (aunque por nuestros casos de uso probablemente no acabemos sufriendo las consecuencias de estas).

Por otro lado, Cloud computing [4], no fue hasta 2006 cuando se puso en circulación dicho concepto, cuando George Gilder expuso un modelo por el cual las empresas ofrecían un servicio en el que estas manejan la infraestructura, virtualizando los recursos y pasando estos a ser parte de la nube virtual, de esta manera el usuario los puede gestionar automáticamente sin necesidad de comprar nuevo hardware. En la nube se pueden encontrar distintas capas: SaaS, IaaS y PaaS.

- **SaaS (Storage as a Service)**, ofrece espacio de almacenamiento solamente.
- **IaaS (Infrastructure as a Service)**, es un modelo de negocio por el cual una empresa contrata a un equipo para que se encargue de gestionar todas las operaciones.
- **PaaS (Platform as a Service)**, es una forma de alquiler de hardware en sí.

Amazon ofrece distintos cursos de formación para que las personas aprendan a gestionar sus servicios, aunque la experiencia de usuario es muy buena, manejar la infraestructura como he comentado anteriormente, es complejo pues necesita de cierto nivel de conocimiento. Es interesante la comparativa que hace durante las primeras fases de la formación con una cafetería, para poder apreciar su potencial. Cuando una persona empieza el negocio, tendrá pocos clientes con una cafetera y un barista bastará, pero, si el negocio se hace más famoso, se necesitará de más personal o de más máquinas de cafe.

AWS o cualquier proveedor de servicios en la nube, permitirá escalar la infraestructura bajo demanda, cuando los propios negocios la manejaban, debían comprar los equipos y configurarlos y si ya no eran necesarios, el gasto no sería suficiente rentable. La nube permite adaptar nuestros recursos a las necesidades del momento. Por ejemplo, si trabajamos en una aplicación de venta de tickets, probablemente necesitaremos más cantidad de recursos los días antes de un concierto o evento, que el resto del año. Por último, Los sistemas de tiempo real por su lado debido a toda la conectividad y exigencias que presentan las tecnologías que acabo de comentar, se estima que para el 2025 habrá 79,4 zettabytes de datos y casi el 30 % requerirán procesamiento en tiempo real mediante sistemas de este tipo. Este procesamiento es fundamental en empresas de robótica, fabricación, atención médica, industrias que necesitan de alta precisión, entre otras, por los beneficios que aporta su implementación, ya que puede ser una sincronización más precisa, una mayor previsibilidad y confiabilidad y una priorización de las tareas que debe realizar nuestro sistema.

En conclusión, estamos en el contexto perfecto para el desarrollo de este trabajo, se trata de un entorno industrial con muchos sensores de los que recogeremos multitud de datos, y en el que tendremos por supuesto la infraestructura ofrecida por la nube y con las latencias lo más bajas posibles gracias a las facilidades tecnológicas de las que nos proveen las redes actuales. Incluso si la implementación fuese real en una fábrica, permitiría el despliegue de una red local exclusiva de alta velocidad, ya que existen empresas que ofrecen este tipo de servicios red. Además, con las previsiones del aumento de datos de los sistemas de tiempo real y su uso cada vez mayor para tareas críticas, proponen un ambiente perfecto para que nuestro proyecto tenga cierto valor en la comunidad empresarial y sienta las bases para futuras investigaciones relacionadas con una mejor gestión de estos sistemas, de manera más eficiente y visual desde el punto de vista del usuario.

1.5. Estructura del trabajo

Como cualquier trabajo de investigación, se pretende exponer el porqué de las decisiones tomadas, mostrar cómo se ha realizado el desarrollo del mismo, exponer y comparar los resultados obtenidos con otros trabajos ya similares y extraer todas las conclusiones posibles, así como sentar las bases de los futuros trabajos que se podrían realizar como consecuencia de este.

En este primer apartado estamos tratando una introducción al trabajo, donde se comenta lo que se va a estudiar en este trabajo principalmente y que se puede esperar de él, además ayudará a entender el núcleo del trabajo, las motivaciones de este y el contexto en el que se desarrolla, así como una breve mención a los paradigmas y tecnologías en los que se desarrollará.

En el segundo apartado, sentaré todas las bases teóricas, mientras que en el tercer capítulo se hablará de cuales son los requerimientos del sistema y que justifican los diagramas y diseños que se exponen en la misma.

En un cuarto apartado, se desarrollarán distintas conclusiones así como los trabajos futuros.

Por último, se mostrarán los resultados, como se recogen los datos de cada uno de los sensores IoT y si el sistema es realmente útil y viable.

Por otro lado contaremos con un apéndice que tendrá toda la información necesaria con las especificaciones más técnicas del desarrollo del proyecto.

Capítulo 2

Conceptos y estado del arte

2.1. Los sistemas de tiempo real

En los últimos años ha crecido mucho el uso de los sistemas empotrados porque la necesidad de automatización de los procesos ha crecido al mismo ritmo que la tecnología. Un ejemplo de esto es el trabajo que ha puesto Amazon en sus almacenes tras la adquisición en 2012 de Kiva Systems, para de esta manera agilizar los procesos, mejorar la seguridad y aumentar la eficiencia a través de la robotización, para así ser capaces de almacenar incluso un 40 % más de inventario. Como se puede deducir por todo ello, el interés por los sistemas de tiempo real también a aumentado, pero a pesar de ello muchas veces se confunde este concepto con sistemas que simplemente tienen una respuesta rápida, sin embargo, cuando hablamos de tiempo real nos referimos a sistemas que son capaces de ofrecer una respuesta en un tiempo acotado.

En multitud de ocasiones, las empresas, como el ejemplo de Amazon previamente mencionado [5] [6], necesitan de la integración de multitud de dispositivos dando lugar a sí al Internet de las cosas (IoT), refiriéndose con este concepto a, **una red colectiva de dispositivos conectados y a toda la tecnología que hace posible la comunicación entre los mismos e incluso con la nube**. El Internet de las cosas pretende integrar los dispositivos de consumo, sin embargo, siempre había sido complicado por el tamaño de los chips pero con el paso del tiempo, su tamaño y coste disminuyó lo que hizo posible que en la actualidad la mayoría de los dispositivos de uso diario tengan de alguna manera conexión a internet. El caso de uso habitual para estos dispositivos es que un usuario con acceso a la nube a través de una aplicación móvil pueda gestionar los distintos dispositivos que tiene en casa, por ejemplo.

Es el caso de Nespresso, que permite con sus últimos modelos de cafeteras gestionar las mismas y que el usuario reciba notificaciones de ellas a través de la aplicación móvil.

La unión de los conceptos de IoT junto con el de los sistemas de tiempo real da lugar a sistemas de control mucho más robustos, pensados para los dispositivos que podríamos encontrar en una fábrica (aunque también podría tener finalidad doméstica dado el caso). Dando acceso a los usuarios de manera fácil a través de una aplicación web para controlar los dispositivos de dicha fábrica y estos a su vez, a través del IoT serán capaces de mandar datos a la nube siempre cumpliendo con el periodo de tiempo designado para cada uno de los procesos (ya sean sus funciones propias o el envío de datos a la nube).

Los lenguajes de programación por su parte han evolucionado siempre adaptándose a la complejidad a la que se han elevado los problemas con el paso del tiempo. En esto, Java surgió como un lenguaje que se podía adaptar a cualquier problema fácilmente como pudiera ser el desarrollo de aplicaciones web hasta los sistemas empotrados[7] y el internet de las cosas, ya que su conocida portabilidad gracias al JVM (Máquina Virtual Java), permite la ejecución del mismo desarrollo en distintos sistemas, sin importar su arquitectura, dando lugar a soluciones escalables y flexibles. Para los sistemas en los que, además, los recursos son limitados, apareció RTSJ, Java Real-Time Specification [8][9], centrándose, en que, como se puede deducir por el nombre, en la ejecución en sistemas de tiempo real donde cada una de las tareas, deben finalizar en un periodo de tiempo determinado.

2.2. Definición, características y tipos

En esta sección del trabajo trataré los principales conceptos relacionados con los sistemas de tiempo real [10], como son la clasificación de tareas, una breve mención algoritmos de planificación centrándonos principalmente en el algoritmo EDF más adelante que será el que usemos para estudiar la viabilidad de planificación de las tareas dentro de los distintos dispositivos de IoT que conformarán nuestro sistema y que tendrán pocos recursos pues se supone que serán pequeños dispositivos dentro de una fábrica y que gracias a los datos que extraen de sus distintos sensores podrán enviarlos a la aplicación web a la que el usuario tendrá acceso a través de cualquier dispositivo. En un sistema concurrente, no es necesario especificar el orden exacto con el que se ejecutan los procesos pues ellos mismos se sincronizan de manera que se imponen restricciones en el orden de ejecución, la salida será la correcta a pesar de la gestión interna que se realice por no ser un sistema determinista. En un sistema de tiempo real se restringe este indeterminismo seleccionando el orden de cumplir las tareas de manera que se cumplan las restricciones temporales, denominándose este proceso de selección, planificación. Para realizarla deberemos saber que existen distintos tipos de tareas y que se clasificarán en ellos dependiendo de las características que posean.

2.2.1. Definición de sistemas de tiempo real y su clasificación

“Los sistemas de tiempo real son aquellos sistemas en los que un resultado correcto no depende solo de la lógica sino del tiempo en el que esos resultados fueron producidos” (Stankovic 1998). Este tipo de sistemas se pueden implementar desde sistemas empotrados de pocos recursos que se componen de un pequeño microprocesador hasta sistemas de gran potencia de cómputo como sistemas de control que tienen que procesar grandes cantidades de datos.

La mayoría de los sistemas embebidos, consisten de una pequeña CPU como un frigorífico o un coche (aunque los coches últimamente tienen gran capacidad de procesamiento como es el caso de los vehículos Tesla que necesitan de esta para el piloto automático que poseen con reconocimiento de objetos de tiempo real). Este tipo de sistemas cuyo resultado depende del tiempo de ejecución de las tareas se pueden clasificar en los siguientes:

- **Sistemas de tiempo real con restricciones de tiempo duros (Hard real-time systems).** Son aquellos sistemas en los que la ejecución de una tarea puede tener consecuencias críticas si no se hace a tiempo.
- **Sistemas de tiempo real con restricciones de tiempo suaves (soft real-time systems).** Aquí la ejecución de tareas puede cumplirse una vez pasado el límite de tiempo sin consecuencias para el sistema, aunque se espera que esto solo ocurra esporádicamente.
- **Sistemas de tiempo real firme (firm real time).** Permite que los procesos no se ejecuten a tiempo, pero esto no implica ningún beneficio pues si no se ha ejecutado a tiempo, su ejecución queda descartada como por ejemplo ocurriría en un sistema multimedia.

Es importante por esto conocer que tipos de tareas pueden componer nuestro sistema para después hablar de cómo se pueden planificar las mismas según el tipo de sistema al que pertenezcan.

2.2.2. Tipos de tareas de tiempo real

Podemos considerar que las tareas son un conjunto de trabajos que se repiten. Normalmente, no todas las tareas tienen la misma importancia a la hora de cumplir los plazos estipulados, dentro de nuestro sistema podrán existir por tanto tareas críticas, menos críticas y tareas que incluso no tendrán plazo de ejecución. Por ello, atendiendo a estas características temporales tendremos:

- **Tareas periódicas.** Se ejecuta periódicamente y en un periodo constante.
- **Tareas esporádicas.** Son igual que las tareas periódicas pero el plazo de finalización estricto por lo que se ejecutan esporádicamente.
- **Tareas aperiódicas.** Al contrario que las tareas periódicas no tienen plazo de finalización y si lo tienen no es estricto.

En cuanto a la prioridad de las mismas se pueden clasificar:

- **Tareas críticas.** El fallo de una tarea de este tipo supone un error definitivo para el sistema
- **Tareas opcionales.** Si se retrasa su ejecución no supondría ningún error, este tipo de tareas podrían ser de monitoreo o mantenimiento.

Además, las tareas pueden clasificarse en dos grupos:

- **Opcionales con plazo (hard aperiodic).** Disponen de un tiempo de ejecución recomendado.
- **Opcionales sin plazo (soft periodic).**

2.2.3. Parámetros para la definición del comportamiento temporal del sistema

Si consideramos que el sistema está formado por tareas periódicas o esporádica convertidas en periódicas podemos tener en cuenta los parámetros aquí definidos:

Tabla 2.1: Parámetros del sistema

N	Número de tareas
P_i	Periodo de activación
C_i	Tiempo máximo de ejecución
D_i	Plazo máximo de terminación
R_i	Tiempo de respuesta máximo
Pr_i	Prioridad de la tarea
S_i	Desfase respecto del momento inicial

Cabe señalar que el periodo se entiende como el tiempo mínimo entre dos activaciones consecutivas, el desfase por su parte solo tiene sentido tras la primera activación. En resumen, podemos observar los distintos parámetros en el diagrama que aquí presento:

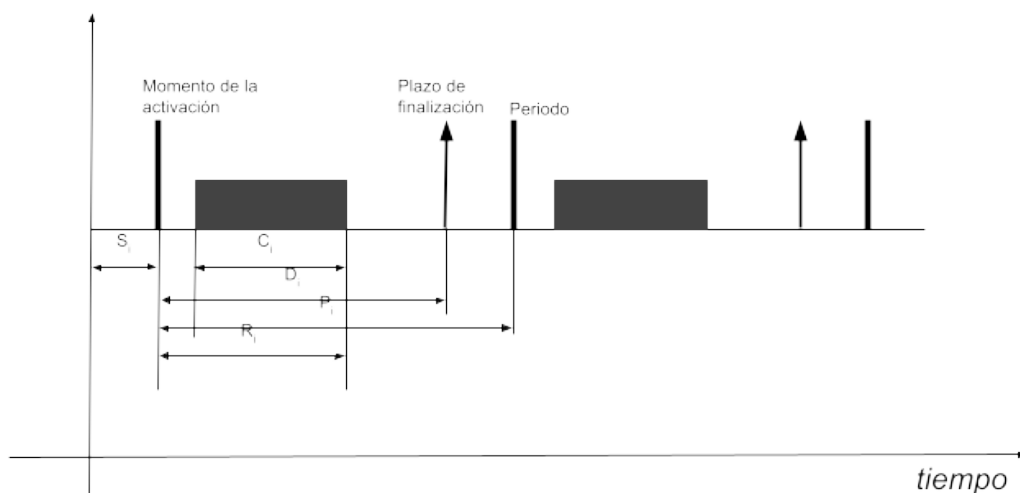


Figura 2.1: Imagen de los parámetros del sistema

2.3. Planificación de sistemas de tiempo real

La planificación del sistema es el procedimiento por el cual se establecen los tiempos de inicio y finalización para un conjunto de tareas cumpliendo con las restricciones de tiempo, prioridad y recursos. Lo que en conjunto se define como política de planificación designa un cúmulo de aspectos que permiten la implementación de este tipo de sistemas, como son el propio **algoritmo de planificación**, que determina el orden de las tareas (normalmente se habla de este como planificador), el **test de garantía**, por su lado se hace uso de este para asegurar que las tareas cumplirán con las restricciones de tiempo.

2.3.1. Estudio de planificabilidad de un conjunto de tareas: Test de garantía

Dependiendo del algoritmo de planificación [11] [12] que se vaya a utilizar se deberá de aplicar un test de garantía diferente, para el caso más general de un conjunto de tareas no periódico simple Rate Monotonic, RMS, el test de garantía viene dado por un teorema matemático en el que se define que un conjunto de tareas será planificable si se cumple la siguiente desigualdad:

$$U = \sum_{i=1}^N \frac{C_i}{T_i} < N \left(2^{\frac{1}{N}} - 1 \right)$$

En nuestro caso trabajaremos con el algoritmo de planificación de prioridades dinámicas EDF, también conocido como el Algoritmo del Primer Plazo más Cercano, para este caso específico, en caso de que los plazos de finalización correspondan con el tiempo máximo de ejecución de la tarea, el sistema será planificable sí y solo sí:

$$D = \sum_{i=1}^N \frac{C_i}{D_i} \leq 1 \quad (D = \text{Densidad}) \quad D_i = P_i \Rightarrow D = U$$

El límite de planificabilidad es el uso total de la CPU. En caso de que los plazos de finalización sean menores que los periodos, la condición es solo suficiente, por lo que habrá que aplicar otros métodos en tal caso para determinar con seguridad si el grupo de tareas es planificable. Sabiendo que:

$$D = \{d_{i,k} | d_{i,k} = k \cdot D_i + d_i, d_{i,k} \leq \min(b_p, h), 1 \leq i \leq n, k \geq 0\}$$

Entonces el conjunto de tareas periódicas con plazos menores a los periodos es planificable sí y solo sí (Donde B_k es el periodo de ocupación de un recurso y D_k es el hiperperiodo):

$$\forall_{1 \leq k \leq n} \left(\sum_{i=1}^k \frac{C_i}{D_i} \right) + \frac{B_k}{D_k} \leq 1$$

Más adelante entraremos en mayor nivel de detalle respecto el algoritmo EDF y la planificabilidad de un conjunto de tareas haciendo uso de este junto con un conjunto de ejemplos además de cuál sería el diagrama de flujo en nuestro código.

2.3.2. Algoritmos de planificación

Existen dos tipos de planificadores principalmente [13], [14] planificadores cíclicos, donde las tareas se van a ejecutar siempre en el mismo orden sin ninguna alteración en ellas mientras que la planificación por prioridades permiten la implementación de sistemas complejos, es por ello por lo que en nuestro caso se puede deducir que el planificador que tendremos será por prioridades. Este tipo de planificadores se basan en asignar una prioridad a cada tarea y decidir en cada instante que tarea se debe ejecutar en función de estas, estas prioridades se expresan con un número entero, tomando habitualmente la nomenclatura propuesta por UNIX donde la mayor prioridad 0 es la mayor y la menor será según aumente dicho valor. Dichas prioridades podrán asignarse de manera estática o dinámica, en el caso de que sean estáticas, tendrá la misma prioridad una tarea determinada durante toda la vida del sistema mientras que si dicha prioridad es variable, lo normal es que la prioridad vaya elevándose en función del tiempo que falte para alcanzar el plazo máximo de terminación. También se podrá clasificar el sistema en función de si es expulsivo o no, será expulsivo en caso de que al lanzarse un proceso con mayor prioridad este sea capaz de arrebatarse la CPU a otra tarea de menor prioridad.

En este tipo de sistemas es primordial tener en cuenta algunos de los parámetros que ya he mencionado anteriormente como son, el tiempo de ejecución (C_i), el deadline o plazo máximo de terminación (D_i) y la prioridad de la tarea (Pri). De las diferentes formas de planificación teniendo en cuenta estos parámetros, destacan:

- Prioridades estáticas:
 - Rate Monotonic (RMS)
 - Deadline Monotonic (DM)
- Prioridades dinámicas:
 - Earliest Deadline First (EDF)
 - Least Laxity First (LLF)

Rate Monotonic (RMS)

Este algoritmo se basa en la asignación de prioridades a las tareas en función de la frecuencia en la que estas se ejecutan. RMS (Rate Monotonic Scheduling) [15] se ha probado como uno de los mejores algoritmos dentro de aquellos algoritmos de prioridades estáticas. Este algoritmo es óptimo cuando las tareas tienen periodos y tiempo de ejecución conocidos y el sistema no está sobrecargado.

Deadline Monotonic (DM)

Este es una extensión de RMS, la diferencia es que en el DMS (Deadline Monotonic Scheduling) [16], las prioridades no se asignan según la frecuencia de ejecución, sino que se asignan en función del deadline, cuanto mayor la prioridad menor será el plazo de finalización de la tarea. Las prioridades al igual que en RM son estáticas, siendo esta la principal diferencia con EDF donde estas son dinámicas.

Earliest Deadline First (EDF)

Este algoritmo por su parte es óptimo para la planificación en procesadores mono núcleo. Para poder planificar las tareas de manera dinámica se usa una cola de prioridad basada en el deadline de cada una de las tareas. La tarea que tenga un plazo de finalización menor forzará la espera de aquella que tiene un plazo de finalización mayor. Si hay alguna tarea que ya se esté ejecutando con un deadline mayor dejará que se ejecute la tarea con el plazo más corto.

Least Laxity First (LLF)

En este caso, la prioridad se asigna dinámicamente según la laxitud de las tareas, por lo que la prioridad no se asignará como en el EDF según el plazo de finalización de las tareas, en este caso, la prioridad se asignará según la laxitud de la tarea tal y como he mencionado, esto es el tiempo restante antes del plazo de finalización y menos tiempo para completarse. Cuanto menor laxitud, mayor será la prioridad.

En nuestro caso elegimos EDF específicamente porque ofrece una gran cantidad de beneficios, como la asignación de prioridades dinámicas respecto de DM y RM, una implementación más simple respecto de LLF entre otros beneficios que dependen del contexto en el que se lleve a cabo su ejecución y que se expondrán más adelante en nuestro trabajo para mostrar la idoneidad del algoritmo para nuestro caso.

2.3.3. Algoritmo de planificación EDF: Estudio en profundidad

La planificación dinámica es un enfoque cada vez más usado. Este algoritmo de planificación basado en prioridades dinámicas fue definido en 1973 por Liu y Layland y es óptimo para procesadores mono núcleo con tareas expulsables y periódicas con plazos iguales a los periodos. Así, este algoritmo nos permite planificar sistemas que con prioridades fijas no lo son y sí con prioridades dinámicas [17].

A pesar de que anteriormente he dispuesto distintas fórmulas que podrían asegurar la planificabilidad de este algoritmo en diferentes contextos, para sistemas de alta complejidad no existe un test de planificabilidad exacto por lo que se han desarrollado técnicas pesimistas para los tiempos de respuesta en el peor de los casos. Cuando se posee un sistema compuesto por tareas periódicas, mono procesador, con un planificador expulsor y plazos de la misma periodicidad, el algoritmo EDF es óptimo siendo el límite de utilización de las condiciones descritas de 1 (100 %). Si alguna tarea tuviese un plazo menor a su periodo, se define el siguiente test pesimista, que ya he descrito previamente:

$$\sum_{i=1}^N \frac{C_i}{\min(D_i, T_i)} \leq 1$$

Solo que en este caso en el denominador no será D_i (plazo máximo de finalización para una tarea) sino que será el mínimo entre D_i y un instante de tiempo absoluto T_k . Si dicho test se cumple, se puede asegurar la planificabilidad. Si no se cumple y la utilización del sistema es menor que el 100 %, hay que utilizar un test exacto para determina la planificabilidad.

Aunque la planificación por prioridades fijas es más aceptada hoy, el uso de prioridades dinámicas comienza a tener más relevancia porque multitud de lenguajes como java con su RTSJ lo soportan, además de ciertos sistemas operativos. No todas son ventajas para el algoritmo EDF, los sistemas son mucho más complejos por el cálculo de prioridad en cada activación de las tareas, y en momentos transitorios de sobrecarga donde el sistema supera el 100 % de la utilización, el sistema se vuelve impredecible. También, en sistemas de prioridad fija podríamos asegurar que las tareas se van a ejecutar dentro de los respectivos plazos, sin embargo, para EDF esta predicción no es exacta por lo que hay que usar técnicas adicionales para asegurar la planificabilidad de las tareas más prioritarias, aumentando como ya he mencionado, la complejidad del sistema.

Entre las propiedades del EDF encontramos:

- La prioridad más alta la tendrá la tarea cuyo deadline o plazo de finalización es más cercano (como ya he mencionado en otro momento).
- Si dos tareas tienen deadlines iguales, se elegirá como más prioritaria una de las dos aleatoriamente.
- Las prioridades al ser dinámicas cambian de una ejecución a otra.
- Se puede definir que la prioridad es fija a nivel de cada ejecución.

A partir de un conjunto de tareas podemos definir un pequeño ejemplo para representar el funcionamiento de este algoritmo:

Tabla 2.2: Tabla de valores

	T	D	C	U
T_1	30	30	10	0.333
T_2	40	40	10	0.250
T_3	50	50	12	0.240

Como la suma de la columna U es igual a $0.823 \leq 1$ ($U \leq 1$) la condición se cumple por lo que podemos asegurar que el sistema es planificable y se garantizarán los plazos para nuestro algoritmo. A continuación, se podría ver cómo sería la ejecución de dichas tareas:

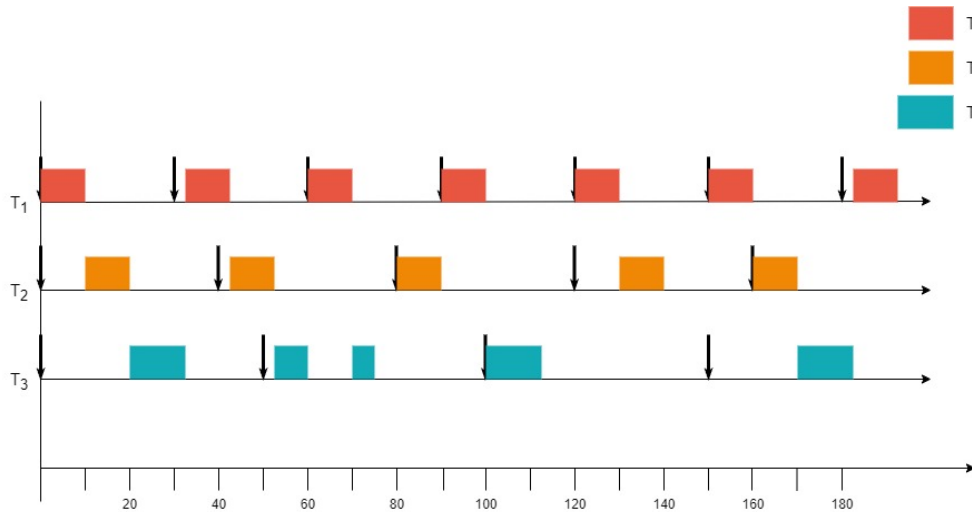


Figura 2.2: Ejemplo de planificación EDF

Cabe destacar como al tratarse cambiar dinámicamente las prioridades según el deadline, la segunda ejecución de la tarea T_3 se para para ejecutar T_1 , debido a que la prioridad en ese momento es mayor puesto que el deadline de esta última es más cercano, una vez finaliza su ejecución se ejecutará la tarea que lo estaba haciendo previamente. Este ejemplo es muy bueno pues nos permite observar que en el caso de que usásemos un algoritmo de prioridades estáticas como por ejemplo RMS, con prioridades $T_1 > T_2 > T_3$, no se garantizarían los plazos porque en la primera ejecución de T_3 no finalizaría porque pasaría a ejecutarse las otras tareas con mayor prioridad y no daría lugar a que finalizase esta última dentro del plazo máximo establecido.

2.4. La nube: Definición y proveedores

Como ya he comentado en otro punto de este trabajo la principal ventaja de los sistemas en la nube es la escalabilidad bajo demanda de los recursos necesarios, permitiendo optimizar los costes de nuestro sistema. En los sistemas convencionales necesitaríamos tantos recursos en nuestro sistema según las necesidades establecidas en el momento en el que el mismo está recibiendo el máximo de las peticiones. No necesitaremos la misma capacidad de procesamiento cuando salen a la venta los tickets de este como cuando es temporada vacacional y no hay eventos, pero la empresa tendrá que seguir pagando el máximo de recursos para prevenir situaciones de mayor demanda. Es por este motivo que las grandes tecnológicas han decidido crear sus propios servicios para proveer de estos recursos como son, Amazon, Microsoft o Google.

2.4.1. Proveedores

Hoy la mayoría de las aplicaciones y servicios webs están desplegadas en algunos servicios de infraestructura de las grandes empresas mencionadas, todos ofrecen servicios similares y de calidad muy parecida, pero es el servicio de Amazon el que opera más volumen de uso diario, probablemente por la facilidad de su interfaz y la buena documentación que tiene, así como los tutoriales que ofrecen. Dichos servicios son los siguientes:

Amazon Web Services (AWS)

En 2006, Amazon comenzó a ofrecer sus servicios de infraestructura [18]. Hoy en día, AWS provee de este servicio a 190 países en el mundo y supone un volumen de mercado de 155 mil millones de dólares.

Azure Cloud

Fue en 2010 cuando Microsoft Azure [19] se convirtió en un servicio accesible por el mercado general. Actualmente ofrece unos 600 servicios diferentes y su volumen de mercado es de 20 mil millones de dólares.

Google Cloud (GCP)

En 2008 anunciaron un servicio parecido al que ya ofrecía Amazon en 2006 [20], sin embargo, no fue hasta finales de 2011 cuando ofrecieron el servicio al público general con soporte total.

2.4.2. IaC

Infrastructure as Code (IaC) es una manera de definir la configuración de los servicios de los proveedores previos sin necesidad de hacerlo de manera manual en cada una de las interfaces de estos. Por ejemplo, Terraform, admite este tipo de definición de atributos de la infraestructura como memoria RAM necesaria, potencia de CPU entre otros. En grandes proyectos, permite la creación de clústeres que escalarán vertical y horizontalmente dependiendo de las necesidades, esta, entre otras, son las principales ventajas que ofrece. También facilita el despliegue de estas infraestructuras en cualquiera de los servicios de manera rápida y eficaz. Además, permite el control de versiones viendo cual es la infraestructura previa y cuál es la nueva.

En nuestro proyecto podría usarse, sin embargo, al ser una aplicación web de cierta simplicidad con una instancia EC2 en AWS sería suficiente, podría también usarse un clúster de EKS donde cada componente se desplegará en un contenedor diferente, ya haremos un análisis en profundidad en el apéndice.

Capítulo 3

Servicio administrador de tareas IoT de tiempo real: RITA

En este capítulo se explica con detalle cómo se ha construido e implementado el nuevo servicio de planificación de tareas de tiempo real de monitorización en dispositivos IoT con tres objetivos principales:

- Facilitar la creación, gestión y planificación de tareas de tiempo real sobre sistemas de tiempo real en dispositivos IoT
- Supervisar la monitorización de distintos posibles parámetros del entorno a través de dispositivos IoT
- Proporcionar una herramienta docente para que los estudiantes puedan comprender y experimentar con la puesta en marcha de tareas de tiempo real.

El nuevo servicio de planificación lo hemos denominado RITA por su abreviatura Real-time IoT Tasks Administrator.

Se incluyen en profundidad los pasos y decisiones tomadas en el proyecto para alcanzar los objetivos definidos, desde los requerimientos hasta las iteraciones del desarrollo, así como la explicación y evaluación de los resultados obtenidos.

3.1. Requerimientos del sistema

Para extraer los requerimientos del sistema partimos de la necesidad de plantear algún tipo de mecanismo que facilite la planificación de tareas de tiempo real centradas en la monitorización del estado del entorno a través de los sensores y actuadores de micro-controladores. Para ello, hemos tratado de determinar como stakeholders el uso que se va a hacer de nuestra aplicación de gestión de dispositivos IoT en tiempo real y evaluar

que funcionalidad se esperaría de ella. Respecto a los requisitos no funcionales, sería necesaria una posición un tanto más técnica para controlar aquellos aspectos que son necesarios para que la aplicación funcione como se espera pero que sin embargo el cliente no ve.

3.1.1. Requisitos funcionales

A continuación, podemos ver los requisitos funcionales previamente mencionados junto con la información (requisitos de datos) que necesitarán cada uno de estos:

Tabla 3.1: RF1 - Registrar usuario

Descripción	El nuevo usuario de la aplicación se registra
Entrada	La información necesaria para crear un nuevo usuario, Nombre de usuario, contraseña, Nombre, apellidos, empresa, correo
Información almacenada	Nombre de usuario, contraseña, Nombre, apellidos, empresa, correo
Salida	Código alfanumérico (ID) que representa al usuario

Tabla 3.2: RF2 - Modificar usuario

Descripción	Modificar datos de un usuario registrado
Entrada	Datos necesarios para modificar al usuario ya sea el Nombre de usuario, contraseña, Nombre, apellidos, empresa, correo
Información almacenada	Usuario ya creado modificado Nombre de usuario, contraseña, Nombre, apellidos, empresa o correo
Salida	

Tabla 3.3: RF3 - Eliminar usuario

Descripción	Eliminar un usuario registrado
Entrada	Nombre de usuario del usuario a eliminar
Información almacenada	
Salida	

De todos los requisitos funcionales que aquí se detallan podemos extraer los distintos casos de uso que el usuario podrá llevar a cabo en nuestra aplicación, dando lugar al siguiente diagrama de casos de uso:

Tabla 3.4: RF4 - Añadir organización

Descripción	Añade una nueva organización al usuario actual
Entrada	Información necesaria para crear una organización; Localización, mapa del edificio, descripción
Información almacenada	Organización ya creada; Localización, mapa del edificio, descripción
Salida	Código alfanumérico (ID) que representa a la organización

Tabla 3.5: RF5 - Modificar organización

Descripción	Modificar una organización ya existente
Entrada	Datos necesarios para modificar una organización, ya sea Localización, mapa del edificio o descripción
Información almacenada	Organización ya creada modificada Localización, mapa del edificio o descripción
Salida	

Tabla 3.6: RF6 - Eliminar organización

Descripción	Eliminar una organización ya existente
Entrada	Nombre de la organización a eliminar
Información almacenada	
Salida	

Tabla 3.7: RF7 - Añadir dispositivo

Descripción	Añadir un nuevo dispositivo a una de las organizaciones existentes
Entrada	Información necesaria para crear dispositivo; Nombre, IP y Puerto
Información almacenada	Dispositivo ya creado; Nombre, IP y Puerto
Salida	Código alfanumérico (ID) que representa al dispositivo

Tabla 3.8: RF8 - Modificar dispositivo

Descripción	Modificar un dispositivo existente
Entrada	Datos necesarios para modificar un Dispositivo existente, ya sea Nombre, IP o Puerto
Información almacenada	Dispositivo ya creado modificado; Nombre, IP y Puerto
Salida	

Tabla 3.9: RF9 - Eliminar dispositivo

Descripción	Eliminar un dispositivo existente
Entrada	Nombre del dispositivo a eliminar
Información almacenada	RD17
Salida	RD18

Tabla 3.10: RF10 - Añadir tarea

Descripción	Añadir una tarea a uno de los dispositivos existentes
Entrada	Información necesaria para crear una tarea; Nombre, periodo, estado, tiempo de cómputo, máximo tiempo de respuesta, número del puerto del que se quieren leer los datos en el dispositivo, un Código alfanumérico (ID) del sensor
Información almacenada	Tarea ya creada con toda la información necesaria; Nombre, periodo, estado, tiempo de cómputo, máximo tiempo de respuesta, número del puerto del que se quieren leer los datos en el dispositivo, un Código alfanumérico (ID) del sensor
Salida	Código alfanumérico (ID) que representa a la tarea

Tabla 3.11: RF11 - Modificar tarea

Descripción	Modificar una tarea existente
Entrada	Datos necesarios para modificar una tarea existente, ya sea; Nombre, periodo, estado, tiempo de cómputo, máximo tiempo de respuesta, número del puerto del que se quieren leer los datos en el dispositivo o un Código alfanumérico (ID) del sensor
Información almacenada	Tarea existente ya modificada, Nombre, periodo, estado, tiempo de cómputo, máximo tiempo de respuesta, número del puerto del que se quieren leer los datos en el dispositivo, un Código alfanumérico (ID) del sensor
Salida	

Tabla 3.12: RF12 - Eliminar tarea

Descripción	Elimina una tarea existente
Entrada	Nombre de la tarea a eliminar
Información almacenada	
Salida	

Tabla 3.13: RF13 - Crear sensor

Descripción	Crea un sensor para poder usarlo a la hora de crear una tarea
Entrada	Datos necesarios para crear un sensor, Nombre, función y métricas
Información almacenada	Sensor ya creado con toda la información que necesita; Nombre, función y métricas
Salida	Código alfanumérico (ID) que representa al sensor

Tabla 3.14: RF14 - Modificar sensor

Descripción	Modificar sensor
Entrada	Datos necesarios para modificar un sensor ya sea; Nombre, función o Métricas
Información almacenada	El sensor ya existente modificado
Salida	

Tabla 3.15: RF15 - Eliminar sensor

Descripción	Eliminar un sensor
Entrada	Nombre del sensor a eliminar
Información almacenada	
Salida	

Tabla 3.16: RF16 - Visualizar métricas

Descripción	Visualizar métricas obtenidas de una tarea
Entrada	Nombre de la tarea de la que hay que obtener los datos
Información almacenada	
Salida	Métricas obtenidas

Tabla 3.17: RF17 - Actualizar los datos

Descripción	Se actualizan los datos que se visualizan de las tareas
Entrada	Nombre de la tarea cuyos datos se quieren actualizar
Información almacenada	
Salida	Métricas obtenidas

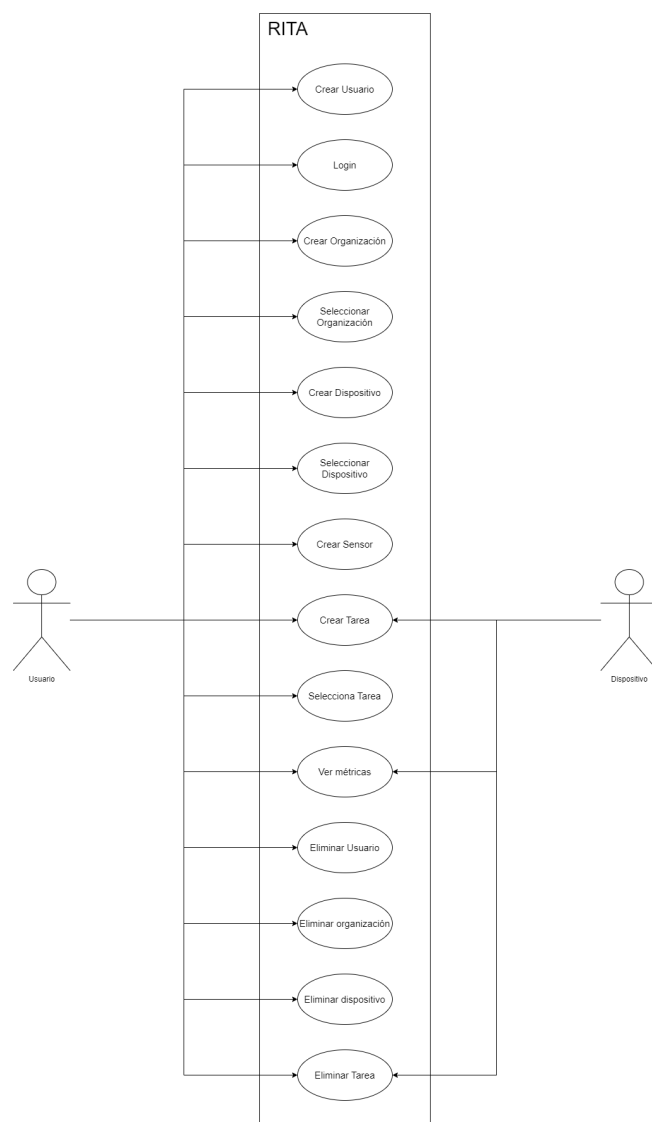


Figura 3.1: Diagrama de casos de uso

3.1.2. Requisitos no funcionales

Dentro de estos requisitos no se tendrán en cuenta aquellos provistos por las tecnologías cloud como puede ser la escalabilidad vertical u horizontal del sistema.

Tabla 3.18: RNF1 - Rendimiento del sistema para crear los distintos elementos

Nombre	Rendimiento del sistema para crear los distintos elementos
Descripción	No debería de tardar más de 5 segundos en crear un usuario, organización o dispositivo dentro del sistema, esto incluye la llamada al back-end, la finalización de la operación en la base de datos y devolver una respuesta visible por el usuario.

Tabla 3.19: RNF2 - Rendimiento del sistema para crear las tareas

Nombre	Rendimiento del sistema para crear tareas
Descripción	No debería de tardar más de 5 segundos en crear una tarea dentro de un dispositivo, esto implica las operaciones de cálculo para ver si esta es planificable o no siguiendo el algoritmo de planificación EDF, la finalización de la operación en base de datos y devolver una respuesta visible al usuario.

Tabla 3.20: RNF3 - El sistema debe aguantar la existencia de usuarios enviando peticiones a la aplicación a la vez.

Nombre	El sistema debe aguantar la existencia de usuarios enviando peticiones a la aplicación a la vez.
Descripción	Este número puede ser ilimitado.

Tabla 3.21: RNF4 - El sistema debe estar siempre disponible al tratarse de sistemas críticos para el usuario.

Nombre	El sistema debe estar siempre disponible al tratarse de sistemas críticos para el usuario.
Descripción	Si el sistema por algún motivo fallara debería de haber un sistema para backup de manera que a ojos del usuario siempre permanezca disponible.

Tabla 3.22: RNF5 - El sistema debe ser seguro en cuanto a los datos sensibles del usuario y del sistema.

Nombre	El sistema debe ser seguro en cuanto a los datos sensibles del usuario y del sistema.
Descripción	En cuanto a esto, datos como nombre, correo, localización de la infraestructura, mapa y disposición de los dispositivos deben ser privados a cada usuario.

Tabla 3.23: RNF6 - El sistema debe ser seguro en cuanto a los datos.

Nombre	El sistema debe ser seguro en cuanto a los datos.
Descripción	Si por algún caso pasase algo con la base de datos, esta debe ser recuperable.

Tabla 3.24: RNF7 - El sistema debe ser escalable horizontal y verticalmente.

Nombre	El sistema debe ser escalable horizontal y verticalmente.
Descripción	Esto mejorará el rendimiento global de la aplicación en momentos críticos siempre y cuando los dispositivos IoT puedan satisfacer la demanda pues los mismos tienen capacidad limitada.

Tabla 3.25: RNF8 - Rendimiento del sistema para actualizar los distintos elementos.

Nombre	Rendimiento del sistema para actualizar los distintos elementos.
Descripción	No debería de tardar más de 5 segundos.

Tabla 3.26: RNF9 - Rendimiento del sistema para eliminar los distintos elementos.

Nombre	Rendimiento del sistema para eliminar los distintos elementos.
Descripción	No debería de tardar más de 5 segundos.

3.1.3. Restricciones semánticas

Las restricciones semánticas por su lado serán las siguientes:

Tabla 3.27: RS1 - Es necesario que no exista otro usuario idéntico

Nombre	Es necesario que no exista otro usuario idéntico
Descripción	No puede haber más de un usuario con los mismos datos en especial el correo electrónico que servirá como identificación para el sistema.

Tabla 3.28: RS2 - Es necesario que no exista otra organización igual

Nombre	Es necesario que no exista otra organización igual
Descripción	No puede existir una organización igual a otra para un usuario, con los mismos datos y en especial que no tenga la misma ubicación.

Tabla 3.29: RS3 - Es necesario que no exista otro dispositivo igual dentro de una misma organización

Nombre	Es necesario que no exista otro dispositivo igual dentro de una misma organización
Descripción	Dentro de la misma organización no pueden existir dos dispositivos iguales, con la misma IP o mismo puerto principalmente pero puede haber otras características a tener en cuenta como que no esté en exactamente la misma ubicación.

Tabla 3.30: RS4 - La tarea debe ser planificable para poder ser añadida a un dispositivo

Nombre	La tarea debe ser planificable para poder ser añadida a un dispositivo
Descripción	Teniendo en cuenta las propiedades del algoritmo EDF, en caso de que la tarea sea planificable, esta se podrá añadir al sistema, dando por hecho que se ha preestablecido un número máximo de tareas para que la potencia de computo del dispositivo IoT sea suficiente como para cumplir los tiempos estipulados en la configuración de la tarea.

Tabla 3.31: RS5 - Para poder eliminar un dispositivo no puede tener tareas previamente

Nombre	Para poder eliminar un dispositivo no puede tener tareas previamente
Descripción	Todas las tareas deben haber finalizado una última vez (sobre todo si se trata de la tarea de actualización de datos emitida por el frontend).

Tabla 3.32: RS6 - Para poder eliminar una organización hay que eliminar todos los dispositivos previamente

Nombre	Para poder eliminar una organización hay que eliminar todos los dispositivos previamente
Descripción	Es decir todos los dispositivos de dicha organización han de haber actualizado los datos mostrados por última vez para finalmente ser eliminados antes de poder eliminar la organización.

Tabla 3.33: RS7 - Es necesario que no exista otra tarea idéntica

Nombre	Para poder crear un tarea es necesario que ya no exista otra idéntica.
Descripción	Es decir no se pueden crear tareas con exactamente las mismas características, principalmente el puerto al que se conecta el sensor.

Tabla 3.34: RS8 - Es necesario que exista un usuario para crear una organización

Nombre	Para poder crear una organización debe existir un usuario previamente.
Descripción	Para que las organización se pueda crear el usuario previamente deberá de obtener sus credenciales de login creando una cuenta en la aplicación

Tabla 3.35: RS9 - Es necesario que exista una organización para crear un dispositivo

Nombre	Para poder crear un dispositivo debe existir una organización previamente.
Descripción	Para que el dispositivo se pueda crear el usuario previamente deberá crear una organización y seleccionarla

Tabla 3.36: RS10 - Es necesario que exista un dispositivo para crear un sensor

Nombre	Para poder crear un sensor debe existir un dispositivo previamente.
Descripción	Para que el sensor se pueda crear el usuario deberá de haber creado un dispositivo y seleccionarlo

Tabla 3.37: RS11 - Es necesario que exista un sensor y un dispositivo para crear una tarea

Nombre	Para poder crear una tarea debe existir un dispositivo y un sensor previamente.
Descripción	Para que la tarea se pueda crear el usuario deberá de haber creado un dispositivo y seleccionarlo, además de crear un sensor pues será parámetro obligatorio en la creación de la tarea.

3.2. Modelo conceptual del sistema

El modelo conceptual del sistema muestra las entidades software principales que tiene que gestionar el servicio de planificación de tareas de tiempo real. Se ha utilizado un diagrama entidad-relación para representar los conceptos y relaciones entre ellos.

El servicio de planificación puede gestionar las necesidades de planificación de tareas de un conjunto de usuarios a través de un conjunto de organizaciones. A través de las organizaciones podemos gestionar un conjunto de dispositivos IoT relacionados junto con las tareas de tiempo real de monitorización que se van a ejecutar sobre dichos dispositivos.

Las tareas de tiempo real de monitorización se dedicarán a capturar información del entorno a través de los sensores que tienen asignados en los endpoint o puertos de los dispositivos generando datos que serán recogidos a través de métricas.

En el esquema de la siguiente página se puede visualizar la relación que hay entre los conceptos principales del sistema.

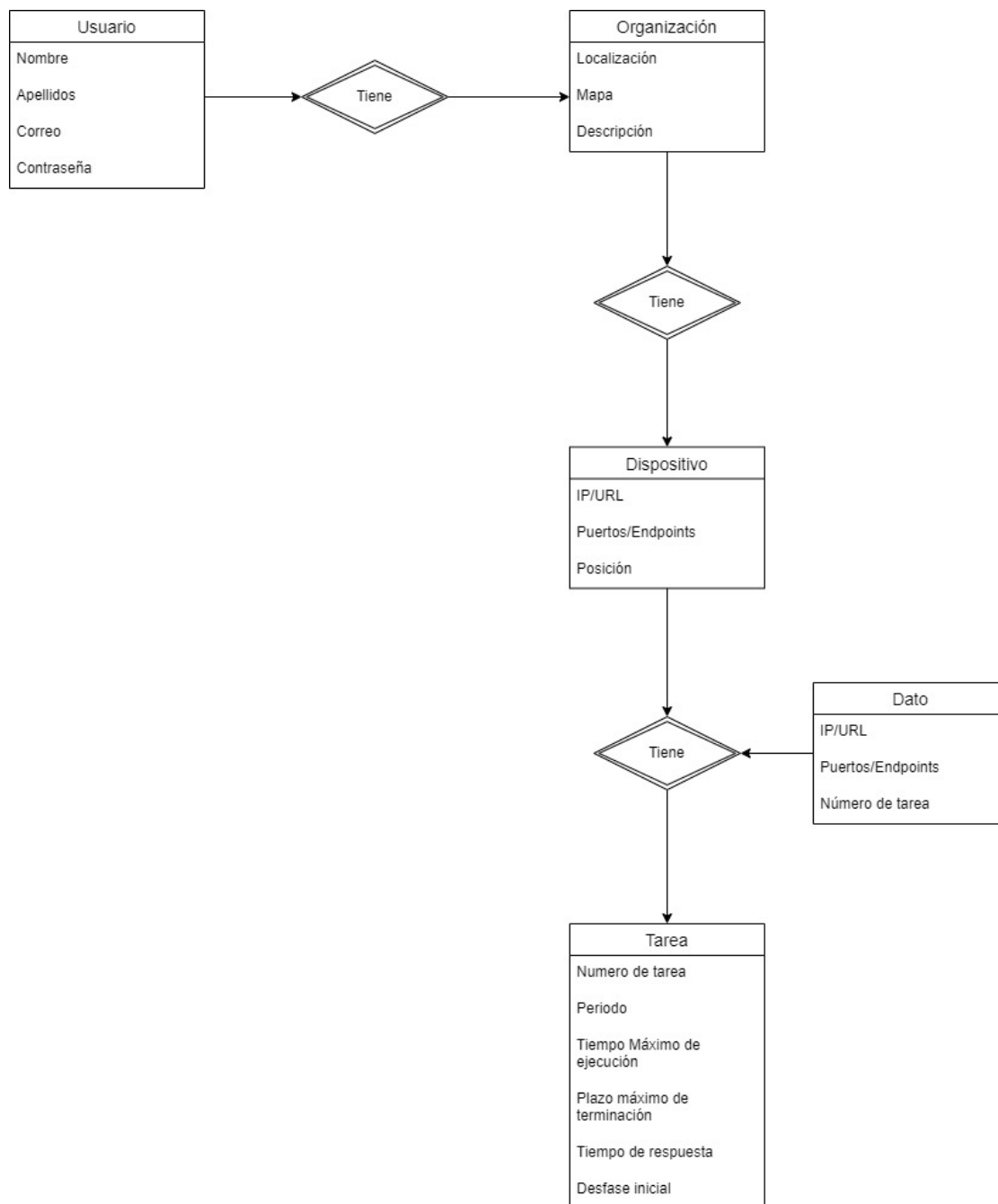


Figura 3.2: Modelo conceptual con un modelo entidad-relación

En resumen, los conceptos principales del sistema son los siguientes:

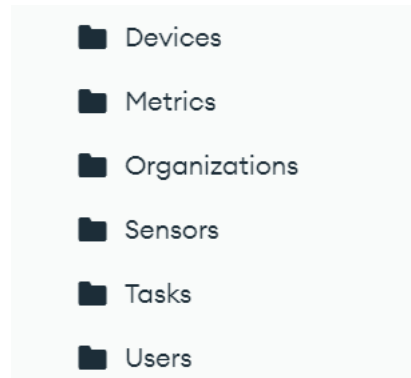


Figura 3.3: Conceptos principales del sistema

Donde cada uno de los elementos del mismo cuentan con la estructura descrita en los siguientes apartados.

3.2.1. Devices

Los dispositivos se identifican de forma única a través de un Id. Este identificador se genera con la librería `uuid.v4()` que nos proporciona Angular y de la misma manera para el resto de los objetos almacenados en la base de datos. UUID (universal unique identifier) es un número de 128 bits con lo que el número posible de combinaciones será 2^{128} . Un ejemplo de estos identificadores es el siguiente: 99b6e31a-f783-4d02-9488-66f5b57b66da.

En resumen, para estos dispositivos tendremos los siguientes atributos:

- **Id.**
- **Name.** Nombre para identificar el dispositivo por parte del usuario en la web.
- **Ip.** Será la ip del dispositivo del que se tendrán que recibir métricas y estado de las tareas y al que se enviarán dichas tareas según la planificabilidad de las mismas.
- **Port.** Este será el puerto del dispositivo.

- **OrganizationId.** Es el id de la organización a la que pertenece dicho dispositivo pues en el caso de uso sobre el que se pensó dicho dispositivo pertenecerá a una de las localizaciones específicas de la empresa o el usuario como puede ser su casa de la playa o su vivienda habitual.

3.2.2. Metrics

Las métricas tendrán un `taskId` que representa la tarea a la que pertenecen y que cuya definición se encuentra en el punto [3.2.5](#). Los atributos que contienen son:

- **Id.**
- **Name.** Nombre para que el usuario pueda identificarla en la interfaz de usuario.
- **Value.** Valor de la métrica.
- **Date.** Fecha en la que se generó dicha métrica.
- **Position.** Este identificador se utiliza para saber en qué posición se encuentra dicha métrica dentro de todas las métricas disponibles para el dispositivo.
- **TaskId.** La tarea con la que está vinculada la métrica.

3.2.3. Organizations

La organización que contiene los dispositivos como he mencionado anteriormente, tiene los siguientes atributos:

- **Id.**
- **Name.** Nombre para que el usuario pueda identificarla en la interfaz de usuario.
- **StreetAddress.** Ubicación de la organización .
- **UserId.** Usuario con el que está enlazada la organización.

3.2.4. Sensors

Los sensores están vinculados a los dispositivos que tendrán la función de calcular el valor final que se mostrará al usuario, estos sensores se conectan a los dispositivos y podrán crearse y reusarse cuando se cree una tarea nueva, ya que la tarea contiene atributos de interés.

- **Id.**
- **Name.** Nombre del sensor.
- **Function.** Función que mapeará los valores recibidos a valores legibles.
- **Metrics.** Métricas relacionadas con este sensor.

3.2.5. Tasks

Las tareas contienen todo lo necesario para que se estudie si dicha tarea es planificable según el algoritmo de planificabilidad usado. Y para que estas, en caso de que sean planificables se envíen al dispositivo.

- **Id.**
- **Name.** Nombre de la tarea para que sea identificable por el usuario.
- **Period.** Periodo de ejecución de la tarea.
- **Status.** Estado que se encuentra la tarea, pending, inactive o active, dependiendo de si está planificada, si se puede ejecutar o no y si se está ejecutando.
- **CpuTime.** Tiempo de ejecución de la tarea.
- **MaxResponseTime.** Tiempo de respuesta máximo de la tarea.

- **PortNumber.** Puerto al que se conecta la métrica a medir.
- **SensorId.** Sensor conectado al dispositivo.
- **DeviceId.** Dispositivo con el que está vinculada la tarea.

3.2.6. Users

El usuario por su parte sería el objeto más simple de nuestro modelo no relacional ya que solo contaría con el nombre del usuario y con la contraseña, credenciales que usaría para iniciar sesión en la web.

Si representamos el modelo conceptual utilizando un diagrama de clases UML en lugar del diagrama entidad-relación quedaría de la siguiente manera:

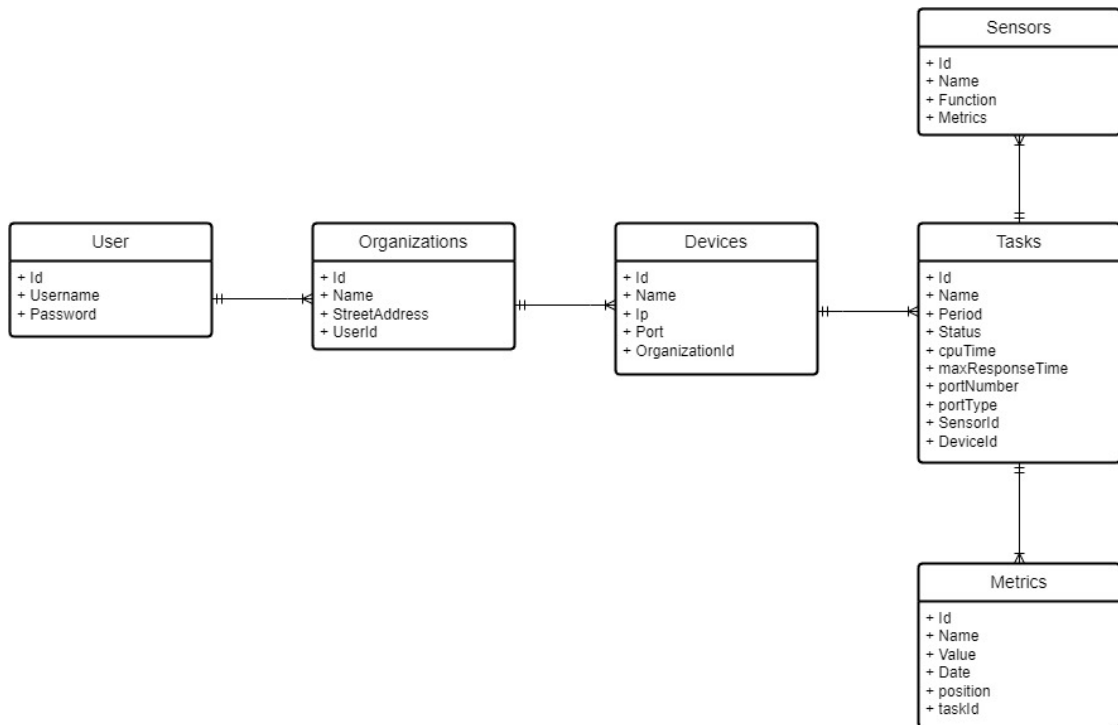


Figura 3.4: Diagrama UML del modelo conceptual

3.3. Arquitectura del sistema

La plataforma RITA tiene una arquitectura multicapa compuesta por tres niveles:

1. La capa de visualización (front-end) en el que los usuarios pueden ver el estado de las tareas de tiempo real y su configuración.
2. La capa de gestión del servicio de planificación (back-end) que almacena y procesa las tareas de tiempo real que se van a ejecutar en cada dispositivo.
3. La capa de dispositivos en la que se ejecutan las tareas de tiempo real de monitorización.

En la figura 3.5 se muestran los componentes y relaciones que hay entre las distintas capas.

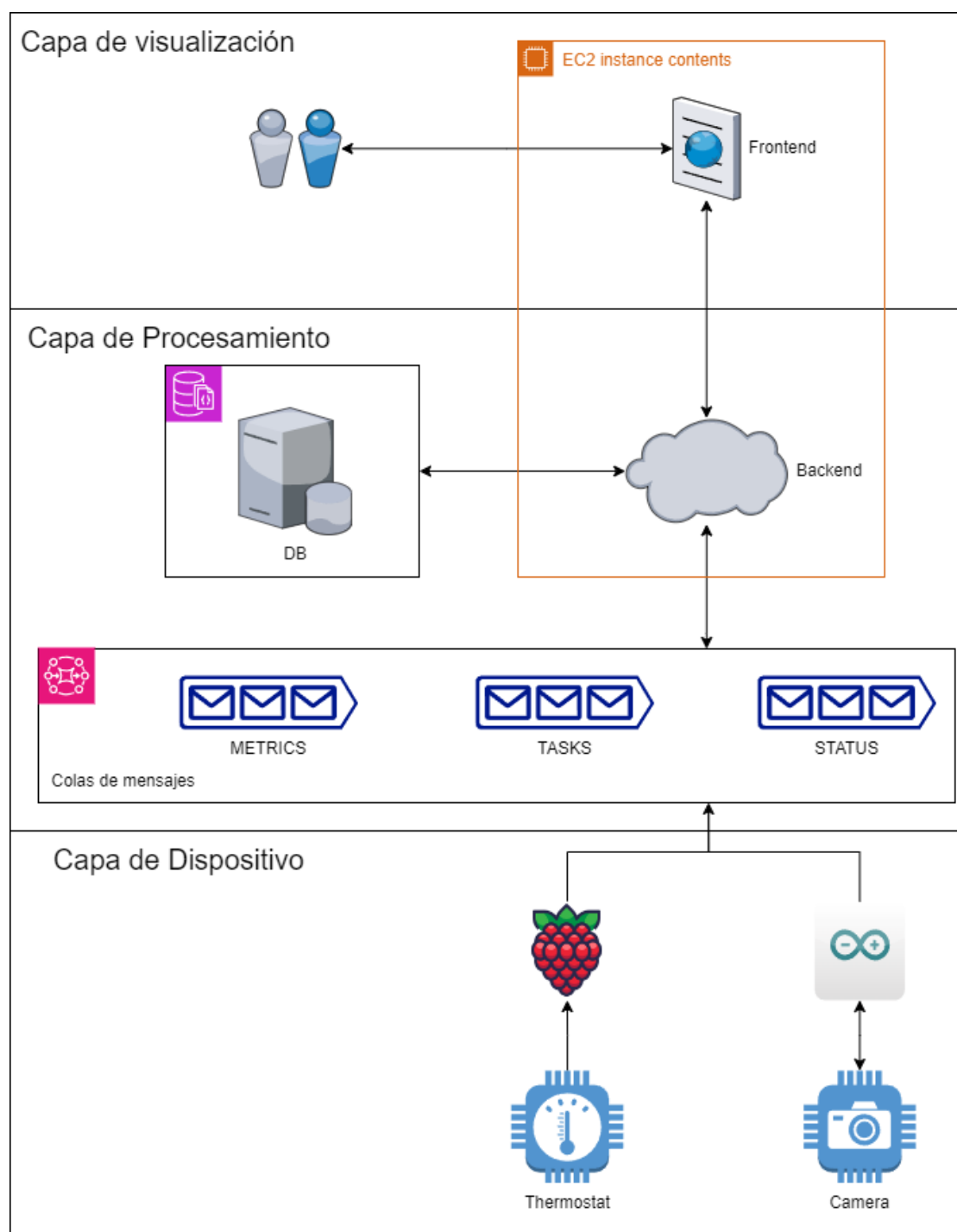


Figura 3.5: Arquitectura del sistema

En la capa de dispositivos se incluyen los dispositivos IoT en los que se va a ejecutar el controlador junto con las tareas de tiempo real de monitorización que se hayan creado y configurado en el servicio de planificación. Cada uno de estos dispositivos incluirá un controlador en el que se ejecutan tareas de tiempo real periódicas que acceden a los datos de los sensores que tengan conectados.

La capa de procesamiento del servicio de planificación (back-end) se encarga de mantener un almacenamiento de las tareas de tiempo real que se configuren en el servicio, los datos que se obtienen de los dispositivos IoT, y de suministrar los datos necesarios a la capa de visualización para que el cliente pueda operar sobre el sistema.

Por tanto, debe realizarse una gestión pormenorizada de la información las tareas (tasks), las métricas (datos que se obtienen de los sensores de los dispositivos IoT) y el status (estado en el que se encuentra el dispositivo). Por esta razón, debe haber un sistema de almacenamiento persistente que almacene dicha información para el acceso de los clientes, generalmente soportado por un gestor de base de datos.

Otro componente importante en el que se apoya la capa de procesamiento es un broker en el que se implementa un gestor de colas y se va a encargar de desacoplar los mensajes que se envían entre la capa de procesamiento y los dispositivos. En nuestro caso, la información de las tareas (tasks), las métricas y el status. Esto se hace para evitar que cualquier caída del sistema de los dispositivos o del servicio de planificación pueda afectar a la otra parte.

En la capa de visualización (front-end) tendremos un cliente a través del cual podremos configurar las tareas de tiempo real y obtener información visual de los datos que se generan por parte de los dispositivos.

Como se puede observar en la figura 3.5 se ha incluido en un componente tanto la capa de front-end y el back-end, porque posteriormente se podrán insertar en un contenedor que puede ser desplegado en una máquina en el cloud o en un servidor local.

3.4. Diseño de la interfaz de usuario

A continuación se muestra el conjunto de pantallas que se han desarrollado como interfaz de usuario del servicio de planificación RITA. Se describe en primer lugar el prototipo de cada una de las pantallas de RITA y en segundo lugar el resultado final, las pantallas que finalmente se han diseñado para nuestro servicio. En primer lugar se muestra el cuadro de mandos o la página Home del servicio de planificación RITA. Como se visualiza en la imagen la información que se presenta al usuario está organizada en base a organizaciones y dispositivos. Para cada dispositivo se deben mostrar las tareas de tiempo real que se van a ejecutar en dicho dispositivo, que este primer boceto conceptual se representa como data en el expansible.

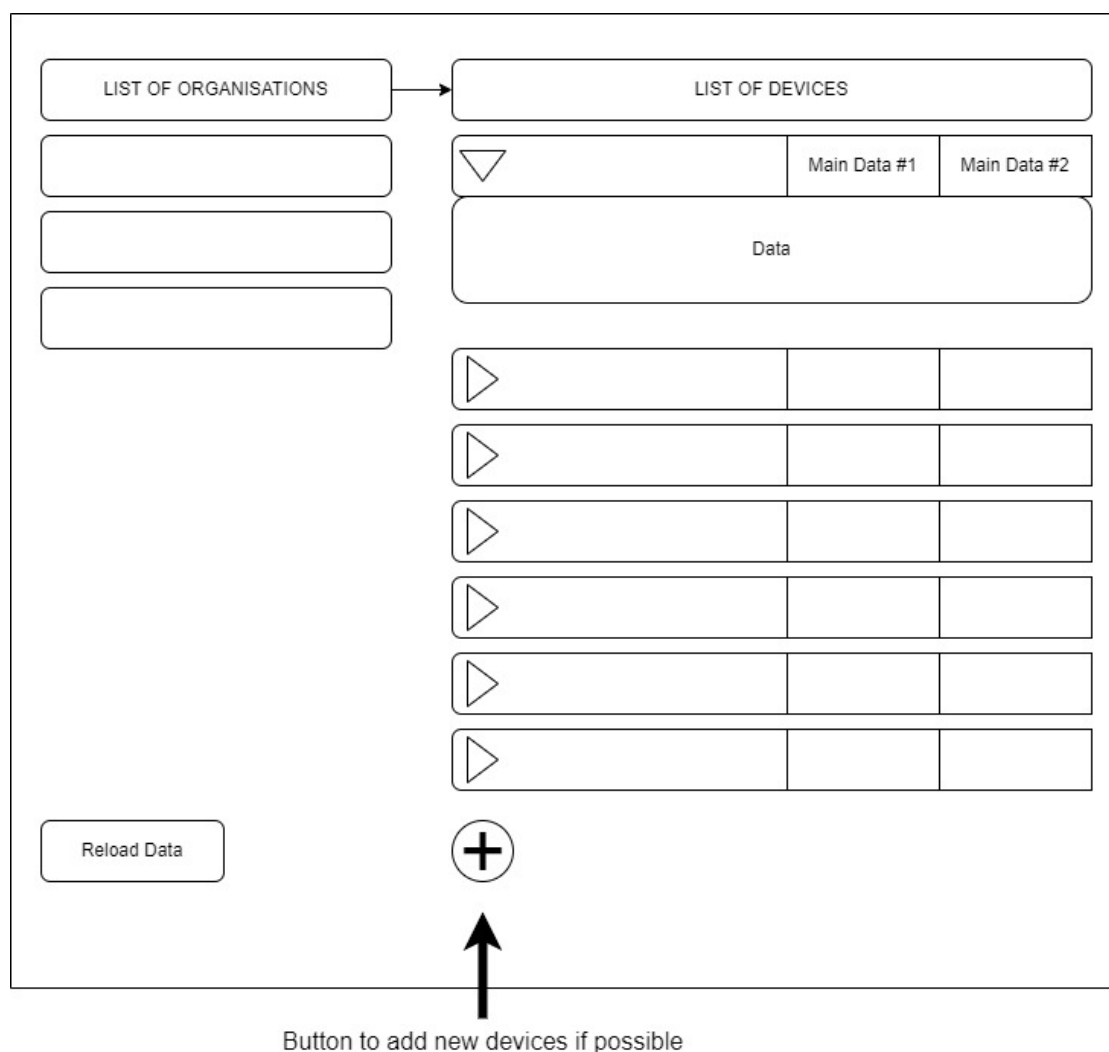


Figura 3.6: Interfaz de usuario dashboard

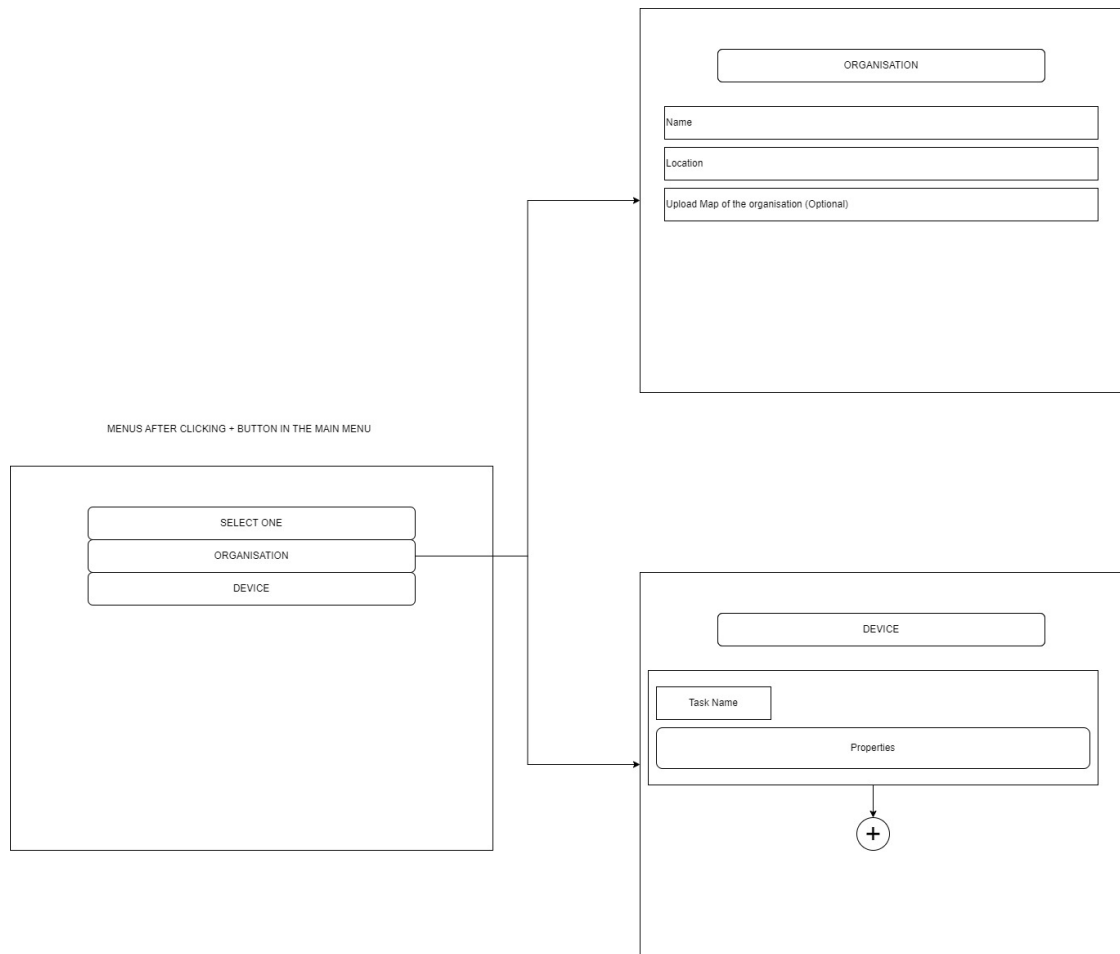


Figura 3.7: Interfaz de usuario para crear dispositivos/organizaciones

En esta imagen previa, se puede visualizar como se pretendía en el boceto original que se manejase la creación de organizaciones, dispositivos o tareas, sin embargo, esto se modificó finalmente como se puede observar a continuación:

- Pantalla de login: La pantalla de login del usuario permite también que este se registre con el botón de Register por el cual podrá acceder al formulario.

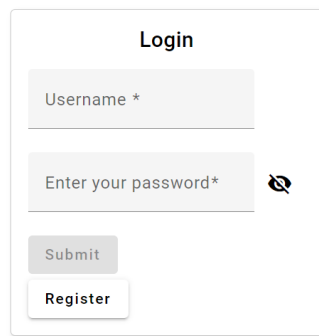
A login form titled "Login" with a light gray background and rounded corners. It contains two input fields: "Username *" and "Enter your password*", both with light gray borders. To the right of the password field is a small eye icon for toggling visibility. Below the input fields are two buttons: a gray "Submit" button and a white "Register" button with a gray border.

Figura 3.8: Login

- Pantalla principal: Como se puede observar de un vistazo, este diseño difiere del original, sin embargo, intenta mantener la misma estructura que se presentaba en el boceto, en esta página, el usuario podrá gestionar todo lo relativo al sistema visualizando principalmente organizaciones y dispositivos con elementos expansibles. Si cambia, la creación de organizaciones que se hará a través del formulario de la posición 1, la de dispositivos, con el formulario que aparece haciendo clic en la organización (posición 2) [3.10](#), creándolo en la posición 3 y desde la que podremos gestionar con distintos diálogos la creación de sensores y tareas [3.11](#)

Organizations

testtest2

Name *

Address*

Create Organization

Devices test

testtest20⋮

Scheduler*▶3

Tasks

task1testPort: 20

Figura 3.9: Página principal

Organizations

testtest

Name *

IP *

Port *+

Name *

Address*

Create Organization

Devices test

testtest20⋮

Figura 3.10: Creación de dispositivos

Create Task

Name *	Period *	CpuTime *
MaxResponseTime *	PortNumber *	PortType *
Sensor* ▼		

No Thanks Ok

Figura 3.11: Formulario de creación de tareas

- Pantalla de métricas: Si hacemos clic en las tareas, tendremos acceso a la visualización de las métricas y sus valores en el tiempo:

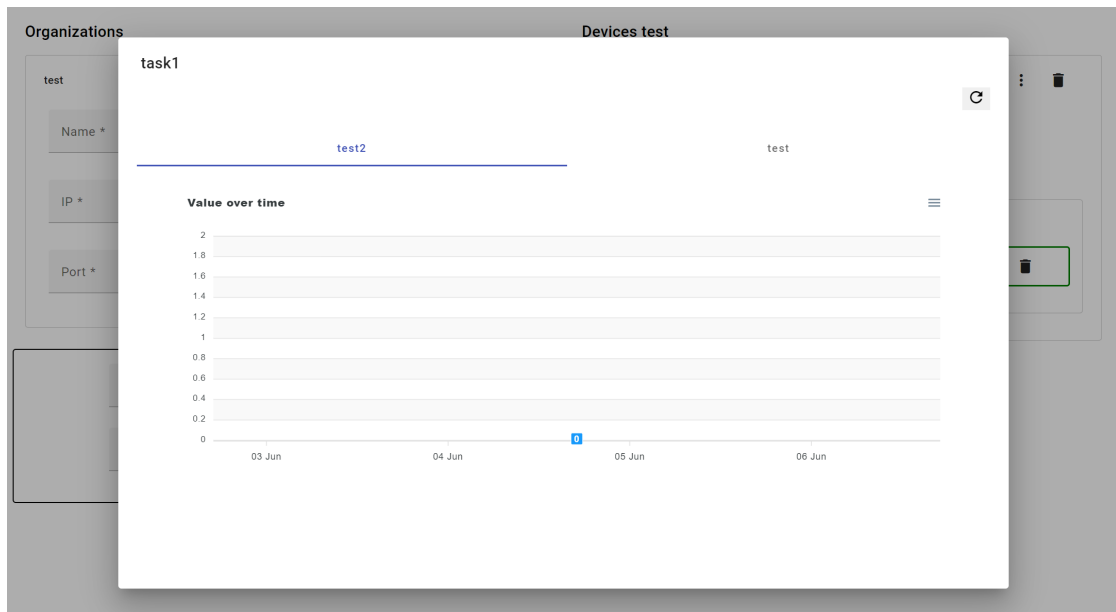


Figura 3.12: Gráfico de métricas

3.5. Implementación

Dada la arquitectura de nuestro sistema nos da la posibilidad de ver en cada una de las capas cuales han sido las distintas decisiones de desarrollo tomadas y cual es el contexto del sistema o sistemas en los que se han llevado a cabo las distintas pruebas para cada capa en concreto. Para esto iremos avanzando desde el punto de vista del usuario (capa de visualización) hasta la última capa (capa de dispositivo), comunicación objetivo para nuestro proyecto RITA. Esta división por capas coincide con la división de la estructura del proyecto como se puede ver en su repositorio <https://github.com/laguilarg99/RITA>, en el cual contamos con:

- *Monitoring-frontend*. Es la capa de visualización.
- *Monitoring-backend*. Es la capa de procesamiento (aunque no se incluye ni la base de datos ni el message broker)
- *Monitoring-iot*. Es la capa de dispositivo y corresponde con el código en C que deberemos desplegar en cada uno de los dispositivos que deseamos monitorizar.
- *docker-compose.yml*. Este fichero especifica el despliegue de cada uno de los componentes de la capa de visualización y procesamiento (incluyendo MongoDB y el message broker, RabbitMQ) como veremos en más detalle en la sección 3.7

3.5.1. Capa de Visualización

La capa de visualización se refiere concretamente al frontend del sistema web. Esta capa se encarga de la interacción persona-ordenador, del usuario contra el servicio de planificación RITA.

Justificación de la tecnología usada

Para decidir la tecnología más adecuada para implementar el front-end, en nuestro caso, lo mejor es evaluar primeramente cual es el nivel de complejidad que alcanzará nuestro sistema. Siguiendo los casos de uso y requisitos funcionales propuestos en la sección 3.1 podemos intuir que con un uso estándar, la aplicación contará con multitud de usuarios, organizaciones, dispositivos, tareas y sensores

Aunque existen múltiples frameworks sobre diferentes lenguajes de programación (python, java, javascript, etc) se decidió utilizar un framework Javascript, fundamentalmente porque los navegadores (clientes del sistema) tienen el motor Javascript nativo que facilita una ejecución eficiente del sistema y, por otra parte, provisiona la creación de lo que se conocen como componentes.

Se pueden utilizar distintos frameworks Javascript como **Angular 18** (en nuestro caso), o cualquier versión de ReactJS. La elección de este framework concretamente no se

ha realizado al azar, sino por la experiencia profesional previa en él. Además, estas tecnologías admiten el uso de librerías creadas por la comunidad de desarrollares, en mi caso he usado **Angular Material**, que es una librería de componentes, facilitando el desarrollo de esta capa.

Componentes del Frontend

Los componentes nos determinan cual será la estructura de nuestro proyecto frontend, la selección de dichos componentes va a ser de vital importancia para mantener la claridad del código y la modularidad del mismo.

Por definición, estos componentes, nos posibilitan gestionar distintos MVC (Modelo vista-controlador) y repetirlos las vistas tantas veces como sea necesario. Los componentes que podemos identificar en nuestro proyecto son los siguientes:

- Home. Página de gestión y visualización de organizaciones, dispositivos y tareas del usuario.
- Login. Página de login del usuario.
- Metrics-dialog. Diálogo para la visualización de las métricas generadas por una tarea.
- Register. Página de registro del usuario.
- Sensor-dialog. Diálogo para el formulario de creación de un sensor para la creación de tareas en los dispositivos.
- Task-Dialog. Diálogo para el formulario de creación de una tarea para un dispositivo.

Descripción del funcionamiento de RITA

La funcionalidad clave de estos frameworks es que a los componentes, se les da un nombre y es este nombre, a través del uso de labels o etiquetas típicas de html de apertura y cierre como se mostrarán los templates o vistas. Una vez esté implementado este MVC podremos mostrarlo usando `<micomponente/>` en cualquier otro de los componentes de mi proyecto.

Según se realizan los desarrollos en estos frameworks, tendremos un fichero `index.html` (`src/index.html`) con el nombre de *app-root* y es a partir de esta usando el enrutador (`src/app/app.routes.ts`), como le diremos a nuestra aplicación que pantallas mostrar en cada momento. En nuestro caso es claro que la pantallas que necesitamos son, una pantalla de registro, una de login y el home.

En caso de que el usuario este logado, nuestro programa siempre lo redirigirá a la página principal (home), de otro modo, si no lo está siempre que acceda a la app, se use el path

que se use, siempre se redirigirá al login, desde el cual podrá registrarse o introducir las credenciales en caso de que sea necesario.

Una vez el usuario rellena el formulario, el botón ejecutará una función del servicio (`src/services/api.service.ts`), *registerUser* o *loginUser* dependiendo desde que pantalla lo estemos haciendo. Normalmente en Javascript necesitaríamos programar en dichas funciones una promesa para hacer la llamada asíncrona de registro o login a nuestro backend, sin embargo, gracias a las librerías que integra Angular, haciendo uso de la inyección de dependencias, podemos inyectar, es decir, crear, el objeto de tipo `HttpClient` con el cual podemos enviar cualquier petición HTTP. Como son peticiones al back-end, la mayoría son POST a excepción de las de eliminación que son DELETE.

```
http: HttpClient = inject(HttpClient);

loginUser(username: string, password: string): Observable<any> {
  const user = new UserLogin('', username, password);
  return this.http.post('/api/login', user, this.httpOptions);
}
```

En caso de que el registro haya sido correcto, se redirige al usuario a la pantalla de login, introduce las credenciales, se ejecuta el formulario y la petición al backend, se crea la cookie (para que la aplicación sepa que usuario está logado) y se dirige a la página principal donde podremos observar las organizaciones, dispositivos y tareas del usuario, así como gestionar los mismos. Decidí no eliminar la cookie por tiempo como se hace habitualmente por facilitar las pruebas.

Una vez llegamos al home, el usuario tendrá la opción de crear con un formulario organizaciones en caso de que no las tenga y dentro del expansible, un formulario para crear dispositivos como se indicó en la sección 3.4. Este formulario se encuentra dentro del expansible de la organización para que el usuario pueda ver con claridad que el dispositivo que está creando es únicamente para esta. Al igual que en los otros casos tendremos una llamada a la API para crear el dispositivo usando el modelo de Device. Una vez seleccionamos la organización, tendremos la posibilidad de crear tareas y sensores de la misma manera. Para no complicar la interfaz de usuario decidí que los formularios de los mismos estuviesen en cuadros de dialogo o Popups, es por esto que existen el *sensor-dialog* (modelo del sensor) y el *task-dialogs* (modelo de tarea), estos popups se activan a través del menú que se representa como tres puntos (representación habitual para el menú) en cada dispositivo para dejar claro que lo que sea que se esté creando se está haciendo para ese dispositivo únicamente. En el expansible, podremos ver la tareas y haciendo clic en ellas se cargará otro cuadro de diálogo pero en este caso con una gráfica de los datos que tenemos y que veremos en la sección de resultados 3.8.

Seguridad, Privacidad y Gestión de los modelos

En cuanto a seguridad, cada que vez que el usuario inicia sesión correctamente en nuestro servicio, se creará una cookie con la que nuestra aplicación sabrá que el usuario está efectivamente loggeado. Una vez se introducen las credenciales oportunas, para esto, en el apartado de services (src/services) creé un servicio llamado storage.service.ts con la finalidad de gestionar dicha cookie, si las credenciales son correctas, este servicio creará una cookie con el user.id en el session storage del navegador.

Los modelos definidos en la sección 3.2, serán implementados por primera vez gracias al uso de typescript en esta capa de visualización y deberán duplicarse en la capa de procesamiento para permitir la comunicación entre capas e incluso para el caso de las tareas y las métricas, tendrán que duplicarse en la capa de dispositivo, por lo que se puede decir que para facilitar la comunicación entre capas los modelos deben ser transversales a todas ellas. Dichos modelos se encuentran en nuestra aplicación Angular en la carpeta de models (src/models) y son los siguientes:

- Dispositivo (Device)
- Métrica (Metric)
- Organización (Organization)
- Sensor (Sensor)
- Tareas (Tasks)
- User (Users)

El modelo User, se utiliza de igual manera tanto en el login como en el registro, en caso de que se modificasen los datos del mismo (por ejemplo añadiendo dirección de correo) habría que crear nuevos modelos, que podrían llamarse UserLogin y UserRegistration por seguir el formato hasta ahora.

Para la comunicación entre capas podríamos haber usado otro formato que no fuese JSON pero debido a su extensión dentro de la industria, es cada vez más común que las principales librerías hagan un mejor soporte a este y usar otros como SOAP se hace más complicado, más en este caso que en la capa de procesamiento tenemos una API REST.

3.5.2. Capa de Procesamiento

La capa de procesamiento se conforma fundamentalmente de 3 partes diferenciadas:

- Back-end. Se encarga de la gestión de las peticiones provenientes de la capa de visualización.
- Base de datos. Es el almacenamiento permanente de los modelos especificados en la sección 3.2.
- Gestor de colas. Se encarga de la comunicación entre esta capa de procesamiento y la capa de dispositivo.

Justificación de las tecnologías usadas

En esta capa se han usado 3 tecnologías distintas, una para cada una de las partes diferenciadas. En el caso del back-end estamos usando Java con el framework Spring boot, la elección de la misma es por la experiencia profesional previa y por la facilidad que existe junto con Maven para la gestión de dependencias de nuestro proyecto, de esta manera se puede añadir funcionalidad que facilite el desarrollo, como pueden ser las librerías encargadas de la comunicación con la base de datos y con el gestor de colas.

En cuanto a la tecnología usada para la base de datos, se ha decidido que esta fuese no relacional, el uso de esta base de datos tiene carácter formativo además de que MongoDB cuenta con gran fama dentro de la comunidad de desarrolladores y del mundo empresarial, realmente si se buscara un sistema que fuese escalable para nuestro caso de uso, lo mejor sería usar una base de datos como InfluxDB, esta se trata de una Base de datos de series temporales y permite el acceso a los datos de series temporales (en nuestro caso los datos recogidos de los sensores) con mucha más rapidez.

Para la gestión de colas se ha usado RabbitMQ, debido a su facilidad para integrarlo con el back-end. Su uso se justifica porque la comunicación que se realiza entre la capa de procesamiento y el dispositivo puede verse interrumpida en multitud de ocasiones por distintas razones (falla en la conexión, fin de la batería del dispositivo, caída del servidor...), el problema reside en que si el usuario crea una de estas tareas y hay un error catastrófico, esta tarea se perdería para siempre, este sistema con el gestor de colas, nos permite que quede en la cola hasta que el servidor se pueda reiniciar o en su defecto, la conexión o cualquiera que sea el problema. Lo mismo aplica para el caso de las métricas que deseamos recibir del dispositivo, así como su estado.

Estructura, comunicación, Seguridad y Modelos.

La estructura de ficheros que tenemos nos permite entender como funciona nuestro programa, como se puede ver dicha estructura es muy simple pero suficiente para albergar

toda la funcionalidad que necesitamos, es así como contamos con las siguientes carpetas dentro de nuestro proyecto backend:

- Components.
- Config.
- Controllers.
- Interfaces.
- Models.
- Repositories.
- Services.

Los controladores son los endpoints a los que la capa de visualización envía las peticiones, como comentaba previamente, gracias a las anotaciones de Java Spring (que son básicamente funcionalidad o características que se le da a determinadas funciones por el simple hecho de indicarlo antes de las mismas), es en estos controladores a través del uso de los modelos, interfaces y repositorios, que se pueden gestionar el ingreso al almacenamiento permanente de los modelos. En la carpeta config, tenemos la configuración para que se puedan enviar los datos respecto del estado de la tarea por el web socket a la capa de visualización. En la carpeta components tenemos los listeners y en la carpeta services los publishers, estos se encargan de la comunicación entre el back-end y el gestor de colas, para de esta manera comunicarse con la capa de dispositivo, recibiendo métricas y estado a través de los listeners (están a la escucha de las colas en las que llegará la información necesaria para actualizar el estado de la tarea o tareas y para crear nuevas métricas) y enviando las tareas a través del publisher, haciendo uso para ello de 3 colas, TASKS, METRICS y STATUS.

En cuanto a las decisiones de diseño tomadas, son pocas, pues la gestión de los modelos, la base de datos y el gestor de colas, está muy estandarizado. La única diferencia se encuentra en el controlador encargado de la creación de tareas, este la convierte en formato JSON y la envía directamente al dispositivo a través de la colada de TASKS con el fin de estudiar en él su viabilidad en función del algoritmo de planificación usado y ejecutarla.

La seguridad de nuestra aplicación se basa en su despliegue, la idea, es que aunque Java ofrece distintas anotaciones relacionadas, la seguridad de nuestro sistema en el caso de la capa de procesamiento y de dispositivo reside en la configuración de la infraestructura y su despliegue en una red local de uso exclusivo para optimizar al máximo los tiempos de comunicación entre ambas. El uso de la librería Java para la gestión de la base de datos (MongoDB) como almacenamiento permanente, hace que se añada un atributo adicional (class) para representar la clase de Java por la que se creó dicho documento (como se

conocen a los datos en MongoDB), creo que es de importancia después de mencionar los modelos y elaborar un modelo conceptual de la aplicación, mostrar un ejemplo de cada uno de los documentos dentro de la misma:

```
_id: "9b073ae5-f3d1-4dad-bc23-6b32b22fe1bb"  
name : "test"  
ip : "localhost"  
port : 4000  
organizationId : "72aca132-44a3-4052-a0ae-74a425833956"  
_class : "com.performance.monitoring.models.Device"
```

Figura 3.13: Ejemplo de dispositivo

```
_id: "922cfe46-ba05-4720-b8e4-838eecd87967"  
name : "test2"  
value : 0  
date : "2024-06-04T17:55:40.360Z"  
position : 1  
taskId : "497c8364-1f57-4463-a1cc-23cbdf2c1156"  
_class : "com.performance.monitoring.models.Metric"
```

Figura 3.14: Ejemplo de métrica

```
_id: "8edef6fb-c8f7-4510-8a4b-ada1ad2bb98e"  
name : "test"  
streetAddress : "test"  
userId : "bcb63620-aa2b-4b80-aff0-f63a7db2159b"  
_class : "com.performance.monitoring.models.Organization"
```

Figura 3.15: Ejemplo de organización

```
_id: "0ceca037-8ba7-4140-897f-c31df22740e9"  
name : "test"  
function : "l+ls"  
metrics : "test,test2"  
_class : "com.performance.monitoring.models.Sensor"
```

Figura 3.16: Ejemplo de sensor

```
_id : "4639fcd6-6593-4846-aa1c-6e91f3f1f2e1"  
name : "test"  
period : 200  
status : "INACTIVE"  
cpuTime : 200  
maxResponseTime : 20  
portNumber : 20  
portType : "Analog"  
sensorId : "0ceca037-8ba7-4140-897f-c31df22740e9"  
deviceId : "a91e0d0c-3bbf-4693-9a7d-d7ce26b7a4e7"  
_class : "com.performance.monitoring.models.Task"
```

Figura 3.17: Ejemplo de tarea

```
_id : "bcb63620-aa2b-4b80-aff0-f63a7db2159b"  
username : "test"  
password : "test"  
_class : "com.performance.monitoring.models.UserLogin"
```

Figura 3.18: Ejemplo de usuario

3.5.3. Capa de Dispositivos

Los dispositivos que se conectan al servicio de planificación tienen que ejecutar un programa controlador para recibir los datos de las tareas de tiempo real que tiene que ejecutar y suministrar los datos de los dispositivos al servicio para su monitorización.

Este programa controlador está compuesto de tres módulos bien diferenciados:

- Módulo principal. Se encarga de inicializar el controlador y el planificador.
- Módulo de comunicación. Se encarga del seguimiento de las tareas que se tienen que ejecutar provenientes del gestor de colas del servicio de planificación y monitorizar los estados o métricas de los sensores del dispositivo que tiene que enviar.
- Módulo de planificación de las tareas. Se encargará de mantener la planificación y ejecución de las tareas de tiempo real que se están ejecutando en el dispositivo.

Inicialmente se esperaba que se pudiera ejecutar sobre dispositivos que dispongan de sistema operativo de tiempo real. Por ese motivo, se realizó una implementación en C, ya que fácilmente se puede trasladar a cualquier sistema operativo de tiempo real que sea compatible con POSIX.

En una primera aproximación, se han probado la ejecución sobre sistemas operativos POSIX como Linux sobre máquinas virtuales y máquinas reales.

El módulo de comunicación tiene una hebra que se encarga de mantener la conexión con el sistema de planificación. Dicha hebra está a la escucha de que el usuario complete la configuración de una tarea de tiempo real en el servicio de planificación y envíe los datos de dicha tarea a la cola de RabbitMQ.

Cuando dicha tarea es recibida a través de la cola de TASKS, este módulo traduce el mensaje JSON que llega, a la estructura que definimos siguiendo el modelo conceptual definido en la sección 3.2 como se ve reflejado en el siguiente código:

```
Task *parse_task_json(const char *json_string) {
    json_error_t error;
    json_t *root = json_loads(json_string, 0, &error);
    if (!root) {
        fprintf(stderr, "Error parsing JSON: %s\n", error.text);
        return NULL;
    }

    Task *task = malloc(sizeof(Task));
    if (!task) {
        fprintf(stderr, "Memory allocation error\n");
        json_decref(root);
    }
}
```

```

        return NULL;
    }

    json_t *id = json_object_get(root, "id");
    json_t *name = json_object_get(root, "name");
    json_t *status = json_object_get(root, "status");
    json_t *period = json_object_get(root, "period");
    json_t *cpuTime = json_object_get(root, "cpuTime");
    json_t *maxResponseTime = json_object_get(root, "maxResponseTime");
    json_t *portNumber = json_object_get(root, "portNumber");
    json_t *portType = json_object_get(root, "portType");
    json_t *sensorId = json_object_get(root, "sensorId");
    json_t *deviceId = json_object_get(root, "deviceId");

    task->id = strdup(json_string_value(id));
    task->name = strdup(json_string_value(name));
    task->status = strdup(json_string_value(status));
    task->period = json_integer_value(period);
    task->cpuTime = json_integer_value(cpuTime);
    task->maxResponseTime = json_integer_value(maxResponseTime);
    task->portNumber = json_integer_value(portNumber);
    task->portType = strdup(json_string_value(portType));
    task->sensorId = strdup(json_string_value(sensorId));
    task->deviceId = strdup(json_string_value(deviceId));

    json_decref(root);
    return task;
}

```

Una vez se reciban los datos de una tarea de tiempo real el módulo de comunicación pasa los datos al módulo de planificación que se encargará primero en configurar una hebra periódica con los atributos temporales indicados (periodo, computo y tiempo de respuesta máxima) y los puertos donde están asociados los sensores y actuadores. Una vez arrancada la hebra, procederá a ejecutar la tarea de forma periódica, obteniendo los datos generados por sensores, que se pasan al módulo de comunicación. Es con este método, gracias a las librerías de RabbitMQ que C soporta, como se enviarán todos los datos recogidos de nuevo a la capa de procesamiento:

```

void publish_message(amqp_connection_state_t conn,
                    char const *queue_name,
                    char const *message) {
    amqp_basic_properties_t props;
    props._flags = AMQP_BASIC_CONTENT_TYPE_FLAG | AMQP_BASIC_DELIVERY_MODE_FLAG;

```

```
props.content_type = amqp_cstring_bytes("text/plain");
props.delivery_mode = 2; // Persistent delivery mode

die_on_error(
    amqp_basic_publish(
        conn,
        1,
        amqp_cstring_bytes(""),
        amqp_cstring_bytes(queue_name),
        0,
        0,
        &props,
        amqp_cstring_bytes(message)),
    "Publishing"
);

printf("Published message to queue '%s': %s\n", queue_name, message);
}
```

Aunque en principio se tenía pensado tener diferentes tipos de planificadores en el controlador del dispositivo (RMS, DMS y EDF), finalmente sólo se encuentra activo el planificador de prioridades estático RMS. Este planificador se asienta muy bien en los sistemas operativos de tiempo real POSIX en cuanto a que los sistemas POSIX tienen definido por defecto un planificador SCHED_FIFO.

SCHED_FIFO promociona la ejecución de las tareas con más prioridad, de modo que en caso de que se ejecute una tarea de menor prioridad, se encargará de cambiar el contexto por la tarea de mayor prioridad hasta su finalización. El planificador RMS prioriza la ejecución primero de las tareas de menor período. Por tanto, el controlador, a partir de los datos que reciba del servicio de planificación, evaluará los periodos de cada tarea y asignará la prioridad estática de cada tarea para posteriormente ponerlo en ejecución con el planificador estático SCHED_FIFO de los sistemas POSIX.

En el siguiente código se puede observar como se crea una hebra con los parámetros señalados por el modulo de comunicación al obtener la tarea de la capa de procesamiento, `infothread` es una estructura propia del código en C y que funcionará como DTO para mover entre el módulo de comunicación y de planificación de tareas la información necesaria en cada momento:

```
//crea la hebra
pthread_t thread;
inicializaThread(&thread, &infothread);

printf("hebra creada\n");
```

```

        sleep(10);

        finalizaThread(&thread, &infothread);

printf("hebra finalizada\n");

```

La estructura de datos `infothread` recibe el nombre de `tipo_periodic` y contiene los siguiente elementos:

```

typedef struct {
//Datos generales del thread
int no_thread; //numero de hilo
struct timespec periodo; //Periodo - T
struct timespec fase;    //1a activacion - fase phi - fija instante critico
struct timespec computo; //tiempo de computo - C
struct timespec deadline; //plazo de respuesta maxima
int prioridad; //prioridad de la tarea

//Estado del thread
long num_activaciones; //numero de activaciones realizadas por la tarea
int isActive; //!=0 significa que esta activo

//Estadisticas
struct timespec tiempo_comienzo; //Marca de inicio del tiempo
struct timespec tiempo_fin; //Marca de fin del tiempo

    //rabbitmq
    amqp_connection_state_t conn;
    char *taskId;
    int portId;
} tipo_periodic;

```

El código del programa se podrá ejecutar en varios dispositivos, leyendo de los sensores bastaría con cambiar la implementación del método `read_sensor_n_publish_metrics` del archivo `Monitoring-iot/amqListenernPublisher.c`.

En el siguiente diagrama se puede apreciar la comunicación entre los módulos:

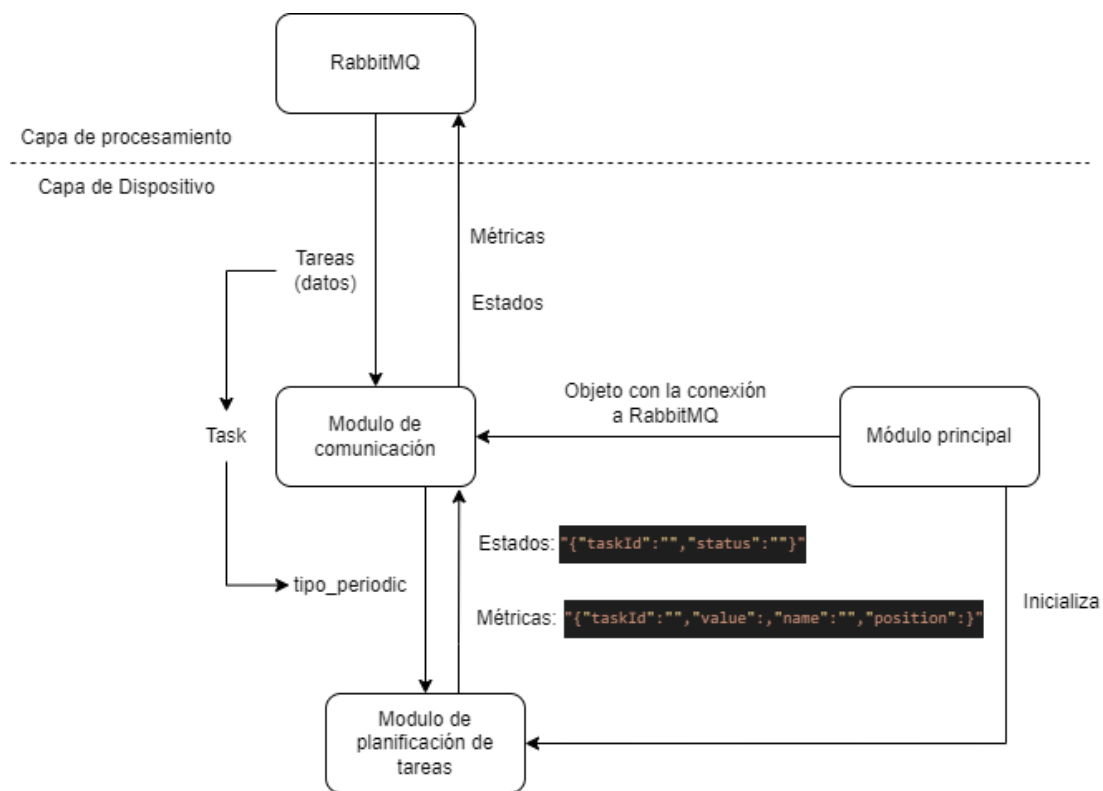


Figura 3.19: Comunicación entre los módulos del dispositivo

3.6. Pruebas

Al haberse realizado un desarrollo en cascada, la fase de verificación se ha llevado a cabo manualmente y no como en un TDD (Test Driven Development), por lo cual no hay pruebas unitarias y de integración a nivel de código, sin embargo, merece la pena elaborar un conjunto de pruebas unitarias y de integración (sabiendo que las pruebas unitarias se refieren a aquellas que prueban una funcionalidad específica y las de integración, prueban cómo esta funcionalidad se comporta correctamente con el resto del sistema) aunque sean ejecutadas de manera manual, estas son suficiente para ver si nuestro servicio RITA cumple con ciertos casos de uso que se esperan que sea capaz de ejecutar una vez finalizado el desarrollo, de esta manera marcaremos con ✓ aquellos que han pasado satisfactoriamente y con ✗ en caso de que no se hayan podido completar:

- En el caso de tests unitarios:
 - Verificar que el usuario se puede crear ✓
 - Verificar que el usuario se puede modificar ✗
 - Verificar que el usuario se puede eliminar ✗
 - Verificar que la organización se puede crear ✓
 - Verificar que la organización se puede modificar ✗
 - Verificar que la organización se puede eliminar ✓
 - Verificar que el dispositivo se puede crear ✓
 - Verificar que el dispositivo se puede modificar ✗
 - Verificar que el dispositivo se puede eliminar ✓
 - Verificar que la tarea se puede crear ✓
 - Verificar que la tarea se puede modificar ✗
 - Verificar que la tarea se puede eliminar ✓
 - Verificar que el sensor se puede crear ✓
 - Verificar que el sensor se puede modificar ✗
 - Verificar que el sensor se puede eliminar ✗
- En el caso de tests de integración:
 - El usuario queda registrado en la base de datos y puede iniciar sesión correctamente. ✓
 - El usuario puede añadir y eliminar elementos y estos ya no existen en la base de datos. ✓
 - El usuario puede crear organizaciones y los dispositivos asociados a esta se visualizan correctamente. ✓

- Al eliminar una organización, dispositivos y tareas vinculados se eliminan. ✓
- Cuando seleccionamos un dispositivo podemos ver todas las tareas asociadas y el estado de estas. ✓
- Cuando seleccionamos una tarea podemos ver las métricas recogidas. ✓
- el usuario al iniciar sesión solo podrá ver los datos vinculados a su usuario. ✓

3.7. Despliegue y puesta en marcha

En nuestro caso al tratarse de un despliegue básico estamos haciendo uso de un fichero de configuración yml para Docker con Docker Compose (este último permite la ejecución de múltiples contenedores como servicio), en caso de que decidiésemos mover la aplicación a un servicio cloud, necesitaríamos de otras tecnologías de IaC como Terraform o Kubernetes.

En dicho yml de configuración, se puede observar:

- Servicio rabbitmq con la imagen que le corresponde, así como la definición explícita de la red a la que estarán asociada el resto de servicios, además podemos ver los puertos que se exponen, el 5672 para recibir las peticiones de envío de mensaje o de lectura y el 15672 para la consola de administración desde la que podemos ver si hay mensajes en las colas para procesar.
- Servicio de MongoDB con la imagen que le corresponde, la red al igual que antes, en este caso el puerto 27017 expuesto y cabe destacar que tenemos un volumen de datos (esto hace que los datos que se crean en la base de datos son persistentes a pesar de que el contenedor en el que se está ejecutando muera, además al tener siempre el mismo path, cuando se reinicie volveremos al estado previo)
- Servicio Java Spring Boot. Usaremos la imagen oficial de Java 17 y exponeremos el puerto 8080
- Servicio Angular. NodeJS 20 y el puerto externo es el 80 y el interno (al contenedor) el 4200

.

Me gustaría señalar, que las URL de los servicios dentro de la red de docker compose que hemos creado, corresponde con el nombre del servicio, por ejemplo, para conectar con la aplicación de spring boot desde nuestro frontend podemos usar la siguiente url `spring_app:8080`. Por lo cual, si se quisiese hacer un despliegue tradicional, habría que modificar los parámetros.

Dicho todo esto para el despliegue de nuestra aplicación simplemente tendremos que ejecutar el docker compose con el comando:

```
docker-compose up
```

Una vez los contenedores de nuestra aplicación están corriendo, crearemos un usuario, una organización y un dispositivo para finalmente copiar el id del dispositivo. Después iremos al dispositivo en el que queremos ejecutar nuestro programa en C para la ejecución de tareas de monitorización y tendremos que crear una variable de entorno, compilar nuestro código y darle permisos si es necesario, para después ejecutarlo:

```
export DEVICE_ID=<device id obtained from the web application>
```

```
gcc amqListenernPublisher.c -o amqListenernPublisher -lrabbitmq  
-ljansson
```

```
chmod +x amqListenernPublisher
```

```
./amqListenernPublisher
```

En caso de que este dispositivo sea real y no esté ejecutándose en la misma máquina que nuestra aplicación multicontenedor, tendremos que ir al método `void *publisher_n_listener_thread(void *arg)` para cambiar la variable `hostname` con el valor que corresponda a la URL donde está desplegada nuestra aplicación.

3.8. Resultados

El usuario puede esperar del sistema el poder registrarse y hacer login con sus credenciales, una vez en el home, tal y como se puede ver en la figura 3.10 se puede ver un sistema que como resultado permite la creación de organizaciones (a la izquierda, con el formulario que se observa en la parte inferior izquierda) y dispositivos (a la derecha, creados con el formulario que aparece al hacer click en la organización y para el que necesitaremos la Ip del dispositivo, el puerto de acceso y el nombre del mismo). Además, al crearse las las tareas con todos los parámetros comentados en la sección de requerimientos 3.1, estas se ejecutarán en el dispositivo recogidas por el listener que se está ejecutando en una de las hebras de este, generando el siguiente output en los logs del mismo:

```
root@sched:/home/scuser/monitoring/monitoring-iot# gcc amqListenernPublisher.c -o amqListenernPublisher -lrabbitmq -ljansson
root@sched:/home/scuser/monitoring/monitoring-iot# ./amqListenernPublisher
Waiting for messages...
Provided TASK: {"id":"42d3101d-fd3f-421a-9116-716dedf21b8b","name":"testFinal1","period":100,"status":"INACTIVE","cpuTime":200,"maxResponseTime":100,"portNumber":10,"portType":"Analog","sensorId":"1d0ecec6-a406-41c8-806b-732055c8d742","deviceId":"9eb2288f-522f-4997-8ae7-c9197482bf5b"}
Published message to queue 'STATUS': {"taskId":"42d3101d-fd3f-421a-9116-716dedf21b8b","status":"ACTIVE"}
inicia hebra
Tiempo: 0.018940
Published message to queue 'METRICS': {"taskId":"42d3101d-fd3f-421a-9116-716dedf21b8b","value":32,"name":"t","position":0}
he ejecutado hebra 1 veces
hebra creada
Tiempo: 100.055637
Published message to queue 'METRICS': {"taskId":"42d3101d-fd3f-421a-9116-716dedf21b8b","value":32,"name":"t","position":0}
he ejecutado hebra 2 veces
Tiempo: 200.058967
Published message to queue 'METRICS': {"taskId":"42d3101d-fd3f-421a-9116-716dedf21b8b","value":54,"name":"t","position":0}
he ejecutado hebra 3 veces
Tiempo: 300.076631
Published message to queue 'METRICS': {"taskId":"42d3101d-fd3f-421a-9116-716dedf21b8b","value":12,"name":"t","position":0}
he ejecutado hebra 4 veces
Tiempo: 400.035064
Published message to queue 'METRICS': {"taskId":"42d3101d-fd3f-421a-9116-716dedf21b8b","value":52,"name":"t","position":0}
he ejecutado hebra 5 veces
Tiempo: 500.083946
Published message to queue 'METRICS': {"taskId":"42d3101d-fd3f-421a-9116-716dedf21b8b","value":56,"name":"t","position":0}
```

Figura 3.20: Logs del dispositivo

Al enviar los datos a la tarea, el usuario podrá visualizarlos solo con hacer click en la tarea vinculada, además como las métricas también son persistentes, podrá tener acceso a ellos cuando sea conveniente, en este caso que estamos intentado simular cambios de temperatura, la siguiente gráfica sería el resultado que podría esperar el usuario de nuestro sistema:

testFinal1



t



Figura 3.21: Gráfica Resultado

Por ejemplo, para mostrar la utilidad de nuestra aplicación, podríamos ejecutar el caso que se describió en la tabla 2.2, para ello, tendríamos que crear un usuario, una vez en el home después de iniciar sesión, tenemos que crear una organización rellenando el formulario, hacer click en ella y rellenar los parámetros del dispositivo en el que queremos ejecutar las tareas especificadas, una vez tenemos el dispositivo hacemos click en él y a través del menú ingresamos al formulario de sensor (en primer lugar, tenemos que crear un sensor pues el sistema solo admite tareas que pretenden monitorizar las métricas obtenidas de un sensor), después crearemos las tres tareas que se especifican en la tabla previamente mencionada correspondiendo T con el parámetro Period, D con el MaxResponseTime y C con el CpuTime (como se puede apreciar en la imagen 3.22), estas se ejecutaran en el dispositivo, recogerán los datos y podremos verlos para cada una de ellas con el formato de la gráfica 3.21.

Create Task

Name *	Period *	CpuTime *
MaxResponseTime *	PortNumber *	PortType *
Sensor*		

No Thanks **Ok**

Figura 3.22: Formulario de la tarea

Capítulo 4

Conclusiones

En esta sección, llega este trabajo a su fin, en la misma, comentaré qué objetivos se han cumplido respecto a los que se propusieron, las conclusiones que extraemos de estos y cual debería ser el trabajo desarrollado de cara a futuro.

4.1. Revisión de los objetivos iniciales y conclusiones obtenidas

Los objetivos que se pueden decir que se han cumplido son los siguientes. En primer lugar, tras la realización del estado del arte se han explorado las distintas soluciones tanto académicas como de mercado existentes para facilitar la gestión de dispositivos de tiempo real, además en dicha sección queda claro cómo funcionan los sistemas de tiempo real, aspectos de su planificación y que limitaciones o ventajas tienen contra otros. Además en el desarrollo hemos seleccionado la infraestructura más adecuada aunque por limitaciones de tiempo, la infraestructura cloud no era una solución viable. Por todos los apartados del desarrollo, queda claro que el servicio queda perfectamente funcional, al menos como MVP que permita al cliente ver el potencial que tiene dicho producto, una vez escalase lo suficiente. Respecto a que tipo de dispositivos IoT pueden actuar como nodo, queda claro que solo aquellos que tienen sensores de monitoreo porque sino la plantilla de tareas con la que contamos, carecería de sentido. En cuanto a aplicar buenas prácticas de ingeniería del software, siempre he promovido el código limpio y modular, aunque si es cierto que no cuenta con tests de ningún tipo aunque sí se han hecho manuales, creo que es suficiente para asegurar la usabilidad y funcionalidades del sistema, según los requisitos definidos. Además, siempre se he procurado abstraer el problema lo máximo posible. El único objetivo que no hemos conseguido es la realización de la demo con varios usuarios pues actualmente no contamos con los recursos necesarios para el despliegue de varias máquinas que lo simulen, nuestro código por lo que por ahora solo ha sido posible conseguirlo con un dispositivo simulado. En conclusión, podríamos decir que tenemos todos los objetivos completados a un 100% menos el último que

estaría en un 90 % debido a que solo requiere del despliegue de otra máquina. En general, hemos conseguido un sistema que cumple completamente con todas las funciones que se esperaban de él,

4.2. Trabajos Futuros

Por todo lo expuesto previamente se pueden deducir que los siguientes trabajos que son interesantes para expandir la aplicación RITA aquí creada:

- Realizar pruebas en entornos reales con multitud de dispositivos.
- Escalabilidad de la infraestructura. Mejorar el Message Broker creando tópicos o exchanges directamente.
- Cambio de la base de datos. Como he comentado previamente InfluxDB tendría más sentido según nuestro tipo de datos.
- Análisis de datos. Implementar funciones para permitir el análisis de los datos.

Bibliografía

- [1] I. Amazon Web Services, “¿qué es iot? - explicación del internet de las cosas - aws.”
- [2] “Historia del internet de las cosas,” octubre 2021.
- [3] A. Alonso and A. Crespo Lorente, “Una panorámica de los sistemas de tiempo real,” *Revista Iberoamericana de Automática e Informática Industrial*, vol. 3, no. 2, pp. 7–18, 2010.
- [4] K. C. Valencia and S. Público, “Historia del cloud computing,” *Rev. Inf. Tecnol. y Soc. versión impresa*, vol. 7, pp. 51–52, 2012.
- [5] E. H. Amazon, “Lo que los robots hacen (y no hacen) en los centros logísticos de amazon,” enero 2022.
- [6] A. Delfanti, *The warehouse. Workers and robots at Amazon*. Pluto Books, 2021.
- [7] J. M. Gutiérrez-Guerrero, J. L. Muros-Cobos, S. Rodríguez-Valenzuela, M. Damas-Hermoso, and J. A. Holgado-Terriza, “Dispositivos empotrados basados en java,” in *Actas de las IV Jornadas de Computación Empotrada (JCE)*, vol. 17, p. 20, 2013.
- [8] “Rtsj main page.”
- [9] P. B. Val, *Técnicas y extensiones para Java de tiempo real distribuido*. PhD thesis, Universidad Carlos III de Madrid, 2007.
- [10] I. Corporation, “Descripción general de los sistemas en tiempo real.” Recuperado 11 de febrero de 2024.
- [11] “Planificación.”
- [12] “Planificación en tiempo real.”
- [13] R. Chandra, X. Liu, and L. Sha, “On the scheduling of flexible and reliable real-time control systems,” *Real-Time Systems*, vol. 24, pp. 153–169, 2003.
- [14] J. Montoliu Villamón, *Diseño e implementación de un planificador de tareas para sistemas de tiempo real*. PhD thesis, Universitat Politècnica de València, 2022.

- [15] C. Verma, V. Stoffová, and Z. Illés, “Rate-monotonic vs early deadline first scheduling: A review,” in *Proceeding of Education Technology-Computer science in building better future*, pp. 188–193, 2018.
- [16] N. C. Audsley, A. Burns, and A. J. Wellings, “Deadline monotonic scheduling theory and application,” *Control Engineering Practice*, vol. 1, no. 1, pp. 71–78, 1993.
- [17] J. A. Jiménez Benítez, *ANÁLISIS DE PLANIFICABILIDAD DE LOS ALGORITMO EDF Y FIFO USANDO LA DISPERSIÓN DE LOS TIEMPOS DE ARRIBO EN TAREAS DE TIEMPO REAL ESPORÁDICAS*. PhD thesis, 2013.
- [18] “Overview of amazon web services - overview of amazon web services.”
- [19] “The history of microsoft azure.”
- [20] “What is google cloud platform (gcp)?.”
- [21] “¿qué es mongodb?.”
- [22] “¿qué es java spring boot? — ibm.”
- [23] “Angular.”
- [24] “Node.js — run javascript everywhere.”
- [25] “Download mongodb community server.”
- [26] Neelabalan, “mongodb-sample-dataset.” GitHub.
- [27] “Rabbitmq: One broker to queue them all — rabbitmq.”
- [28] Amazon Web Services, Inc., “Conexión segura de dispositivos iot – precios de aws iot core – amazon web services.” <https://aws.amazon.com/es/iot-core/pricing/>, n.d. Accessed: 2024-06-22.

Apéndice A

Planificación y gestión del proyecto

Al tratarse de un proyecto de tipo práctico de desarrollo de software, incluyo este apéndice con vistas a incluir todas las especificaciones técnicas del mismo como puede ser los stakeholders del proyecto, justificación de las tecnologías, así como la oportunidad de mercado que supone.

Por otro lado en cuanto a la planificación, se indicarán las tareas que se han desarrollado en el proyecto ordenadas cronológicamente así como un diagrama de Gantt de las épicas que han tenido lugar, incluso de describirán aquellas que no fueron creadas como tickets en Jira pero que se desarrollaron al estar implícitas en otras.

Se describirán qué método de trabajo se ha usado, las tecnologías usadas, así como una justificación de las mismas con los obstáculos y riesgos de ellas y las tareas para las que se han usado.

También hablaremos de tests, es decir, que métodos de validación se han usado para comprobar que la calidad del desarrollo es suficiente.

Respecto a los costes y sostenibilidad, se tendrá en cuenta tanto las horas de desarrollo que han supuesto como los recursos que se necesitan para el despliegue de la aplicación en la nube según el modelo de precios que se estipula en Amazon Web Services.

En conclusión, este apéndice recogerá todo lo que se necesita para comprender los pasos que se han llevado a cabo para obtener la aplicación final, así como los recursos (software, hardware y humanos) que han hecho posible el producto obtenido.

A.1. Metodología/Ciclo de vida

Las metodologías son esenciales para entender la planificación y ejecución del desarrollo de un proyecto, en nuestro caso hemos hecho uso principalmente de Agile y aunque la gestión de las tareas que se ha llevado a cabo no ha sido la más precisa, si refleja que se ha aplicado satisfactoriamente la metodología usada, que en este caso es una metodología Ágil (Agile), ya que el enfoque es iterativo buscando en cada uno de los sprints (un total de 12) y en cada uno de ellos se pretendía entregar un valor a nuestro desarrollo, las ventajas que ofrece esta misma es que es flexible y se adapta muy bien a cambios, en caso de que hubiese un cliente esto consituiría una gran ventaja pues se podrían añadir nuevas tareas en caso de que el cliente las necesitase en mitad de un sprint. En cuanto a su desventaja es que puede ser caótico si no se gestiona correctamente, lo cual se demuestra en parte en este proyecto pues no todas las tareas realizadas han quedado reflejadas de manera explícita quedando huecos en las iteraciones del desarrollo aunque estos se quedan cubiertos por aquellos que se hicieron de manera implícita. Por ejemplo, en el caso del desarrollo de la interfaz de usuario hubieron algunos elementos para los que no se creó una tarea en específico pero que venía implícito con la misma, como es el caso del botón de eliminación de un dispositivo pues la tarea era “el usuario puede registrar un dispositivo”, esta de manera implícita incluye el manejo de las mismas permitiendo su eliminación.

Por otro lado, en cuanto al Ciclo de vida del proyecto, se pueden diferenciar distintas fases:

- **Concepto.** Una primera fase de concepto en la que se indentifica cual es el objetivo del proyecto y que se pretende desarrollar, en nuestro caso, coincidiría con la tarea de “Propuesta de TFM” cuyo prósito era la definición formal de este trabajo.
- **Planificación del sprint.** En este momento se planifican cada una de las tareas que se van a desarrollar durante el sprint.
- **Ejecución del sprint.** Consiste en el desarrollo de las tareas con vistas a entregar un valor al final de cada sprint.
- **Review.** Cada una de las reuniones que se han tenido, tutor/alumno tenían como objetivo revisar el sprint o iteración previa y planificar la siguiente así en las 12 ocasiones que han ocurrido.

A.2. Herramientas utilizadas

La utilización de las distintas tecnologías tiene distintos motivos como expondré en los siguientes puntos.

A.2.1. MongoDB

En el caso de MongoDB [21] se eligió por la velocidad que tendrán las consultas a la hora de extraer los datos y al tratarse de una base de datos relacional, no hay ningún problema en que se repitan los mismos valores. Además, a nivel personal, la selección del uso de bases de datos no relacionales pretendía mejorar mi aprendizaje sobre ellas y como se pueden manejar desde el punto de vista de Java con Spring Boot.

MongoDB permite el almacenamiento de documentos cuya estructura y datos pueden modificarse, teniendo estos un formato similar al JSON. El modelo del documento se asigna a objetos del código de su aplicación para facilitar el trabajo. En nuestro caso dichos objetos se ven claramente representados en los modelos de Java, por ejemplo, en el caso de los dispositivos, esta es su representación en Java:

```
@Document(collection = "Devices")
public class Device {
    @Id
    private String id;

    @Indexed(unique = true)
    private String name;
    private String ip;
    private int port;
    private String organizationId;
```

Esta base de datos, al distribuirse por núcleo permite que tenga una alta disponibilidad, escalabilidad horizontal y distribución geográfica están integradas y son fáciles de usar.

Esta es la interfaz de nuestra base de datos no relacional:

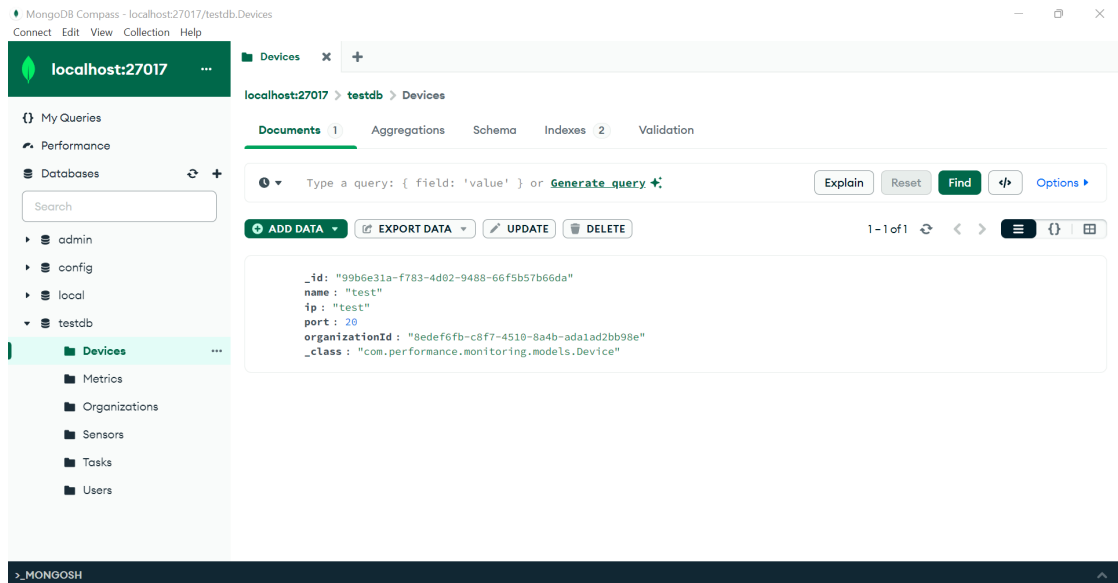


Figura A.1: Interfaz de usuario de MongoDB

En la interfaz (la cual hay que descargar aparte MongoDBCompass), permite conectarse con tantas bases de datos como tengamos, en este caso la de nuestro programa está desplegada en localhost:27017.

La interfaz tiene otras funcionalidades como, medir el rendimiento de la base de datos: Incluso, se puede ver el rendimiento por colección. Por otro lado, se pueden almacenar las queries más usadas y ver todos los atributos de cada una de las colecciones que componen la base de datos.

A.2.2. Java con Spring Boot

Spring boot [22] es un framework de Java que permite la creación de aplicaciones para su ejecución en el JVM (Java Virtual Machine). Este framework, permite y facilita el desarrollo de microservicios y aplicaciones web.

Una de las principales ventajas de este framework es la inyección de dependencias que permite a los objetos definir dependencias propias que el contenedor spring les inyecta. Además del potencial que tienen todas las librerías Java a las que se tiene acceso por usar este lenguaje.

La selección de esta tecnología es por la experiencia profesional que tengo con ella y la facilidad que eso me permitiría para añadir la máxima funcionalidad al backend y como ya he mencionado, las librerías que se ofrecen gracias a la inyección de dependencias y al amplio uso de Java mundialmente, facilita el desarrollo de toda la funcionalidad,

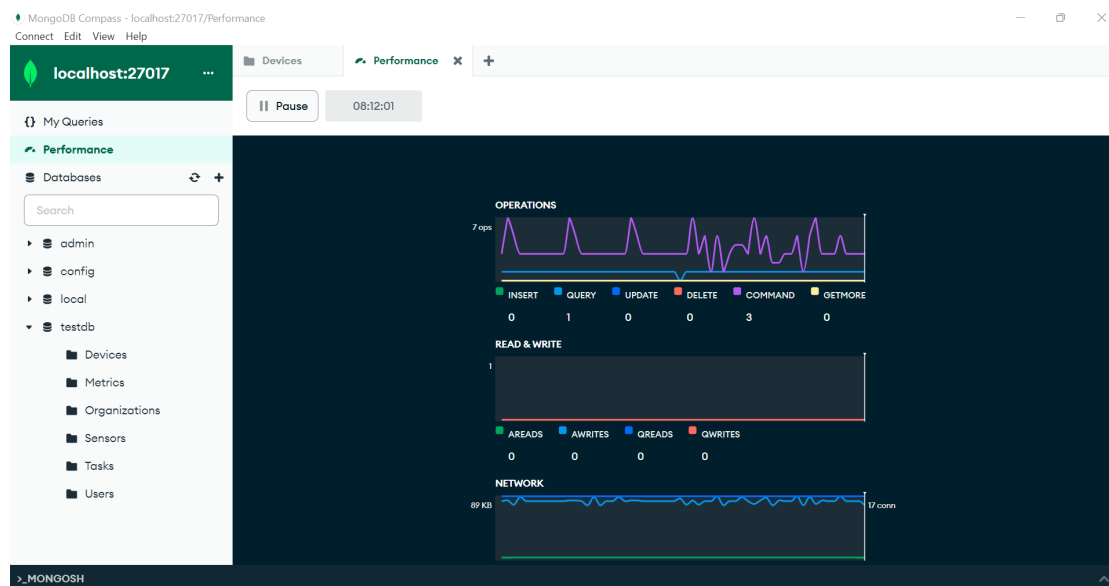


Figura A.2: Gráficas de rendimiento en MongoDB

empezando por los modelos de MongoDB que serán automáticamente creados en la base de datos una vez se guarde el primero de los objetos siguiendo la plantilla definida. Además, existen librerías que nos permiten leer mensajes del MQ (message broker) con facilidad y crear websockets con el frontend para permitir la comunicación continua del estado de las tareas y que el usuario pueda ver en tiempo real el valor de estas.

En definitiva, cabe destacar que las versiones usadas en el proyecto han sido, Java 17 y la última versión de Spring boot, 3.2.4 a fecha de este proyecto.

A.2.3. Angular

He usado en este caso la última versión de Angular [23], a fecha de este proyecto, Angular 18. Angular es un framework que permite la creación de aplicaciones web, está mantenido por un equipo dedicado de Google y provee una amplia gama de herramientas, APIs y librerías para facilitar el flujo de trabajo.

La elección de este framework en específico ha sido por la experiencia profesional que ya tenía en él, aunque como hace ya un tiempo que no trabajo con él, había olvidados muchos conceptos y quería refrescarlos así que tomé ventaja de este proyecto para ello. Además de todas las librerías, como Angular Material (con componentes prefabricados para facilitar la aplicación como formularios y botones entre otros) y la creación de componentes tienen un grandísimo potencial porque, por ejemplo, todos los dispositivos tendrán la misma estructura a nivel de interfaz de usuario, como las tareas, las métricas y las organizaciones.

A.3. Gestión del proyecto. Planificación temporal. Diagrama de Gantt

A.3.1. Iteraciones del desarrollo

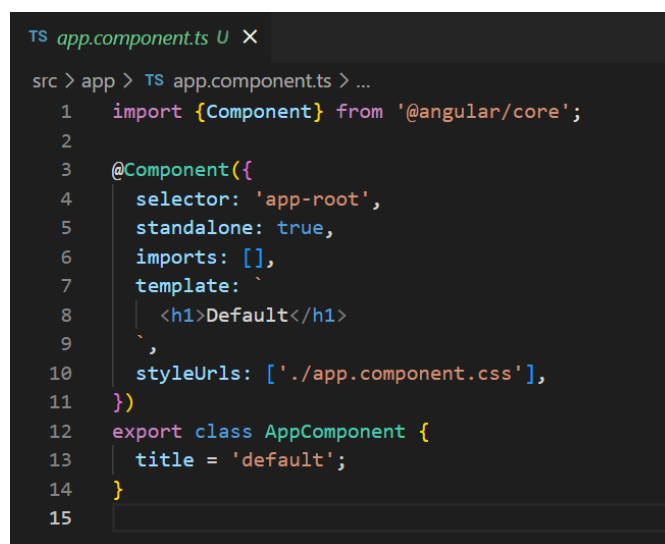
Para el caso del frontend a pesar de haber comentado que tenía experiencia previa, quería aprovechar la ocasión de este proyecto, para de esta manera comenzar a estudiar de nuevo el funcionamiento del frontend con Angular desde su base, esta es, el proyecto de ejemplo que proporciona la documentación, pero para ello comenzamos el desarrollo instalando npm y configurando el entorno de dicho ejemplo tal y como propone la documentación de Angular 18.

Para la instalación de node, en mi caso que estoy usando Windows 11, iremos a la página oficial de Node Js [24] y descargaremos la última versión estable (LTS), luego será cuestión de seguir los pasos del instalador.

Por suerte angular provee en su última versión de una excelente documentación, a través de la que, con solo seguir los pasos del tutorial de inicio de Angular, tendremos dicho proyecto en un momento.

Tras descargar el código fuente del ejemplo que como ya he mencionado, ellos proveen, será cuestión de ejecutar npm install para instalar todas las librerías necesarias y ng serve para desplegar la aplicación localmente en nuestro entorno de desarrollo, que por defecto será en el puerto 4200.

Como se puede observar, esta aplicación:



```
TS app.component.ts U X
src > app > TS app.component.ts > ...
1  import {Component} from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    standalone: true,
6    imports: [],
7    template: `
8      <h1>Default</h1>
9    `,
10   styleUrls: ['./app.component.css'],
11 })
12 export class AppComponent {
13   title = 'default';
14 }
15
```

Figura A.3: Código de la aplicación de ejemplo

Dará lugar a una página en blanco con el título H1 que contendrá la palabra Default:



Figura A.4: Ejecución de la aplicación de ejemplo

Después de esto para acoger todos los conceptos necesarios para el desarrollo de nuestro proyecto me propuse trabajar en el desarrollo de una aplicación web que propone Angular siguiendo los pasos que se especifican. Esta aplicación que comento se trata de una web para visualizar un conjunto de casas con distintos atributos, como su precio, su imagen, dirección, entre otras. Se crearán un conjunto de componentes para la visualización y de interfaces para la gestión de los datos y algún servicio para el acceso a estos además de un enrutador encargado de acceder a los enlaces de manera dinámica.

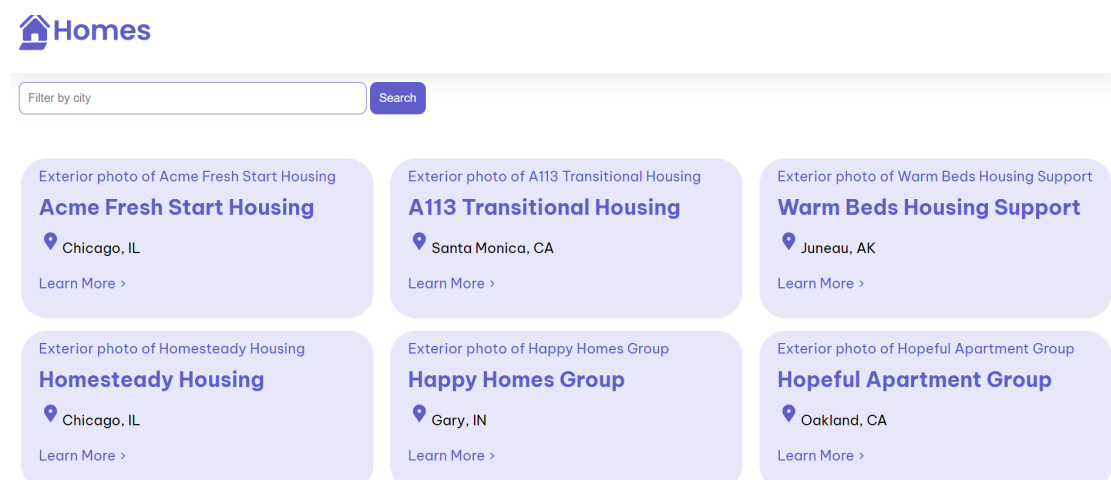


Figura A.5: Ejecución de la aplicación de ejemplo avanzada

Si hacemos click en cualquiera de las opciones podremos ver información más detallada:

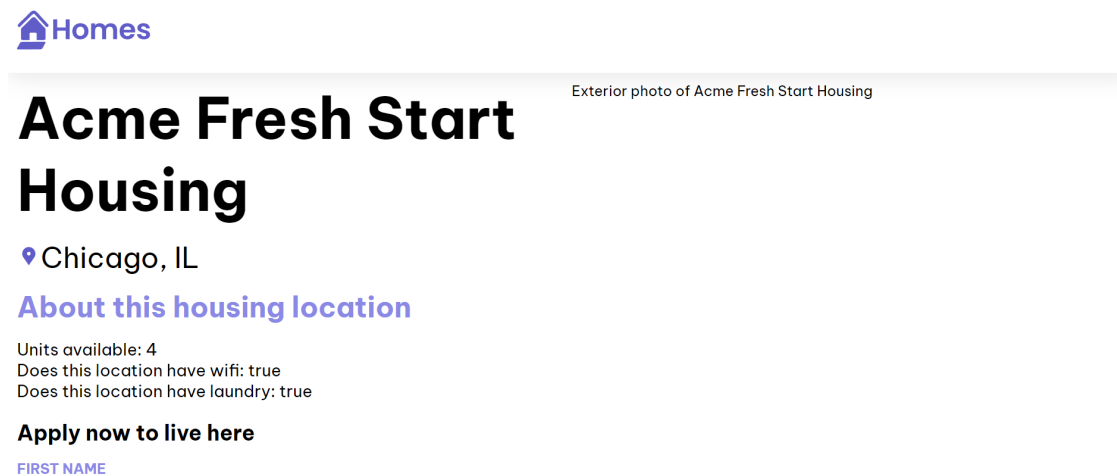


Figura A.6: Detalle de la ejecución de la aplicación de ejemplo avanzada

Gracias a los conceptos aprendidos durante el desarrollo de esta simple aplicación podremos crear el frontend que necesitamos con facilidad.

Una vez hemos sentado las bases, nuestro siguiente punto será investigar, el software de mongoDB para de esta manera ser capaces de almacenar toda la información que necesitamos para el manejo de nuestra aplicación de manera persistente. Al hacerlo en MongoDB la información podrá estar duplicada, sin embargo, esto mejorará las velocidades de acceso a los datos de manera drástica sobre todo en nuestro caso donde es fundamental tener la mínima latencia posible para este tipo de tareas.

Para estas pruebas necesitaremos descargar MongoDB [25] y simplemente seguir los pasos de la instalación.

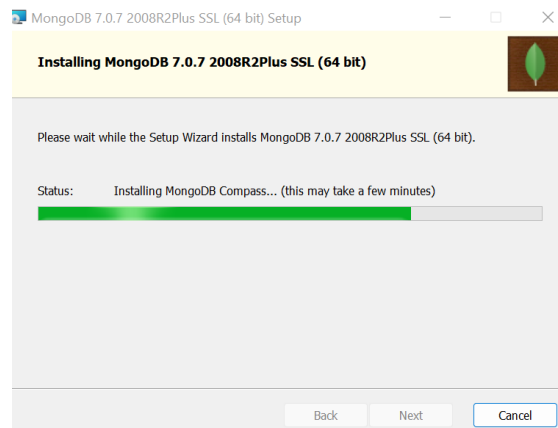


Figura A.7: Detalle de la instalación de MongoDB

Una vez instalamos MongoDB aparecerá esta aplicación (mientras el servidor se ejecutará en segundo plano):

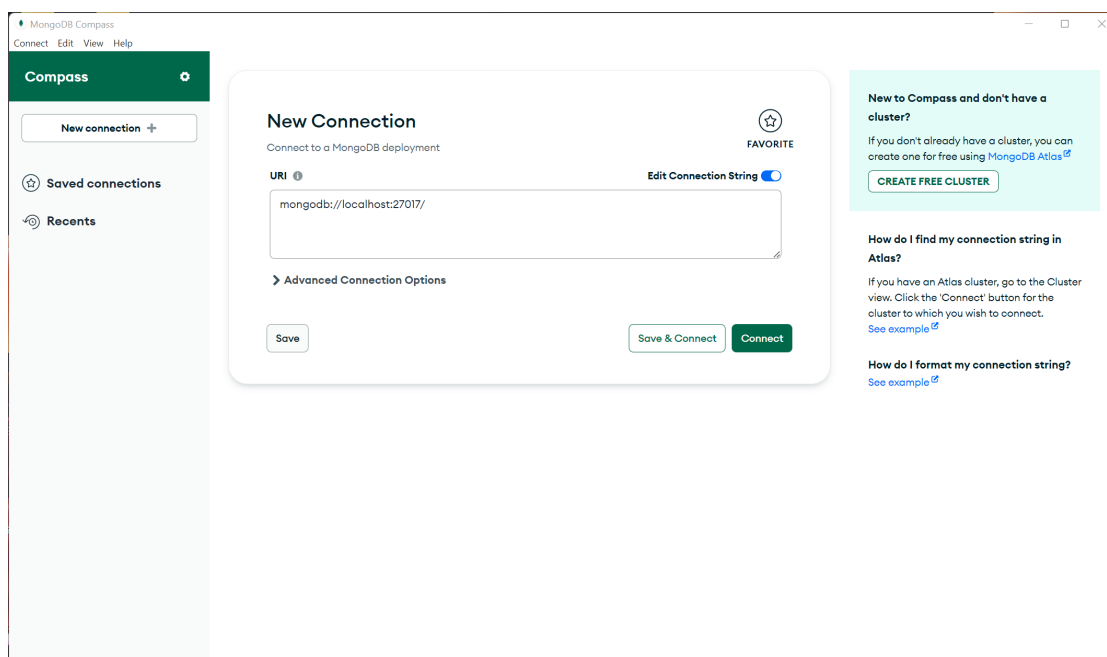


Figura A.8: Nueva conexión de MongoDB

Como he comentado previamente, se trata de la interfaz de usuario que nos permite operar con la base de datos.

Desde un repositorio cualquiera he cogido algunos datos aleatorios para hacer pequeñas pruebas [26] después he intentado filtrar los datos haciendo pruebas con algunas queries simples como la siguiente: `{"airTemperature.value": {"\"$lt\": -2.3}}`.

Creamos proyecto en angular con el comando del cliente (`ng new`):

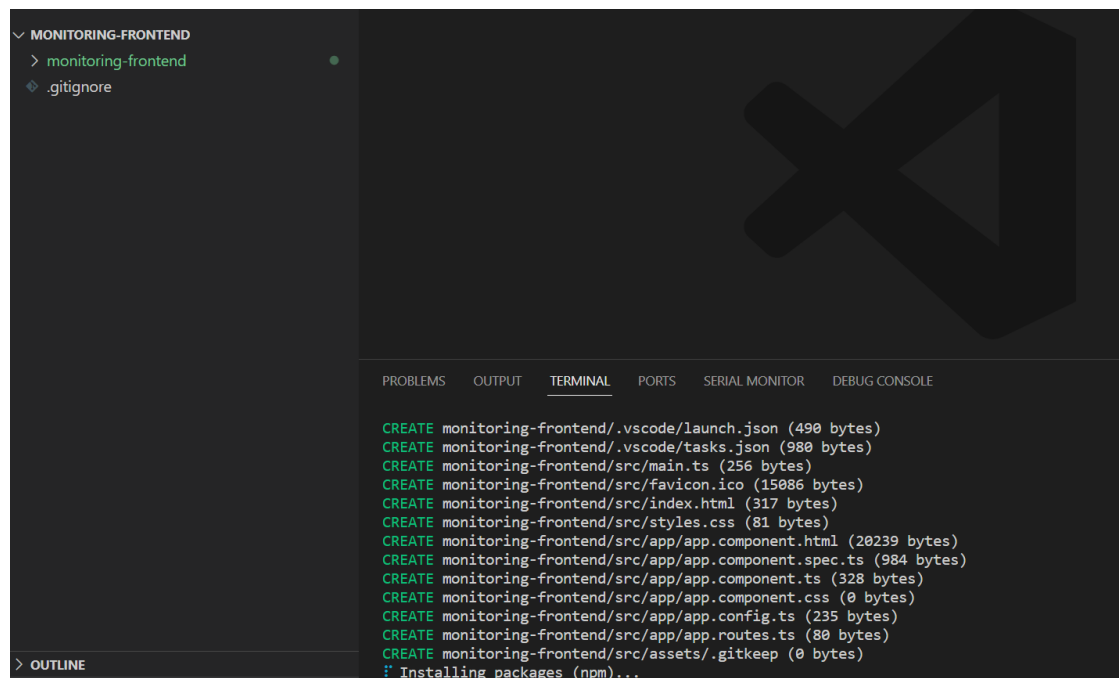


Figura A.9: Nuevo proyecto en Angular

Después crearemos los formularios de registro y login como componentes, será tan fácil como crear el formulario con ayuda de Angular y darle los estilos al formulario con angular material. Creamos el componente con (`ng generate component`):

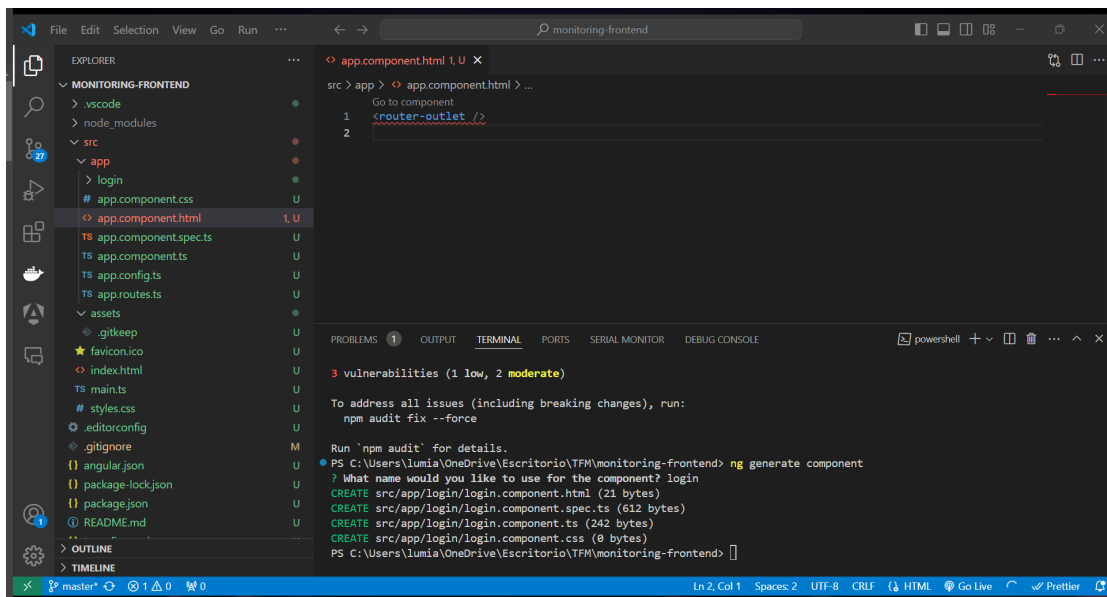


Figura A.10: Creando nuevo componente en Angular

Para que estos formularios tengan funcionalidad real, será necesario que se creen los servicios para que se envíen las peticiones necesarias al backend y después a la base de datos.

Una vez hecho el login con los componentes de login y registro, así como sus servicios para enviar la petición al backend.

```
registerUser( username: string, password: string): Observable<any>
{
    const id = uuid.v4().toString();
    const user = new UserLogin(id, username, password);
    return this.http.post('/api/register', user, this.httpOptions);
}
```

Todos los servicios CRUD (*Create, Read, Update, Delete*) tendrán dicha forma o al menos una muy parecida pues si no es la creación del objeto en la base de datos no necesitaremos asignar un nuevo Id al modelo. El método consistirá en la creación del modelo y el envío de la llamada la backend con el método HTTP pertinente.

Una vez se hace el login habrá que crear una cookie para mantener la sesión abierta y hacer la redirección al tablero principal donde se verán las organizaciones y los dispositivos junto con todas las tareas y sus métricas. Una versión simplificada del código sería la siguiente:

```
submitForm() {  
  this.apiService.loginUser(  
    <string>this.registerForm.value.username,  
    <string>this.registerForm.value.password  
  ).subscribe({  
    next:(response) => {  
      this.storageService.saveUser(response.id);  
      this.router.navigate(["/home"]);  
    },  
    error: (e) => {  
      console.log(e)  
    }  
  })  
}
```

Este tablero será un solo componente que llamaremos Home. El primer paso será crear un formulario para crear organizaciones que serán un elemento expansible de Angular Material. Una vez tenemos dicho formulario con todo lo necesario para crear las organizaciones en la base de datos (formulario, servicio y petición al backend, así como el modelo y el endpoint del backend):

```
@PostMapping("/organizations/save")  
public Organization saveOrganization(  
  @RequestBody Organization organization  
) {  
  return organizationRepository.saveOrganization(organization);  
}
```

Para poder almacenar la organización necesitaremos de un modelo para la base de datos no relacional en java como ya especifiqué para el dispositivo (*a continuación muestro una versión simplificada del modelo de la organización, sin getters ni setters*):

```
@Document(collection = "Organizations")
public class Organization {
    @Id
    private String id;

    @Indexed(unique = true)
    private String name;
    private String streetAddress;
    private String userId;

    public Organization(String id,
                        String name,
                        String streetAddress,
                        String userId
    ) {
        this.id = id;
        this.name = name;
        this.streetAddress = streetAddress;
        this.userId = userId;
    }
}
```

No solo el modelo sino que necesitaremos de un repositorio como servicio para poder interactuar con la base de datos, tal y como se especifica:

```
@Service
public class OrganizationRepository {

    @Autowired
    private OrganizationInterface repository;

    public Organization saveOrganization(Organization organization) {
        return repository.save(organization);
    }

    public Organization getOrganizationById(String id) {
        return repository.findById(id).get();
    }

    public List<Organization> getOrganizationsByUser(String userId) {
        return repository.findById(userId);
    }
}
```

```
public List<Organization> getAllOrganizations(){
    return repository.findAll();
}

public void deleteOrganization(String id) {
    repository.deleteById(id);
}
}
```

Además de una interfaz que permitirá la interacción directa con MongoDB gracias a las librerías de Java:

```
public interface OrganizationInterface
    extends MongoRepository<Organization, String> {
    List<Organization> findByUserId(String userId);
}
```

Tras crear unas cuantas organizaciones para mostrarlas haremos uso de la directiva **for** de Angular, con la que crearemos tantos elementos como organizaciones haya:

```
...
@for ( organization of organizations.organizations;
      track organization)
...

```

Después tendríamos que crear otro formulario para los dispositivos y todo lo necesario siguiendo la estructura comentada (formulario, servicio, bucle For para el display en el home y a nivel de backend; la interfaz, el servicio y el modelo).

```
...
@for (device of devices.devices;
      track device)
...

```

Una vez tenemos los dispositivos, tendremos que crear sensores y tareas que tendrán dicho sensor asignado, para ello dentro de las tareas añadiremos un botón para abrir un diálogo con el que crear una nueva tarea para el mismo dispositivo o un sensor que, aunque lo creamos desde este, podremos usarlo en todos. En el caso de las tareas cabe destacar, que el estado se actualizará en tiempo real gracias al websocket abierto entre el backend y el frontend, este websocket permite que se tenga constancia del estado de las tareas en todo momento según el valor de este atributo en la base de datos, para esto crearemos dicho websocket en el frontend:

```
webSocket: WebSocket =  
    new WebSocket('ws://localhost:8080/taskStatus');
```

Al inicio del tablero, se definirá que se hará con el evento de tipo `onMessage` del `web-socket`:

```
this.webSocket.onmessage = (event) => {  
    const task = JSON.parse(event.data);  
    this.tasks.tasks = this.tasks.tasks.map((t) => {  
        if(t.id === task.id) {  
            return task;  
        }  
        return t;  
    });  
}
```

Cuando este evento se dispara se mapearán los valores de las `tasks` de la base de datos para el dispositivo seleccionado:

```
getBorder(status: string) {  
    switch (status) {  
        case 'ACTIVE':  
            return '2px solid green';  
        case 'INACTIVE':  
            return '2px solid red';  
        case 'PENDING':  
            return '2px solid yellow';  
        default:  
            return 'black';  
    }  
}
```

Dando un borde u otro dependiendo del estado. Desde el backend definiremos dos clases para su configuración:

- `TaskStatusWebSockerHandler.java`
- `WebSocketConfig.java`

La segunda, será configurar el `websocket` como tal, declarando cual será el método de `handler` una vez se establece la conexión usando la inyección de dependencias con el `handler` definido en la primera clase, de esta manera el código se vería de la siguiente manera:

```
@Override
@SuppressWarnings("null")
public void registerWebSocketHandlers(
    WebSocketHandlerRegistry registry
) {
    registry.addHandler(taskStatusWebSocketHandler(), "/taskStatus")
        .setAllowedOrigins("*");
}

@Bean
public WebSocketHandler taskStatusWebSocketHandler() {
    return new TaskStatusWebSocketHandler();
}

@Override
@SuppressWarnings("null")
public void afterConnectionEstablished(WebSocketSession session)
    throws Exception {
    while(true){
        Thread.sleep(1000);
        taskRepository.getAllTasks().forEach(task -> {
            try {
                session.sendMessage(
                    new TextMessage(
                        objectMapper.writeValueAsString(task)
                    )
                );
            } catch (Exception e) {
                e.printStackTrace();
            }
        });
    }
}
```

Como se puede ver el método se encarga de tomar todas las tareas y enviarlas al frontend para poder actualizar sus estados.

Finalmente, haremos uso de [RabbitMQ \[27\]](#) como message broker, de manera que se permita la comunicación entre el dispositivo y el backend sin que se pierda información en caso de que la comunicación termine. Para ello tendremos un Listener en el lado del backend para el Status y las Métricas y un listener en el lado del dispositivo para las tareas. Este sería el sender en el lado del dispositivo:


```
public void sendToTasks(String message) {
    System.out.println("Task message is sent to RabbitMQ");
    amqpTemplate.convertAndSend(RabbitMqConfiguration.IOT_EXCHANGE,
        RabbitMqConfiguration.TASKS,
        new Message(message.getBytes()));
}

public void sendToMetrics(String message) {
    System.out.println("Metrics message is sent to RabbitMQ");
    amqpTemplate.convertAndSend(RabbitMqConfiguration.IOT_EXCHANGE,
        RabbitMqConfiguration.METRICS,
        new Message(message.getBytes()));
}

public void sendToStatus(String message) {
    System.out.println("Status message is sent to RabbitMQ");
    amqpTemplate.convertAndSend(RabbitMqConfiguration.IOT_EXCHANGE,
        RabbitMqConfiguration.STATUS,
        new Message(message.getBytes()));
}
```

Y esta sería la configuración:

```
public static final String METRICS = "METRICS";
public static final String STATUS = "STATUS";
public static final String TASKS = "TASKS";
public static final String IOT_EXCHANGE = "IOT_EXCHANGE";

@Bean
Queue queueMetrics() {
    return new Queue(METRICS, false);
}

@Bean
Queue queueStatus() {
    return new Queue(STATUS, false);
}

@Bean
Queue queueTasks() {
    return new Queue(TASKS, false);
}
```

```
@Bean
DirectExchange directIoTExchange() {
    return new DirectExchange(IOT_EXCHANGE);
}
```

```
@Bean
Binding bindTasksQueueToExchange() {
    return BindingBuilder
        .bind(queueTasks())
        .to(directIoTExchange())
        .withQueueName();
}
```

```
@Bean
Binding bindMetricsQueueToExchange() {
    return BindingBuilder
        .bind(queueMetrics())
        .to(directIoTExchange())
        .withQueueName();
}
```

```
@Bean
Binding bindStatusQueueToExchange() {
    return BindingBuilder
        .bind(queueStatus())
        .to(directIoTExchange())
        .withQueueName();
}
```

```
@Bean
public AmqpTemplate amqpTemplate(ConnectionFactory connectionFactory) {
    final RabbitTemplate template = new RabbitTemplate(connectionFactory);
    return template;
}
```

Sin embargo, para que todo esto sea posible, primero es necesario instalar el servidor de rabbitmq, para ello usaremos el instalador de paquetes de windows que en mi caso es Chocolatey y ejecutaremos el comando “choco install rabbitmq” que correrá por defecto en el puerto localhost:5672 tal y como se puede ver en el fichero de configuración:

```
#RabbitMQ Configuration
spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
```

Por defecto, tenemos también el RabbitMQ manager desplegado en el puerto 15672 que nos permitirá ver las colas y topics actuales para nuestra aplicación como se observa en la siguiente página.

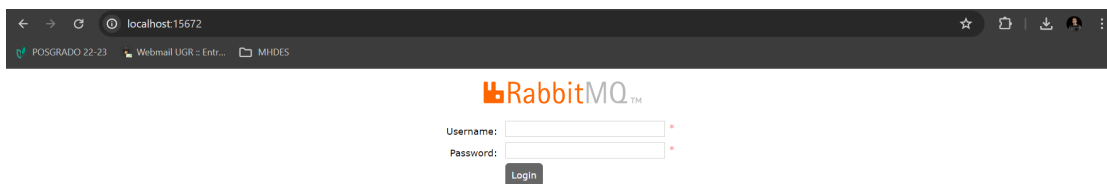


Figura A.11: RabbitMQ Manager

92 A.3. Gestión del proyecto. Planificación temporal. Diagrama de Gantt

Como podemos ver en la siguiente imagen:

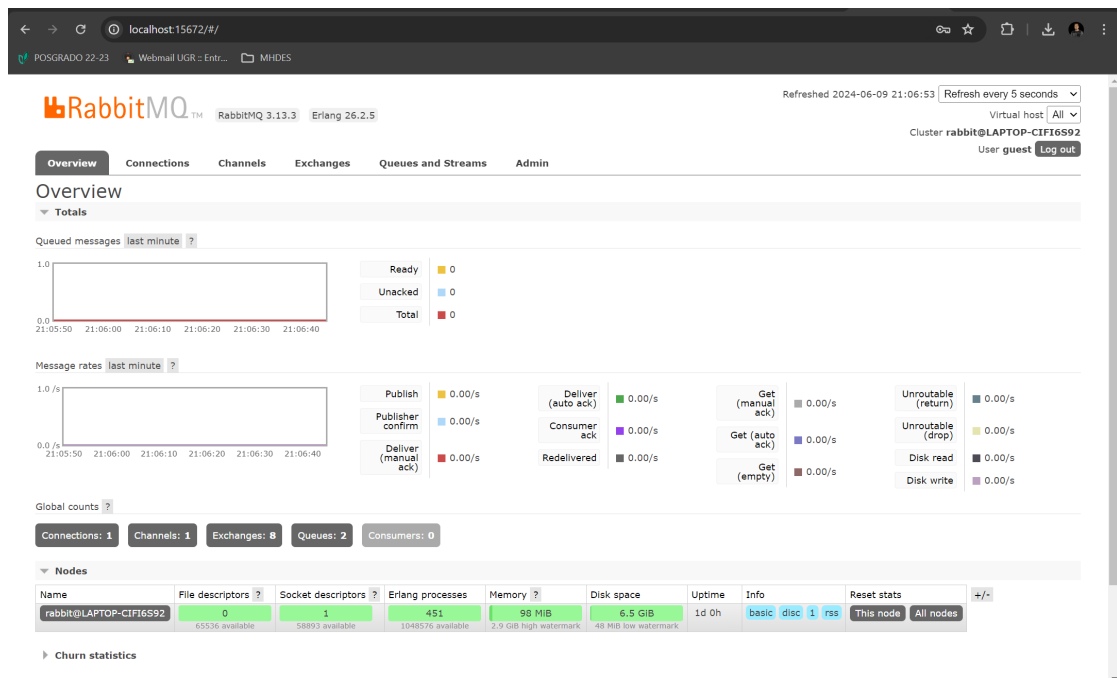


Figura A.12: RabbitMQ Manager página principal

Finalmente tendremos que ejecutar el fichero en C de la carpeta monitoring-iot en el dispositivo que queremos comunicar con nuestro backend:

```
export DEVICE_ID=<device id obtained from the web application>

gcc amqListenernPublisher.c -o amqListenernPublisher -lrabbitmq
-ljansson

chmod +x amqListenernPublisher

./amqListenernPublisher
```

A.3.2. Planificación temporal

En esta sección evaluaremos las tareas llevadas a cabo a lo largo del tiempo en cada uno de los sprints así como las reuniones que se realizaron tutor-alumno para acordar la realización de las mismas, dichas reuniones siempre se han intentado que tuviesen lugar cada dos semanas (duración acordada de los sprints), aunque en ocasiones por festivos o problemas personales ha sido complicado mantener estos plazos de tiempo con exastitud.

Sprint Review 1 - 15 nov 2023

Durante este primer sprint se definieron cuales eran las tareas que se debían de llevar a cabo, también se revisaron algunos documentos del algoritmo EDF sin entrar en mucho detalle y se realizó una lectura de algunos de los trabajos realizados durante los años anteriores. En esta reunión en específico se discutió que para dar por finalizada cada una de las tareas de investigación era necesario la creación de algunas páginas o textos que definieran cuales han sido, sobre cada uno de los temas, los principales hallazgos para que la elaboración de la memoria final fuese mucho más fácil.

Sprint Review 2 - 29 nov 2023

En este sprint el principal objetivo fue crear la documentación de las tareas de investigación realizadas en el sprint previo y relacionadas con la estructura general del trabajo así como del algoritmo EDF. Se discutió que los documentos no tenían la extensión que se esperaba y que se desarrollarían con mayor detalle en sprints futuros.

Sprint Review 3 - 13 dic 2023

Durante esta reunión se abordó cuales eran los principales objetivos de todo el trabajo que se iba a realizar, incluyendo, la memoria, estado de arte entre otros (en general, me refiero a las épicas que constituyen el diagrama de Gantt [A.3.4](#)). Además se comentó que era necesario la creación de los requerimientos para de esta manera elaborar las tareas. [3.1](#)

Sprint Review 4 - 27 dic 2023

Durante este sprint creé los distintos requerimientos, aumenté la documentación respecto de cual será la interfaz de usuario [3.4](#) y la base de datos [3.2](#).

Sprint Review 5 - 12 ene 2024

Esta fue la primera reunión tras las vacaciones de navidad, en ella se discutieron los resultados de los requerimientos, para los cuales el tutor puso los comentarios oportunos para aquellos que necesitasen de corrección. También se discutió que sería necesaria una licencia de QNX para la realización o desarrollo de algunas de las tareas del proyecto cuya obtención estaría a cargo del tutor también en este caso (esta licencia trataría principalmente del sistema operativo del que constarían nuestros microcontroladores de cara a la ejecución de tareas de tiempo real)

Se debatió también en esta misma reunión que una de las tareas a realizar y que se olvidó, es hacer un pequeño resumen de cada una de las reuniones que tenemos. Se comentó también que mis tareas para este sprint serían la realización de los resúmenes y conseguir realizar con la mayor extensión una redacción de los objetivos finales del TFM y el estado del arte del mismo.

Sprint Review 6 - 29 ene 2024

Segunda reunión del año y en ella no se discutieron demasiados aspectos del trabajo, simplemente que se seguirían trabajando en las secciones que fuese posible no relacionadas con el desarrollo, de esta manera en los siguientes sprints podríamos centrarnos en el desarrollo de la aplicación en su totalidad.

Sprint Review 7 - 12 feb 2024

En esta reunión he mostrado parte del contenido que he estado realizando sobre el archivo de OneDrive (word) sobre el que se aplicarían distintos estilos para obtener la versión final (finalmente se decidió que la memoria se realizara en latex). Hemos hablado de la bibliografía y del formato que debería de tener esta. También se ha evaluado que la estructura del trabajo sea la correcta y queda pendiente por evaluar la complejidad de cada una de las tareas que se va realizando (aunque esto no ocurrió finalmente). También se mencionó que las tareas del desarrollo debían ser lo más simple posibles de manera que el desarrollo de la aplicación final sea lo más ágil posible.

Sprint Review 8 - 28 mar 2024

Este sprint en lugar de durar dos semanas se ha extendido un total de más de un mes por motivos personales y distintas complicaciones a la hora de agendar las reuniones. Durante este sprint he avanzado ampliamente en la documentación sentando las bases teóricas de nuestro proyecto además de algunos de los detalles de las partes que se llevarán a cabo en el desarrollo. Este mismo, comenzará en el frontend en el próximo sprint, para el que realizaré tareas que irán desde aprender las bases de Angular, que es la tecnología que usaremos, hasta el desarrollo de cada una de las funcionalidades definidas en los requisitos expresos en la memoria [3.1](#).

Sprint Review 9 - 11 abr 2024

En este sprint se avanzó ampliamente el desarrollo de la aplicación tal y como se comentó en el sprint previo, se ha desarrollado el frontend prácticamente en su totalidad.

Sprint Review 10 - 25 abr 2024

En este sprint se ha completado el desarrollo de la aplicación final casi en su totalidad, acordamos que el tutor se encargaría del desarrollo de un template para la gestión de tareas en el dispositivo de tiempo real y que yo implementaría con la ayuda de mockito o cualquier otro message broker (finalmente fue RabbitMQ), la comunicación entre el dispositivo y nuestro backend, así se podrán gestionar las tareas de este y desde el backend, los estado y métricas de manera asíncrona sin perder información en caso de que ocurriese cualquier desconexión.

Sprint Review 11 - 27 may 2024

Este sprint se ha extendido más de lo habitual debido a que he sido capaz de trabajar en distintas tareas del desarrollo sin tener ninguna reunión con el tutor. Sin embargo, en esta reunión hemos abordado cuales serán los pasos a realizar durante las próximas semanas que serán las últimas antes de la entrega del proyecto, acordamos que mejoraría el contenido de la memoria hasta que se acordase cual deberá ser el formato de la misma y en cuanto a la parte de desarrollo, sigue pendiente la implementación del software de gestión de tareas en el lado del microcontrolador.

Sprint Review 12 - 14 jun 2024

En esta reunión se acordó que finalmente la memoria se hará en latex debido a que la plantilla para la memoria en word tardará un poco más en subirse y hacerlo en latex hará también demostración del conocimiento del contenido del máster pues el aprendizaje de esta tecnología es parte de una de las asignaturas del mismo. Además se acordó que la

aplicación se desplegaría en uno de los servidores de la escuela. Durante las siguientes semanas habrán numerosas comunicaciones entre el alumno y el tutor de cara a la recta final del desarrollo de este proyecto.

A.3.3. Lista de tareas

Clave	Resumen	Sprint de Finalización
IM-1	Diseño de arquitectura ejemplo para el sistema	Tablero Sprint 3
IM-2	Investigación del algoritmo de planificación EDF - Introducción	Tablero Sprint 3
IM-3	Diseño conceptual de la interfaz de usuario	Tablero Sprint 5
IM-4	Diseño de la base de datos del sistema	Tablero Sprint 5
IM-5	Investigación del estado del arte	Tablero Sprint 12
IM-8	Propuesta TFM	Tablero Sprint 3
IM-14	Requerimientos del sistema	Tablero Sprint 5
IM-15	Librerías Java que podrían ser útiles	Tablero Sprint 12
IM-19	Creación objetivos (general y específicos) [Parte 1]	Tablero Sprint 7
IM-20	Actualizar la sprint review de los sprints pasados	Tablero Sprint 7
IM-22	Preparar entorno de desarrollo y licencia de QNX	Tablero Sprint 12
IM-23	Documentación relacionada con el middleware y la asignatura de sistemas de tiempo real	Tablero Sprint 12
IM-24	Instalar Node JS y crear una aplicación básica en Angular	Tablero Sprint 9
IM-25	Crear al completo la aplicación de ejemplo angular 17 para entenderlo	Tablero Sprint 9
IM-26	Investigar MongoDB y crear pequeño ejemplo para aprender sobre él	Tablero Sprint 9
IM-27	Crear base de datos del proyecto en MongoDB	Tablero Sprint 9
IM-28	Inicializar proyecto en Angular 17	Tablero Sprint 10
IM-29	El usuario puede registrarse	Tablero Sprint 10
IM-30	El usuario puede iniciar sesión	Tablero Sprint 10
IM-31	Crear dockerfile para mongoDB y un volumen	Tablero Sprint 9
IM-34	El usuario puede visualizar la información del sistema (organizaciones/dispositivos)	Tablero Sprint 10
IM-35	El usuario puede visualizar el listado de organizaciones	Tablero Sprint 10
IM-36	El usuario puede visualizar el listado de dispositivos	Tablero Sprint 10

IM-37	El usuario puede visualizar los detalles de una organización	Tablero Sprint 10
IM-38	El usuario puede visualizar el detalle de un dispositivo	Tablero Sprint 10
IM-40	El frontend puede acceder al listado de dispositivos y sus detalles	Tablero Sprint 10
IM-41	El frontend puede acceder al listado de organizaciones y sus detalles	Tablero Sprint 10
IM-42	El frontend puede registrar a un usuario	Tablero Sprint 9
IM-43	El frontend puede hacer log in del usuario	Tablero Sprint 9
IM-44	El usuario puede registrar una organización	Tablero Sprint 10
IM-45	El usuario puede registrar un dispositivo	Tablero Sprint 10
IM-46	El usuario puede definir atributos como IP/URL y puerto (Dispositivos)	Tablero Sprint 10
IM-47	El frontend podrá almacenar todos los atributos (Dispositivos)	Tablero Sprint 10
IM-48	El usuario puede crear tareas	Tablero Sprint 10
IM-49	El frontend podrá almacenar tareas en la BBDD	Tablero Sprint 10
IM-50	El usuario puede definir sensores	Tablero Sprint 10
IM-51	El frontend podrá almacenar sensores	Tablero Sprint 10
IM-52	Explorar la implementación de tareas periódicas abstracta en C o en Java	Tablero Sprint 12
IM-53	Implementar Consumer y Publisher con un Message Broker entre el Device y el backend	Tablero Sprint 11

A.3.4. Diagrama de Gantt

En la siguiente página podremos ver como las tareas anteriores se han ido desarrollando en el tiempo. Es curioso que a pesar de que yo he realizado el diagrama de Gantt manualmente [A.3.4](#), Jira tiene funcionalidad propia para crear este tipo de diagramas en el tiempo [A.14](#).

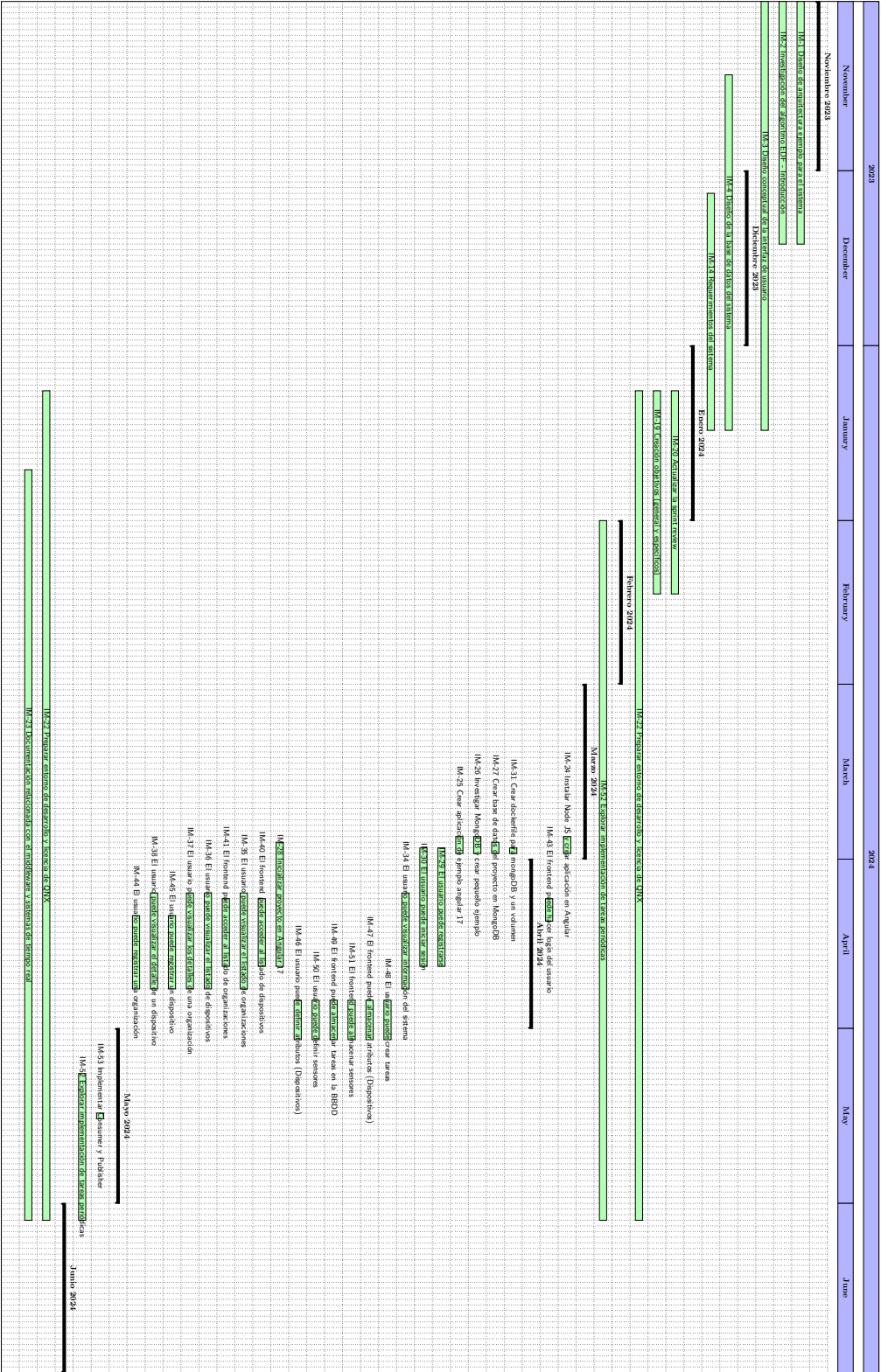


Figura A.13: Diagrama de Gantt para el proyecto desde noviembre de 2023 hasta junio de 2024.



Figura A.14: Diagrama de Gantt desde Jira

Apéndice B

Innovación

Este proyecto presenta un desarrollo de software aunque simple, muy innovador, pues a pesar de que ya existen soluciones para este tipo de problemas de gestión para dispositivos IoT (como veremos en la sección [B.1.3](#)), no muchas de ellas presentan la necesidad de ejecutar tareas en tiempo real, es aquí, donde prevalece nuestra solución. Además este tipo de soluciones no tienen en cuenta la pequeña/mediana empresa (PYME) o particulares pues este tipo de servicios solo se los pueden permitir fábricas y empresas que cuentan con una infraestructura hardware y software suficiente.

B.1. Estudio del Mercado

En esta sección evaluaremos de manera más extensa y en cada una de las secciones, las características comerciales del software desarrollado, identificaremos, oportunidades, competidores y segmentos de clientes específicos, entre otras. En general, se demostrará todo el potencial de nuestro software como producto, fortalezas y debilidades.

B.1.1. Segmentos de Clientes

Los principales clientes con los que contaría nuestro software serían, pequeñas y medianas empresas que no cuentan con los recursos económicos, software o hardware de los que necesitan las grandes empresas (y principales competidores) como son Amazon, Microsoft o Google Cloud. Otros de nuestros clientes, serían particulares dispuestos a domotizar o automatizar ciertos servicios en su casa o incluso en cultivos y otros entornos sin carácter lucrativo. Por ejemplo, un conocido intentó domotizar un cultivo para conocer el estado de las líneas de riego, con esta aplicación podría colocar sensores en dicha línea y llevar control no solo de las métricas de caudal obtenidas, sino también del estado de los sensores para poder reemplazarlos en el momento en que alguno de estos dejase de funcionar.

B.1.2. Competidores

AWS IoT

Como ya he llegado a comentar en la sección 1.1, Amazon cuenta con multitud de servicios que permiten al usuario gestionar dispositivos IoT pero también tienen multitud de limitaciones y al igual que una de las ventajas de la nube es su versatilidad para ser escalable vertical y horizontalmente, esta escalabilidad hay que pagarla y el coste en el caso del IoT consta de lo siguiente:

- **Número de minutos de conexión.** En cuanto a tiempo de conexión, cada minuto será 0.08\$ por 1000000 min que puede parecer mucho tiempo pero teniendo en cuenta que un mes tiene 44000 min aproximadamente y cada dispositivo tendría que estar conectado las 24h, no necesitaríamos de tantos dispositivos para llegar al 1000000 de min.
- **Número de mensajes.** En media, cada mensaje MQTT y HTTP costará 0.70\$ a partir de los 5 mil millones de mensajes, que teniendo en cuenta el número de mensajes que se procesan en este tipo de servicios no son tantos.
- **Numero de operaciones de registro de dispositivos.** En cuanto al registro de dispositivos sería 1.25\$ por 1000000 de peticiones de registro, es más complicado a que se llegue a este número registros en PYME y en casos particulares.
- **Reglas desencadenadas y acciones aplicadas.** Al igual que en el caso anterior, el número de reglas y acciones aplicadas no sería el suficiente en nuestro sector de clientes como para que tenga un impacto real.

Amazon es un gran competidor por todo el potencial hardware que tiene en lo que respecta a escalar el sistema pues en nuestro caso, el usuario, necesitará de cierta infraestructura para el despliegue de nuestra aplicación de gestión, sin embargo, el coste de esta como ya hablaremos en la sección de B.1.9 para el usuario será fijo.

Azure IoT

Azure IoT en este caso es uno de los numerosos servicios que ofrece la nube de Microsoft y al igual que en el caso de Amazon, las fortalezas con las que cuenta es la integración con el resto de servicios de esta nube y su gran potencial a nivel de infraestructura. Sin embargo, puede ser costoso y complicado para las PYMEs y particulares.

Con Google Cloud IoT (Cloud IoT Core) ocurre lo mismo que con los servicios en la nube que he mencionado hasta ahora. Sin embargo, existen otros servicios que no tienen porque ser servicios en la nube, como puede ser Particle o ThingsBoard. Estos dos últimos son los competidores reales (además de otros) pues al igual que en nuestro caso, requieren de más recursos y conocimientos técnicos para su gestión, además no tienen la potencia y escalabilidad de las soluciones cloud, aunque tienen, flexibilidad, personalización y adecuación para particulares y cualquier caso de uso.

B.1.3. Análisis DAFO - FODA

Como en cualquier análisis DAFO, analizaremos las debilidades, amenazas, fortalezas y oportunidades de nuestro producto:

Debilidades de la empresa

En cuanto a las debilidades que tenemos pueden ser la limitación de recursos respecto a los principales competidores, la baja visibilidad y la dependencia de infraestructura de terceros, sin embargo, este último punto no es del todo cierto, pues la idea es que a pesar de que el producto actual tiene ese enfoque, el producto final, se vendería como un software con licencia y no como un servicio. Además dependemos de que la red que conectan los dispositivos con nuestro software sea lo suficientemente robusta allí donde se despliegue. Por ejemplo, en el caso que he comentado antes de monitorizar un regadío podría ser complicado por las condiciones climáticas mantener la conexión o incluso de teneral. Otras empresas como he comentado, podrían ofrecer servicios e instalación de red en el sitio por la cantidad de recursos que tienen tanto económicos como hardware o software.

Amenazas

La competencia está muy solidificada en el mercado, son famosas y ofrecen soluciones muy robustas, además estas cuentan con la capacidad de recursos humanos para adaptarse continuamente a los nuevos cambios tecnológicos mientras que para nosotros será más complicado.

Fortalezas

El enfoque de la ejecución de tareas en tiempo real como he comentado en este trabajo, es lo que nos diferencia principalmente del resto de soluciones, además estará plenamente adaptado y personalizado para las PYMEs, ofreciendo una solución mucho más asequible y focalizada a las necesidades del cliente.

Oportunidades

Las oportunidades son principalmente, el crecimiento del mercado de IoT que es cada vez mayor en diferentes sectores además de que estaríamos explotando un nicho de mercado (PYMEs y particulares) que las grandes empresas no están dispuestas a hacer.

B.1.4. Propuesta de Valor

Dentro del valor del producto encontramos su accesibilidad pues al presentar un bajo costo y modelos de suscripción flexible para pagar solo por las funcionalidades que necesitan hacen el producto muy atractivo para el nicho de mercado que se pretende atacar evitando que las PYMEs o particulares tengan que hacer inversiones iniciales significativas, sino que nuestro producto se va amoldando a las necesidades del cliente. Nuestro principal valor es la ejecución de tareas en tiempo real, esto ofrece un monitoreo continuo del sistema en tiempo real, volviendo al ejemplo de nuestro cliente que cuenta con un sistema de riego, no basta solo con que pudiese ver el estado de los sensores sino que este sea en tiempo real pues una fuga en el mismo podría costarle muy caro. Buscamos con nuestro producto, una solución sencilla, sin toda la complejidad de las grandes empresas a un problema común y creciente dado que el paradigma de IoT también lo es. Además al tratarse de un producto que permite su instalación fuera de las nubes de terceros, tendremos un sistema robusto pues la comunicación será totalmente cerrada dentro del sistema sin necesidad de comunicarse con internet. En resumen, estamos ante una solución para la gestión de dispositivos IoT en tiempo real con un enfoque de simplicidad, seguridad y personalización con el que no cuentan las grandes proveedoras de servicios en la nube, potenciando que PYMEs y particulares puedan aprovechar las ventajas de todo lo que envuelve IoT sin los desafíos de conocimientos y recursos de los que necesitan las otras soluciones.

B.1.5. Recursos Clave

En cuanto a los recursos clave de nuestro proyecto, necesitamos lo que cualquier empresa de software necesitaría:

- Un departamento de desarrollo que incluirá.
 - Diseñadores de UX/UI.
 - Desarrolladores de software.
 - Testers.
 - Analistas de datos.

Y con menor importancia en un primer momento:

- Un departamento de Ventas.
- Un departamento de Recursos Humanos.
- Un departamento de Marketing

Por otro lado, necesitaremos de toda la infraestructura tecnológica para el despliegue y testeo de la aplicación, además de multitud de sensores y dispositivos para asegurar que su compatibilidad con nuestro sistema.

B.1.6. Actividades Clave

Las actividades clave siempre serán como ya he dicho en otra ocasión, actividades de I+D con la intención de mantener a las ultimos avances nuestra aplicación. Así, también deberemos mantener al día la seguridad de nuestra aplicación de manera que nadie externo tenga acceso o control a los dispositivos conectados a la misma. También será importante mantener cierto soporte técnico para los nuevos usuarios de manera que puedan integrar sus dispositivos fácilmente.

B.1.7. Socios y Alianzas Clave

Un socio clave podría ser Vodafone o cualquier otro proveedor de comunicaciones, de manera que no tuviésemos que depender de sistemas de red de terceros para el despliegue de nuestro sistema.

B.1.8. Estructura de Costes

Costes de la solución actual

En cuanto a los costes de la solución actual podemos dividir:

- (Horas de trabajo) $40h * 25€ = 1000€$
- Ordenador para el desarrollo y despliegue inicial = 600€. Cabe destacar que para el despliegue inicial se han tenido en cuenta os precios de los servicios que se han especificado durante el desarrollo de este trabajo para AWS [\[28\]](#).
- Microcontrolador (alrededor de 60€)
- Sensor (Aproximadamente 1€)

En total por todo sería: 1661€ aproximadamente.

Costes del producto final

El producto final puede ser muy costoso teniendo en cuenta que habría que multiplicar los costes de la solución actual, tantas veces como se quiera escalar el sistema, por ejemplo, si se necesitan de miles de dispositivos, o cientos de desarrolladores para mantener el software actualizado a las últimas innovaciones.

B.1.9. Vías de Ingresos

Se usará un sistema de licencia de software, que se podrá hacer de manera anual, mensual o de por vida, dependiendo de las necesidades del cliente y dependerá de la funcionalidad que necesite. La versión por vida será la más cara (barata en cuanto a \$/mes), aunque será la más estandar en cuanto a funcionalidad. Por su lado, si el pago se realiza de manera mensual o anual se permitirán cuatro tipos de suscripción (gratuita, standard, premium, ultra), la gratuita y la standard contarán con la misma funcionalidad, solo cambiará el número de dispositivos que puede gestionar nuestro sistema así como el número de mensajes. Por otro lado la premium daría acceso a la máxima funcionalidad de la aplicación, mientras que la ultra permite la personalización de la aplicación, de esta manera, se adaptaría el máximo posible a las necesidades de cliente.

B.1.10. Canales de Distribución

El canal de distribución será la página web oficial de la empresa.

B.1.11. Relación con los Clientes

La relación con el cliente será cercana con un soporte lo más cercano al 24/7 posible, para asegurar de que todos los sistemas funcionan correctamente con nuestro software, aunque no sea una persona, si debería de haber al menos un sistema de feedback para mantener el control de cuales puedan ser los defectos de la aplicación.

B.1.12. Descripción del Producto Mínimo Viable

El producto mínimo viable, sería el llevado a cabo en este proyecto, queda mucho trabajo por hacer para que sea un producto completo (como se menciona en el apartado de 4.2 y usable dentro del marco de suscripción flexible propuesto pues si no existe funcionalidad suficiente que diferencien las distintas cuotas, estas carecerían de sentido.

B.1.13. Roadmap de Desarrollo

Seguiríamos el desarrollo que sigue cualquier proyecto de software, en este caso Devsecops:

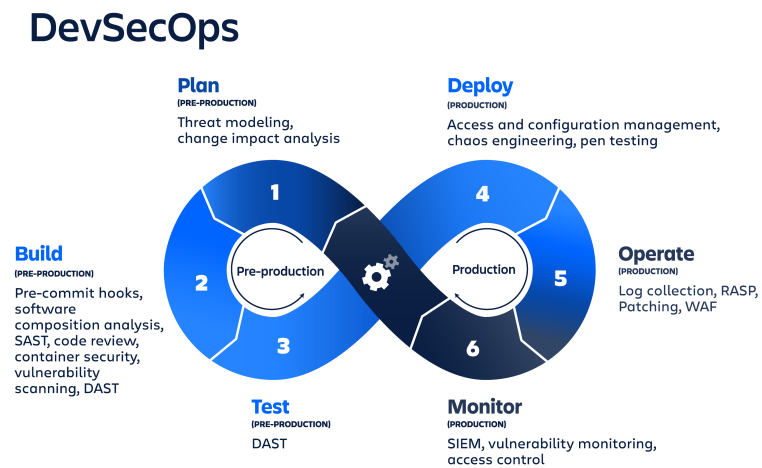


Figura B.1: Diagrama de Devsecops

Una fase de planificación (posterior en primer lugar por una investigación y un estudio de viabilidad), le sigue una fase de desarrollo, testing, despligue y finalmente de operaciones y monitoreo (estas dos últimas, clave para asegurar el bienestar del cliente)

