Grado en Ingeniería Informática

Trabajo de Final de Grado

Aplicación de procesamiento y análisis de medidas

Autor:
Daniel Fernández Martínez

Supervisor:
José Luis Martínez Pérez
Tutor académico:
Vicente Cholvi Juan

Fecha de lectura: 14 de julio de 2020
Curso académico 2019/2020

Resumen

En este documento se describe el proyecto realizado como Trabajo de Final de Grado durante la estancia en prácticas en la empresa IoTsens, el cual consiste en desarrollar una serie de módulos que amplíen la funcionalidad de la plataforma ya existente en la empresa, la cual permite obtener información de distintos sensores y realizar acciones sobre ellos, entre otras funciones.

En concreto, se han creado dos módulos, uno encargado de procesar y almacenar las medidas de los sensores pertenecientes a diversas organizaciones, y otro encargado de utilizar dichas medidas para mostrar cálculos relacionados con los costes y gastos asociados a mantener estos sensores.

Para realizar la planificación y el desarrollo del proyecto se ha utilizado una metodología ágil, en concreto Scrum. Además de esto, en este documento se describen las fases de análisis, diseño, implementación y pruebas del proyecto. Por último, se muestran brevemente las conclusiones tras la realización de este Trabajo de Final de Grado y un anexo que describe una prueba de rendimiento realizada sobre una de las bases de datos de la empresa.

Palabras clave

Internet de las cosas, medidas, análisis, Java, Angular, monitorización.

Keywords

Internet of things, measures, analysis, Java, Angular, monitoring.

Índice general

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

1.1.1. Sobre el Grupo Gimeno

El Grupo Gimeno tiene más de 145 años de historia desde su fundación, y actualmente cuenta con empresas que cubren múltiples áreas de actividad. Su objetivo principal es proveer servicios ciudadanos a nivel global. En la figura 1.1 se pueden ver todas las áreas de actividad del grupo [1].

La empresa en la que se ha llevado a cabo el proyecto (IoTsens) pertenece actualmente al Grupo Gimeno. Esta empresa se ubica en el área de tecnología, junto a GiDITEK y UVAX.

1.1.2. Sobre IoTsens

IoTsens es una empresa proveedora de soluciones verticales del Internet de las Cosas. Se dedica a recolectar y almacenar medidas de sensores de distintos tipos, con el objetivo de procesar dicha información y ofrecer a sus clientes una plataforma mediante la cual poder monitorizar toda la actividad que se desee.

Además de esta plataforma, IoTsens ofrece más servicios y productos, como por ejemplo dispositivos y sensores orientados a distintos mercados o soluciones de tele lectura de contadores inteligentes, entre otros.

Inicialmente, esta empresa trabajaba para otras pertenecientes al Grupo Gimeno, y fue en 2014 cuando salió al mercado a nivel nacional e internacional tras completar varios proyectos exitosos dentro del grupo. En la actualidad, IoTsens continúa expandiéndose y trabajando con distintas empresas, ayuntamientos y, en general, cualquier entidad que precise de los productos y servicios que ofrecen.

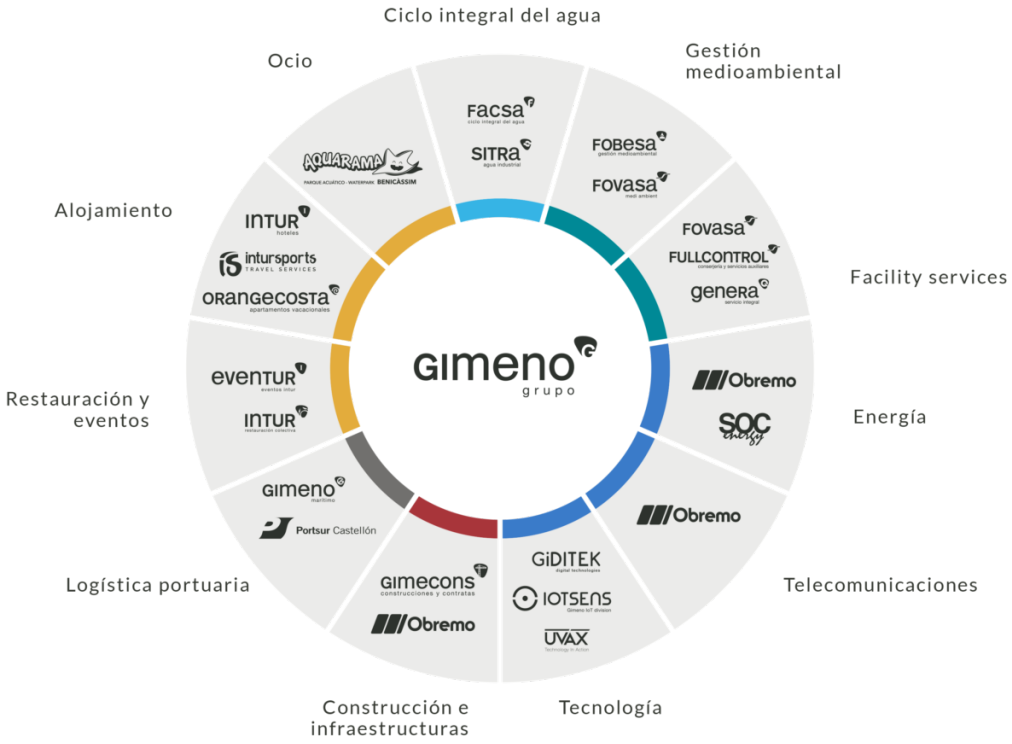

Figura 1.1: Áreas de actividad del Grupo Gimeno

En IoTsens se han desarrollado múltiples proyectos y soluciones que se pueden clasificar en las siguientes áreas de actividad:

- **Ciudades inteligentes (Smart city):** proporciona control de sensores y dispositivos de forma remota en las ciudades, y ofrece soluciones para todo lo relacionado con el procesamiento y análisis de los datos recogidos por los sensores.
- **Gestión del agua (Smart water):** ofrece a las empresas y a sus clientes soluciones de tele medida de contadores inteligentes.
- **Industria:** ofrece el control de plantas industriales a distancia, y la recogida y análisis de la información que se desee mediante sensores.
- **Personalizados:** otros proyectos IoT ajustados a las necesidades de clientes concretos y que no caigan en ninguna de las anteriores categorías, .

1.1.3. Sobre el proyecto

El proyecto realizado es una expansión de la plataforma horizontal mencionada en el apartado anterior. Esta expansión permite a cada cliente comprobar la cantidad de medidas que sus sensores han realizado en un periodo de tiempo determinado. Tanto los datos de las medidas realizadas como el precio de éstas se muestran de forma que sean fáciles de leer y analizar, mediante el uso de gráficas y desgloses de precios, con formato de informe.

Para facilitar su desarrollo y desacoplar funcionalidad, el proyecto se ha dividido en dos módulos: uno encargado de procesar las medidas entrantes de los sensores y almacenarlas en una base de datos, y otro encargado de realizar cálculos sobre las medidas (para obtener precios y costes, para mostrar las medidas clasificadas según distintos criterios, etc.) y mostrarlos al usuario.

En el primer módulo se ha priorizado la eficiencia, ya que hay un volumen muy elevado de datos entrantes, mientras que en el segundo se ha realizado un desarrollo teniendo más en cuenta la experiencia del usuario final.

1.2. Objetivos y alcance

El objetivo principal de este proyecto es desarrollar e incorporar ambos módulos *software* como un componente adicional a usar en la plataforma de IoTsens, para ampliar la funcionalidad de la misma. A parte de esto, el objetivo principal se puede desglosar en los siguientes subobjetivos, los cuales establecen el alcance del proyecto:

- Procesar las medidas recibidas desde los sensores para que sean utilizables por cualquier servicio que las necesite.
- Realizar cálculos del consumo de cada organización teniendo en cuenta distintas variables que afecten al precio del servicio.
- Ofrecer al usuario final los datos sobre el consumo de forma clara, visual y personalizable.
- En general, de cara al desarrollo de ambos módulos, crear componentes extensibles y reusables por otros módulos de la empresa.

Además de esto, cabe mencionar que el alcance del proyecto tan solo incluye el desarrollo de los componentes *software*. Esto quiere decir que las bases de datos y el resto de componentes *hardware* necesarios han sido proporcionados por la empresa. La configuración mínima del servidor de la base de datos (crear el servidor e instalar lo básico para poder crear bases de datos de Timescale) también ha sido realizada por un empleado de la empresa.

Finalmente, el *pipeline* a través de la cuál se mueven las medidas de los sensores ya existe en la empresa, y ha sido usada en el proyecto para recibir y procesar dichas medidas.

Por otra parte, el objetivo del producto es simplemente mejorar la experiencia del usuario final de la plataforma, el cuál dispondrá de opciones adicionales para gestionar sus consumos. El alcance del producto se puede desglosar en funcional, organizativo e informático.

Funcional

El alcance funcional dependerá el módulo del que se hable. Por una parte, está el módulo que procesa las medidas de los sensores y las guarda en la base de datos, y por otra está el

módulo que realiza cálculos sobre los datos guardados, y los muestra al usuario en forma de informes y gráficas.

El primer módulo no tiene funcionalidad accesible por el usuario final. Tan solo se encargará de recibir y procesar las medidas. La funcionalidad del segundo módulo se puede clasificar según el usuario. La aplicación tendrá dos tipos de usuario, el administrador y el resto de usuarios comunes:

- **Administrador:** este usuario será capaz de acceder a los datos de cualquier organización, y dispondrá de una interfaz adicional que le permitirá elegir la organización de la cual quiere obtener datos. Los datos de los que dispone se mencionan a continuación.
- **Resto de usuarios:** estos usuarios tan solo podrán acceder a los datos de su propia organización. Entre los datos que se quieren mostrar, tenemos información de las medidas tomadas por cada sensor que pertenezca a la organización, y el coste de dichas medidas. Además, estos usuarios también podrán ver las medidas individuales (fecha en la que se tomaron, tipo de dato que recoge el sensor, valor recogido, id del sensor, etc.). Finalmente, podrán ver gráficas mostrando las medidas clasificadas por categoría del sensor y un desglose de costes. La categoría de un sensor indica la función que cumple en una organización dada (por ejemplo, sensor de nivel de agua, calidad del aire, detección de luz, etc.).

Organizativo

Esta nueva aplicación afectará principalmente al departamento administrativo en la empresa, ya que el segundo módulo (el que contiene la aplicación web) facilitará la monitorización de las medidas mediante el modo administrador que se menciona en el apartado anterior. A parte de esto, no hay más departamentos que se beneficien especialmente del producto final.

Informático

Finalmente, en lo que a aspectos técnicos se refiere, el desarrollo del proyecto permitirá:

- **Comprobar como interactúan nuevos módulos con la plataforma:** se comprobará, de forma indirecta, cómo de fácil es añadir nuevos módulos a la plataforma ya existente en la empresa, y cómo se comportan dichos módulos con los ya existentes.
- **Comprobar la eficiencia de la base de datos elegida:** en concreto, esta es TimescaleDB, la cual nunca ha sido usada en la empresa. Principalmente, se observará la eficiencia de las inserciones de datos en masa realizadas por el primer módulo. Esto es relevante dado el volumen de datos que trata la empresa a diario (se reciben aproximadamente mil millones de medidas al día). De funcionar correctamente y de forma eficiente, la empresa puede usar esto como una primera prueba para plantearse migrar su base de datos actual a TimescaleDB.

1.3. Descripción del proyecto

1.3.1. Tecnologías usadas

Ambos módulos tienen una base de Java y utilizan el framework Spring [2], el cual se usa comúnmente dentro de la empresa para la creación de los distintos microservicios. Además de esto, hay tecnologías específicas de cada módulo. Las más relevantes son:

- **TimescaleDB:** se usa en el primer módulo. Esta base de datos, la cual está creada con PostgreSQL como base, permite agilizar las inserciones de grandes cantidades de datos y facilitar las peticiones al clasificar sus filas en segmentos según su marca de tiempo, entre otros [3]. Esto es ideal para el proyecto, dada la naturaleza de los datos a tratar.

- **MariaDB:** inicialmente, este sistema de gestión de bases de datos derivado de MySQL se iba a utilizar en el segundo módulo para almacenar todos los cálculos relacionados con las medidas recibidas por organización y su coste. Para facilitar el acceso a los datos se optó por utilizar solamente la base de datos de Timescale al comenzar el segundo módulo, pero la empresa sigue considerando la opción de usar MariaDB y, en el momento de la creación de esta memoria, siguen realizando comparativas entre varios sistemas de gestión de bases de datos para tomar una decisión.

- **Angular:** se ha utilizado la versión 8 de este framework para realizar las vistas del segundo módulo. Está desarrollado en TypeScript y se utiliza para crear aplicaciones web SPA (de una sola página) [4].

A parte de esto, se ha usado MQTT (MQ Telemetry Transport) como protocolo para el envío y recepción de mensajes, IntelliJ IDEA como entorno de desarrollo y Git/GitHub como software de control de versiones.

1.3.2. Contexto adicional

El proyecto se ha llevado a cabo en un entorno cambiante, ya que varias plataformas de la empresa estaban siendo actualizadas y rediseñadas en el momento del desarrollo. Esto ha llevado a algunos cambios menores de la dirección del proyecto, los cuales se comentan en el apartado del seguimiento del proyecto.

A parte de esto, se espera que el proyecto mejore la experiencia del usuario final que usa la plataforma principal de IoTsens, ofreciendo funcionalidad adicional y más control y detalle sobre sus gastos.

1.4. Estructura de la memoria

Tras esta introducción, el resto de la memoria describe las distintas etapas del desarrollo del software asociadas a este proyecto. A continuación, se explican los contenidos de cada capítulo:

- Capítulo 2, planificación del proyecto: explica todos los aspectos relacionados con la planificación y el seguimiento del proyecto. Además, comenta la metodología elegida y los motivos de su selección.
- Capítulo 3, análisis del sistema: muestra la definición y el análisis de los requisitos del sistema.
- Capítulo 4, diseño del sistema: muestra el diseño de ambos módulos del sistema, así como el diseño de interfaces de usuario.
- Capítulo 5, implementación y pruebas: documenta distintos detalles sobre la implementación en sí, y las pruebas realizadas para comprobar el correcto funcionamiento del producto final.
- Capítulo 6, conclusiones: muestra algunas de las conclusiones a las que se llegaron tras la realización de este proyecto.

Capítulo 2

Planificación del proyecto

2.1. Metodología

2.1.1. Scrum

En la empresa en la que se ha desarrollado el proyecto se utilizan metodologías ágiles para el desarrollo de los distintos proyectos. En concreto, se usa Scrum [5], pero con algunas diferencias con respecto a los procesos y roles originales. En este apartado se explican las variaciones que suelen usar:

- **Esprints:** pese a aplicar el resto de procesos de la metodología Scrum, en ocasiones esta empresa no trabaja por esprints. Esto quiere decir que, según el tipo del proyecto, puede que la empresa plantee una pila del producto inicial y trabaje durante todo el desarrollo como si tan solo fuese un único esprint. A parte de esto, realizan todas las reuniones diarias y de análisis que se dan durante un esprint normal de Scrum.

- **Kanban:** como en el punto anterior, este sistema solo es usado en determinados proyectos. Además de esto, todos los proyectos activos que lo usen se plasman en el mismo tablero Kanban, y se añaden filtros para que cada empleado pueda seleccionar su nombre y ver las tareas que tiene asignadas para cada proyecto.

- **Pila del producto:** se da otra variación en la forma de entender la pila del producto. En la empresa, prefieren crear una pila del producto basada en tareas a realizar. Tras esto, estas tareas se dividen en subtareas al comenzar un esprint, tal y como se hace en la metodología Scrum original. Esta forma de entender la pila del producto se ha adoptado en el proyecto a desarrollar, por los motivos que se explican en la sección de la pila del producto.

2.1.2. Adaptación al proyecto

Para la realización de este proyecto se ha usado una adaptación de la metodología Scrum a un proyecto individual. Además de Scrum, se usará Kanban para organizar las distintas tareas a realizar (mediante la herramienta Jira). Los cambios realizados sobre el método Scrum original son:

- **Scrum diario:** estas reuniones han sido ligeramente modificadas, ya que el equipo tan solo contaba con un miembro. Los primeros 15 minutos de cada día se han dedicado a realizar una revisión de las tareas completadas y una evaluación del progreso en el proyecto. Además, se ha realizado una revisión semanal adicional, junto al supervisor, con la duración de un Scrum diario normal, para asegurar el correcto desarrollo del mismo.
- **Revisión, retrospectiva y planificación del esprint:** estas reuniones se han realizado junto al supervisor asignado en IoTsens. En dichas reuniones se trataban los temas que habitualmente se tratarían en el Scrum original, pero dada la longitud del proyecto, éstas serán más cortas (todas ellas de una duración aproximada de 30 minutos).

Finalmente, los esprints han tenido una duración de 3 semanas, lo cual implica que en las 12 semanas de la estancia en prácticas han tenido lugar 4 esprints.

2.2. Estimación de recursos y costes

En los siguientes apartados se comentan los distintos recursos que se han utilizado para el desarrollo del proyecto, y una aproximación de los costes totales teniendo en cuenta las estimaciones realizadas.

2.2.1. Recursos humanos

En este caso, el equipo cuenta con tan solo un desarrollador y un supervisor. Teniendo en cuenta la naturaleza del proyecto y la experiencia de los integrantes del equipo, podemos asumir que el desarrollador es un programador *junior* y el supervisor actúa como jefe del proyecto.

Además de esto, un desarrollador *senior* se dedicó a preparar el servidor de la base de datos. Esto también se tendrá en cuenta en el cálculo de costes totales. Pese a que este desarrollador tan solo haya trabajado unas 4h totales en el proyecto, su trabajo ha sido esencial para el desarrollo del mismo.

2.2.2. Otros recursos

Además de los recursos humanos, se han de tener en cuenta los elementos *software* y *hardware* que se han utilizado en el proyecto.

Por un lado, está el *software* usado. Todo el *software* utilizado ha sido gratuito y de código abierto, por lo que no ha sido necesario adquirir nuevas licencias. Este se menciona en el apartado 1.3.1 (tecnologías usadas). Por otro lado, está el *hardware* utilizado, un ordenador para el desarrollo del proyecto y un servidor para almacenar la base de datos:

- **Ordenador:** este fue proporcionado por la empresa, y se usó a lo largo de todo el desarrollo.
- **Servidor:** no fue necesario adquirir un nuevo servidor, ya que la empresa contaba con uno de antemano, el cual simplemente fue reutilizado para el propósito del proyecto.

2.2.3. Costes totales

El principal coste del proyecto, teniendo en cuenta lo anteriormente mencionado, es una suma de los sueldos del programador *junior* principal encargado del desarrollo, el jefe del proyecto y el programador *senior* encargado de preparar la base de datos. En el caso del programador *senior*, tan solo se menciona el coste total del trabajo que hizo según las horas que trabajó, y por ello no se le asignan unas horas diarias o un sueldo mensual. En la tabla 2.1 se detallan estos sueldos.

Rol	Horas diarias	Sueldo mensual (€)	Coste total (€)
Programador *junior*	5	1700	5100
Jefe del proyecto	0,5	170	510
Programador *senior*	-	-	50

Cuadro 2.1: Sueldos y costes totales de los recursos humanos

Todos los salarios se han obtenido en base a los datos más recientes proporcionados por el Instituto Nacional de Estadística (los sueldos mensuales han sido ajustados teniendo en cuenta las horas trabajadas diarias). Exceptuando las del desarrollador principal, las horas se han calculado de forma aproximada en función del tiempo invertido en aportaciones al proyecto.

A parte de esto, se han de tener en cuenta los costes del *software* y *hardware* utilizado. El *software* no supone un coste adicional, ya que era gratuito durante el desarrollo del proyecto. Por otra parte, los costes del *hardware* se pueden ver desglosados en la tabla 2.2. La empresa ya contaba con el *hardware* que se muestra en esta tabla de antemano, pero se menciona de todas formas al haber sido usado en el desarrollo.

Componente	Precio aproximado (€)
Ordenador (HP PRODESK 600 G3 SFF) + monitor	800
Servidor	1200

Cuadro 2.2: Costes de los componentes hardware

Tras esto, podemos calcular los costes totales, los cuales se pueden ver en la tabla 2.3. Además, se ha de tener en cuenta que, para calcular el coste de los empleados, se han añadido

los costes de contratación (un 20 % adicional) y los costes indirectos (otro 20 % adicional). Ambos porcentajes son aproximaciones.

Fuente	Coste (€)
Software	0
Hardware	2000
Empleados	8150
TOTAL	**10150**

Cuadro 2.3: Costes totales de los recursos usados en el desarrollo

2.3. Identificación y gestión de riesgos

Dado el reducido tamaño del equipo de desarrollo, muchos de los riesgos más comunes en proyectos de *software* no se aplican a este caso concreto, especialmente todos aquellos relacionados con conflictos entre miembros del equipo o conflictos en la dirección del proyecto.

De esta forma, en la tabla 2.4 se listan los riesgos según su tipo y ámbito, en la tabla 2.5 se analizan, y en la tabla 2.6 se detallan los planes de prevención y contingencia para cada uno de los riesgos. Todos los riesgos son riesgos del proyecto y no del producto, ya que será la empresa quien continúe el desarrollo con las pruebas y la gestión de calidad del programa creado.

A lo largo del proyecto, no ha sido necesario aplicar ningún plan de corrección o continencia, ya que se han realizado los planes de prevención para todos ellos. Tan solo se ha dado una forma muy leve del riesgo R03, la cual se ha solucionado en unas 10 horas en total (2 días de trabajo) incluyendo la adición de la nueva tarea y la realización de sus tareas, y no ha supuesto retrasos en el resto de esprints. Además de esto, no han surgido riesgos que no se hubiesen previsto inicialmente.

ID	Descripción	Tipo
R01	Falta de conocimiento de las tecnologías a usar	General
R02	Retrasos en los esprints	General
R03	Requisitos incompletos	General
R04	Cambios en la plataforma durante el desarrollo del proyecto	Específico
R05	Primer módulo no procesando los mensajes lo suficientemente rápido	Específico
R06	Base de datos no respondiendo lo suficientemente rápido a las peticiones	Específico

Cuadro 2.4: Lista de riesgos

ID	Análisis del riesgo

R01	▪ Magnitud: variable según el tiempo restante para el desarrollo. A menor tiempo restante, mayor magnitud. ▪ Descripción: el desarrollador principal está trabajando con tecnologías que no ha usado anteriormente. ▪ Impacto: puede causar retrasos importantes en todo el desarrollo si no se soluciona rápidamente. ▪ Indicadores: el desarrollador trabaja a un ritmo muy lento.
R02	▪ Magnitud: variable según el tiempo restante para el desarrollo. A menor tiempo restante, mayor magnitud. ▪ Descripción: los esprints se terminan con tareas pendientes. ▪ Impacto: puede causar retrasos importantes en todo el desarrollo si no se soluciona rápidamente. ▪ Indicadores: el desarrollador trabaja a un ritmo muy lento, o la cantidad de tareas a realizar en un esprint es excesiva.
R03	▪ Magnitud: baja durante el primer esprint, alta en el resto del proyecto. ▪ Descripción: surgen nuevas necesidades durante el desarrollo que no se habían tenido en cuenta en las fases iniciales del proyecto. ▪ Impacto: puede causar retrasos en el desarrollo e insatisfacción por parte del cliente si no se soluciona rápidamente. ▪ Indicadores: ninguno.

R04	▪ Magnitud: variable según el tamaño de los cambios. A cambios más significativos, mayor magnitud. ▪ Descripción: la plataforma en la cual se van a integrar los módulos en desarrollo se actualiza a un ritmo demasiado alto. ▪ Impacto: puede causar que el módulo en desarrollo trabaje con versiones obsoletas de la plataforma si no se soluciona a tiempo. ▪ Indicadores: la versión más actualizada de la plataforma tiene funcionalidad esencial que la versión usada para el desarrollo no tiene.
R05	▪ Magnitud: alta, ya que la eficiencia es un requisito esencial, especialmente en el primer módulo. ▪ Descripción: el primer módulo no es capaz de procesar y almacenar los mensajes entrantes sin pérdidas. ▪ Impacto: puede llegar a inutilizar el primer módulo por completo en casos más graves, o requerir un gasto adicional de tiempo en optimización en casos más leves, retrasando así el ▪ resto del desarrollo. Indicadores: el tiempo de procesado de mensajes es más lento de lo esperado.
R06	▪ Magnitud: baja, ya que en la empresa pueden añadir más potencia a la base de datos en cualquier momento de ser necesario. ▪ Descripción: las peticiones a la base de datos son demasiado lentas. ▪ Impacto: bajo, ya que se puede solucionar de forma relativamente sencilla. ▪ Indicadores: el tiempo de respuesta de la base de datos es mayor a lo esperado.

Cuadro 2.5: Análisis de riesgos

ID	Plan de Prevención/Acción	Plan de Corrección/Contingencia
R01	Formar al programador antes de que comience el desarrollo en caso de que vaya a usar tecnologías desconocidas para él.	Pausar el desarrollo para formar al programador durante el tiempo que sea necesario.
R02	Realizar estimaciones de tiempo adecuadas en las fases iniciales del proyecto.	Trasladar tareas del esprint actual al siguiente o, en casos más graves, eliminar tareas que no se puedan completar por falta de tiempo.
R03	Realizar una pila del producto adecuada en las fases iniciales del proyecto.	Añadir nuevas tareas a la pila del producto para satisfacer los nuevos requisitos.
R04	Acordar, en las fases iniciales del proyecto, la versión de la plataforma sobre la que se desarrollará el proyecto.	Añadir la funcionalidad necesaria esencial a la versión de la plataforma que se esté usando para el desarrollo.
R05	Acordar cuál es la velocidad de procesamiento de mensajes objetivo al iniciar el proyecto y formar al desarrollador en técnicas de optimización.	Invertir tiempo adicional del esprint en formar al desarrollador en técnicas de optimización.
R06	Asignar los recursos necesarios a la base de datos para que soporte la velocidad de inserciones y peticiones de datos.	Solicitar a la empresa la asignación de recursos adicionales a la base de datos.

Cuadro 2.6: Planes de prevención y contingencia para los riesgos

2.4. Planificación temporal

2.4.1. Pila del producto

En este apartado se detalla la versión inicial de la pila del producto, la cual ha sido modificada ligeramente a lo largo del proyecto cuando ha sido necesario (los cambios se comentan en el apartado del seguimiento del proyecto). Las tareas de esta pila se han desarrollado a lo largo de los **4 esprints** del proyecto (12 semanas).

Se ha realizado una pila del producto basada en tareas, al ser este el método preferido en la empresa, además de porque la interacción con el usuario es una parte muy pequeña del proyecto, por lo que una pila basada en historias de usuario no tendría sentido. Estas tareas se dividen en subtareas en cada esprint, tal y como se haría en una aplicación normal de la metodología Scrum. Además, se han asignado puntos de esfuerzo (equivalentes a los puntos de historia en una pila del producto basada en historias de usuario) a cada requisito para determinar el esfuerzo que requiere su realización, al ser este el método preferido cuando se aplica una metodología ágil.

El desarrollador y el jefe del proyecto han trabajado juntos para determinar los puntos de esfuerzo de cada tarea, utilizando un método conocido como Planning Poker [6]. Este método

es usado habitualmente para determinar el esfuerzo que requiere cada elemento de la pila del producto cuando se utiliza Scrum. Todas las tareas tienen una prioridad alta, ya que todas son estrictamente necesarias para el correcto funcionamiento de la aplicación final.

A continuación se muestran las distintas tareas de la pila del producto inicial:

T01 - Crear primer módulo (PE: 1)

Tareas a realizar: crear el proyecto Maven adecuado, el repositorio en GitHub y el esqueleto del proyecto.

Definition of Done:

- Una versión inicial del proyecto se encuentra en el repositorio de la empresa.

T02 - Implementar conector MQTT (PE: 5)

Tareas a realizar: crear la conexión con el microservicio de la empresa que envía los mensajes con las medidas, crear las clases del modelo adecuadas y convertir el mensaje recibido en instancias de dichas clases.

Definition of Done:

- Se reciben los mensajes correctamente a través del microservicio adecuado.
- Los mensajes son interpretados correctamente como clases del modelo.

T03 - Diseñar la base de datos de medidas (PE: 5)

Tareas a realizar: realizar y validar el diseño de la base de datos.

Definition of Done:

- El diseño está terminado y no da lugar a ambigüedades.
- El diseño se adapta a lo esperado según las necesidades de los datos a tratar.

T04 - Implementar la extracción de organización por mensaje (PE: 3)

Tareas a realizar: crear la conexión con el microservicio de la empresa que proporciona el nombre de la organización de cada sensor.

Definition of Done:

- Dada una medida, se puede extraer el identificador del sensor que la ha tomado.

- Se puede obtener la organización de un sensor mediante el microservicio adecuado.

T05 - Crear conector a la base de datos de medidas (PE: 5)

Tareas a realizar: crear una conexión a la base de datos de medidas, y crear las clases del modelo que permiten realizar la traducción de datos.

Definition of Done:

- Se pueden leer y escribir medidas en la base de datos desde el proyecto en Java.

T06 - Crear segundo módulo (PE: 1)

Tareas a realizar: crear el proyecto Maven adecuado, el repositorio en GitHub y el esqueleto del proyecto.

Definition of Done:

- Una versión inicial del proyecto se encuentra en el repositorio de la empresa.

T07 - Diseñar la base de datos de consumos (PE: 13)

Tareas a realizar: realizar y validar el diseño de la base de datos.

Definition of Done:

- El diseño está terminado y no da lugar a ambigüedades.
- El diseño se adapta a lo esperado según las necesidades de los datos a tratar.

T08 - Crear vistas para consultar los consumos por organización (PE: 13)

Tareas a realizar: crear las vistas necesarias para que cada organización pueda consultar su consumo, y crear las clases del modelo necesarias para realizar el paso de datos.

Definition of Done:

- Las vistas están creadas y cumplen los estándares de la empresa.
- Los datos se muestran de forma adecuada.

T09 - Crear vistas para consultar los consumos por sensor (PE: 8)

Tareas a realizar: crear las vistas necesarias para que cada organización pueda obtener un desglose de su consumo según el tipo de sensor.

Definition of Done:

- Las vistas están creadas y cumplen los estándares de la empresa.
- Los datos se muestran de forma adecuada.

T10 - Crear conector a la base de datos de consumos (PE: 5)

Tareas a realizar: crear una conexión a la base de datos de consumos, y crear las clases del modelo que permiten realizar la traducción de datos.

Definition of Done:

- Se pueden leer y escribir medidas en la base de datos desde el proyecto en Java.

T11 - Implementar consultas a la base de datos de medidas (PE: 3)

Tareas a realizar: definir las consultas a la base de datos de medidas, según las necesidades de las vistas, e implementarlas en Java.

Definition of Done:

- Todas las vistas pueden solicitar los datos que necesiten en el formato adecuado.

T12 - Crear gráficas comparativas para consultar los consumos (PE: 20)

Tareas a realizar: añadir gráficas a las vistas que permitan a los usuarios interpretar los datos de forma más cómoda.

Definition of Done:

- Las gráficas están creadas, son accesibles por el usuario y muestran la información correcta.

2.4.2. Primer esprint

En este apartado se muestran las tareas y subtareas contenidas en el primer esprint. El primer esprint consta de las siguientes tareas de la pila del producto:

- T01 - Crear primer módulo
- T02 - Implementar conector MQTT
- T03 - Diseñar la base de datos de medidas

- T04 - Implementar la extracción de organización por mensaje
- T05 - Crear conector a la base de datos de medidas

Inicialmente, se esperaba que las subtareas a realizar tuviesen la siguiente duración aproximadamente:

Tarea	**Duración (h)**
Crear el proyecto Maven adecuado.	2
Crear el repositorio en GitHub y el esqueleto del proyecto.	3
Crear la conexión con el microservicio de la empresa que envía los mensajes con las medidas.	15
Crear las clases del modelo adecuadas y convertir el mensaje recibido en instancias de dichas clases.	5
Realizar y validar el diseño de la base de datos.	20
Crear la conexión con el microservicio de la empresa que proporciona el nombre de la organización de cada sensor.	10
Crear una conexión a la base de datos de medidas.	10
Crear las clases del modelo que permiten realizar la traducción de datos.	10

2.5. Seguimiento del proyecto

2.5.1. Esprints

En esta sección se describe el progreso a lo largo de los distintos esprints, así como las tareas que fueron asignadas durante la planificación de cada esprint y el desglose por subtareas. Finalmente, también se indicarán los problemas que surgieron, de haberlos.

Esprint 1

En este primer esprint se omiten las tareas y subtareas, ya que se mencionan en la sección anterior.

Este primer esprint fue el más complicado, ya que varios días se tuvieron que dedicar a la formación del desarrollador principal en varios sentidos. Por otra parte, la empresa proporcionó las herramientas adecuadas a tiempo, lo cual facilitó este proceso y permitió que el esprint se completase a tiempo.

En concreto, la formación fue de las tecnologías a usar, y de la metodología de trabajo usada en la empresa, así como estándares de programación generales usados en el desarrollo de los distintos proyectos de IoTsens. La formación en tecnologías fue principalmente independiente

por parte del desarrollador, mientras que el resto se dio con ayuda del supervisor asignado o jefe del proyecto.

Además de la formación, en este esprint se terminó el primer módulo, el encargado de recibir, procesar y almacenar en la base de datos las medidas entrantes de los sensores. La eficiencia, algo que se tenía que tener muy en cuenta en este módulo, no fue un obstáculo, ya que el procesamiento necesario de las medidas resultó ser más sencillo de lo esperado. El único cuello de botella que quedó fue la velocidad a la que la base de datos guardaba los mensajes, lo cual indicó que el módulo se había desarrollado satisfactoriamente.

En general no surgieron problemas durante este esprint, por lo que se pudo completar a tiempo y sin tener que actualizar la pila del producto.

Esprint 2

Tareas del esprint:

- T06 - Crear segundo módulo
- T08 - Crear vistas para consultar los consumos por organización
- T11 - Implementar consultas a la base de datos de medidas
- **T12 - Crear gráficas comparativas para consultar los consumos**

Subtareas del esprint:

Crear el segundo módulo y configurar Maven.
Crear las peticiones al backend necesarias.
Crear la comunicación entre backend y frontend.
Crear gráficas comparativas por categoría del sensor.
Crear gráficas comparativas por día del mes.
Añadir tablas necesarias para mostrar las medidas por sensor.

La formación volvió a ocupar varios días en este esprint, ya que en la empresa se usaban tecnologías muy concretas para el desarrollo de las interfaces de usuario. En concreto, el desarrollador se tuvo que formar en Angular, además de aprender a programar en TypeScript y a utilizar bibliotecas como PrimeNG y Highcharts, entre otros.

Las tareas asignadas al esprint se pudieron completar sin problemas, pero, a causa de un malentendido durante la planificación del proyecto, se tuvo que añadir una nueva tarea a la pila del producto, la del desarrollo de la vista de administrador. Este cambio se comenta en el siguiente apartado, pila del producto final.

Finalmente, como sobró tiempo tras terminar las tareas del esprint, la tarea T12 se añadió al esprint actual y se completó.

Esprint 3

Tareas del esprint:

- T07 - Diseñar la base de datos de consumos
- T09 - Crear vistas para consultar los consumos por sensor
- T10 - Crear conector a la base de datos de consumos
- T13 - Crear las vistas del administrador

Subtareas del esprint:

Decidir los datos a guardar en la base de datos de consumos
Terminar de diseñar la base de datos de consumos
Crear las tablas necesarias
Crear vistas para consultar las medidas por sensor
Añadir información sobre los costes de las medidas por sensor
Crear vistas para consultar las medidas individuales de cada sensor
Crear vistas de administrador
Securizar vistas de administrador

Tras este esprint, el proyecto quedó prácticamente completado, dejando para el último esprint todo lo relacionado con terminar de pulir el producto final en medida de lo posible. Se asignaron y terminaron todas las tareas restantes junto con sus respectivas subtareas.

A parte de esto, en este esprint se tomó la decisión de incluir en Timescale, la base de datos del primer módulo, las tablas de la base de datos del segundo módulo (o "base de datos de consumos"). De esta forma, el proyecto terminó usando tan solo una base de datos, lo cual facilitó las consultas y agilizó la creación del segundo módulo.

Esprint 4

Tareas del esprint:

- **T14 - Mostrar un desglose más detallado de los costes de cada organización**

Subtareas del esprint:

Definir los datos a mostrar en el desglose de costes
Añadir los datos necesarios al desglose
Terminar las interfaces de usuario
Corregir todos los errores menores encontrados en el desarrollo
Realizar análisis de Timescale

Este esprint se dedicó principalmente a terminar de pulir las interfaces gráficas (utilizar estilos agradables a la vista, homogeneizar la distribución de los componentes, etc.), y a corregir todos los errores menores que tenía la aplicación para facilitar a la empresa el proceso de pruebas.

Además de esto, se hizo un análisis del rendimiento de la base de datos de Timescale, realizando inserciones masivas de datos y comprobando los tiempos de respuesta de las peticiones de datos tras dichas inserciones. Este análisis se detalla en el anexo A. El análisis no se añadió como tarea, ya que no aporta nada al proyecto desarrollado.

Finalmente, se añadió una nueva tarea, la T14, dado el tiempo restante del esprint que sobró tras completar el análisis, las interfaces de usuario y la corrección de errores.

En general, este último esprint y el proyecto en general se pudieron completar sin problemas mayores o desviaciones importantes con respecto a la planificación inicial.

2.5.2. Pila del producto final

Tras el desarrollo del proyecto, el aspecto de la pila del producto es muy similar. Tan solo se han modificado algunos conceptos (como el hecho de que al final se haya usado una sola base de datos, y no dos), pero el trabajo ha sido el mismo. Además, como es obvio tras ver el desglose de los esprints, algunas tareas se habían estimado como más grandes de lo que realmente eran en lo que a coste temporal se refiere, pero no ha habido ningún cambio drástico.

Finalmente, se han añadido dos nuevas tareas a la pila del producto durante el desarrollo, que son las siguientes:

T13 - Crear las vistas del administrador (PE: 5)

Tareas a realizar: crear las vistas necesarias para que el administrador del sistema sea capaz de ver los datos de cada organización disponible.

Definition of Done:

- Las vistas están creadas y cumplen los estándares de la empresa.
- El administrador puede seleccionar entre todas las organizaciones disponibles.

T14 - Mostrar un desglose más detallado de los costes de cada organización (PE: 3)

Tareas a realizar: añadir más detalle a los desgloses de costes ya existentes.

Definition of Done:

- El desglose de costes muestra la información adicional solicitada por el jefe del proyecto.

Capítulo 3

Análisis del sistema

3.1. Definición de requisitos

Dado que las tareas a realizar y los requisitos que estas satisfacen ya están definidos y explicados en la pila del producto, en esta sección se va a tener en cuenta el punto de vista del usuario, indicando por lo tanto los requisitos de la aplicación web. Para ello se han usado historias de usuario, una forma de especificación de requisitos usada comúnmente en metodologías ágiles.

Tal y como se explica en secciones anteriores, no se ha usado este conjunto de historias de usuario como pila del producto porque la interacción con el usuario es una parte muy pequeña del proyecto, por lo que estas historias tan solo se mencionan de forma orientativa y sin entrar en mucho detalle. A lo largo del resto del capítulo se realizará el análisis del resto de componentes que no interactúan con el usuario de forma directa.

Épica 1

- HU01: Como usuario, quiero poder ver la cantidad de medidas realizadas por mis sensores según su categoría para entender fácilmente mis consumos.

- HU02: Como usuario, quiero poder ver la cantidad de medidas realizadas por mis sensores cada día para entender fácilmente mis consumos.

- HU03: Como usuario, quiero poder ver la cantidad de medidas realizadas por cada uno de mis sensores para entender fácilmente mis consumos.

- HU04: Como usuario, quiero poder ver la información almacenada de las medidas individuales tomadas por uno de mis sensores para entender fácilmente mis consumos.

Épica 2

- HU05: Como usuario, quiero saber el coste total de las medidas tomadas por uno de mis sensores en concreto para conocer mis gastos.
- HU06: Como usuario, quiero saber qué tipo de servicio he contratado y su precio para conocer mis gastos.
- HU07: Como usuario, quiero saber la suma del coste total de las medidas de todos mis sensores para conocer mis gastos.

Épica 3

- HU08: Como usuario, quiero poder elegir el rango de fechas usado para realizar los distintos cálculos para tener más control sobre la información que recibo.

3.2. Análisis de requisitos

3.2.1. Componentes del sistema

En este apartado se explican las funciones que cumplen los principales componentes del sistema, fácilmente reconocibles tras definir los requisitos y la pila del producto. Además de esto, se muestran las dos versiones de la arquitectura a alto nivel: la versión inicial con dos bases de datos y la versión final con tan solo una. Cabe mencionar que el objetivo de esta sección no es describir el diseño de los componentes en sí (este diseño se encuentra en el siguiente capítulo), sino analizar los componentes que se sabía de antemano que petenecerían al sistema final.

Por una parte, los elementos más fáciles de reconocer son los dos módulos a desarrollar, cuyas funciones son las siguientes:

- Primer módulo o módulo "medidas": el objetivo principal de este componente es almacenar las medidas recibidas de los distintos sensores que poseen las organizaciones en la base de datos. Además de esto, dichas medidas tendrán que ser almacenadas junto con dos datos adicionales que no vienen en la medida: la organización que posee el sensor y la categoría. Esto quiere decir que este primer módulo no solo se encargará del almacenamiento de dichas medidas, sino también del procesado que sea necesario.
- Segundo módulo o módulo "web": por otro lado, este módulo es más similar a una aplicación web estándar. Su objetivo será leer datos de la base de datos en la que almacena las medidas el primer módulo, y realizar los cálculos necesarios para mostrar dicha información al usuario final. Esto lo hará por medio de una aplicación web. En concreto, este módulo se acoplará a la plataforma ya existente en IoTsens. Esta plataforma es usada por las organizaciones clientes de la empresa para consultar diversos datos sobre sus sensores y consumos.

En conjunto, estos dos módulos pretenden ofrecer funcionalidad extra a los clientes de IoTsens, en concreto para conocer un desglose de sus gastos. Algunos ejemplos de informes y gráficas que se desean mostrar son los siguientes:

- Medidas clasificadas por categoría del sensor: cada sensor está asociado a una categoría, la cual resume la función del mismo (por ejemplo: sensor de nivel de agua, calidad del aire, detección de luz, etc.). Una información valiosa que los clientes podrían necesitar es la cantidad de medidas que sus sensores han realizado, según esta categoría.
- Medidas realizadas por cada sensor: de disponer de este dato, una organización podría identificar rápidamente si un sensor está siendo demasiado caro para el uso que se le está dando, o incluso permitiría detectar si un sensor concreto no está funcionando de forma correcta, y a partir de ahí ponerse en contacto con IoTsens sabiendo de antemano el causante del problema.
- Desglose de costes general: esto sería simplemente un resumen de lo que se está pagando al mes por cada motivo: precio de las medidas, precio base del servicio, ofertas, etc. útil si se quieren conocer los costes de un determinado momento con un solo golpe de vista.

Además de esto, la aplicación depende de algunos módulos externos ya existentes en la empresa. Estos son los siguientes:

- Tubería de medidas: sin entrar en mucho detalle, ya que el funcionamiento de esta tubería es complejo y poco relevante para este proyecto, este servicio permite recibir las medidas tomadas por todos los sensores de las organizaciones clientes de la empresa como mensajes MQTT. En la sección del diseño se explica con más detalle el funcionamiento de este componente.
- Otros servicios: internamente, la empresa cuenta con servicios adicionales, los cuales permiten obtener datos sobre sus clientes, productos, sensores, servicios, etc. En concreto, el primer módulo usará estos servicios para obtener los siguientes datos:
  - Organización que posee el sensor emisor de la medida: esta información no se encuentra almacenada de forma directa en la medida. Para obtener la organización, hace falta realizar una petición por medio de uno de los métodos de un servicio de la empresa, cuya entrada es un identificador que se encuentra en la propia medida y cuya salida es el nombre de la organización en texto plano.
  - Categoría del sensor: de forma muy similar al punto anterior, la categoría de un sensor se obtiene por medio de otro de los métodos de un servicio de la empresa, cuya entrada es un campo que se encuentra en la medida (el identificador del sensor) y cuya salida es la categoría al que pertenece el sensor.

Tras identificar todos estos componentes básicos y sus funciones, se puede pasar a la visualización de la interacción entre ellos. En la figura 3.1 se muestra un esquema realizado en la planificación del proyecto. En él se ve, además de los componentes y sus interacciones, el flujo principal de información, desde que se reciben las medidas hasta que el usuario lee los datos

calculados. Este flujo es meramente orientativo, ya que la comunicación final será bidireccional en la mayoría de casos (por ejemplo, si se tiene en cuenta el envío de la petición y no solo la recepción de la respuesta). Este esquema terminó variando ligeramente en fases posteriores del proyecto.

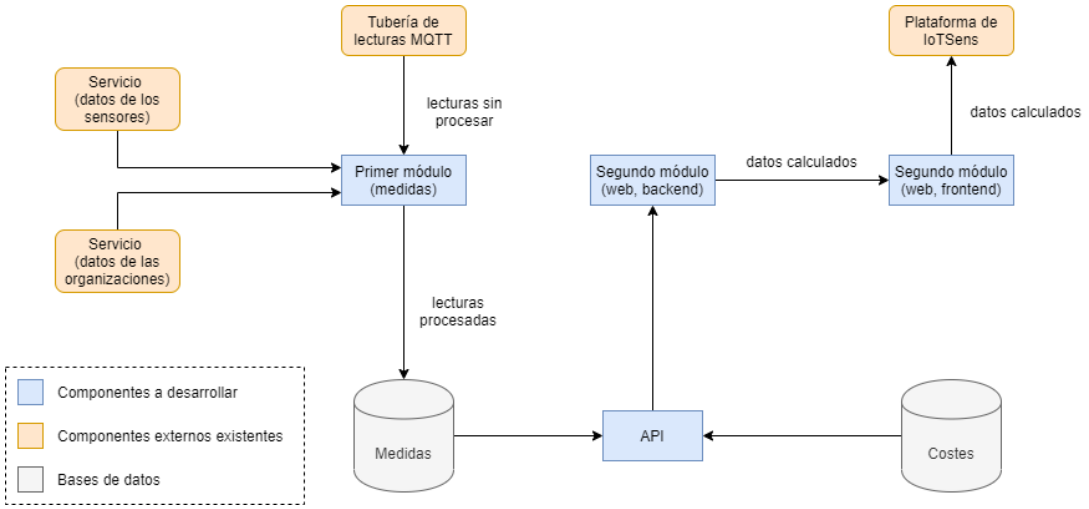

Figura 3.1: Esquema de componentes inicial a alto nivel

Como se comentaba al principio de esta sección y en el apartado de seguimiento del proyecto, durante el desarrollo se tomó la decisión de usar una sola base de datos. Además de esto, durante las primeras fases del proyecto, surgió la idea de comunicar ambos módulos con una API. Esta idea se terminó descartando, ya que resultaba poco eficiente e innecesaria, y no se menciona en otras secciones ya que nunca llegó a verse reflejada en la pila del producto. La versión final de la arquitectura a alto nivel se puede ver en la figura 3.2, tras aplicar estos dos cambios.

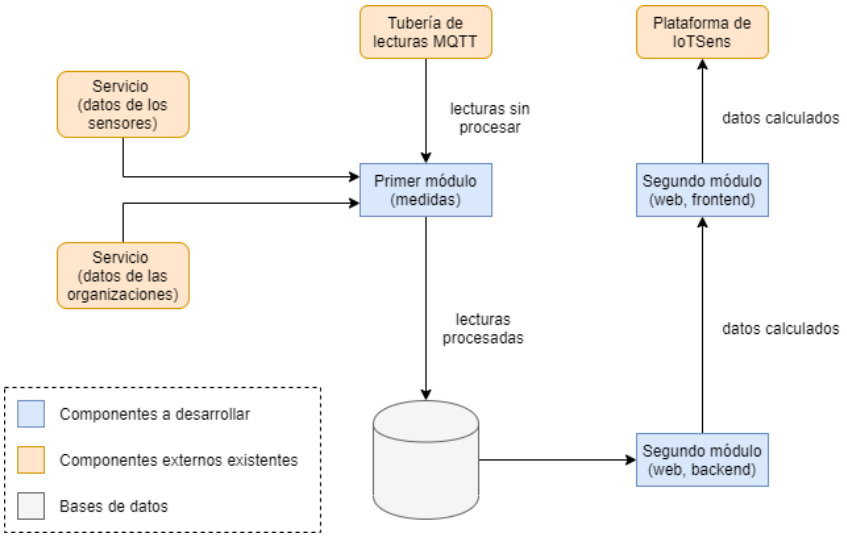

Figura 3.2: Esquema de componentes final a alto nivel

3.2.2. Requisitos de datos

En este apartado se describen los requisitos de datos del sistema, así como quién realizó la propuesta inicial de cómo almacenar dichos datos. Esta información será almacenada y usada por la aplicación final. A continuación, se muestran estos requisitos:

D01: Medida

Fuentes: jefe del proyecto

Datos específicos a mantener: identificador de la medida, identificador del sensor, tipo de dato leído, dato leído, marca de tiempo, identificador de la organización, nombre de la organización, categoría del sensor.

Descripción y detalles: esta es la información que se guarda de cada medida tras ser procesada. Con esta información se realizarán los cálculos necesarios a mostrar en el módulo web.

D02: Organización

Fuentes: desarrollador *junior*

Datos específicos a mantener: identificador de la organización, nombre de la organización, precio por medida, precio base del servicio, plan contratado.

Descripción y detalles: estos son todos los datos necesarios para calcular los costes de los servicios de IoTsens en lo que a medidas se refiere.

D03: Sensor

Fuentes: desarrollador *junior*

Datos específicos a mantener: identificador del sensor, precio por medida, organización que posee el sensor.

Descripción y detalles: estos son los datos necesarios para calcular costes de sensores concretos.

D04: Planes

Fuentes: jefe del proyecto

Datos específicos a mantener: identificador del plan, precio del plan, tipo de cobro.

Descripción y detalles: estos son los planes de los servicios contratables por las organizaciones.

Finalmente, es necesario mencionar el caso del precio por medida, ya que esto puede parecer un error o un caso de datos duplicados, al guardarse este dato asociado tanto a las organizaciones como a los sensores. Esto se debe a que algunos sensores pueden tener un precio por medida fijo concreto, al tratarse de sensores especiales o distintos al resto en algún sentido. De esta forma, se priorizará el precio por medida de cualquier sensor que tenga uno asociado cuando se realicen los cálculos de costes. En caso de no tener ninguno, se usará el precio por medida de la organización.

Capítulo 4

Diseño del sistema

4.1. Diseño de la arquitectura

4.1.1. Primer módulo

En el primer módulo, o módulo "medidas", se pueden distinguir 3 elementos principales: la base de datos, los servicios externos de la empresa y el programa a desarrollar. Estos tres elementos se van a detallar principalmente desde el punto de vista funcional, ya que los datos que tratan son muy simples.

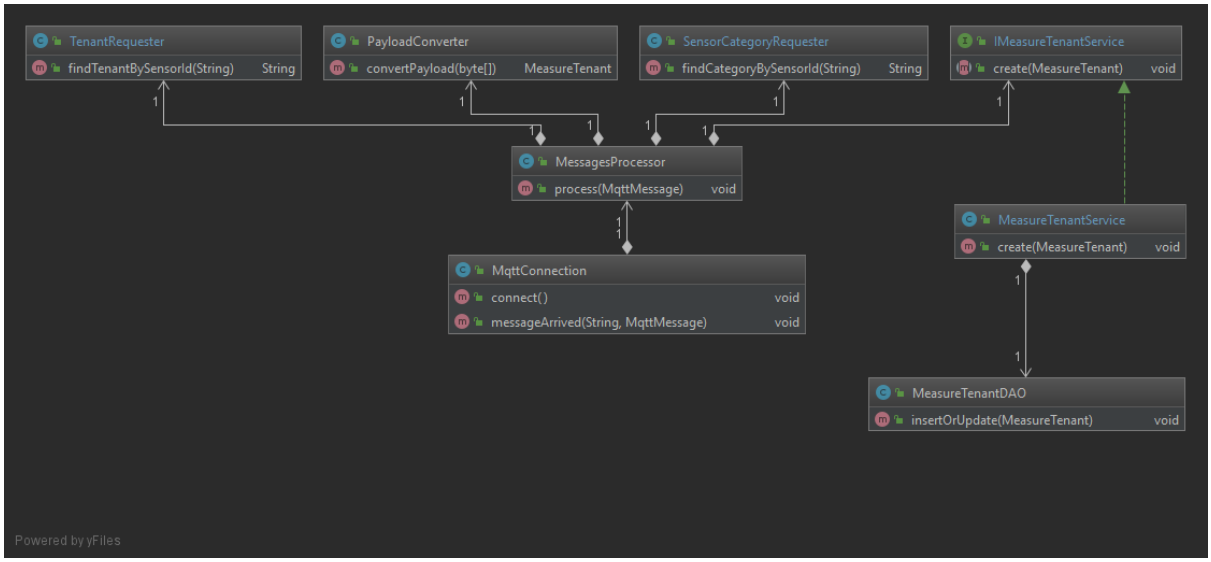

Figura 4.1: Dependencias entre componentes del primer módulo

Como se puede observar en la figura 4.1, no se ha realizado un diagrama de clases para explicar el diseño el primer módulo, ya que los datos a tratar son muy simples (se pueden ver en

la figura 4.2). Por ello, se ha añadido este diagrama de dependencias entre clases. Además, las clases tienen sus métodos más relevantes. Estos métodos pueden cambiar en la implementación final y son orientativos para explicar la funcionalidad de las clases, pero las dependencias se mantendrán. A continuación, se explican estas clases:

- **MeasureTenant:** representa todos los datos que se van a tratar en el primer módulo, es decir, las medidas de los sensores.
- **MqttConnection:** se conectará a la tubería MQTT y recibirá los mensajes entrantes (las medidas). Tras recibir un mensaje, lo enviará al componente MessagesProcessor.
- **MessagesProcessor:** se encargará de procesar los mensajes entrantes, como su nombre indica. El primer paso será traducir los datos entrantes a datos entendibles por el sistema mediante PayloadConverter, el segundo será añadir los campos adicionales necesarios usando las clases Requester, y el tercero será enviar el mensaje procesado a la base de datos mediante IMeasureTenantService.
- **PayloadConverter:** mediante la documentación de la empresa y los diagramas del diseño de la tubería de mensajes MQTT se puede saber que los datos se intercambian como listas de bytes, por lo que será necesario un componente que traduzca estos datos en clases del sistema (en objetos MesureTenant).
- **TenantRequester y SensorCategoryRequester:** realizarán peticiones a distintas bases de datos mediante servicios externos de la empresa para obtener los dos datos adicionales que no vienen con el mensaje MQTT (organización y categoría del sensor).
- **MeasureTenantService y MeasureTenantDAO:** almacenarán la medida procesada en la base de datos.

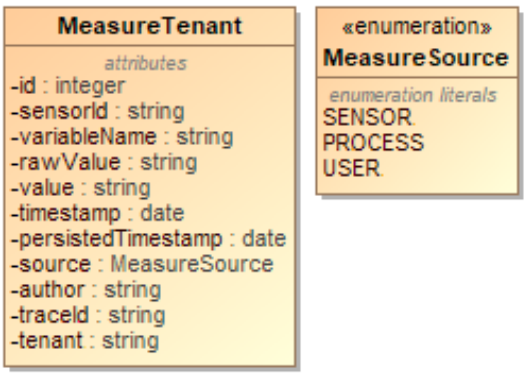

Figura 4.2: Datos a almacenar en el primer módulo

Por otro lado, están los módulos externos. Estos son simplemente servicios que realizan peticiones a las bases de datos de la empresa y, como se comentaba anteriormente, estas peticiones se realizan desde las clases Requester, que se pueden ver en la figura 4.1. La información que necesitan y la que ofrecen se comenta en el capítulo de análisis.

Finalmente, la base de datos es muy simple. Tan solo se compone de una tabla, la tabla "medida", en la cual se almacenarán constantemente todas las medidas tras ser procesadas. Los campos de dicha tabla ya se pueden ver en la figura 4.2. No se ha creado un diagrama entidad-relación, ya que sería redundante teniendo en cuenta la simplicidad del caso.

4.1.2. Segundo módulo

El segundo módulo, o módulo "web", se parece mucho más a una aplicación web tradicional, y los datos a tratar son algo distintos en comparación al módulo anterior. De esta forma, el diagrama conceptual (siguiendo el modelo entidad-relación) de la figura 4.3 muestra las tres entidades principales a tener en cuenta: las organizaciones, los sensores y los planes. Estas entidades y sus datos se describen en el capítulo del análisis.

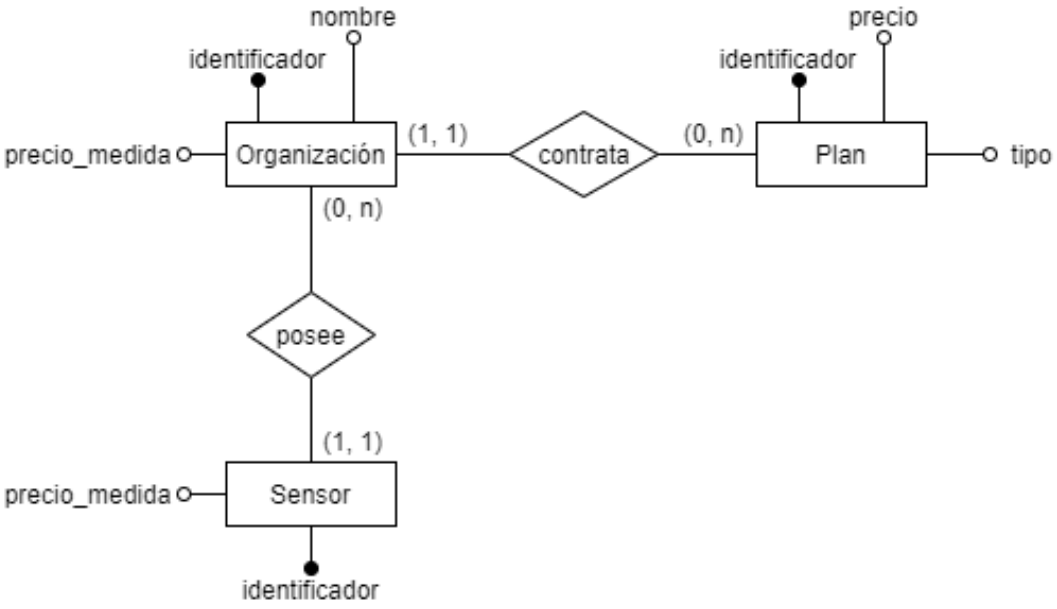

Figura 4.3: Diseño conceptual mediante el modelo entidad-relación del segundo módulo

Además de esto, se ha de tener un punto adicional en cuenta. Como se puede ver en el diagrama entidad-relación, los sensores tienen un identificador, y este identificador es el que se almacena en cada medida en el primer módulo. Pero esta relación no se debe tener en cuenta, ya que no todos los sensores existentes gestionados por la empresa se representarán mediante la entidad Sensor, al no tener todos ellos un coste por medida exclusivo (solo se almacenan los sensores que sí que tengan dicho coste exclusivo). Para el resto de sensores se utiliza el coste por medida de la organización.

Finalmente, la relación entre el nombre de la organización en las medidas y el campo nombre en la entidad Organización no se puede dar. En este caso, el motivo es la eficiencia, ya que se busca que las inserciones de medidas sean lo más rápidas posibles, y una unión de este tipo ralentizaría el proceso. Este último punto está más relacionado con la implementación que con el diseño, por lo que esto se detalla en dicho capítulo.

Todos los cálculos que puede necesitar la parte web del módulo se realizan mediante peticiones a la base de datos, por lo que la aplicación se limita a enviar esta información calculada a las vistas de la aplicación. Las comunicaciones entre el backend y el frontend del módulo web se explican en el capítulo de implementación. Por último, las vistas y su contenido se muestran en el siguiente apartado, diseño de la interfaz.

4.2. Diseño de la interfaz

4.2.1. Diseño plano

El diseño plano o *flat design* es un estilo de diseño de interfaces que busca simplificar el aspecto de una aplicación para facilitar su uso [7]. En este estilo se suprimen la mayoría de decoraciones que podrían caracterizar otros tipos de diseño, como los degradados, las texturas, las sombras, etc. Algunos ejemplos de interfaces que usen este estilo son las de Windows Phone y Windows 10. En este caso en concreto se utilizan iconos muy grandes y con la información justa y necesaria para identificar su función, además de una paleta de colores reducida, con elementos que contrastan entre sí.

Este es el estilo que se usa comúnmente en la empresa en la que se desarrolla el proyecto, ya que las interfaces de usuario suelen estar cargadas de información, dada la necesidad de simplificar el gran volumen de datos que se muestran y las múltiples formas de representarlos que se usan. En este proyecto se ha utilizado un diseño plano en todos los componentes, las cuales se pueden ver en la siguiente subsección.

4.2.2. Vistas de la organización

Antes de la creación de las vistas finales, se realizaron mockups usando los propios componentes que se iban a utilizar en la aplicación final, pero con datos falsos fijos guardados en el propio código. La figura 4.4 es un ejemplo de estos mockups.

Las vistas finales terminaron variando ligeramente en el estilo y el posicionamiento de los elementos, pero conservan ese diseño plano que se buscaba. En la figura 4.5 se puede ver la vista final principal de la organización. Si el usuario selecciona un elemento de la tabla de sensores, se le redirigirá a la vista 4.6, donde puede ver en detalle las medidas tomadas por cada sensor.

Los datos que se ven en ambas vistas son datos falsos que se insertaron tan solo para realizar pruebas de la interfaz de usuario, ya que la empresa prefería que el acceso a la base de datos real no se diese por motivos de seguridad.

4.2.3. Vista del administrador

En la figura 4.7 se encuenta la vista principal del administrador, en la cual se pueden ver las organizaciones almacenadas en la base de datos. De nuevo, al usarse una base de datos ajena a

la base de datos principal de la empresa por motivos de seguridad, la información que se ve es falsa, insertada solamente para poder probar la interfaz de usuario.

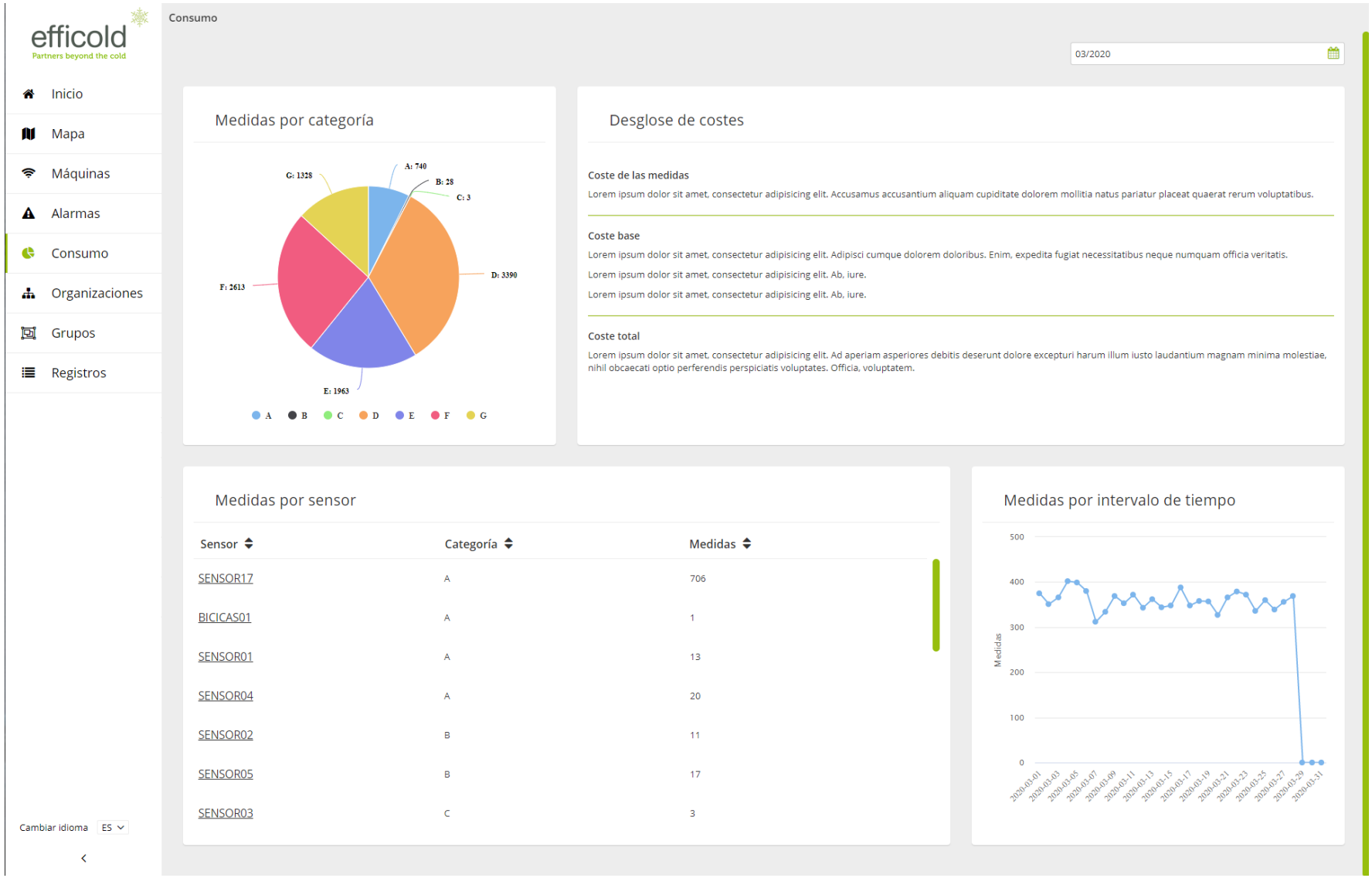

Figura 4.4: Mockup inicial de la vista principal de la organización

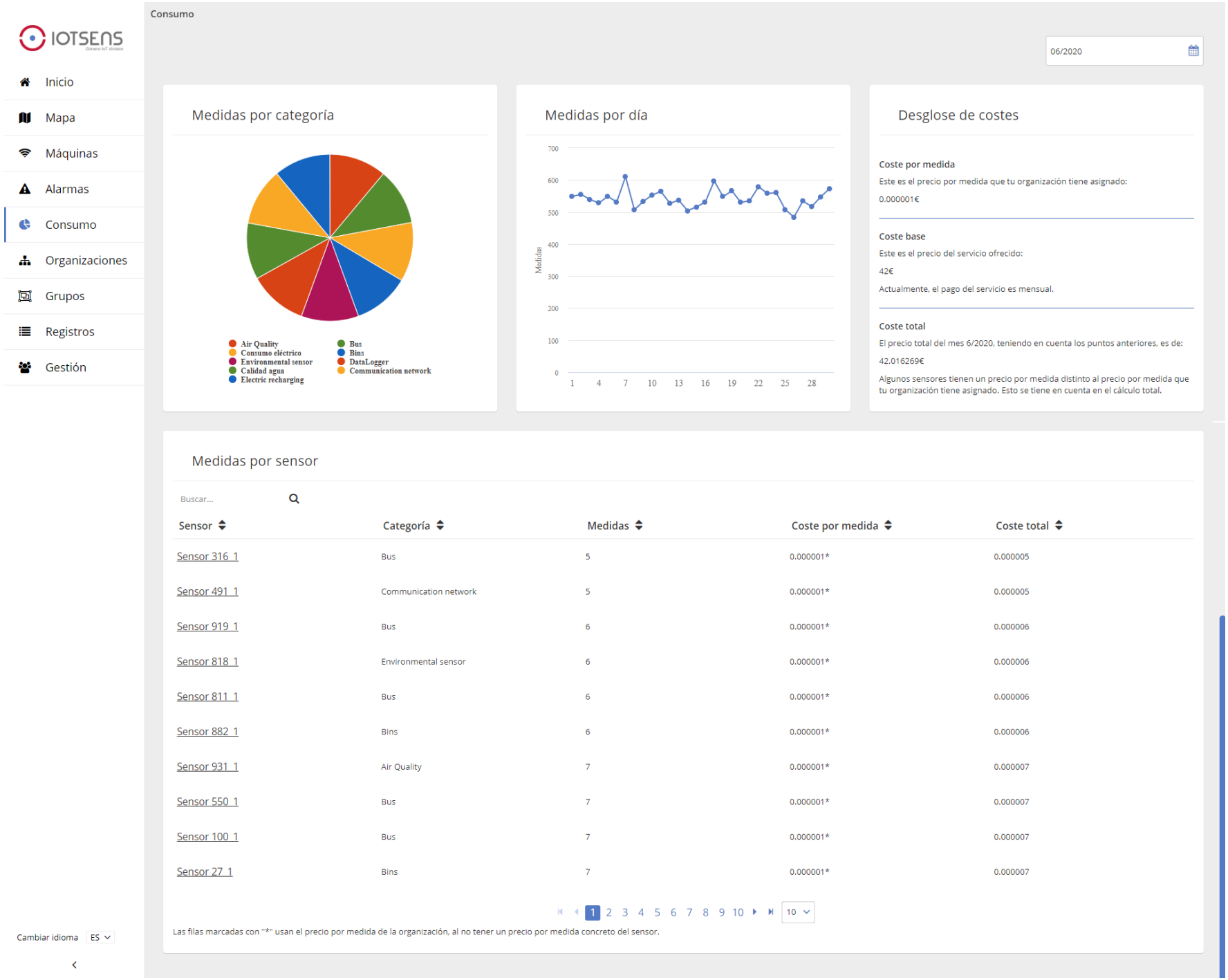

Figura 4.5: Vista principal de la organización

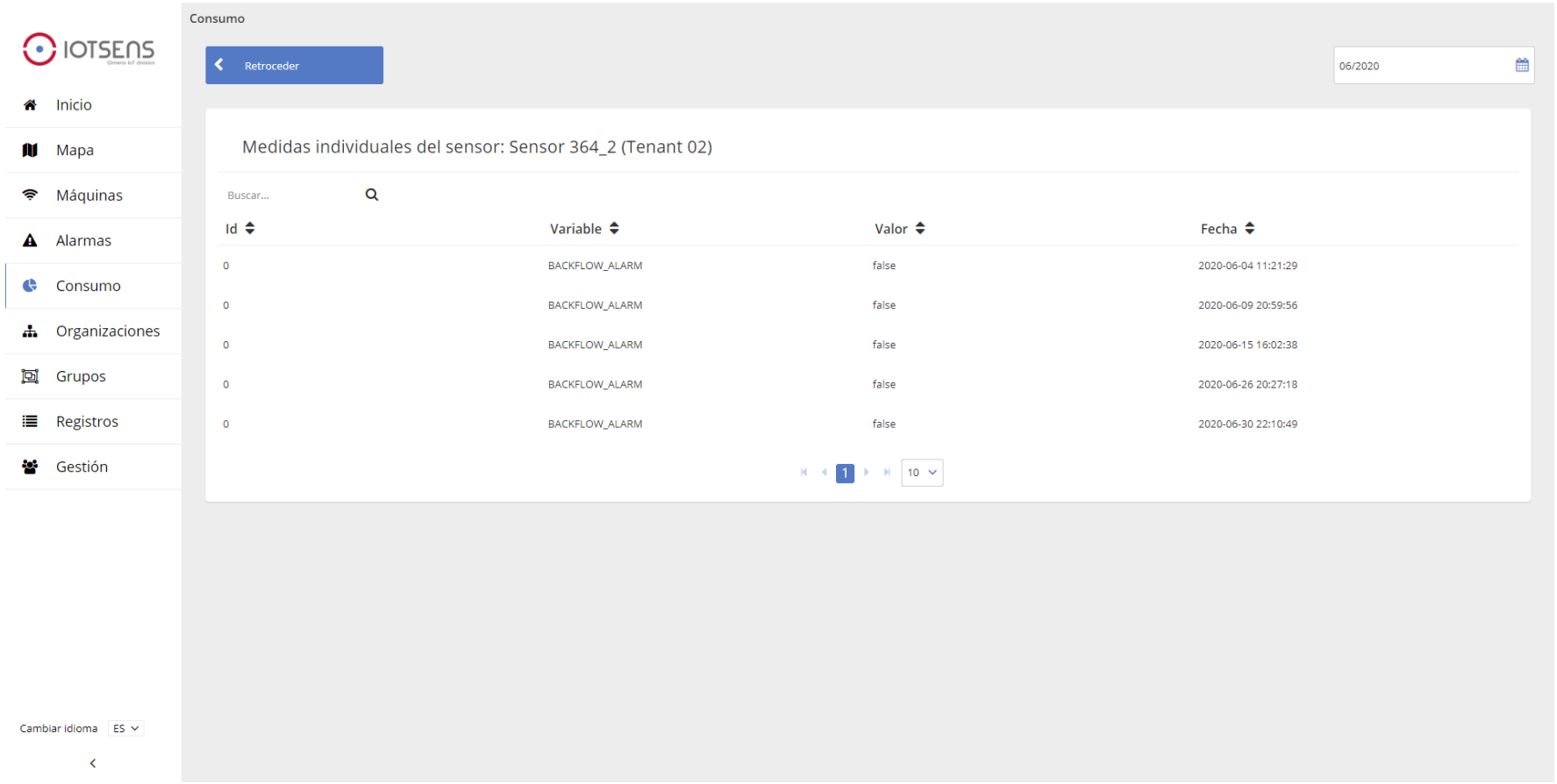

Figura 4.6: Vista de las medidas de un sensor

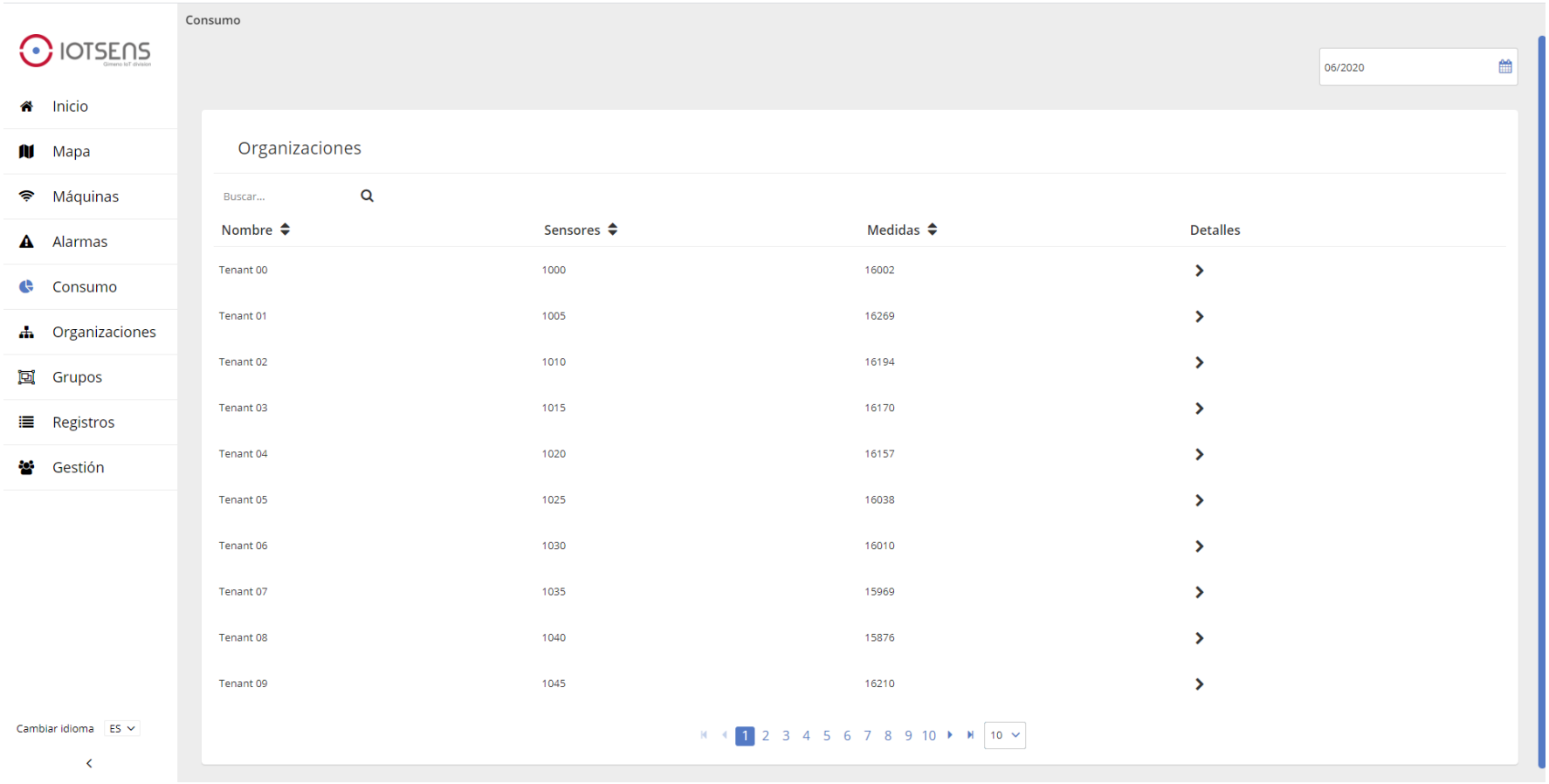

Figura 4.7: Vista del administrador

Capítulo 5

Implementación y pruebas

5.1. Detalles de implementación

5.1.1. Patrones de diseño

A lo largo del desarrollo del proyecto se han usado varios patrones de diseño distintos, los cuales se describen en esta sección. Estos patrones facilitan la resolución de problemas muy conocidos en el entorno del desarrollo de *software*, aportando soluciones que se pueden aplicar rápidamente y evitando el tener que diseñar múltiples veces la misma solución para problemas similares.

Strategy

Este patrón de diseño permite el intercambio entre componentes que realizan la misma tarea, pero de formas distintas. Esto suele ser especialmente útil cuando nos encontramos ante un problema que puede ser resuelto de formas distintas, permitiéndonos incluso utilizar diferentes algoritmos para resolver el mismo problema en tiempo de ejecución. En este proyecto, el patrón Strategy se utiliza en el primer módulo, en concreto en el servicio encargado de almacenar las medidas en la base de datos. Como se puede ver en la figura 5.1, el servicio concreto MeasureTenantService (el que actualmente se encarga de guardar los datos en la base de datos) puede ser intercambiado en cualquier momento por otro servicio que, por ejemplo, guarde los datos en un fichero, o realice operaciones distintas con ellos.

Pese a que la aplicación no cuente con distintos componentes a intercambiar por el momento, una implementación de este tipo permite extender la funcionalidad del programa en un futuro sin necesidad de modificar las clases ya existentes. Esto, a su vez, cumple con uno de los principios SOLID, en concreto el Open/closed principle (principio de abierto/cerrado). Este principio representa la idea de que cada entidad de *software* debería estar abierta para su extensión, pero cerrada para su modificación. Esta idea, junto al resto de principios SOLID, permite el desarrollo de código más limpio y fácilmente entendible por cualquier desarrollador.

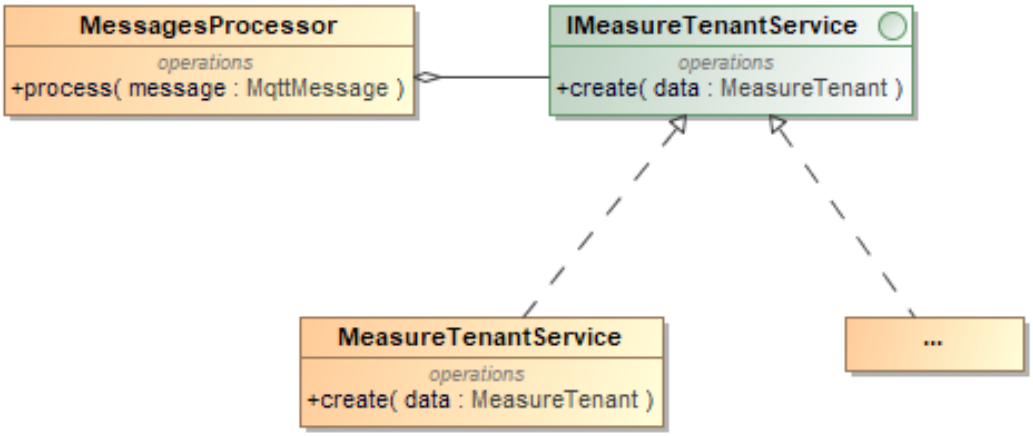

Figura 5.1: Uso del patrón Strategy

Observer

Este patrón también se da en el primer módulo. El patrón observer permite que un objeto notifique al resto cuando sea necesario, normalmente cuando sucede un cambio de algún tipo. Esto evita que el objeto observador (el que es notificado) realice peticiones constantemente al objeto que cambia para conocer su estado.

En concreto, este patrón se puede ver aplicado en la case MqttConnection, al ser esta clase es un observador de la tubería MQTT de medidas de la empresa. Cuando esta tubería recibe un mensaje nuevo, se produce una llamada al método messageArrived() de la clase MqttConnection, incluyendo en esta llamada el mensaje en sí. Tras esto, el mensaje es procesado internamente en el resto del programa. En la figura 5.2 se muestra una posible implementación del patrón Observer aplicada al caso actual, ya que la sección de código de la empresa en la que se encuentra la implementación concreta del patrón no es accesible.

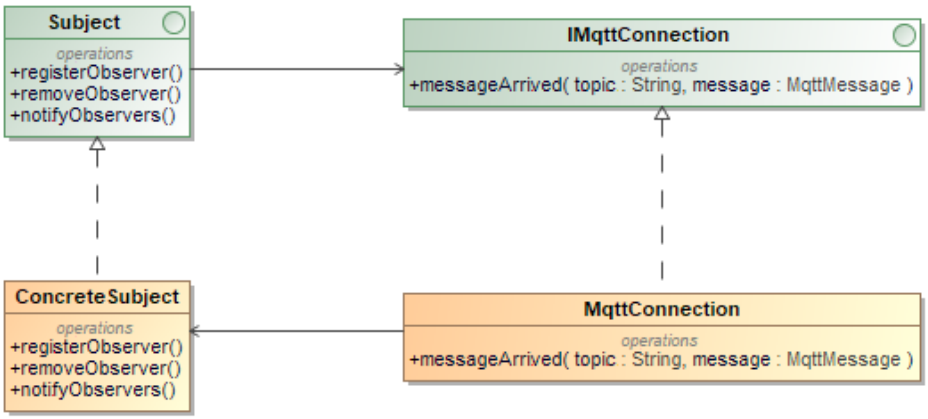

Figura 5.2: Aproximación al uso del patrón Observer

Inyección de dependencias

La inyección de dependencias es un patrón usado comúnmente en la programación orientada a objetos, a través del cual se puede implementar la inversión de control en una aplicación. Esto permite, en conjunto, que no se creen objetos de forma directa en las clases en las cuales no se quiere tener una dependencia. En su lugar, los objetos necesarios para su funcionamiento son inyectados en la propia clase, normalmente mediante métodos set o a través del constructor.

En este proyecto, la inyección de dependencias es más fácil de implementar y usar, ya que se trabaja con el entorno Spring. Este entorno permite indicar mediante etiquetas qué objetos se deben crear para el correcto funcionamiento de la aplicación. Durante la ejecución del programa, estos objetos serán proporcionados a las clases que los usen, sin la necesidad de instanciar objetos de forma directa y, por lo tanto, sin crear dependencias innecesarias adicionales o clases que actúen como fábricas de objetos.

Este patrón o técnica se ha usado a lo largo de todo el proyecto, pero en la figura 5.3 se muestra un ejemplo de código que aprovecha las etiquetas de Spring (en concreto @Autowired) para realizar esta inyección, y en la figura 5.4 se ve una aplicación más común del patrón mediante el método set en el segundo módulo o módulo web, el cual permite intercambiar entre distintos objetos de la misma clase en tiempo de ejecución.

```
public class MessagesProcessor {

  @Autowired
  private IMeasureTenantService measureTenantService;

  @Autowired
  private TenantRequester tenantRequester;

  @Autowired
  private SensorCategoryRequester sensorCategoryRequester;

  @Autowired
  private PayloadConverter payloadConverter;

  // ...
}
```

Figura 5.3: Inyección de dependencias de Spring

5.1.2. Peso de la eficiencia en el proyecto

Un factor muy relevante a tener en cuenta a lo largo de todo el proyecto ha sido la eficiencia. Las velocidades de procesamiento y almacenamiento de datos han condicionado en muchos sentidos el aspecto del producto final. Esto ha provocado que se tomen decisiones y precauciones adicionales adaptadas a cada problema que ha surgido durante el desarrollo. En este apartado se comentan los problemas más relevantes, así como las soluciones propuestas e implementadas.

En primer lugar, el módulo encargado de recibir, procesar y almacenar medidas tenía un

```
public class ConsumptionsService {

  //...

  public void setMeasureTenantDAO(MeasureTenantDAO measureTenantDAO) {
    this.measureTenantDAO = measureTenantDAO;
  }

}
```

Figura 5.4: Inyección de dependencias mediante un método *set*

margen de tiempo muy estrecho con el que operar. La tubería de medidas de IoTsens recibe al día un número de medidas cercano a los mil millones. Para la realización de este proyecto se necesitaba procesar aproximadamente 20 millones de medidas diarias, unas 230 por segundo. El módulo terminó siendo capaz de tratar unas 400 medidas por segundo, incluyendo los tiempos de almacenamiento en la base de datos y de procesamiento de la medida recibida. Las modificaciones realizadas para agilizar este proceso fueron las siguientes:

Caché en las clases Requester: uno de los principales cuellos de botella de la aplicación era el uso de los servicios de la empresa para averiguar la organización y la categoría del sensor que toma la medida. Las clases encargadas de interactuar con estos servicios (clases Requester) se pueden ver en la figura 5.5. Cada una de estas peticiones tardaba unos 270 milisegundos en terminar. Teniendo en cuenta que se realizan 2 peticiones por medida, esto haría que el módulo no pudiese ni procesar 2 medidas por segundo. Este problema se solucionó añadiendo cachés en las clases TenantRequester y SensorCategoryRequester para almacenar estos datos como mapas en memoria principal. De esta forma, cuando todos los sensores registrados en la empresa hayan enviado al menos una medida, la obtención de estos datos pasará a ser prácticamente instantánea.

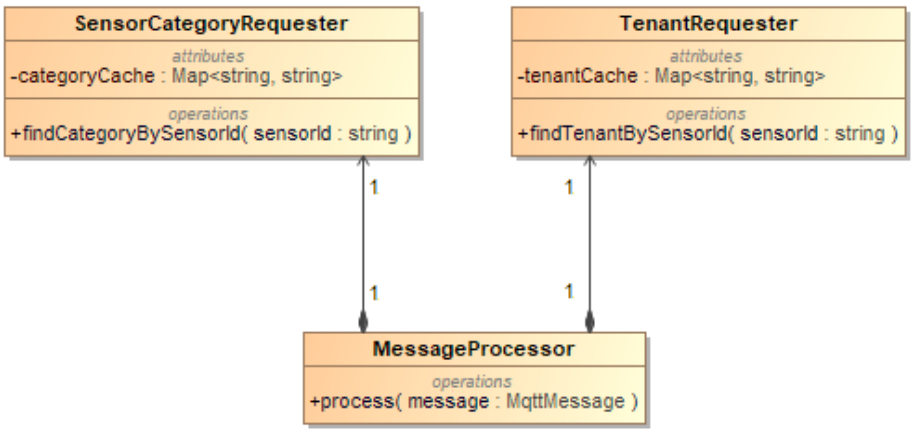

Figura 5.5: Clases Requester del primer módulo

Optimización de inserciones: otro cuello de botella que afectaba al primer módulo era la velocidad de las inserciones de datos en la base de datos, ya que identificar los campos

necesarios para definir cada medida no resultan obvios sin conocimiento del funcionamiento interno de los sensores. Para ello, se consideraron distintas combinaciones de claves primarias que identificasen de forma inequívoca cada medida, pero usando el mínimo número de campos posible. Se terminó usando una combinación del identificador del sensor, la marca temporal y el nombre de la variable (ya que un mismo sensor puede realizar en el mismo momento dos medidas de dos variables distintas). Una mejora que se planteó fue usar un único campo que agrupase la información de todos estos datos mediante un hash, pero esto terminó siendo menos eficiente que la solución usada, además de suponer un coste de almacenamiento adicional. La selección de claves primarias adecuadas redujo el tiempo por inserción de 7ms a tan solo 2ms.

Optimización de peticiones: este problema afectó al segundo módulo, al ser este el que hace peticiones a la base de datos para realizar los cálculos necesarios a mostrar al usuario final. Este terminó siendo un problema muy difícil de solucionar, ya que el principal cuello de botella se encontraba en el *hardware* de la base de datos, algo que escapa al alcance del proyecto. Por otro lado, el lado *software* de este problema (las peticiones en sí realizadas desde el programa) se trató en medida de lo posible. Algunas peticiones no podían ser reducidas dada su simplicidad (por ejemplo, obtener el número total de medidas de una organización), pero otras más complejas pudieron simplificarse poco a poco para ahorrar todo el tiempo posible. Este proceso de optimización se produjo durante el cuarto esprint, llegando a reducir a la mitad los tiempos de las peticiones más pesadas.

5.2. Verificación y validación

En esta sección se describen las pruebas que se han realizado para comprobar el correcto funcionamiento de la aplicación. En concreto, se han separado las pruebas realizadas según su nivel de especificidad (de unidad, de integración y de sistema). Estas pruebas no son todas las que se van a realizar sobre el producto final, y servían simplemente como ayuda durante el desarrollo, ya que serán empleados especializados en la empresa quienes se dediquen a realizar pruebas en mucho más detalle, y quienes se encargarán de terminar de integrar los componentes desarrollados en la plataforma real.

Pruebas de unidad

Estas pruebas se realizaron sobre los componentes encargados de realizar los cálculos en el segundo módulo o módulo web. Dado que el resto de las peticiones eran relativamente simples y fáciles de comprobar, se terminaron realizando pruebas unitarias sobre las dos más complejas. Además, como ambas peticiones tienen los mismos datos de entrada, su análisis de clases de equivalencia se muestra una sola vez en la tabla de la figura 5.6. Las fechas se almacenaban como enteros, de ahí las clases válidas e inválidas para estas entradas.

Para la realización de estas pruebas, y teniendo en cuenta que tan solo se realizaban consultas y no modificaciones, la base de datos se llenó una única vez de datos generados aleatoriamente, los cuales fueron comparados con los datos obtenidos de las peticiones durante la realización de las pruebas.

Dato	Clases válidas	Clases inválidas
in: fecha_inicial	V1: fecha_inicial > 0	I1: fecha_inicial <= 0
in: fecha_final	V2: fecha_final > 0	I2: fecha_final <= 0
in: organización	V3: organización != null && organización != “”	I3: organización == null
		I4: organización == “”
(fecha_inicial, fecha_final)	V4: fecha_inicial <= fecha_final	I5: fecha_inicial > fecha_final
out: datos solicitados	VS1: conjunto de datos solicitados	IS1: “Fechas no válidas”
		IS2: “La organización no existe”

Figura 5.6: Análisis de clases de equivalencia de las pruebas unitarias

Obtención de medidas por categoría

Esta petición era la encargada de devolver el número de medidas tomadas por los sensores en un rango de fechas dado, clasificados según la categoría del sensor que tomó las medidas. En la figura 5.7 se puede ver la tabla con los casos de prueba.

ID	Entrada	Salida esperada	Escenarios cubiertos
1	(-1, 1589298862), “Facsa”	“Fechas no válidas”	I1, IS1
2	(1589298862, 1586706862), “Facsa”	“Fechas no válidas”	I5, IS1
3	(1589298862, -1), “Facsa”	“Fechas no válidas”	I2, IS1
4	(1586706862, 1589298862), “Facsa”	{"A":35, "B":46, "C":15}	V1, V2, V3, V4, VS1
5	(1586706862, 1594569262), “Facsa”	{"A":45, "B":48, "C":22, “D”:12, “E”:7}	V1, V2, V3, V4, VS1
6	(1586706862, 1589298862), “”	“La organización no existe”	I4, IS2
7	(1586706862, 1589298862), null	“La organización no existe”	I3, IS2
8	(1586706862, 1589298862), “Faksa”	“La organización no existe”	IS2

Figura 5.7: Casos de prueba de la petición de medidas por categoría

Obtención de medidas por sensor

En este caso, el objetivo de la petición era devolver el número de medidas que habían realizado los sensores en el rango de fechas especificado, clasificados según el identificador del sensor. Se usaron las mismas entradas y los mismos datos en la base de datos que en el caso anterior, al tratarse de una petición muy similar en la que solo cambia el formato de los datos de salida. En la figura 5.8 se puede ver la tabla con sus casos de prueba.

ID	Entrada	Salida esperada	Escenarios cubiertos
1	(-1, 1589298862), “Facsa”	“Fechas no válidas”	I1, IS1
2	(1589298862, 1586706862), “Facsa”	“Fechas no válidas”	I5, IS1
3	(1589298862, -1), “Facsa”	“Fechas no válidas”	I2, IS1
4	(1586706862, 1589298862), “Facsa”	{"SN01":21, "SN02":8, "SN03":22, "SN0	V1, V2, V3, V4, VS1
5	(1586706862, 1594569262), “Facsa”	{"SN01":26, "SN02":15, "SN03":39, "SN	V1, V2, V3, V4, VS1
6	(1586706862, 1589298862), “”	“La organización no existe”	I4, IS2
7	(1586706862, 1589298862), null	“La organización no existe”	I3, IS2
8	(1586706862, 1589298862), “Faksa”	“La organización no existe”	IS2

Figura 5.8: Casos de prueba de la petición de medidas por sensor

Pruebas de integración

El proyecto se ha desarrollado adoptando una estrategia top-down, es decir, de alto nivel a bajo nivel. Esto significa que los primeros componentes que se desarrollaron fueron aquellos con nivel mayor de abstracción y con menos dependencias.

En la realización de pruebas de integración también se utilizó la estrategia top-down, una estrategia incremental, la cual acompañaba al desarrollo. Por lo que, cada vez que se implementaba un nuevo componente, éste se creaba como un *stub* (un *stub* simula el funcionamiento de un componente real, sin ser el componente real final). Tras esto, se implementaba su funcionalidad real y se creaban los stubs adicionales necesarios, es decir, una imitación de los futuros componentes reales que dependerían del componente recién implementado. En la figura 5.9 se puede ver, como ejemplo, el orden de implementación de los componentes del primer módulo.

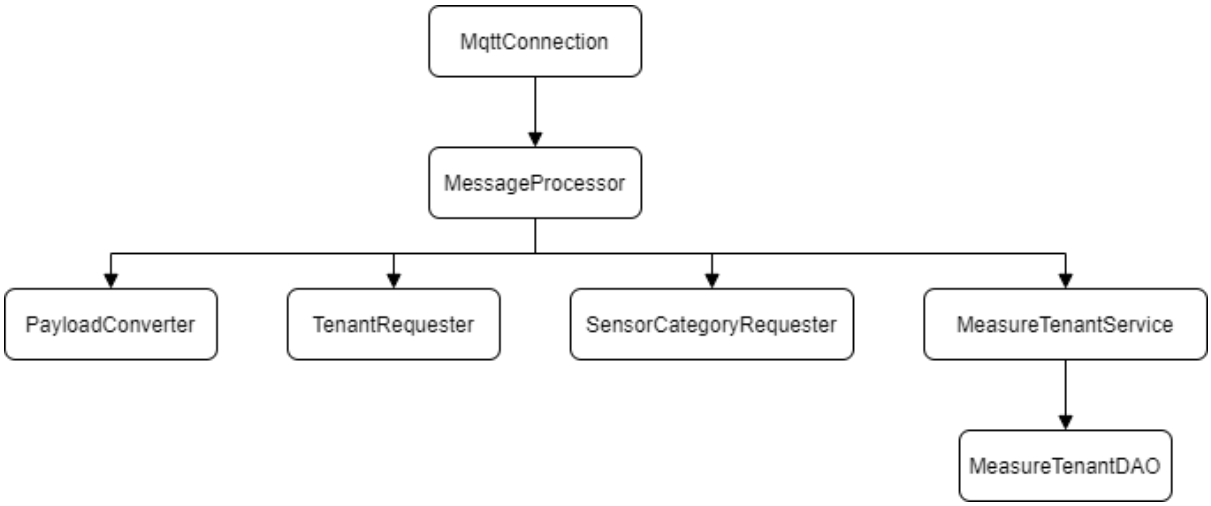

Figura 5.9: Orden de implementación de los componentes del primer módulo

La mayoría de pruebas de integración que se realizaron durante el desarrollo del proyecto

fueron manuales (sin automatizar mediante JUnit o similares), ya que muchos de los componentes eran muy simples y se limitaban a enviarse datos entre sí o a intercambiar datos con servicios externos (recibir mensajes MQTT, pedir el nombre de la organización de un sensor, etc.).

Pruebas de sistema

Para describir el tipo de pruebas de sistema que se realizaron, primero es necesario separar en primer módulo y segundo módulo, ya que ambos tenían funciones muy distintas y actuaban como componentes independientes (pese a pertenecer al mismo proyecto). De cada módulo se comentarán el tipo de pruebas que se realizaron según su objetivo (de instalación, de regresión, de rendimiento, de usabilidad, etc.). De esta forma, las pruebas realizadas fueron las siguientes:

Primer módulo: la eficiencia era un punto clave a tener en cuenta durante del desarrollo, en especial en el primer módulo. De esta forma, el principal tipo de pruebas que se realizaron fueron de rendimiento. En estas pruebas, el primer módulo recibía mensajes constantemente, y se registraban los tiempos medios dedicados a cada etapa del procesamiento. Los tiempos obtenidos se comentan en la sección anterior, en la cual también se comentan las distintas medidas que se tomaron para optimizar los distintos procesos.

Segundo módulo: en este caso, y al haber probado ya las operaciones más complejas que se daban en el backend de este módulo mediante pruebas de unidad, las pruebas de sistema que se realizaron fueron principalmente de usabilidad, y algunas de ellas de seguridad (al tener una vista de administrador no accesible por cualquier usuario). En todas estas pruebas, el jefe del proyecto y otro empleado de la empresa actuaron como usuarios finales, dando feedback durante todo el desarrollo del frontend.

Capítulo 6

Conclusiones

El proyecto ha resultado ser un reto bastante interesante. Considero que el trabajar con gente experimentada, tanto en las prácticas de la empresa como en el campo de la informática en general, ha sido un proceso del cual he aprendido mucho, a un ritmo mucho más rápido del que esperaba. Además, la empresa me ha proporcionado las herramientas necesarias para experimentar y trabajar con bastante libertad, siempre y cuando el progreso fuese visible, y esto es algo que se agradece.

En cuanto a la metodología elegida, considero que ha sido un acierto, dado el entorno de cambio en el que se ha desarrollado todo el proyecto (cambio en los requisitos solicitados por la empresa, cambios en la plataforma de la empresa, etc.). El hecho de poder reconsiderar y modificar los requisitos y las tareas a realizar tras cada esprint, es algo que me ha permitido enfocarme más en lo que desarrollaba en cada momento, en vez de tener que realizar una planificación completa y detallada de un proyecto entero desde un inicio. Además, considero que esto ha ayudado a que el proyecto se haya podido terminar sin problemas o desviaciones importantes.

Finalmente, he de mencionar que la experiencia en conjunto ha sido muy positiva, en gran parte porque tanto la empresa como mis compañeros de departamento me han ayudado mucho con todo lo que he ido necesitando, y el hecho de ver cómo se trabaja en proyectos tan grandes como los que lleva la empresa me ha ayudado a ver el esfuerzo que hay detrás de ellos.

Bibliografía

[1] Grupo Gimeno, "Nuestro grupo." https://www.grupogimeno.com/grupo-gimeno/. [Consulta: 17 de Junio de 2020].

[2] Spring, "Spring framework." https://spring.io/projects/spring-framework. [Consulta: 17 de Junio de 2020].

[3] Oles, Bart, "An introduction to timescaledb." https://severalnines.com/database-blog/introduction-timescaledb. [Consulta: 17 de Junio de 2020].

[4] D. Uluca, *Angular for Enterprise-Ready Web Applications*. Packt Publishing Ltd., 2018.

[5] N. K. R. y Frank Turley, *Los Fundamentos de Agile Scrum*. Van Haren Publishing, 2019.

[6] Casanova, Samuel, "Estimación ágil con la técnica planning poker." https://samuelcasanova.com/2016/01/estimacion-agil-con-la-tecnica-planning-poker/. [Consulta: 17 de Junio de 2020].

[7] Contreras, Ana, "Buenas prácticas para un flat design." https://blog.interactius.com/buenas-pr%C3%A1cticas-para-un-flat-design-8b5810d9992b. [Consulta: 17 de Junio de 2020].

Anexo A

Prueba de rendimiento de la base de datos

A.1. Contexto

Esta fue una tarea ajena al proyecto que se desarrolló principalmente durante el cuarto esprint del desarrollo, cuyo objetivo era comprobar los tiempos de inserción y de petición de la base de datos usada para almacenar medidas.

Para ello se desarrollaron dos proyectos a parte, uno en Java para generar e insertar datos en masa y obtener los tiempos de cada petición tras insertar 100.000 datos, y otro en Python para interpretar los resultados obtenidos como las gráficas que se muestran en este anexo. Ambos proyectos eran relativamente simples, por lo que no se va a detallar su implementación.

Inicialmente se quería llegar a los mil millones de medidas en la base de datos, pero tanto las inserciones como las peticiones fueron ralentizándose demasiado a medida que aumentaban los datos almacenados, dadas las limitaciones de *hardware*. Pese a esto, se llegaron a realizar unas 800 iteraciones de 100.000 inserciones cada una, es decir, se llegaron a insertar unas 80.000.000 medidas.

En el siguiente apartado se pueden ver los resultados obtenidos.

A.2. Resultados

En todas las gráficas de esta sección se presentan los datos de la misma forma: en el eje horizontal se encuentra la iteración, y en el vertical el tiempo transcurrido, siendo la gráfica de las inserciones la excepción, la cual muestra en el eje vertical las inserciones por segundo, al ser una métrica más fácil de interpretar. Las iteraciones eran de 100.000 medidas cada una. Esto quiere decir que, tras insertar 100.000 medidas, se guardaba el tiempo transcurrido en las inserciones. Tras esto, se realizaban todas las distintas peticiones a probar y se guardaban los

tiempos que tardaban en ejecutarse.

En cuanto a los resultados en sí, todos los tiempos crecen linealmente conforme aumenta el volumen de los datos almacenados en la base de datos. Además de esto, en todas las peticiones (y en las inserciones, en parte) se ve un pico en torno a la iteración 600. Lo que pasó en ese momento es que la base de datos se quedó sin espacio y se amplió. Se puede ver como muchas peticiones se volvieron más rápidas tras esto, mientras que las inserciones de datos se ralentizaron. Los motivos de la ralentización de las inserciones tras la ampliación se desconocen, pero muy probablemente se deba a algún problema de configuración interno de Timescale.

Inserts

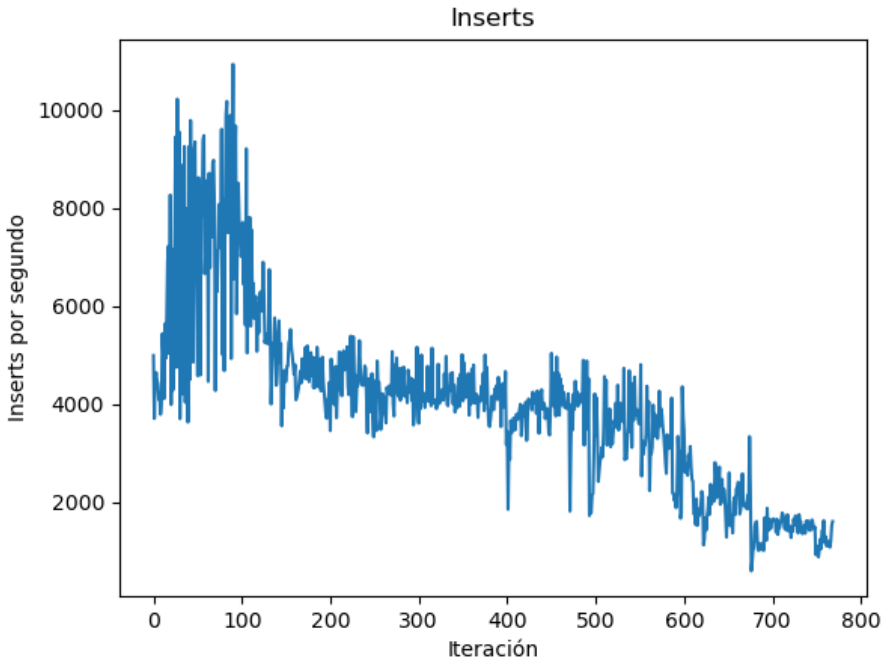

Figura A.1: Tiempos de las inserciones de datos

Las inserciones se estabilizaron en torno a los 4000-5000 por segundo, exceptuando el pico inicial (iteración 0-100) y el pico tras ampliar la base de datos comentado anteriormente, como se puede ver en la figura A.1.

Peticiones normales

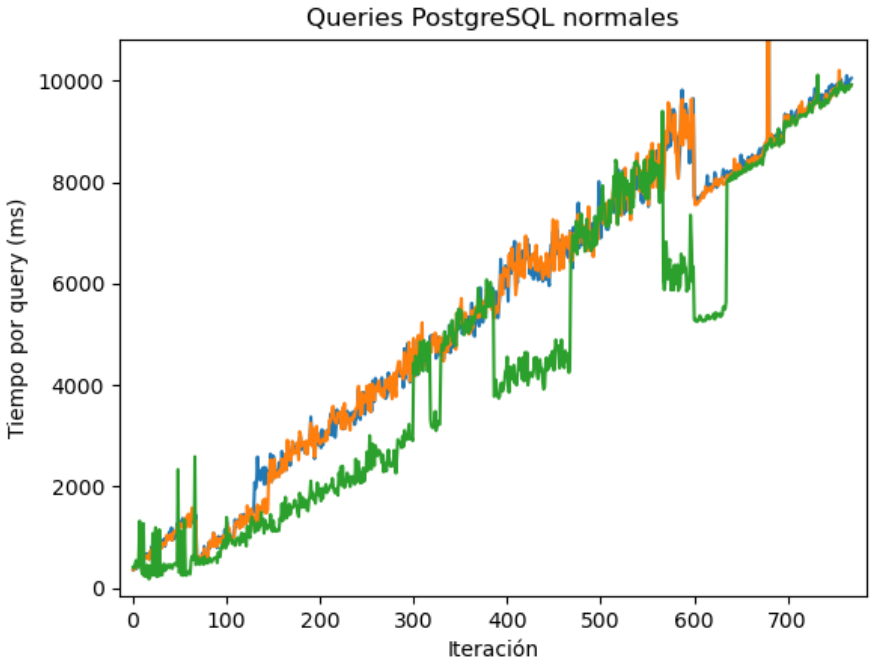

Figura A.2: Peticiones sin métodos especiales de Timescale

Estas peticiones están sacadas directamente del proyecto desarrollado. El tiempo por petición crece linealmente conforme aumenta el tamaño de la tabla. Son peticiones que no usan métodos propios de Timescale (figura A.2).

Peticiones timebucket (simples)

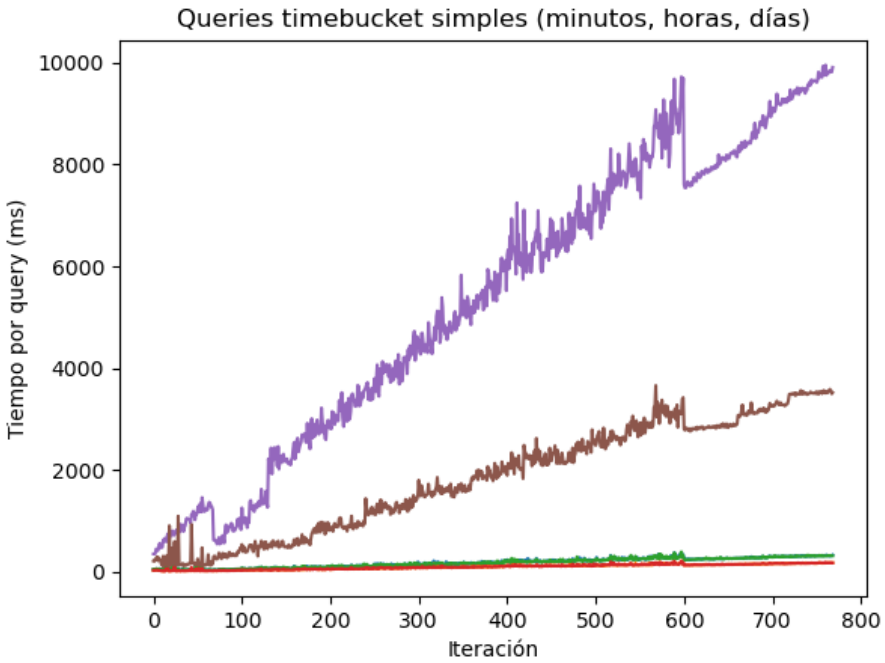

Figura A.3: Peticiones con rango de fechas corto usando métodos especiales de Timescale

Estas peticiones sí que usan métodos propios de Timescale (figura A.3). En concreto, time_bucket_gapfill, que es una variante de time_bucket. Las peticiones son:

- Línea roja: mostrar el número de medidas de una organización en intervalos de 5 minutos a lo largo de un día entero.
- Línea verde: mostrar el número de medidas de una organización en intervalos de 1 hora a lo largo de un día entero.
- Línea marrón: mostrar el número de medidas de una organización en intervalos de 1 día a lo largo de 10 días.
- Línea morada: mostrar el número de medidas de una organización en intervalos de 1 día a lo largo de un mes entero.

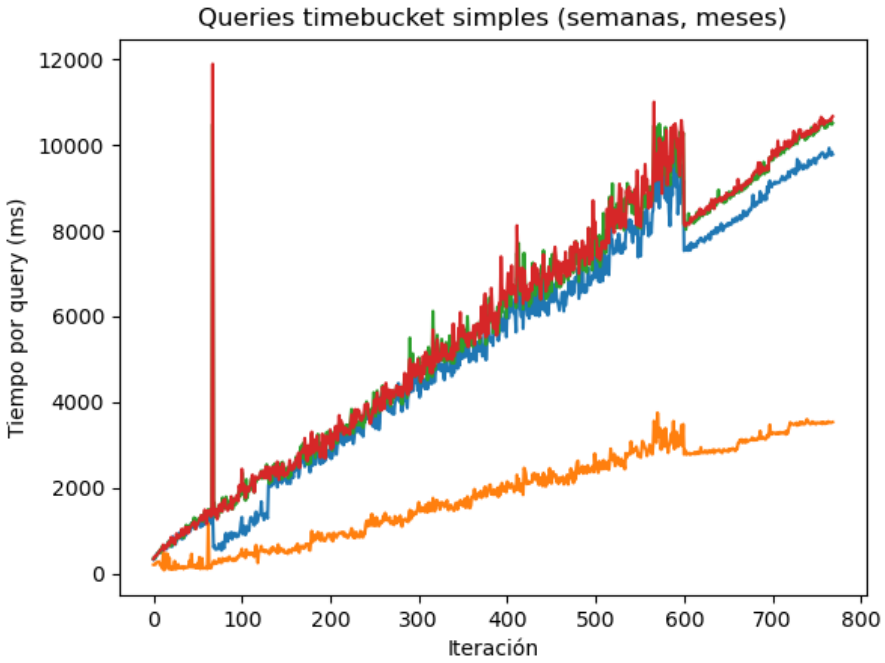

Figura A.4: Peticiones con rango de fechas medio usando métodos especiales de Timescale

Estas peticiones (figura A.4) son iguales que las anteriores, pero con distintos rangos e intervalos:

- Línea naranja: mostrar el número de medidas de una organización en intervalos de 1 semana a lo largo de 10 días.
- Línea azul: mostrar el número de medidas de una organización en intervalos de 1 semana a lo largo de un mes entero.
- Línea verde: mostrar el número de medidas de una organización en intervalos de 1 mes a lo largo de 14 meses.
- Línea roja: mostrar el número de medidas de una organización en intervalos de 1 mes a lo largo de 14 meses (rango de fechas distinto a la línea verde).

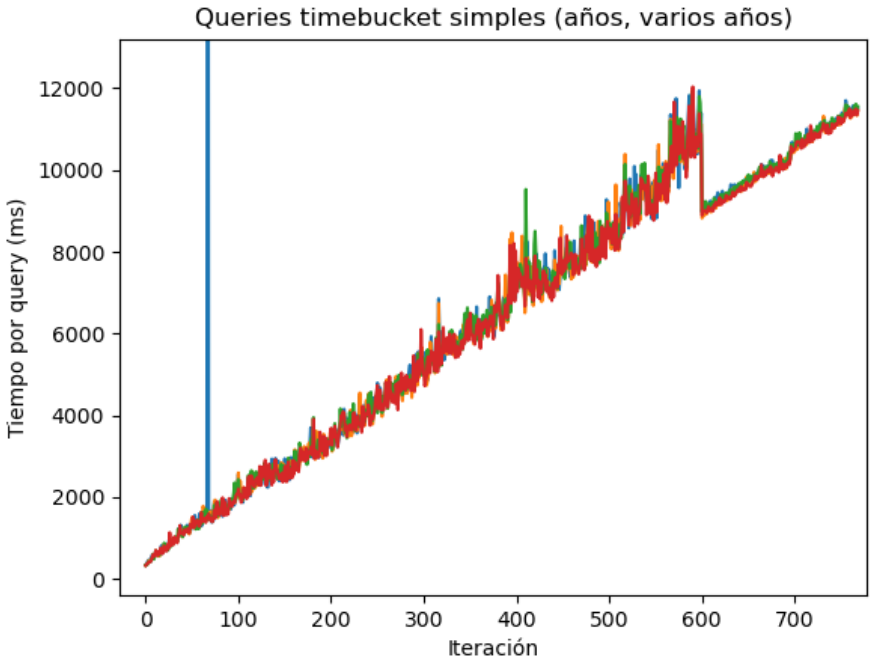

Figura A.5: Peticiones con rango de fechas largo usando métodos especiales de Timescale

Estas peticiones (figura A.5) son iguales que las anteriores, pero con distintos rangos e intervalos:

- Línea azul: mostrar el número de medidas de una organización en intervalos de 1 año a lo largo de 4 años.
- Línea naranja: mostrar el número de medidas de una organización en intervalos de 1 año a lo largo de 4 años (rango de fechas distinto a la línea azul).
- Línea verde: mostrar el número de medidas de una organización en intervalos de 2 años a lo largo de 4 años.
- Línea roja: mostrar el número de medidas de una organización en intervalos de 2 años a lo largo de 4 años (rango de fechas distinto a la línea verde).

Peticiones timebucket (complejas)

Las peticiones de las figuras A.6 y A.7 usan los mismos intervalos y rangos que las anteriores, pero tienen filtros adicionales (por ejemplo, búsqueda por categoría o por id de sensor).

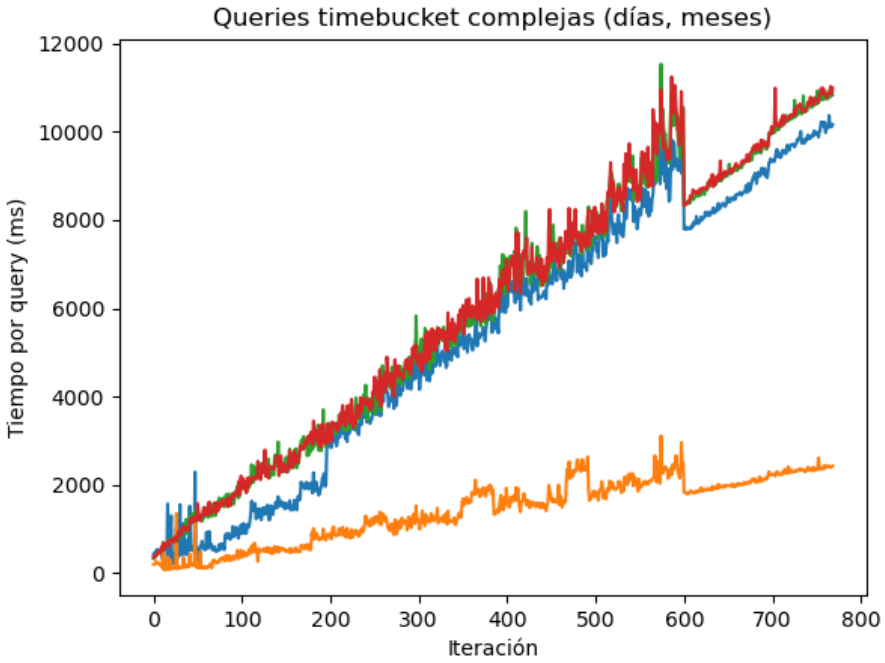

Figura A.6: Peticiones complejas con rango de fechas medio usando métodos especiales de Timescale

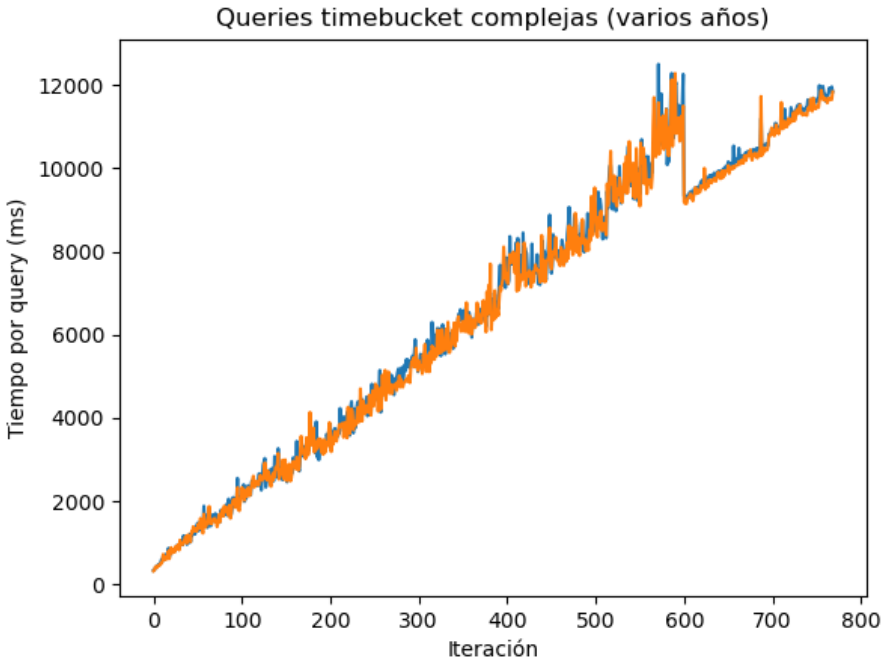

Figura A.7: Peticiones complejas con rango de fechas largo usando métodos especiales de Timescale

Peticiones inmediatamente anterior

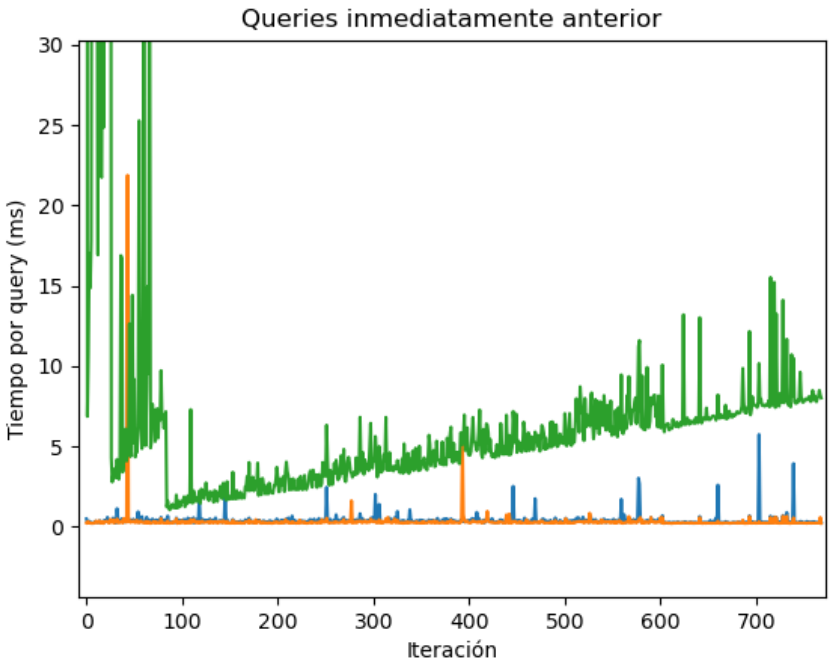

Figura A.8: Peticiones de tipo inmediatamente anterior

Estas peticiones (figura A.8) son del tipo "dada una fecha, obtener la medida inmediatamente anterior", y son prácticamente instantáneas. Se diferencian en lo siguiente:

- Línea azul: sin filtros.
- Línea naranja: especificando el identificador del sensor.
- Línea verde: especificando la organización, y buscando por identificador de sensor y categoría.