UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

MÁSTER UNIVERSITARIO DE INVESTIGACIÓN EN INGENIERÍA DE SOFTWARE Y SISTEMAS INFORMÁTICOS

ITINERARIO DE INGENIERÍA DE SOFTWARE. 31105151

Desarrollo de una solución vertical para la monitorización del suministro de agua mediante la plataforma IoTsens

---

Autor:	*Director:*	*CO-Director:*
Carlos	Pedro Javier	Ismael
CASTILLO MELGAREJO	HERRERA CARO	ABAD CARDIEL

SEPTIEMBRE, 2016/2017

MÁSTER UNIVERSITARIO DE INVESTIGACIÓN EN INGENIERÍA DE SOFTWARE Y SISTEMAS INFORMÁTICOS

ITINERARIO DE INGENIERÍA DE SOFTWARE.
31105151

Desarrollo de una solución vertical para la monitorización del suministro de agua mediante la plataforma IoTsens

Tipo B

Autor:
Carlos
CASTILLO MELAGREJO

Director:
Pedro Javier
MERRERA CARO

DECLARACIÓN JURADA DE AUTORÍA DEL TRABAJO CIENTÍFICO, PARA LA DEFENSA DEL TRABAJO FIN DE MASTER

Fecha: 02/08/2017.

Quién suscribe:
Autor(a): **Carlos Castillo Melgarejo**
D.N.I/N.I.E/Pasaporte.: **53787847-P**

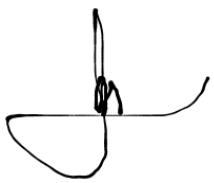
Hace constar que es la autor(a) del trabajo:
Desarrollo de una solución vertical para la monitorización del suministro de agua mediante la plataforma IoTsens.

En tal sentido, manifiesto la originalidad de la conceptualización del trabajo, interpretación de datos y la elaboración de las conclusiones, dejando establecido que aquellos aportes intelectuales de otros autores, se han referenciado debidamente en el texto de dicho trabajo.

DECLARACIÓN:

- Garantizo que el trabajo que remito es un documento original y no ha sido publicado, total ni parcialmente por otros autores, en soporte papel ni en formato digital.
- Certifico que he contribuido directamente al contenido intelectual de este manuscrito, a la génesis y análisis de sus datos, por lo cual estoy en condiciones de hacerme públicamente responsable de él.
- No he incurrido en fraude científico, plagio o vicios de autoría; en caso contrario, aceptaré las medidas disciplinarias sancionadoras que correspondan.

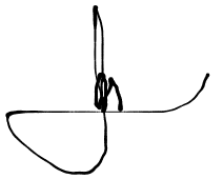
Fdo.



Autorización

Autorizo/amos a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del/los Autor/es



Resumen

El presente documento recoge la memoria del trabajo final de máster el cual consiste en el diseño y desarrollo de una solución vertical, sobre la plataforma IoTsens, para el sector del abastecimiento del agua.

Se ha realizado un estudio previo del procedimiento actual que utiliza una empresa del sector del abastecimiento del agua, para la gestión, control de consumo y facturación. A partir de este, se han detectado posibles mejoras y diseñado una serie de requisitos y funcionalidades para mejorar, optimizar y agilizar este procedimiento. También se han estudiado varias plataformas para el Internet de las cosas y, diferentes tecnologías y herramientas para el desarrollo de aplicaciones web. Posteriormente, se ha procedido con el diseño e implementación de la solución.

El trabajo realizado consta de una aplicación web que interactúa con la plataforma IoTsens a través de un API REST y permite la gestión de contadores, monitorización de datos en tiempo real como son la lectura y consumo de agua, detección de fugas, cálculos de balances, visualización de los contadores en un mapa, así como también, la de las zonas donde más agua se consume mediante un mapa de calor.

Una vez finalizado el desarrollo de la aplicación web, IoTsens ha facilitado el acceso a varios de sus sensores Watchmeter Data Logger para realizar pruebas sobre su plataforma. Estos sensores se han instalado previamente en contadores y registran datos sobre el consumo de agua mediante la detección de "patrones de flujo". Gracias al acceso a estos sensores se han podido realizar una serie de pruebas y comprobar el correcto funcionamiento de la aplicación.

Palabras clave

Internet de las cosas, IoTsens, sector vertical, abastecimiento de agua, solución vertical.

Contenidos

Figuras

Tablas

1. Introducción

En la actualidad se escuchan, y cada vez con más frecuencia, los términos Internet de las cosas y Smart city. El Internet de las cosas (IoT, por sus siglas en inglés, derivadas de Internet of Things) es un movimiento tecnológico emergente que pretende, mediante la interconexión de objetos cotidianos a Internet u otro tipo de red, la recolección e intercambio datos de forma constante para dotarlos de cierta inteligencia e independencia.

Sin embargo, el Internet de las cosas no solo tiene fines domésticos. Según un estudio realizado por la firma de análisis IDC, se estima que en 2016, sectores verticales del ámbito industrial, transporte y servicios públicos invirtieron alrededor de 260 billones de dolares en Internet de las cosas. [3]

IoTsens es una plataforma horizontal para el desarrollo de soluciones escalables e interoperables basadas en el Internet de las cosas con el fin de recolectar e intercambiar datos que conecten el mundo físico con los sistemas basados en ordenadores resultando en una mejora de su eficiencia, exactitud y beneficio económico.

En esta sección se describe de forma más amplia el contexto en el que se desarrolla el proyecto, así como su motivación y principales objetivos.

1.1 Internet de las cosas

El Internet de las cosas, es un concepto originado en el Instituto de Tecnología de Massachusetts (MIT) por Kevin Ashton en 1999, que se refiere a la interconexión entre objetos cotidianos a través de Internet o una red. El objetivo del IoT es hacer que, mediante sensores y actuadores, estos objetos se comuniquen entre sí y, por consiguiente, sean más independientes e “inteligentes”.

Un ejemplos prácticos de IoT podría ser una ciudad inteligente o Smart City, donde varios sensores sean capaces de medir la luminosidad, humedad o temperatura y actúen en consecuencia a través del encendido o apagado de farolas, sistema de riego, o, por ejemplo, un contenedor de basura capaz de determinar el estado de llenado y avisar a la empresa encargada de la recolección de residuos en caso de que supere el límite de llenado.

Figura 1.1: Diagrama representativo del concepto de IoT

Con la repercusión generada y el notable crecimiento del Internet de las cosas, han aparecido plataformas horizontales para el IoT. Estas plataformas permiten la integración sensorial y se encargan de diferentes procesos como son la gestión de sensores, almacenamiento y procesamiento de datos, etc.

1.2 IoTsens: Smart City as a Service

IoTsens es una empresa española fundada a principios de 2014 con el objetivo de proporcionar soluciones escalables e interoperables basadas en el Internet de las cosas. Disponen de una plataforma horizontal que permite el desarrollo rápido de soluciones y servicios verticales. Además de la plataforma, IoTsens también fabrica y vende una gran variedad sensores, entre los cuales se pueden encontrar de luminosidad, calidad el aire, lectura de agua, temperatura, humedad, ruido, etc.

La arquitectura de IoTsens, permite integrar todo tipo de agentes (sensores, actuadores y sistemas de información, tanto suyos como de terceros), la comunicación bidicreccional

de estos, y el tratamiento homogéneo de los datos para su análisis y explotación.

Figura 1.2: Logotipo de IoTsens

Mediante el uso de estándares se establecen redes de comunicaciones con los sensores que transmiten y consumen mensajes a través de sistemas de colas altamente escalables. El servidor central o core de la plataforma, procesa estos mensajes heterogéneos utilizando componentes específicos para cada tecnología. Una vez procesados, los datos son analizados según su naturaleza para generar alarmas, medidas, etc.

Gracias a su API REST, se pueden aprovechar la funcionalidad principal de la plataforma y crear soluciones verticales despreocupándose de toda la parte de comunicación, procesamiento, almacenamiento de mensajes, etc.

1.3 Justificación y motivación del proyecto

Desde siempre el agua ha sido un bien muy preciado, incluso en la actualidad hay países donde escasea por la mala gestión, desaprovechamiento o simplemente por circunstancias naturales. No obstante, las empresas encargadas de la gestión y abastecimientos de aguas, parecen no comulgar con las tecnologías actuales o, simplemente, no aprovecharlas suficientemente.

Estas conclusiones se han obtenido a partir de la investigación de las metodologías y procedimientos utilizados para la gestión, control de consumo y facturación en empresas de este sector. En concreto, se ha estudiado una empresa, la cual ha accedido a proporcionar información de sus metodologías de trabajo desde hace unos años hasta la actualidad, además de contar la manera en la que han evolucionado y las tecnologías y sistemas actuales que utilizan.

A finales del siglo XIX se fundó dicha empresa y actualmente cuenta con un equipo formado por más de 700 profesionales multidisciplinares, repartidos por un amplia zona geográfica. Desde aquel entonces, hasta hace pocos años, esta empresa y muchas otras empresas de este sector, mandaban operarios una vez al mes a leer y anotar en una libreta la

lectura de los contadores, casa por casa. Actualmente, con la llegada de los Smartphones y la repercusión que han tenido en la sociedad, esta empresa ha optado por dar un paso al frente y desarrollar un sistema con una arquitectura orientada a servicios, una aplicación web y una aplicación móvil, mediante la cual los operarios introducen la lectura del contador y envían al instante la lectura a los sistemas centrales de la empresa.

Como se ha visto en la asignatura *Arquitecturas Orientadas a Servicios*, este tipo de arquitectura permite tener centralizada la información de una empresa y facilita la integración con otros sistemas, entre muchas otras ventajas. En este caso ha sido un acierto ya que con este sistema han conseguido centralizar la información de las distintas oficinas repartidas por diferentes localidades y ha permitido desarrollar diferentes aplicaciones, integradas en el sistema, que facilitan la gestión y acceso a dicha información.

A pesar del avance que esto supone, aún sigue siendo ineficiente, pues en grandes ciudades se necesita una gran cantidad de operarios para poder leer todos los contadores repartidos por la ciudad, lo cual supone un gran gasto de dinero. Además, al ser un proceso costoso, solo se realiza una vez al mes, por tanto, si surge alguna anomalía, como podría ser una fuga, podría tardarse hasta un mes en ser consciente de ello.

En lo referente a cifras económicas y localidades no se ha proporcionado ningún dato, pues la empresa a insistido en la confidencialidad de los mismos. No obstante, se puede puntualizar que la adopción de nuevas tecnologías, por parte de la empresa, ha devenido en un crecimiento económico considerable.

A raíz de la anterior investigación y conocimientos en materias como la computación ubicua, plataformas para el Internet de las cosas, arquitecturas orientados a servicios, *Software As A Service*, patrones de diseño y arquitecturas software, obtenidos durante el presente año en las asignaturas *Computación Ubicua*, *Arquitecturas orientadas a servicios* y *Arquitecturas para sistemas software*, se ha decidido desarrollar una solución vertical basada en el Internet de las cosas integrada con una plataforma horizontal para el desarrollo de aplicaciones de este tipo, que permita mejorar las carencias mencionadas anteriormente en las empresas del abastecimiento del agua.

La plataforma horizontal seleccionada para desarrollar la solución es IoTsens, ya que proporciona un API REST pública que permite almacenar, gestionar y analizar información de sensores, conociendo lo que ocurre en tiempo real y pudiendo actuar de forma inmediata. Otros de los motivos por los que se ha seleccionado esta plataforma es, como se ha mencionado anteriormente, la contribución de la empresa al desarrollo de este proyecto, proporcionado acceso a sensores capaces de medir el consumo de agua, previamente instalados en contadores reales.

Gracias a esta plataforma, se puede obtener datos sobre consumo de agua y derivados de este. De esta manera, la solución a desarrollar constituirá un sistema de información a

través del cual poder administrar la información de los distintos contadores, consultar el consumo de estos, facturar y tomar decisiones inmediatas y precisas a través de la medición y tratamiento de datos sobre consumo de agua en tiempo real.

Por tanto, la aplicación debe permitir dar de alta y gestionar en el sistema contadores, así como asociarlos a pólizas de clientes y sensores. Además, una vez proporcionado el acceso a los sensores colocados físicamente en los contadores de agua, debe poderse realizar una monitorización de los datos y obtener estadísticas y datos relevantes para la compañía mediante la representación de estos datos en una gráfica. Además, debe poder ver si los sensores han generado alguna alarma, ya sea de inactividad, manipulación, etc.

1.4 Objetivos del proyecto

El principal objetivo del proyecto es diseñar y desarrollar una aplicación web vertical para el sector del abastecimiento de agua sobre la plataforma horizontal IoTsens. Esta aplicación debe que permita a los trabajadores de este tipo de empresas la visualización de datos relevantes como puede ser sobre el consumo de agua, alarmas, así como la gestión de contadores, en tiempo real.

Mediante este nuevo sistema se pretende:

- Dar de alta y gestionar contadores.
- Poder buscar y visualizar información de contadores
- Ver la evolución de la lectura de los contadores.
- Consultar el histórico del consumo de contadores.
- Tener un panel que muestre información relevante en tiempo real.
- Consultar alarmas generadas por los contadores.
- Ver en un mapa los contadores.
- Ver en un mapa de calor las zonas donde más se consume.

Este nuevo tipo de solución debe ser el camino básico a seguir los próximos años por cualquier empresa que requiera de soluciones IoT, sustituyendo las actuales propuestas ad-hoc impuestas como soluciones a medida para este tipo de proyectos. Al igual que pasó hace años con las arquitecturas orientadas a servicios y el *Software As A Service*, estas nuevas plataformas horizontales deben seguir el mismo camino convirtiéndose en algo que

podría denominarse como *"IoT As A Service"*, de esta manera, sectores verticales tendrán mas facilidades para desarrollar este tipo de soluciones e integrarlas con sus actuales sistemas de información, puesto que no sería necesario implementar ni preocuparse de toda la lógica, comunicación, persistencia y demás tareas correspondiente a la plataforma IoT.

Para desarrollar esta solución, se ha realizado un proceso íntegro de ingeniería de software que a continuación se detallará. En primer lugar, se expondrá la fase de recopilación de requisitos y análisis; en segundo lugar, se describirá la arquitectura, y el diseño a nivel de componentes; finalmente, la fase de implementación y pruebas.

1.5 Estructura del documento

El contenido de este documento está dividido en siete capítulos: Introducción, Plataformas para el IoT, Planificación del proyecto, Entorno tecnológico, Desarrollo de la aplicación, Validación y verificación y Conclusiones y trabajo futuro.

A lo largo de la Introducción se pretende ofrecer una visión general sobre algunos conceptos del Internet de las cosas, para el comprender el contexto del trabajo descrito en el documento, así como la plataforma IoTsens sobre la cual se va a desarrollar dicho proyecto.

El segundo capítulo describe las plataformas horizontales del Internet de las cosas y presenta varias de ellas estudiadas, previamente, en la asignatura de *Computación Ubicua* durante el presente curso. Finalmente se describe con profundidad las características de la plataforma IoTsens así como sus ventajas.

En el tercer capítulo, se presenta la planificación que se ha realizado para el desarrollo del proyecto. Se describe la metodología utilizada y las tareas en las que se ha desglosado, así como la estimación de coste temporal.

En el cuarto capítulo, se detallan algunas de las tecnologías más relevantes que se han utilizado para la implementación y desarrollo del proyecto.

En el Desarrollo de la aplicación se recoge toda la parte de análisis y diseño e implementación de proyecto. Se presentan los casos de uso, requisitos funcionales y de datos, la arquitectura de la solución, diagramas de clases y los patrones de diseño utilizados en el desarrollo, muchos de ellos aprendidos durante el presente curso en la asignatura *Arquitecturas para sistemas software*.

Una vez presentado el Desarrollo de la aplicación se presenta el apartado de Validación y verificación, donde se recogen las distintas pruebas que se han realizado para asegurar el buen funcionamiento del producto desarrollado.

En el último capitulo se recogen las conclusiones y reflexiones alcanzadas tras la finalización del proyecto y se exponen algunas funcionalidades o enfoques que no hayan podido implementarse y puedan ser implementadas en un futuro.

2. Plataformas para el IOT

En el presente capítulo se muestra una visión general sobre algunas de las actuales plataformas horizontales para el Internet de las cosas, como funcionan, cual suele ser su arquitectura, así como las características, ventajas y desventajas de estas.

Como se ha dicho anteriormente IoTsens es la plataforma escogida para el desarrollo de esta solución. A continuación, se va a tratar esta plataforma en detalle para poder conocer mejor como funciona, como está compuesta y que ventajas ofrece.

2.1 Plataformas horizontales para el Internet de las cosas

Durante el transcurso del presente curso, en la asignatura *Computación Ubicua* se han estudiado las plataformas horizontales para el Internet de las cosas y trabajado con una de ellas. Entre las estudiadas destacan Kaa, Macchhina.io, Iotivity y SiteWhere.

Todas las plataformas mencionadas anteriormente presentan ventajas e inconvenientes, no obstante todas comparten una arquitectura parecida y varias similitudes más. De entre todas estas, las más destacables son:

- Todas tienen un punto de entrada para medidas y mensajes generados por sensores. La gran mayoría soportan protocolos para intercambio de mensajes como MQTT (Message Queue Telemetry Transport), STOMP (Simple Text Oriented Message Protocol), además de otro tipo de protocolos como HTTP REST.

- Almacenan los mensajes en bases de datos distribuidas NoSQL enfocadas al BigData. Entre este tipo bases de datos se pueden destacar algunas como MongoDb,

Elasticsearch, Hadoop o Spark.

- La gran mayoría dispone de una aplicación Admin o plataforma web que permite realizar varias operaciones genéricas y de configuración sobre la plataforma.

- Todas proporcionan un API REST mediante el cual poder comunicarse y acceder a sus recursos. Por norma general para acceder a estos recursos de se debe identificar de alguna manera, ya sea en la cabecera de la petición, Basic Auth, etc.

- Proporcionan mecanismos para la comunicación bidireccional con los sensores.

A continuación, se muestra la arquitectura de una de las plataformas mencionadas anteriormente y de esta manera tener una mejor visión de su estructura y componentes.

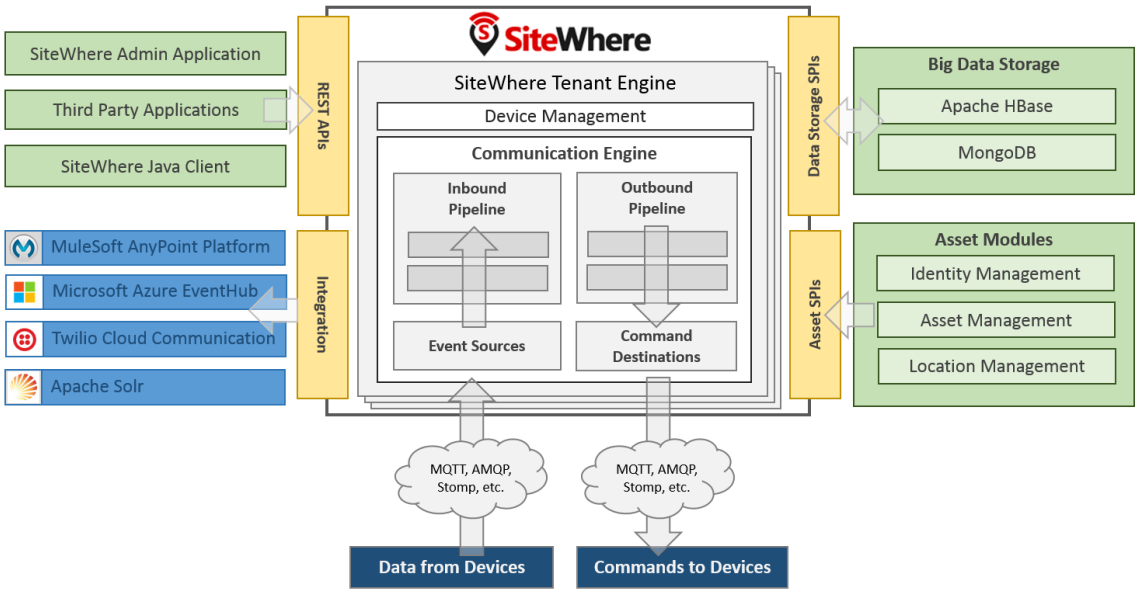


Figura 2.1: Arquitectura de la plataforma SiteWhere

Las diferencias que presentan estas plataformas suelen ser el lenguaje de programación en el que están diseñadas o soportan, ser de código abierto o no, proporcionar clientes en distintos lenguajes para su uso, funcionalidades que no poseen las otras plataformas, etc.

2.2 IoTsens

Como se ha dicho anteriormente, IoTsens es una plataforma horizontal para el Internet de las cosas fundada en España a principios de 2014 con el objetivo de proporcionar soluciones escalables e interoperables basadas en el Internet de las cosas. Se ha escogido

esta plataforma frente a las demás, pues parece mucho más completa en términos generales respecto a las mencionadas anteriormente, sobretodo en funcionalidades como las operaciones sobre medidas y definición de alarmas.

IoTsens no es una plataforma abierta y dispone de varios modelos de negocio: Como plataforma *Software As A Service* o instalación *in house* de la plataforma. A pesar de ello, IoTsens proporciona a estudiantes y universidades acceso gratuito a su plataforma y datos de algunos de sus sensores (en caso de no disponer de sensores propios). De esta manera estudiantes y universidades pueden pueden realizar pruebas o investigaciones, lo cual ha sido un motivo más de los que han hecho decantarse por la plataforma.

A continuación, se va a dar una breve descripción de los componentes y funcionalidades más importantes o con más relevancia en el desarrollo de este proyecto, para poder tener una mejor comprensión de esta plataforma y como interactúa con la solución.

2.2.1 Aplicación Web

IoTsens incluye una aplicación web que se puede utilizar para administrar el funcionamiento del sistema. Para acceder a esta aplicación es necesario iniciar sesión con los credenciales proporcionados al solicitar acceso a la plataforma.

Figura 2.2: Aplicación web IoTsens

Desde esta aplicación se puede consultar y administrar los dispositivos, usuarios, alarmas, permisos, medidas y muchos más componentes de los que dispone el sistema.

2.2.2 Funcionalidades

La arquitectura IoTSens sigue un estilo de microservicios; por lo tanto, la funcionalidad principal es proporcionada por un conjunto de servicios REST que proporcionan pequeñas piezas de la funcionalidad. Estos interactúan entre si para realizar las operaciones necesarias y devolver al API pública el resultado de la acción solicitada por el cliente. De todas las funcionalidades se pueden destacar las siguientes, ya sea por su uso en el desarrollo del proyecto o para futuras mejoras del mismo:

- **Configuración de sensores:** Permite gestionar los sensores, incluyendo crear y borrar, definir sus variables, propiedades, etc.

- **Autorización:** A pesar que en el desarrollo de este prototipo la autorización se gestiona desde la propia aplicación, IoTsens proporciona un sistema de autorización que permite la administración de los sensores que pueden acceder a cada usuario o aplicación cliente y la funcionalidad que puede ejecutar (ver, modificar, etc.).

- **Consulta de medidas:** Permite consultar los datos que son enviados por los sensores, además de realizar operaciones sobre estos como son obtener la suma, máximo, mínimo o media de las medidas dentro de un rango.

- **Consulta y definición de Alarmas:** Permite definir y consultar alarmas generadas por los sensores. Estas se definen y generan sobre las medidas enviadas del sensor para cada variable.

2.2.3 APIs públicas

IoTSens proporciona tres mecanismos para interactuar con el sistema desde fuentes externas. Estos mecanismos pueden utilizarse para la construcción de nuevas aplicaciones o herramientas verticales basadas en la plataforma.

- **API REST pública:** El API REST pública proporciona una fachada para el API interna, por lo que la mayoría de la funcionalidad interna se publica para su uso para clientes externos. El API pública agrega una capa de autenticación con HMAC u OAuth2 y otras funcionalidades como estrategias de mitigación ante ataques DDoS. El API se define utilizando el estándar OpenAPI para facilitar la documentación y el uso del cliente en diferentes lenguajes de programación. Además, está documentada utilizando la tecnología Swagger.

- **Callbacks externos:** El API pública proporciona funcionalidad para consultar la plataforma, pero en muchos sistemas IoT es muy importante trabajar con la información en el mismo instante que es recibida o generada para proporcionar oportunamente las acciones necesarias. IoTSens apoyan la definición de Callbacks

REST para medidas y eventos, por lo que los Callbacks de los sistema externos serán llamados con los nuevos datos tan pronto como se reciban o generen. Siguiendo este enfoque, una sistema externo recibirá datos de los sensores a los que haya suscrito sus callbacks tanto en llamadas Http o en colas MQTT para su propia consumo y procesamiento.

- **API REST INBOX:** Este API está diseñado para enviar nuevos mensajes a la plataforma desde fuentes que no utilizan los mecanismos MQTT predeterminados, que puede ser el caso de dispositivos de baja capacidad o sistemas fuera de la red de sensores. Funciona como una pasarela REST que reenvía los mensajes recibidos a las colas MQTT internas para su decodificación y procesamiento.

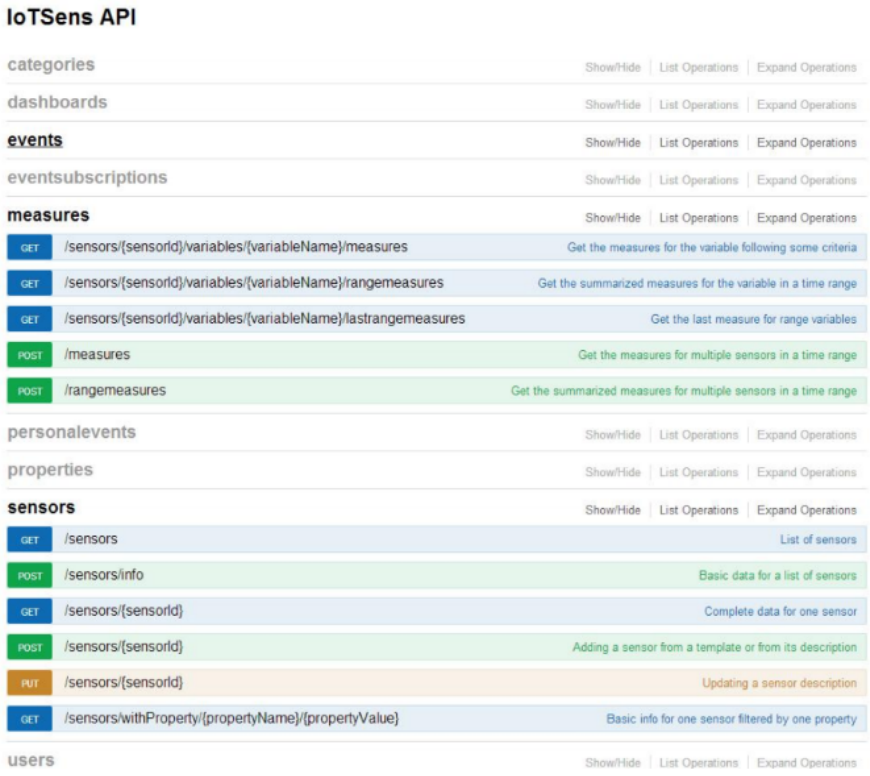


Figura 2.3: Documentación del API con Swagger

2.2.4 Almacenamiento de medidas

El almacenamiento de medidas de IoTSens está bastante desacoplado del resto del sistema. Las medidas son insertadas por un microproceso dentro de un pipeline de procesos que se comunican entre si mediante MQTT, para decodificar, procesar y almacenar los mensajes.

Para soportar el gran volumen de medidas, alarmas y eventos producidas por los sensores, IoTsens dispone de una base de datos NoSQL enfocada al BigData, en concreto ElasticSearch. Esta ofrece capacidades de alto rendimiento y clusterización, ya que permite la agrupación y el equilibrado automático, de modo que se pueden añadir nuevos nodos dinámicamente para aumentar el rendimiento y la fiabilidad del sistema.

3. Planificación del proyecto

En el presente capítulo se muestra una visión general de la etapa de planificación del proyecto. Primero de todo, se describe la metodología escogida, seguida de la planificación temporal.

Esta etapa es muy importante en el desarrollo de un proyecto software para conseguir una mejor definición del proyecto de mediante el desglose en tareas más pequeñas y la estimación los costes temporales de cada una de ellas.

3.1 Metodología de trabajo

Para el desarrollo de este proyecto se ha considerado seguir el uso de la metodología ágil de desarrollo de productos software SCRUM. Se ha tomado esta decisión por los siguientes motivos:

- Efectividad y flexibilidad ante cambios de requisitos.
- Metodología basada en iteraciones de una a cuatro semanas.
- Al final de cada iteración se dispone de un producto funcional.
- Evitar problemas como retrasos de tiempo y complejidad.

A pesar de que la metodología SCRUM está pensada para equipos de entre tres y ocho personas, se ha elegido para el desarrollo de este proyecto (de una única persona), ya que permite detectar y reaccionar con efectividad y flexibilidad ante cambios de requisitos

originados por dificultades o errores en el desarrollo.

En la siguiente figura se ilustra el ciclo de vida de SCRUM a través de un diagrama típico, donde las diferentes partes se detallan a continuación:

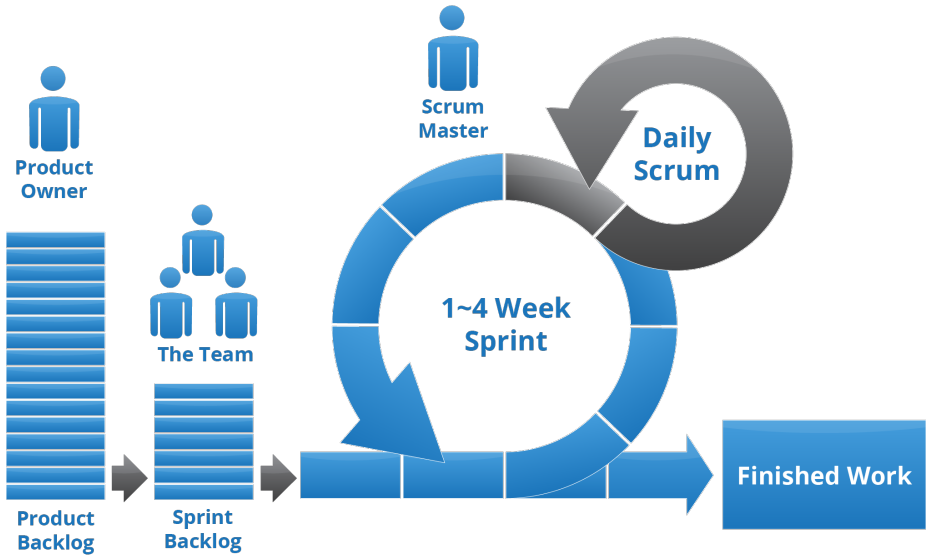


Figura 3.1: Ciclo de vida de Scrum

- **Product Backlog** es la pila de requisitos que debe implementar el sistema. Se compone de funcionalidades y tareas que debe realizar el programador que está en constante evolución durante todo el ciclo de vida, hasta el cierre del sistema.

- **Sprint Backlog** es la pila de tareas que se van a desarrollar en el Sprint actual. Para extraer estas tareas del Product Backlog se pueden llevar a cabo técnicas de priorización de tareas, según las necesidades del usuario y la complejidad de estas. Para llevar el seguimiento del estado de estas tareas, se ha utilizado la técnica Kanban, que se describe posteriormente.

- **Sprint** es la iteración propiamente dicha, que dura entre una y cuatro semanas. En esta iteración, los programadores desarrollan las tareas definidas en el Sprint Backlog y, al finalizar uno de estos, el resultado es un producto funcional que el cliente es capaz de ejecutar y tiene un valor añadido respecto a la iteración.

Además, al inicio y fin de cada iteración se realizan reuniones de la entrega y construcción del Backlog, e inspección del incremento integrado en cada Sprint. La metodología también define que se han de realizar reuniones diarias de 15 minutos aproximadamente para identificar problemas y obstáculos para resolverlos lo antes posible.

El equipo de desarrollo de SCRUM está formado por los siguientes roles:

- **Product Owner:** Decide la funcionalidad del producto y es el responsable de priorizar las tareas a desarrollar en cada iteración. Representa el usuario del sistema.

- **Scrum Master:** Motiva y coordina al equipo y es responsable de detectar los problemas que pueden surgir durante el proceso.

- **Team Scrum:** Crean el producto en sí y son un equipo multidisciplinar de programadores, testers, analistas, etcétera. Todos los miembros del equipo de desarrollo han de conocer con detalle la visión del Product Owner y han de colaborar regularmente y de manera directa con este.

Como se ha mencionado anteriormente, este proyecto se ha basado en la metodología ágil SCRUM, pero ha sido desarrollado por un solo usuario adoptando todos los roles. A parte de ello, en cada iteración también se ha realizado un proceso de ingeniería de software clásico; el apartado de desarrollo de la aplicación se divide en cuatro fases bien diferenciadas:

- **Análisis de requisitos** En esta fase, se han estudiado las necesidades que pueden tener los clientes de este sector y que pueden esperar que el sistema ofrezca. También, se han identificado los posibles actores que pueden estar involucrados así como qué rol van a tomar en el sistema. Como resultado, se obtiene un diagrama de casos de uso, una recopilación de requisitos de usuario formalizada y una serie de prototipos sencillos de la interfaz de usuario.

- **Diseño e implementación** En esta fase tiene lugar el diseño de la arquitectura, a cual ofrece una visión externa del sistema, con todos los componentes que lo forman, a alto nivel. Además, se realiza el diseño a nivel de componentes, diseño de clases, identificación de patrones de diseño, etc. Finalmente, consultando los prototipos obtenidos en la fase anterior, se obtiene el diseño real de la interfaz de usuario y se lleva a cabo la construcción propiamente dicha del sistema.

- **Validación y verificación** Antes de lanzar el sistema a producción es necesario probarlo en distintos escenarios para asegurar su correcto funcionamiento. Se han llevado a cabo pruebas unitarias y pruebas de usuario para asegurar el correcto funcionamiento y calidad del sistema.

- **Despliegue o implantación** En esta fase, el producto es lanzado en un entorno de producción. En este caso, se ha subido a un servidor de aplicaciones, como es Heroku donde también se han realizado pruebas para detectar y reportar errores.

Como se ha dicho anteriormente, en este proyecto se han puesto en práctica

metodologías ágiles, por tanto, el desarrollo de estas cuatro fases no ha sido lineal. Para cada iteración se ha llevado a cabo un pequeño proceso de ciclo de vida clásico formado por estas fases. De esta forma, al finalizar cada Sprint, se obtiene una parte funcional del sistema que se puede validar.

3.2 Planificación temporal

En este apartado se muestran las tareas a realizar para llevar a cabo este proyecto. Debido a que no se ha podido disponer de gran cantidad de tiempo libre para realizar el proyecto debido a motivos laborales, se han realizado las siguientes estimaciones, aplicando el juicio por experiencia en el desarrollo de proyectos similares.

Como se ha mencionado anteriormente este proyecto se ha desarrollado siguiendo la metodología ágil Scrum. En la siguiente tabla se muestran algunas de las historias de usuario que se han definido para el desarrollo del proyecto.

Identificador	Puntos de historia	Historia de usuario
HU01	6	Como usuario quiero iniciar sesión en la aplicación con mi usuario
HU03	6	Como usuario quiero poder ver en un listado los contadores de agua presentes en el sistema
HU05	3	Como usuario quiero poder ver en un mapa los contadores de agua presentes en el sistema
HU06	3	Como usuario quiero poder ver información de un contador (identificador y consumo) al pasar el ratón por encima de un marcador del mapa
HU10	6	Como usuario quiero poder agregar las medidas en días, semanas, meses y años, y, poder realizar operaciones sobre ellas como sumar, obtener el máximo, mínimo o media
HU14	6	Como usuario quiero poder ver en un listado las alarmas generadas por los contadores

Table 3.1: Definición de algunas de las historias de usuario

La Figura 3.2 representa un diagrama de Gantt, donde se muestra la duración y la disposición en el tiempo de cada una de las fases de análisis y desarrollo del proyecto. Estas tareas son las típicas de un desarrollo ciclo de vida clásico. En cambio, el proyecto se ha desarrollado mediante la metodología SCRUM. En cada Sprint se han llevado a cabo un análisis, diseño e implementación de los requisitos de usuario que se pretendía desarrollar en ese periodo.

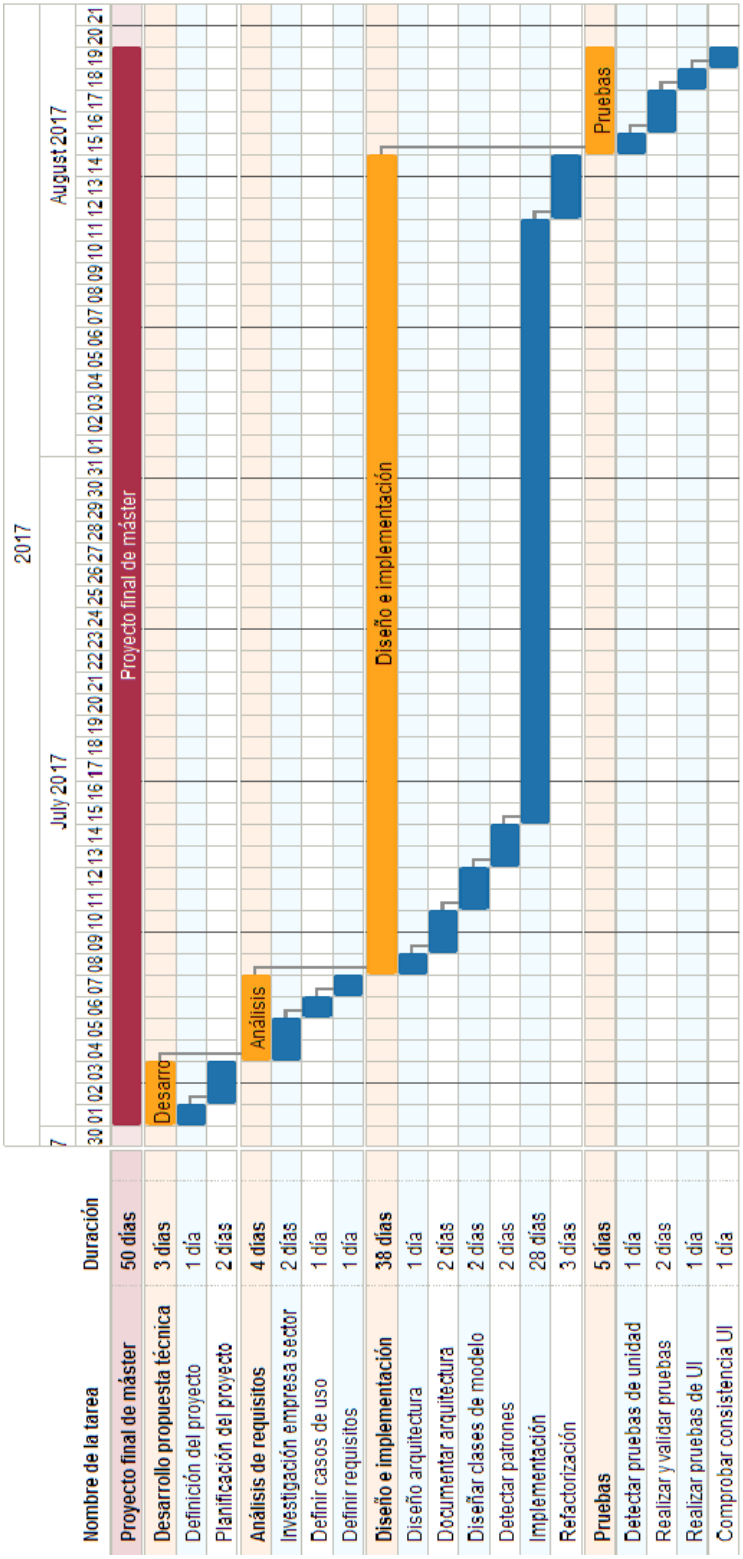


Figura 3.2: Diagrama de Gantt

Se han necesitado un total de cinco Sprints para desarrollar el producto, uno cada semana. Además, la pila del producto no ha estado cerrada completamente desde el principio, sino que se han ido añadiendo y cambiando algunas historias de usuario a medida que se avanzaba.

4. Entorno tecnológico

En el presente capítulo se muestran las tecnologías utilizadas durante el desarrollo del proyecto. Primero de todo, se va a ofrecer una visión detallada del entorno de desarrollo utilizado para el desarrollo de este proyecto. Seguidamente, se van a describir las tecnologías más relevantes en el desarrollo del proyecto y cómo se han utilizado en él.

A la hora de desarrollar un proyecto, es muy importante dedicar cierto tiempo a investigar y conocer las diversas tecnologías existentes para desarrollo de proyectos similares al que se pretende desarrollar. Una mala elección de tecnologías puede conllevar al fracaso de un proyecto, al igual que una buena elección puede conducir al éxito.

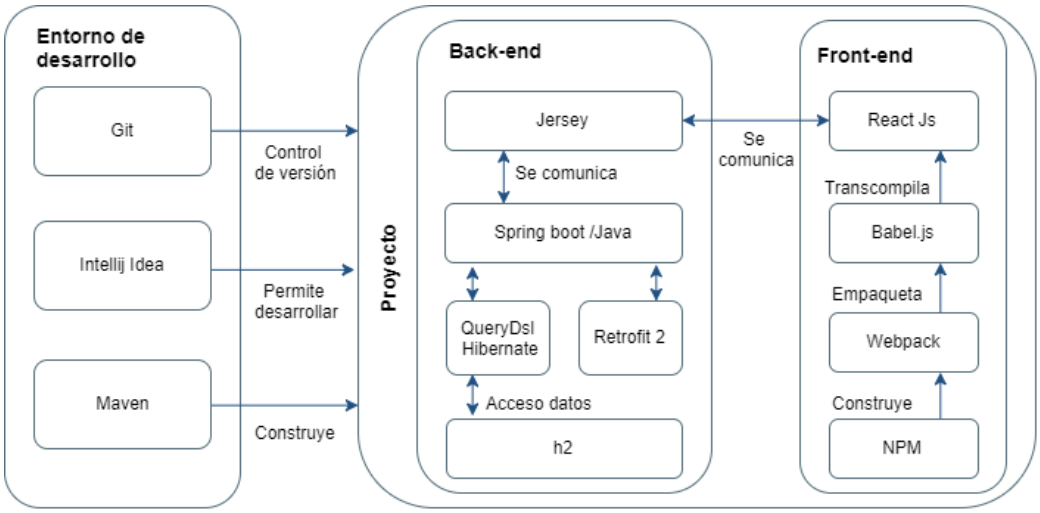


Figura 4.1: Entorno tecnológico

4.1 Entorno de desarrollo

Para desarrollar la aplicación, se ha trabajado sobre una máquina con sistema operativo Windows 10, la cual cuenta con un procesador multi-núcleo Intel Core i5-3337U a una velocidad de 1.8GHz y 6GB de memoria RAM.

4.1.1 IntelliJ Idea

Como IDE para el desarrollo de la aplicación, se ha utilizado IntelliJ IDEA en su versión 15. Este potente entorno de trabajo ofrece soporte a Java y gran cantidad de ventajas que favorecen considerablemente la productividad como las siguientes:

- Autocompletado de código inteligente que agiliza el desarrollo.
- Integración con sistemas de control de versiones como Git, el cual ha sido utilizado en el proyecto.
- Gran cantidad de plugins que ofrecen funcionalidades que pueden resultar de ayuda al desarrollo.
- Soporte de herramientas de gestión y construcción de proyectos como Maven.
- Sensación de fiabilidad y robustez muy superior respecto a otros entornos.
- Constante análisis estático del código que avisa de errores y mejoras en este.
- Potente herramienta de refactorización.

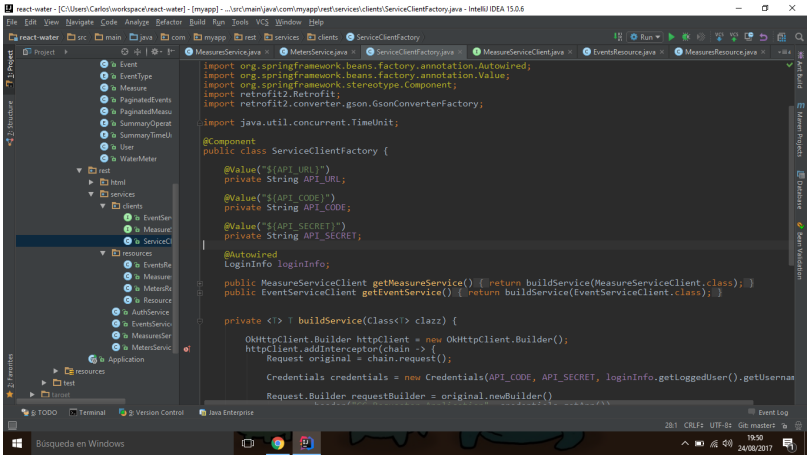


Figura 4.2: IDE: IntelliJ Idea

4.1.2 Git

Para llevar a cabo el control de versiones de la aplicación se ha utilizado Git, un software de control de versiones diseñado por Linus Torvalds en 2005. Este tipo de tecnología permite alojar el código fuente de proyecto en una máquina remota, evitando pérdidas de información en caso de surgir algún incidente en la máquina local de desarrollo, y conservar un registro histórico de los cambios realizados durante el desarrollo del proyecto. Además, permite compartir el código fuente y recursos que componen el proyecto entre varios desarrolladores, y tener versiones actualizadas en todo momento.

Figura 4.3: Logo de Git

En caso de trabajar varias personas en el proyecto, para evitar conflictos en cambios en el código, Git permite crear ramas paralelas al desarrollo donde cada uno de los participantes desarrollan nuevas funcionalidades que posteriormente se unirán a la rama principal, con supervisión de algún desarrollador si es necesario, evitando así conflictos en el código.

4.1.3 Apache Maven

Maven es una herramienta para la gestión y la construcción de proyectos software escritos en lenguaje Java,creada por Jason van Zyl, de Sonatype, en 2002. Esta herramienta facilita algunos procesos en el desarrollo como la resolución de dependencias o el nombrado de versiones. Utiliza un fichero XML llamado Project Object Model (POM), alojado en el directorio raíz del proyecto, para describir el proyecto de software a construir, sus dependencias de otros módulos, componentes y plugins externos, como pueden ser JUnit, Hibernate, QueryDSL, etc.

Figura 4.4: Logo de Maven

Esta herramienta se ha utilizado en este proyecto para importar y gestionar librerías java que facilitan y ayudan en el desarrollo del proyecto como JPA/Hibernate y QueryDSL. Además, también se han importado plugins como el cliente de Heroku, para desplegar la aplicación en sus servidores o *frontend-maven-plugin* que permite construir y ejecutar la aplicación front-end al mismo tiempo que lo hace el back-end.

4.2 Tecnologías web en el back-end

En esta sección se van a describir las tecnologías, más relevantes, utilizadas en la parte servidor del proyecto.

4.2.1 Java

Para el desarrollo de este proyecto, en la parte del servidor, se ha utilizado Java. Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems. En 2012, se convirtió en uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones web, con unos 10 millones de usuarios reportados. Se define como un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible.

Con este lenguaje se pretende que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo, tal y como dice el axioma de Java, "write once, run anywhere". Para ello, se compila el código fuente escrito en lenguaje Java y se genera un código conocido como “bytecode”, que son básicamente instrucciones máquina simplificadas específicas de la plataforma Java.

Los "bytecode" generados se ejecutan en la máquina virtual (JVM), un programa escrito en código nativo de la plataforma destino (que es el que entiende su hardware), que interpreta y ejecuta el código. Además, se suministran bibliotecas adicionales para acceder a las características de cada dispositivo (como los gráficos, ejecución mediante hebras o threads, la interfaz de red) de forma unificada.

Figura 4.5: Logo de Java

Se ha escogido este lenguaje principalmente por los conocimientos previos y experiencia con el. Como se ha dicho anteriormente, es uno de los lenguajes de programación más populares para aplicaciones web, por lo que al desarrollar con este lenguaje de programcación, se dispone de una gran comunidad y recursos que ayudan al desarrollo. Además, al estar pensado para que se pueda ejecutar en en cualquier dispositivo que tenga la máquina virtual de Java instalada, ofrece un amplio abanico de posibilidades para su despliegue.

4.2.2 Spring boot

Spring Boot es un sub-proyecto del conocido framework para el desarrollo de aplicaciones y contenedor de inyección de dependencias de código abierto para la plataforma Java, Spring.

El objetivo de Spring boot es facilitar y simplificar tanto como sea posible, la creación de proyectos con el framework Spring, eliminando la necesidad de crear los largos archivos de configuración XML presentes en las aplicaciones Spring. Spring Boot provee configuraciones por defecto para Spring y otra gran cantidad de librerías, además, provee un modelo de programación parecido a las aplicaciones java tradicionales que se inician en el método main.

Figura 4.6: Logo Spring boot

Se ha decidido utilizar Spring boot para el desarrollo de este proyecto pues ofrece todas las ventajas del framework Spring, como son su filosofía de programar orientado a interfaces, la Inyección de Dependencias o Inversión de Control, programación orientada a Aspectos y el hecho de contar con gran cantidad de módulos, cada uno de los cuales ofrece una serie de bibliotecas o funcionalidades, como pueden ser JDBC, JPA, Hibernate, etc. Además, permite crear aplicaciones standalone o despliegues WAR, lo cual facilita en gran medida el despliegue de la aplicación en otros sistemas.

4.2.3 Jersey

Jersey es una librería o framework de código abierto desarrollado por Sun, que implementa JAX-RS, un API del lenguaje de programación Java que proporciona soporte en la creación de servicios web de acuerdo con el estilo arquitectónico Representational State Transfer (REST).

Jersey es una pieza muy importante en el desarrollo de este proyecto, ya que es el que permite la comunicación y transferencia de datos desde la aplicación cliente al servidor. Mediante anotaciones se crean end-points que en conjunto crean un API REST, la cual es llamada por la aplicación front-end para solicitar datos o enviarlos en caso de querer modificar o almacenar nuevos datos.

4.2.4 Retrofit 2

Retrofit 2 es un cliente HTTP para Android y Java, desarrollado por Square. Este cliente tiene la virtud de ser muy sencillo de utilizar, pues permite traducir un API REST en

interfaces Java, sin necesidad de implementar gran cantidad de código para tener un cliente.

A continuación, se muestra un cliente desarrollado en el proyecto, para poder apreciar como de sencillo es traducir un end-point REST a una interfaz Java.

```
1
2  public interface MeasureServiceClient {
3
4      @FormUrlEncoded
5      @POST("rangemeasures")
6      Call<PaginatedMeasures> searchMeasures(
7          @Field("sensorId") List<String> sensorsIds,
8          @Field("variableName") String variableName,
9          @Field("from") String from,
10         @Field("until") String until,
11         @Field("limit") Integer limit,
12         @Field("rangeUnit") SummaryTimeUnit rangeUnit,
13         @Field("unit") Integer unit,
14         @Field("summaryOperation") SummaryOperation
15         summaryOperation );
16
17 }
```

Del ejemplo anterior se pueden destacar tres cosas:

- La anotación @POST que indica el tipo de petición que se va a realizar y a que URI a partir de la URL base que se define en las propiedades de la aplicación.
- Los parámteros @Field del método que se corresponden a los parámetros que espera el API.
- El hecho de que no sea necesario implementar nada de código, ya que la librería lo genera a partir de la especificación de la interfaz.

Como la mayoría del software de código abierto, Retrofit está construido sobre OkHttp otro cliente Http del mismo desarrollador. Sin embargo, Retrofit no tiene un convertidor JSON integrado para convertir de objetos JSON a Java, pero en su lugar se ha utilizado la librería Gson para ello, ya que es compatible.

En este proyecto, Retrofit 2 se ha utilizado para realizar comunicaciones y compartir datos con la plataforma IoTsens. IoTsens dispone de un cliente en Github para el lenguaje Java que permite interactuar con su plataforma, sin embargo el cliente es muy simple, ya que se encuentra en desarrollo y no permite realizar varias de las peticiones que se han

utilizado en este proyecto. De ahí la necesidad de crear un cliente a parte.

4.2.5 H2

Puesto que el desarrollo de este proyecto es un prototipo para poder realizar pruebas y comprobar las mejoras que supone respecto al punto de partida de la investigación, se ha utilizado una base de datos embebida el back-end para la persistencia de datos.

Figura 4.7: Logo de H2

Esta base de datos es H2, una base de datos programada en Java, que puede ser incorporada en aplicaciones Java o ejecutarse de modo cliente-servidor. Una de las características más importantes de H2 es que se puede integrar completamente en aplicaciones Java y acceder a la base de datos lanzando SQL directamente, sin tener que pasar por una conexión a través de sockets.

4.2.6 QueryDSL + Hibernate/JPA

Como se ha mencionado anteriormente el SGBD (Sistema gestor de bases de datos) que se ha utilizado en el proyecto es H2 embebido dentro de la aplicación. Para acceder a los datos de la base de datos se han realizado consultas utilizando QueryDSL junto a Hibernate.

Por un lado, Hibernate es una herramienta de Mapeo objeto-relacional (ORM) para la plataforma Java. Implementa el estándar JPA, que es parte de esta plataforma y esto facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante anotaciones en los beans de las entidades que permiten establecer estas relaciones.

Por otro lado QueryDSL es un librería de código abierto que permite construir consultas type-safe muy similares a Sql. QueryDSL se puede utilizar con JPA y por consiguiente con Hibernate.

Se han utilizado estas dos tecnologías en el proyecto junto al patrón DAO, ya que facilitan y agilizan en gran medida el acceso a datos y la construcción de consultas. A continuación, se muestra un ejemplo de una entidad mapeada con JPA/Hibernate y una

consulta utilizando QueryDSL.

```
@Entity
@Table(name = "user")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    Long id;

    @Column
    String name;

    @Column
    String pswd;

    @Column
    String username;

        ...
}
```

```
@Repository
public class UserDAO {


    @PersistenceContext
    private EntityManager entityManager;

    QUser qUser = QUser.user;

    public List<User> getUsers() {
        return new JPAQuery(entityManager).from(qUser)
                .list(qUser);
    }

    public void addUser(User user) {
        entityManager.persist(user);
    }
}
```

4.3 Tecnologías web en el front-end

En esta sección se van a describir las tecnologías, más relevantes, utilizadas en la parte cliente del proyecto.

4.3.1 React js

React js es una librería Javascript creada por Facebook en base a unas necesidades generadas por el propio desarrollo de la web de esta popular red social. Es software libre y a partir de su liberación una creciente comunidad de desarrolladores ha decidido adoptar esta tecnología.

Esta librería permite construir interfaces de usuario para crear aplicaciones SPA(single page application) más eficientes, gracias a la creación de componentes interactivos y reutilizables. Mediante la creación de estos componentes, se puede desarrollar aplicaciones web de una manera más ordenada y con menos código que el resultante de utilizar Javascript puro o librerías como jQuery centradas en la manipulación del DOM.

Figura 4.8: Logo de React Js

Una de las ventajas de React Js es que tiene las vistas asociadas a los datos, por lo que no es necesario escribir código para manipular la página cuando los datos cambian. Cuando se cambian los datos de uno de los componentes, este llama a su método Render y actualiza sólo la parte de la vista que ocupa en la pantalla.

En el proyecto se ha utilizado como framework principal para el desarrollo de la parte front-end. Gracias a la creación de componentes, se ha podido reutilizar gran parte del código para crear interfaces similares. Además, se han utilizado componentes externos creados por diferentes desarrolladores, evitando así, tener que desarrollar ciertos componentes, y, agilizando el proceso de creación.

4.3.2 Babe.Js

Babel.js es un transcompilador que permite convertir código de JavaScript ES6 en código de ES5. Se ha utilizado en el proyecto, pues ES6 ofrece nuevas características y ventajas que hacen atractivo trabajar con este lenguaje.

Lamentablemente, hoy en día la gran parte de los navegadores y distintas plataformas de JavaScript no soportan completamente ES6. Este es uno de los motivos por el que se ha

decidido utilizar Babel.Js, ya que permite aprovechar todas las ventajas y novedades de ES6 sin tener problemas de compatibilidad ya que es posteriormente convertido a ES5.

4.3.3 Webpack

Webpack es un empaquetador de módulos muy eficiente, hecho específicamente para desarrollo de grandes aplicaciones front-end contienen mucho código JavaScript.

Soporta módulos descargados tanto desde NPM y gracias a su increíble sistema de loaders es posible extenderlo para poder usarlo con CSS, HTML, y más. Además, una vez empaquetado permite ofuscar el código para evitar su manipulación.

Figura 4.9: Logo de Webpack

Webpack se utiliza en el proyecto en el momento de construcción y compilación de la aplicación. Mediante un plugin de Maven se ejecuta un comando que llama a Webpack. Una vez se llama, este empaqueta toda la parte front-end de la aplicación que después es cargada en un fichero index.html estático.

4.3.4 NPM

NPM (node package manager) es un gestor de paquetes javascript de NODE.JS. Al igual que Maven, NPM permite descargar cualquier librería o modulo disponible en forma de dependencia y el nombrado de versiones.

Figura 4.10: Logo de NPM

Al igual que maven tiene un fichero pom.xml, NPM tiene el archivo package.json. En archivo se definen las dependencias necesarias así como otra información del proyecto, como la versión, autor, descripción y script de ejecución entre otras.

En el proyecto se ha utilizado básicamente para gestionar todas las dependencias de la parte front-end.

5. Desarrollo de la aplicación

En los siguientes apartados se va a especificar la recopilación de requisitos de usuario, seguido del análisis del entorno y actores involucrados, diseño a nivel de arquitectura, componentes e interfaz gráfica y, finalmente, la implementación propiamente dicha del software.

5.1 Análisis

En esta fase se realiza un análisis detallado de los requisitos y componentes que constituyen el sistema. Para ello, se exponen los requisitos funcionales del sistema y de datos. Previamente, se mostrará un diagrama de casos de uso para apreciar qué actores realizan qué determinadas acciones sobre el sistema.

A continuación, en la figura 5.1 se va a mostrar una visión global de los elementos del sistema a alto nivel. Por lo tanto, mediante las relaciones de los elementos del sistema, tal como muestra muestra esta representación, se puede entender mejor el funcionamiento de la aplicación y la lógica de negocio que sigue.

5.1.1 Casos de uso

Los casos de uso son una herramienta para el análisis de proyectos software. Haciendo uso de ellos se puede observar gráficamente la interacción entre los actores involucrados y el propio sistema. En este proyecto, se pueden identificar 3 actores: El técnico de la empresa de agua, el usuario particular cliente de la empresa y el sistema de IoTsens. El último no interactúa directamente con la aplicación desarrollada pero es necesario tenerlos en cuenta para comprender el funcionamiento del software que se está analizando.

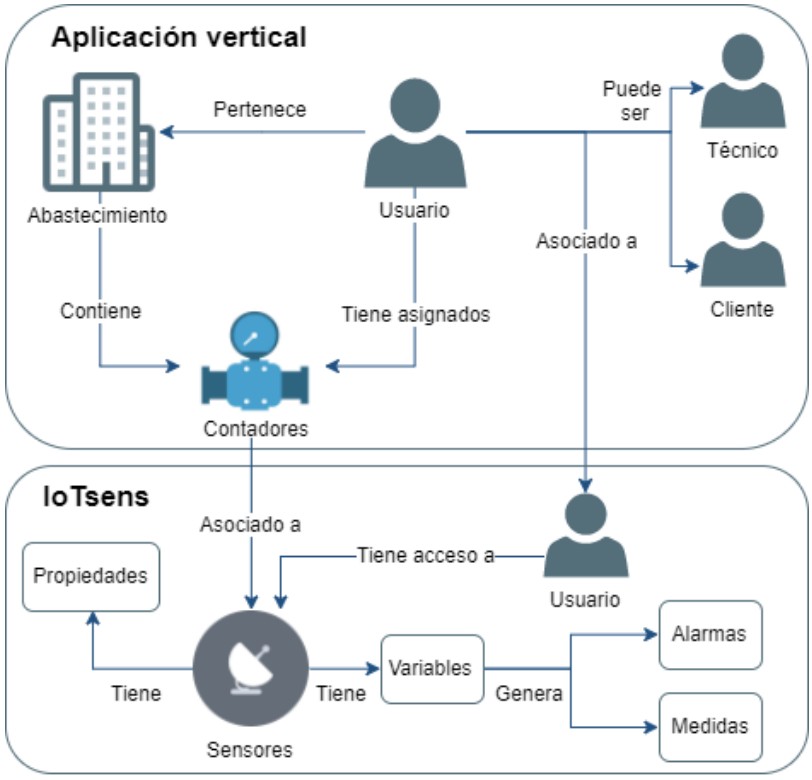


Figura 5.1: Representación de los elementos del sistema a alto nivel

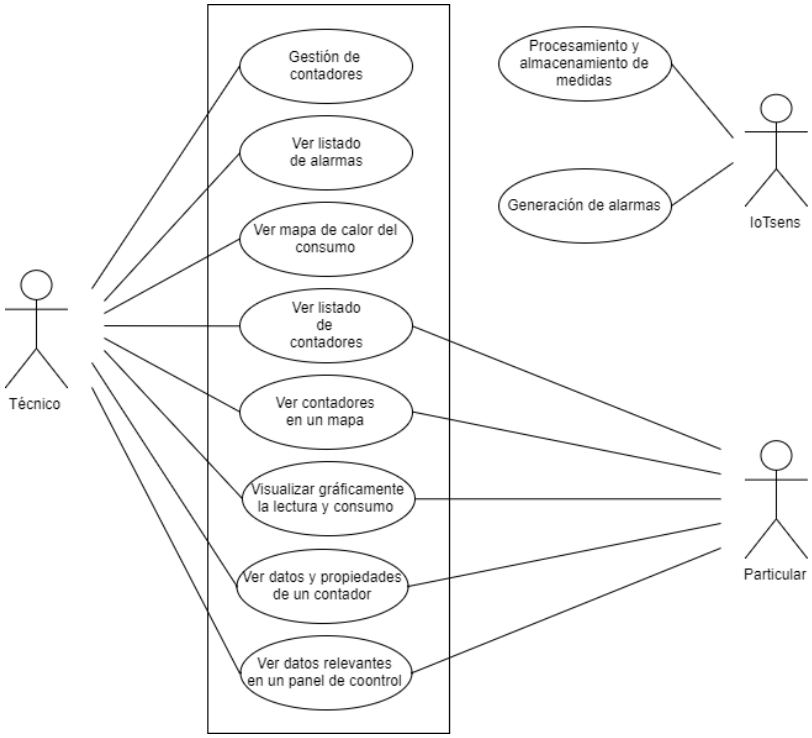


Figura 5.2: Diagrama de casos de uso

Como se puede apreciar, se muestran los tres actores mencionados anteriormente, relacionados con las acciones que realiza cada uno sobre el sistema. El rectángulo que se puede observar representa el alcance del sistema, es decir, aquellas acciones que se realizan dentro del contexto de la aplicación.

5.1.2 Requisitos funcionales

El objetivo de los requisitos funcionales de usuario es definir el alcance del sistema, es decir, las características, a nivel funcional, que debe satisfacer el software. La definición formal de estos requisitos es necesaria para comprender el problema a resolver.

A continuación, se muestra un listado con los requisitos y las tablas que definen estos. Cada requisito contiene un identificador, nombre, descripción y una secuencia de pasos que definen su ciclo de vida.

- **FR01:** Listado de contadores
- **FR02:** Crear contador
- **FR03:** Actualizar contador
- **FR04:** Eliminar contador
- **FR05:** Ver detalles de un contador
- **FR06:** Visualizar gráficamente lectura y consumo
- **FR07:** Ver contadores en un mapa
- **FR08:** Ver consumo de contadores en un mapa de calor
- **FR09:** Ver listado de alarmas
- **FR10:** Ver listado de alarmas de un contador
- **FR11:** Ver datos relevantes en un panel de control

Requisito funcional	
Código	FR01
Nombre	Listado de contadores
Descripción	El sistema debe permitir ver un listado con los contadores registrados
Secuencia normal	
1	El usuario accede a la aplicación
2	El usuario selecciona la pestaña "Water Meters" de la barra de navegación de la aplicación
3	El sistema muestra un listado con los contadores registrados en el sistema

Table 5.1: FR01: Listado de contadores

Requisito funcional	
Código	FR02
Nombre	Crear contador
Descripción	El sistema debe permitir crear un contador nuevo. Al crearse el contador también debe crearse un sensor en IoTsens
Secuencia normal	
1	El usuario accede a la aplicación
2	El usuario selecciona la pestaña "Water meters" de la barra de navegación de la aplicación
3	El usuario selecciona el botón "New water meter"
4	El usuario rellena el formulario con los datos necesarios
5	El usuario selecciona el botón "Save"

Table 5.2: FR02: Crear contador

Requisito funcional	
Código	FR03
Nombre	Actualizar contador
Descripción	El sistema debe permitir editar los datos de un contador.
Secuencia normal	
1	El usuario accede a la aplicación
2	El usuario selecciona la pestaña "Water meters" de la barra de navegación de la aplicación
3	El usuario selecciona el icono con forma de lápiz en un contador
4	El usuario actualiza el formulario con los datos necesarios
5	El usuario selecciona el botón "Save"

Table 5.3: FR03: Actualizar contador

Requisito funcional	
Código	FR04
Nombre	Eliminar contador
Descripción	El sistema debe permitir eliminar los datos de un contador.
Secuencia normal	
1	El usuario accede a la aplicación
2	El usuario selecciona la pestaña "Water meters" de la barra de navegación de la aplicación
3	El usuario selecciona el icono con forma de papelera en un contador
4	El usuario presiona el botón "Delete" de la ventana de confirmación

Table 5.4: FR04: Eliminar contador

Requisito funcional	
Código	FR05
Nombre	Ver detalles de un contador
Descripción	El sistema debe permitir ver los detalles de un contador.
Secuencia normal	
1	El usuario accede a la aplicación
2	El usuario selecciona la pestaña "Water meters" de la barra de navegación de la aplicación
3	El usuario selecciona el icono con forma de papelera en un contador
4	El sistema muestra una ventana con detalles y propiedades de un contador

Table 5.5: FR05: Ver detalles de un contador

Requisito funcional	
Código	FR06
Nombre	Visualizar gráficamente lectura y consumo
Descripción	El sistema debe permitir visualizar la lectura y consumo de un contador a través de una gráfica.
Secuencia normal	
1	El usuario accede a la aplicación
2	El usuario selecciona la pestaña "Water meters" de la barra de navegación de la aplicación
3	El usuario selecciona un contador
4	El usuario selecciona la pestaña "Measurements"
5	El sistema muestra una ventana con un filtro y una gráfica

Table 5.6: FR06: Visualizar gráficamente lectura y consumo

Requisito funcional	
Código	FR07
Nombre	Ver contadores en un mapa
Descripción	El sistema debe permitir visualizar los contadores en un mapa. Además, al pulsar sobre un marcador del mapa, debe aparecer información sobre el y su consumo del día anterior
Secuencia normal	
1	El usuario accede a la aplicación
2	El usuario selecciona la pestaña "Map" de la barra de navegación de la aplicación
3	El sistema muestra un mapa con los contadores
4	El sistema muestra un listado con las alarmas generadas por el sistema

Table 5.7: FR07: Ver contadores en un mapa

Requisito funcional	
Código	FR08
Nombre	Ver consumo de contadores en un mapa de calor
Descripción	El sistema debe permitir visualizar los contadores en un mapa. Además, al pulsar sobre un marcador del mapa, debe aparecer información sobre el y su consumo del día anterior
Secuencia normal	
1	El usuario accede a la aplicación
2	El usuario selecciona la pestaña "Map" de la barra de navegación de la aplicación
3	El usuario presiona el botón "Heat map"
4	El sistema muestra un mapa de calor con los consumos del día anterior

Table 5.8: FR08: Ver consumo de contadores en un mapa de calor

Requisito funcional	
Código	FR09
Nombre	Ver listado de alarmas
Descripción	El sistema debe permitir visualizar las alarmas generadas por el sistema. Además, debe poderse filtrar por sus campos
Secuencia normal	
1	El usuario accede a la aplicación
2	El usuario selecciona la pestaña "Events" de la barra de navegación de la aplicación
3	El sistema muestra un listado con las alarmas generadas por el sistema

Table 5.9: FR09: Ver listado de alarmas

Requisito funcional	
Código	FR10
Nombre	Ver listado de alarmas de un contador
Descripción	El sistema debe permitir visualizar las alarmas de un contador concreto
Secuencia normal	
1	El usuario accede a la aplicación
2	El usuario selecciona la pestaña "Water meters" de la barra de navegación de la aplicación
3	El usuario selecciona un contador
4	El usuario selecciona la pestaña "Events"
5	El sistema muestra un listado con las alarmas generadas por el sistema para el contador seleccionado

Table 5.10: FR10: Ver listado de alarmas de un contador

Requisito funcional	
Código	FR11
Nombre	Ver datos relevantes en un panel de control
Descripción	El sistema debe permitir visualizar datos relevantes como el numero de contadores, alarmas activas, consumo total del día anterior, consumo el actual mes, etc.
Secuencia normal	
1	El usuario accede a la aplicación
2	El usuario selecciona la pestaña "Dashboard" de la barra de navegación de la aplicación
3	El sistema visualiza un panel de control con datos relevantes

Table 5.11: FR11: Ver datos relevantes en un panel de control

5.1.3 Requisitos de datos

Para satisfacer los requisitos expuestos anteriormente, es necesario definir las entidades y atributos que se van a almacenar, es decir, los requisitos de datos. Las entidades como las medidas o alarmas no se se consideran, ya que no se persisten, sino que sirven para la compartir datos entre ambos sistemas como DTOs (Data Transfer Objects).

A continuación, se muestra una lista con las entidades,las tablas que las representan y los propiedades que contendrán cada una de estas.

- **DR01:** Abastecimiento
- **DR02:** Usuario
- **DR03:** Contador
- **DR04:** Abastecimiento_Usuario

- **DR05:** Abastecimiento_Contador
- **DR06:** Usuario_Contador

Requisito de datos	
Código	DR01
Nombre	Abastecimiento
Datos	Un abastecimiento representa una zona geográfica acotada. Un técnico de la empresa del agua, puede ver todos los contadores de agua contenidos dentro del abastecimiento al que se le ha asignado. Por tanto, para cada abastecimiento, se necesita almacenar un código de abastecimiento y su nombre.
Comentarios	Cuando se crea un nuevo contador, se añade al abastecimiento desde el cual ha sido creado.

Table 5.12: DR01: Abastecimiento

Requisito de datos	
Código	DR02
Nombre	Usuario
Datos	Para cada usuario se almacena un código único de usuario, la contraseña, su nombre y el rol en el sistema.
Comentarios	-

Table 5.13: DR02: Usuario

Requisito de datos	
Código	DR03
Nombre	Contador
Datos	Para cada contador se almacena un identificador único, un identificador del sensor en IoTsens, la dirección, descripción, latitud, longitud y número de póliza asociada. Los datos referentes a las variables se almacenan en IoTsens y si es necesario se accede a ellos tanto para leer como para editar datos.
Comentarios	Cuando se crea o elimina un contador, se debe crear o eliminar el sensor con el identificador de IoTsens.

Table 5.14: DR03: Contador

Requisito de datos	
Código	DR04
Nombre	Abastecimiento_Usuario
Datos	Establece la relación que hay entre un abastecimiento y sus usuarios asociados. Se almacena el Identificador de ambos
Comentarios	-

Table 5.15: DR04: Abastecimiento_Usuario

Requisito de datos	
Código	DR05
Nombre	Abastecimiento_Contador
Datos	Establece la relación que hay entre un abastecimiento y los contadores asociados. Se almacena el Identificador de ambos
Comentarios	Sólo el perfil técnico puede acceder a los contadores de un abastecimiento

Table 5.16: DR05: Abastecimiento_Contador

Requisito de datos	
Código	DR06
Nombre	Usuario_Contador
Datos	Establece la relación que hay entre un usuario y sus contadores asociados. Se almacena el Identificador de ambos
Comentarios	Un usuario particular sólo puede acceder a los contadores que él tenga asociados.

Table 5.17: DR06: Usuario_Contador

5.1.4 Requisitos de software y hardware

La aplicación que se describe en este proyecto ha sido probada en varios ordenadores con sistema operativo Windows 10 y Ubuntu 14. Los requisitos para que un sistema pueda desplegar la aplicación son los siguientes:

- Un procesador Multi-Core como puede ser Dual-Core, i3, i5, etc.
- Tener JVM instalado.
- Tener Node y NPM instalado.
- Conexión a Internet para exponer la aplicación, comunicar con IoTsens, bases de datos externas, etc.
- Almacenamiento interno mínimo para almacenar la base de datos local. La base de datos también puede alojarse en otro sistema distribuido.

5.2 Diseño e implementación

Una vez terminada la fase de análisis, se procede a modelar el sistema a nivel de arquitectura y componentes de software. La etapa de diseño permite a los desarrolladores tener una idea más clara y concreta de cómo será el sistema, reduciendo la abstracción respecto a la fase anterior.

5.2.1 Arquitectura

El primer paso en la fase de diseño es definir la arquitectura del sistema a alto nivel. La arquitectura es una especificación de la estructura a nivel global del sistema. Se centra en aspectos de más alto nivel que los algoritmos, funciones o tipos de datos. Estos últimos forman parte del diseño a nivel de componentes.

En este caso, la arquitectura presente se puede definir como cliente/servidor ya que tenemos una aplicación vertical (cliente) que interactúa con la plataforma IoTsens (Servidor).Además, la comunicación que hay desde el front-end al back-end y este a la plataforma sigue el patrón de diseño Back-end For Front-end, ya que el front-end no ataca directamente contra el API REST de IoTsens, sino que el back-end define su propio API REST dedicado a la aplicación y delega sobre si mismo muchas tareas.

Una vez definida la arquitectura a alto nivel, que engloba todos los elementos más importantes del sistema, se define la arquitectura de la aplicación. Con la definición de la arquitectura de la aplicación, se puede obtener una aproximación del comportamiento y composición de esta.

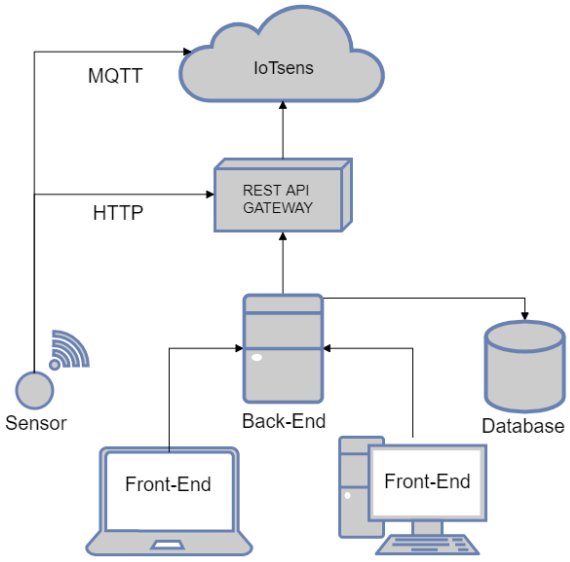


Figura 5.3: Diagrama de la arquitectura

En la siguiente figura se puede observar la arquitectura de la aplicación.

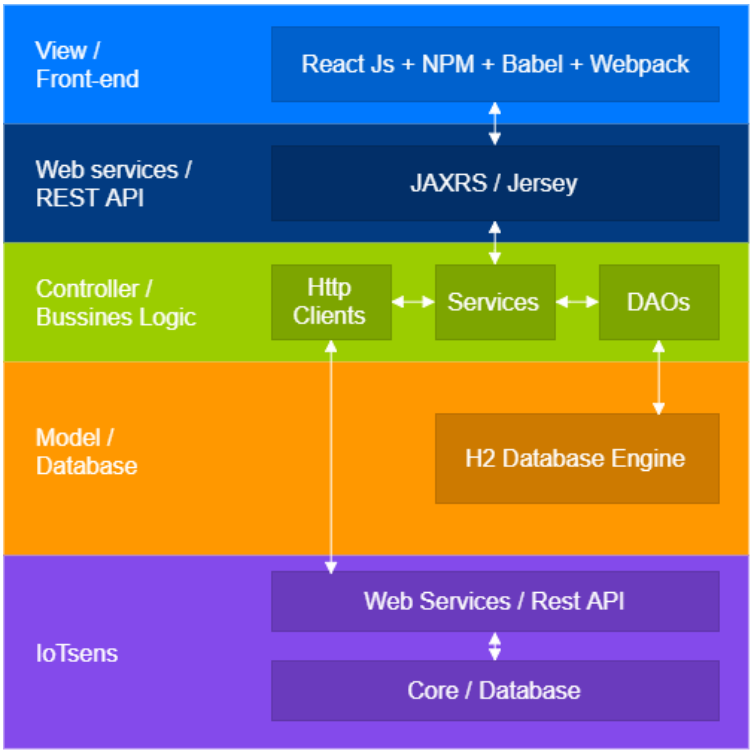


Figura 5.4: Diagrama del la arquitectura de la aplicación

Como se puede observar, la aplicación se divide en varias capas que interactúan entre sí.

- Front-end o vista es la capa con la que interactúa el usuario. Está compuesta por una aplicación en React JS y varias tecnologías web.
- Servicios web en forma de API REST que permiten la comunicación entre la vista y el servidor. Comunica con los controladores para realizar las operaciones necesarias.
- La capa de controladores se encarga de toda la lógica de negocio, clientes http y persistencia. Está desarrollada utilizando Spring boot, Hibernate, Retrofit, etc.
- Modelo o base de datos contiene el sistema gestor de bases de datos H2.
- IoTsens es la plataforma horizontal para IOT con la que interactúa el sistema.

5.2.2 Arquitectura IoTsens

Para poder comprender mejor la arquitectura total de todos los elementos que participan en el proyecto desarrollado, es necesario presentar brevemente la arquitectura que sostiene toda la plataforma IoTsens.

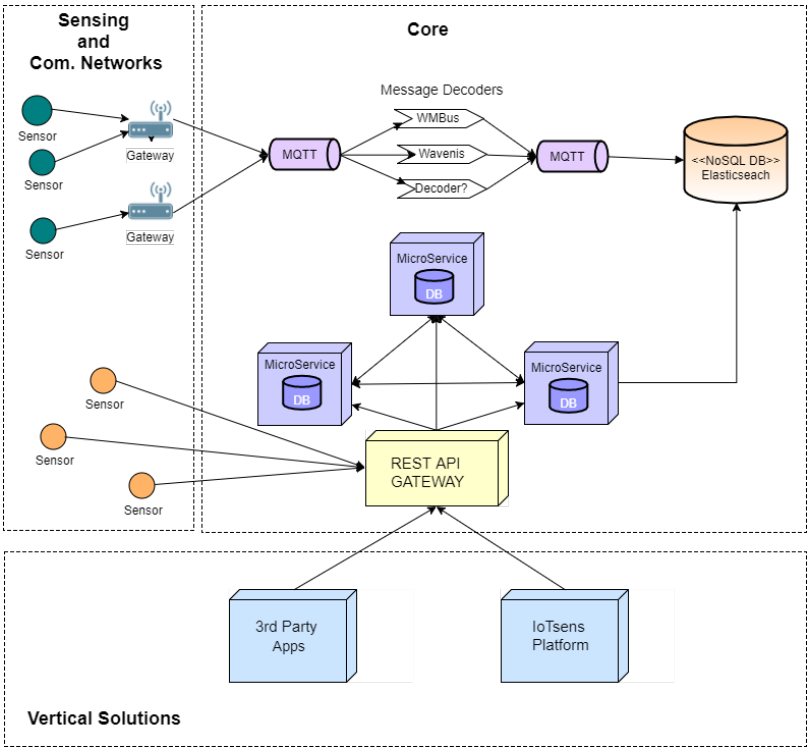


Figura 5.5: Diagrama de la arquitectura de IoTsens

5.2.3 Patrones de diseño

A medida que se ha desarrollado el proyecto, ha aumentando en complejidad. Como se ha visto durante el presente curso en la asignatura *Arquitectura para sistemas Software*, los patrones de diseño pretenden identificar diferentes problemas a los que se enfrenta la ingeniería del software en repetidas ocasiones y proveer una solución específica para cada uno de ellos de una manera limpia y elegante. Por tanto cualquier desarrollador que se enfrente a un problema que se haya repetido en infinidad de casos, probablemente pueda acudir a un patrón de diseño que lo solucione, adaptándolo a su contexto y de esta manera solucionar de la manera más elegante posible este problema.

Gracias a la experiencia previa en el desarrollo de aplicaciones utilizando lenguajes de programación orientada a objetos, se ha podido podido anticipar a algunos de los problemas más comunes, mediante el uso de patrones de diseño.

5.2.3.1 Patrón MVC (Modelo Vista Controlador)

El patrón de diseño o arquitectónico Modelo Vista Controlador es el patrón que engloba toda la aplicación. Todos los componentes del sistema, se pueden ubicar dentro de uno de estos tres módulos. El objetivo de este patrón es separar la lógica de negocio, de la interfaz de usuario y los datos del modelo. A continuación, se definen los tres módulos que componen este patrón:

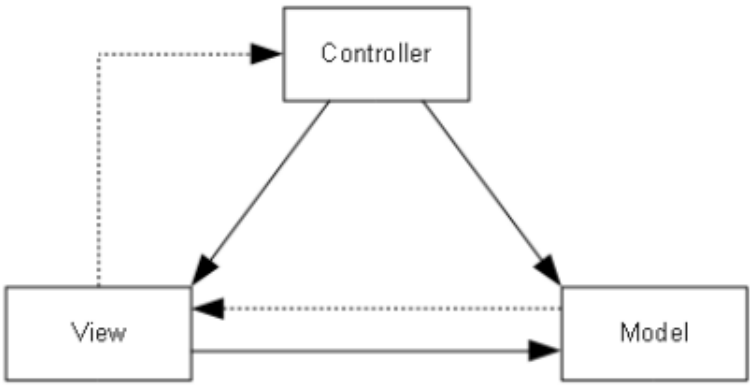


Figura 5.6: Diagrama del patrón Modelo Vista Controlador

- **Modelo:** Este componente tiene que ver con las representaciones de la información que va a ser utilizada en el sistema. Un componente del modelo en el sistema es la clase Estudio, ya que es una representación de una entidad de información. La base de datos también se encuentra dentro de este componente.

- **Vista:** Es el componente con el que el usuario interactúa: la interfaz de usuario o GUI. En una implementación pura de MVC, la vista no tiene ningún estado y no

realiza acciones, sino que las delega en el controlador.

- **Controlador:** Este componente es el encargado de gestionar la lógica de la aplicación, responder a eventos, peticiones de la vista por parte del usuario, etc. En este proyecto, el controlador se encarga de realizar las acciones solicitadas por la vista, además de observar constantemente el estado de la transferencia para reflejarlo la vista.

  El uso de este patrón ha permitido afrontar de una manera más sencilla los constantes cambios de la interfaz durante el desarrollo del proyecto. Gracias a tener la lógica separada de la vista se han podido realizar cambios en la interfaz sin necesidad de cambiar la lógica de la aplicación, puesto que solo se necesita modificar la representación de la información, no su tratamiento. Además, toda la información de la vista se mantiene actualizada sin necesidad de que el usuario lo solicite manualmente.

5.2.3.2 Patrón DAO

Como se ha visto anteriormente en la arquitectura, para el acceso a datos se han utilizado objetos DAO. Data Access Object (DAO) es un patrón de diseño que proporciona una interfaz para acceder a los datos, ya sean en base de datos, un fichero o cualquier otro medio de almacenamiento. Este patrón oculta al programador que usa la interfaz la forma en que se almacenan los datos. Esto permite más abstracción entre la capa de aplicación y la capa de persistencia.

En esta aplicación se han desarrollado tres interfaces DAO. Cada una de estas permite el acceso a datos de diferente ámbito. Las interfaces son las siguientes:

- **AbastecimientoDAO:** Mediante esta interfaz se accede a los datos relativos a los abastecimientos.
- **UserDAO:** Mediante esta interfaz se accede a los datos relativos a los usuarios.
- **WatermeterDAO:** Mediante esta interfaz se accede a los datos relativos a los contadores.
- **LoggedUserDAO:** Mediante esta interfaz se gestiona la información del usuario autenticado.

Cada una de estas interfaces tiene una clase concreta que la implementa, y es donde se realiza el acceso a la base de datos local. Una ventaja de esta utilizar este patrón es el desacoplamiento entre la lógica de negocio y la persistencia, ya que en caso de un cambio en la tecnología de persistencia, como podría ser, solo sería necesario modificar los DAOs sin necesidad de cambiar la lógica del resto de la aplicación.

A continuación, se va a mostrar un ejemplo de una clase DAO utilizada en el proyecto.

```
@Repository
public class WaterMeterDAO {

    @PersistenceContext
    private EntityManager entityManager;

    QWaterMeter qWaterMeter = QWaterMeter.waterMeter;

    public void addWaterMeter(WaterMeter meter) {
        entityManager.persist(meter);
    }

    public WaterMeter updateWaterMeter(WaterMeter meter) {
        return entityManager.merge(meter);
    }

    public void deleteWaterMeter(WaterMeter meter) {
        entityManager.remove(meter);
    }

    public WaterMeter getWaterMeterById(String meterId) {
        return new JPAQuery(entityManager)
                .from(qWaterMeter)
                .where(qWaterMeter.meterId.eq(meterId))
                .singleResult(qWaterMeter);
    }
    ...
}
```

5.2.3.3 Inyección de dependencias

El patrón Inyección de dependencias es un patrón de diseño que deriva de un patrón más genérico llamado Inversión de Control. Inyección de dependencias hace uso de la modularidad, la reutilización, y trata de solucionar las necesidades de creación de los objetos de una manera práctica, útil, escalable y con una alta versatilidad del código.

Como se ha comentado en diversas ocasiones en el documento, la aplicación se ha desarrollado utilizando el framework Spring Boot de Java. El patrón Inyección de dependencias es quizás la característica más destacable de este framework y facilita en gran medida el uso de este.

A continuación, se va a mostrar parte del código donde se implementa este patrón. Se

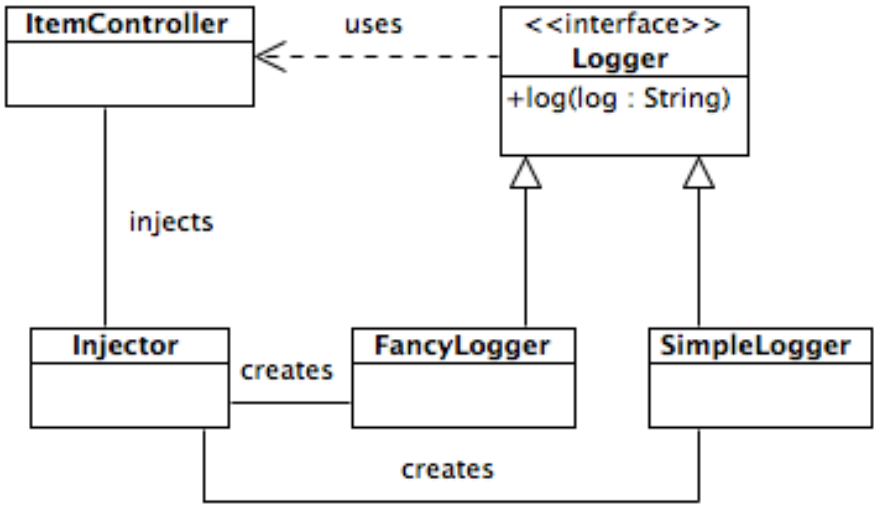


Figura 5.7: Diagrama del patrón Inyección de dependencias

van a omitir partes del código pues no son necesarias para la comprensión del patrón y probablemente las dificulten.

```
@Configuration
public class AppConfig {

    @Value("${API_URL}")
    private String API_URL;

    @Value("${API_CODE}")
    private String API_CODE;

    @Value("${API_SECRET}")
    private String API_SECRET;

    @Bean
    public MeasureServiceClient measureServiceClient() {
        return new MeasureServiceClientFactory()
        .build( API_URL, API_CODE, API_SECRET);
    }
    ...
}
```

```
@Service
public class MeasuresService {
    ...
    @Autowired
    MeasureServiceClient measureServiceClient;
    ...

    public List<Measure> getVariableMeasuresInRange(...){
        return measureServiceClient.searchMeasures(
                variableName,
                fromParam, untilParam,
                MEASURES_RESPONSE_LIMIT,
                aggregationUnit,operation)
                .execute().body();
    }
}
```

En el código anterior se puede observar como la clase AppConfig define una serie de Beans para su posterior inyección. Mediante la anotación @Autowired, Spring busca entre todos los Beans definidos como dependencias para inyectarlas en el atributo anotado con la anterior anotación. De esta manera se consigue recoger la creación de objetos en el AppConfig y separarse del resto de aplicación donde es usada.

5.2.3.4 Patrón Factory

En el ejemplo anterior del patrón Inyección de dependencias, se puede observar como al definir un Bean se ha utilizado una clase Factory. Esto es a causa del uso del patrón Factory para la creación de los diferentes clientes Http para realizar peticiones a la plataforma IoTsens.

El patrón de diseño Factory consiste en utilizar una clase constructora abstracta con unos cuantos métodos definidos y otros abstractos. Los métodos definido suelen tener como objetivo la construcción de las partes genéricas del objeto a construir y llamar a los métodos abstractos, que proporcionan la parte específica de cada tipo diferente de objeto.

En este proyecto se ha utilizado el patrón Factory para construir los distintos clientes http que son posteriormente utilizados para llamar la plataforma IoTsens. A continuación, se va a mostrar parte del código donde se implementa este patrón, para así poder tener una visión de cómo ha sido utilizado.

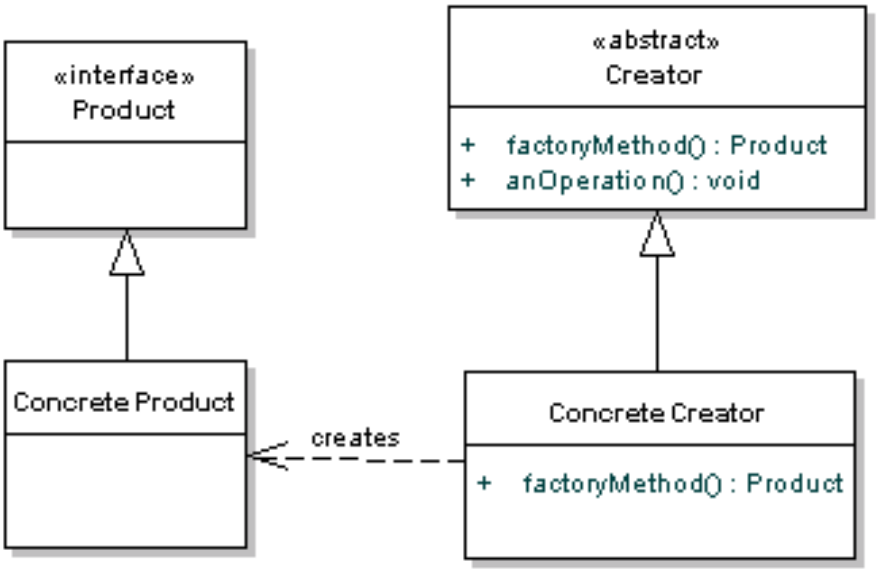


Figura 5.8: Diagrama del patrón Factory

```
1  public abstract class ServiceClientFactory {
2
3      abstract Class getSpecifClientType();
4
5      public <T> T build(String API_URL, String API_CODE,
6      String API_SECRET){
7          return (T) buildService(getSpecifClientType(),
8          API_URL, API_CODE, API_SECRET);
9      };
10
11
12     protected <T> T buildService(Class<T> clazz,
13     String API_URL, String API_CODE, String API_SECRET) {
14
15         OkHttpClient.Builder httpClient = new OkHttpClient
16         .Builder();
17         httpClient.addInterceptor(chain -> {
18             Request original = chain.request();
19
20             Request.Builder requestBuilder = original
21             .newBuilder().header(...);
22            ...
23         });
```

```

        return new Retrofit.Builder()
                .baseUrl(API_URL)
                .client(httpClient.build())
                .build().create(clazz);
    }
}
```

```
public class EventServiceClientFactory
extends ServiceClientFactory {

    @Override
    Class getSpecifClientType() {
        return EventServiceClient.class;
    }
}
```

```
public interface EventServiceClient {

    @FormUrlEncoded
    @POST("events")
    Call<PaginatedEvents> searchEvents(
        @Field("sensorId") List<String> sids,
        @Field("variableId") String v,
        @Field("name") String n,
        @Field("from") String f,@Field("until") String u,
        @Field("limit") Integer l,...
    );
}
```

Como se puede apreciar en las muestras de código anteriores, EventServiceClient corresponde a un producto concreto, ServiceClientFactory es la clase Factory abstracta y EventServiceClientFactory es la clase Factory concreta que extiende a ServiceClientFactory.

Gracias al uso de este patrón, la creación de los clientes realmente fácil como se puede apreciar a continuación. Además, al utilizarlo junto al patrón Inyección de dependencias sólo hace falta crear un bean e inyectarlo en la parte del código donde se vaya a utilizar.

```
    @Bean
    public MeasureServiceClient measureServiceClient() {
        return new MeasureServiceClientFactory()
            .build( API_URL, API_CODE, API_SECRET);
    }
```

5.2.3.5 Patrón Builder

Builder es un patrón de diseño creacional, es decir, su cometido es la creación de instancias de alguna clase de forma desacoplada y transparente.

Para las clase del modelo WaterMeter y Usuario se ha desarrollado otra clase con el mismo nombre pero con terminación Builder. Esta clase tiene un método estático *create* que se encarga de crear una instancia del builder, y otros métodos *with* para definir los atributos concretos de una instancia de la clase que queremos construir. Finalmente, un método *build* genera la instancia en sí con los atributos correspondientes.

A continuación, se muestra un ejemplo de la clase WaterMeterBuilder:

```
1  public class WaterMeterBuilder {
2
3      String meterId;
4      String address;
5      String description;
6      ...
7
8      public static WaterMeterBuilder create () {
9          return new WaterMeterBuilder ();
10     }
11
12     public WaterMeterBuilder withMeterId (String m) {
13         this.meterId = m;
14         return this;
15     }
16     ...
17     public WaterMeter build () {
18         WaterMeter wm = new WaterMeter();
19         wm.setMeterId(this.meterId);
20         wm.setAdrdess(this.address);
21         wm.setDescription(this.description);
22         ...
23         return wm;
24     }
25 }
```

De esta forma, el usuario obtendría una nueva instancia de la clase WaterMeter ejecutado la siguiente instrucción:

```
WaterMeter waterMeter = WaterMeterBuilder.create()
        .withMeterId(meterId)
    .withAddress(address)
    .build();
```

5.3 Capturas de pantalla

Puesto que el proyecto desarrollado consta de una aplicación web, las diferentes interfaces de usuario de ésta se han implementado, como se ha comentado anteriormente, con el fraework React Js junto a los componentes de Bootstrap con el fin de aportar una apariencia más vistosa, dinámica y robusta a la aplicación.

Para diseñar la interfaz se han consideraron los siguientes aspectos:

- **Interfaz familiar:** La mayoría personas pasa gran parte de tiempo visitando páginas web. Por ello se decidió diseñar una interfaz parecida a la de otra aplicación similar a la desarrollada para que le resulte al usuario más familiar la navegación por la aplicación.

- **Consistencia:** La interfaz ha de mantener una consistencia entre sus diferentes ventanas, ya que una interfaz consistente permite al usuario entender mejor como funcionarán las cosas, incrementando su eficiencia.

- **Jerarquía visual:** Se ha diseñado la interfaz de una manera que permite al usuario centrarse en lo más importante a la hora de realizar cada tarea. El tamaño, color y posicionamiento de cada elemento hacen más sencillo entender la interfaz.

- **Proporcionar Feedback:** La interfaz ha de comunicarse con el usuario cuando sus acciones han sido realizadas de manera correcta, incorrecta o anda perdido.

- **Reflejar estado:** En todo momento la interfaz debe reflejar el estado en que se encuentra. No hay que obligar al usuario a recordar.

- **Diseño simple:** Se ha optado por un diseño simple sin muchos elementos innecesarios que distraigan la atención del usuario.

5.3.1 Login

El usuario que vaya a utilizar la aplicación debe estar dado de alta previamente en la base de datos del sistema. Además de sus datos personales, nombre de usuario y contraseña, también tiene que tener asociado uno o varios abastecimientos.

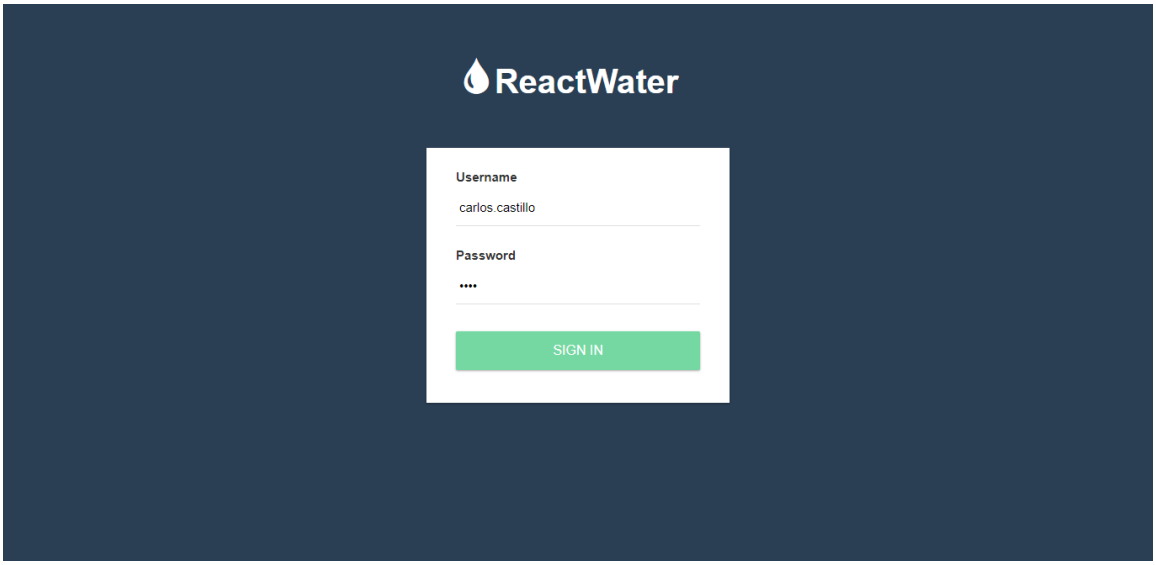


Figura 5.9: Login

5.3.2 Dashboard

Una vez el usuario consigue autenticarse en la plataforma la primera pantalla que visualiza es el Dashboard o panel de control. En esta pantalla se muestran un resumen del estado de los contadores y otros datos de interés como son: Comparación del consumo de los contadores en el mes actual respecto al anterior, número de alarmas activas, numero de sensores, porcentaje de sensores leídos el día anterior, etc.

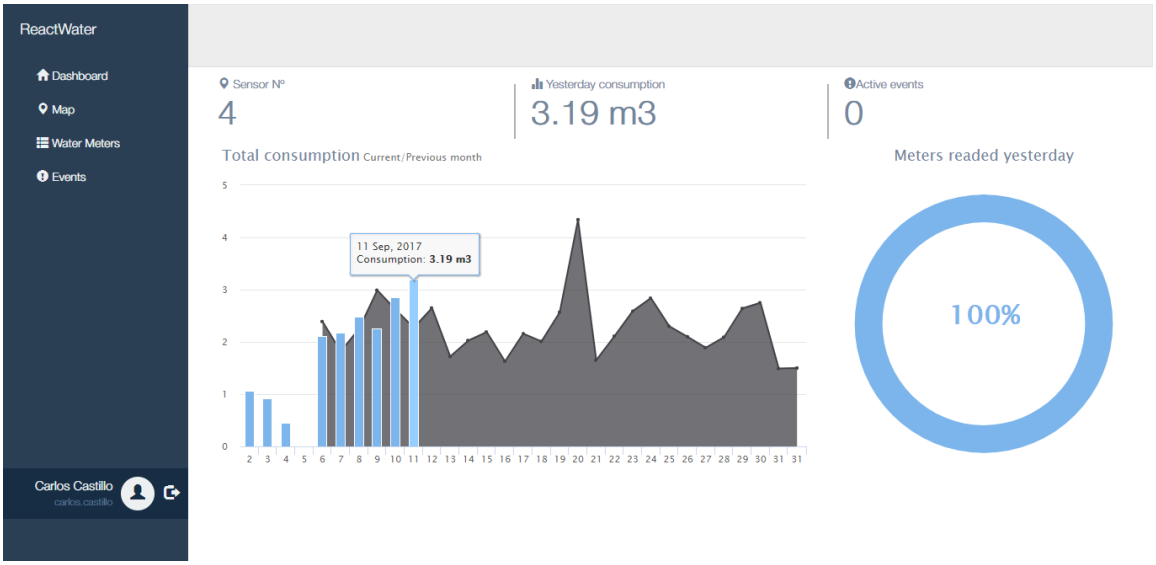


Figura 5.10: Dashboard/Panel de control

5.3.3 Mapa y mapa de calor

Seleccionando en el menú lateral la opción del mapa, el usuario puede navegar hasta esta pantalla donde se visualizan todos los contadores a los que tiene acceso. Pulsando en cualquier contador se abre un Popup con información del contador.

Si el usuario pulsa el icono situado en la parte superior derecha del mapa, se muestra un mapa de calor que representa el consumo de los sensores el día anterior.

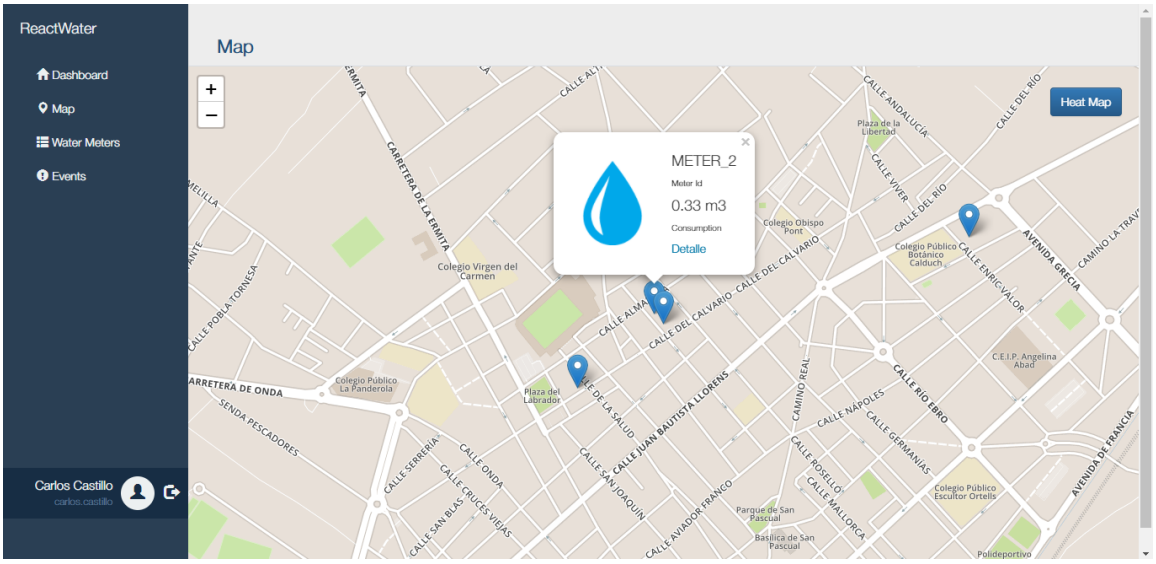


Figura 5.11: Mapa

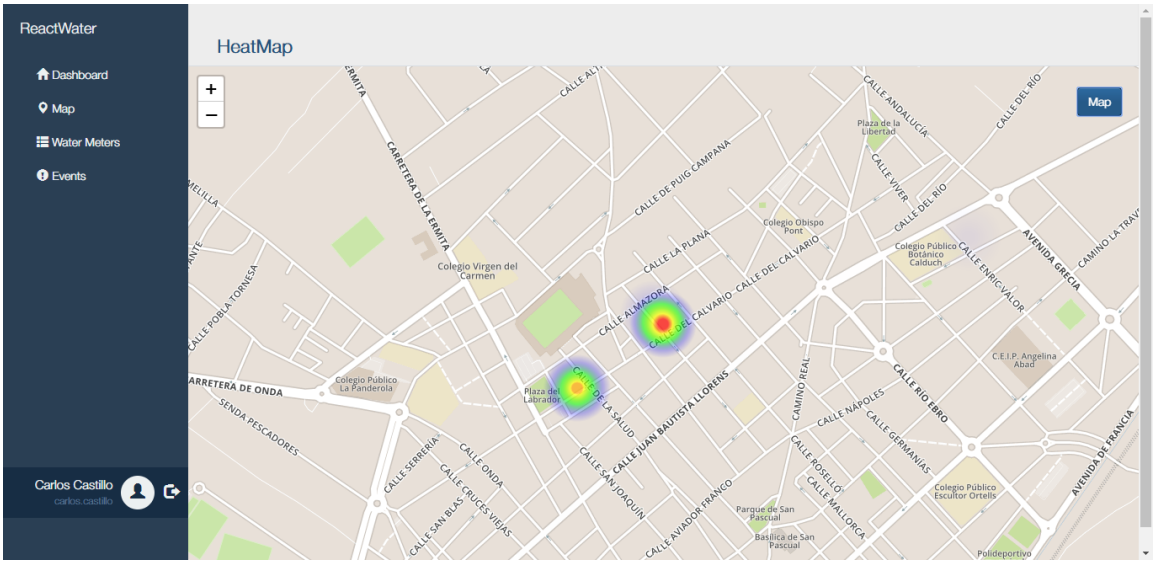


Figura 5.12: Mapa de calor

5.3.4 Listado de contadores

Seleccionando en el menú lateral la opción "Water meters", el usuario puede navegar hasta esta pantalla donde se listan todos los contadores a los que tiene acceso. En la parte superior se dispone de un filtro donde el usuario puede introducir un valor y obtener los contadores cuyo identificador, póliza, dirección o descripción coincida con el valor introducido.

Justo debajo del filtro, situado a la derecha hay un botón que permite a los usuarios tipo técnico dar de alta nuevos contadores. Si se pulsa el botón, redirige al usuario a un formulario mediante el cual se pueden dar de alta contadores.

En la parte derecha de cada uno de los resultados mostrados en la lista, aparecen de uno a tres botones dependiendo del tipo de usuario que esté accediendo a la página. El primero de los botones, es visible para todo tipo de usuarios y sirve para acceder a los detalles de un contador. El segundo y tercero, solo son visibles para el tipo de usuario técnico y permiten editar datos de un sensor y eliminarlo.

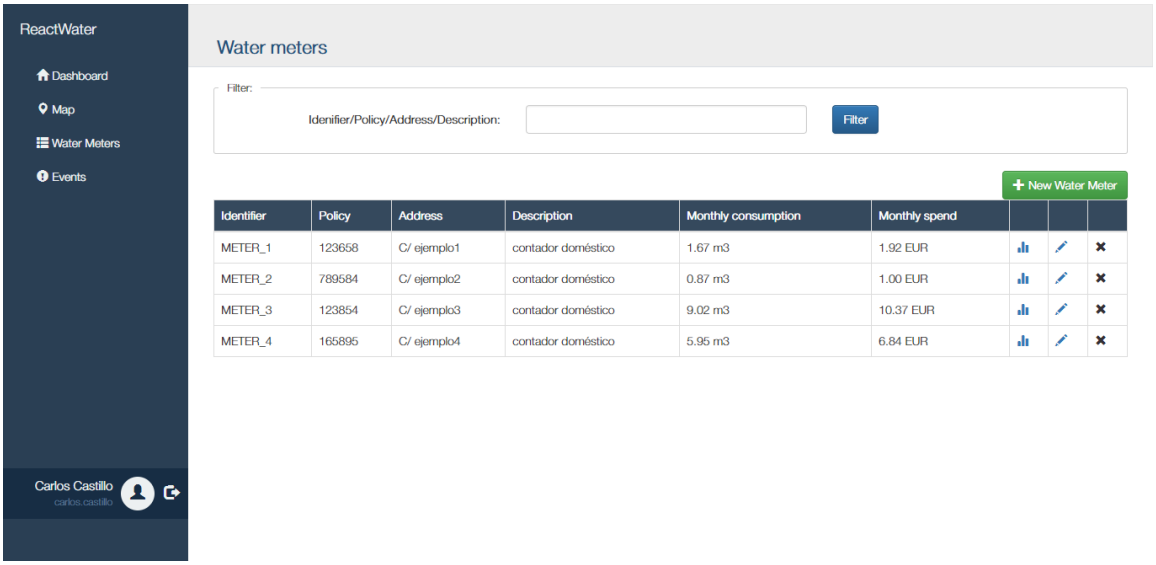


Figura 5.13: Listado de contadores

En caso de pulsar sobre el botón de eliminar un contador, aparece una ventana que requiere de confirmación para poder realizar la operación, como se puede apreciar en la figura 5.13.

5.3.5 Dar de alta/editar contador

Como se ha visto en la pantalla anterior, un usuario de tipo técnico puede crear y editar contadores. Para ello debe pulsar en el botón correspondiente del listado de contadores.

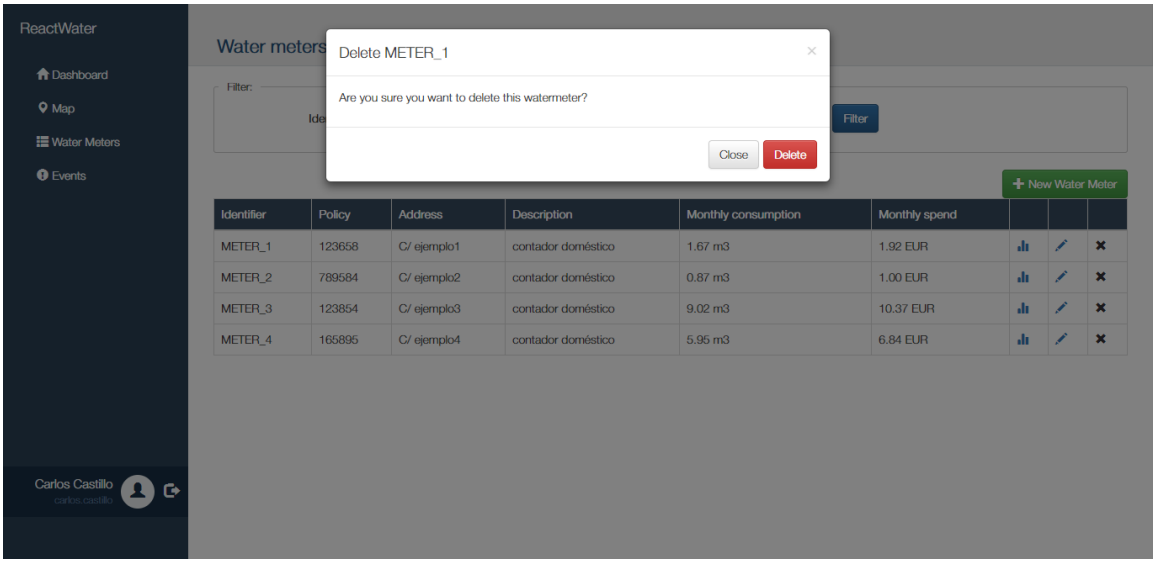


Figura 5.14: Confirmación eliminar contador

Una vez dentro del formulario, ya sea para editar o para crear un nuevo contador, se dispone de varias entradas de datos. En la parte superior hay una entrada para el identificador del contador, que en caso de edición esta inhabilitado, ya que no se permite cambiar una vez creado. A continuación, hay varios campos más como la póliza, dirección o descripción y un mapa mediante el cual se selecciona la localización del contador.

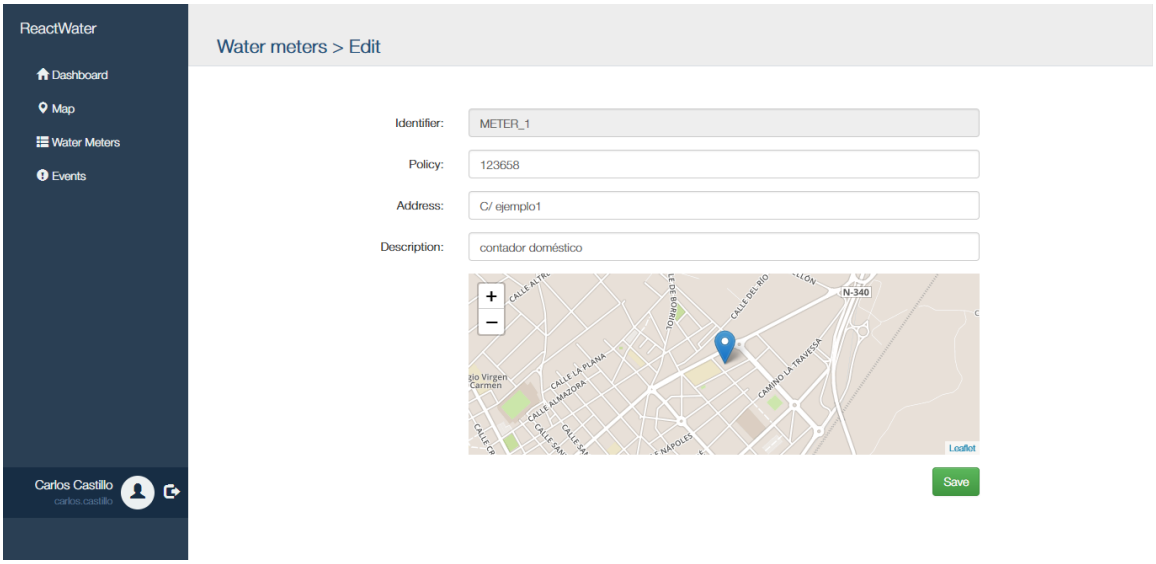


Figura 5.15: Pantalla para crear o editar contador

Cuando el usuario ha rellenado el formulario y desea finalizar, se debe pulsar sobre el

botón situado en la parte inferior derecha para guardar los cambios. Una vez se han enviado los datos al servidor y procesado la información, en caso de éxito se redirige al usuario a la pantalla anterior con todo el listado de contadores.

5.3.6 Detalles de contador

Si el usuario quiere ver los detalles de un contador, debe acceder al listado de contadores seleccionando la opción "Water meters" del menú lateral y seguidamente pulsar en el botón para ver los detalles del contador deseado. Una vez pulse el botón se redirigirá al usuario a la pantalla que se ve a continuación.

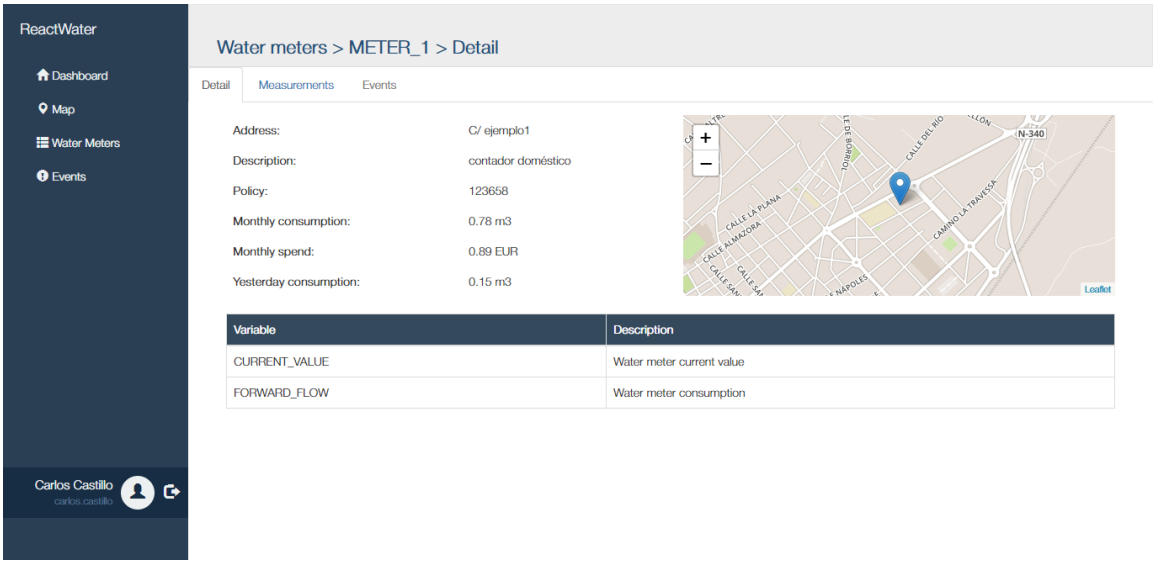


Figura 5.16: Pantalla de detalles de un contador

En esta pantalla se muestran varios datos del contador como sus propiedades, consumo mensual, variables y localización en el mapa. Además, en la parte superior hay un pequeño menú de navegación para poder navegar a otras páginas como son la de visualización de medidas y listado de eventos de un contador.

5.3.7 Visualización de medidas

Para visualizar las medidas de un contador se debe acceder primero a los detalles de un contador como se ha visto en la pantalla anterior. Hecho esto, se debe pulsar en la opción "Measurements" situada en las pestañas de navegación de la parte superior y accederemos a una pantalla como la que se muestra a continuación.

Esta pantalla esta compuesta por una gráfica en la parte inferior y un filtro en la parte superior. El filtro permite seleccionar la variable sobre la que queremos consultar los datos y

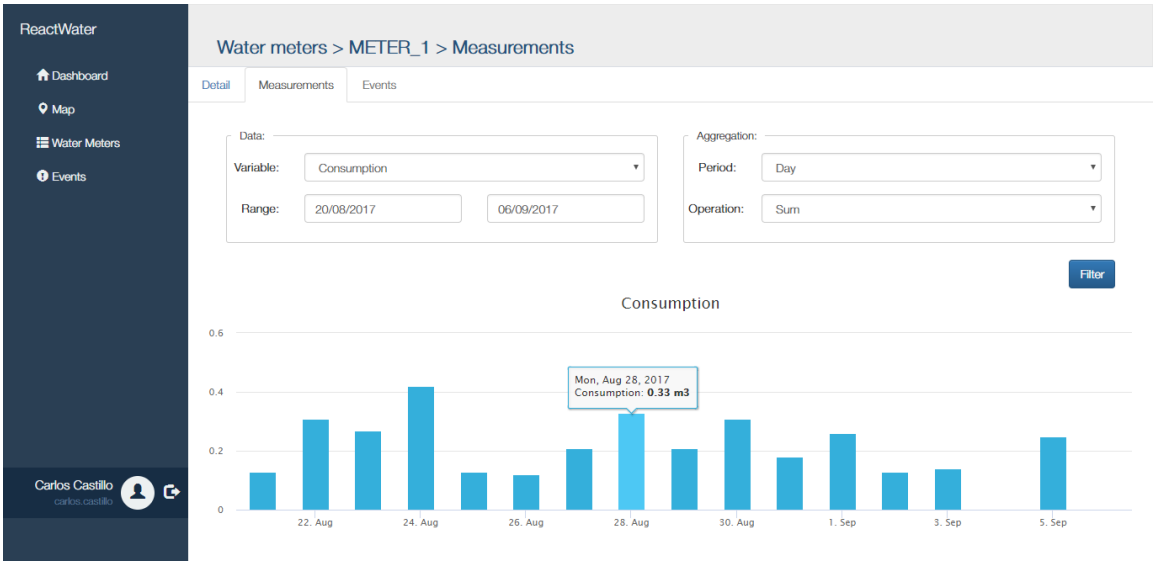


Figura 5.17: Visualización de consumo diario de un contador

un rango de fechas. Además, en la parte derecha del filtro están las opciones de agregación que permiten realizar operaciones con las medidas. En la imagen se puede apreciar como el usuario esta realizando la suma de las medidas del consumo y agrupándola por días, de esta manera obtiene el consumo total de cada día.

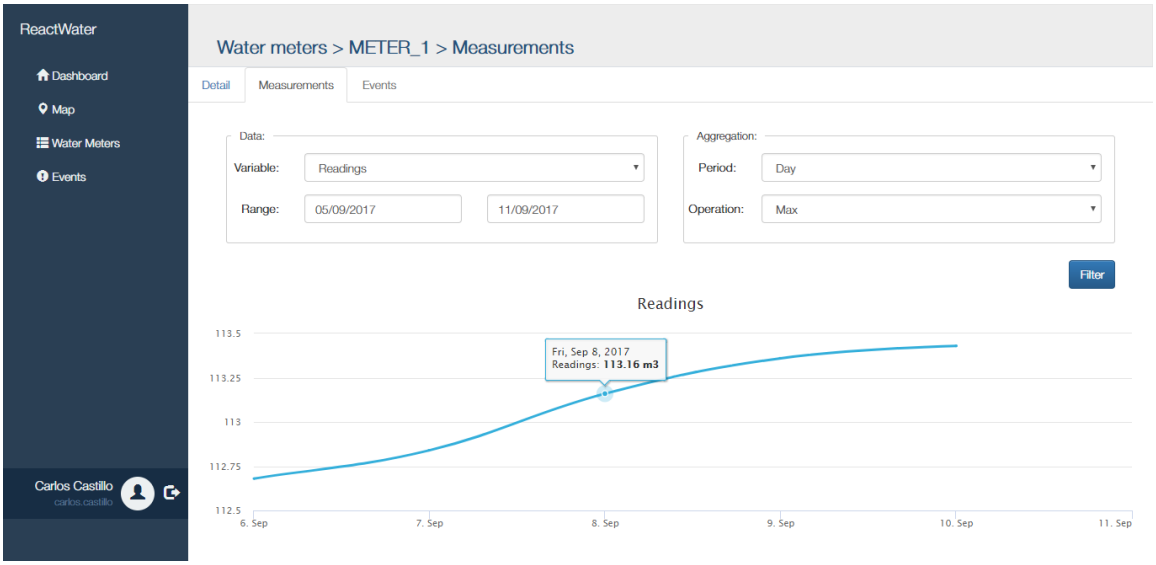


Figura 5.18: Visualización de la lectura de un contador

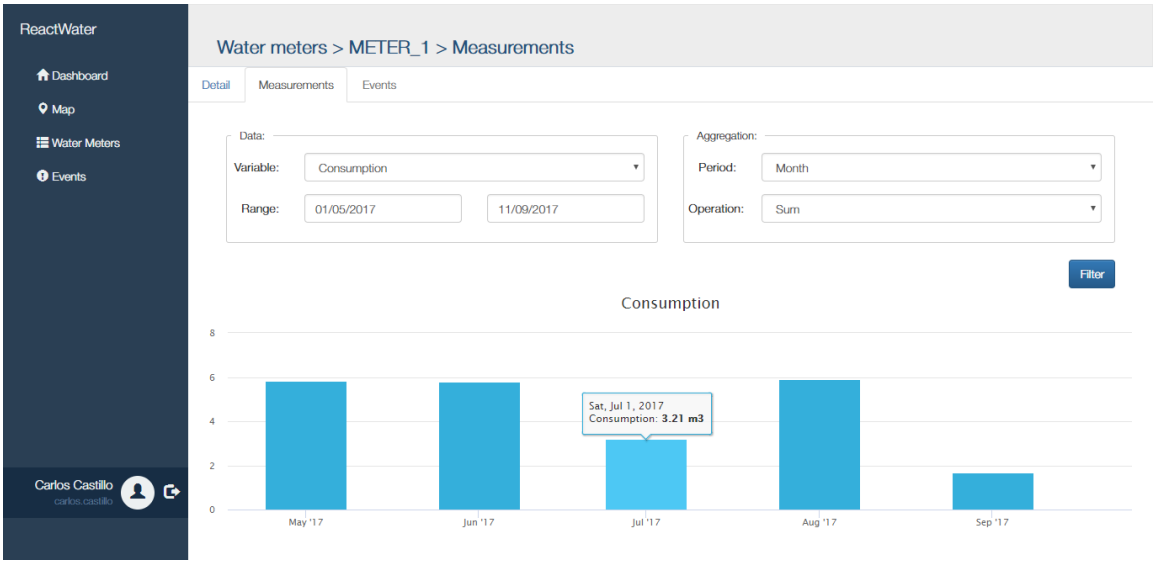


Figura 5.19: Visualización del consumo mensual de un contador

5.3.8 Listado de eventos

Para poder acceder al listado de eventos se puede hacer desde dos lugares. Desde el menú lateral y desde el navegador superior en el detalle de un contador. En el primer de los casos se pueden ver todas las alarmas sin ninguna restricción, mientras que en el segundo sólo aparecen las alarmas del contador desde el cual se ha accedido.

En la parte superior hay un filtro que permite filtrar por rango de fechas, estado y nombre de alarma. En la parte inferior aparece una lista con todas las alarmas paginadas.

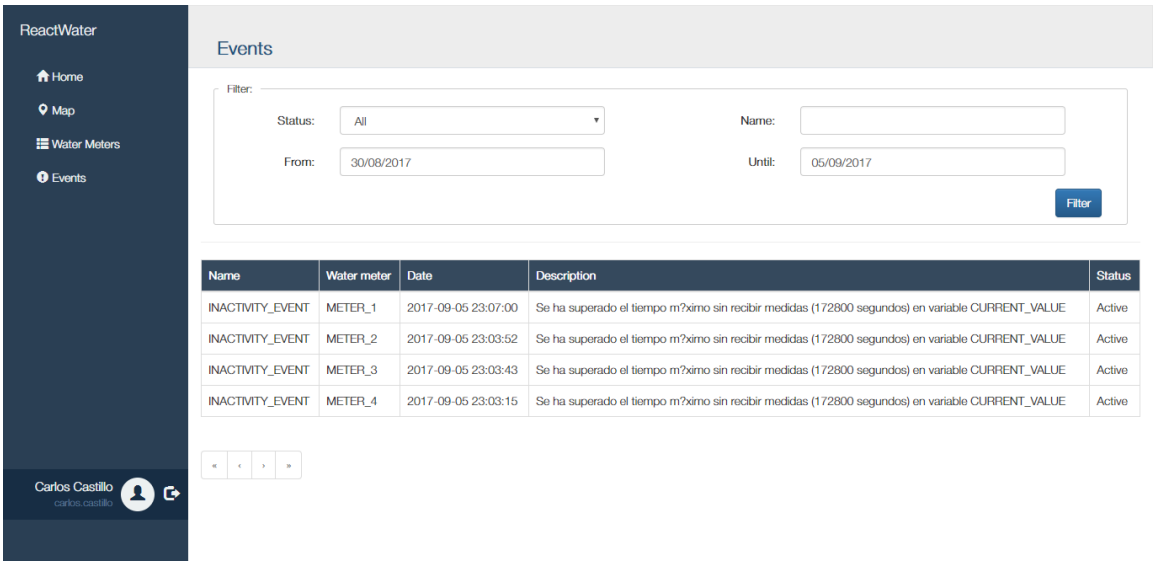


Figura 5.20: Listado de eventos

6. Validación y verificación

El objetivo de la fase de validación y verificación del sistema es comprobar el funcionamiento de este a todos los niveles. Esta fase es esencial en el proceso de desarrollo de software ya que proporciona información sobre la calidad del sistema que se está desarrollando. Existen varios tipos de pruebas según su alcance o según sea el tipo de componente que se está probando como podrían ser las siguientes:

- Pruebas de penetración
- Pruebas unitarias
- Pruebas Fuzz
- Pruebas de usabilidad
- Pruebas de rendimiento
- Pruebas de aceptación

El objetivo de todo desarrollo software no es llevar a cabo todos los tipos de pruebas, sino que para cada caso concreto hay que realizar un estudio con el objetivo de definir qué pruebas son útiles y viables, teniendo en cuenta variables como el tiempo o el presupuesto.En este proyecto se han realizado pruebas unitarias, de rendimiento y de usabilidad.

6.1 Pruebas unitarias

Las pruebas unitarias son aquellas que se encargan de comprobar el correcto funcionamiento de cada módulo o componente que constituye el sistema. La prueba de cada módulo es independiente de los demás, de esta forma se puede asegurar de que cada uno de los módulos trabaja como se espera por separado.

Para desarrollar las pruebas unitarias en este proyecto se ha utilizado la herramienta JUnit. Es un framework libre y de código abierto que permite definir casos de pruebas unitarias para programas escritos en Java. Esto permite desarrollar pruebas unitarias para componentes que tienen una alta dependencia con otros, sin llegar a ser pruebas de integración.

A continuación, se muestra un ejemplo de test unitario implementado y ejecutado con JUnit. En este caso se trata de un test que verifica el correcto funcionamiento del builder de contadores que se invoca cuando se desea crear un nuevo contador a partir de las propiedades recibidas del front-end.

```
1  @Test
2  public void buildAWaterMeterProperly() {
3
4      WaterMeter waterMeter = WaterMeterBuilder.create()
5          .withMeterId("METER1")
6          .withAddress("ADDRESS)
7          .withDescription("DESCRIPTION")
8          ...
9          .build();
10
11         assertThat(waterMeter.getMeterId(),is("METER1"));
12         assertThat(waterMeter.getAddress(),is("ADDRESS"));
13         assertThat(waterMeter.getDescription(),
14         is("DESCRIPTION"));
15         ...
16         }
```

Como se puede apreciar, los métodos que constituyen tests unitarios se identifican con la anotación @Test y es habitual asignarles nombres que describan su comportamiento.

Los test unitarios implementados se han ido ejecutando con frecuencia durante el desarrollo del proyecto para asegurar que cada cambio no ha afectado a la lógica implementada hasta el momento. No obstante, se debe tener en cuenta que en algunos casos la implementación de las pruebas es más costoso que el valor que proporcionan a la aplicación, por tanto, sólo se han realizado pruebas para aquellos módulos o funciones en

los que realmente ha merecido la pena invertir tiempo en ello.

6.2 Pruebas de usuario

Las pruebas de usuario se basan en la observación de cómo un grupo de usuarios lleva a cabo una serie concreta de tareas encomendadas por el evaluador, analizando los problemas de usabilidad que se encuentran.

Aún cuando el diseñador tenga amplios conocimientos sobre usabilidad, resulta recomendable evaluar el diseño con usuarios. Esto se debe a que, conforme más tiempo dedica un diseñador a un proyecto, menor es su perspectiva y más difícilmente detectará posibles problemas.

Se puede decir que gran parte de lo que el diseñador percibe cuando mira su propia obra, es una construcción mental; ve aquello que tienen en mente, no aquello que sus usuarios tendrán ante sus ojos.

El numero de participantes que se recomienda para realizar la prueba son 15, pero en este proyecto se ha realizado únicamente con 4 usuarios, todos ellos de un entorno cercano y con distintos perfiles. Cada usuario, ha realizado sobre el proyecto desarrollado una serie de tareas típicas que un usuario real llevaría a cabo: Buscar un contador, visualizar medidas, filtrar alarmas, modificar los datos de un contador, ver el mapa de calor, etc.

A medida que se han ido realizando estas pruebas, se han encontrado errores como escasa prevención de errores, gran número de pasos para realizar algunas acciones y uso de terminología técnica en varios mensajes.

Para solucionar la escasa prevención de errores se han añadido permisos para evitar que ciertos usuarios tengan acceso a ciertas funcionalidades. También se han añadido validadores en las entradas de datos, tanto en la parte front como en el servidor, para evitar que los usuarios introduzcan en la aplicación datos no deseados, los cuales pueden conllevar a errores.

Respecto al uso de terminología técnica y gran numero de pasos, se han solucionado reescribiendo los mensajes de la aplicación y re-diseñando la secuencia de pasos de algunos procedimientos, utilizando otros elementos de la interfaz más potentes que resultan más eficientes en este aspecto.

Gracias a las pruebas realizadas a lo largo del proyecto, se han detectado y corregido estos errores, lo cual ha favorecido al desarrollo del proyecto y a alcanzar los objetivos en el tiempo estimado.

6.3 Pruebas de rendimiento

Las pruebas de rendimiento sirven para determinar las velocidad en la que se realiza una tarea determinada del sistema en condiciones particulares de trabajo. También sirven para validar otros factores de la calidad del sistema, como son la fiabilidad y uso de recursos.

En este proyecto se han realizado pruebas de estrés y de carga en varias tareas, para determinar cuántos contadores es capaz de soportar el sistema y corregir cuellos de botella.

En un entorno con 50 contadores, el sistema tardaba más de la cuenta en varías pantallas como los mapas, dashboard y listado de contadores. Sin embargo después de realizar estas pruebas se detectaron cuellos de botella generados por un gran numero de peticiones a IoTsens, ya que se realizaba una llamada para recuperar las medidas del día anterior para cada uno de los contadores.

Para solucionar este problema, en el listado de contadores se añadió paginación y además, se descubrió que el API de IoTsens, tiene soporte para devolver medidas individuales para un listado de sensores. De esta manera, en lugar de realizar 50 peticiones para cada contador, se realiza una petición para los 50 contadores y posteriormente se trata el resultado para asignar a cada contador la medida recibida, si la tiene.

Gracias a estas pruebas y cambios, actualmente, el sistema es capaz de soportar sobre 10000 contadores, sin afectar en gran medida al rendimiento de la aplicación. Esto supone que el sistema puede soportar una carga normal a la carga recibida en un entorno cotidiano en pequeñas y medianas localidades. No obstante, aún se podrían realizar estrategias para optimizar mucho más el sistema, sin embargo al tratarse este proyecto de un prototipo no merece la pena dedicar tanto tiempo en esta tarea.

7. Conclusiones y trabajos futuros

En este capítulo se comentan las conclusiones a las que se ha llegado al final del desarrollo del proyecto, además de algunas mejoras que se podrían implementar en la aplicación en el futuro.

7.1 Conclusiones

El propósito de este proyecto ha sido investigar el sector del abastecimiento de agua, su actual modo de operar y diseñar una solución vertical basada en el Internet de las cosas que mejore los sistemas que tienen actualmente. La consecución de los objetivos ha sido satisfactoria, y todo el trabajo se ha completado con éxito en el tiempo estipulado, lo cual me ha hecho sentir bastante satisfecho, ya que no disponía de gran cantidad de tiempo libre a causa del trabajo.

Antes de iniciar el desarrollo de la aplicación, ha sido necesario aprender una serie de tecnologías, como React Js, de la cual no tenía ninguna experiencia previa. A parte de la implementación propiamente dicha, se ha realizado una fase de análisis a partir de los datos obtenidos de la empresa estudiada, ya que no tenía conocimientos previos de empresas de este sector. Además, puesto que una parte importante de la aplicación es la parte gráfica, ha sido necesario profundizar en aspectos de diseño, componentes gráficos y usabilidad, ya que es indispensable que resulte fácil de utilizar al usuario y mejor sus actuales sistemas.

Las mejoras que ofrece esta solución respecto al sistema que tiene la empresa estudiada y supongo que la gran mayoría de empresas de este sector son más que evidentes. De todas ellas, se van a listar las ventajas más importantes a continuación.

- La empresa del agua puede tener a diario datos sobre el consumo de agua de todos sus clientes.

- Bien es cierto que la solución supone un desembolso inicial de dinero para obtener e instalar sensores en los contadores. No obstante, a la larga resultará en un ahorro de dinero, ya que no hará falta tener tantos operarios contratados para ir leyendo contador a contador por cada edificio de la localidad.

- Con este sistema se le puede ofrecer al cliente la oportunidad de llevar un control en tiempo real de su consumo.

- Gracias al sistema de alarmas que ofrece IoTsens, se pueden detectar problemas y actuar con mucha más velocidad que con el sistema actual.

- Se pueden obtener estadísticas más precisas sobre consumos.

En lo referente a este nuevo enfoque, el cual integra los sistema de un entorno vertical con plataformas horizontales para el Internet de las cosas, creo que es el camino que las empresas deberían seguir en un futuro, dejando a un lado las propuestas ad-hoc que en la actualidad se imponen como soluciones a medida para los proyectos de este tipo. Este nuevo enfoque presenta muchas ventajas, entre las cuales cabe destacar.

- Es muy rápido y ágil desarrollar nuevas aplicaciones para un sector vertical, ya que no hay que desarrollar una gran parte del sistema.

- Gracias a su arquitectura orientada a servicios es muy fácil integrar con los sistemas de una organización.

- La empresa no debe preocuparse por problemas de rendimiento y escalabilidad. Este tipo de plataformas están pensadas para escalar y soportar un gran volumen de comunicaciones y datos. Esto es un gran punto a favor cuando no se dispone de una gran infraestructura.

- Estas plataformas añaden una capa mas de seguridad a los sistemas, por lo que hace mas seguras las aplicaciones.

- En caso de la aparición de una nueva tecnología de sensores, la plataforma es la responsable de añadir nuevos decodificadores y mecanismos necesarios para soportarla. De esta modo, los desarrolladores de la empresa no necesitan adquirir nuevos conocimientos para cada tecnología nueva que surja.

Finalmente, como aprendizaje personal, la realización este proyecto me ha servido cómo oportunidad para poder poner a prueba varios de los conocimientos adquiridos durante el

transcurso de los estudios de máster en las diferentes asignaturas cursadas. Además, he aprendido a investigar y detectar necesidades en un ámbito, así como plantear los problemas a abordar, desarrollar una solución en base a los recursos intelectuales disponibles y justificar en gran medida de lo posible la mejoría de la solución aportada.

7.2 Trabajos futuros

Como trabajo futuro, se han propuesto varias mejoras y funcionalidades para este proyecto, que en la fase de planificación se desestimaron por falta de tiempo. Estas mejoras son de bastante interés para la empresa de gestión de aguas. Entre estas cabe destacar las siguientes:

- La posibilidad de crear, gestionar y obtener datos de grupos de balance. Un grupo de balance está formado por tres grupos de sensores: Entrada, Salida y Consumidores. Suele representar un edificio, o barrio donde se tiene un

- Poder personalizar y crear informes los cuales contengan datos como consumo de sensores, grupos de balance, etc. Además, la posibilidad de poder descargarlos en Excel o PDF.

- Crear alarmas personalizadas gracias a la funcionalidad que proporciona IoTsens para ello. Un ejemplo de alarma personalizada podría ser superar cierto consumo, o tener instalado un sensor en una fuente que sea capaz de analizar el agua y avisar en caso de presentar indicios de *Legionella*.

- Ofrecer más información en los mapas de calor como alarmas o edad de los contadores, y poder visualizar la evolución de esta en un rango de tiempo.

- Permitir bidireccionalidad en la aplicación y poder cortar el suministro de casas, fuentes, etc. desde el centro de control, sin necesidad de mandar operarios.

Bibliografía

[1] K. Sierra. E. Freeman B. Bates. *Head first Design patterns.* O'Reilly Media, 2004.

[2] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. 1st edition. Prentice Hall, 2005.

[3] Mass Framingham. "Internet of Things Spending Forecast to Grow 17.9% in 2016 Led by Manufacturing, Transportation, and Utilities Investments, According to New IDC Spending Guide". In: (Jan. 2017).

[4] A Al-Fuqaha. "TOWARD BETTER HORIZONTAL INTEGRATION AMONG IOT SERVICES". In: (2015).

[5] F. Javier Soriano Camino y J. José Moreno Navarro J. Garbajosa Sopeña. *Informe de Vigilancia Tecnológica madri+d - Tecnologías software orientadas a servicios*. Fundación madri+d para el Conocimiento, 2008.

[6] R.L. Mack J. Nielsen. *Usability Inspection Methods*. Wiley, 1994.

[7] Nicolai M. Josuttis. *SOA in Practice. The Art of Distributed System Design*. 1st edition. O'Reilly, Aug. 2007.

[8] Y Li. "A SDN-based Architecture for Horizontal Internet of Things Services". In: (2016).

[9] Sam Newman. *Building Microservices*. O'Reilly Media, 2015.

[10] R.S. Pressman. *Software engineering: A practitioner's approach*. McGraw-Hill Higher Education, 2014.

[11] Stoyan Stefanov. *React: Up & Running: Building Web Applications*. O'Reilly Media, Dec. 2015.