UNIVERSITAT JAUME·I

GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO FINAL DE GRADO

Desarrollo de una aplicación web para la gestión y monitorización de datos de plantas industriales

Realizado por:
ANDRÉS GIUSTINI COLÁS

Supervisado por:
JAVIER MUÑOZ FERRARA

Tutorizado por:
ISMAEL SANZ BLASCO

Fecha de lectura: 1 de julio de 2016
Curso académico 2015/2016

Resumen

Este documento contiene la memoria del trabajo final de grado en el que se ha desarrollado una aplicación web para la gestión de plantas industriales implantadas por parte de Sitra, una empresa perteneciente al grupo empresarial Grupo Gimeno.

El proyecto se ha llevado a cabo en la empresa ADC Infraestructuras y Sistemas S.L., que ofrece cobertura tecnológica al resto de empresas que componen el grupo.

El trabajo realizado constituye un módulo de la aplicación I2OT (IoT Industrial), una aplicación para monitorizar el comportamiento de plantas industriales y que ofrece posibilidad de actuar sobre éstas de forma remota. Esta comunicación se lleva a cabo a través de señales, con las que se obtienen datos en tiempo real del estado de las plantas, pero que también permiten el envío de órdenes hacia los componentes de éstas.

Concretamente, el proyecto ha consistido en el desarrollo de los sistemas de administración de plantas industriales, usuarios, roles y señales, permitiendo funcionalidades relevantes para el mantenimiento de las plantas como la visualización de los datos en gráficas de líneas o la posibilidad de manipular dichos datos.

Para ello se ha realizado un análisis de los requisitos y las funcionalidades demandadas por el cliente, se han estudiado las diferentes tecnologías y herramientas adoptadas por la empresa para el desarrollo de aplicaciones y, posteriormente, se ha procedido con la implementación propiamente dicha de la aplicación. Para finalizar el desarrollo se han llevado a cabo una serie de pruebas para comprobar que el funcionamiento del producto final es satisfactorio.

Palabras clave

IoTsens, Internet de las cosas, plantas industriales, sensores, Java, aplicación web.

Abstract

This document presents the results of the end-of-degree project, which comprises the development of a web application that manages industrial plants operated by Sitra, a subsidiary of Grupo Gimeno business group.

The project has been undertaken at ADC Infraestructuras y Sistemas S.L., a company that offers technical support and IT solutions to the other companies in the group.

The work presented in this paper represents a module within I2OT (Industrial IoT), an application designed to monitor the behavior of industrial plants that offers the ability to interact with the sites remotely. This communication is performed through signals which transmit real time data about the state of the plant but that also allow sending commands to plant components.

More specifically, the project consisted of developing the industrial site management system, with users, user roles and signals allowing the relevant maintenance operations to be performed such as graphing sensor data or data manipulation.

To begin with, a requirements analysis has been performed taking into consideration the functionality desired by the client, then various technologies and tools have been evaluated and a working set has been chosen and then the application development began. Before product delivery, extensive testing has been performed to ensure the application fulfilled customer requirements and met quality specification.

Keywords

IoTsens, Internet of Things, industrial plants, sensors, Java, webapp.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Contenidos

El *Internet de las cosas* es una corriente tecnológica emergente que pretende dotar de cierta inteligencia e independencia a objetos de la vida cotidiana como puede ser la persiana de una ventana o un frigorífico.

IoTsens[1] es una plataforma horizontal implantada por la división IoT de ADC Infraestructuras y Sistemas S.L. que proporciona 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.

Una de las aplicaciones o "verticales" de IoTsens es I2OT o IoT Industrial, una aplicación para la gestión de plantas industriales que permite una comunicación bidireccional, de manera que se puede tanto recibir datos de las plantas como enviar órdenes o consignas hacia ellas.

En esta sección se describe de forma más ámplia el contexto en el que se desarrolla el proyecto, así como sus principales objetivos.

[1] www.iotsens.com

1.1 Descripción de la empresa

El grupo empresarial *Grupo Gimeno* se funda en Castellón hace más de 140 años mediante la constitución de *FACSA*, considerada en la actualidad como la empresa española con más experiencia en la gestión del ciclo integral del agua.

Sin embargo, con el paso de los años, *Grupo Gimeno* ha ido evolucionando y consolidando su presencia en otros sectores económicos como la construcción, el turismo y el ocio. Fruto de esta evolución, surgen tres divisiones: *Gimeno Servicios*, *Gimeno Construcción* y *Gimeno Turismo y Ocio*.

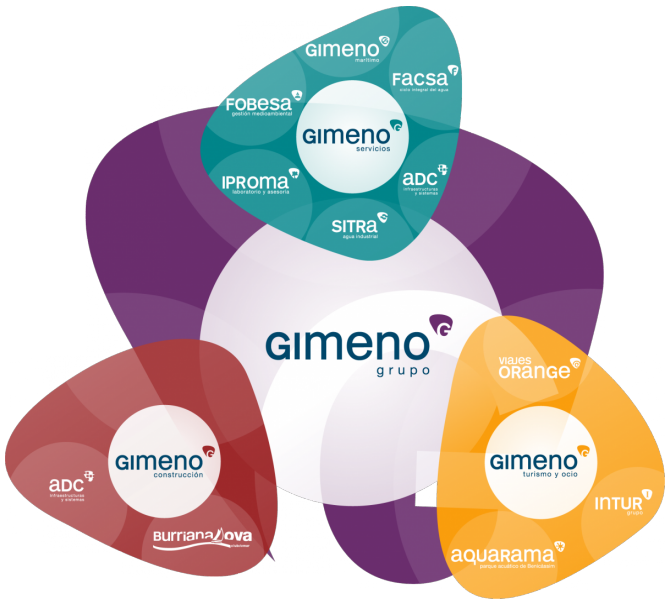

Figura 1. Organigrama de las divisiones de *Grupo Gimeno*

El proyecto a desarrollar durante la estancia en prácticas tendrá lugar en la empresa *ADC Infraestructuras y Sistemas, S.L.*, perteneciente a la división de servicios. Esta empresa ofrece cobertura tecnológica al resto de las empresas que componen el Grupo Gimeno y se divide, a su vez, en tres departamentos según el ámbito informático que cubren: *Software*, *Datacenter y Comunicaciones* y *Microinformática*.

Puesto que la gestión de agua es una de los servicios más relevantes del Grupo Gimeno, ADC es una empresa especializada en el desarrollo de los sistemas de telecontrol del ciclo integral del agua. ADC crea sus propias aplicaciones informáticas para la gestión de abastecimientos y el control de estaciones depuradoras.

Por su parte, Sitra (otra de las empresas de la división de servicios), está especializada en la gestión del ciclo integral del agua en la industria. Sus principales actividades son el diseño, construcción y puesta en marcha de plantas industriales para la potabilización de

aguas industriales, acondicionamiento de aguas para usos industriales y la implantación de estaciones depuradoras de aguas residuales industriales y sistemas de regeneración.

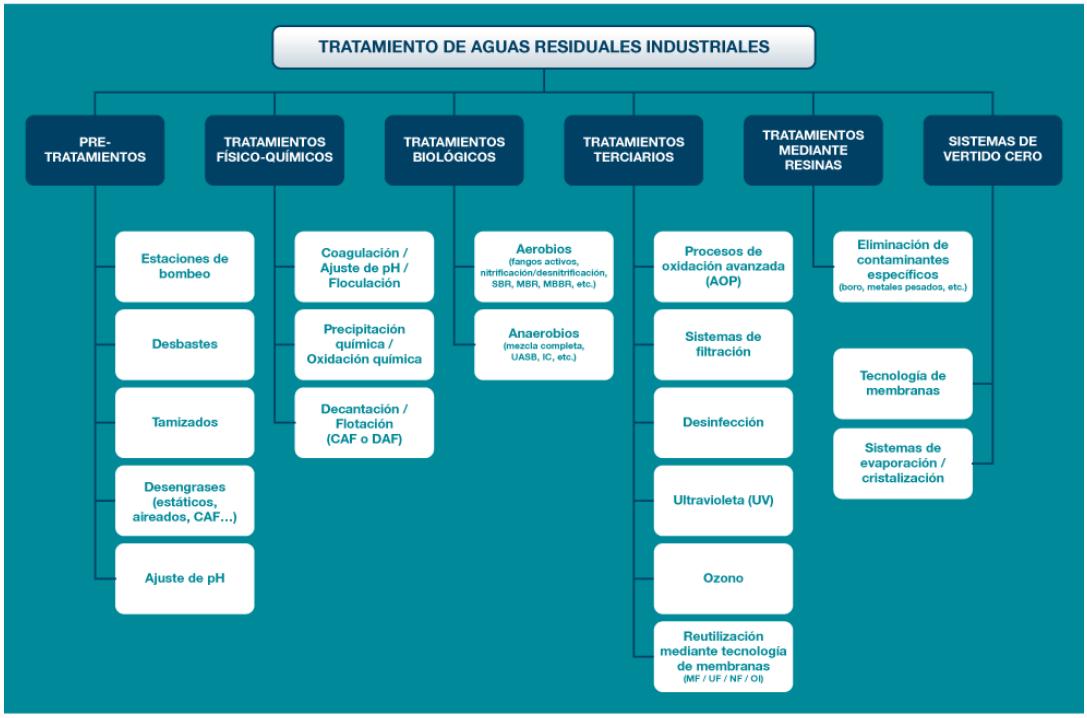

Figura 2. Ejemplo de actividades realizadas por *SITRA*

1.2 El mundo del IoT

El IoT (*Internet of Things* o Internet de las cosas en castellano) es un concepto que se refiere a la interconexión de objetos cotidianos a través de internet. El objetivo del IoT es hacer que, mediante ciertos dispositivos o tecnologías, estos objetos se comuniquen entre sí y, por consiguiente, sean más independientes e "inteligentes" [1].

Ejemplos prácticos de IoT podrían ser, por ejemplo, una vivienda que mida la temperatura y humedad del hogar y actúe en consecuencia a través de la activación del sistema de aire acondicionado, o un frigorífico capaz de determinar qué alimentos hay en su interior y encargar al proveedor especificado aquellos que se encuentren agotados.

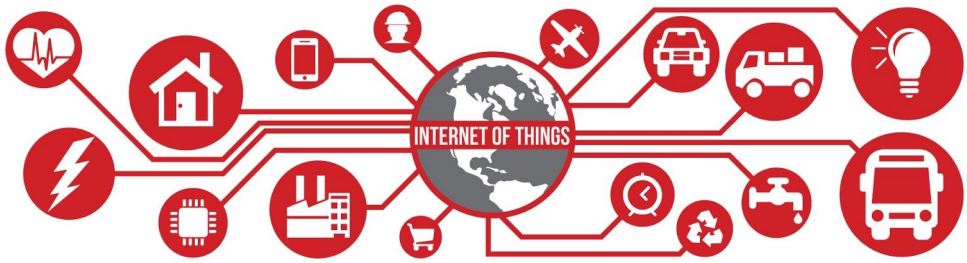

Figura 3. Diagrama representativo del concepto de IoT

IoTsens se basa en este concepto para la implantación de ciudades inteligentes, proporcionando servicios en el ámbito del Internet de las cosas a empresas como Sitra o incluso a otras empresas u organismos externos al grupo.

1.3 IoTsens: Smart City as a service

IoTsens proporciona una red de comunicaciones y una plataforma software de procesamiento de la información basados en estándares para *Smart Metering* y ciudades inteligentes que permite el desarrollo rápido de soluciones y servicios verticales. La arquitectura de IoTsens permite la integración dinámica de todo tipo de agentes (sensores, actuadores, sistemas de información externos, etc.) y el tratamiento homogéneo de los datos para su análisis y explotación (paneles de control, alarmas, análisis de tendencias, etc.).

Figura 4. Logotipo de IoTsens

La solución IoTsens permite integrar de manera dinámica agentes heterogéneos en un sistema integral de *Smart City* con comunicación bidireccional. 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 procesa o genera estos mensajes heterogéneos utilizando componentes especializados para cada tecnología, integrados de manera dinámica. Un vez procesados, los datos son analizados según su naturaleza para generar alarmas, producir informes, etc.

Con esta arquitectura tan flexible es posible desarrollar soluciones verticales para

Ciudades o Espacios Inteligentes (telemetría, ahorro energético, etc.) que satisfacen los requisitos, en este caso, de Sitra.

1.4 Contexto y motivación

Este proyecto surge de la necesidad por parte de Sitra de monitorizar la actividad de las plantas industriales, de manera que sea posible optimizar los recursos para maximizar la eficiencia de la gestión del agua a través de la medición y tratamiento de diferentes variables.

La aplicación a desarrollar formará parte del ámbito de IoTsens, la plataforma horizontal descrita con anterioridad que permite recolectar, almacenar y analizar información de diferentes lugares, conociendo lo que ocurre en tiempo real y pudiendo actuar de forma inmediata. Y precisamente son éstos los requisitos de Sitra para la aplicación desarrollada.

La plataforma consta de una estructura de cuatro capas, que se enumeran a continuación:

1. *Capa de sensores y dispositivos*. Se sitúan los dispositivos requeridos en los lugares donde se pretende realizar el control de actividad.

2. *Capa de conectividad*. La forman las redes de comunicación a través de las cuales se transmitirán los datos.

3. *Capa de almacenamiento de datos*.

4. *Capa de aplicación*. La representan las aplicaciones que utilizarán los usuarios finales, y que permiten la monitorización de los datos obtenidos y la gestión de las actividades controladas.

Donde la capa de aplicación es a la que pertenece la aplicación desarrollada en el proyecto.

Gracias a esta plataforma, se puede tanto obtener como enviar información. Es decir, permite una comunicación bidireccional a través de señales, con la que poder controlar a distancia la actividad de las plantas industriales. Para ello se utilizarán paneles sinópticos o sistemas *SCADA* (Supervisión, Control y Adquisición de Datos), que representarán la planta industrial y sus procesos en tiempo real.

Sin embargo, la comunicación *aplicación web - planta industrial* queda fuera del alcance de este proyecto, puesto que el trabajo a realizar durante la estancia en prácticas constituye únicamente ciertos módulos que se detallarán más adelante.

En general, la aplicación a desarrollar constituirá un sistema de información mediante el cual poder administrar la información de las diferentes plantas industriales y usuarios de

las mismas, además de ser una herramienta con la que poder tomar decisiones inmediatas y precisas a través de la medición y tratamiento de datos en tiempo real.

Por tanto, la aplicación debe permitir dar de alta o añadir en el sistema las plantas industriales de las cuales se pretende monitorizar su actividad. Además, la aplicación debe permitir la gestión de usuarios del sistema, incluyendo los usuarios de cada planta industrial. De esta manera, debe ser posible asignar a cada usuario un cierto rol dentro de una planta industrial (Administrador, Operario, Técnico o Cliente), dando una serie de permisos o privilegios a cada tipo de usuario dentro de cada planta.

Es posible monitorizar la actividad de cada planta gracias a una serie de sensores colocados físicamente en los lugares donde se pretende conseguir información. Estos datos serán recogidos y enviados a la aplicación web, la cual será capaz de mostrarla ordenadamente por medio de gráficas, permitiendo obtener estadísticas relevantes para la administración de la planta.

1.5 Objetivos del proyecto

El objetivo principal de este proyecto es tanto diseñar como implementar una aplicación web que permita a los usuarios de las plantas industriales de *SITRA* la visualización de datos relevantes en tiempo real, así como la gestión de la información de la planta y de los usuarios del sistema.

El objetivo principal se puede desglosar en los siguientes subobjetivos:

- Facilitar, mediante una interfaz, la gestión de plantas industriales. Esto es, dar de alta plantas en el sistema, actualizar su información y poder visualizar todas las presentes mediante un listado.

- Permitir, mediante una interfaz, la gestión de usuarios. Es decir, poder añadir usuarios en el sistema, actualizar su información y poder visualizar todos los usuarios dados de alta.

- Posibilidad de asignar un usuario a una planta mediante un rol de los definidos en el sistema. Cada rol otorgará al usuario una serie de permisos en una planta.

- Visualizar mediante un listado paginado las señales de la planta, así como su naturaleza o su dirección.

- Visualizar los datos requeridos de cada planta a través de gráficas.

- Posibilidad de descargar dichos datos como un documento *Excel* o como una imagen.

1.6 Estructura de la memoria

Esta memoria está dividida en siete capítulos: *Introducción*, *Planificación del proyecto*, *Tecnologías utilizadas*, *Análisis de requisitos*, *Implementación*, *Verificación y validación* y *Conclusiones y trabajo futuro*, además de dos anexos con información relevante suplementaria de algunos capítulos.

A lo largo de la *Introducción* se pretende ofrecer una visión general de la empresa en la que se ha realizado el proyecto, así como exponer algunos conceptos como el IoT, indispensables para comprender el trabajo descrito a lo largo del documento.

El segundo capítulo se dedica a la *Planificación del proyecto*, las distintas tareas en las que se debe descomponer y el coste temporal asignado a cada una de ellas.

En el siguiente capítulo, *Tecnologías utilizadas*, se detallan algunas de las herramientas de desarrollo que se han utilizado para la implementación del proyecto, además de ciertos estándares y convenios que se adoptan en la empresa y que ha sido necesario seguir.

En el *Análisis de requisitos* se presentan descritos de forma precisa los bloques que componen la aplicación, a través de un diagrama de casos de uso, especificaciones tanto de los requisitos de datos como de los requisitos funcionales, prototipos de las interfaces de usuario, entre otros.

El capítulo *Implementación* contiene detalles sobre la arquitectura de capas de la aplicación y algunas decisiones de desarrollo que se han tomado. También se muestra la apariencia final de las interfaces de usuario de las distintas secciones de la aplicación.

Como es natural tras terminar la implementación se deben realizar un conjunto de pruebas para asegurar el buen funcionamiento del producto desarrollado. Esto se trata en el siguiente capítulo, denominado *Verificación y validación*.

Por último, en el capítulo *Conclusiones y trabajo futuro* se exponen las impresiones y reflexiones alcanzadas tras la finalización del proyecto, además de exponer brevemente algunas funcionalidades que está previsto implementar en I2OT en el futuro.

Capítulo 2

Planificación del proyecto

Contenidos

En este capítulo se pretende dar una visión general de la etapa de planificación. Se describirá primero la metodología seguido de la planificación temporal y el desglose de tareas. Por último se hará una estimación de los recursos necesarios para llevar a cabo el proyecto y un pequeño apartado con su seguimiento.

La etapa de planificación es crucial para definir el proyecto de manera precisa desglosándolo en tareas más pequeñas y estimando los costes relacionados con la ejecución de cada una.

2.1 Metodología de trabajo

Con la finalidad de conseguir una aplicación de calidad en relación con las tecnologías, estándares y convenios utilizados en la empresa ADC, la primera fase de la estancia en prácticas constó de un proceso de formación mediante el cual introducirse a la manera de trabajar en la empresa y tener una primera toma de contacto con algunos de los proyectos ya realizados similares al que se ha de desarrollar.

Una vez recibida toda la formación inicial necesaria, y alguna información general referente al proyecto concreto a desarrollar, se comienza la etapa de implementación del software.

Para ello, se ha trabajado haciendo uso de las metodologías ágiles de la ingeniería del software, concretamente SCRUM. ADC ha tomado esta decisión, puesto que considera que este tipo de metodología ofrece efectividad y flexibilidad ante cambios de requisitos. Periódicamente la empresa se reúne con el cliente para mostrar los progresos. Estas reuniones permiten detectar tempranamente errores en el desarrollo y posibilitan que la modificación de los requisitos no supongan trabajo realizado en vano.

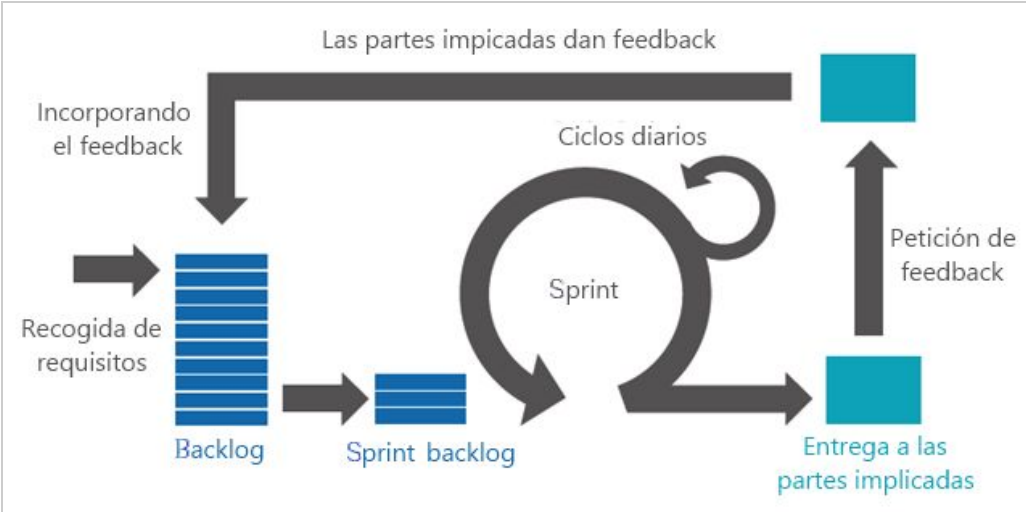

Figura 5. Diagrama típico del ciclo de vida de Scrum

En la Figura 5 se ilustra el ciclo de vida de SCRUM a través de un diagrama típico, donde las diferentes partes se detallan a continuación:

- *Backlog*. Es la pila de requisitos que debe implementar el sistema. Se compone de funcionalidades y tareas que debe realizar el programador que están 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 con la herramienta Jira, que se describe posteriormente.

- *Sprint*. Es la iteración propiamente dicha, que dura de una a 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 anterior.

Además, al inicio y fin de cada iteración se realizan reuniones de planificación de la entrega y construcción del backlog, e inspección del incremento integrado en cada sprint. La metodología también define que se han de realizar *daily meetings* (o reuniones diarias) de 15 minutos aproximadamente para identificar posibles problemas y obstáculos, y resolverlos lo antes posible. El equipo de desarrollo de SCRUM está formado habitualmente 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 Manager*. 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.

Aunque este proyecto se ha basado en una metodología ágil, en cada iteración se ha realizado un proceso de ingeniería de software clásico y, por claridad, este proyecto se ha documentado de esta forma; el apartado de desarrollo de la aplicación se divide en cuatro fases bien diferenciadas:

- Análisis de requisitos. En esta fase, se estudian las necesidades del cliente y qué espera que el sistema ofrezca. También, se identifican los distintos actores que estarán 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, que 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, etcétera. 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. Para lanzar el sistema a producción es necesario probarlo en distintos escenarios para asegurar su correcto funcionamiento. Se han llevado a cabo pruebas unitarias, en las que se prueba la lógica interna de un componente independientemente de otros, y pruebas de integración, donde se enlazan los distintos componentes para asegurar que estos cooperan correctamente.

- Despliegue o implantación. En esta fase, el producto es lanzado en un entorno real de producción. En algunos casos, esta fase puede dividirse en dos: lanzar el producto para un grupo reducido de usuarios y estos reportar errores, en caso de encontrarlos, y

finalmente, una vez el sistema funciona correctamente durante un tiempo establecido, realizar el lanzamiento masivo para todos los usuarios.

Por lo tanto, el software se desarrollará en iteraciones de una a cuatro semanas y, al final de cada iteración, se dispondrá de un producto final completamente funcional. De esta manera también se evitan problemas de retrasos de tiempo y complejidad.

2.2 Planificación temporal

La planificación temporal del proyecto fue definida junto con la empresa ADC Infraestructuras y Sistemas S.L., para establecer unos plazos y un horario de trabajo en la empresa que permitan completar el proyecto. Se han definido una serie de actividades que se deberán completar y se ha estimado el coste temporal de cada una.

Las tareas principales del proyecto se pueden ver en el esquema de descomposición de trabajo (EDT) de la Figura 6.

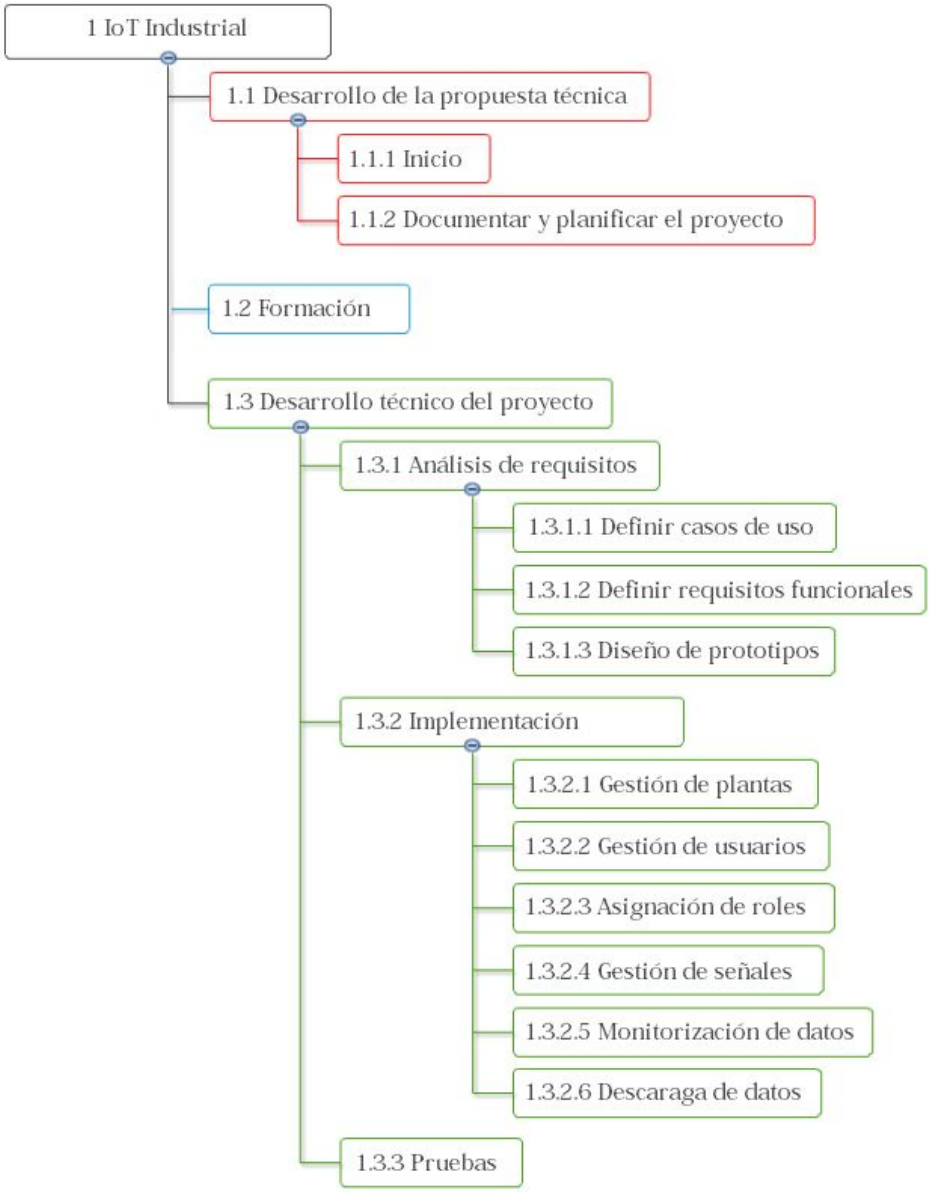

Figura 6. EDT del proyecto

Además de las actividades destinadas al desarrollo del proyecto ha sido necesario establecer otro tipo de tareas como por ejemplo para la creación de la propuesta técnica o la formación relativa a las tecnologías y convenios de la empresa. La redacción de esta memoria y todo el trabajo relacionado no forman parte de las 300 horas calculadas de estancia, y han supuesto aproximadamente unas 129 horas de trabajo que aparecen desglosadas más adelante en la Tabla 1.

2.2.1 Calendario de trabajo

Para estimar los costes temporales de las diferentes actividades se ha supuesto que el alumno trabajará en la empresa nueve horas los lunes y miércoles, y seis horas los martes, jueves y viernes durante diez semanas. El calendario completo de la estancia en prácticas se puede ver en la Figura 7.

Noviembre 2015						
L	M	M	J	V	S	D
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

Diciembre 2015						
L	M	M	J	V	S	D
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Enero 2016						
L	M	M	J	V	S	D
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Figura 7. Calendario de trabajo durante la estancia en prácticas

La previsión de horas detallada a la semana es la siguiente:

Noviembre (**93 h.**)

Semana 11 al 13: 9 + 6 + 6 = **21 h.**
Semana 16 al 20: 9 + 6 + 9 + 6 + 6 = **36 h.**
Semana 23 al 27: 9 + 6 + 9 + 6 + 6 = **36 h.**

Diciembre (**153 h.**)

Semana 30 al 4: 9 + 6 + 9 + 6 + 6 = **36 h.**
Semana 7 al 11: 9 + 6 + 6 = **21 h.**
Semana 14 al 18: 9 + 6 + 9 + 6 + 6 = **36 h.**
Semana 21 al 25: 9 + 6 + 9 + 6 = **30 h.**
Semana 28 al 1: 9 + 6 + 9 + 6 = **30 h.**

Enero (**54 h.**)

Semana 4 al 8: 9 + 6 + 6 + 6 = **27 h.**
Semana 11 al 14: 9 + 6 + 9 + 3 = **27 h.**

De esta manera se ocupan las 300 horas de las que debe constar el proyecto. El desglose de tareas durante dichas horas se detalla en la siguiente sección.

2.2.2 Costes temporales y diagrama de Gantt

En esta sección se puede ver un desglose del coste temporal de cada tarea y el diagrama de Gantt que representa la evolución temporal y la interdependencia entre las tareas del proyecto.

En la Tabla 1 se presenta el desglose temporal de las tareas del proyecto a llevar a cabo durante las horas relativas a la estancia en prácticas.

TAREA	ACT. PRECEDENTES	DURACIÓN (horas)
1 Proyecto Final de Grado		**300**
1.1 Desarrollo de la propuesta técnica		**26**
1.1.1 Inicio		**4**
1.1.1.1 Definir el proyecto con el tutor y el supervisor		1
1.1.1.2 Definir el método de trabajo y documentación	1.1.1.1	3
1.1.2. Documentar y planificar el proyecto		**8**
1.1.2.1 Buscar información del contexto y ámbito del proyecto	1.1.1	4
1.1.2.2 Identificar alcance y objetivos	1.1.1	4
1.1.3. Planificar el proyecto		**14**
1.1.3.1 Definir tareas y estimar fechas	1.1.2	4
1.1.3.2 Documentar la propuesta del proyecto	1.1.3.1	10
1.1.3.3 Entregar la propuesta técnica	1.1.3.2	0
1.2 Formación		**21**
1.2.1 Introducción e historia del Grupo Gimeno		1
1.2.2 Funciones de ADC dentro del grupo	1.2.1	1
1.2.3 Formación en cuanto a las tecnologías a utilizar	1.2.2	19
1.3 Desarrollo técnico del proyecto		**253**
1.3.1 Análisis de requisitos		**30**
1.3.1.1 Definir casos de uso	1.2.3	10
1.3.1.2 Definir requisitos funcionales	1.3.1.1	6
1.3.1.3 Diseñar prototipos casos de uso	1.3.1.1	8
1.3.1.4 Validar el análisis	1.3.1.3	6
1.3.2 Implementación		**211**
1.3.2.1 Gestión de plantas	1.3.1.4	72
1.3.2.2 Gestión de usuarios	1.3.1.4	40
1.3.2.3 Asignación de roles	1.3.1.4, 1.3.2.1, 1.3.2.2	36
1.3.2.4 Gestión de señales	1.3.1.4	24
1.3.2.5 Monitorización de datos	1.3.1.4, 1.3.2.4	30
1.3.2.6 Descarga de datos en formato *Excel* o *PNG*	1.3.1.4, 1.3.2.5	9
1.3.3 Pruebas		**12**
1.3.3.1 Detectar pruebas de unidad	1.3.2	2
1.3.3.2 Realizar y validar pruebas	1.3.3.1	4
1.3.3.3 Realizar pruebas de interfaz de usuario	1.3.2	4
1.3.2.4 Comprobar consistencia estética	1.3.2	2

Tabla 1. Desglose del coste temporal de las tareas que conforman el proyecto

La planificación temporal relativa a tareas no presenciales se detalla en la Tabla 2.

TAREA	ACT. PRECEDENTES	DURACIÓN (horas)
1.4. Documentación y presentación del TFG		**129**
1.4.1. Redacción de informes quincenales		8
1.4.2. Redacción de la memoria	1.2, 1.3	100
1.4.3. Entrega de la memoria	1.4.2	0
1.4.4. Preparación de la presentación oral	1.4.2	20
1.4.5. Presentación oral	1.4.4	1

Tabla 2. Desglose del coste temporal de las tareas no presenciales

En la Figura 8 se pueden observar tareas presenciales en un diagrama de Gantt[2], donde se muestra la duración y la disposición en el tiempo de cada una.

[2] El diagrama de Gantt es una herramienta gráfica cuyo objetivo es mostrar el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado.

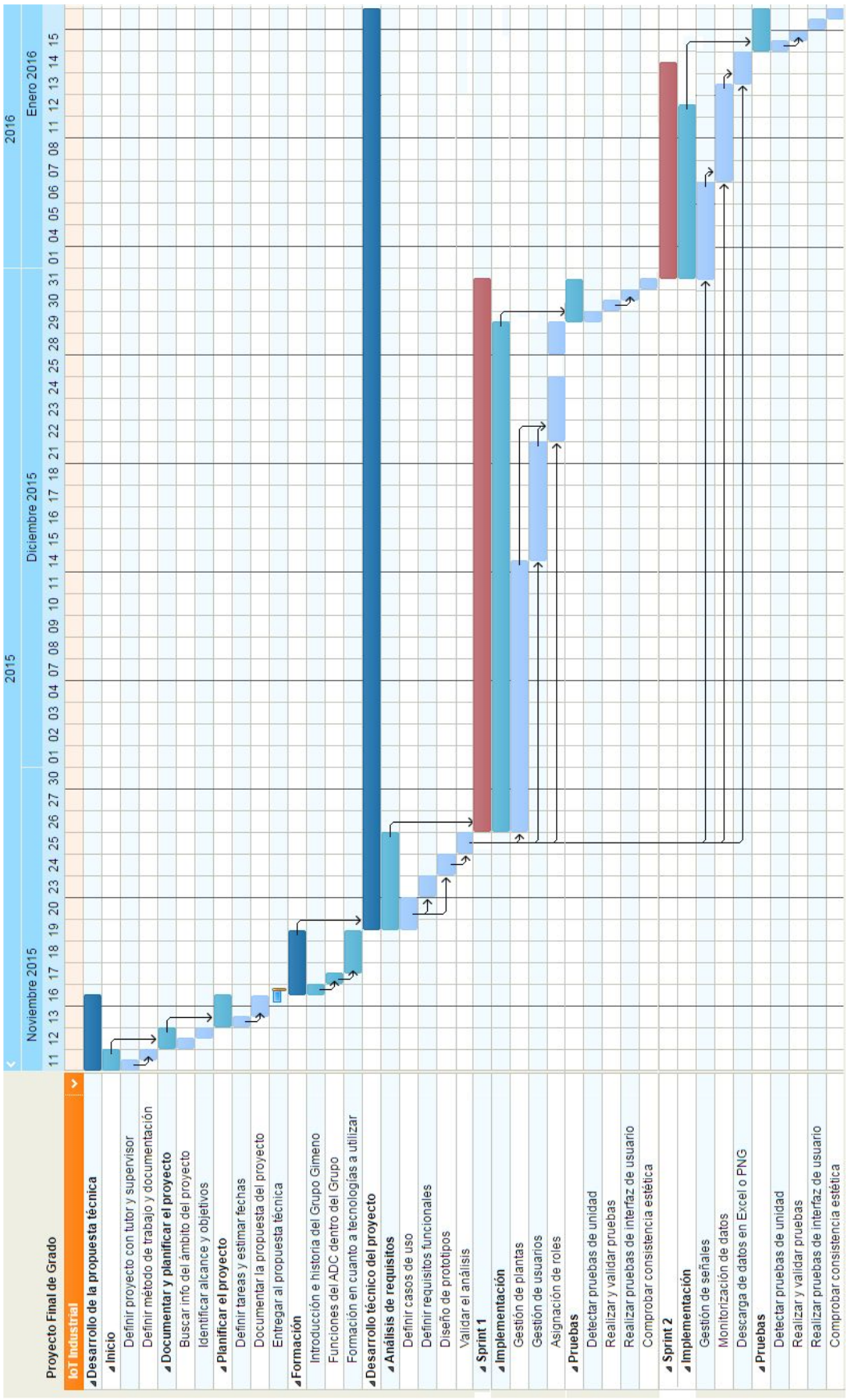

Figura 8. Diagrama de Gantt

Como se ha comentado en el apartado anterior, las tareas que se exponen en esta planificación son las típicas de un desarrollo con un ciclo de vida clásico, aunque el proyecto se ha desarrollado mediante una metodología ágil. Inicialmente se ha realizado un análisis de los requisitos de lo aplicación y, en cada iteración de SCRUM, se ha llevado a cabo el diseño e implementación y pruebas de los módulos incluidos en ese periodo. Es decir, las historias de usuario escogidas de la pila del producto para esa iteración.

2.3 Estimación de recursos

En este apartado se trata de calcular el coste del proyecto en un entorno real, como si el estudiante fuera un trabajador y la empresa tuviera que tener en cuenta los costes asociados a las horas de trabajo.

2.3.1 Modelo de estimación LDC

Las líneas de código (LDC) miden en forma directa el tamaño del producto de software. Se calculan contando las instrucciones de código fuente de cada elemento del producto de software excluyendo los comentarios [2].

Descomposición en paquetes de trabajo

- Gestión de plantas (GP)
- Gestión de usuarios (GU)
- Asignación de roles (AR)
- Gestión de señales (GS)
- Monitorización de datos (MD)
- Descarga de datos en formato Excel o PNG (DD)

En base a datos históricos obtenemos las líneas de código esperadas suponiendo que vamos a desarrollar el proyecto mayoritariamente en los lenguajes JAVA (para la programación *back-end*[3]) y JavaScript (para la programación *front-end*[4]). El valor esperado de las líneas de código necesarias se calcula como:

$$E = \frac{O + (4 * MP) + P}{6}$$

[3] Es la parte del software que permite al usuario interactuar con la aplicación. En este proyecto, el front-end se corresponde con las interfaces del sitio web desarrollando.

[4] El back-end es la parte que procesa la entrada desde el front-end. En este proyecto, se trata del software que se ejecuta en el lado del servidor y se encarga de la parte lógica de la aplicación, recibe peticiones del cliente y se comunica con la capa de almacenamiento de datos.

Donde E es el valor esperado, O es el optimista, MP es el más probable y P es el pesimista.

	Optimista	Más probable	Pesimista	Esperado
GP	550	600	700	608.33
GU	550	600	700	608.33
AR	300	400	500	400
GS	500	600	700	600
MD	800	900	1200	933.33
DD	150	250	350	250
Total				**3399.99**

Tabla 3. Estimación en LDC del coste del proyecto

Así pues, en la Tabla 3 se recogen las estimaciones de líneas de código de los módulos a implementar y del total del proyecto.

2.3.2 Modelo constructivo de costes (COCOMO)

El modelo COCOMO permite aproximar de forma matemática los costes totales del proyecto. Utilizando el modelo COCOMO básico se calcula el esfuerzo y el coste del proyecto en función de las LDC estimadas anteriormente, en este caso medidas en miles (KLDC):

$$KLDC = 3.399$$

Elegimos el tipo orgánico puesto que el proyecto no supera las 50 KLDC y, por tanto, es el más apropiado en este caso para obtener las variables a, b, c y d del modelo COCOMO según la Tabla 4.

Proyecto	a	b	c	d
Orgánico	2.4	1.05	2.5	0.38
Semiacoplado	3.0	1.12	2.5	0.35
Empotrado	3.6	1.20	2.5	0.32

Tabla 4. Tipos de proyecto y sus variables

E = Esfuerzo por persona y mes

$$E = (a) \times (KLDC)^b = 2.4 \times 3.399^{1.05} \simeq 8.67 \; personas/mes$$

D = Tiempo de desarrollo en meses

$$D = (c) \times (E)^d = 2.5 \times (8.67)^{0.38} = 5.68 \; meses$$

Como el tiempo de desarrollo estaba limitado por el periodo de la estancia (aproximadamente 2 meses) se ha calculado el número de personas necesarias para llevarlo a cabo.

CosteH = Personas necesarias

$$CosteH = E \; / \; D = 8.67 \; / \; 2 \; meses = 4.335 \; personas$$

Según las previsiones serían necesarias 4.335 personas durante 2 meses. Por supuesto las personas no son divisibles, el objetivo de estos cálculos es aproximar los recursos necesarios basados en datos estadísticos. Como consecuencia, según la previsión sería necesario que trabajaran el supervisor, el alumno y dos programadores junior.

A continuación se muestra qué profesionales se necesitan, así como su salario neto estimado mensualmente:

- Analista (Supervisor): 1.200 €/mes
- Programador Junior: 1.000 €/mes

Con un salario medio neto de 1.050 €/mes por cada trabajador que forma parte del grupo de desarrollo del proyecto y suponiendo un 30% de gastos que ocasionan los contratantes, obtendremos el siguiente coste del proyecto aproximado (CosteM):

$$CosteM = CosteH \times salario\_medio \times meses \times 1.30$$

$$CosteM = 4 \times 1050 \times 2 \times 1.30 = 10920 \; euros$$

Debido a su simplicidad no hay grandes costes planificados fuera de los estrictamente asignados a recursos humanos (licencias de software o equipo informático). Por supuesto este es un coste simbólico ya que el trabajo se completará durante la estancia en prácticas, pero ayuda a conocer el alcance y la envergadura del proyecto.

Capítulo 3

Tecnologías utilizadas

Contenidos

En este apartado se tratan las tecnologías utilizadas durante el desarrollo de la aplicación, así como algunos estándares y paradigmas adoptados por la empresa. Además, se

presentan algunas otras herramientas orientadas a facilitar la planificación y el seguimiento del proyecto.

La mayoría de las tecnologías utilizadas propias de la implementación de la aplicación han sido impuestas por la empresa, pues son las utilizadas en todas las aplicaciones que constituyen la plataforma IoTsens. Utilizar tecnologías y estándares comunes a todo un grupo de aplicaciones favorece la consistencia de la arquitectura definida, así como la reducción de la curva de aprendizaje para desarrollar futuras aplicaciones.

3.1 Planificación y seguimiento

Como se ha comentado en el apartado "Metodología de trabajo" del capítulo "Planificación del proyecto", a lo largo de la aplicación desarrollada se han seguido las reglas que define la metodología ágil SCRUM. Entre otras cosas, esto implica un continuo seguimiento por parte del *Scrum Master* del estado del proyecto.

Con el fin de llevar a cabo esta tarea, además de para tener en todo momento una idea clara del progreso y definir las funcionalidades requeridas por el cliente o *historias de usuario*, se ha utilizado Jira, una completa herramienta que permite el seguimiento y la planificación de tareas fácilmente mediante un tablero Kanban virtual. Algunas alternativas gratuitas podrían ser *Trello*, *Aha!* o *Taskworld*, pero la empresa dispone de una licencia completa de herramientas de desarrollo de software de la compañía Atlassian, donde se incluye Jira.

Por otro lado, y a modo de apoyo informativo, la empresa cuenta con una elaborada *Wiki* integrada en el software *Confluence* (también de Atlassian) donde poder consultar información en cualquier momento a nivel interno. En ella se detallan algunas de las particularidades de las tecnologías utilizadas para el desarrollo de proyectos, información relevante acerca del diseño de la arquitectura o incluso algunos ejemplos de utilización de librerías propias de la empresa.

3.1.1 Jira

Jira es una herramienta web de la compañía Atlassian que permite al usuario gestionar las tareas de un proyecto de cualquier ámbito. Dicha herramienta está fuertemente enfocada a proyectos que se llevan a cabo utilizando metodologías ágiles, pues su página principal es un tablero Kanban donde el usuario puede crear tareas y, según avanza el progreso para completarlas, clasificarlas según su estado. A lo largo de este proyecto se han considerado cuatro estados en los que poder clasificar las diferentes tareas incluidas en un *sprint* activo:

- *Por hacer*. Es el estado inicial de una tarea en cuanto se inicia el *sprint*. Una tarea permanece en este estado hasta que alguien decida comenzar a implementar la funcionalidad que representa.

- *En progreso*. Cuando una tarjeta Jira está en esta columna, quiere decir que ya ha sido comenzada por algún usuario, pero que todavía no se ha dado por terminada. Es habitual que no haya más de una tarea en este estado por usuario en un instante determinado.

- *En pruebas*. Es un estado intermedio entre "En progreso" y "Hecho". Los usuarios del *Team Scrum* que finalizan una tarea pueden colocarla en esta columna, pero nunca en "Hecho". Será el *Scrum Manager* quien pueda dar como terminada una tarea definitivamente pasándola al estado "Hecho".

- *Hecho*. El *Scrum Manager* tiene asignada la responsabilidad de "resolver" incidencias, es decir, dar tareas por terminadas una vez ha comprobado que la implementación de la funcionalidad del producto que representan es satisfactoria.

Además, Jira permite al usuario crear *sprints*, definir su fecha de inicio y de fin y asignar tareas a cada uno de ellos (en el caso de este proyecto esta tarea es también responsabilidad del *Scrum Manager*). Así, cuando el proyecto se encuentra en un determinado *sprint*, en el tablero *Kanban* únicamente aparecen las tarjetas que simbolizan funcionalidades a implementar en el *sprint* activo.

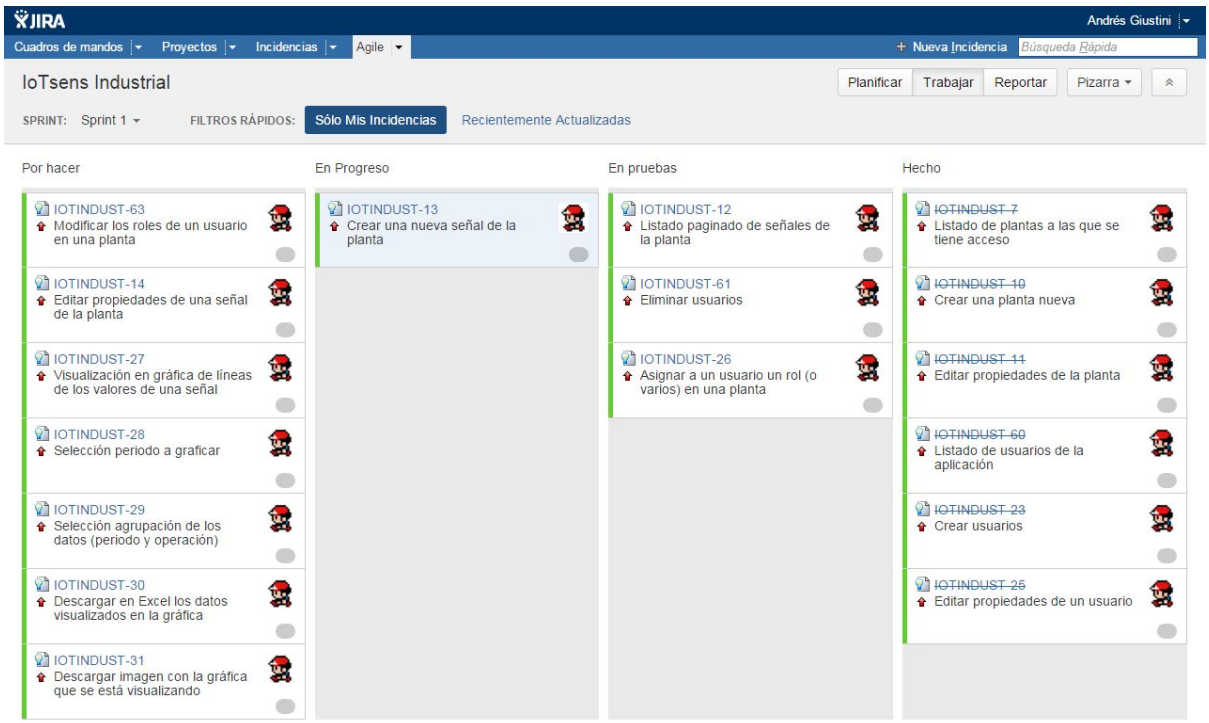

Figura 9. Estado del tablón de JIRA al finalizar la cuarta semana de prácticas

En la Figura 9 se ilustra el estado del tablero Kanban de Jira en el *sprint* 1, concretamente al finalizar la cuarta semana de trabajo. En ella se pueden apreciar las historias completadas y validadas por parte del *Scrum Manager* (columna "Hecho"), las

historias del *sprint* terminadas pero todavía sin validar (columna “En pruebas”) y la tarea que estaba en ese instante en desarrollo.

Cabe destacar que las tarjetas Jira pueden representar tanto historias de usuario (funcionalidades), como defectos (modificaciones a historias de *sprints* anteriores que han resultado no desempeñar correctamente su función) o mejoras (extensiones impuestas por el cliente a tareas de *sprints* anteriores), y que cada una de las historias pueden contener sub-historias, en caso de que la funcionalidad de la tarea padre sea muy compleja.

3.1.2 Wiki

Se conoce como *wikis* a las páginas web de carácter informativo, directamente editables desde el navegador y, por tanto, donde son los propios usuarios los creadores de contenido. Una de las principales ventajas de una wiki es, pues, que permite crear y mejorar las páginas de forma inmediata, dando una gran libertad al usuario, y por medio de una interfaz muy simple. Esto hace que más gente participe en su modificación y comparta sus conocimientos, a diferencia de los sistemas tradicionales, donde resulta más difícil que los usuarios del sitio contribuyan a mejorarlo.

La *wiki* de la empresa ha resultado muy útil especialmente durante la fase de formación, donde era menester fijar muchos conceptos muy rápidamente. Además, tener esta herramienta de apoyo disponible en cualquier momento ha minimizado considerablemente las interrupciones en el desarrollo debidas al desconocimiento de estándares y librerías internas.

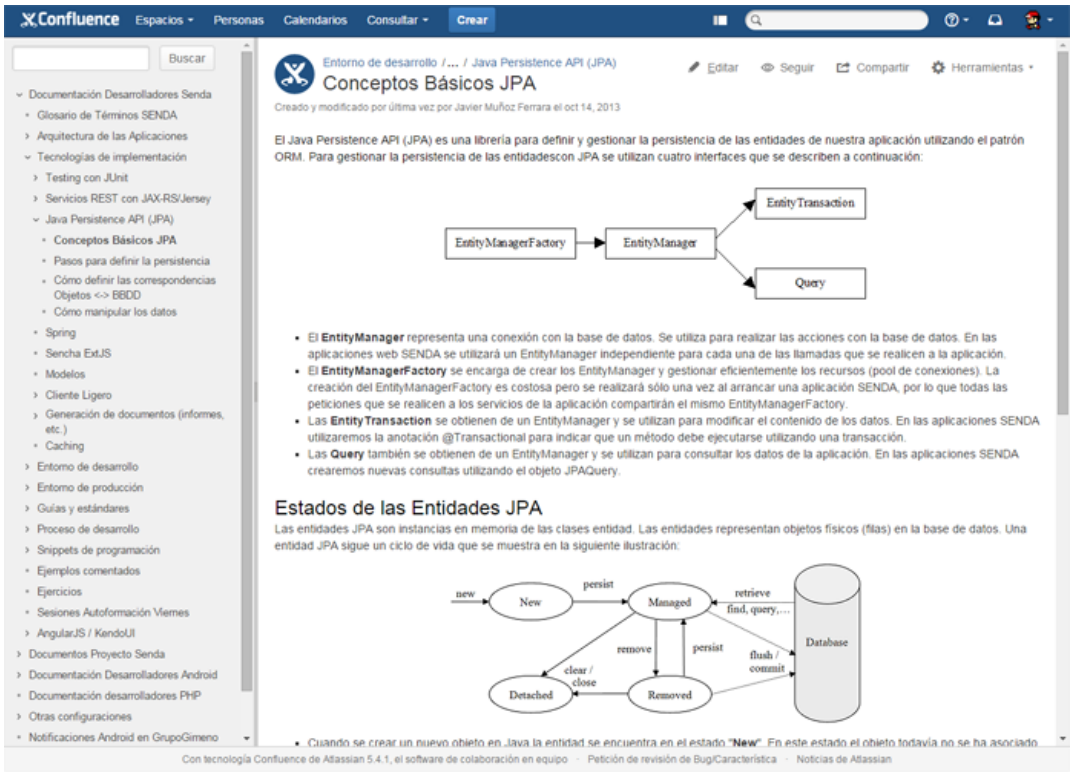

Figura 10. Ejemplo de artículo en la *Wiki* de la empresa

En la Figura 10 se puede observar una de las páginas de la *wiki* de la empresa, concretamente un artículo informativo sobre *JPA*.

3.2 Estándares y paradigmas

A lo largo del desarrollo del proyecto se han utilizado algunos estándares y paradigmas que son habituales en la implementación de aplicaciones web en la empresa, los cuales se detallan en esta sección.

3.2.1 REST

Representational State Transfer (REST) es una arquitectura software que define la comunicación distribuida entre un cliente y un servidor. Posee una arquitectura de cliente-servidor, ya que cada petición HTTP es independiente y ni el cliente ni el servidor almacenan ningún estado. Esta comunicación se realiza mediante los métodos GET, POST, PUT y DELETE intercambiando elementos de información llamados recursos.

Una tecnología alternativa a REST podría ser RPC (Remote Procedure Call), donde se invoca a un método que se ejecuta en una máquina remota. La ventaja de REST respecto a RPC es el mínimo acoplamiento, ya que en RPC el cliente necesita saber la signatura del método

que va a invocar mientras que en REST cada recurso es identificado por una URI única y, mediante los métodos estándar de HTTP mencionados anteriormente, puede realizar cualquier acción que se desee en el servidor.

En este proyecto, los recursos que se intercambian se codifican en formato JSON (JavaScript Object Notation), que sólo admite tipos básicos. Este formato es menos extensible respecto al formato XML pero superior en interoperabilidad.

A continuación, se muestra un ejemplo de petición HTTP al API REST existente en la aplicación que se va a desarrollar:

Imaginemos que un usuario accede a la pestaña "Usuarios" dentro de una planta industrial concreta. En ese momento la aplicación necesita obtener dichos usuarios para poder mostrarlos, en este caso, en un listado. La secuencia de acciones que se llevan a cabo es la siguiente:

1. La aplicación cliente realiza una petición GET a la URL correspondiente del servicio REST que proporciona el servidor. Según los principios REST, la URL en cuestión debe contener el *path* de las plantas industriales, un identificador de la planta, y el *path* de los usuarios. En este caso, la URL sería así:

   http://dominiodelservidor.com/services/industrialplants/codigoPlanta/users

2. El servidor recibe la petición, realiza las consultas y cálculos pertinentes y devuelve el recurso solicitado en formato *JSON*[5]:


```
{
    "success": true,
    "data": [{
        "id": 77,
        "phone": "",
        "email": "agiustini@grupogimeno.com",
        "name": "Andrés Giustini",
        "userName": "andres.giustini"
    }, {
        "id": 78,
        "phone": "600123123",
        "email": "otro-usuario@grupogimeno.com",
        "name": "Otro usuario",
        "userName": "otro.usuario"
    }, {
        "id": 76,
        "phone": "600321321",
        "email": "un-usuario-mas@grupogimeno.com",
        "name": "Un usuario más",
        "userName": "un-usuario-mas"
    }]
}
```


Figura 11. Ejemplo de respuesta en formato *JSON*

3. El cliente obtiene la información deseada y la muestra por pantalla en el listado de usuarios de la planta.

[5] *JavaScript Object Notation*, es un formato ligero para el intercambio de datos. *JSON* es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.

En este proyecto se ha utilizado el estándar *JAX-RS*, y más concretamente la implementación de referencia *Jersey*, que provee de anotaciones Java en aplicaciones *RESTful* para la construcción de servicios web. En el ejemplo anterior, el servicio que se encuentra en la URL mencionada se define como se indica en la Figura 12.

```
@GET
@Path("services/industrialplants/{plantCode}/users")
@Produces(MediaType.APPLICATION_JSON)
public List<User> getPlantUsers(@PathParam("plantCode") String plantCode) {

    // ... Operaciones para obtener y devolver la lista de usuarios ...

}
```

Figura 12. Definición de un recurso con *Jersey*

Mediante las anotaciones se define la URL o *Path* del servicio y el tipo de operación que recibirá (GET en este caso). Además, se puede indicar el tipo de respuesta que producirá.

En este proyecto se han utilizado algunas otras anotaciones para la definición de los recursos y la transferencias de datos desde el cliente, entre las que destacan las siguientes:

- *@InjectParam*. Jersey utiliza su propio mecanismo de inyección de dependencias sobre los recursos. Con esta anotación se indica que Jersey debe inyectar un objeto del tipo requerido.

- *@PathParam*. Se utiliza para indicar los parámetros que vienen en la propia URL.

- *@QueryParam*. Además de los parámetros en la ruta, un servicio REST puede recibir también parámetros *query* definidos en la petición HTTP. Para indicar que un servicio puede recibir parámetros por *query* hay que etiquetar uno de los parámetros del método con la anotación *@QueryParam("nombreDelParámetro")*. Además, se puede utilizar la anotación *@DefaultValue("valorPorDefecto")* para indicar el valor que tomará la variable en caso de no recibir el parámetro por *query*.

- *@FormParam*. En el caso de acciones que requieren el envío de información adjunta (POST/PUT).

3.2.2 AJAX

En el método tradicional de petición-respuesta entre cliente y servidor, el cliente solicitaba una página completa ya lista para mostrar. *AJAX* (*Asynchronous JavaScript And XML*) es una técnica de desarrollo web para crear aplicaciones interactivas o *Rich Internet*

Applications. Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.

En este proyecto se ha utilizado esta técnica para obtener datos del servidor y únicamente refrescar dichos datos en caso de que sea necesario, en lugar de tener que actualizar la página completa.

3.2.3 TDD

Durante la implementación del *backend* de la aplicación, en ocasiones ha resultado interesante el uso de TDD (*Test driven development* o desarrollo guiado por pruebas). TDD es una práctica de ingeniería del software que involucra crear pruebas de software antes incluso de desarrollar dicho software, para luego implementar la funcionalidad deseada y refactorizar.

Esta práctica se ha llevado a cabo, por ejemplo, para los métodos de algunos servicios que necesitan obtener datos de la base de datos del sistema. De esta manera, se sabe *a priori* cuál debe ser el resultado de la consulta y se actúa en consecuencia.

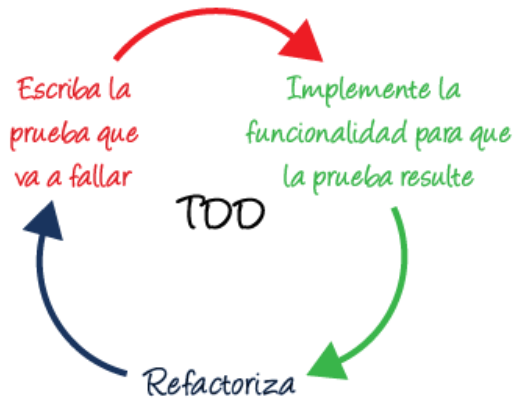

Figura 13. Ciclo de desarrollo guiado por pruebas [3]

Este tipo de metodología supone unos elevados requisitos iniciales, por lo que sólo se ha utilizado en las ocasiones donde además de permitir una visión temprana de ciertos requisitos, se conseguía proporcionar un valor añadido a la creación del software del proyecto, resultando una aplicación de más calidad y más susceptible a ser mantenida en el futuro.

3.3 Herramientas de desarrollo

En esta sección se presentan las principales herramientas de desarrollo utilizadas para facilitar el proceso de implementación de la aplicación.

3.3.1 IntelliJ

Como *ambiente de desarrollo integrado* o IDE se ha utilizado IntelliJ IDEA en su versión 13. Este potente entorno de trabajo ha resultado ser muy cómodo tanto para escribir, depurar, refactorizar o probar código. Algunas de sus herramientas como el *Data flow analysis*, el autocompletado de código y sugerencias, el *VCS* (sistema de control de versiones) o la navegación entre métodos han favorecido considerablemente la productividad.

El lenguaje utilizado en el lado del servidor ha sido Java, concretamente *Java Development Kit* en su versión 7 (JDK7).

3.3.2 Spring

Spring es un framework para el desarrollo de aplicaciones y contenedor de inyección de dependencias de código abierto para la plataforma Java. *Spring* cuenta con una serie de módulos, cada uno de los cuales ofrece una serie de bibliotecas o funcionalidades, como pueden ser *Struts*, *JDBC*, *JPA* o *Hibernate*.

En este proyecto se ha utilizado *Spring*, pues es el *framework* que se utiliza para la mayoría de proyectos de IoTsens.

3.3.3 Subversion

Para llevar a cabo el control de versiones de la aplicación se ha utilizado *Subversion*. Los sistemas de control de versiones permiten compartir el código fuente y todos los recursos que componen un proyecto entre los desarrolladores que lo están llevando a cabo, y tener versiones actualizadas en todo momento. Además, mantienen el alojamiento de dicho proyecto en una máquina remota, evitando así pérdidas de información si se producen problemas en las máquinas de los programadores, y conservando un registro histórico de los cambios realizados por los diferentes componentes del equipo de trabajo.

3.3.4 Acceso a datos

Como SGBD (Sistema gestor de bases de datos) se ha utilizado MySQL, con la completa interfaz de administración que ofrece *phpMyAdmin*. Desde la aplicación, las consultas a la base de datos se han realizado utilizando *QueryDSL* con *Hibernate*. A nivel de programación, las únicas clases que han operado directamente con la base de datos han sido los DAO (*Data Access Object* u objeto de acceso a datos), que a su vez suministran dichos datos a los servicios web para que estén disponibles en forma de recursos para el cliente.

La ventaja de usar objetos de acceso a datos es que cualquier objeto de negocio (aquel que contiene detalles específicos de operación o aplicación) no requiere conocimiento directo del destino final de la información que manipula, por lo que se mejora la abstracción. Esto supone, entre otras cosas, que la tecnología utilizada para el acceso a datos puede ser actualizada o cambiada sin cambiar otras partes de la aplicación.

3.3.5 Apache Maven

Maven es una herramienta para la gestión y la construcción de proyectos software escritos en lenguaje Java. Esta herramienta facilita algunos procesos en el desarrollo como la resolución de dependencias o el nombrado de versiones. Además, permite la gestión de perfiles como podrían ser *producción* o *desarrollo*. De esta forma, se puede disponer de una configuración distinta para cada perfil.

Toda la configuración del proyecto referente a Maven, ya sean perfiles, dependencias u otros se encuentra en un fichero XML (pom.xml) que se encuentra alojado en el directorio raíz del proyecto. En esta ocasión, se ha utilizado el fichero de configuración de Maven para incluir las dependencias del proyecto como puede ser el modelo de datos de IoTsens, un cliente de IoTsens que referencia a los servicios web del mismo[6] o bibliotecas como *JUnit* o *Mockito*, pero también para incluir ciertos *plugins* como *QueryDSL* o *Jetty*.

3.4 Tecnologías web en el servidor

Puesto que el proyecto a desarrollar consta de una aplicación web, se han utilizado muchas otras tecnologías que facilitan la creación de sitios web así como de páginas dinámicas. En este apartado se presenta una librería para el manejo de plantillas HTML de forma sencilla incluida en *Spring*: *Thymeleaf*.

[6] Más adelante se explica la arquitectura de IoTsens y cómo afecta dicha arquitectura a este proyecto.

3.4.1 Thymeleaf

Thymeleaf es una librería Java que implementa un motor de plantillas de XML/XHTML/HTML5 (también extensible a otros formatos) que puede ser utilizado tanto en modo web como en otros entornos no web.

Thymeleaf permite "publicar" recursos que serán accesibles por las plantillas HTML, como puede ser el nombre del usuario *loggeado* o la planta industrial en la que se encuentra el usuario. En la Figura 14 se ilustra el uso de *Thymeleaf* para que el servidor comunique al cliente quién es el usuario *loggeado*.

```
@GET
@Path("/")
@Produces(MediaType.TEXT_HTML)
public Template index() throws Throwable {

    Template template = getTemplate("index.html");

    User user = permissionServices.getLoggedUser();

    template.put("user", user);

    return template;
}
```

Figura 14. Ejemplo de utilización de *Thymeleaf* en el servidor

En este recurso se define que cuando el cliente haga una petición queriendo obtener el archivo "*index.html*", se publicará en dicha plantilla HTML el usuario autenticado. Como se puede observar, dicho usuario se obtiene mediante uno de los servicios de la aplicación, contenido en el objeto "*permissionServices*". De esta manera, en el fichero HTML se puede acceder a este recurso mediante las anotaciones "th:" de *Thymeleaf*, como se indica en la Figura 15.

```
<li th:if="${isAdmin == true}" th:class=" ${section == 'USERS'} ? active ">
    <a href="/usuarios"><i class="fi-torsos size-60"></i>Usuarios</a>
</li>
<li class="divider"></li>
<li>
<a href="/j_spring_cas_security_logout" class="logout">
    <i class="fi-power size-60"></i>Desconectar<br/>
        <span class="user-info">
            <span th:utext="${user.name}"></span>
                (<span th:utext="${user.userName}"></span>)
        </span>
    </a>
</li>
```

Figura 15. Ejemplo de utilización de *Thymeleaf* en el cliente

El resultado de acceder al recurso desde la plantilla, y que es lo que el usuario final ve por pantalla, es el que se muestra en la Figura 16.

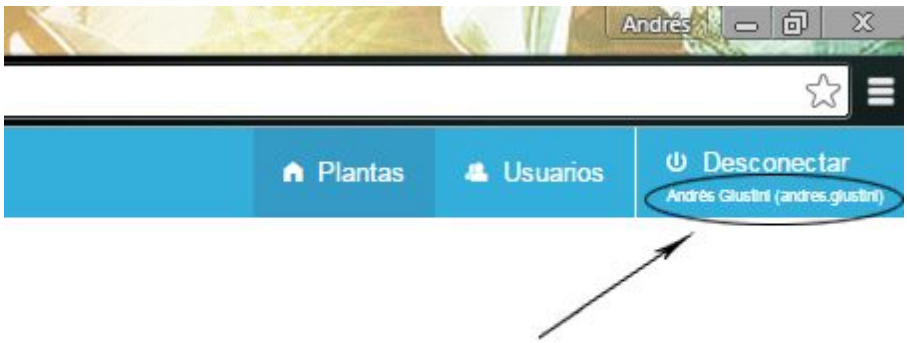

Figura 16. Resultado de utilización de *Thymeleaf*

Como se puede observar, se trata de una herramienta muy potente y las posibilidades que ofrece son muy a tener en cuenta.

3.5 Tecnologías web en el cliente

Durante la implementación de este proyecto se han utilizado muchas tecnologías y herramientas para la creación de páginas web, de las cuales las más significativas se enumeran en esta sección.

3.5.1 HTML5

HTML5 es la quinta revisión importante del lenguaje *HTML* (*HyperText Markup Language* o lenguaje de marcas de hipertexto), y que se ha convertido en un estándar web avalado por la W3C (*World Wide Web Consortium*), una organización nacida en 1994 cuya función principal es la de estandarizar las tecnologías que rigen internet.

En este proyecto se ha utilizado *HTML5* como lenguaje de marcas para implementar las plantillas que se mostrarán en el navegador web.

Figura 17. Logotipos de HTML5, CSS3 y JavaScript

3.5.2 CSS3

CSS3 constituye también un estándar como lenguaje para definir la presentación de un documento estructurado escrito en *HTML*. Una de las principales ventajas de utilizar *CSS* (*Cascading Style Sheets* u hojas de estilo en cascada) es la separación de la definición de un documento de su presentación, mejorando el mantenimiento de un sitio web.

Para el desarrollo de la parte cliente de la aplicación se han utilizado HTML5 y CSS3 junto con *Foundation*.

3.5.3 Foundation

Foundation es un framework de interfaz de usuario de código abierto. Foundation proporciona un *grid* o cuadrícula *responsive*, además de componentes de interfaz de usuario HTML y CSS, plantillas, y fragmentos de código, incluyendo tipografía, formularios, botones, barras de navegación y muchos otros componentes.

Además de la cuadrícula, en el proyecto han resultado imprescindibles el menú de navegación, los *foundation icons* y las denominadas *responsive-tables*.

3.5.4 JavaScript

JavaScript es el lenguaje de programación más ámpliamente utilizado en la web. Es un lenguaje interpretado, dialecto del estándar *ECMAScript*[7], que se define como orientado a objetos, basado en prototipos, imperativo, dinámico y débilmente tipado.

[7] *ECMAScript* es una especificación de lenguaje de programación publicada por *ECMA International*, que define un lenguaje de tipos dinámicos ligeramente inspirado en Java y soporta algunas características de la programación orientada a objetos mediante objetos basados en prototipos y pseudoclases.

En este proyecto se ha utilizado para dotar a las páginas web de dinamismo, aumentando la interacción con el usuario y dando una continua sensación de fluidez en cada una de las acciones que se realizan.

Para el desarrollo de la parte del cliente con *JavaScript*, han sido de gran ayuda algunas librerías que se detallan a continuación:

- *RequireJS*. Es un cargador de módulos de *JavaScript*, por lo que utilizándolo se evitan las preocupaciones que generan las dependencias en proyectos de gran envergadura. *RequireJS* hace uso de un fichero "*dependencies.js*" donde el programador referencia la ubicación de los módulos que se cargarán en las distintas páginas. En la Figura 18 se muestra un fragmento de un fichero de dependencias de *RequireJS*.

Figura 18. Fragmento de fichero de dependencias de *RequireJS*

De esta forma, en cada fichero *JS* que implemente el programador, se podrá definir fácilmente qué módulos cargar, como se muestra en la Figura 19.

Figura 19. Ejemplo de carga de módulos por parte de *RequireJS* y su utilización

- *jQuery*. Es una biblioteca que simplifica considerablemente la manera de interactuar con el documento *HTML*, manipular el *DOM*[8], manejar eventos, etc. Además, en este proyecto se han utilizado los métodos de *jQuery* que facilitan las peticiones *AJAX* para obtener datos del servidor.

- *jQuery UI*. Se puede considerar una extensión de *jQuery* que añade un conjunto de componentes, *widgets* y efectos visuales. En el proyecto se ha utilizado, por ejemplo, para la ordenación de las señales en el listado de señales, como se muestra en la Figura 20.

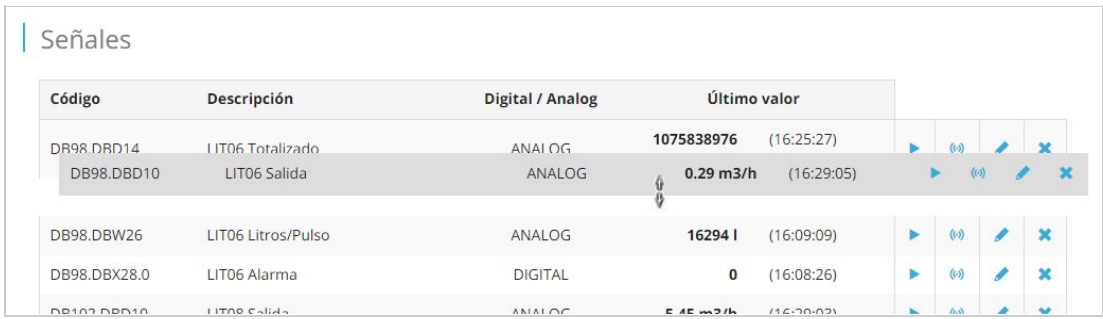

Figura 20. Ordenación de las filas de una tabla con *jQuery UI*

- *Moment.js*. Completa librería para análisis, validación, manipulación y formateo de fechas.

- *Handlebars*. Se trata de un *template system* basado en *Mustache*, que sirve para generar *HTML* a partir de objetos con datos en formato *JSON*. *Handlebars* ha sido muy útil para, por ejemplo, actualizar los datos de los listados en las diferentes pantallas de la aplicación cuando se realizan operaciones sobre ellos que implican peticiones *AJAX*.

- *Highcharts*. Es una potente librería para generar distintos tipos de gráficas dinámicas e interactivas. En este proyecto se ha utilizado para mostrar de manera gráfica las medidas obtenidas de las diferentes señales.

[8] El *Document Object Model* es esencialmente una interfaz de plataforma que proporciona un conjunto estándar de objetos para representar documentos *HTML*. El *DOM* permite el acceso dinámico a través de la programación para acceder, añadir y cambiar dinámicamente contenido estructurado en documentos a través de lenguajes como *JavaScript*.

Capítulo 4

Análisis de requisitos

Contenidos

En este apartado se realiza un análisis detallado de los requisitos del sistema y se exponen las características que debe implementar. Todos los requisitos se acordaron con la empresa antes de empezar con el desarrollo del proyecto y se han utilizado como guía para llevarlo a cabo.

El principal objetivo del análisis es descomponer el proyecto en pequeños paquetes que permitan llevar a cabo un desarrollo más eficiente.

4.1 Historias de usuario

Las historias de usuario son uno de los pilares principales del desarrollo ágil de proyectos. Una historia de usuario es una definición de alto nivel de un requisito de software,

conteniendo la información necesaria para que los desarrolladores puedan estimar el esfuerzo necesario para implementarlo. La Tabla 5 contiene las historias de usuario recopiladas durante el proceso de definición de requisitos.

Identificador	Historia de usuario
US01	Como usuario quiero poder ver en un listado las plantas industriales presentes en el sistema
US02	Como administrador quiero poder añadir plantas industriales al sistema
US03	Como administrador quiero poder editar las propiedades de las plantas industriales presentes en el sistema
US04	Como usuario quiero poder ver en un listado los usuarios del sistema
US05	Como administrador quiero poder añadir usuarios al sistema
US06	Como administrador quiero poder editar los datos de los usuarios
US07	Como administrador quiero poder eliminar usuarios del sistema
US08	Como administrador de una planta quiero poder asignar roles a los usuarios para esa planta
US09	Como administrador de una planta quiero poder modificar los roles de los usuarios de esa planta
US10	Como usuario de una planta quiero poder acceder a un listado paginado de señales de esa planta
US11	Como administrador de una planta quiero poder añadir señales a esa planta
US12	Como administrador de una planta quiero poder editar las propiedades de las señales de esa planta
US13	Como usuario de una planta quiero poder visualizar en gráficas de líneas los valores de las señales de esa planta
US14	Como usuario quiero poder establecer el periodo a graficar al visualizar los valores de una señal
US15	Como usuario quiero poder visualizar los datos de una señal teniendo en cuenta un cierto tipo de agregación
US16	Como usuario quiero poder descargar en formato Excel los valores de una señal
US17	Como usuario quiero poder descargar en formato PNG la señal que estoy visualizando en la gráfica

Tabla 5. Definición de las historias de usuario

4.2 Diagrama de 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 [4]. Permite mostrar la relación que tienen los diferentes usuarios con cada caso de uso que se debe implementar. En la Figura 21 se muestra el diagrama de casos de uso

desarrollado para este proyecto, creado en los primeros días de trabajo tras la primera reunión con la empresa.

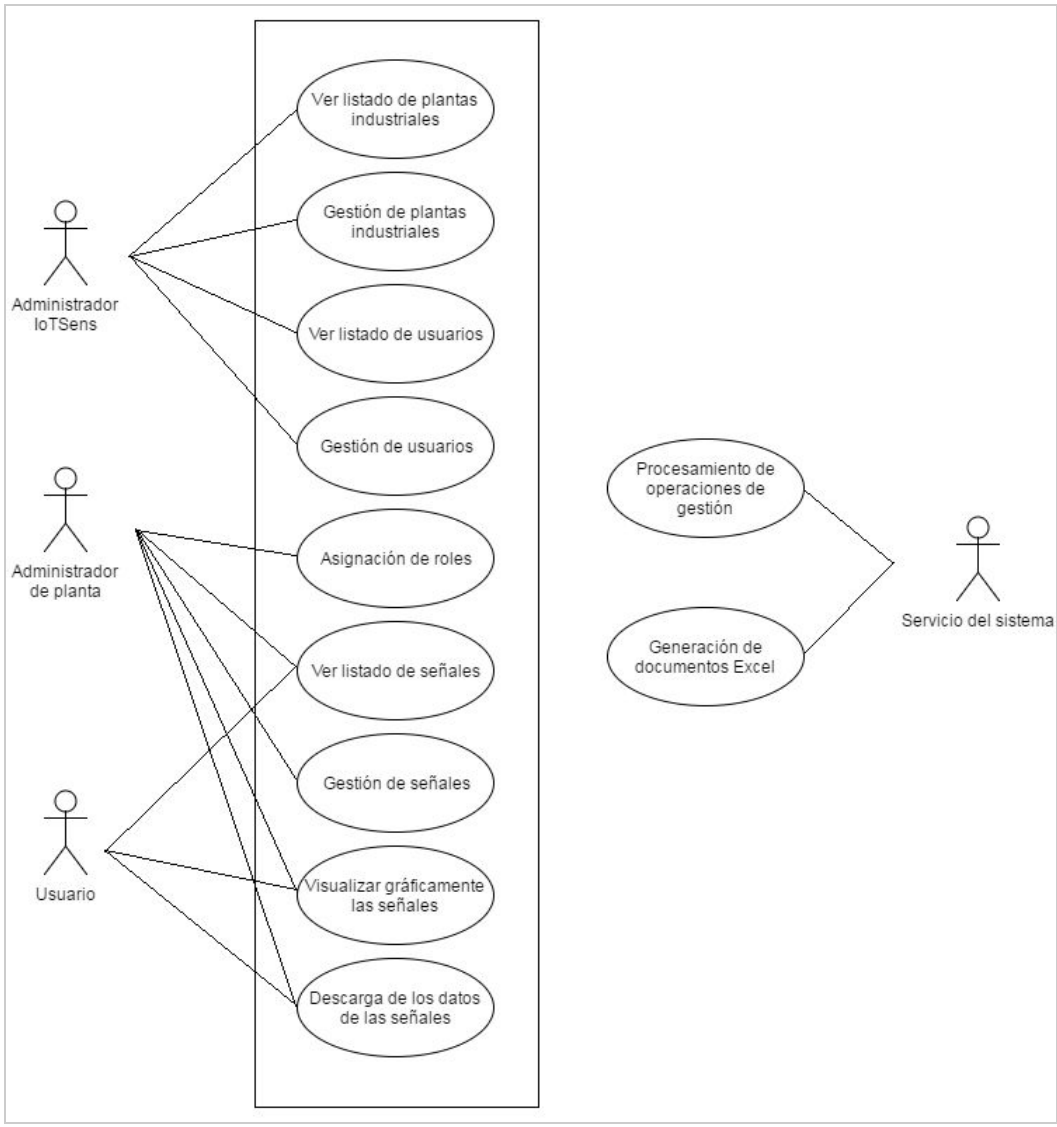

Figura 21. Diagrama de casos de uso

4.3 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.

En el Anexo A se muestran las tablas que definen estos requisitos. Cada una de ellas contiene un identificador, nombre, descripción y una secuencia de pasos que definen su ciclo de vida.

4.4 Requisitos de datos

Para satisfacer los requisitos de usuario expuestos anteriormente, es necesario definir las entidades y atributos que se van a almacenar, es decir, los requisitos de datos.

Los requisitos de datos del proyecto, obtenidos del diagrama de casos de uso y el documento de requisitos, se encuentran en el Anexo B. Dicha especificación está sujeta a los cambios que deban realizarse durante el proceso de implementación, y trata de dar una especificación de las entidades fundamentales que requiere el proyecto.

4.5 Diagrama de clases

Las entidades que se recogen en los requisitos de datos, así como las relaciones entre ellas, quedan reflejadas en el diagrama de clases de la Figura 22.

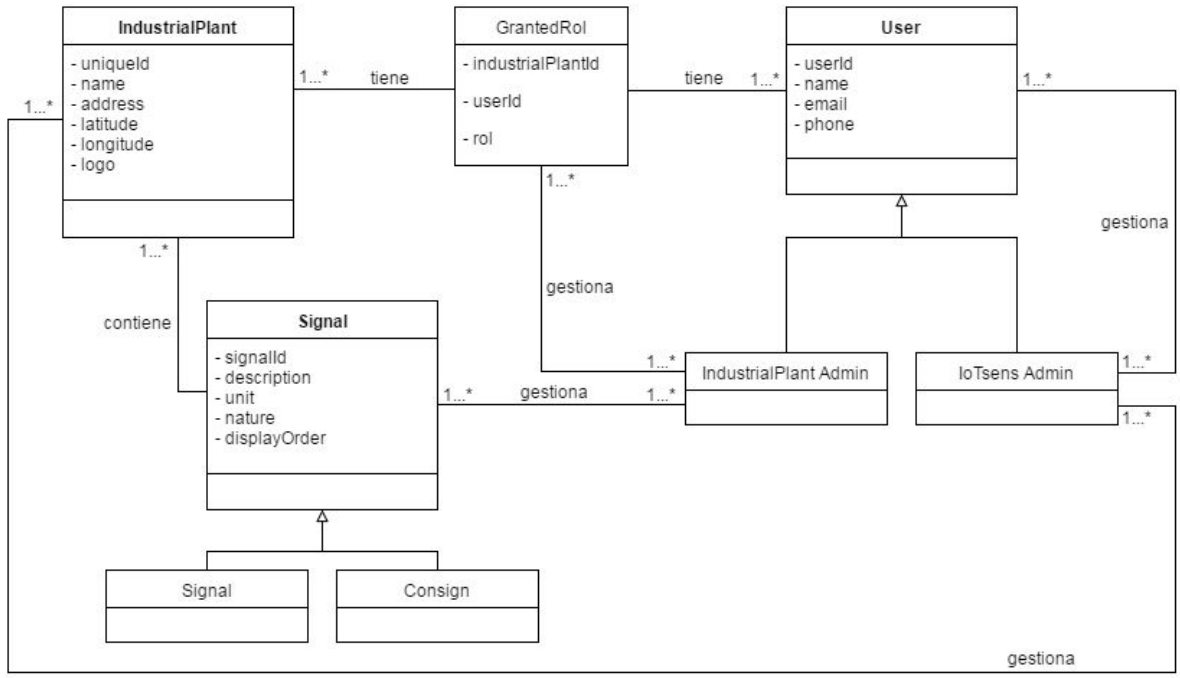

Figura 22. Diagrama de clases

Como algunos aspectos importantes a destacar, en el diagrama se puede ver que únicamente los administradores de IoTsens pueden gestionar las plantas y los usuarios, mientras que son los administradores de cada planta los que pueden llevar a cabo la gestión de señales y de roles en las mismas.

También se observa que cada una de las señales de una planta puede ser de entrada o de salida. Es decir, las señales se dividen a su vez en *Signals* y en *Consigns*, siendo las primeras los datos que se reciben de la planta y las segundas las órdenes que se envían hacia la planta.

En este proyecto han quedado definidos los dos tipos de señales pero, como se ha comentado en capítulos anteriores, la comunicación *aplicación - planta industrial* está pendiente de implementarse a nivel de arquitectura y, por lo tanto, las consignas tomarán relevancia más adelante.

4.6 Prototipos de la GUI

Para concluir el análisis de requisitos se han creado prototipos de cada una de las interfaces de usuario con las que deberá contar la aplicación definitiva. Además de para acordar con el cliente el aspecto final de las distintas pantallas, esta fase ha servido para hacerse una idea más detallada de cómo debería ser el aspecto de la aplicación y comprender mejor algunas de las funcionalidades que se implementarán en la fase de desarrollo.

Los prototipos ayudan a organizar las ideas y darles forma sin perder demasiado tiempo, ofreciendo una visión aproximada de lo que podría ser el diseño definitivo no tanto en precisión sino en la posición de los distintos controles, etcétera. Para su desarrollo se ha utilizado la herramienta de diseño *app.moqups.com*.

Por lo tanto, los prototipos no representan el aspecto definitivo de las pantallas, sino que su objetivo es plasmar una idea más visual de la aplicación para permitir la verificación de las funcionalidades por parte del cliente, sin entrar en aspectos de diseño.

A continuación se muestran los prototipos de las interfaces más significativas de la aplicación a desarrollar: la administración de plantas industriales (listar, añadir y editar), la administración de usuarios (listar, añadir y editar), la gestión de roles y, por último, la administración y visualización de señales.

4.6.1 Administración de plantas

Con el fin de poder administrar las plantas industriales del sistema, la aplicación mostrará las ya existentes mediante un listado donde aparecerá el código, el nombre, la dirección y las coordenadas de cada una. Además, si el usuario está registrado como administrador en IoTsens, el sistema ofrecerá la posibilidad de añadir nuevas plantas o editar las presentes, a través de un botón en la parte inferior del listado o un icono a la izquierda de cada planta, respectivamente. El siguiente prototipo muestra la interfaz de administración de plantas para un usuario administrador:

Figura 23. Prototipo de la pantalla de Administración de plantas

En caso de querer añadir una planta nueva se pulsará sobre el botón "Añadir", lo que hará aparecer un formulario en una ventana modal. El usuario únicamente deberá rellenar los campos solicitados y aceptar el envío del formulario:

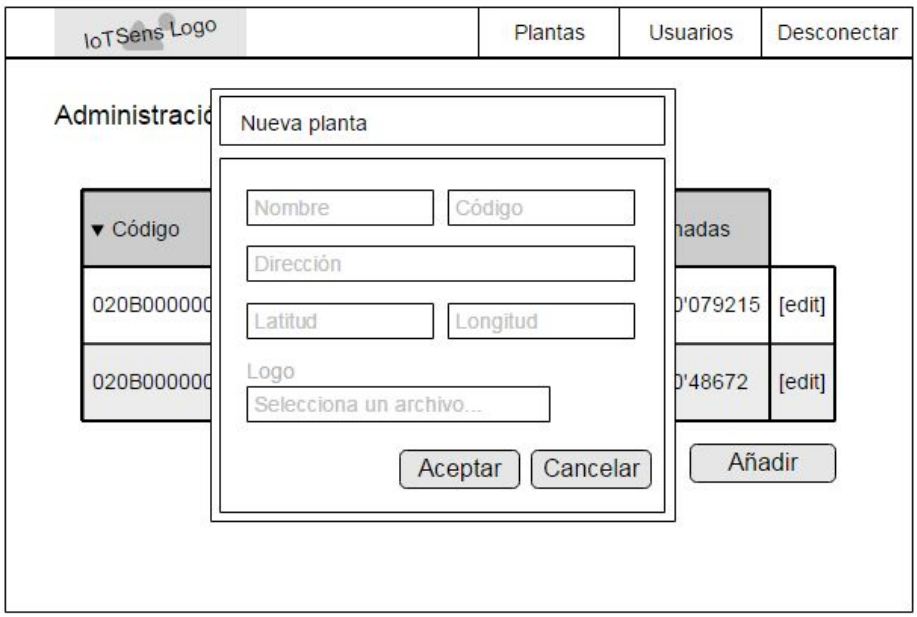

Figura 24. Prototipo del formulario para añadir plantas industriales al sistema

4.6.2 Administración de usuarios

La administración de usuarios es similar a la de plantas. A continuación, en la Figura 25, se muestra el prototipo del listado de usuarios en caso de que el usuario autenticado sea administrador.

Figura 25. Prototipo de la pantalla de Administración de usuarios

Análogamente a la funcionalidad de añadir plantas, cuando se desea añadir un usuario el sistema mostrará un formulario con los datos a rellenar del usuario a añadir. Cabe destacar que para dar de alta un usuario, éste debe formar parte de IoTsens. Por lo tanto, para añadirlo en IoTIndustrial únicamente será necesario introducir su nombre de usuario, pues el resto de datos se leerán de la base de datos de IoTsens.

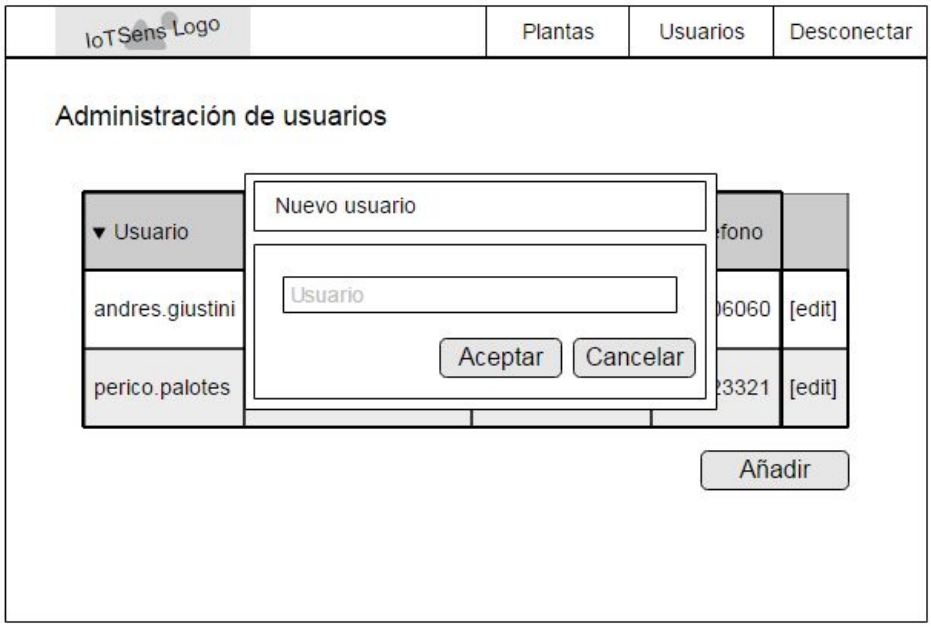

Figura 26. Prototipo del formulario para añadir usuarios al sistema

4.6.3 Gestión de roles

La interfaz para la gestión de roles constará de un listado con los usuarios y sus roles asignados a la planta en la que se encuentra el usuario. Por lo tanto para poder acceder a esta página, el usuario debe haber seleccionado previamente una de las plantas del listado de plantas expuesto anteriormente. Cuando el usuario se encuentra en una planta, la aplicación mostrará una segunda barra de navegación, a través de la cual poder navegar entre las secciones que contendrán las distintas funcionalidades de la propia planta.

Para añadir un usuario a una planta (o asignar roles a un usuario en una planta) la aplicación mostrará un botón “Añadir”, accionando el cual se mostrará una ventana modal en la que poder seleccionar uno de los usuarios registrados en IoTIndustrial y el rol o los roles que se le desea asignar en la planta.

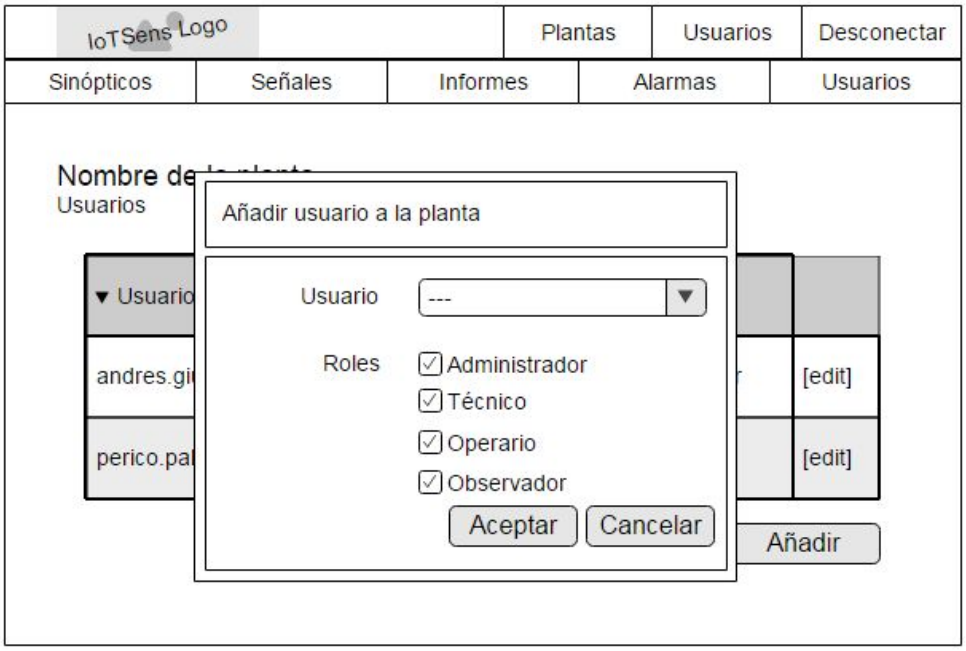

Figura 27. Prototipo de la ventana de asignación de roles

Como se puede apreciar, los pasos a realizar en cualquiera de las pantallas de administración/gestión (tanto para añadir o editar elementos, etc.), son similares. De esta manera se pretende alcanzar un nivel máximo de autosuficiencia por parte del usuario.

4.6.4 Administración y visualización de señales

Los últimos prototipos que se presentan corresponden a las pantallas de administración y visualización de señales. Los procesos de visualización, adición y edición de las señales son idénticos a los propios de plantas industriales y usuarios ilustrados en los apartados anteriores.

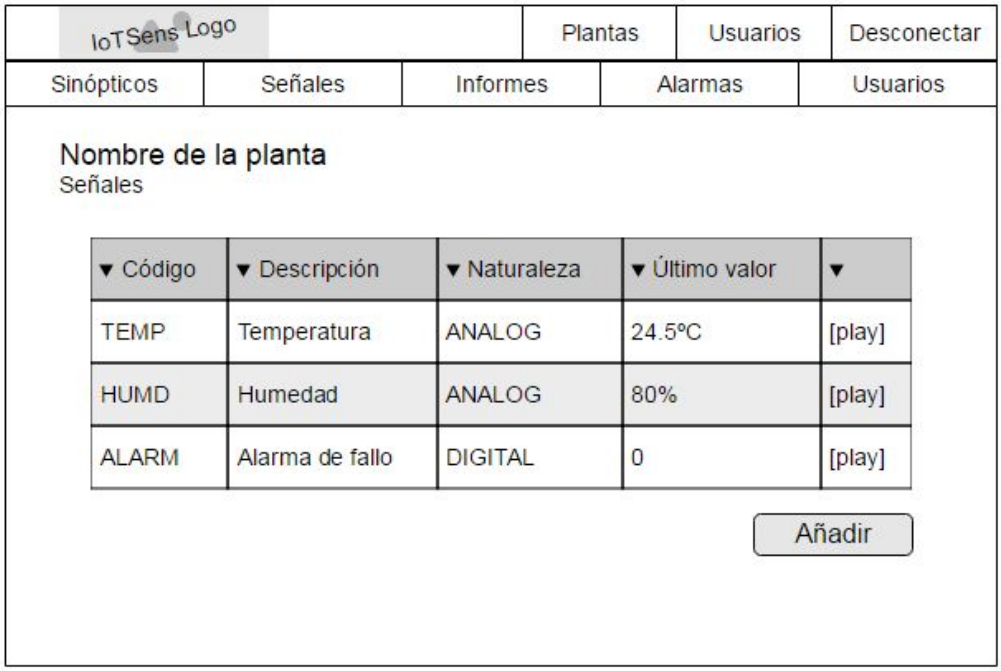

Figura 28. Prototipo de la pantalla de Administración de señales

La aplicación permitirá visualizar mediante una gráfica los valores de las últimas medidas recibidas de cada señal. Además, será posible regular el rango de tiempo que comprenderá la señal visualizada, así como descargar tanto un archivo Excel (.xls) como una imagen (PNG) con dicha señal.

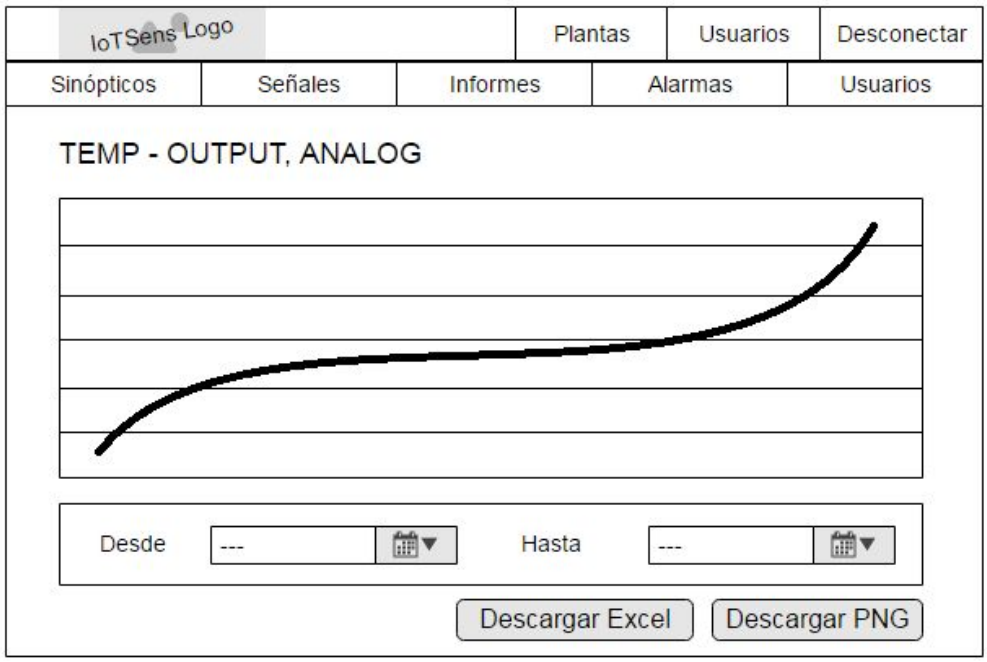

Figura 29. Prototipo de la pantalla de Visualización de señales

Capítulo 5

Implementación

Contenidos

Una vez terminada la fase de análisis, se procede a modelar el sistema a nivel de arquitectura y componentes de software. Así pues, en este capítulo se expondrán los detalles de implementación, incluyendo algunas capturas de la aplicación desarrollada en funcionamiento.

5.1 Entorno de desarrollo

Para desarrollar la aplicación, se ha trabajado de forma remota sobre una máquina con Ubuntu 12. En la empresa se adopta esta forma de trabajar para mantener configuradas

las diferentes máquinas, así como facilitar el mantenimiento y la asistencia remota por parte del departamento de *Datacenter*.

Por otra parte, las máquinas disponen de una estructura de directorios común, así como de una serie de ficheros de credenciales necesarios para ejecutar las diferentes aplicaciones que se ejecuten para la plataforma.

5.2 Arquitectura de la aplicación

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.

La arquitectura presente en la aplicación desarrollada es de cliente/servidor, ya que existe un cliente ligero que solicita y envía datos al servidor a través de peticiones a recursos. Además, IoT Industrial podría ser considerada en sí misma un cliente pues, como se detalla más adelante, a su vez solicita y envía datos al servidor central de IoTsens.

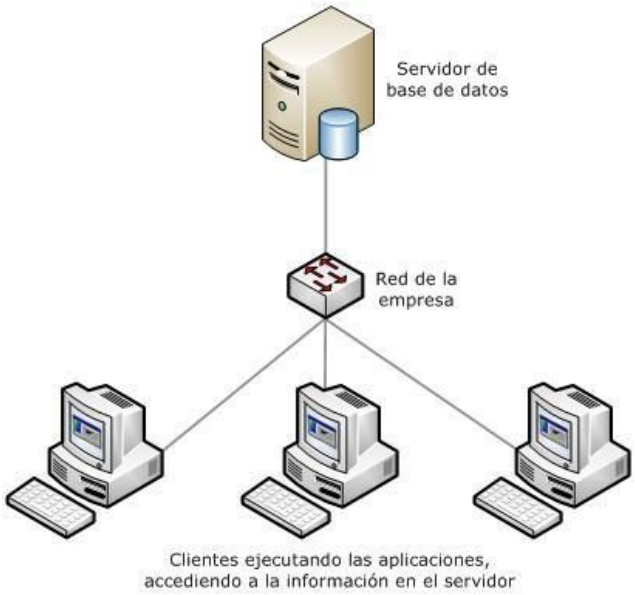

Figura 30. Modelo de arquitectura cliente/servidor

A continuación se pasará a detallar la arquitectura de software diseñada para la aplicación, además de la arquitectura de capas de IoTsens. Todas las aplicaciones de esta plataforma comparten el núcleo y las capas de datos, pues IoTsens fue diseñado horizontalmente para que los diferentes verticales puedan operar de manera homogénea.

5.2.1 Arquitectura de IoTsens

Para comprender la arquitectura de la aplicación desarrollada en este proyecto, es necesario introducir brevemente la arquitectura que sostiene toda la plataforma, tanto en lo relativo al software como al hardware.

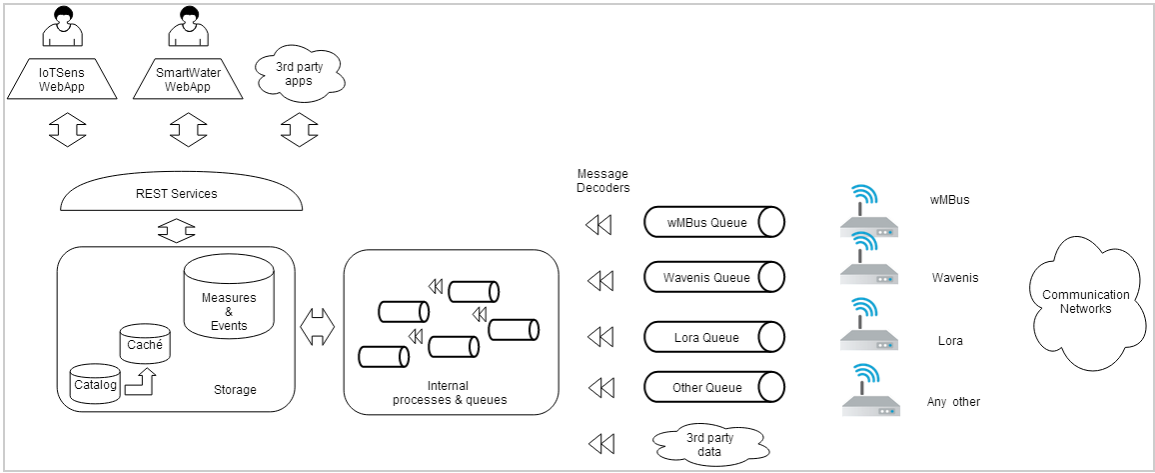

Figura 31. Diagrama de la arquitectura de IoTsens

Para poder implantar los conceptos propios del *Internet of things*, se sitúan una serie sensores en los lugares en los que se desea obtener información, como pueden ser sensores lumínicos, termómetros o limnímetros. Estos sensores envían mensajes con las medidas obtenidas a una serie de colas *MQTT*[9], las cuales procesan la información recibida y la decodifican según el tipo de sensor del que proceden. Una vez decodificados, estos mensajes se pueden almacenar en la base de datos central y pasan a ser accesibles por las diferentes aplicaciones [5].

La arquitectura que se va a describir en los siguientes apartados es la de nivel más superior, es decir, desde la base de datos hasta las aplicaciones web cliente.

5.2.2 Arquitectura de las aplicaciones de IoTsens

Las aplicaciones de IoTsens se encuentran diseñadas utilizando una arquitectura basada en capas y servicios.

En líneas generales, los usuarios interactúan con una interfaz web o con una aplicación en un dispositivo móvil. Estas interfaces se comunica con servicios REST para leer

[9] *Message Queue Telemetry Transport* es un protocolo para la comunicación *machine-to-machine* orientado a la comunicación de sensores debido a que consume poco ancho de banda y puede ser utilizado por dispositivos con pocos recursos.

datos, modificarlos o realizar otras operaciones de la aplicación. Los servicios son los encargados de leer los objetos de la capa de persistencia, modificarlos o ejecutar alguno de sus métodos si es necesario y, finalmente, enviarlos de nuevo a la capa de persistencia para almacenarlos.

La capa de persistencia es la encargada de almacenar los objetos en tablas de la base de datos y de cargar objetos a partir de la información de las tablas de la base de datos.

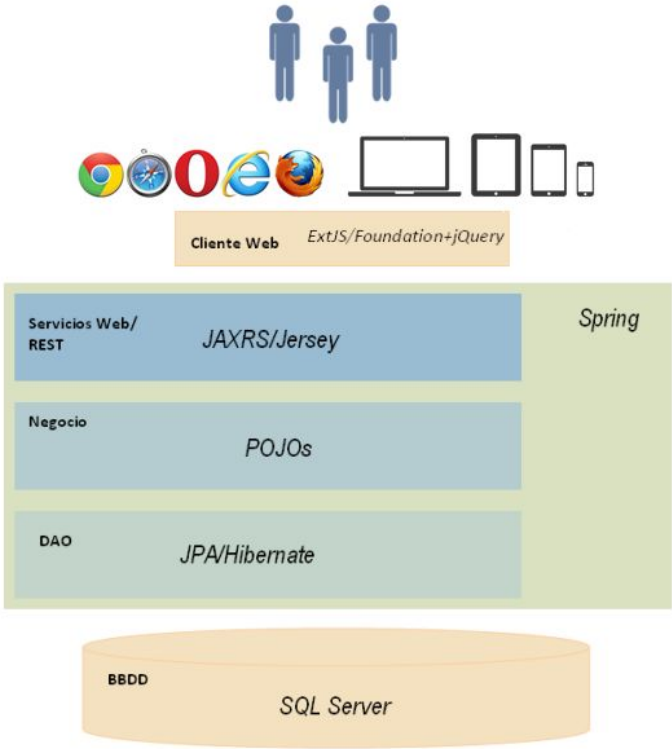

Figura 32. Arquitectura de las aplicaciones de IoTsens

La Figura 32 muestra de manera gráfica las distintas piezas de una aplicación de IoTsens, que se describen con algo más de detalle a continuación.

- *Servicio Web/REST*. Los servicios web de las aplicaciones de IoTsens siguen la arquitectura REST, por lo que definen una serie de recursos sobre los que se pueden realizar las acciones habituales en el protocolo HTTP: GET, PUT, POST y DELETE. Un servicio está asociado a una recurso y una acción; y puede recibir una serie de parámetros adicionales. El servicio habitualmente leerá los objetos de la capa de persistencia utilizando un objeto DAO, realizará alguna acción sobre ellos y devolverá al cliente el resultado. El servicio devuelve los datos de manera estructurada en formato JSON que se genera automáticamente a partir de los objetos del negocio.

- *Negocio*. Los objetos de negocio en las aplicaciones de IoTsens son objetos Java puros; es decir, que no extienden ni dependen de ningún framework externo. Habitualmente estarán anotados con información para indicar cómo se deben persistir en base de

datos o cómo se deben convertir a JSON, pero esto no impide que pueda crearse y manipularse de manera completamente independiente.
En un objeto de negocio se definirán los campos de información que maneja, las relaciones con otros objetos y las operaciones que se pueden realizar sobre el objeto. En estas operaciones se implementarán las reglas y condiciones que deben cumplir los datos. De esta manera es posible automatizar de manera sencilla los tests que comprobarán que la aplicación desarrollada cumple todas las reglas definidas por los usuarios/clientes.

- *Persistencia/DAO*. La capa de persistencia se implementa utilizando el patrón Data Access Object (DAO). Siguiendo este patrón, unos objetos DAO proporcionan métodos para recuperar y almacenar objetos en el sistema de persistencia que, en el caso de IoTsens, es una base de datos relacional. Los DAO de las aplicaciones de IoTsens reciben y devuelven en sus métodos objetos de negocio (o datos primitivos, si es necesario), pero nunca ningún tipo de datos que tenga que ver con el sistema de persistencia.

- *Clientes web ligeros*. Los clientes web ligeros en IoTsens se implementan utilizando páginas HTML generadas con el motor de plantillas Thymeleaf. La maquetación de estas páginas se realiza basándose en el framework Foundation, lo que facilita que se adapten al tamaño del dispositivo desde el que se visualiza. En los clientes web ligeros, la interacción con los servicios suele ser sencilla y para ello se realizan directamente llamadas AJAX con jQuery y un motor de plantilla Javascript que genera el HTML necesario.

Cabe destacar que en todas las aplicaciones de IoTsens (y, por lo tanto, en IoT Industrial) se siguen una serie de recomendaciones y principios de diseño para la implementación de objetos de negocio, los cuales se enumeran a continuación:

- *Principio de responsabilidad única*. Según este principio, una clase debe tener una única razón para cambiar; es decir, debe utilizarse sólo para una cosa.

  En ocasiones es tentador añadir métodos y propiedades a una clase porque tienen alguna relación con ella y acabamos con clases enormes que sirven para hacer muchas cosas. Es mejor tener varias clases pequeñas con pocos campos y métodos que se dedican a realizar sólo una tarea, ya que de esta manera son más fáciles de mantener y evolucionar en caso de que sea necesario.

- *Priorizar composición sobre herencia*. La programación orientada a objetos introduce el concepto de herencia como mecanismo para reutilizar estructura o comportamiento entre clases. Aunque en ocasiones resulte la mejor opción de implementación, la herencia establece una dependencia muy fuerte entre clases por lo que dificulta su mantenimiento y evolución.

  Si al cambiar los requisitos hay que cambiar una de las clases de la jerarquía de herencia, este cambio puede afectar inesperadamente a las otras clases de la jerarquía

y romper su funcionamiento. Si alguno de los métodos de la clase padre no tiene sentido para la clase hija es mejor evitar la herencia y utilizar composición.

- *Inyectar las dependencias y depender de interfaces*. Cuando un objeto necesita utilizar funcionalidad o información de otro objeto (un escenario muy habitual) es recomendable que la referencia al objeto del que depende la reciba por el constructor o por algún método "setter".

  Además, en lugar de declarar la dependencia al tipo de una clase concreta es recomendable utilizar una interfaz. De esta manera es posible cambiar el comportamiento enviándole en distintos momentos distintos objetos como dependencia siempre que cumplan la interfaz definida, por lo que se incrementa la flexibilidad y se facilita la implementación de pruebas automáticas.

  En general, en un objeto de negocio hay que evitar las instrucciones "new" para construir nuevos objetos, ya que esto crea una dependencia fuerte entre clases y dificulta su evolución.

- *Métodos cortos y con nombres descriptivos*. Como norma general, es recomendable implementar métodos cortos que permitan leer el algoritmo de manera natural[6]. Esto se puede conseguir creando muchos métodos pequeños con nombres que describan correctamente el objetivo del método y que no contengan abreviaturas.

  También deberían nombrarse todas las constantes que se utilicen en el código con un nombre descriptivo (por ejemplo, static final String HOURS_IN_A_DAY = 24).

- *Uso adecuado de excepciones*. Java proporciona el mecanismo de excepciones para interrumpir la secuencia habitual de ejecución cuando se detectan errores o problemas.

  En lugar de devolver "null" o códigos de error es recomendable utilizar excepciones, ya que de esta manera se puede definir de manera clara por una parte la secuencia habitual y por otra parte el tratamiento de las excepciones. Para ello será necesario crear objetos excepción propios de la aplicación de IoTsens que se desarrolle que representen los errores que pueden detectarse.

5.2.3 Arquitectura de IoT Industrial

IoT Industrial es una aplicación del ámbito de IoTsens, por lo que las decisiones en cuanto al diseño de la arquitectura descritas en el apartado anterior se han llevado a cabo en este proyecto.

Sin embargo, existen algunas particularidades que es importante detallar para comprender el funcionamiento de la aplicación desarrollada y su comunicación con el servidor central.

Como es natural, la aplicación desarrollada tiene su propia base de datos donde mantiene la información de las entidades propias. Además, los servicios web se han separado en *recursos* y *servicios*. Los primeros simplemente definen los puntos de entrada para las peticiones de los clientes y también son los encargados de realizar la validación de la información recibida. Por otra parte, los servicios reciben los parámetros validados de los recursos y realizan las operaciones deseadas comunicándose, si es necesario, con los DAO.

La arquitectura de capas de la aplicación se muestra en la Figura 33.

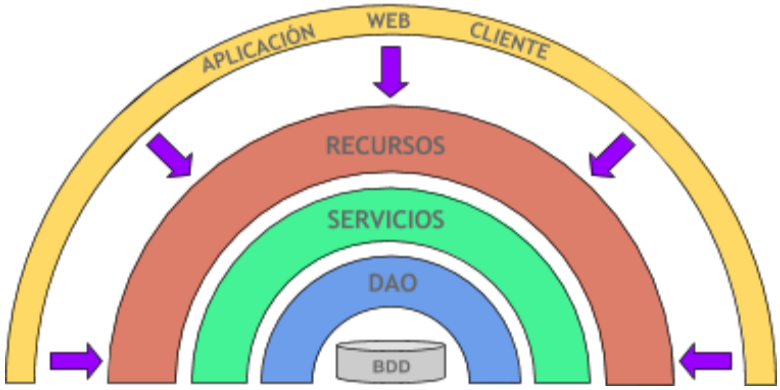

Figura 33. Arquitectura de capas de IoT Industrial

Sin embargo, en momentos determinados se necesita leer información de la base de datos central de IoTsens. Como IoT Industrial es un a aplicación de IoTsens, todos sus objetos (las plantas industriales, los usuarios, las señales, etc.) son también objetos de IoTsens. En la base de datos de la aplicación se almacena únicamente cierta información de dichos objetos, de manera que no se replique dicha información que ya está en la base de datos central de IoTsens.

Es por esto que, por ejemplo, cuando en IoT Industrial se desean mostrar los datos de un usuario, se buscan esos datos en la base de datos central a partir del código de usuario, que sí se encuentra persistido en la base de datos de IoT Industrial. Por lo tanto, la arquitectura completa de la aplicación, desde la aplicación web cliente hasta el núcleo de datos es la que se muestra en la Figura 34.

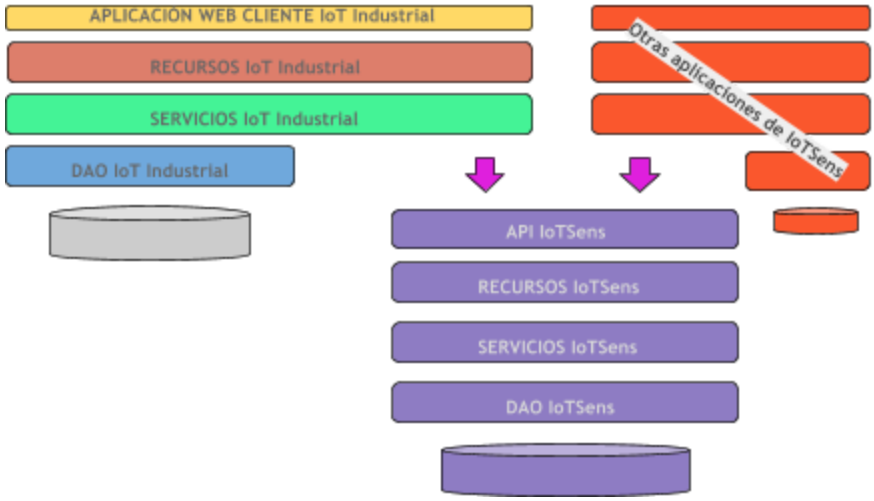

Figura 34. Arquitectura completa de capas de IoT Industrial hasta la BDD central de IoTsens

Como se puede observar, IoTsens ofrece una API para que los distintos verticales (como puede ser IoT Industrial) puedan acceder fácilmente a los datos de la base de datos central. Es esta librería la que realiza las peticiones a los recursos de IoTsens y devuelve los datos a las aplicaciones que los solicitan.

5.3 Patrones de diseño

El diseño a nivel de componentes es la etapa que se centra en la organización de las clases e interfaces y cómo se relacionan entre sí. En este apartado se van a mostrar algunos diagramas relativos a los patrones de diseño que se optó por usar durante esta etapa, y algunas explicaciones relevantes. Debido a la complejidad del proyecto, se mostrarán aquellos que se consideran más relevantes o representativos del producto final obtenido.

5.3.1 Patrón MVC

El patrón de diseño o arquitectónico Modelo-Vista-Controlador es un patrón que engloba toda la aplicación. Todos los componentes del sistema se pueden ubicar dentro de uno de los tres módulos que lo componen. El objetivo de este patrón es desacoplar la lógica de negocio de la interfaz de usuario y los datos del modelo. En la Figura 35 se detallan los módulos que componen este patrón, así como el principal flujo de acciones que se realizan.

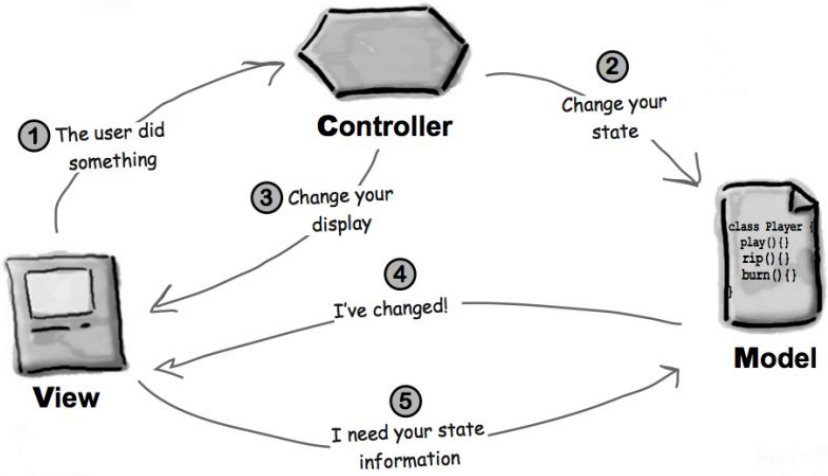

Figura 35. Diagrama del patrón de diseño MVC [7]

- *Modelo*. Este componente tiene que ver con las representaciones creadas de los datos que va a usar la aplicación. Mantiene los datos, el estado y la lógica del programa. Recibe solicitudes de cambio desde el controlador y, tras aplicarlas, notifica a la vista de los cambios.

  Cada elemento debe tener una representación bien definida y única, con el objetivo de no repetir código y evitar información redundante. Algunos componentes del modelo de la aplicación desarrollada podrían ser las clases *IndustralPlant*, *User* o *Signal*, pues son representaciones de entidades de información.

- *Vista*. Se corresponde con la interfaz de usuario y, por tanto, es el componente con el que el usuario interactúa. En una implementación pura de MVC, la vista no tiene ningún estado y no realiza acciones, sino que las delega en el controlador. En este proyecto la vista se corresponde con la parte *front-end* de la aplicación web.

- Controlador: Es el encargado de gestionar la lógica de la aplicación. Recibe la entrada del usuario (de la vista) y actualiza el modelo en consecuencia.

Gracias a este patrón se consigue que la aplicación sea más escalable, extensible y capaz de intercambiar alguno de los tres componentes, mientras se cumpla la misma interfaz. Además, al estar los componentes bien separados, es fácil repartir las tareas de desarrollo y de diseño de interfaz entre los participantes en el proyecto.

5.3.2 Patrón DAO

Como se ha comentado anteriormente, el aplicación desarrollada consta de una capa que se encarga de gestionar la persistencia de los objetos. Esta capa está compuesta por una serie de objetos DAO que almacenan o recuperan objetos desde el sistema de persistencia.

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.

Los métodos que proporciona un objeto DAO para gestionar la persistencia dependen de las necesidades específicas de cada aplicación. Es habitual tener métodos para recuperar un objeto consultando por su identificador o para almacenar una versión actualizada de todas las propiedades de un objeto; aunque también es posible si es necesario crear métodos que, por ejemplo, almacenen únicamente algunas de sus propiedades o que devuelvan colecciones de objetos que cumplan cierta condición.

Aunque no hay una norma estricta respecto a cuántos objetos DAO se deben crear, es habitual definir un DAO con los métodos para almacenar y recuperar cada tipo de objeto de negocio. En esta aplicación se han implementado cinco objetos DAO, donde cada uno de ellos permite el acceso a datos de diferente ámbito:

- *IndustrialPlantDAO*. Para persistir y recuperar (por nombre, por usuario, etc.) datos relativos a las plantas industriales.

- *UserDAO*. Mediante este DAO se accede a la información de los usuarios. registrados en el sistema.

- *LoggedUserDAO*. Se puede considerar un DAO auxiliar. Se ha utilizado para facilitar el acceso a la información del usuario autenticado.

- *GrantedRolDAO*. Para acceder a la información sobre los permisos de cada usuario en las plantas. Algunos de sus métodos son el que permite obtener todos los usuarios de una planta o el que indica si un usuario determinado tiene permisos en una planta.

- *SignalDAO*. Para persistir y recuperar información de las señales.

El patrón DAO proporciona una serie de beneficios a las aplicaciones que lo utilizan, como puede ser el desacoplamiento entre la lógica de negocio y la persistencia (la estructura de los datos persistidos puede cambiar sin afectar a la lógica o viceversa, pues la capa DAO se encargará de realizar las conversiones necesarias para que lleguen a la aplicación los datos siguiendo la estructura de objetos definida en el negocio), una evolución tecnológica más sencilla (la independencia de la capa DAO permite evolucionar de mecanismo de persistencia sin afectar al resto de la aplicación), o la centralización de la gestión de la persistencia (todo el código de la aplicación encargado de la gestión de la persistencia de datos se encuentra centralizado una serie de objetos, por lo que si es necesario realizar algún cambio que afecte a la persistencia está claramente acotado cual es el código de la aplicación que puede verse afectado).

5.3.3 Patrón Builder

El patrón *Builder* pertenece al ámbito de los patrones de diseño creacional, es decir, su cometido es la creación de instancias de alguna clase de forma desacoplada y transparente.

Este patrón ha resultado muy útil en los tests unitarios en los que, para la clase a probar, se ha implementado 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 "*withAttribute"* 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.

Un ejemplo de clase *builder* podría ser la que se muestra en la Figura 36, donde se implementa dicho patrón sobre la clase *User*.

```
public class User {

    private String userID;
    private String name;
    private String email;
    private String phone;

    public static UserBuilder create () {
        return new UserBuilder;
    }

    public UserBuilder withUserID (String userID) {
        this.userID = userID;
        return this;
    }

    public UserBuilder withName (String name) {
        this.name = name;
        return this;
    }

    public UserBuilder withEmail (String email) {
        this.email = email;
        return this;
    }

    public UserBuilder withPhone (String phone) {
        this.phone = phone;
        return this;
    }

    public User build () {
        User user = new User();

        user.setUserID(userID);
        user.setEmail(email);
        user.setPhone(phone);

        return user;
    }
}
```

Figura 36. Ejemplo de clase *builder*

De esta forma, el usuario obtendría una nueva instancia de la clase User ejecutando la instrucción que se muestra en la Figura 37.

```
User newUser = UserBuilder.create()
    .withUserID("andres.giustini")
    .withName("Andrés Giustini")
    .withEmail("agiustini@grupogimeno.com")
    .build();
```

Figura 37. Ejemplo de utilización del patrón *builder*

Utilizando este patrón, nos aseguramos que la creación es transparente al programador que instancia dicha clase, ya que no se utiliza la palabra reservada *new*. En un futuro, se podría cambiar la estructura interna de la clase Material y no afectaría al funcionamiento. Si se añadiesen atributos, se podría ampliar el *builder* en lugar de crear constructores, que acoplan el código y hacen difícil su mantenimiento.

5.4 Interfaz de usuario

Puesto que el proyecto desarrollado consta de una aplicación web, las diferentes interfaces de usuario de ésta se han de implementado, como se ha comentado anteriormente, con el lenguaje de marcas HTML y, con el fin de aportar una apariencia más vistosa y amigable a las páginas, se han utilizado las hojas de estilo CSS. Además, con el lenguaje interpretado Javascript, se ha dotado a la aplicación de un cierto dinamismo y una mayor capacidad de interacción con el usuario.

En el proceso de implementación de la interfaz de usuario se han tomado como pilares fundamentales los siguientes principios de diseño y usabilidad:

- *Diseño sencillo*. La interfaz no debe abrumar al usuario con elementos innecesarios. La atención del usuario debe estar constantemente puesta en lo que quiere hacer.

- *Interfaz familiar*. Se ha pretendido en todo momento facilitar al usuario la navegación, aprovechando al máximo los componentes comunes a la mayoría de sitios web. Es el caso, por ejemplo, del menú superior de navegación entre secciones o destacar los botones de confirmación por encima de los de anulación.

- *Metáforas visuales.* La mayoría de los botones que conducen a acciones van acompañados de un icono que representa su funcionalidad. De esta manera, el usuario puede hacerse una idea *a priori* de lo que va a suceder.

- *Consistencia*. Las distintas páginas que componen la interfaz poseen un aspecto muy parecido en cuanto a colores y a posición de los componentes. Además, acciones similares producen efectos similares.

- *Jerarquía visual*. Cuando se desea realizar una tarea, es importante que la interfaz permita al usuario centrarse en los elementos más relevantes para llevarla a cabo [8]. La jerarquía visual se consigue mediante el tamaño, color, contraste, proximidad o la alineación de los diferentes elementos que intervienen.

- *Feedback*. Es absolutamente indispensable que la aplicación informe en todo momento al usuario de lo que está sucediendo. De esta forma se consigue evitar la sensación de incertidumbre que se produce, por ejemplo, cuando no se sabe si una acción se ha podido llevar a cabo con éxito. El *feedback* aporta mucho valor a la aplicación como producto.

- *Reflejar estado*. Es importante no obligar al usuario a recordar, por lo que en todo momento la interfaz debe reflejar el estado en que se encuentra.

- *Mensajes de confirmación*. Se debe pedir confirmación antes de realizar operaciones importantes, como por ejemplo eliminar un usuario del sistema.

Como resultado de tomar como base esta serie de principios, se ha conseguido una interfaz intuitiva, estéticamente agradable y fácil de utilizar. A continuación se presentan las interfaces de las páginas más representativas que conforman la aplicación desarrollada.

5.4.1 Gestión de plantas

La gestión de plantas la componen las operaciones para visualizar las plantas industriales del sistema, añadir nuevas plantas y editar las existentes.

En la Figura 38 se puede observar el aspecto del listado de plantas con tres plantas industriales registradas. En el listado se presenta, a petición del cliente, el código y el nombre de cada una de las plantas, así como su dirección y un logo de la empresa a la que pertenece.

Figura 38. Pantalla de Administración de plantas

En este caso, puesto que se ha iniciado la sesión como usuario administrador, están disponibles el botón e iconos para añadir y editar plantas, respectivamente. En caso de que el usuario autenticado no fuera identificado como administrador, se mostraría simplemente el listado de plantas, que permite acceder a cada una de ellas, pero no modificar su información.

Cuando se desea dar de alta una nueva planta, el administrador seleccionaría el botón etiquetado como “Añadir”, lo que haría aparecer una ventana modal con un formulario, como muestra la Figura 39.

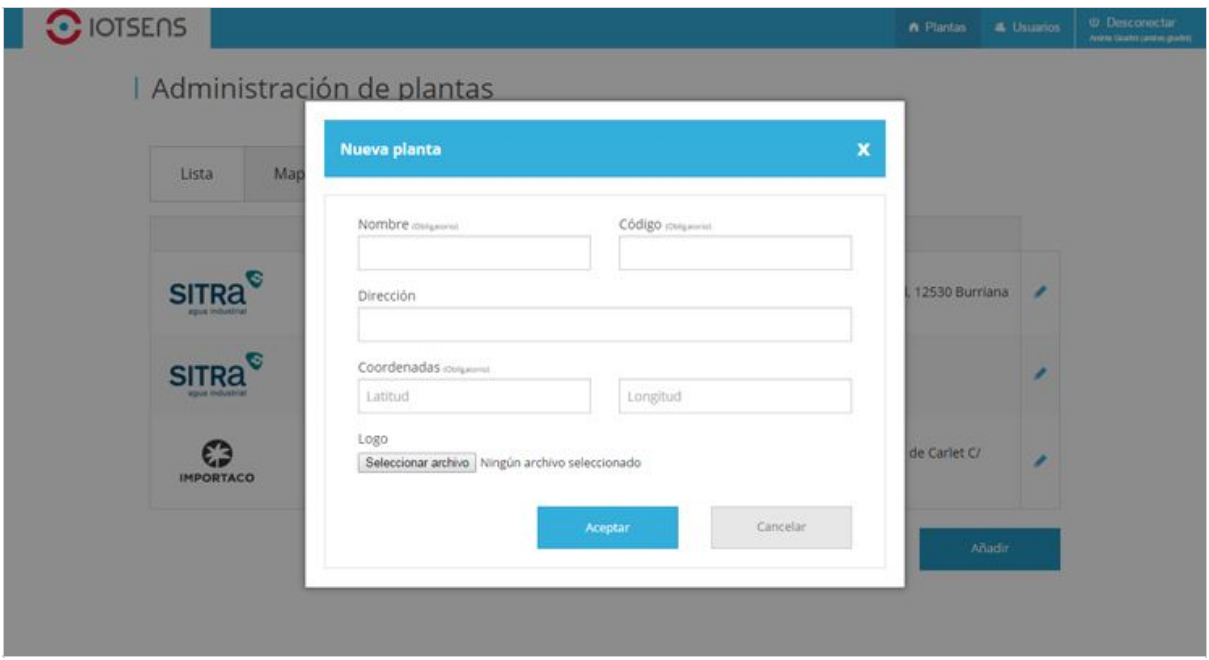

Figura 39. Formulario de alta de plantas

En este momento, el usuario administrador únicamente debe rellenar los campos con la información requerida y seleccionar “Aceptar”.

La operación “editar una planta” se lleva a cabo pulsando el icono del lápiz correspondiente a la planta de la cual se desea actualizar su información. Al llevar a cabo esta acción, se muestra la misma ventana modal que en el caso anterior, pero con los campos del formulario completados con la información actual de la planta a editar, como se puede ver en la Figura 40.

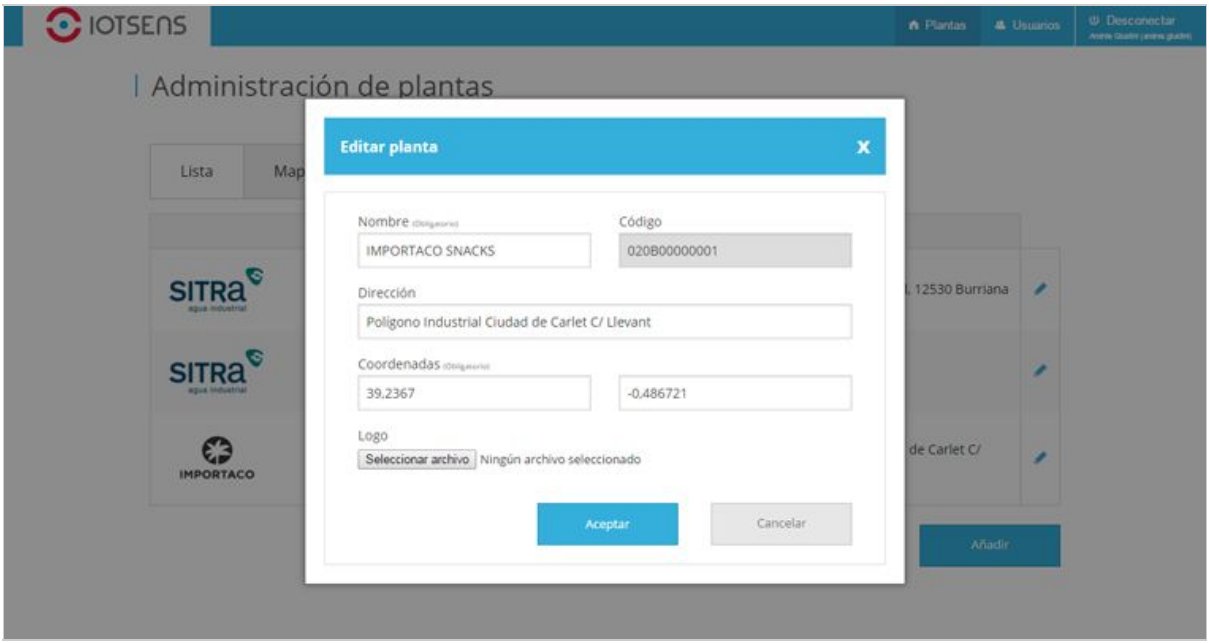

Figura 40. Formulario de edición de plantas

De esta forma resulta más cómodo para el usuario modificar la información de sólo aquellos campos que quiere actualizar.

5.4.2 Gestión de usuarios y roles

Como se ha comentado en los principios seguidos para el desarrollo de la interfaz, se ha considerado importante mantener una cierta coherencia entre las diferentes pantallas, de manera que se facilite la navegación del usuario y se aumente su autosuficiencia. Por este motivo, los diferentes listados (gestión de plantas, de usuarios y de roles) así como las funcionalidades que ofrecen y la forma de llevarlas a cabo es prácticamente análoga.

Por ejemplo, la forma de editar la información de un usuario o sus roles, como se muestra en las Figuras 41 y 42, es idéntica a la edición de plantas: se editan los campos que se desea actualizar y se selecciona “Aceptar”.

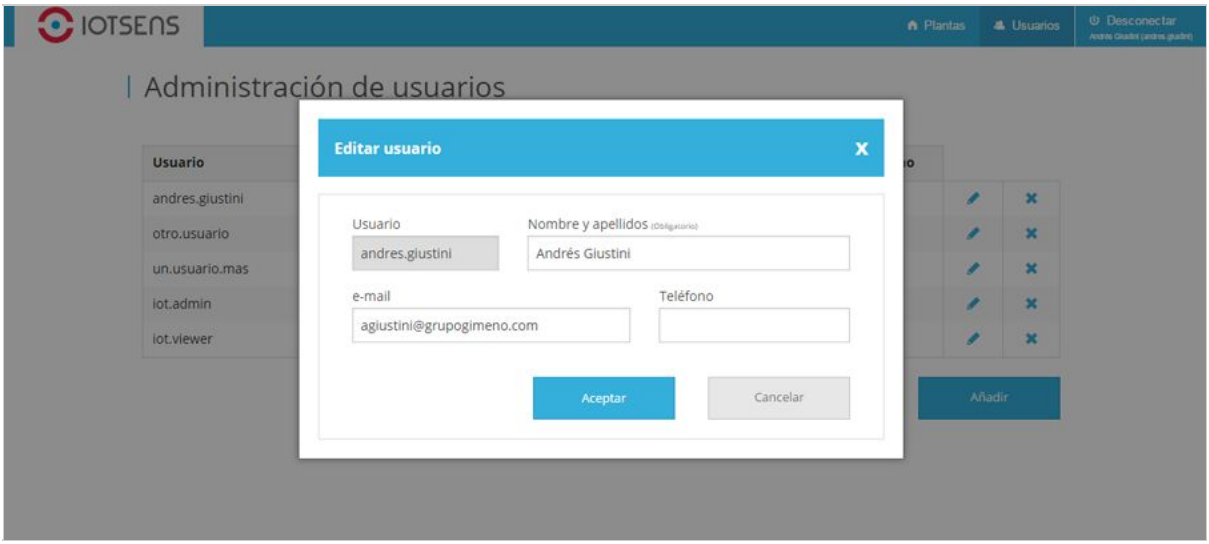

Figura 41. Formulario de edición de usuarios

Como se ha explicado en secciones anteriores, en la aplicación únicamente se almacena para cada usuario su código o nombre de usuario. El resto de atributos se leen de la base de datos central de IoTsens, pero a través de la interfaz se ha pretendido que este tipo de operaciones permanezcan transparentes para el usuario de la aplicación, pues no tiene por qué ser consciente de los detalles de diseño propios de la arquitectura de la aplicación.

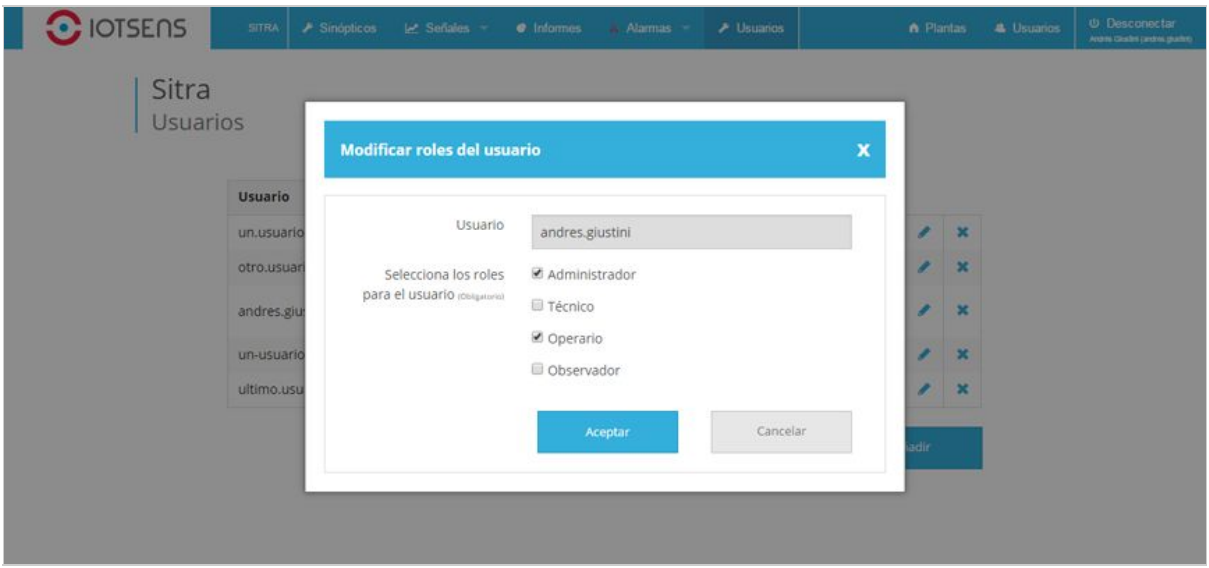

Figura 42. Formulario de edición de roles de usuarios

Para que la operación de llevar a cabo la asignación de roles sea lo más intuitiva posible, se ha optado por la utilización de *checkboxes*, donde cada uno se corresponde con uno de los posibles roles que un usuario puede tener en una planta. De esta forma se puede ver de

un simple vistazo qué roles tiene y cuáles no tiene asignados un determinado usuario en un momento dado.

Al completarse cada una de las operaciones expuestas, se informa al usuario a través de un mensaje indicando si todo se ha llevado a cabo con éxito o si se ha producido algún tipo de error. En la Figura 43 se puede ver un mensaje informando de que la asignación de roles a un determinado usuario se ha realizado correctamente.

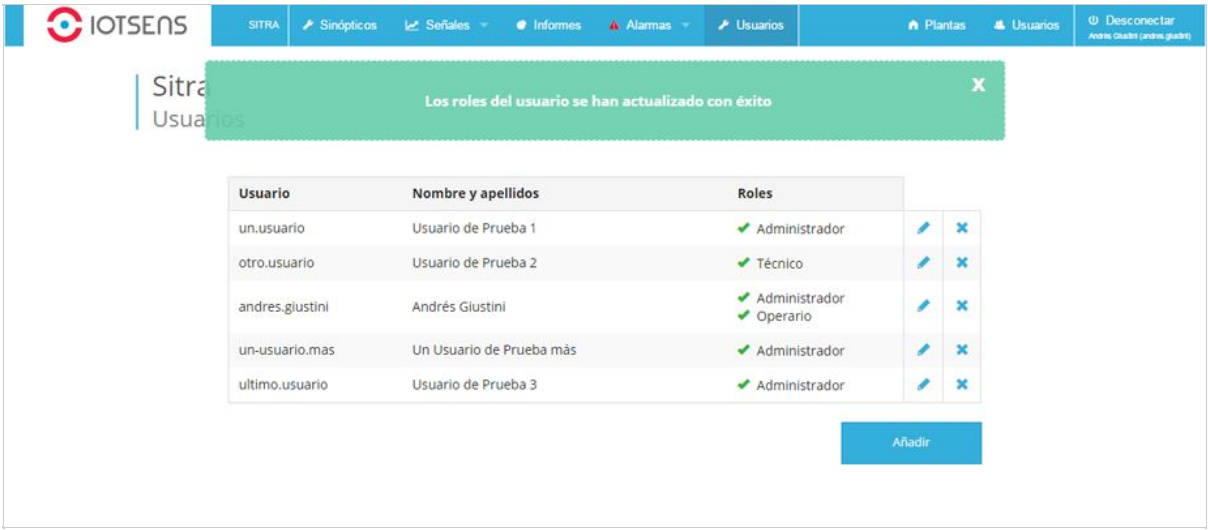

Figura 43. Ejemplo de mensaje *feedback* de éxito

5.4.3 Gestión y visualización de señales

Una vez que un usuario se ha familiarizado con los listados mencionados hasta ahora, no debería resultarle difícil desenvolverse en la pantalla de "Administración de señales". En esta sección se pueden visualizar, añadir, editar y eliminar señales, que representan la información que recibe la aplicación de las plantas industriales.

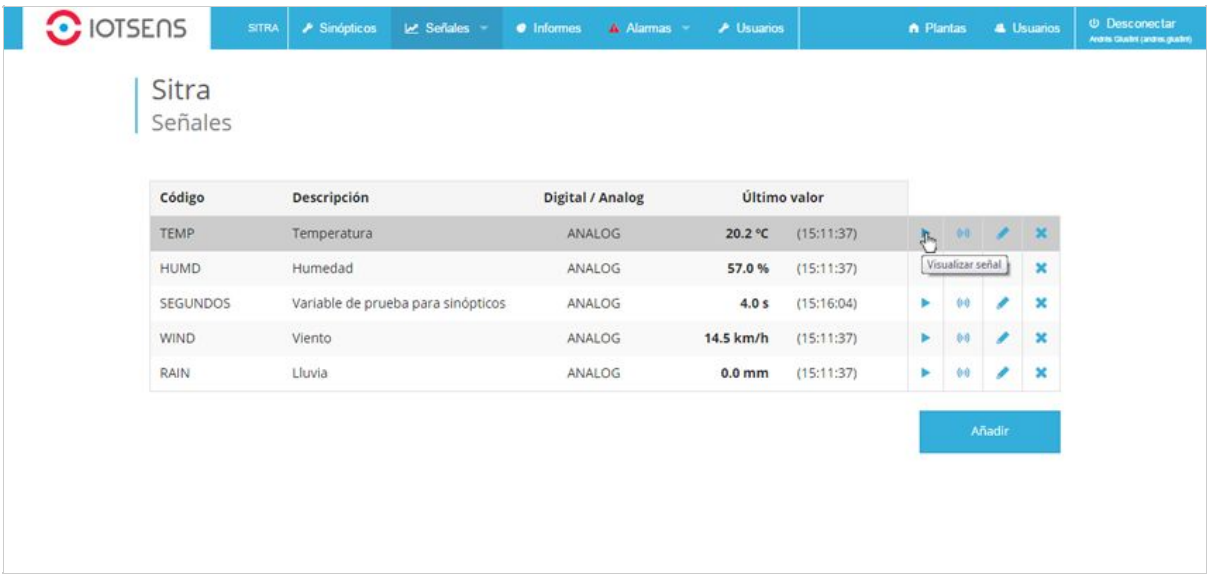

Figura 44. Pantalla de Administración de señales

En este listado, que aparece representado en la Figura 44, además de las operaciones de añadir, editar y eliminar señales, es posible visualizar cada una de las señales a través de una gráfica. En la Figura 45 se muestra la visualización de una señal en una gráfica de líneas.

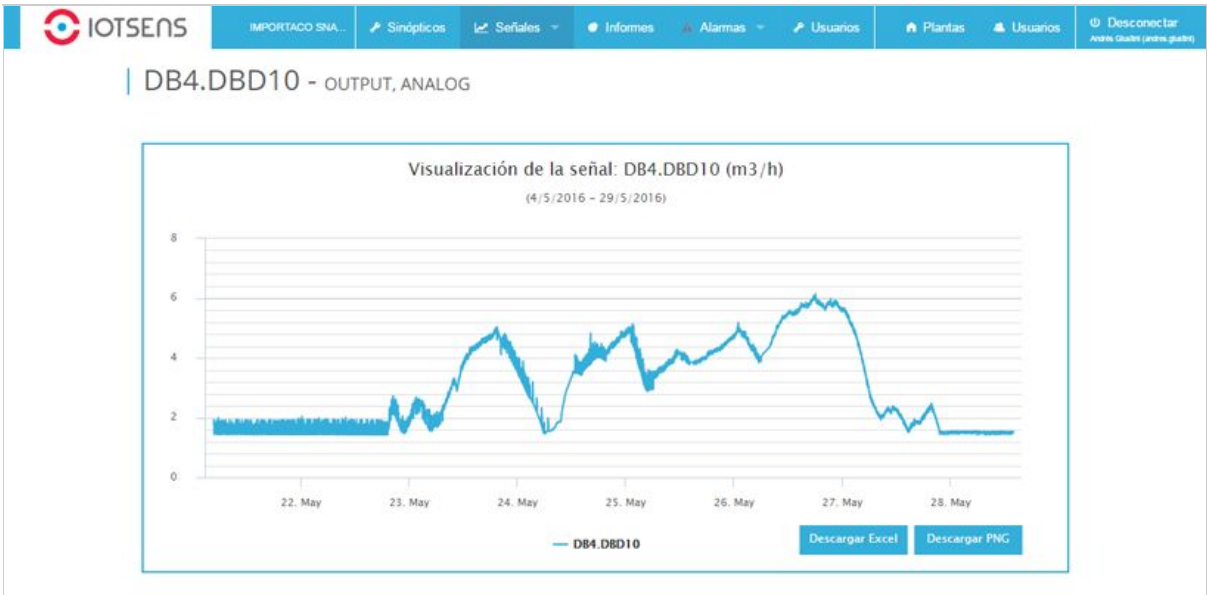

Figura 45. Pantalla de visualización de señales

Sobre dicha gráfica pueden realizarse una serie de operaciones a petición del cliente, como seleccionar el rango de tiempo a graficar o si se desea aplicar algún tipo de operación de agregación. Una agregación es una forma de representar las medidas de una señal, resumiendo en un único valor todos los datos de un intervalo de tiempo a definir. La manera de resumir todos los datos en uno se especifica a partir de un tipo de operación, como puede ser el máximo, el mínimo, la suma, etc.

En la Figura 46 se ha representado una señal de temperatura en un rango de diez días, con una agregación por días donde la operación de agregación es la media. Así pues, en la gráfica se visualizan los valores de esa señal entre los días 13 y 23 de mayo de 2016, donde para cada día únicamente se muestra el valor medio de temperatura. De esta forma se puede observar rápidamente las temperaturas medias de esos días y cuál de ellos ha sido el que ha tenido la mayor temperatura media, por ejemplo.

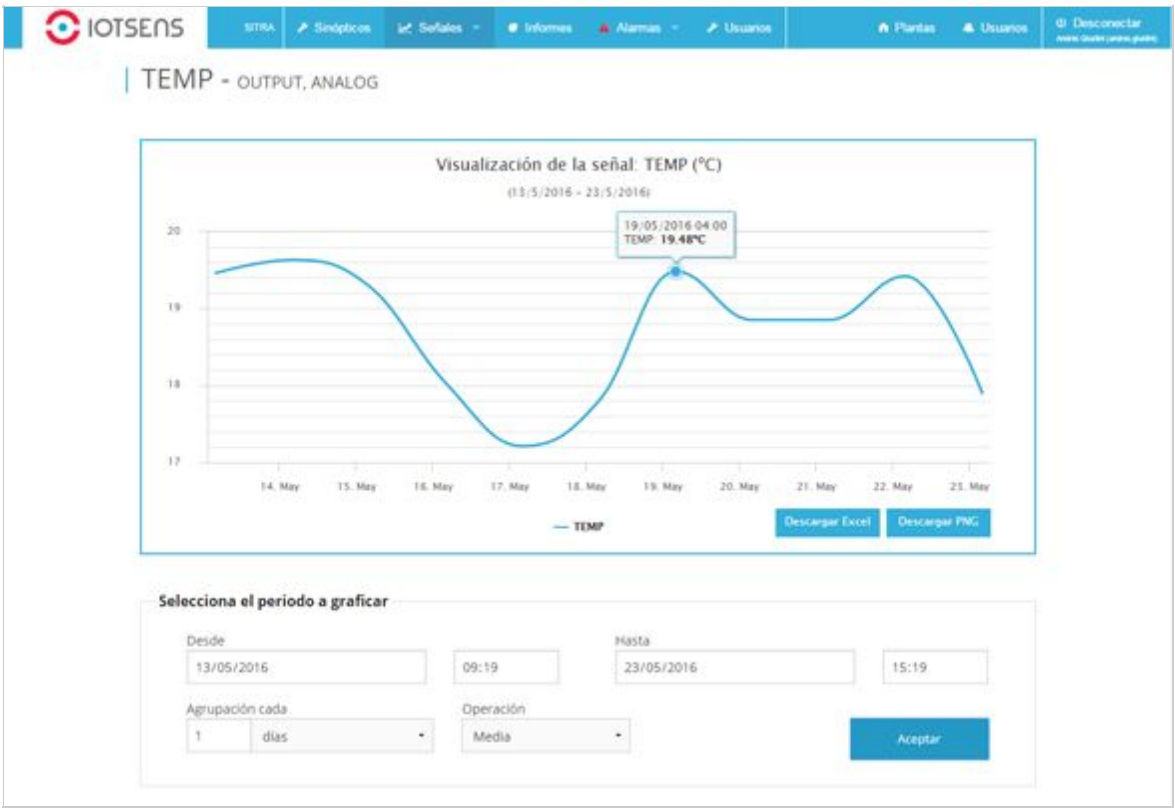

Figura 46. Ejemplo de visualización de datos agrupados

Puesto que en las gráficas se pueden llegar a presentar una gran cantidad de datos, el tiempo de respuesta de la aplicación puede ser relativamente elevado. Para que, mientras la aplicación recupera los valores a mostrar, no se dé al usuario una sensación de que la aplicación se ha quedado "bloqueada" o que algo ha dejado de funcionar, se ha optado por ofrecer cierto *feedback* en forma de *spinner*, como se puede apreciar en la Figura 47.

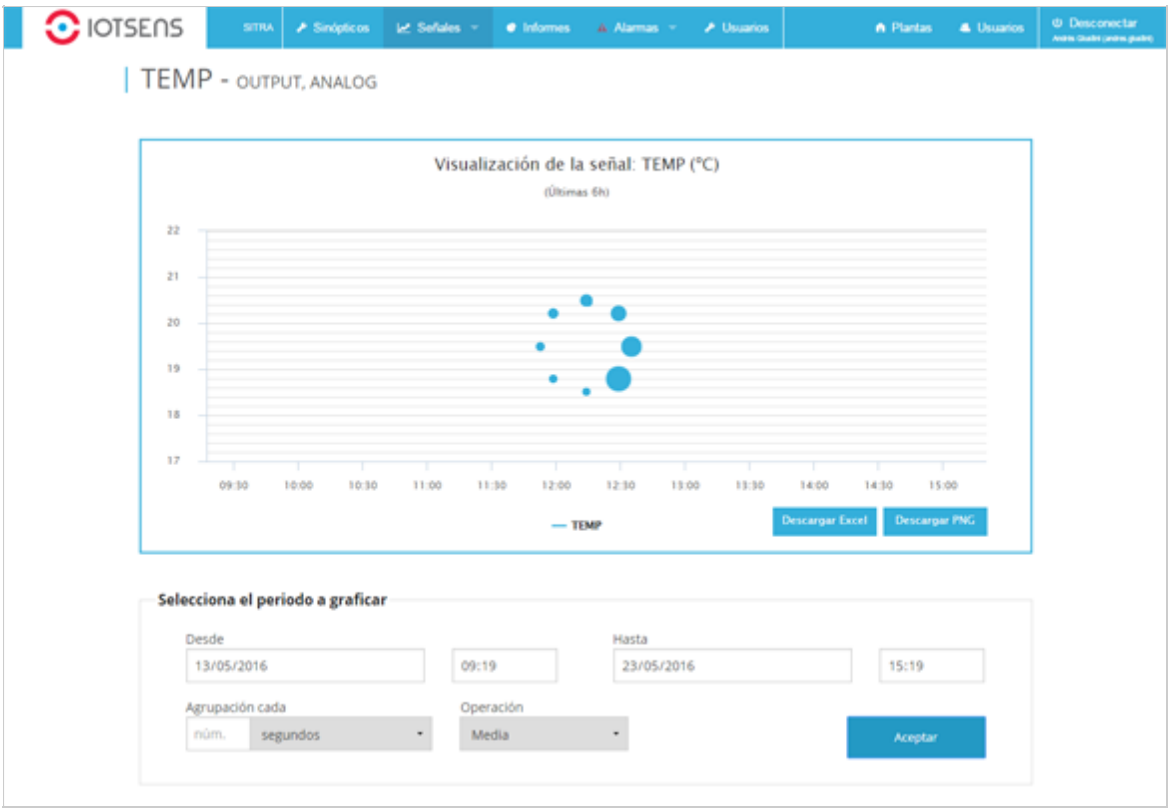

Figura 47. Mensaje de *feedback* de tipo *spinner*

Con esto se consigue una cierta tranquilidad por parte del usuario al saber que, aunque todavía no se han mostrado los datos, la aplicación está "pensando".

Por último, cabe destacar que antes de llevar a cabo cada una de las operaciones importantes (como pueden ser las de eliminación), el sistema muestra siempre un mensaje esperando confirmación del usuario. En la Figura 48 se puede ver un mensaje de confirmación al eliminar una señal de las presentes en el sistema.

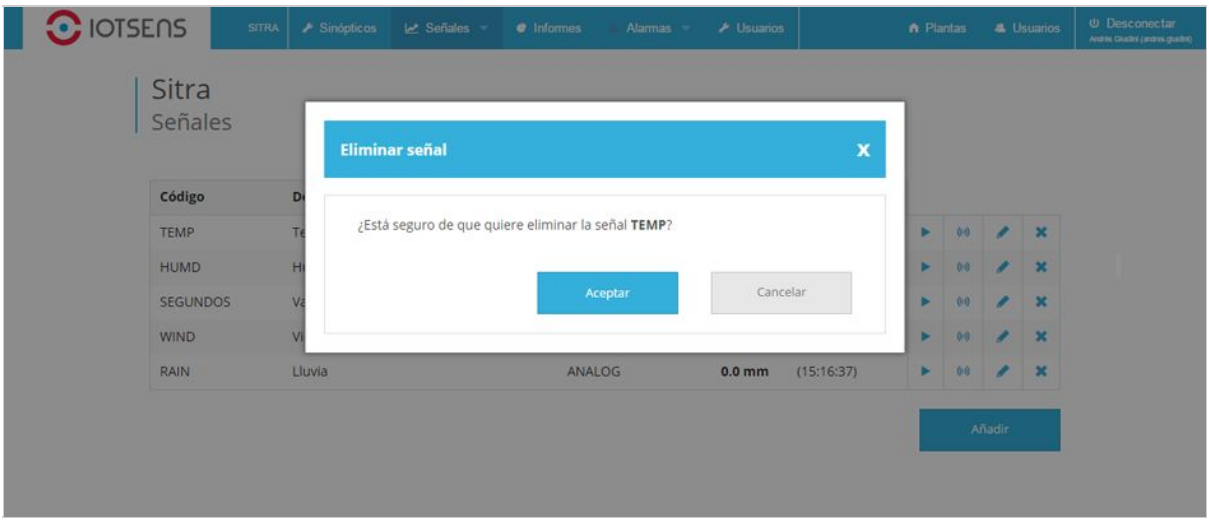

Figura 48. Ventana de confirmación de eliminación de una señal

Capítulo 6

Verificación y validación

Contenidos

El objetivo de la fase de verificación y validación del sistema es comprobar el correcto funcionamiento de éste 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 el tipo de componente que se está probando:

- Pruebas unitarias
- Pruebas de integración
- Pruebas de carga y estrés
- 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 desarrollado pruebas unitarias, de integración y de aceptación.

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 nos aseguramos de que cada uno de los módulos trabaja como se espera por separado.

Para desarrollar las pruebas unitarias en este proyecto se han utilizado las herramientas *JUnit* y *Mockito*. *JUnit* es una librería de código abierto que permite definir casos de pruebas unitarias para programas escritos en *Java*. *Mockito* se ha utilizado para la simulación de comportamientos en componentes complejos. Esto permite desarrollar pruebas unitarias para componentes que tienen una alta dependencia con otros, sin llegar a ser pruebas de integración.

Puesto que se ha utilizado *Maven* como software de gestión de dependencias, para tener disponibles estas dos librerías únicamente ha sido necesario añadir las líneas que se muestran en la Figura 49 al fichero *pom.xml*.

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.8.1</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-all</artifactId>
    <version>1.8.1</version>
    <scope>test</scope>
</dependency>
```

Figura 49. Declaración de las dependencias *JUnit* y *Mockito* en *Maven*

En la Figura 50 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 servicio que se invoca cuando se desea modificar las propiedades de una de las señales de una planta.

```
@Test
public void modificarSeñalCreaCorrectamente() throws Throwable {

    SignalServices services = getBasicSignalServices();

    Signal signal = services.updateSignal("MY_PLANT", 500L, "MY_SIGNAL", "OUTPUT", "ANALOG", "Description", "UNITS", 10);

    assertThat(signal.getId(), is(500L));
    assertThat(signal.getPlantCode(), is("MY_PLANT"));
    assertThat(signal.getCode(), is("MY_SIGNAL"));
    assertThat(signal.getDirection(), is(SignalDirection.OUTPUT));
    assertThat(signal.getNature(), is(SignalNature.ANALOG));
    assertThat(signal.getDescription(), is("Description"));
    assertThat(signal.getUnits(), is("UNITS"));
    assertThat(signal.getIsDerived(), is(false));

    verify(services.ioTSensClient, times(1)).updateSensorVariable(eq("LOGGED"), eq("MY_PLANT"), any(SensorVariable.class));
    verify(services.signalDAO, times(1)).updateSignal(any(Signal.class));

}
```

Figura 50. Ejemplo de test con *JUnit*

Como se puede apreciar, los métodos que constituyen tests unitarios se identifican con la anotación *@Test* y es habitual asignarles nombres muy descriptivos.

Todas las pruebas implementadas se han estado ejecutando con mucha frecuencia para asegurar que cada cambio no ha afectado a la lógica implementada hasta el momento. Por esto, las pruebas que se han desarrollado han sido lo más independientes y simples posible, ya que el objetivo es consumir el menor tiempo posible en ejecución de tests unitarios.

No obstante, se debe tener en cuenta que en algunos casos la implementación de las pruebas es más costoso que la ganancia que obtendría la calidad de la aplicación, por lo que 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 integración

Una vez todas las pruebas unitarias se han aprobado satisfactoriamente, se realizan las pruebas de integración. En éstas, se verifica que un subconjunto de componentes del sistema colaboran juntos y funcionan como se espera.

En este proyecto se han llevado a cabo las pruebas del sistema mediante el modelo de integración continua. Este paradigma consiste en automatizar las pruebas de integración, de manera que un software se encarga de descargar el código fuente de un repositorio de control de versiones, compilarlo, ejecutar pruebas y generar informes cada cierto tiempo. Esta práctica permite detectar errores en el código lo antes posible y conocer el estado del software en todo momento.

Figura 51. Logotipo de *Jenkins*

Para llevar a cabo la integración continua durante el desarrollo de este proyecto se ha utilizado el software Jenkins, que ha sido configurado de manera que cada vez que una nueva versión sea subida al repositorio de control de versiones que se le indica, se lleve a cabo una tarea concreta (*build*), que será la prueba de integración propiamente dicha. Esta prueba supone recompilar el código, ejecutar pruebas de unidad y, en caso de error, enviar un correo electrónico a los desarrolladores implicados en el proyecto.

6.2 Pruebas de aceptación

Las pruebas de aceptación de una aplicación las realizan generalmente un conjunto de usuarios finales y sirven para asegurarse de que la aplicación es capaz de llevar a cabo todas las historias de usuario establecidas durante la etapa de requisitos.

En este proyecto las pruebas de aceptación las han realizado tanto el alumno como el *Scrum Manager* al realizar el paso de cada tarea del estado "En pruebas" al estado "Hecho". Se ha detallado una lista con todas las pruebas a pasar para cada módulo implementado antes de la entrega del producto final, mediante las cuales se comprueban los siguientes casos (entre paréntesis se muestran los requisitos correspondientes de los descritos en el Anexo A):

- **Administración de plantas (IOTINDUST-7, IOTINDUST-10, IOTINDUST-11)**
  - Se listan correctamente las plantas a las que se tiene acceso.
  - Se pueden añadir y/o editar plantas sólo si el usuario autenticado es Administrador de IoTsens.
  - Se pueden añadir plantas rellenando todos los campos del formulario.
  - Se pueden añadir plantas rellenando sólo los campos obligatorios del formulario.
  - Se muestra un mensaje de error si el código introducido ya existe.
  - Se muestra un mensaje de error si no se han completado todos los campos obligatorios.
  - Se muestra un mensaje de error si se produce algún tipo de excepción ajena a la aplicación, como puede ser la imposibilidad de conectarse con la base de datos.
  - Al editar una planta, el sistema rellena automáticamente los campos del formulario con la información actual de la planta.
  - El sistema no permite editar el código de una planta.
  - Se puede editar una planta tanto rellenando todo los campos como sólo los obligatorios.
  - Se muestra un mensaje de éxito tanto cuando se añade como cuando se edita correctamente una planta.

- **Administración de usuarios (IOTINDUST-60, IOTINDUST-23, IOTINDUST-25, IOTINDUST-61)**

- Se listan correctamente los usuarios del sistema.
- Se pueden añadir, editar y/o eliminar usuarios sólo si el usuario autenticado es Administrador de IoTsens.
- Se puede añadir al sistema un usuario que pertenece a IoTsens.
- Se puede editar la información de un usuario, excepto su código.
- Se muestra un mensaje de error si el usuario que se intenta añadir no pertenece a IoTsens.
- Se muestra un mensaje de error si se produce alguna excepción ajena a la aplicación que impida que se complete la operación que se intenta realizar.
- Se pueden eliminar usuarios del sistema, tengan o no roles asignados en alguna planta.
- Se muestra un mensaje de confirmación antes de eliminar un usuario.
- Se muestra un mensaje de éxito tanto cuando se añade como cuando se edita o se elimina correctamente un usuario.

- **Gestión de roles (IOTINDUST-26, IOTINDUST-63)**
  - Se listan correctamente los roles de todos los usuarios de una planta.
  - Sólo se pueden asignar roles a un usuario que pertenezca al sistema.
  - Se puede tanto añadir un usuario a una planta como editar sus roles en dicha planta.
  - Se muestra un mensaje de éxito tanto cuando se añade un usuario a una planta como cuando se editan sus roles en dicha planta.
  - Se muestra un mensaje de error si se produce alguna excepción ajena a la aplicación que impida que se complete la operación de asignación o de edición de roles.
  - No se permite que un usuario Administrador se quite a sí mismo el rol de Administrador de la planta.
  - Se muestra un mensaje de confirmación antes de eliminar todos los roles de un usuario (lo que supone que dicho usuario deje de pertenecer a la planta).

- **Administración y visualización de señales (IOTINDUST-12, IOTINDUST-13, IOTINDUST-14, IOTINDUST-27, IOTINDUST-28, IOTNDUST-29)**
  - Se listan correctamente todas las señales de una planta de manera paginada.
  - Se puede tanto añadir una señal a una planta rellenando todos los campos del formulario como sólo los obligatorios.

- Se muestra un mensaje de error si el código introducido ya existe.
- Se muestra un mensaje de error si se ha dejado alguno de los campos obligatorios por rellenar.
- Se pueden ordenar las señales.
- Se pueden eliminar señales, aceptando previamente el mensaje de confirmación que muestra el sistema.
- Se muestran mensajes de éxito tanto cuando se lleva a cabo la inserción de una nueva señal como la edición o la eliminación de alguna de las existentes.
- Se muestra un mensaje de error si se produce alguna excepción ajena a la aplicación que impida que se complete la operación que se desea llevar a cabo.
- A través del listado se puede acceder a la visualización de cada una de las señales.
- Se visualizan correctamente las señales.
- Al visualizar una señal, se muestra un *feedbak* en forma de *spinner* mientras el sistema está obteniendo los datos a representar.
- Se puede seleccionar el periodo a graficar.
- Se muestra un mensaje informativo si durante el periodo seleccionado no existen datos.
- Se muestra un mensaje de error si, al seleccionar un periodo, se intenta introducir una fecha no válida o si la fecha "desde" introducida es posterior a la fecha "hasta".
- Se agrupan correctamente los datos cuando se selecciona alguna operación de agregación.

- **Descarga de datos en formato PNG o Excel (IOTINDUST-30, IOTINDUST-31)**
  - Se puede descargar la gráfica que se está visualizando como una imagen en formato PNG.
  - Se puede descargar la gráfica que se está visualizando como un archivo Excel.
  - En el archivo Excel generado, todas las columnas contienen datos correctos.

Capítulo 7

Conclusiones y trabajo futuro

Contenidos

En este capítulo se comentan las conclusiones a las que se ha llegado al final de la estancia, además de algunas mejoras que está previsto implementar en la aplicación IoT Industrial en el futuro.

7.1 Conclusiones personales

El propósito de este proyecto ha sido implementar una aplicación web enfocada a la gestión de plantas industriales. La consecución de los objetivos ha sido satisfactoria, y todo el trabajo se ha completado con éxito en un tiempo ligeramente menor al previsto, lo cual me ha hecho sentir bastante satisfecho.

En general, las sensaciones que me han transmitido tanto la empresa como el proyecto y la forma de llevarlo a cabo han sido positivas.

Por una parte, la estancia en prácticas me ha servido para desarrollar mis aptitudes para trabajar en un entorno colaborativo y para administrar correctamente los recursos de los que se dispone para llevar a cabo un proyecto de la ingeniería del software.

Creo que la metodología ágil utilizada permite cierta flexibilidad en el trabajo, por lo que es sencillo rehacer o mejorar partes del código de forma incremental, lo que a su vez favorece una planificación más precisa. Gracias a este tipo de trabajo iterativo e incremental se evita la sensación de incertidumbre, pudiendo identificar en cada momento el punto de desarrollo en el que el programador se encuentra.

Por otra parte, este proyecto me ha servido para crecer profesionalmente también en cuanto a tecnologías y herramientas de desarrollo. He aprendido a identificar soluciones para distintos problemas en el ámbito de las aplicaciones web, así como a desarrollar mi capacidad para llevarlas a cabo de manera competente.

7.1 Mejoras y trabajo futuro

Como se ha mencionado en otras partes de este documento, debido a la (relativamente) corta duración de la estancia en prácticas, el trabajo realizado engloba únicamente ciertos módulos de la aplicación I2OT, de la que que se espera terminar de implementar todas las funcionalidades requeridas por el cliente en los próximos meses. Algunas de las mejoras a incluir en el producto final son las siguientes:

- Paneles sinópticos que permitan visualizar en tiempo real el estado de ciertos componentes de las plantas industriales, como motores, electroválvulas o el nivel de los tanques de residuos.

- Sistema de alarmas para que se notifique vía e-mail o SMS a los usuarios que lo deseen cuando se cumpla una condición establecida. Por ejemplo, cuando la señal que indica el nivel de uno de los tanques sea demasiado baja.

- Generaciones de informes Excel a partir de plantilla y el envío automático de dichos informes a los destinatarios que se preestablezcan.

- Bidireccionalidad en las señales. Esto es, poder actuar de forma remota sobre las plantas a través de los paneles sinópticos.

Además, debido al crecimiento exponencial que está teniendo IoTsens, también se están valorando otras mejoras en cuanto a la metodología de trabajo y herramientas:

- Incluir *daily meetings* al ciclo de trabajo y reuniones conjuntas de planificación de los *sprints*, de forma que todo el equipo tenga conocimiento del resto de proyectos.

- Migrar los distintos proyectos a *Git* para facilitar el uso de *git-flow*, *feature-branches*[10] y *merge requests*[11] a través del software GitLab.

- Incluir *code reviews* por parte de todo el equipo para que toda la carga de trabajo relativa a la validación de tareas no caiga sobre el *Scrum Master*.

[10] Una *feature-branch* es una rama de desarrollo que se crea cuando se va a implementar una nueva funcionalidad, de manera que se evite tener código inestable en la rama de producción.

[11] Las *merge request* son las peticiones de validación de la funcionalidad implementada en una *feature branch*. Cuando el miembro del grupo al que se le ha asignado la *merge-request* valida dicha funcionalidad, puede unir la nueva rama con la rama de producción.

Bibliografía

[1] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos; “Context Aware Computing for The Internet of Things: A Survey (2013)”. Communications Surveys Tutorials, IEEE.

[2] Juha Taina, University of Helsinki. “Lines of code estimation for multiple programming languages”. https://www.cs.helsinki.fi/u/taina/ohtu/fp.html.

[3] Empower agile, empresa dedicada a promover los valores del desarrollo ágil. Ciclo del desarrollo guiado por pruebas. http://www.empoweragile.com.

[4] Roger S. Pressman; “Software engineering: A practitioner’s approach”.

[5] Wiki interna del Grupo Gimeno.

[6] Robert C. Martin; “Clean code: A Handbook of Agile Software Craftsmanship”.

[7] Bert Bates, Eric Freeman, Elisabeth Freeman; “Head First Design Patterns”, O’REILLY & ASSOCIATES, 2004.

[8] Nielsen, J, and Mack; Heuristic evaluation in “Usability Inspection Methods”, John Wiley & Sons, New York, NY.

Anexos

Contenidos

Anexo A

Requisito funcional	
Identificador	IOTINDUST-7
Nombre	Listado de plantas
Autores	Andrés Giustini
Fecha de creación	12/11/2016
Versión	1
Descripción	El sistema debe permitir ver un listado con las plantas industriales registradas
Nivel	Tarea principal
Actor principal	Administrador
Actores secundarios	Operario, Técnico, Cliente
Relaciones	IOTINDUST-10, IOTINDUST-11
Precondición	El usuario debe estar autenticado en el sistema de autenticación central de IoTsens y, además, debe estar registrado en IoTIndustrial
Trigger	Un usuario registrado y autenticado desea ver el listado de plantas industriales
Secuencia normal	Mostrar listado de plantas industriales
	1 El usuario accede a la aplicación **2** El usuario selecciona la pestaña “Plantas” de la barra de navegación de la aplicación **3** El sistema muestra un listado con las plantas industriales registradas en el sistema
Excepciones	-
Frecuencia esperada	Varias veces al día
Importancia	Muy alta
Prioridad	A corto plazo
Comentarios	En el listado se mostrarán sólo aquellas plantas para las que el usuario autenticado tiene algún tipo de rol asignado (Administrador, Operario, Técnico o Cliente)

Tabla 6. Requisito funcional IOTINDIST-7

Requisito funcional	
Identificador	IOTINDUST-10
Nombre	Crear una nueva planta
Autores	Andrés Giustini
Fecha de creación	12/11/2016
Versión	1
Descripción	La aplicación debe permitir añadir plantas de manera que queden registradas en el sistema
Nivel	Tarea principal
Actor principal	Administrador
Actores secundarios	-
Relaciones	IOTINDUST-7, IOTINDUST-11
Precondición	El usuario debe estar autenticado en el sistema de autenticación central de IoTsens, debe estar registrado en IoTIndustrial y, además, debe ser administrador en IoTsens
Trigger	Un usuario administrador de IoTsens, registrado y autenticado en la aplicación desea añadir una nueva planta industrial en el sistema
Secuencia normal	Añadir una nueva planta industrial
	1 El usuario accede a la aplicación **2** El usuario selecciona la pestaña "Plantas" de la barra de navegación de la aplicación **3** Si el usuario es administrador, en la parte inferior derecha del listado de plantas aparecerá un botón "Añadir" **4** El usuario pulsa el botón y el sistema muestra un formulario de alta de plantas industriales **5** El usuario rellena los campos y envía el formulario **6** El sistema comprueba los campos y registra la nueva planta
Excepciones	
	1 La planta ya existe en el sistema y se muestra un mensaje informativo **2** Alguno de los campos obligatorios no se ha rellenado y el sistema muestra un mensaje advirtiendo al usuario
Frecuencia esperada	Cada vez que se instale una nueva planta industrial
Importancia	Media
Prioridad	A corto plazo
Comentarios	En caso de que el usuario no sea administrador en IoTsens no se mostrará el botón "Añadir", lo que impide posibles excepciones en la validación por parte del servidor

Tabla 7. Requisito funcional IOTINDIST-10

Requisito funcional	
Identificador	IOTINDUST-11
Nombre	Editar propiedades de la planta
Autores	Andrés Giustini
Fecha de creación	12/11/2016
Versión	1
Descripción	La aplicación debe permitir editar plantas de manera que sus datos se actualicen en el sistema
Nivel	Tarea principal
Actor principal	Administrador
Actores secundarios	-
Relaciones	IOTINDUST-7, IOTINDUST-10
Precondición	El usuario debe estar autenticado en el sistema de autenticación central de IoTsens, debe estar registrado en IoTIndustrial y, además, debe ser administrador en IoTsens
Trigger	Un usuario administrador de IoTsens, registrado y autenticado en la aplicación desea editar una nueva planta de las presentes en el sistema
Secuencia normal	Editar una planta industrial
	1 El usuario accede a la aplicación **2** El usuario selecciona la pestaña "Plantas" de la barra de navegación de la aplicación **3** Si el usuario es administrador, a la derecha de cada planta del listado aparecerá un icono con forma de lápiz **4** El usuario pulsa el botón y el sistema muestra un formulario de edición de la planta con los datos de la misma cargados en los campos correspondientes **5** El usuario modifica los campos que desea actualizar y envía el formulario **6** El sistema comprueba los campos y actualiza la planta
Excepciones	
	1 Alguno de los campos obligatorios no se ha rellenado y el sistema muestra un mensaje advirtiendo al usuario
Frecuencia esperada	Pocas veces al mes
Importancia	Media
Prioridad	A corto plazo
Comentarios	En caso de que el usuario no sea administrador en IoTsens no se mostrará el botón para editar, lo que impide posibles excepciones en la validación por parte del servidor Además, no se permite editar el código de la planta, pues dicho código debe ser único e invariable

Tabla 8. Requisito funcional IOTINDIST-11

Requisito funcional	
Identificador	IOTINDUST-60
Nombre	Listado de usuarios de la aplicación
Autores	Andrés Giustini
Fecha de creación	12/11/2016
Versión	1
Descripción	El sistema debe permitir ver un listado con los usuarios registrados en la aplicación
Nivel	Tarea principal
Actor principal	Administrador
Actores secundarios	Operario, Técnico, Cliente
Relaciones	IOTINDUST-23, IOTINDUST-25, IOTINDUST-61
Precondición	El usuario debe estar autenticado en el sistema de autenticación central de IoTsens y, además, debe estar registrado en IoTIndustrial
Trigger	Un usuario registrado y autenticado desea ver el listado de usuarios
Secuencia normal	Listar los usuarios de la aplicación
	1 El usuario accede a la aplicación **2** El usuario selecciona la pestaña “Plantas” de la barra de navegación de la aplicación **3** El sistema muestra un listado con las plantas industriales registradas en el sistema
Excepciones	-
Frecuencia esperada	Varias veces al mes
Importancia	Alta
Prioridad	A corto plazo
Comentarios	-

Tabla 9. Requisito funcional IOTINDIST-60

Requisito funcional	
Identificador	IOTINDUST-23
Nombre	Crear usuarios
Autores	Andrés Giustini
Fecha de creación	12/11/2016
Versión	1
Descripción	La aplicación debe permitir añadir usuarios de manera que queden registrados en el sistema
Nivel	Tarea principal
Actor principal	Administrador
Actores secundarios	-
Relaciones	IOTINDUST-60, IOTINDUST-25, IOTINDUST-61
Precondición	El usuario debe estar autenticado en el sistema de autenticación central de IoTsens, debe estar registrado en IoTIndustrial y, además, debe ser administrador en IoTsens
Trigger	Un usuario administrador de IoTsens, registrado y autenticado en la aplicación desea añadir un nuevo usuario
Secuencia normal	Añadir una nuevo usuario
	1 El usuario accede a la aplicación **2** El usuario selecciona la pestaña "Usuarios" de la barra de navegación de la aplicación **3** Si el usuario es administrador, en la parte inferior derecha del listado de usuarios aparecerá un botón "Añadir" **4** El usuario pulsa el botón y el sistema muestra un formulario de alta de usuarios **5** El usuario rellena los campos y envía el formulario **6** El sistema comprueba los campos y registra el nuevo usuario
Excepciones	
	1 El usuario que se desea dar de alta ya existe en el sistema y se muestra un mensaje informativo **2** Alguno de los campos obligatorios no se ha rellenado y el sistema muestra un mensaje notificándolo **3** El usuario que se desea dar de alta no está registrado previamente en la base de datos central de IoTsens
Frecuencia esperada	Pocas veces al mes
Importancia	Media
Prioridad	A corto plazo
Comentarios	En caso de que el usuario no sea administrador en IoTsens no se mostrará el botón "Añadir", lo que impide posibles excepciones en la validación por parte del servidor

Tabla 10. Requisito funcional IOTINDIST-23

Requisito funcional	
Identificador	IOTINDUST-25
Nombre	Editar propiedades de un usuario
Autores	Andrés Giustini
Fecha de creación	12/11/2016
Versión	1
Descripción	La aplicación debe permitir editar plantas de manera que sus datos se actualicen en el sistema
Nivel	Tarea principal
Actor principal	Administrador
Actores secundarios	-
Relaciones	IOTINDUST-60, IOTINDUST-23, IOTINDUST-61
Precondición	El usuario debe estar autenticado en el sistema de autenticación central de IoTsens, debe estar registrado en IoTIndustrial y, además, debe ser administrador en IoTsens
Trigger	Un usuario administrador de IoTsens, registrado y autenticado en la aplicación desea editar la información de uno de los usuarios
Secuencia normal	Editar un usuario
	1 El usuario accede a la aplicación **2** El usuario selecciona la pestaña "Usuarios" de la barra de navegación de la aplicación **3** Si el usuario es administrador, a la derecha de cada usuario del listado aparecerá un icono con forma de lápiz **4** El usuario pulsa el botón y el sistema muestra un formulario de edición de la información del usuario con los datos del mismo cargados en los campos correspondientes **5** El usuario modifica los campos que desea actualizar y envía el formulario **6** El sistema comprueba los campos y actualiza el usuario
Excepciones	
	1 Alguno de los campos obligatorios no se ha rellenado y el sistema muestra un mensaje advirtiendo al usuario
Frecuencia esperada	Pocas veces al mes
Importancia	Media
Prioridad	A corto plazo
Comentarios	En caso de que el usuario no sea administrador en IoTsens no se mostrará el botón para editar, lo que impide posibles excepciones en la validación por parte del servidor Además, no se permite editar el código del usuario, pues dicho código debe ser único e invariable

Tabla 11. Requisito funcional IOTINDIST-25

Requisito funcional	
Identificador	IOTINDUST-61
Nombre	Eliminar usuarios
Autores	Andrés Giustini
Fecha de creación	12/11/2016
Versión	1
Descripción	El sistema debe permitir borrar uno de los usuarios registrados en la aplicación
Nivel	Tarea principal
Actor principal	Administrador
Actores secundarios	-
Relaciones	IOTINDUST-23, IOTINDUST-25, IOTINDUST-61
Precondición	El usuario debe estar autenticado en el sistema de autenticación central de IoTsens, debe estar registrado en IoTIndustrial y, además, debe ser administrador en IoTsens
Trigger	Un usuario administrador de IoTsens, registrado y autenticado en la aplicación desea eliminar un usuario
Secuencia normal	Eliminar un usuario de la aplicación
	1 El usuario accede a la aplicación **2** El usuario selecciona la pestaña "Usuarios" de la barra de navegación de la aplicación **3** Si el usuario es administrador, a la derecha de cada usuario del listado aparecerá un icono con forma de cruz **4** El usuario pulsa el botón y el sistema muestra una ventana de confirmación **5** El usuario acepta la advertencia de que se va a eliminar el usuario seleccionado **6** El sistema elimina el usuario
Excepciones	**1** El usuario se está intentando eliminar a sí mismo, por lo que la aplicación advierte de que dicha operación no está permitida
Frecuencia esperada	Pocas veces al año
Importancia	Alta
Prioridad	A corto plazo
Comentarios	-

Tabla 12. Requisito funcional IOTINDIST-61

Requisito funcional	
Identificador	IOTINDUST-26
Nombre	Asignar a un usuario un rol (o varios) en una planta
Autores	Andrés Giustini
Fecha de creación	12/11/2016
Versión	1
Descripción	El sistema debe permitir asignar a los usuarios registrados una serie de roles establecidos (Administrador, Técnico, Operario o Cliente). Esta operación también se puede denominar como "Añadir usuarios a una planta"
Nivel	Tarea principal
Actor principal	Administrador
Actores secundarios	-
Relaciones	IOTINDUST-23, IOTINDUST-63
Precondición	El usuario debe estar autenticado en el sistema de autenticación central de IoTsens, debe estar registrado en IoTIndustrial y, además, debe tener asignado el rol de Administrador en la planta a la que quiere añadir usuarios
Trigger	Un usuario de una planta, registrado y autenticado como Administrador en la aplicación desea añadir usuarios a dicha planta
Secuencia normal	Asignar roles a usuarios para una planta
	1 El usuario accede a la aplicación **2** El usuario selecciona la pestaña "Plantas" de la barra de navegación de la aplicación **3** En el listado de plantas, se accede a la que se desea añadir **4** usuarios El usuario selecciona la pestaña "Usuarios" de la barra de **5** navegación de la planta en la que se encuentra En la parte inferior del listado de usuarios de la planta aparecerá **6** un botón "Añadir" **7** El usuario pulsa el botón El sistema muestra un formulario donde el usuario marca los
Excepciones	roles que quiere asignar y envía el formulario -
Frecuencia esperada	Pocas veces al mes
Importancia	Alta
Prioridad	A corto plazo
Comentarios	-

Tabla 13. Requisito funcional IOTINDIST-26

Requisito funcional	
Identificador	IOTINDUST-63
Nombre	Modificar los roles de un usuario en una planta
Autores	Andrés Giustini
Fecha de creación	12/11/2016
Versión	1
Descripción	El sistema debe permitir editar los roles de los usuarios de una planta. Esta operación también se puede denominar como "Editar usuarios de una planta"
Nivel	Tarea principal
Actor principal	Administrador
Actores secundarios	-
Relaciones	IOTINDUST-23, IOTINDUST-26
Precondición	El usuario debe estar autenticado en el sistema de autenticación central de IoTsens, debe estar registrado en IoTIndustrial y, además, debe tener asignado el rol de Administrador en la planta que quiere editar los usuarios
Trigger	Un usuario de una planta, registrado y autenticado como Administrador en la aplicación desea editar los usuarios de dicha planta
Secuencia normal	Modificar los roles de los usuarios de una planta
	1 El usuario accede a la aplicación
	2 El usuario selecciona la pestaña "Plantas" de la barra de navegación de la aplicación
	3 En el listado de plantas, se accede a la que se desea añadir
	4 usuarios
	El usuario selecciona la pestaña "Usuarios" de la barra de
	5 navegación de la planta en la que se encuentra
	A la derecha de cada usuario se mostrará un icono en forma de
	6 lápiz
	El usuario pulsa el icono si quiere modificar los roles de alguno
	7 de los usuarios existentes en la planta
	El sistema muestra un formulario donde el usuario marca/desmarca los roles que quiere asignar/eliminar y envía el
Excepciones	formulario
	1
	El usuario se está intentando eliminar a sí mismo el rol de administrador
Frecuencia esperada	Pocas veces al mes
Importancia	Alta
Prioridad	A corto plazo
Comentarios	-

Tabla 14. Requisito funcional IOTINDIST-63

Requisito funcional	
Identificador	IOTINDUST-12
Nombre	Listado paginado de señales de una planta
Autores	Andrés Giustini
Fecha de creación	16/11/2016
Versión	1
Descripción	El sistema debe mostrar las señales de cada planta mediante un listado paginado, donde cada una de las páginas sea de, como máximo, 10 señales
Nivel	Tarea principal
Actor principal	Operario, Técnico, Cliente
Actores secundarios	Administrador
Relaciones	IOTINDUST-13, IOTINDUST-14, IOTINDUST-27
Precondición	El usuario debe estar autenticado en el sistema de autenticación central de IoTsens, debe estar registrado en IoTIndustrial y, además, debe tener asignado algún rol en la planta cuyas señales quiere consultar
Trigger	Un usuario de una planta, registrado y autenticado en la aplicación desea consultar las señales de una planta
Secuencia normal	Ver las señales de una planta
	1 El usuario accede a la aplicación **2** El usuario selecciona la pestaña “Plantas” de la barra de navegación de la aplicación **3** En el listado de plantas, se accede a la planta de la que se quiere ver sus señales **4** El usuario selecciona la pestaña “Señales” de la barra de navegación de la planta en la que se encuentra **5** El sistema muestra un listado paginado con las señales de la planta en la que se encuentra
Excepciones	-
Frecuencia esperada	Varias veces al día
Importancia	Alta
Prioridad	A corto plazo
Comentarios	-

Tabla 15. Requisito funcional IOTINDIST-12

Requisito funcional	
Identificador	IOTINDUST-13
Nombre	Crear una nueva señal de la planta
Autores	Andrés Giustini
Fecha de creación	16/11/2016
Versión	1
Descripción	El sistema debe permitir añadir señales a las plantas industriales registradas
Nivel	Tarea principal
Actor principal	Operario, Técnico
Actores secundarios	Administrador
Relaciones	IOTINDUST-12, IOTINDUST-14, IOTINDUST-27
Precondición	El usuario debe estar autenticado en el sistema de autenticación central de IoTsens, debe estar registrado en IoTIndustrial y, además, debe tener asignado el rol de Administrador en la planta
Trigger	Un usuario administrador de IoTsens, registrado y autenticado en la aplicación desea añadir una nueva señal a una planta
Secuencia normal	Añadir una señal a una planta
Excepciones	**1** Una vez se ha accedido a la planta, el usuario selecciona la pestaña “Señales” de la barra de navegación **3** El sistema muestra un listado paginado con las señales de la planta en la que se encuentra **4** Si el usuario es administrador, en la parte inferior derecha del listado de señales aparecerá un botón “Añadir” **5** El usuario pulsa el botón y el sistema muestra un formulario de alta de señales **6** El usuario rellena los campos y envía el formulario **7** El sistema comprueba los campos y añade la nueva señal a la planta **1** La señal ya existe en el sistema y se muestra un mensaje informativo **2** Alguno de los campos obligatorios no se ha rellenado y el sistema muestra un mensaje advirtiendo al usuario
Frecuencia esperada	Varias veces al día cuando se acaba de instalar una nueva planta industrial. Después, pocas veces al año
Importancia	Alta
Prioridad	A corto plazo
Comentarios	-

Tabla 16. Requisito funcional IOTINDIST-13

Requisito funcional	
Identificador	IOTINDUST-14
Nombre	Editar propiedades de una señal de la planta
Autores	Andrés Giustini
Fecha de creación	16/11/2016
Versión	1
Descripción	El sistema debe permitir editar las propiedades de las señales de una planta
Nivel	Tarea principal
Actor principal	Operario, Técnico
Actores secundarios	Administrador
Relaciones	IOTINDUST-12, IOTINDUST-13, IOTINDUST-27
Precondición	El usuario debe estar autenticado en el sistema de autenticación central de IoTsens, debe estar registrado en IoTIndustrial y, además, debe tener asignado el rol de Administrador en la planta
Trigger	Un usuario de IoTsens, registrado y autenticado en la aplicación desea editar una de las señales de una planta
Secuencia normal	Editar una señal de una planta
Excepciones	**1** Una vez se ha accedido a la planta, el usuario selecciona la pestaña "Señales" de la barra de navegación **3** El sistema muestra un listado paginado con las señales de la planta en la que se encuentra **4** Si el usuario es administrador, en la parte derecha de cada señal aparecerá un icono con forma de lápiz **5** El usuario pulsa el icono y el sistema muestra un formulario de edición de la señal con los datos de la misma cargados en los campos correspondientes **6** El usuario rellena los campos y envía el formulario **7** El sistema comprueba los campos y añade la nueva señal a la planta -
Frecuencia esperada	Pocas veces al año
Importancia	Alta
Prioridad	A corto plazo
Comentarios	En caso de que el usuario no tenga el rol de Administrador en la planta no se mostrará el botón para editar, lo que impide posibles excepciones en la validación por parte del servidor Además, no se permite editar el código de la señal, pues dicho código debe ser único e invariable

Tabla 17. Requisito funcional IOTINDIST-14

Requisito funcional	
Identificador	IOTINDUST-27
Nombre	Visualización en gráfica de líneas de los valores de una señal
Autores	Andrés Giustini
Fecha de creación	23/11/2016
Versión	1
Descripción	El sistema debe permitir visualizar gráficamente los valores de las señales durante un periodo de tiempo definido
Nivel	Tarea principal
Actor principal	Operario, Técnico, Cliente
Actores secundarios	Administrador
Relaciones	IOTINDUST-12, IOTINDUST-13, IOTINDUST-24, IOTINDUST-28, IOTINDUST-29, IOTINDUST-30, IOTINDUST-31
Precondición	El usuario debe estar autenticado en el sistema de autenticación central de IoTsens, debe estar registrado en IoTIndustrial y, además, debe tener asignado algún rol en la planta que contiene la señal
Trigger	Un usuario de IoTsens, registrado y autenticado en la aplicación desea visualizar gráficamente una señal
Secuencia normal	Visualizar una señal de una planta
	1 Una vez se ha accedido a la planta, el usuario selecciona la pestaña "Señales" de la barra de navegación **3** El sistema muestra un listado paginado con las señales de la planta en la que se encuentra **4** En la parte derecha de cada señal aparecerá un triángulo a modo de icono de "play" **5** El usuario pulsa el icono y el sistema carga una gráfica con los datos de la señal durante las últimas 6 horas
Excepciones	En caso de que no se hayan recibido datos de esa señal durante las últimas 6 horas, el sistema mostrará un mensaje indicándolo
Frecuencia esperada	Varias veces al día
Importancia	Alta
Prioridad	A corto plazo
Comentarios	-

Tabla 18. Requisito funcional IOTINDIST-27

Requisito funcional	
Identificador	IOTINDUST-28
Nombre	Selección periodo a graficar
Autores	Andrés Giustini
Fecha de creación	23/11/2016
Versión	1
Descripción	El sistema debe permitir seleccionar el periodo de la gráfica de visualización de los datos de una señal
Nivel	Tarea principal
Actor principal	Operario, Técnico, Cliente
Actores secundarios	Administrador
Relaciones	IOTINDUST-27, IOTINDUST-29, IOTINDUST-30, IOTINDUST-31
Precondición	El usuario debe estar autenticado en el sistema de autenticación central de IoTsens, debe estar registrado en IoTIndustrial y, además, debe tener asignado algún rol en la planta que contiene la señal
Trigger	Un usuario de IoTsens, registrado y autenticado en la aplicación desea visualizar gráficamente los datos de una señal durante un periodo personalizado
Secuencia normal	Seleccionar el periodo a gaficar una señal
	1 Una vez se ha accedido a la gráfica de datos de una señal, en la parte inferior de la misma aparecerá un formulario de selección de periodo **2** El usuario introduce las fechas y horas “Desde” y “Hasta” y envía el formulario **3** El sistema comprueba los campos y refresca la gráfica teniendo en cuenta el nuevo periodo
Excepciones	El usuario introduce una fecha “Desde” posterior a la fecha “Hasta”, por lo que el sistema muestra un mensaje informando del error
Frecuencia esperada	Varias veces al día
Importancia	Alta
Prioridad	A corto plazo
Comentarios	-

Tabla 19. Requisito funcional IOTINDIST-28

Requisito funcional	
Identificador	IOTINDUST-29
Nombre	Selección agrupación de los datos (periodo y operación)
Autores	Andrés Giustini
Fecha de creación	23/11/2016
Versión	1
Descripción	El sistema debe permitir realizar agrupaciones con los datos de la gráfica. Una agrupación es un resumen de los datos durante un cierto periodo aplicando una determinada operación. Por ejemplo, mostrar los datos en la gráfica donde para cada *día* se mostrará un único valor que se corresponderá a la *media* de los datos originales
Nivel	Tarea principal
Actor principal	Operario, Técnico, Cliente
Actores secundarios	Administrador
Relaciones	IOTINDUST-27, IOTINDUST-28, IOTINDUST-30, IOTINDUST-31
Precondición	El usuario debe estar autenticado en el sistema de autenticación central de IoTsens, debe estar registrado en IoTIndustrial y, además, debe tener asignado algún rol en la planta que contiene la señal
Trigger	Un usuario de IoTsens, registrado y autenticado en la aplicación desea visualizar gráficamente los datos de una señal de manera agrupada
Secuencia normal	Seleccionar un determinado periodo y una operación de agrupación para los datos de una señal
	1 Una vez se ha accedido a la gráfica de datos de una señal, en la parte inferior de la misma aparecerá un formulario de agrupación de los datos **2** El usuario introduce el periodo de la agrupación (minutos, horas, días, etc.) y una operación (media, máximo, mínimo, cuenta, etc.) de las disponibles **3** El sistema comprueba los campos y refresca la gráfica teniendo en cuenta la agrupación introducida
Excepciones	-
Frecuencia esperada	Varias veces al día
Importancia	Alta
Prioridad	A corto plazo
Comentarios	Si se introduce un periodo de agrupación más pequeño que el intervalo que existe entre la recepción de un dato y el siguiente, el sistema tomará como periodo de agrupación dicho intervalo

Tabla 20. Requisito funcional IOTINDIST-29

Requisito funcional	
Identificador	IOTINDUST-30
Nombre	Descargar en Excel los datos visualizados en la gráfica
Autores	Andrés Giustini
Fecha de creación	23/11/2016
Versión	1
Descripción	El sistema debe permitir descargar los datos de una señal en un archivo .xls
Nivel	Tarea principal
Actor principal	Operario, Técnico, Cliente
Actores secundarios	Administrador
Relaciones	IOTINDUST-27, IOTINDUST-28, IOTINDUST-29, IOTINDUST-31
Precondición	El usuario debe estar autenticado en el sistema de autenticación central de IoTsens, debe estar registrado en IoTIndustrial y, además, debe tener asignado algún rol en la planta que contiene la señal
Trigger	Un usuario de IoTsens, registrado y autenticado en la aplicación desea descargar los datos de la señal en forma de tabla Excel
Secuencia normal	Descargar los datos de una señal en una tabla Excel
Excepciones	**1** Una vez se ha accedido a la gráfica de datos de una señal, en la parte inferior de la misma aparecerá un botón “Descargar Excel” **2** El usuario pulsa el botón y el sistema genera un Excel con los datos cargados en la gráfica y lo envía al navegador para que lo descargue -
Frecuencia esperada	Varias veces al día
Importancia	Alta
Prioridad	A corto plazo
Comentarios	En la tabla Excel aparecerán dos columnas: “Fecha” y “Valor”. Cada fila de la tabla se corresponderá con un dato de la señal, para el cual se mostrará la fecha en la que se ha recibido el dato y su valor

Tabla 21. Requisito funcional IOTINDIST-30

Requisito funcional	
Identificador	IOTINDUST-31
Nombre	Descargar imagen con la gráfica que se está visualizando
Autores	Andrés Giustini
Fecha de creación	23/11/2016
Versión	1
Descripción	El sistema debe permitir descargar la gráfica de una señal en un archivo .png
Nivel	Tarea principal
Actor principal	Operario, Técnico, Cliente
Actores secundarios	Administrador
Relaciones	IOTINDUST-27, IOTINDUST-28, IOTINDUST-29, IOTINDUST-30
Precondición	El usuario debe estar autenticado en el sistema de autenticación central de IoTsens, debe estar registrado en IoTIndustrial y, además, debe tener asignado algún rol en la planta que contiene la señal
Trigger	Un usuario de IoTsens, registrado y autenticado en la aplicación desea obtener una imagen con la gráfica de una señal
Secuencia normal	Descargar la gráfica de una señal en una imagen
	1 Una vez se ha accedido a la gráfica de datos de una señal, en la parte inferior de la misma aparecerá un botón “Descargar PNG” **2** El usuario pulsa el botón y el sistema genera una imagen con la gráfica que se está visualizando y lo envía al navegador para que lo descargue
Excepciones	-
Frecuencia esperada	Varias veces al día
Importancia	Alta
Prioridad	A corto plazo
Comentarios	-

Tabla 22. Requisito funcional IOTINDIST-31

Anexo B

Requisito de datos	
Código	DR01
Nombre	Planta industrial
Datos	Para cada planta se deberá almacenar su código único, el nombre, la dirección en formato texto, las coordenadas y opcionalmente una imagen a modo de logo.
Comentarios	Cada planta industrial se corresponde a un sensor de IoTsens.

Tabla 23. Requisito de datos DR01

Requisito de datos	
Código	DR02
Nombre	Usuario
Datos	Para cada usuario, se almacenará únicamente su código único y su nombre, pues el resto de datos (tales como el email, el número de teléfono o si es un usuario administrador) están almacenados en IoTsens y se leerán de dicha base de datos en caso de ser necesario
Comentarios	-

Tabla 24. Requisito de datos DR02

Requisito de datos	
Código	DR03
Nombre	Señal
Datos	Cada uno de los sensores (plantas) envían datos para diferentes variables. En IoTIndustrial las variables se denominan señales y, para cada una de ellas, se debe almacenar su código único, una descripción, la unidad de los datos que se reciben y su naturaleza (digital o analógica)
Comentarios	-

Tabla 25. Requisito de datos DR03

Requisito de datos	
Código	DR04
Nombre	Consigna
Datos	Las consignas son señales que se envían desde la aplicación a las plantas industriales, por lo que para cada una se almacenarán los mismos datos que para las señales.
Comentarios	La funcionalidad propia de las consignas queda fuera del alcance de este proyecto, aunque de momento se deben tener en cuenta en la gestión de señales, pues son un tipo de señal

Tabla 26. Requisito de datos DR04

Requisito de datos	
Código	DR05
Nombre	GrantedRol
Datos	Con el fin de controlar los permisos de cada usuario, es necesario mantener la relación de éstos con las plantas y los posibles roles. Así pues el sistema deberá almacenar el código de una planta, el código de un usuario y el nombre del rol para cada rol que se asigne
Comentarios	-

Tabla 27. Requisito de datos DR05