FACULTAD DE CIENCIAS

Facultad de Ciencias

Proyecto Fin de Carrera

Aplicación móvil para un Sistema de Gestión de Consumo Energético
(Mobile application for an Energetic Use Management System)

Para acceder al Título de

INGENIERO EN INFORMÁTICA

Autor: María Pérez Peñil
Director: Marta Elena Zorrilla Pantaleón
Codirector: Verónica Román Morante

Febrero - 2016

Índice de contenidos

Agradecimientos 6

Resumen 7

Abstract 8

1 Introducción 9

1.1 Motivación y objetivos 9

1.2 Metodologías 10

1.3 Contenido de la memoria 11

2 Tecnologías y herramientas 12

2.1 Tecnologías 12

2.1.1 MySQL 12

2.1.3 Sails 13

2.1.4 Apache Cordova 13

2.1.5 AngularJS 14

2.1.6 Onsen UI 14

2.2 Herramientas 14

2.2.1 Eclipse 14

2.2.2 Android SDK 15

2.2.3 MySQL Workbench 15

2.2.4 Sublime Text 15

2.2.5 MagicDraw UML 15

3 Análisis del sistema 16

3.1 Especificación de casos de uso 16

3.2 Especificación de requisitos funcionales 21

3.2 Especificación de requisitos no funcionales 22

4 Diseño del sistema 23

4.1 Diseño arquitectónico 23

4.2 Diseño detallado 25

5 Implementación del sistema 30

5.1 Capa de persistencia 30

5.2 Capa de negocio 31

5.3 Capa de presentación 34

5.3.1 Aplicación web 34

5.3.2 Aplicación móvil 39

5.4 Implementación de rutinas 44

5.4.1 Rutinas de volcado de datos 44

5.4.2 Rutinas de envío de alarmas 45

6 Pruebas 48

6.1 Pruebas unitarias 48

6.2 Pruebas de integración 48

6.3 Pruebas de sistema 49

6.4 Pruebas de aceptación 49

7 Conclusiones y trabajos futuros 50

7.1 Conclusiones 50

7.2 Trabajos futuros 51

Referencias 52

Índice de ilustraciones

Ilustración 1. Esquema del ciclo de vida iterativo 11
Ilustración 2. Diagrama de usuarios de la aplicación 16
Ilustración 3. Diagrama de gestión de locales 17
Ilustración 4. Diagrama de gestión de alarmas 18
Ilustración 5. Diagrama de gestión de usuarios 18
Ilustración 6. Esquema de arquitectura en tres capas 24
Ilustración 7. Arquitectura MVC 25
Ilustración 8. Diagrama de despliegue del sistema 26
Ilustración 9. Diagrama de componentes de la aplicación web 27
Ilustración 10. Diagrama de componentes de la aplicación móvil 28
Ilustración 11. Diseño UML de la base de datos 29
Ilustración 12. Estructura de clases del servidor 31
Ilustración 13. Modelo de la tabla alarmas producidas 32
Ilustración 14. Hoja de rutas del servidor 33
Ilustración 15. Implementación de función del servidor 33
Ilustración 16. Implementación de función del servidor 34
Ilustración 17. Plantilla de la aplicación web 35
Ilustración 18. Función del controlador de la aplicación web 36
Ilustración 19. Pantalla de acceso a la aplicación web 36
Ilustración 20. Pantalla de gestión de alarmas 37
Ilustración 21. Pantalla de creación de alarmas 37
Ilustración 22. Pantalla de gestión de usuarios 38
Ilustración 23. Pantalla de alta de usuarios 38
Ilustración 24. Pantalla de gestión de permisos de usuario 39
Ilustración 25. Plantilla de la aplicación móvil 39
Ilustración 26. Función del controlador de la aplicación móvil 40
Ilustración 27, Llamada a la API web 41
Ilustración 28. Pantalla de acceso a la aplicación móvil 41
Ilustración 29. Pantallas de menú de opciones 42
Ilustración 30. Pantallas de consulta de datos 42
Ilustración 31. Pantalla de comparativas 43
Ilustración 32. Pantalla de alarmas producidas 44
Ilustración 33. Rutina de volcado de datos 45
Ilustración 34. Función para almacenar alarmas 46
Ilustración 35. Función para enviar notificación 46
Ilustración 36. Procedimiento de envío de notificación 47

Índice de tablas

Tabla 1. Especificación de caso de uso 'Consultar medidas 19
Tabla 2. Especificación de caso de uso 'Crear usuario' 20
Tabla 3. Especificación de requisitos funcionales 22
Tabla 4. Especificación de requisitos no funcionales 22

Agradecimientos

Me gustaría dar las gracias a todas las personas que se han cruzado en mi vida durante todos mis años de aprendizaje, pues, en mayor o menor medida, ellas han formado parte de este camino.

A mi familia y amigos, por ser un pilar fundamental y formar parte de mi día a día.

Al equipo de Netkia Soluciones SL, que ha confiado en mí para realizar este proyecto. En especial a Jonatan e Iván, por compartir conmigo su conocimiento.

Por último, pero no menos importante, a todos los maestros y profesores que han tomado parte en mi educación, pues todos ellos han contribuido a construir la persona que soy y seré.

Resumen

El proyecto que aquí se expone consiste en el desarrollo completo de un sistema software, orientado al control del consumo energético en empresas, construido para 'Xener', una plataforma, de la compañía ITM, dedicada al sector de la energía.
En este documento se realiza una descripción general de todo el proceso, partiendo desde su propuesta hasta alcanzar su puesta en marcha.

El gasto energético es un aspecto clave para la economía y el desarrollo de una compañía. Y su control, en todo momento, en cualquier lugar, es posible gracias a los dispositivos móviles.
ITM deseaba añadir a los servicios de la plataforma 'Xener' una aplicación móvil que tratase unos datos recogidos por un conjunto de medidores, repartidos en varios locales a lo largo de la península ibérica.
Esta aplicación, se encargaría de mostrar, en tiempo real, los valores de las principales magnitudes relacionadas con el consumo de energía de los locales, así como cierta información relevante para su control.
Además, el software debía implementar un sistema de alarmas, que permitiese controlar que los valores recogidos estuviesen siempre dentro de unos límites establecidos, enviando una notificación al dispositivo móvil en caso de no cumplirse las condiciones deseadas.
De esta forma, sus clientes tendrían, en todo momento y lugar, el control energético de sus franquicias y locales.

El proyecto ha sido desarrollado en su totalidad en la empresa 'Netkia Soluciones SL' y consta de una aplicación móvil y una aplicación web. La aplicación móvil, que recoge los datos de una API web, previamente implementada, se apoya en el gestor web, que permite la administración de sus distintas funcionalidades, como los usuarios o las alarmas.

Palabras clave: aplicación móvil, control de consumo energético, gestión de alarmas, aplicación web, tecnología Sails, tecnología Cordova.

Abstract

The project referred to above, consists in the full development of a software system, oriented towards the energy use control in enterprises, built for 'Xener', an ITM platform, which is dedicated to the energy sector.
This document describes the whole process from the proposal to the launch.

Energy use is a key part of a company's development and economy. And it's control, at any time and anywhere, is now possible because of mobile devices.
ITM wanted, for Xener platform, a mobile application to represent data collected by an amount of energy meters, located on a group of business premises throughout the country.
This application, must show, in real time real time, values of the main electrical variables, related to energy use in the business premises, as well as some relevant information as to its control.
Also, the software must contain an alarm system, which will ensure the measured are inside the established limits, sending a notification to the mobile device when the value is outside of these limits.
This way, their customers will have, at anytime and anywhere, control of the energy use in their companies.

The project has been fully developed in the Company 'Netkia Soluciones SL', and consists of a mobile application and a web appliction.
The mobile application, which takes the data from a web API, already implemented by Xener, relies on the web application, which allows the management of its funcionalities, as users and alarms.

Keywords: mobile application, energy use control, alarms management, web application, Sails technologies, Cordova technologies.

1 Introducción

1.1 Motivación y objetivos

En los últimos años, la venta de dispositivos móviles ha experimentado un notable auge debido a la aparición de los Smartphone. Atrás quedan aquellos años en los que un teléfono móvil era un simple instrumento de comunicación. Las nuevas tecnologías han conseguido dotar a estos pequeños aparatos de sistemas operativos con un gran número de funcionalidades.
Gracias a ellos, y a través de una variada oferta de aplicaciones, este tipo de dispositivos ha pasado a formar parte del día a día de los usuarios, llegando incluso a desplazar a los ordenadores para según qué hábitos.
Y es que el mercado de las aplicaciones móviles cuenta con una gran ventaja: el acceso a la información en cualquier momento y en cualquier lugar.

Pero no sólo el público general ha sabido apreciar las capacidades de este tipo de dispositivos. En el mundo empresarial las aplicaciones móviles han marcado un antes y un después, tanto en la forma de trabajar, como en la comunicación con los clientes.
Así, los empleados de una misma organización están en continuo contacto, entre ellos y con la empresa, independientemente del lugar en el que se encuentren.
Éste es uno de los motivos fundamentales por el que este proyecto se ha llevado a cabo. El control de ciertos aspectos de una empresa a tiempo real desde cualquier punto geográfico.

El otro es la gestión del consumo energético, un factor indispensable para cualquier negocio. La cantidad de potencia y energía consumidas en cada momento, a lo largo de cada día, o de cada mes, son datos de gran interés a la hora de gestionar los gastos de una empresa. Comprobar si las medidas son las esperadas o si se está haciendo un uso poco eficiente de los recursos, puede ayudar en gran medida a que una compañía sea sostenible, eficiente y rentable.

La plataforma ‘Xener’, dedicada al sector energético y perteneciente a la empresa ITM, disponía de un conjunto de dispositivos con una serie de sensores instalados que medían, en intervalos cortos de tiempo, los valores de ciertas magnitudes relacionadas con el consumo de energía.
A través de estos medidores, colocados en varios locales de la península, y mediante una aplicación web, no demasiado intuitiva, daba servicio a un determinado número de franquicias.
Los miembros de las empresas que habían contratado sus servicios buscaban, de una forma sencilla y constante, el control de la situación energética de sus locales. Es así como surge el objetivo de este proyecto.

Se desea realizar una aplicación móvil, a través de la cual los distintos miembros de una compañía tengan acceso, desde cualquier punto geográfico y en tiempo real, a ciertas

medidas de consumo de sus locales, indispensables para mantener su gasto energético controlado.
Los usuarios recibirán además, en su dispositivo, alarmas en forma de notificación. Estas alarmas, programadas por los mismos usuarios, saltarán cuando los datos, recogidos por los distintos medidores que tengan contratados, se encuentren fuera de los límites deseados.

Los datos son recogidos a través de una API web, ya desarrollada y facilitada por "Xener". Para la programación de alarmas y la administración de usuarios será necesaria, además, una aplicación web sencilla e intuitiva.
Por ello, será imprescindible crear de una base de datos para la capa de persistencia, y la programación de un módulo para la comunicación con las dos aplicaciones.

1.2 Metodologías

A la hora de realizar un proyecto software una de las principales decisiones gira en torno a la metodología de desarrollo.
El ciclo de vida [1] es un factor clave que marcará las etapas y pasos a seguir durante todo el proceso, por lo que elegir uno que se ajuste a las necesidades, tanto del cliente como de los desarrolladores, es fundamental para obtener los resultados deseados.

Es por eso que, para este proyecto, se han tenido en cuenta dos factores esenciales:

- El sistema sería desarrollado por una sola persona, por lo que su proceso de desarrollo debía ser lo más sencillo posible.
- El cliente deseaba participar en este proceso de desarrollo, comprobando que los pasos que se tomaban eran los adecuados y que el software se adaptaba a sus necesidades.

Se optó, por lo tanto, por un modelo iterativo, derivado del ciclo de vida en cascada.
El modelo en cascada, establece un orden riguroso de las etapas de desarrollo, de modo que no se comienza una, hasta que no está completamente terminada la anterior.
Éste cumplía el primer requisito, pues es adecuado para proyectos pequeños y su seguimiento es sencillo.
El principal problema que presentaba era la dificultad para añadir cambios al diseño, pues era necesario volver a realizar todas las fases para contemplar estas modificaciones.

El modelo iterativo, en cambio, cumplía las dos premisas. En la práctica, se adapta mejor que su predecesor a las necesidades que pueden surgir durante un proceso de desarrollo. Rara vez los requisitos son claros y detallados desde el principio y el desarrollador lleva a cabo con exactitud lo que el cliente demanda.

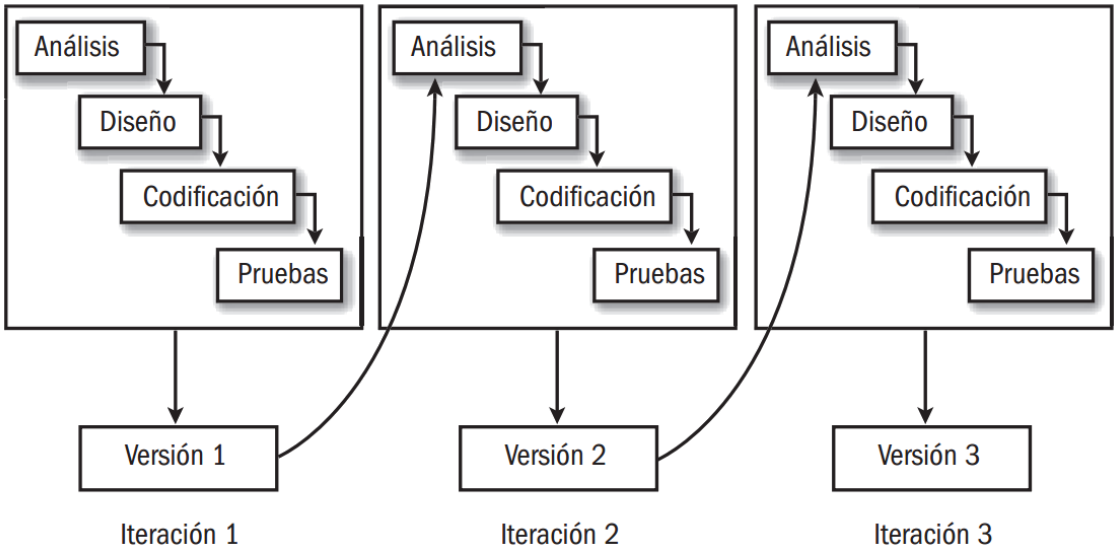

Ilustración 1. Esquema del ciclo de vida iterativo

Basado en la concatenación de varios ciclos en cascada, este modelo permite mostrar al cliente, al final de cada iteración, un producto funcional. De esta forma, si el cliente no ve cubiertas sus expectativas o han surgido nuevos requisitos, se pueden llevar a cabo los cambios y mejoras en la siguiente iteración.

1.3 Contenido de la memoria

Este documento consta de varios apartados, que describen, en su conjunto el proceso de desarrollo de la aplicación. Los diferentes apartados se han distribuido de la siguiente manera:

- En el siguiente capítulo se describen las diferentes herramientas y tecnologías que se han utilizado para la implementación del sistema.
- El capítulo 3 engloba el análisis de requisitos del sistema. Expone los requisitos funcionales y no funcionales, así como los casos de uso, que describirán el funcionamiento de la aplicación.
- El capítulo 4 está dedicado al diseño del software. En él se describen los modelos arquitectónicos que ha adoptado la aplicación, así como el diseño detallado de sus principales componentes.
- En el capítulo 5 se explica el proceso de implementación de la aplicación, detallando las partes más complejas que dan al software la funcionalidad deseada.
- El capítulo 6 recoge una explicación de las pruebas realizadas tras la implementación del sistema.
- Por último, la memoria concluye con un resumen del proyecto, una reflexión sobre su influencia y una mirada hacia futuras mejoras.

2 Tecnologías y herramientas

A continuación, se detallan las herramientas y tecnologías utilizadas para el desarrollo de este proyecto. Todas ellas han sido elegidas buscando adaptarse lo mejor posible a las necesidades de la empresa contratante, intentando conseguir un máximo rendimiento del tiempo y las herramientas disponibles.
Una buena elección será crucial para obtener un buen proceso de desarrollo y un mejor resultado.

2.1 Tecnologías

La elección de unas tecnologías y herramientas adecuadas, no sólo pueden facilitar el proceso de desarrollo del proyecto, ahorrando costes y esfuerzo, también pueden ser de vital importancia a la hora de implementar las funcionalidades deseadas.

Puesto que el proyecto comenzaba de cero, y era necesaria la implementación de las capas de negocio y persistencia, además de las aplicaciones, no había restricciones de compatibilidad preestablecidas a la hora de elegir. Es por eso que las decisiones acerca de qué tecnologías utilizar, se tomaron teniendo en cuenta las necesidades del desarrollador y del cliente, buscando siempre las más actuales y que mejor se adaptasen al producto deseado.

2.1.1 MySQL

MySQL [2] es el sistema gestor de base de datos más utilizado para el desarrollo de aplicaciones web. Orientado al modelo relacional, destaca por su escalabilidad y flexibilidad.

Una de las principales razones por las que se optó por MySQL como SGBD para este proyecto es porque fue diseñado y optimizado para aplicaciones web, además de su compatibilidad con las principales plataformas y un gran número de lenguajes de programación.

2.1.2 NodeJS

NodeJS [3] es una librería y entorno de ejecución de entrada/salida, dirigido por eventos, muy ligero y eficiente, que ejecuta JavaScript en el lado del servidor. Al estar construido sobre el motor V8 de Google, que compila JavaScript en lenguaje máquina, alcanza grandes velocidades de ejecución.

Una de las principales características, que hace de NodeJS un entorno rápido y eficaz, es que sus operaciones de entrada/salida se realizan de forma asíncrona, gestionadas por un *event loop*, al que se envían con un *callback*. Así cuando se produce una operación de este tipo, el programa puede continuar ejecutándose sin esperar a que termine.

El principal motivo por el que se ha usado NodeJS para la implementación del servidor, ha sido el de implementar toda la aplicación con JavaScript, consiguiendo una aplicación sencilla y eficiente.
Además, NodeJS ofrece una gran cantidad de frameworks y librerías para hacer más ligero el proceso de desarrollo.

2.1.3 Sails

SailsJS [4] es un framework de desarrollo de NodeJS orientado a la implementación de aplicaciones web con patrón de diseño MVC. Basado en el concepto CoC (Convention over Configuration), su objetivo es el de hacer el desarrollo de aplicaciones más sencillo y rápido, ofreciendo un entorno simple para la implementación de su 'back-end' y liberando al desarrollador de tareas de configuración.
Es compatible con otros frameworks orientados al 'front-end' como son Angular o JQuery y se adapta a cualquier sistema gestor de base de datos.

La plataforma estructura las diferentes partes de una aplicación en varios directorios:

- api : Sus contenidos, divididos en algunos subdirectorios, componen el modelo de la aplicación.
- assets : Almacena la parte del cliente: vistas, scripts… de forma que compone la vista y el controlador de la aplicación.
- config: Contiene todos los archivos y carpetas necesarios para dar forma a la aplicación, establecer conexiones, entornos…

La aplicación web ha sido realizada íntegramente con Sails, cuyo modelo ha sido utilizado también para la aplicación móvil. La principal razón es que ofrece un entorno capaz de integrar el 'front-end' con el 'back-end', de forma sencilla y eficiente. La capa de presentación ha sido desarrollada con HTML5, CSS y AngularJS, mientras que la de negociación se ha implementado con NodeJS.

2.1.4 Apache Cordova

Cordova [5] es un framework para el desarrollo de aplicaciones móviles multiplataforma. Permite la construcción de aplicaciones con tecnologías de desarrollo estándar como HTML, CSS y JavaScript, evitando los lenguajes de programación nativos de cada plataforma.

Aunque el proyecto ha sido desarrollado para móviles con sistema operativo Android, no se descarta la ampliación a diferentes plataformas en un futuro. Por eso, por su sencillez al permitir trabajar con las principales tecnologías de diseño de aplicaciones y por su adaptabilidad, se ha elegido esta herramienta para su implementación.

Además, Cordova cuenta con una variada oferta de plugins, desarrollados por la plataforma, o por usuarios que desean compartir sus creaciones.
Algunos de ellos han sido usados para implementar ciertas funcionalidades de la aplicación móvil:

- Dialogs: Ofrece acceso a elementos de dialogo nativos del dispositivo, como son las alertas o notificaciones.
- SplashScreen: Muestra la pantalla deseada al inicio de la aplicación, en el caso de la que nos ocupa, un logotipo de la plataforma Xener.
- PushPlugin: Permite la gestión y recepción de notificaciones.

2.1.5 AngularJS

AngularJS [6] es un framework de JavaScript, diseñado especialmente para trabajar con vistas dinámicas. Una de sus principales diferencias respecto a otras herramientas que trabajan con HTML, es el enlazado de datos o 'data binding'. AngularJS ofrece una forma muy sencilla de actualizar la vista con los cambios del modelo y viceversa, a través de controladores enlazados con cada una de las páginas, dejando atrás la manipulación del DOM.

En este proyecto ha sido utilizado AngularJS, junto con HTML5 y CSS3 para la implementación de las dos aplicaciones, móvil y web. Una de las principales razones por las que se ha optado por esta tecnología es su sencillez a la hora de manipular las vistas.

2.1.6 Onsen UI

Onsen UI [7] es un framework de HTML5 para la creación de interfaces de usuario en aplicaciones híbridas. Facilita una serie de componentes que dan a la aplicación el aspecto de una nativa y se complementa a la perfección con AngularJS.

Las interfaces gráficas de la aplicación móvil han sido implementadas con Onsen. Esto ha facilitado el diseño en gran medida, gracias a sus plantillas y variedad de componentes.

2.2 Herramientas

Existen multitud de programas y herramientas, de los cuales los desarrolladores se nutren, que sirven de ayuda en todas las fases del desarrollo de un sistema software.

Durante la creación de este proyecto se han utilizado varias herramientas de edición, programación, depuración y modelado, que han agilizado el proceso en gran medida.
A continuación se expone un breve resumen de todas ellas.

2.2.1 Eclipse

Eclipse [8] es un programa de código abierto que ofrece un conjunto de herramientas de programación multiplataforma y que soporta múltiples lenguajes.

En este proyecto, Eclipse ha sido utilizado para la implementación y depuración de la aplicación móvil. Se ha elegido Eclipse porque ofrecía un entorno conocido, ya que se había utilizado durante varias asignaturas en la carrera, y eficiente.
Además, era posible integrar, sin demasiada dificultad, el conjunto de herramientas de desarrollo para Android, del que se hablará a continuación.

2.2.2 Android SDK

Android SDK (Software Development kit) [9] es un conjunto de herramientas de desarrollo y depuración que permite la implementación de aplicaciones Android.

En este proyecto se ha vinculado al programa eclipse para llevar a cabo la implementación de la aplicación móvil.

2.2.3 MySQL Workbench

MySQL Workbench [2] es una herramienta visual unificada, para arquitectos de bases de datos y desarrolladores, que permite el modelado de datos y el desarrollo en SQL y ofrece herramientas de administración para la configuración de servidores y la gestión de usuarios y de backups entre muchas otras cosas. Está disponible para los principales sistemas operativos: Windows, Linux y Mac OS X.

En el desarrollo de este proyecto se ha utilizado MySQL Workbench para la implementación de la base de datos.

2.2.4 Sublime Text

Sublime [10] es un editor de texto ideal para escribir código, pues ofrece múltiples funcionalidades de edición de código y soporta una gran cantidad de lenguajes de programación. Aunque no es un programa de código abierto, tiene una versión gratuita.

En este proyecto se ha utilizado para la implementación del código de la aplicación web y el servidor porque, además de ser una completa herramienta de edición, permite la importación de proyectos enteros, respetando la estructura de sus directorios, algo muy útil a la hora de trabajar con Sails.

2.2.5 MagicDraw UML

MagicDraw UML [11] es una extendida herramienta de modelado de software, muy intuitiva, que facilita el análisis y diseño de sistemas y bases de datos orientados a objetos.
Durante el desarrollo de este proyecto, se ha utilizado MagicDraw para la elaboración de los diagramas presentes en esta memoria, como son los casos de uso y el diseño de la arquitectura.

3 Análisis del sistema

El análisis de un sistema software [12] es un paso crucial para su desarrollo. En él se especifican los servicios que debe ofrecer el sistema, así como sus restricciones; se reflejan las necesidades del cliente y se marca un camino para el desarrollo.

A continuación se muestra el análisis de requisitos de este proyecto. Éste se ha ido modelando a lo largo de su desarrollo, añadiendo los cambios necesarios tras cada reunión con el cliente, de forma que el producto final cumpliese sus expectativas.

3.1 Especificación de casos de uso

Los diagramas de casos de uso son una forma de modelar las interacciones del software con los actores que en ellas intervienen. Para este proyecto se han realizado tres diagramas casos de uso que escenifican diferentes aspectos del sistema y uno que determina el tipo de usuarios que intervendrán en él.

En primer lugar se expondrá el diagrama con la jerarquía de usuarios (ver ilustración 2), un aspecto clave para la aplicación.
El cliente deseaba que el sistema distinguiese distintos tipos de usuarios, cada uno con unos permisos que, por defecto, pertenecerían a su perfil, pero que podrían ser cambiados por un usuario situado en lo más alto de la pirámide jerárquica.

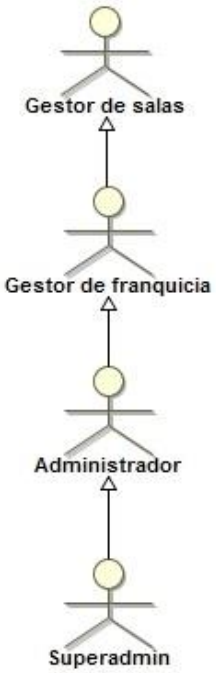

Ilustración 2. Diagrama de usuarios de la aplicación

Los tipos de usuario que podrán acceder a la aplicación serán los siguientes:

- Gestor de salas: Podrá acceder a la información perteneciente a los locales que se le han asignado, así como crear y recibir alarmas sobre sus medidores.
  Éste perfil está diseñado para gerentes y empleados de las franquicias que contraten la aplicación.

- Gestor de franquicia: Podrá acceder a toda la información perteneciente a la franquicia que se le ha asignado, así como crear y recibir alarmas sobre todos los locales pertenecientes a esta. También podrá crear y gestionar usuarios con perfil Gestor de sala, relacionados con su franquicia asignada.
  Este tipo de usuario está diseñado para los dueños de las franquicias que contraten la aplicación.

- Administrador: Tendrá acceso a todos los locales y todas las franquicias. Podrá consultar su información y crear alarmas sobre ellos. Además podrá crear y gestionar usuarios con un perfil inferior al suyo.
  Éste perfil está diseñado para los encargados de la plataforma 'Xener'.

- Superadmin: Tendrá acceso a toda la información perteneciente a la aplicación: alarmas, franquicias, usuarios… y podrá gestionarla con total libertad. Además podrá controlar los permisos que se asignan por defecto a cada tipo de usuario y eliminar usuarios (el resto de perfiles solo podrán desactivarlos).
  Éste perfil está diseñado para los desarrolladores de la aplicación.

Una vez establecidos los tipos de usuario, se muestran los diagramas de casos de uso, que muestran las interacciones de estos con la aplicación.

El primer diagrama (ver ilustración 3) es el de gestión de locales, que recoge todas las acciones disponibles para el usuario en relación al control de los locales a los que tiene acceso.

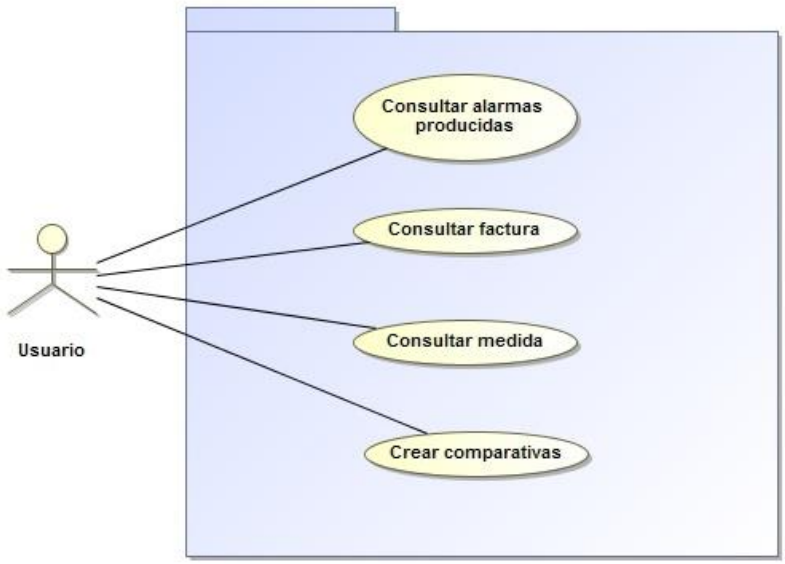

Ilustración 3. Diagrama de gestión de locales

En el apartado de gestión de alarmas (ver ilustración 4), las acciones son comunes a todos los usuarios, independientemente de su perfil, por lo que los actores que intervienen en ambas se han unificado bajo el nombre de 'Usuario'.

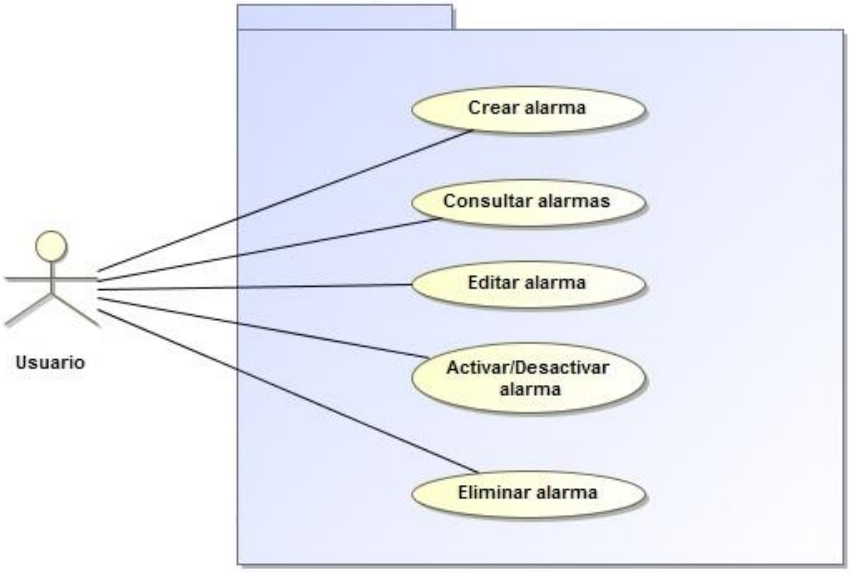

Ilustración 4. Diagrama de gestión de alarmas

Por último, se muestran los escenarios de gestión de usuarios (ver ilustración 5). Estas acciones están reservadas a ciertos perfiles de usuario, por lo que se especifica qué actores pueden desempeñar cada función.

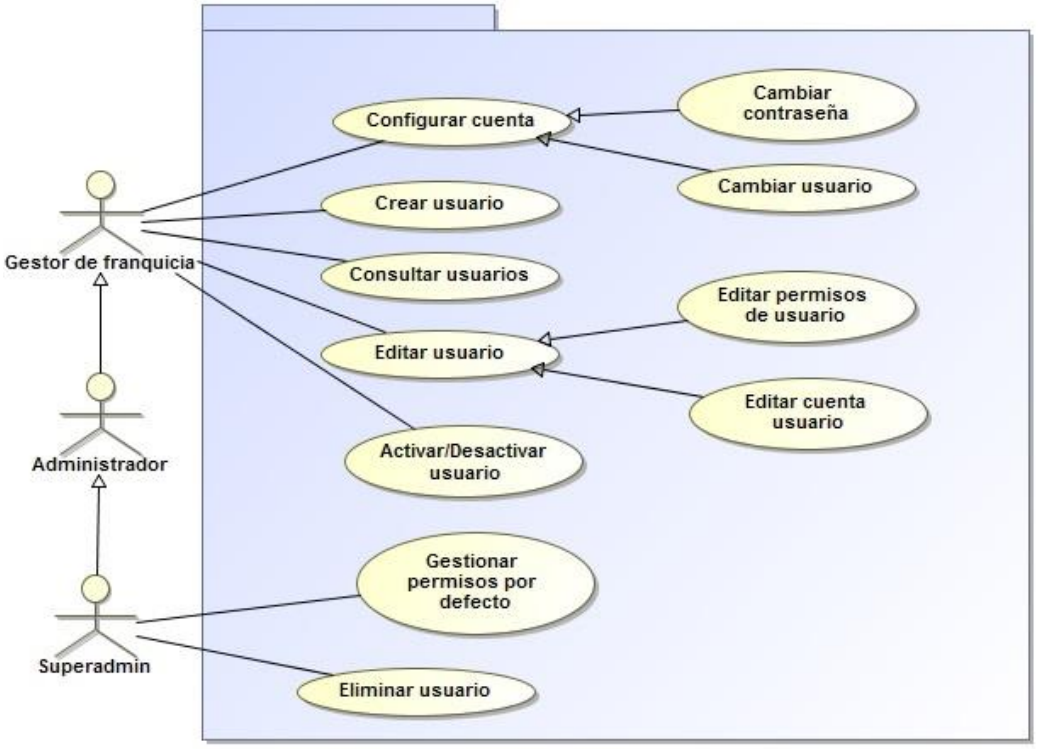

Ilustración 5. Diagrama de gestión de usuarios

Como puede observarse, hay dos partes bien diferenciadas en los casos de uso. Una dedicada a la consulta de datos, y otra a la gestión de algunos aspectos del sistema.
La implementación de todas estas funcionalidades dotarían de demasiado contenido a una aplicación móvil, además de hacerla menos intuitiva y sencilla de manejar.
Es por eso que se decide trasladar la parte de gestión a una aplicación web, que funcione como gestor de algunos parámetros de la aplicación, como son los usuarios y las alarmas.

A continuación, en las tablas 1 y 2 se procede a la descripción detallada de algunos casos de uso que darán una idea más aproximada del funcionamiento de la aplicación.

Identificador	ID1
Nombre	Consultar medidas
Descripción	El usuario accede a la aplicación para consultar datos de alguna magnitud
Precondición	El usuario está registrado en la aplicación
Secuencia normal	1. El usuario accede a la aplicación e introduce su nombre de usuario y contraseña. 2. La aplicación envía los datos al servidor para ser comprobados. 3. El servidor comprueba que los datos son correctos y devuelve una lista con las franquicias a las que el usuario tiene acceso. 4. El usuario selecciona una franquicia de la lista de franquicias disponibles. 5. La aplicación solicita al servidor los locales pertenecientes a la franquicia seleccionada, y muestra una lista con los datos recibidos. 6. El usuario selecciona una sala de la lista de locales disponibles. 7. La aplicación solicita al servidor los puntos de medición pertenecientes al local seleccionado, y muestra una lista con los datos recibidos. 8. El usuario selecciona el punto de medición del que desea comprobar los datos. 9. La aplicación solicita al servidor las magnitudes disponibles para el punto de medición seleccionado y muestra una lista con los datos recibidos. 10. El usuario selecciona la magnitud deseada. 11. La aplicación recoge de la API web la información referente a la magnitud seleccionada, del punto de medición al que pertenece y lo muestra.
Secuencia alternativa	4.1 Si el perfil de usuario es el de “Gestor de franquicia”, pasa directamente al paso 5. 6.1 Si el usuario sólo tiene acceso a una sala, pasa directamente al paso 7.

Tabla 1. Especificación de caso de uso 'Consultar medidas

Identificador	ID2
Nombre	Crear usuario
Descripción	El usuario crea un nuevo usuario.
Precondición	El usuario está logueado y cuenta con permiso para crear usuarios.
Secuencia normal	1. El usuario logueado accede a la opción ‘Crear usuario’. 2. La aplicación solicita al servidor los tipos de usuario con rango inferior al usuario logueado y los muestra. 3. El usuario logueado rellena los datos del nuevo usuario, despliega la lista con los tipos de usuario disponibles y selecciona un tipo de usuario. 4. La aplicación solicita al servidor las franquicias disponibles y muestra una lista con los resultados. 5. El usuario logueado asigna una franquicia al nuevo usuario. 6. La aplicación solicita al servidor las salas pertenecientes a la franquicia seleccionada muestra una lista con los resultados. 7. El usuario logueado selecciona las salas a las que tendrá acceso el nuevo usuario y pulsa la opción de “Guardar”. 8. La aplicación envía al servidor los datos del nuevo usuario. 9. El servidor registra en la base de datos el nuevo usuario y le asigna los permisos que por defecto corresponden a su tipo de usuario.
Secuencia alternativa	4.1 Si el perfil de nuevo usuario es el de ‘Administrador’, pasa directamente al paso 8. 6.1 Si el perfil de nuevo usuario es el de ‘Gestor de franquicias’, pasa directamente al paso 8.

Tabla 2. Especificación de caso de uso 'Crear usuario'

3.2 Especificación de requisitos funcionales

Los requisitos funcionales son aquellos que describen los servicios que el sistema debe ofrecer.
Durante el desarrollo del proyecto han ido surgiendo nuevas ideas y cambios por parte del cliente que se han ido incorporando, todas ellas están finalmente reflejadas en la tabla 3.

Identificador	Descripción
RF01	Los usuarios deberán loguearse, indicando su nombre de usuario y contraseña, para acceder a la aplicación.
RF02	Los usuarios podrán configurar los parámetros de su cuenta.
RF03	Los usuarios podrán crear usuarios con perfiles de rango inferior.
RF04	Los usuarios podrán consultar los usuarios registrados con perfiles de rango inferior.
RF05	Los usuarios podrán editar las cuentas de los usuarios con perfiles de rango inferior.
RF06	Los usuarios podrán editar los permisos de los usuarios con perfiles de rango inferior.
RF07	Los usuarios podrán activar o desactivar las cuentas de los usuarios con perfiles de rango inferior.
RF08	El usuario con perfil 'Superadmin' podrá eliminar usuarios.
RF09	El usuario con perfil 'Superadmin' podrá gestionar los permisos por defecto de cada tipo de usuario.
RF10	Los usuarios podrán crear nuevas alarmas, estableciendo diferentes parámetros en cada una de ellas, sobre locales o franquicias a los que tengan acceso.
RF11	Los usuarios podrán consultar las alarmas que han creado.
RF12	Los usuarios podrán editar las alarmas que han creado.
RF13	Los usuarios podrán activar o desactivar las alarmas que han creado.
RF14	Los usuarios podrán eliminar las alarmas que han creado de forma definitiva.
RF16	Los usuarios recibirán una notificación en su dispositivo cuando una de sus alarmas haya saltado.
RF17	Los usuarios podrán consultar las alarmas producidas en cada sala.
RF18	Los usuarios podrán realizar gráficas comparativas de los distintos puntos de medición de cada sala.
RF19	Los usuarios podrán obtener información sobre la temperatura de cada medidor perteneciente a una de las salas a las que tengan acceso. Esta consistirá en dos datos (temperatura del local y temperatura exterior) tomados en el momento de la consulta, y una gráfica con las temperaturas tomadas a lo largo del día.
RF20	Los usuarios podrán obtener información sobre distintas medidas de consumo (energía, potencia, voltaje, intensidad) de cada medidor

	perteneciente a una de las salas a las que tengan acceso. Esta consistirá en los datos tomados en el momento de la consulta, y una gráfica con los datos tomados a lo largo del día.
RF21	Los usuarios podrán consultar una aproximación de la factura eléctrica de cada sala a la que tengan acceso, calculada en función a las medidas de consumo tomadas a lo largo del mes.

Tabla 3. Especificación de requisitos funcionales

3.2 Especificación de requisitos no funcionales

Los requisitos no funcionales establecen ciertas propiedades que debe tener el sistema para cumplir unos objetivos de funcionamiento. Éstos pueden hacer referencia factores como la seguridad, el rendimiento o la compatibilidad.

A continuación se muestra la tabla 4 con los requisitos no funcionales que se han considerado para el proyecto que nos ocupa.

Identificador	Descripción	Categoría
RNF01	El sistema deberá distinguir los usuarios logueados, permitiendo su acceso a unas u otras funcionalidades, según su tipo.	Seguridad
RNF02	El sistema deberá cifrar las contraseñas, para garantizar la protección de las cuentas de los usuarios.	Seguridad
RNF03	El sistema deberá llevar un registro 'log' de todas las acciones realizadas sobre la base de datos.	Seguridad
RNF04	La aplicación móvil deberá ser compatible con todos los dispositivos con sistema operativo Android.	Compatibilidad
RNF05	La aplicación móvil deberá estar abierta a cambios para ser introducida a nuevos sistemas operativos.	Compatibilidad Escalabilidad
RNF06	Las aplicaciones deben tener un diseño 'responsive' para su visualización en cualquier tipo de pantalla.	Compatibilidad

Tabla 4. Especificación de requisitos no funcionales

4 Diseño del sistema

El diseño de un sistema software es la fase previa a su implementación. En esta etapa se tomarán decisiones necesarias para que el sistema cumpla todos los requisitos especificados en la fase de análisis.

En este documento se tratarán dos clases de diseño: el diseño arquitectónico, que expondrá el modelo de arquitectura software que se ha adoptado para llevar a cabo la aplicación; y el diseño detallado, a través del cual se explicarán los elementos base para su posterior codificación.

4.1 Diseño arquitectónico

El diseño arquitectónico es el primer paso en la etapa de diseño de un sistema. Su función es la de establecer la estructura del software, identificando sus diferentes módulos y la forma en que se comunicarán.
A la hora de diseñar la arquitectura de un sistema software, el desarrollador no parte de cero, existen unos modelos o patrones de diseño en los que se basan la mayoría de sistemas.

Para la realización de este proyecto se han combinado dos de los principales patrones arquitectónicos: la arquitectura en tres capas y el Modelo-Vista-Controlador.

La arquitectura en tres capas (ver ilustración 6) divide un sistema software en tres secciones independientes y con unas funciones específicas:

- La capa de presentación, encargada de la interacción con el usuario, estaría formada por las dos aplicaciones, móvil y web.

- La capa de negocio, que gestiona la actividad del sistema, dotándolo de las funcionalidades necesarias y controlando el acceso a los datos, estaría recogida en un servidor de negociación implementado en la plataforma Sails.

- Por último, la capa de persistencia o de datos, encargada del almacenamiento y manipulación de los datos, queda recogida en la base de datos MySQL.

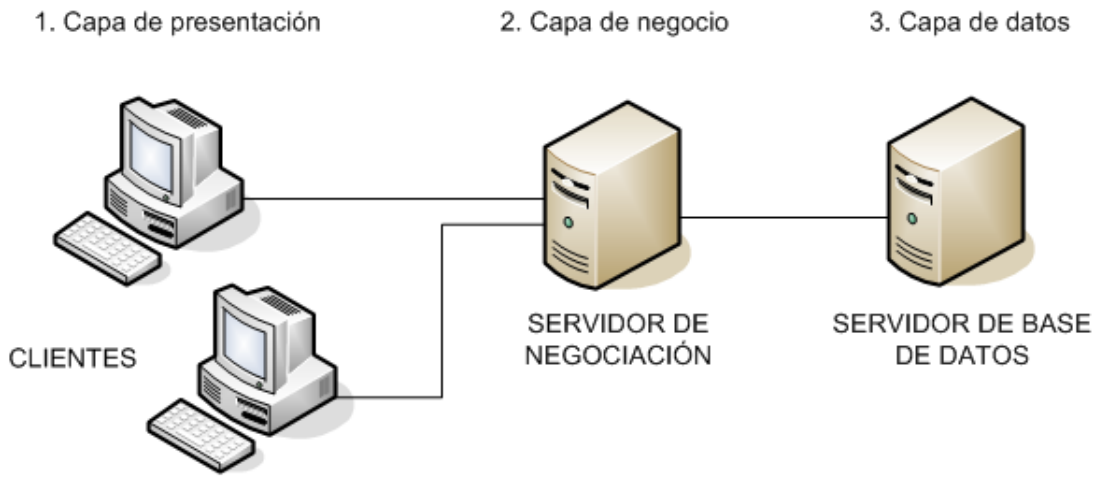

Ilustración 6. Esquema de arquitectura en tres capas

Por su parte, la comunicación mediante las capas de presentación y negocio, se ha implementado siguiendo un patrón Modelo-Vista-Controlador, que divide este fragmento del sistema, a su vez, en otras tres capas:

- Modelo: Es la capa encargada del acceso a los datos, así como de su control y manipulación. Se comunica directamente con la capa de persistencia.

- Vista: Es la capa encargada de la presentación de los datos y las interfaces de usuario. La parte visual que se encarga de la representación de la información.

- Controlador: Gestiona las peticiones de los usuarios, encargándose de la comunicación con el modelo. Se encarga de la actualización de las vistas.

Para la implementación de la capa de negocio, gestionada por un servidor, se ha utilizado Sails un framework que ofrece un patrón Modelo-Vista-Controlador.
El modelo y el controlador implementados con Sails, serán utilizados por ambas aplicaciones, de modo que el acceso a los datos quedaría cubierto con esta parte.
Además este patrón es el más flexible en cuanto a los cambios en los datos o en su forma de representarlos, pues se puede modificar un factor, sin que esto influya en el otro. Esta faceta es una gran ventaja a la hora de aumentar las funcionalidades de la aplicación o añadir nuevas entidades en los datos.
Para la capa de presentación, ambas aplicaciones implementarán sus vistas y controladores, encargados de la gestión de eventos producidos por el usuario y de la comunicación con el controlador del servidor, completando así el patrón Modelo-Vista-Controlador.
A través de la ilustración 7, se procederá a explicar cómo funcionará el sistema a través del mismo.

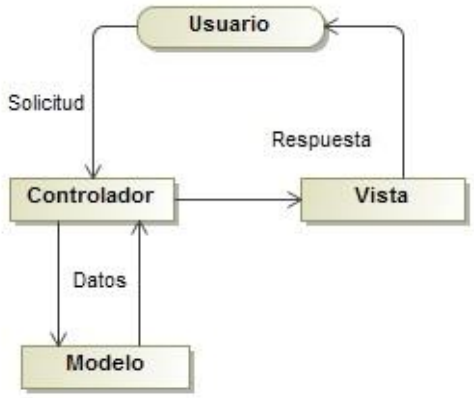

Ilustración 7. Arquitectura MVC

El usuario desea realizar una acción a través de la interfaz de una de sus aplicaciones (móvil o web). Se envía así una petición al controlador de la aplicación, que se dispone a dar respuesta al evento.
Si la acción solicitada, requiere acceso a los datos, a través de una petición HTTP, el controlador de la aplicación se comunica con el del servidor, que a su vez solicita los cambios necesarios al modelo.
El modelo hace los cambios y consultas necesarios en la base de datos, generando otros datos como respuesta que enviará de vuelta al controlador.
Cuando el controlador del servidor recibe la respuesta, se la envía a través de HTTP al controlador de la aplicación, que actualiza la vista con los datos necesarios, que serán mostrados al usuario a través de la interfaz de una de las aplicaciones.

Es posible, en este proyecto, que algunas de las peticiones realizadas al controlador de la aplicación, vayan dirigidas a la API web generada por la plataforma 'Xener'. En ese caso, en lugar de comunicarse con el servidor, el controlador enviará una solicitud HTTP a la API, que le devolverá los datos en formato de objeto JSON.

4.2 Diseño detallado

Una vez adoptado el patrón de arquitectura adecuado, se procede a la realización del diseño detallado. En este paso se toman las decisiones necesarias para que el proyecto cumpla los requisitos establecidos, adaptándose al patrón seleccionado.

Como se ha expuesto en el apartado anterior, el sistema software diseñado para este proyecto presentará una arquitectura en tres capas, desarrollando dos de ellas bajo el patrón Modelo-Vista-Controlador.
A continuación se muestra un diagrama de despliegue del sistema (ver ilustración 8), con los distintos módulos, que darán forma al software, ya definidos.

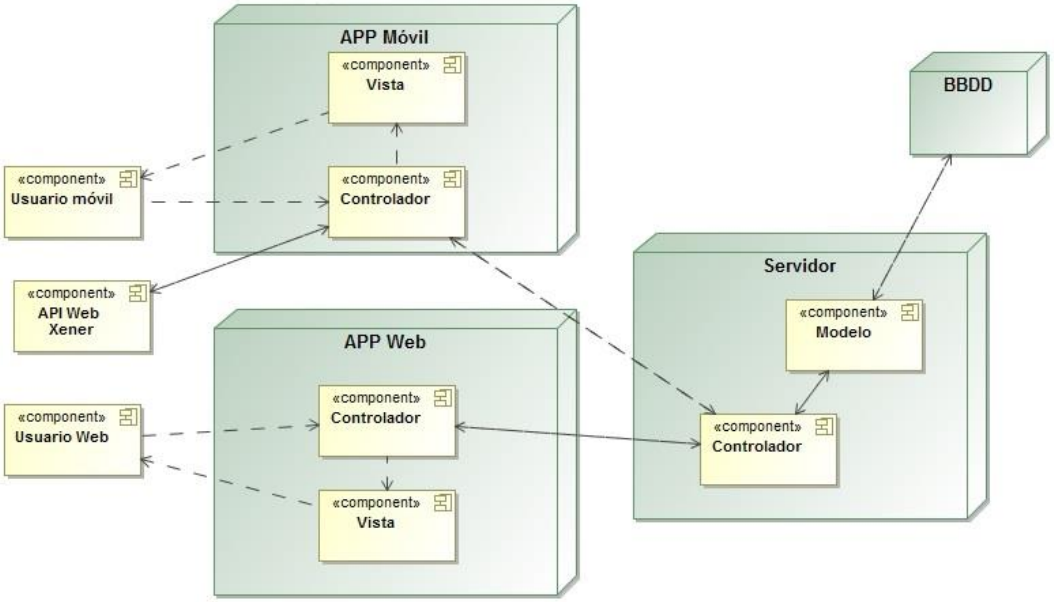

Ilustración 8. Diagrama de despliegue del sistema

En el diagrama de despliegue se aprecian los módulos pertenecientes a las dos aplicaciones, que forman parte de la capa de presentación. Una construida sobre Cordova (aplicación móvil) y otra sobre Sails (aplicación web).
Sails proporciona un entorno para desarrollar una aplicación MVC completa, por lo que se aprovechó la construcción del servidor, que constituye la capa de negocio, en este framework, para implementar también la aplicación web.
La capa de persistencia, está formada por la base de datos, construida sobre el sistema gestor de MySQL.

Cada aplicación implementa sus propias vistas y controladores, que se comunican con el controlador del servidor.
Por otra parte, el controlador de la aplicación móvil puede acceder también a la API web que proporciona los datos tomados por los medidores en tiempo real. Las llamadas a esta se realizarán mediante el protocolo HTTP que facilita AngularJS, ya que será la tecnología utilizada para la implementación de ambos controladores, y los parámetros de la llamada irán en la propia URL. Este proceso se explicará en profundidad en el apartado de implementación.

Para profundizar más en el diseño de cada módulo, se mostrarán a continuación dos diagramas de componentes (ver ilustraciones 9 y 10), uno perteneciente a cada aplicación, en los que se detallarán las distintas vistas y los controladores a los que están asociadas, diferenciando así las tres capas del patrón MVC.

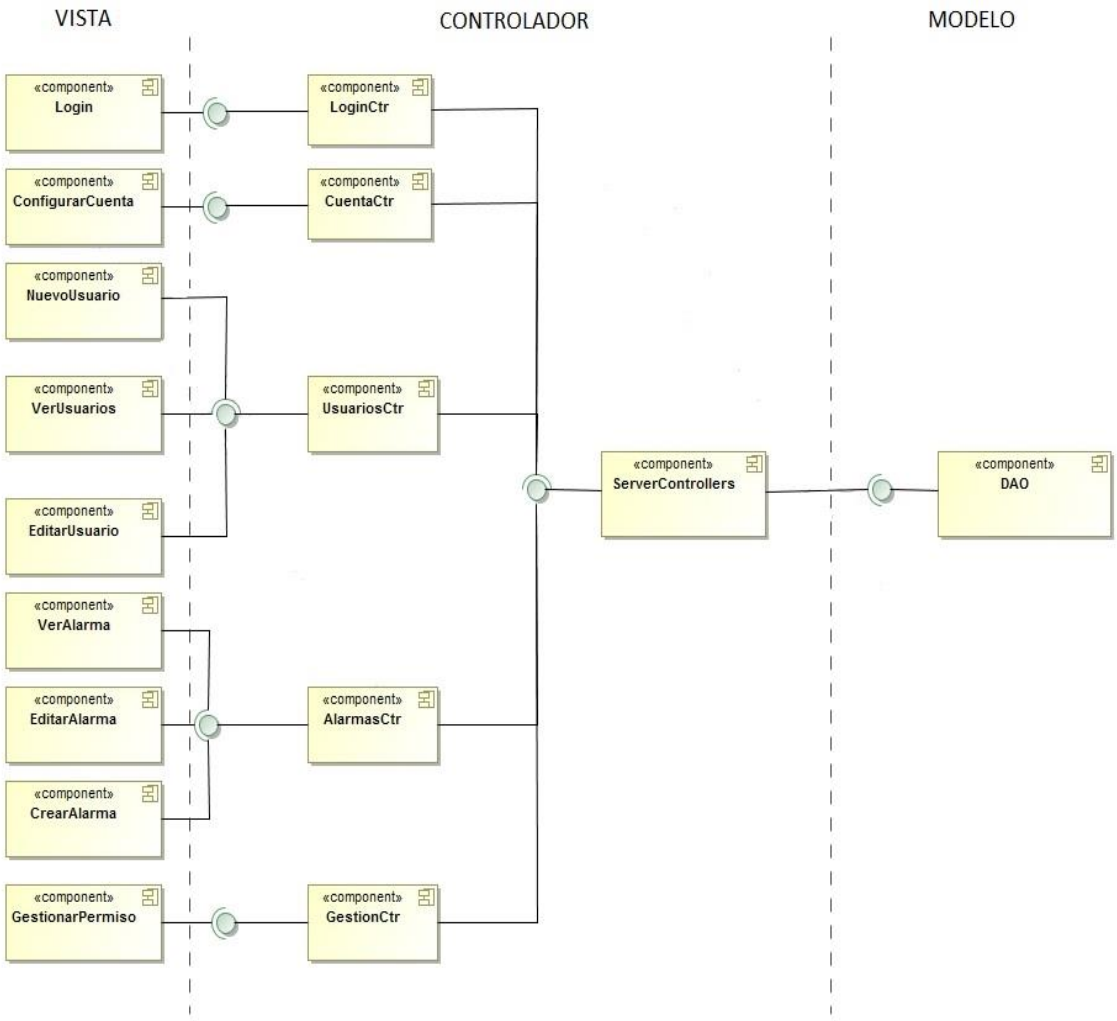

Ilustración 9. Diagrama de componentes de la aplicación web

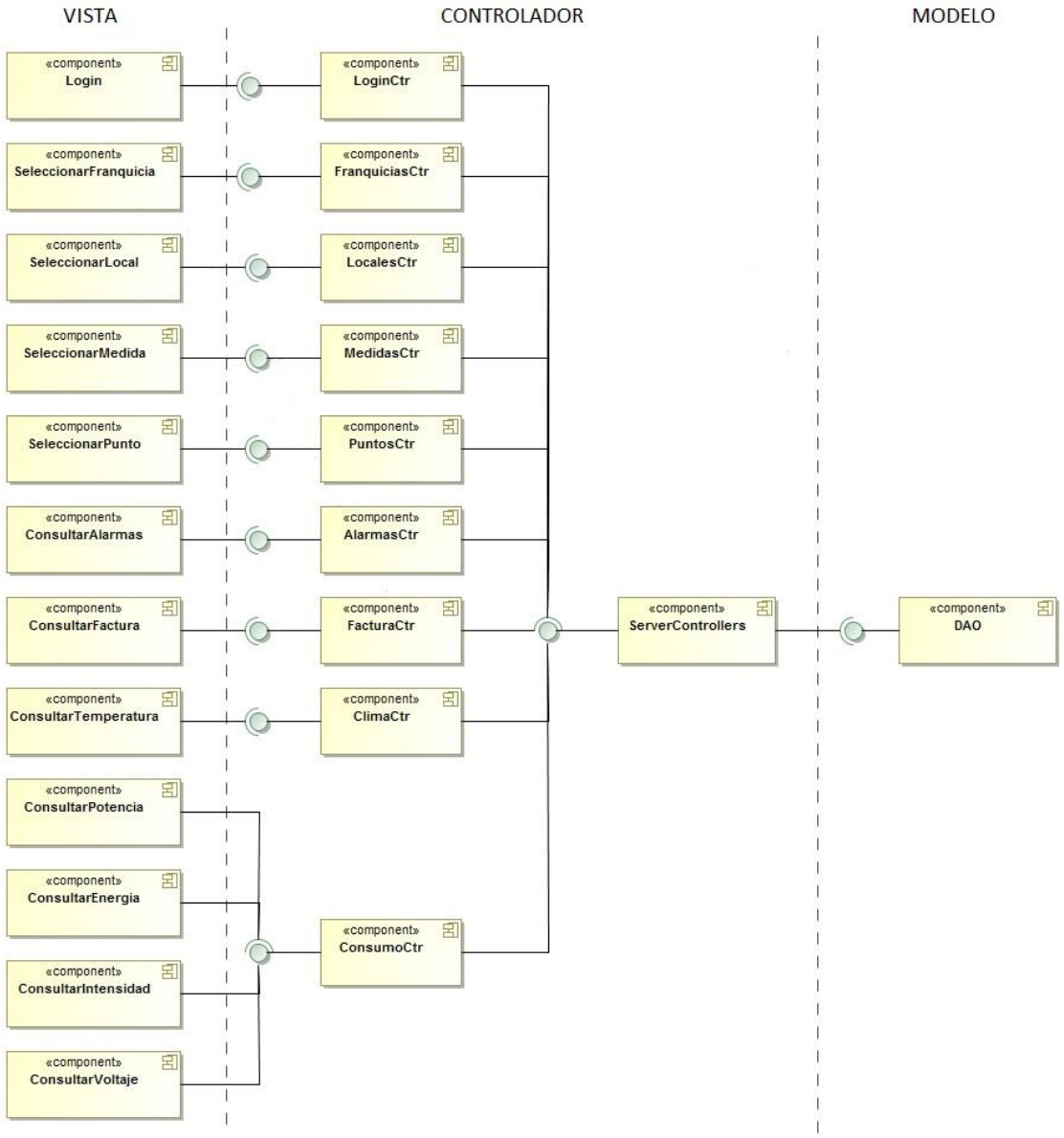

Ilustración 10. Diagrama de componentes de la aplicación móvil

La capa de modelo y los controladores del servidor no se han detallado por su gran número de elementos, pues cada relación de la base de datos, tiene su representación en el modelo, con un controlador asociado para el acceso a los datos.
Cada componente de la capa del controlador residente en las aplicaciones, hace peticiones a los distintos controladores residentes en el servidor, a través de una URL recogida en la hoja de rutas de Sails, que se detallará en el apartado de implementación.

Por último, se muestra el diseño de la base de datos sobre la que se construye la aplicación (ver ilustración 11). Las tablas y relaciones proporcionan una idea de los modelos y controladores que se implementarán en el servidor.

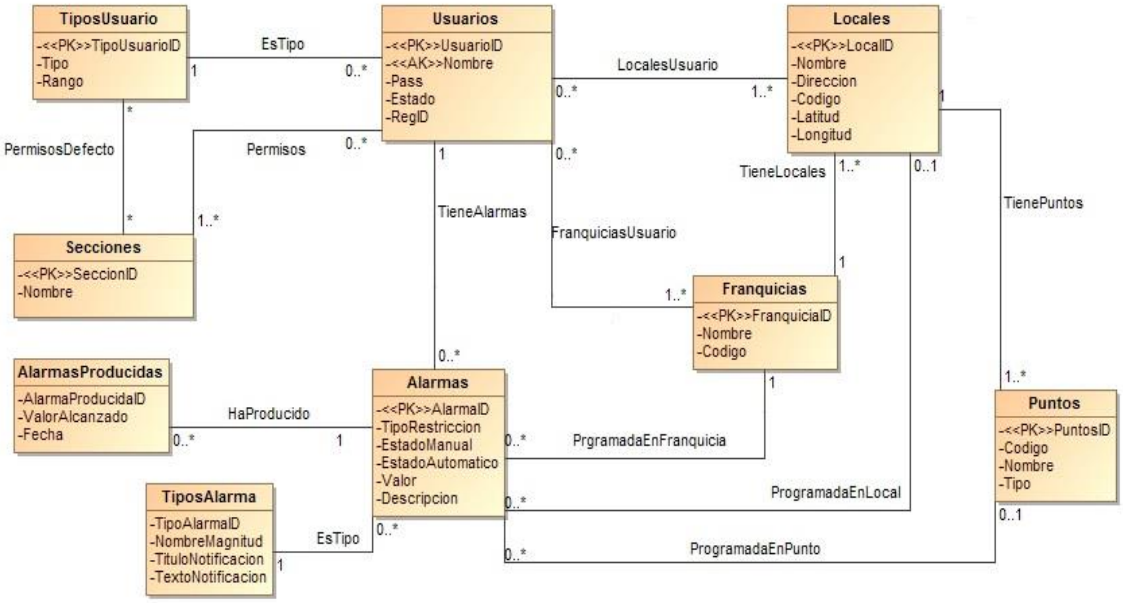

Ilustración 11. Diseño UML de la base de datos

5 Implementación del sistema

Una vez analizados los requisitos del sistema, y tomadas las decisiones más importantes sobre su arquitectura y estructura, es hora de describir su implementación.

Puesto que éste sistema estaba diseñado para ser desarrollado por una sola persona, la implementación se llevó a cabo en varias partes, dividiéndose en subsistemas:

- En primer lugar se creó la base de datos, trasladando al gestor el diseño UML que se había realizado.

- Una vez creada la base de datos, se comenzó con la implementación del servidor en Sails. Éste trabajaba sobre la base de datos y daría servicio a las aplicaciones web y móvil.

- A continuación, se comenzó la codificación de la aplicación web, modificando los controladores del servidor a medida que surgían nuevas necesidades.

- Cuando se contaba ya con una aplicación web más o menos funcional, se implementó la aplicación móvil. El orden establecido fue ese debido a que era en el gestor web donde se creaban algunos de los contenidos de la aplicación móvil como las alarmas o los usuarios.

- Por último, se implementaron una serie de rutinas para volcar datos de la API web en la base de datos, y se puso en funcionamiento el envío de alarmas, cuya implementación se detallará en los siguientes apartados.

5.1 Capa de persistencia

La capa de persistencia ha sido implementada con el SGBD de MySQL. Su función no se reduce solamente a la representación de los datos. La base de datos diseñada se encarga también, en gran medida, de su gestión y control, reduciendo así la carga del servidor.
Para ello se han diseñado una serie de disparadores que cumplen distintas funciones:

- Asignar a un usuario los permisos por defecto, correspondientes a su tipo, al ser creado.
- Controlar que un gestor de franquicia pueda tener sólo una franquicia asignada.
- Asignar a los usuarios del tipo ‘Gestor de franquicia’, ‘Administrador’ y ‘Superadmin’, todos los locales pertenecientes a las franquicias que controlan, al insertar o actualizar las franquicias y los locales.
- No permitir a un usuario la creación de dos alarmas con los mismos parámetros, evitando así la redundancia de datos.

Además, la base de datos implementa un registro de las acciones realizadas sobre los datos, cumpliendo así el requisito no funcional número tres: “El sistema deberá llevar un registro ‘log’ de todas las acciones realizadas sobre la base de datos”.
Para ello cuenta con una tabla LogWeb, en la cual almacena el usuario que realiza la acción, la fecha, una breve descripción de los cambios realizados, y si la operación ha sido errónea o se ha llevado a cabo.

5.2 Capa de negocio

Como se ha expuesto en los anteriores apartados, el servidor de este sistema se ha implementado a través de la plataforma Sails.
Sails facilita esta tarea en gran medida, ya que ofrece recursos para manejar de forma sencilla la configuración del sistema y es posible crear la estructura de las capas de modelo y controlador a través de comandos en la consola.

Estas dos capas están compuestas por varias entidades que toman su nombre. Para este proyecto se ha creado un controlador por cada modelo, de forma que cada controlador implemente las principales operaciones que se realizarán sobre la tabla a la que representa el modelo.
Sails facilita esta opción a través del comando “sails generate api <<nombre>>”, que se ejecutará una vez por cada modelo creado.
El servidor del software aquí descrito, presenta la siguiente estructura (ver ilustración 12).

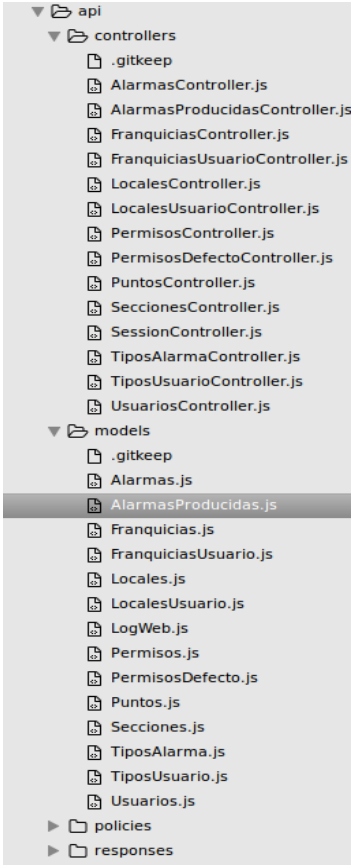

Ilustración 12. Estructura de clases del servidor

Como se puede observar, existe, en efecto, un controlador por cada modelo, a excepción del equivalente a la tabla que recoge el Log de la aplicación, pues las operaciones de inserción de este tipo de datos son ejecutadas en los demás controladores, a medida que se modifican los datos de los que se quiere guardar registro.
Hay también un controlador que no equivale a ningún modelo, SessionController. Este es el encargado de trabajar con los datos que dependen de la sesión iniciada en la aplicación, es decir, los datos que varían de un usuario a otro.

Cada modelo, representa una colección estructurada de datos, el equivalente, en MySQL, a una tabla.
En él se han de representar los diferentes atributos, relaciones y restricciones reflejados en la tabla, de la forma más fiel posible, para mantener la coherencia entre el modelo y la base de datos.

A continuación se muestra la estructura que presentará un modelo (ver ilustración 13). En él se indica la tabla que representa y los diferentes atributos con sus tipos y restricciones.

```
/**
* AlarmasProducidas.js
*/

module.exports = {

  tableName: 'alarmasproducidas',
  migrate: 'safe',
  autoCreatedAt: false,
  autoUpdatedAt: false,

  attributes: {
    alarmaProducidaID :{
      type : 'INTEGER',
      primaryKey : true,
    },
    fecha: {
      type: 'string',
      required: true,
    },
    alarma: {
      model: 'Alarmas'
    },
  }//fin  atributes
};
```

Ilustración 13. Modelo de la tabla alarmas producidas

Sails ofrece también una serie de operaciones básicas (*create*, *destroy*, *update*, *find*) sobre sus modelos, mapeadas con operaciones de la base de datos a la que representan sus modelos.
Esto facilita en gran medida la manipulación de los datos, y hace más sencillos los controladores.

En Sails, los controladores, están formados por un conjunto de acciones o métodos, que reciben como parámetros objetos de petición y respuesta.
Cada acción puede ser mapeada en la hoja de rutas, siendo así accesible para cualquier cliente. A continuación se muestra un fragmento de la hoja de rutas empleada en este proyecto (ver ilustración 14).

```
'POST /sesion/usuario' : 'SessionController.usuario',
'POST /sesion/rango' : 'SessionController.rango',
'POST /sesion/permisos' : 'SessionController.permisos',
'POST /sesion/franquicias' : 'SessionController.franquicias',
'POST /sesion/locales' : 'SessionController.locales',
'POST /sesion/tiposUsuario' : 'SessionController.tiposUsuario',
'POST /sesion/alarmas' : 'SessionController.alarmas',
'POST /sesion/cerrar' : 'SessionController.cerrar',
'POST /franquicias/cargaLocales' : 'FranquiciasController.cargaLocales',
'POST /locales/cargaPuntos' : 'LocalesController.cargaPuntos',
'POST /locales/cargaPuntosEnergia' : 'LocalesController.cargaPuntosEnergia',
'POST /locales/almacenaAuto' : 'LocalesController.almacenaAuto',
'POST /usuarios/crea' : 'UsuariosController.crea',
'POST /usuarios/login' : 'UsuariosController.login',
'POST /usuarios/almacenaRegID' : 'UsuariosController.almacenaRegID',
```

Ilustración 14. Hoja de rutas del servidor

Los controladores del servidor desempeñan la función de manipular el modelo, trabajando con los datos que representa. Para ello pueden acceder a estos datos a través de las operaciones que ofrece Sails, como ocurre en el siguiente ejemplo (ver ilustración 15).

```
/*
*Función que carga los puntos almacenados en la base de datos, pertenecientes a una sala
*pasada como parámetro.
*/
cargaPuntos: function(req, res) {
    var usuario=req.session.user['id']; //Usuario logueado
    var msgErr;
    var msgLog;

    //Datos de la sala
    var sala=req.param('sala');
    //Recoge la lista de puntos y la retorna
    Puntos.find({sala : sala},function(err, puntos) {
        //Si ocurre un error lo registra en el log y lo trata.
        if (err){
                var msgErr=("Error en la base de datos");
                var msgLog=("Listar puntos. Mensaje Error: "+msgErr);
                LogWebService.log(usuario,'Error',msgLog);
                return res.serverError(msgErr);
            }
        return res.send({puntos: puntos});
    });
},
```

Ilustración 15. Implementación de función del servidor

La función cargaPuntos recibe como parámetro el ID del local del que desea conocer sus puntos de medición. A través de la función *find*, el controlador pide al modelo de Puntos, los puntos con referencia a la sala pasada como parámetro.
Una vez devueltos los datos necesarios, o un mensaje de error, el controlador envía la respuesta.

Sails también ofrece la posibilidad de ejecutar consultas más complejas a través del comando *query*. A continuación se muestra un ejemplo de su uso (ver ilustración 16).

```
/*
*Función que carga los puntos de energía almacenados en la base de datos, pertenecientes a una sala
*pasada como parámetro.
*/
cargaPuntosEnergia: function(req, res) {
    var usuario=req.session.user['id']; //Usuario loggeado
    var msgErr;
    var msgLog;

    //Datos de la sala
    var sala=req.param('sala');

    var query="SELECT puntos.puntoID, puntos.idPunto, puntos.nombre, puntos.sala, puntos.latitud
              "FROM puntos, puntosEnergia "+
              "WHERE puntos.puntoID=puntosEnergia.puntoID AND puntosEnergia.localID="+sala;
    //Recoge la lista de puntos y la retorna
    Puntos.query(query,function(err, puntos) {
        //Si ocurre un error lo registra en el log y lo trata.
        if (err){
                var msgErr=("Error en la base de datos");
                var msgLog=("Listar puntos. Mensaje Error: "+msgErr);
                LogWebService.log(usuario,'Error',msgLog);
                return res.serverError(msgErr);
            }
        return res.send({puntos: puntos});
    });
},
```

Ilustración 16. Implementación de función del servidor

El comando *query*, ejecuta la sentencia asociada sobre la base de datos, devolviendo, al igual que cualquier otra operación adaptada, los datos requeridos a través de una función *callback*.

A través de los modelos y controladores expuestos, Sails ofrece un perfecto nexo de unión entre las capas de presentación y de persistencia.

5.3 Capa de presentación

Las dos aplicaciones forman la capa de presentación, a través de la cual el usuario entrará en contacto con el sistema.
Ambas están implementadas con HTML5, CSS3 y AngularJS; y presentan una estructura similar, con vistas y controladores asociados a ellas.

5.3.1 Aplicación web

La aplicación web ofrece al usuario la capacidad de gestionar sus alarmas, su cuenta y los usuarios que estarán bajo su control.
En el apartado de diseño se ha ofrecido un esquema general de sus vistas y controladores, y a continuación, se mostrarán pequeños fragmentos de su implementación.

```
<div ng-show="muestraAlarmas">
    <div class="titulo">
        <h4>Lista de alarmas</h4>
    </div>
    </br>
    <input ng-model="searchText">
    </br></br>
    <ul class="list-group">
        <li class="list-group-item"
        data-ng-repeat="alarma in alarmas | filter:searchText">
            <label>Local: </label> {{alarma.nombreSala}}</br>
            <label>Medidor: </label> {{alarma.nombrePunto}} </br>
            <label>Parámetros de la alarma: </label> {{alarma.tipo}}
            {{alarma.restriccion.toLowerCase()}} que {{alarma.valor}}</br>
            <label>Descripción:</label> {{alarma.descripcion}}</br>

            <div class="btn-group btn-group" role="group" aria-label="...">
                <button class="btn btn-primary boton-vertical" type="button"
                ng-click="eliminarAlarma(alarma)">
                    Eliminar alarma
                </button>
                <button class="btn btn-primary boton-vertical" type="button"
                ng-click="editarAlarma(alarma)">
                    Editar alarma
                </button>
                <button class="btn  btn-primary boton-vertical" type="button"
                ng-click="cambiarEstado(alarma)">
                    {{alarma.accion}} alarma
                </button>
            </div>
        </li>
    </ul>
</div>
```

Ilustración 17. Plantilla de la aplicación web

La ilustración 17 muestra un fragmento de una de las páginas que forman la vista de la aplicación web.
A través del atributo *ng-controller*, situada en la etiqueta de cada página HTML, Angular enlaza las vistas con sus respectivos controladores.
Por otra parte, los elementos escritos entre corchetes, están asociados a variables del controlador, por lo que un cambio en ellas se vería automáticamente reflejado en la vista y viceversa. Este 'data-binding' es una de las principales ventajas de AngularJS.

Angular ofrece también la posibilidad de asociar eventos a distintos elementos del HTML, como botones, selectores o menús desplegables. Esta facultad facilita en gran medida el código con respecto a JavaScript.
Un ejemplo es el siguiente fragmento de código (ver ilustración 18), que muestra una función vinculada al botón 'Eliminar alarma', mostrado en la anterior imagen.

```
/*
*Función ejecutada al pulsar Eliminar alarma, que la elimina de la base de datos.
*/
$scope.eliminarAlarma=function(alarma){

    //Llamada a la base de datos para eliminar la alarma.
    $http.post('/alarmas/eliminaAlarma', {alarma: alarma['alarmaID']}).
    success(function(data, status, headers, config) {
        alert("La alarma ha sido eliminada");
        //Se elimina también de la lista de alarmas.
        var indice=$scope.alarmas.indexOf(alarma);
        $scope.alarmas.splice(indice,1);
    }).
    error(function(data, status, headers, config) {
        alert(data);
    });

}
```

Ilustración 18. Función del controlador de la aplicación web

En un controlador de Angular, se usa *$scope* para designar todos los objetos que pueden ser referenciables en una vista. Así, la función eliminarAlarma(alarma), aparece vinculada al botón 'Eliminar alarma'.
A través de la rutina *$http*, Angular ofrece una forma sencilla de realizar intercambio de datos con el servidor.

A continuación se mostrarán las principales vistas de la aplicación web. La primera (ver ilustración 19) corresponde la pantalla de acceso al gestor.

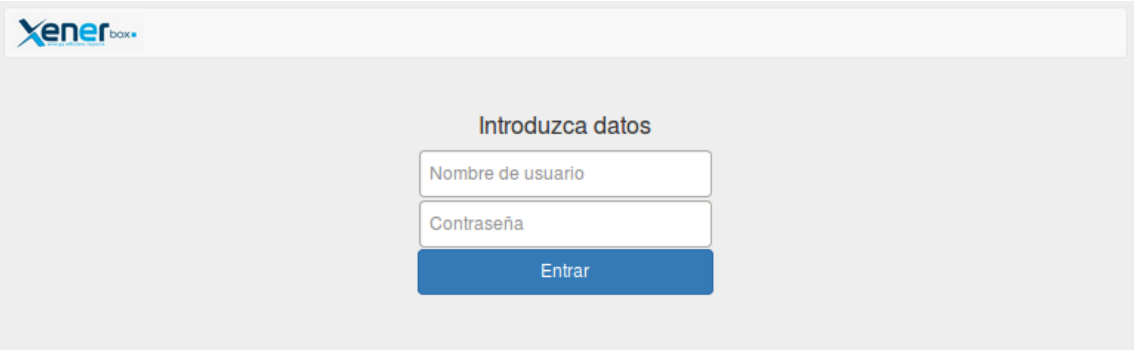

Ilustración 19. Pantalla de acceso a la aplicación web

Las ilustraciones 20 y 21 son las principales vistas correspondientes a la gestión de alarmas. La ilustración 20 muestra la lista de alarmas creadas por el usuario logueado, así como algunas opciones para su manejo. La ilustración 21 muestra el proceso de creación de una alarma.

Ilustración 20. Pantalla de gestión de alarmas

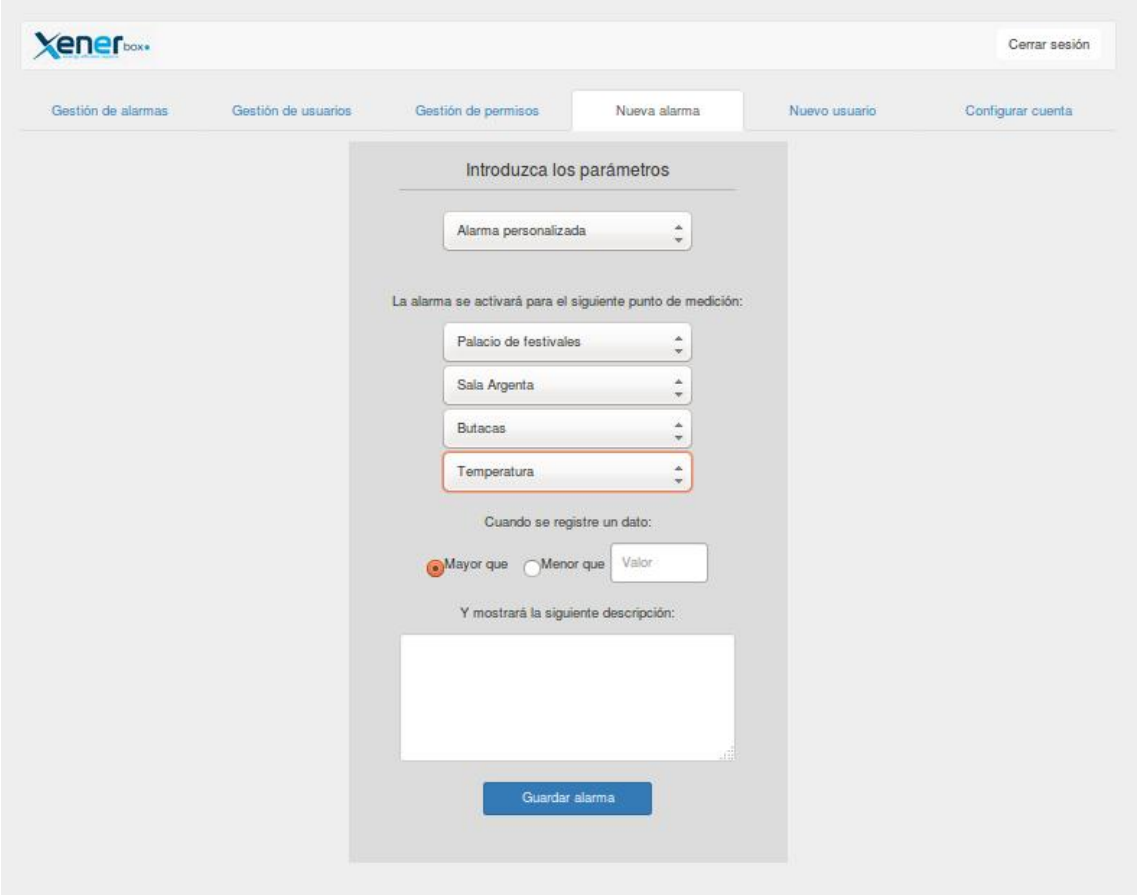

Ilustración 21. Pantalla de creación de alarmas

Respecto a la gestión de usuarios, sus principales vistas están recogidas en las ilustraciones 22, 23 y 24.
La primera muestra la pantalla con la lista de usuarios inferiores jerárquicamente al usuario logueado, la segunda el proceso de creación de un nuevo usuario y la tercera la gestión de permisos de usuarios.

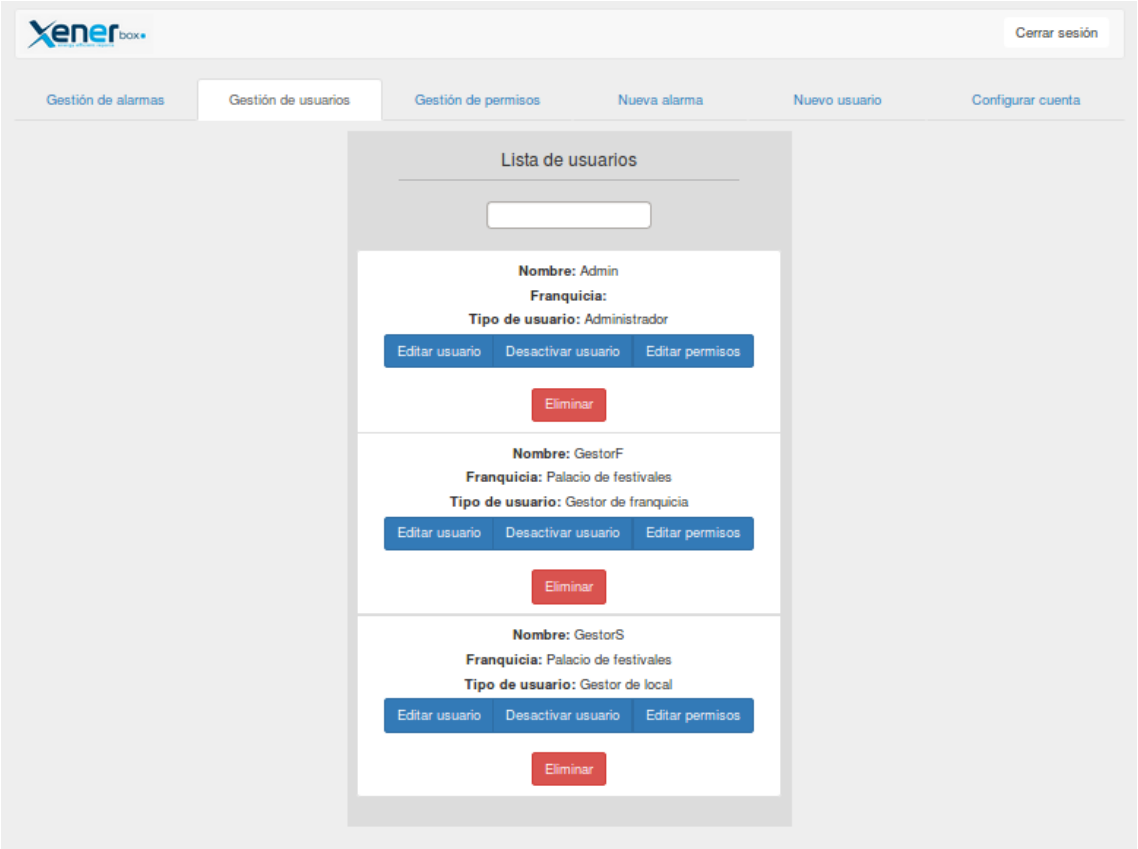

Ilustración 22. Pantalla de gestión de usuarios

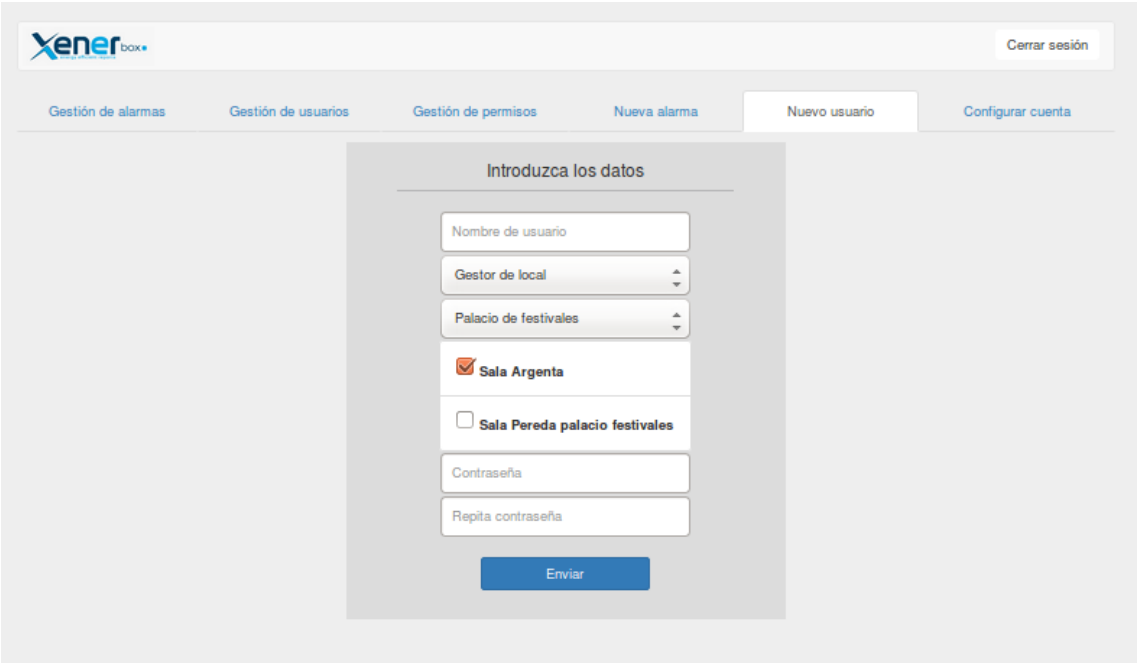

Ilustración 23. Pantalla de alta de usuarios

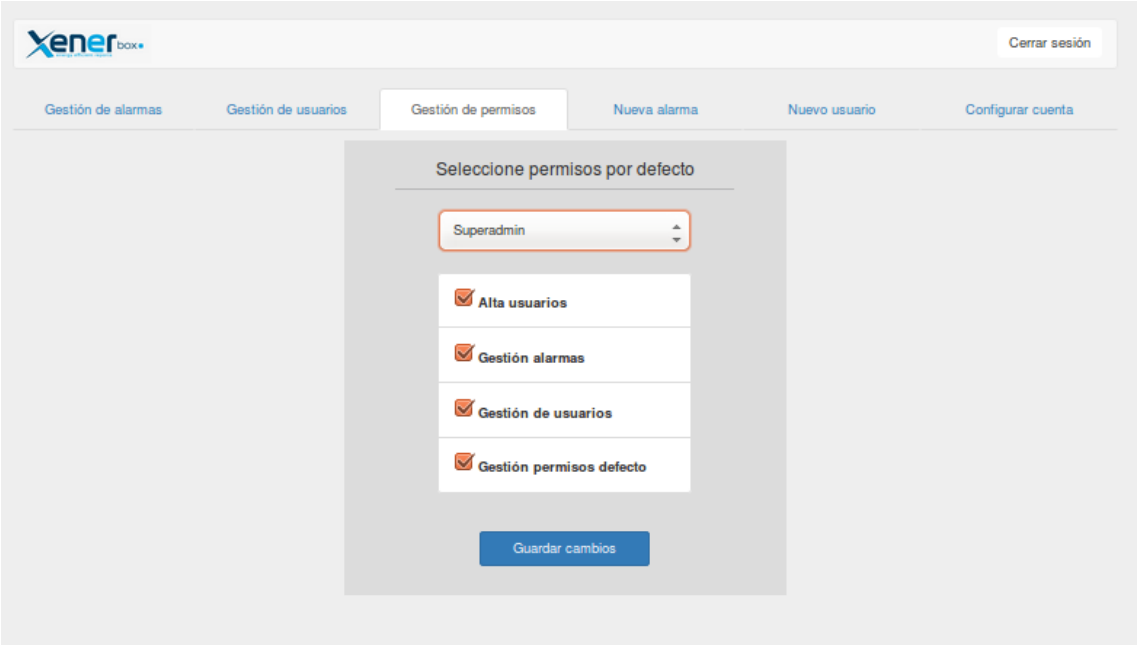

Ilustración 24. Pantalla de gestión de permisos de usuario

5.3.2 Aplicación móvil

Las vistas de la aplicación móvil, aunque bastante similares, presentan alguna diferencia respecto al gestor web. Para su implementación se ha utilizado el framework de HTML5 Onsen UI, que ofrece algunos elementos prediseñados que aportan a la aplicación el aspecto de una nativa.

```
<div class="contenido">

    <ons-list class="lista-salas">
        <ons-list-item  modifier="tappable" class="list-item-container" ng-repeat="punto in puntos"
        ng-click="accederPunto(punto)">
            <ons-row>
                <ons-col width="30%">
                    <img src="styles/img/03_SELECCION/icono_medidor.png" class="thumbnail">
                </ons-col>
                <ons-col>
                    <div class="name">
                        {{punto.nombre}}
                    </div>
                </ons-col>
            </ons-row>
        </ons-list-item>
    </ons-list>

</div>
```

Ilustración 25. Plantilla de la aplicación móvil

En este ejemplo (ver ilustración 25) se crea una lista con la etiqueta *ons-list*, que muestra los elementos de una lista creada en el controlador, enlazada con la vista a través del comando de Angular *ng-repeat*.
A continuación se muestra la función que genera esta lista (ver ilustración 26), llamada a través del comando *ng-init*, situado en la etiqueta de creación de la página, que indica las acciones que han de ejecutarse al cargar la plantilla.

```
//Función que recoge la lista de puntos disponibles, para la sala guardada en la variable
//global.
$scope.cargaPuntos = function(){
    var salaID=salaSelec['id']
    //Llamada al controlador de salas de la APP web para recoger los puntos correspondientes a la
    //sala pasada como parámetro.
    $http.post(url+'/locales/cargaPuntos',{sala : salaID}
      ).success(function(data, status, headers, config) {
          var longitud=data['puntos'].length;//Longitud del array=numero de puntos

          //Bucle que recorre todos los puntos, añadiéndolos al array.
          for (i=0;i<longitud;i++){
              //Añadir al array los datos del punto
              var nombrePunto=(data['puntos'][i]['nombre']).toUpperCase();
              var codPunto=(data['puntos'][i]['idPunto']);
              var idPunto=(data['puntos'][i]['puntoID']);
              var tipoPunto=(data['puntos'][i]['tipo']).toUpperCase();

              if(tipoPunto==nombreMedida){
                  $scope.puntos.push({nombre:nombrePunto,codigo:codPunto,id:idPunto});
              }
          }
      }).
      //Si ha habido un error al cargar los puntos, lo notifica.
      error(function(data, status, headers, config) {
          navigator.notification.alert(
                "No es posible mostrar los puntos en este momento",
                function(){},
                "Ha ocurrido un error",
                "Aceptar"
          );
          //alert("No es posible mostrar los puntos en este momento");
      });//fin post */
};//fin function
```

Ilustración 26. Función del controlador de la aplicación móvil

La función hace una petición HTTP al servidor, solicitando los puntos pertenecientes a la sala pasada como parámetro. Los puntos devueltos son almacenados en la lista que se muestra en la plantilla usada como ejemplo.

La aplicación móvil realiza también peticiones a la API web facilitada por Xener, solicitando los datos recogidos por los medidores.
Estas llamadas se hacen a través de una URL, en la cual van incluidos los parámetros de la llamada, que devuelve un objeto JSON con los resultados.

Los parámetros de la llamada incluyen una clave de acceso, facilitada para el desarrollo de esta aplicación; los códigos de las franquicias, salas y magnitudes de las cuales se requiera información; parámetros para definir el formato de los datos como días o rangos horarios; y unas palabras clave que indican el tipo de información solicitada.

La ilustración 27 muestra un ejemplo de una función que realiza una llamada a la API.

```
//Recoge las tarifas de energia
direccion=('https://www.seinon.com/API/API2.asp?Q=DThplOrvC52S6FR2Aq&CP='
            +salaSelec.cod+'&IDPTO='+puntoSelec.cod+'&OUT=SUMI');
$http.post(direccion, {}
).success(function(data, status, headers, config) {
    if(JSON.stringify(data[0])!="{\"STATUS\":\"ERROR\"}"){

        //Calcula las tres tarifas
        calculaTarifas(data['SUMI'][0]['te1'],data['SUMI'][0]['te2'],data['SUMI'][0]['te3']);
        calculaEnergia(datosE);
        calculaPotencia(datosP);

        $scope.impuestoElectrico=($scope.totalEnergia+$scope.totalPotencia)*1.05113*0.04864;
        $scope.impuestoElectrico=parseFloat($scope.impuestoElectrico.toFixed(2));

        $scope.IVA=($scope.totalEnergia+$scope.totalPotencia+$scope.impuestoElectrico)*0.21;
        $scope.IVA=parseFloat($scope.IVA.toFixed(2));

        $scope.totalFactura=$scope.totalEnergia+$scope.totalPotencia+$scope.impuestoElectrico+$scope.IVA;
        $scope.totalFactura=parseFloat($scope.totalFactura.toFixed(2));
    }else{
        navigator.notification.alert(
                "No es posible acceder a los datos del suministro en este momento",
                function(){},
                "Ha ocurrido un error",
                "Aceptar"
        );
    }

}).
```

Ilustración 27, Llamada a la API web

En la URL van incluidos los parámetros referentes a la clave de acceso, el código de la sala y del punto del cual se quieren conocer datos del suministro eléctrico.
En caso de que el resultado no sea erróneo, se acceden a los valores del objeto necesarios para la función y se opera con ellos.

Por último, se mostrarán las principales vistas ya implementadas de la aplicación móvil. La primera ilustración 28 muestra la pantalla de acceso a la aplicación.

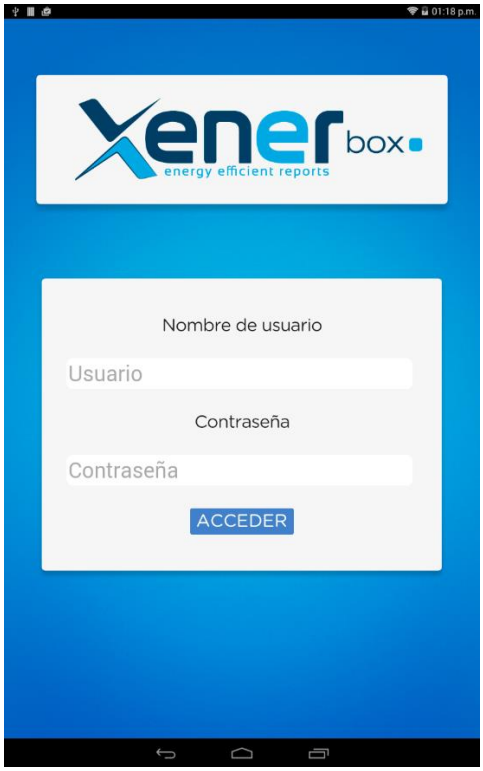

Ilustración 28. Pantalla de acceso a la aplicación móvil

La ilustración 29 muestra dos ejemplos de los menús implementados en la aplicación La pantalla de la izquierda muestra el menú de opciones, con los distintos servicios que ofrece la aplicación. La pantalla de la derecha muestra una lista con los puntos de medición que ofrecen un servicio seleccionado en un local concreto.

Ilustración 29. Pantallas de menú de opciones

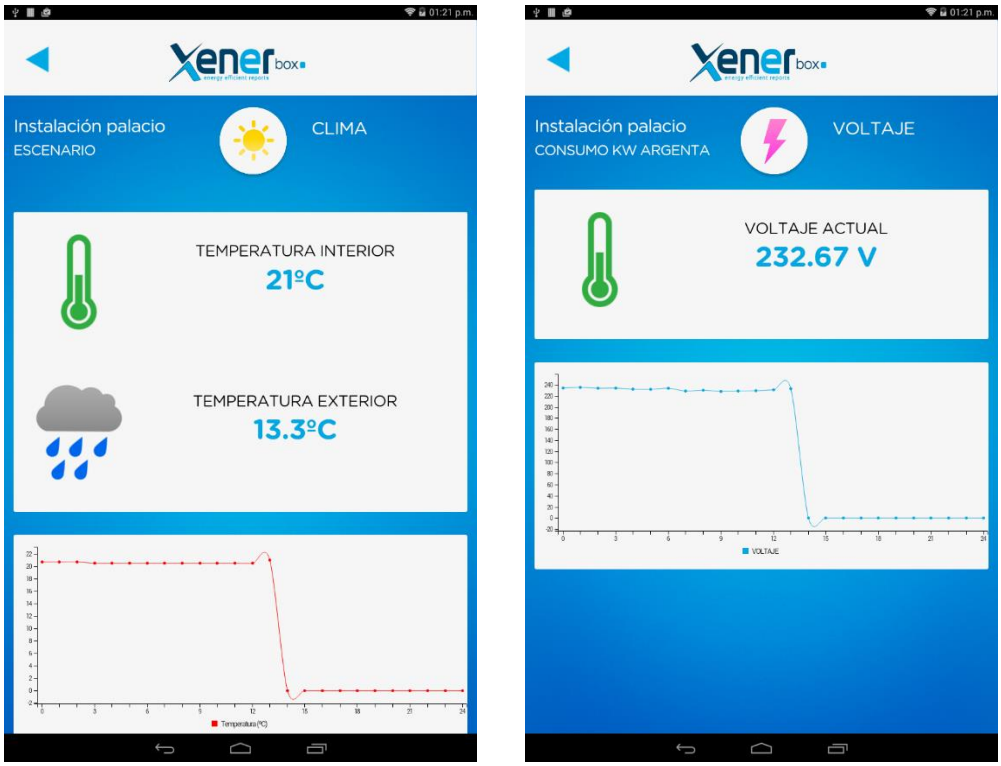

Ilustración 30. Pantallas de consulta de datos

Las pantallas expuestas en la ilustración 30, ofrecen al usuario información sobre ciertas magnitudes de los puntos de medición. Todas siguen un mismo modelo, mostrando los valores recogidos en el momento de la consulta y una gráfica con los datos almacenados a lo largo del día.

La aplicación ofrece también comparativas de las distintas magnitudes, en las que los parámetros son seleccionados por el usuario (ver ilustración 31.)

Ilustración 31. Pantalla de comparativas

La aplicación también hace posible la consulta de las distintas alarmas producidas (ver ilustración 32).

Ilustración 32. Pantalla de alarmas producidas

5.4 Implementación de rutinas

El sistema cuenta también con algunas rutinas, ejecutadas periódicamente, que complementan algunas funcionalidades de las aplicaciones. Éstas están programadas en NodeJS y son ejecutadas por el crontab del sistema, con una periodicidad establecida.

5.4.1 Rutinas de volcado de datos

Las franquicias, locales y puntos de medición que debe recoger la aplicación son proporcionadas por la API web de la plataforma Xener.
Para mantener la aplicación actualizada, se han implementado una serie de rutinas, ejecutadas varias veces al día, que recogen todos los datos referentes a los medidores registrados por la plataforma, y los vuelcan en la base de datos.
De esta forma, el sistema se mantiene al día de cualquier cambio o inserción.

Las rutinas, que forman parte del servidor, están programadas en NodeJS y presentan la siguiente estructura (ver ilustración 33).

```
/*
*Función que dada una franquicia, recoge de la API todas sus salas y las almacena en la base de datos.
*/
function almacenaSalas(franquicia){
    async.waterfall([
        //Tarea que recoge las salas de la franquicia.
        function(callback){
            recoge.recogeSalas(franquicia,function(err, salas){
                callback(null,salas);
            });
        }
    //Tarea final que guarda en la base de datos las salas
    ],function(err,salas){
        salas=JSON.stringify(salas);
        var data = querystring.stringify({salas: salas, franquicia:franquicia});
        //Opciones de llamada al controlador que almacenará las salas.
        var options = {
            host: 'localhost',
            port: 1337,
            path: '/locales/almacenaAuto',
            method: 'POST',
            headers: {
                'Content-Type': 'application/x-www-form-urlencoded',
                'Content-Length': data.length
            }
        };
        //Llamada que recoge y trata los datos
        var req = http.request(options, function(res) {
            res.setEncoding('utf8');
            //Una vez leídos los trata
            res.on( 'end' , function(){
            });
        });
        req.write(data);
        req.end();
    });
}
```

Ilustración 33. Rutina de volcado de datos

Las rutinas utilizan la librería *async* para controlar la concurrencia, ya que NodeJS es asíncrono.
Por otro lado, las llamadas HTTP a los controladores del servidor, que se encargan de la gestión de los datos se hacen mediante la interfaz HTTP de Node.

5.4.2 Rutinas de envío de alarmas

Uno de los principales servicios que ofrece la aplicación es la programación y envío de alarmas.
Al igual que para el volcado de datos, se han creado una serie de rutinas, implementadas en NodeJS, que se ejecutan con un periodo de cinco minutos.

Estas rutinas recorren todas las alarmas activas almacenadas en la base de datos, comprueban sus valores límites y los comparan con los recogidos en ese instante por los medidores.
Si alguno de los valores medidos se sale de los límites establecidos, se envía una notificación al dispositivo móvil del usuario a cuyo nombre está registrada la alarma.

El envío de notificaciones se realiza mediante el plugin PushPlugin de Cordova, que utiliza los servicios GCM de Google.

Su funcionamiento es el siguiente:

- Cuando un usuario se loguea en la aplicación, el regID de su dispositivo móvil se almacena en la base de datos.

- Las rutinas de envío de alarmas se ejecutan periódicamente y cuando detectan una alarma, la almacenan en la tabla de la base de datos que recoge las alarmas producidas (ver ilustración 34), para su posterior consulta.

```
/**
*Función que almacena la alarma producida correspondiente al ID pasado como parámetro.
*/
function almacenaAlarma(alarmaID,valor,callback){
    var data = querystring.stringify({alarmaID: alarmaID, valor:valor});
    //Opciones de llamada al controlador que almacenará la alarma
    var options = {
        host: 'localhost',
        port: 1337,
        path: '/alarmasProducidas/almacenaAlarmaAuto',
        method: 'POST',
        headers: {
            'Content-Type': 'application/x-www-form-urlencoded',
            'Content-Length': data.length
        }
    };
    //Llamada al controlador
    var req = http.request(options, function(res) {
        res.setEncoding('utf8');
        res.on( 'end' , function(){
            callback(null);
        });
    });
    req.write(data);
    req.end();

}
```

Ilustración 34. Función para almacenar alarmas

- A su vez, se ejecuta un procedimiento programado en Ruby, pasando como parámetros el regID del dispositivo móvil al que enviar la alarma (almacenado en la base de datos, en el registro de usuarios), el título y el texto de la notificación (ver ilustración 35).

```
function mandaNotificacion(notId, sala, datos,callback){
    var notificacion="Producida en "+sala;
    var llamada="ruby /var/www/xenerbox/xener/mobile/pushGCM.rb "+
    datos['regID']+" '"+datos['titulo']+"' '"+notificacion+"' "+notId;
    exec(llamada);
}
```

Ilustración 35. Función para enviar notificación

- El procedimiento Ruby, que se muestra en la ilustración 36, envía la alarma al dispositivo con el regID, el título y el texto recibidos como parámetro.

```
# encoding: UTF-8
require 'rubygems'
require 'pushmeup'
regid = ARGV[0];
titulo = ARGV[1];
mensaje = ARGV[2];
notId=ARGV[3];
GCM.host = 'https://android.googleapis.com/gcm/send'
GCM.format = :json
GCM.key = "AIzaSyCEWW_Cpk5zIAIiOQq3mx2xkQfp2Mng9Ik"
destination = [regid]
data = {:notId=> notId,:title => titulo ,:message => mensaje, :msgcnt => 1, :soundname => "beep.wav"}

GCM.send_notification( destination, data)

puts 1
```

Ilustración 36. Procedimiento de envío de notificación

- El plugin de la aplicación móvil se encarga de recibir la notificación y mostrarla en el dispositivo.

6 Pruebas

La última fase de desarrollo de un sistema software es la de pruebas y tiene dos objetivos fundamentales:

- Comprobar que el sistema cubre las necesidades y requisitos del cliente, aportando los contenidos y funcionalidades esperados.

- Detectar fallos en el sistema, corrigiendo los errores que puedan producirse o los comportamientos no deseados.

Para ello se realizan varios periodos de pruebas, en los que se verifica el software a distintos niveles [12], que serán descritos a continuación.

6.1 Pruebas unitarias

Las pruebas unitarias tienen como objetivo asegurar que cada módulo del sistema funciona según lo esperado.
Para ello los divide en unidades lógicas fáciles de probar y define sobre cada una de ellas unos casos de prueba, comprobando que los resultados son correctos en todos los caminos que pueda tomar el software.

En este proyecto las pruebas unitarias se han realizado sobre el modelo y los controladores, comprobando para cada función, que el sistema se comportaba de forma correcta y los datos obtenidos o almacenados en la base de datos eran los esperados.

Se ha puesto especial hincapié en el tratamiento de los datos recogidos de la API facilitada por la plataforma 'Xener', pues la aplicación funciona en torno a estos datos.
Por una parte, ha sido de vital importancia asegurar que la información mostrada fuese siempre la correcta, coincidiendo con los últimos valores tomados de los medidores; y que mantuviese una cierta coherencia, absteniéndose de mostrar los datos que, debido a posibles errores en los dispositivos de medición, se salían de unos límites aceptables.
Por otra, se ha tenido especial cuidado a la hora de realizar el cálculo de las facturas, ya fuera para mostrar en la aplicación móvil o para la programación de alarmas de consumo, comprobando en repetidas ocasiones que las operaciones y cálculos realizados eran los correctos.

6.2 Pruebas de integración

Una vez se ha comprobado que todos los módulos desempeñan sus tareas según lo esperado de forma independiente, es hora de probar su conjunto.
Las pruebas de integración tratan de identificar los errores producidos tras la unificación de los módulos, verificando que tanto el 'back-end' como el 'front-end' funcionan correctamente.

En este proyecto se han realizado pruebas de integración para comprobar que el conjunto de las interfaces de usuario realizaban sus funciones siguiendo el flujo de eventos deseado.
Se ha comprobado también, con especial cuidado, la correcta comunicación entre los distintos módulos del sistema, desde la capa de presentación hasta la de persistencia.

6.3 Pruebas de sistema

Las pruebas de sistema tienen como objetivo comprobar que el sistema final funcione correctamente, en todo su conjunto.
Para ello han de probarse los distintos casos de uso, verificando que el comportamiento y los resultados son los deseados.

Para este proyecto, se han simulado los diferentes escenarios, comprobando que todos los requisitos y especificaciones del sistema se hubiesen cumplido.
Estas pruebas fueron decisivas para el resultado final del apartado de alarmas, en el que intervienen las tres capas del sistema.

6.4 Pruebas de aceptación

En esta fase, es el cliente quien tiene que verificar si el sistema cumple sus requisitos, ofreciendo los servicios esperados.

Durante el desarrollo de este proyecto, al final de cada iteración, se realizaba una reunión con el cliente para mostrarle los cambios y avances realizados.
En estas reuniones se probaban las distintas funcionalidades del software, cotejando que todo estaba implementado según lo deseado e indicando los cambios y ampliaciones necesarios para el proyecto.

7 Conclusiones y trabajos futuros

7.1 Conclusiones

Tras la finalización del proyecto, es hora de sacar las conclusiones de un trabajo que ha supuesto una inversión de tiempo y esfuerzo.

La plataforma 'Xener' cuenta ahora con una aplicación lista para su puesta en marcha y que cumple sus objetivos finales, gracias a la cual, sus clientes podrán hacer un mejor uso y sacar un mayor rendimiento a los dispositivos instalados en sus locales.
Se espera que esta aplicación le reporte un incremento en sus ventas, además de una mejora en sus servicios.

A título personal, el proyecto me ha aportado conocimiento, confianza y seguridad en mi trabajo.
Por una parte ha dado forma a todos mis años de carrera, pues en su desarrollo he tenido que emplear muchos de los conocimientos que he ido adquiriendo a lo largo de los cursos y las asignaturas. Conocimientos que en su momento me parecieron demasiado abstractos, pues a pesar de las muchas prácticas, no encontraba la manera de integrarlos entre sí, de sacarlos de ese marco teórico. El desarrollo de este proyecto, más grande y real, ha hecho que les diese uso y los ha llevado a la práctica.
Por otro lado, el haberlo realizado fuera de la universidad me ha preparado en cierto modo para lo que viene después. Durante mis prácticas en Netkia Soluciones S.L. he sido una más; he aprendido cómo se trabaja en una empresa y cómo aplicar mis conocimientos; y he descubierto la verdadera importancia de lo estudiado hasta ahora.
Tras las prácticas me siento más preparada y confiada para entrar en el mundo laboral.

En cuanto al aspecto técnico, me ha brindado la oportunidad de probar nuevas tecnologías y profundizar un poco en el desarrollo de aplicaciones móviles, algo que no tenía demasiado tratado hasta el momento.

La sugerencia por parte de la empresa de utilizar la herramienta Sails, me ha descubierto un gran framework, que facilita en gran medida la programación del 'back-end' de cualquier aplicación. Es una herramienta potente y que presenta una gran compatibilidad, tanto con diferentes modelos de bases de datos, como con un gran número de tecnologías de desarrollo de front-end.
En lo que a Angular se refiere, supone un gran avance para la programación de aplicaciones web y móviles con respecto a cualquier versión de JavaScript. Aunque en principio no sea del todo sencillo su aprendizaje, una vez adquiridos los conocimientos necesarios, su uso sí que lo es.
Ambas han sido el gran descubrimiento de este proyecto y no me cabe duda de que les daré uso más adelante.

En definitiva, estos han sido unos meses de intenso aprendizaje, de los que no me llevo sino experiencias positivas y que son un perfecto final para mi carrera.

7.2 Trabajos futuros

La aplicación desarrollada está abierta a posibles ampliaciones para aumentar su compatibilidad y adaptabilidad.

La ampliación más factible a día de hoy es la de adaptarla a distintas plataformas móviles como son iOS o Windows Phone.
En un principio, el cliente había encargado a Netkia una aplicación para dispositivos Android, pues era el tipo de dispositivo más usado por los usuarios de la plataforma 'Xener'.
De cara a un futuro próximo se podría contemplar adaptarla a dispositivos iOS, pues la empresa en la que se ha desarrollado, ya cuenta con aplicaciones para esta plataforma.
Un poco más a largo plazo sería posible su adaptación también a dispositivos Windows Phone, cubriendo así los tres grandes sistemas operativos para dispositivos móviles.

Por otra parte, podría ampliarse también el área geográfica que contempla el desarrollo de la aplicación.
Por motivos de diferencias en los parámetros de facturación, ahora mismo la aplicación está diseñada para la gestión de locales de la península. Esto podría extender, sin demasiada dificultad a locales de todo el país, incluyendo las islas y las ciudades autónomas.

Referencias

[1] Laboratorio Nacional de Calidad del Software (2009) "Ingeniería software: Metodologías y ciclo de vida" Inteco

[2] MySQL.com "10 razones para elegir MySQL para las aplicaciones web de la próxima generación" accedido en https://www.mysql.com/ [última visita 25-01-2016]

[3] NodeJS.org "NodeJS" accedido en https://nodejs.org/en/ [última visita 25-012016]

[4] SailsJS.org "SailsJS" accedido en http://sailsjs.org/ [última visita 25-01-2016]

[5] Cordova.apache.org "Cordova" accedido en https://cordova.apache.org/ [última visita 02-02-2016]

[6] AngularJS.org "AngularJS" accedido en https://angularjs.org/ [última visita 02-02-2016]

[7] Onsen.io "Onsen UI" accedido en https://onsen.io/ [última visita 02-02-2016]

[8] Eclipse.org "Eclipse" accedido en https://eclipse.org/ [última visita 30-01-2016]

[9] Developer.android.com "SDK" accedido en http://developer.android.com/ [última visita 31-01-2016]

[10] SublimeText.com "Sublime Text" accedido en https://www.sublimetext.com/ [última visita 03-02-2016]

[11] NoMagic.com "MagicDraw" accedido en http://www.nomagic.com/ [última visita 03-02-2016]

[*12] Ian Sommerville (2006) Ingeniería del Software. 7ªEdición Pearson.*

[13] Jorge Hernan Abad Lodoño (2005) "Tipos de pruebas Software" accedido en http://ing-sw.blogspot.com.es/2005/04/tipos-de-pruebas-de-software.html [última visita 06-02-2016]