Escuela Politécnica Superior

Panel frontal para el controlador IoT de Smart University

Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:
Alejandro Torres Mateu

Tutor/es:
José Vicente Berná Martínez

Universitat d'Alacant
Universidad de Alicante

Septiembre 2017

Motivación, justificación y objetivo general

Todavía recuerdo el día en el que conseguí conectar un servomotor a una placa Arduino y controlarlo remotamente usando mi teléfono móvil. Se lo enseñé a mis padres y, cuando los vi jugar con aquel prototipo, sentí que un día estas tecnologías cambiarían la sociedad tal y como la conocemos.

En el último año de carrera, uno de mis profesores me comentó la idea de unirme a un proyecto que estaban llevando a cabo en la universidad mediante una beca de prácticas. Se trata del proyecto Smart University, que estudia la creación de universidades y ciudades inteligentes en las que gracias a los datos podamos conocer las condiciones ambientales de estos espacios y así mejorar nuestra calidad de vida tanto dentro como fuera de estos.

Cuando entré al equipo y me contaron el proyecto con detalles sentí lo mismo que con el servomotor: la ilusión de poder hacer algo innovador que, de una forma u otra, afectase a la vida de las personas. En realidad, dotar de inteligencia a objetos que a priori no son inteligentes es algo que ha intentado el ser humano desde mucho antes de que Mary Shelley escribiera la novela *Frankenstein* en 1818. Así, aunque es cierto que el proyecto en sí no pretende dotar de inteligencia "humana" a los objetos, sí pretende conectarlos a Internet y, de esta forma, permitir que las ciudades en su conjunto sean más inteligentes mediante el uso intensivo de las tecnologías de la información y la inferencia de conocimiento a través de las enormes y vastas cantidades de datos que estos sistemas son capaces de generar. Esto incluye paradigmas que conocemos hoy en día como *Internet Of Things* (IoT)*, Big Data, Data Science* o *Smart Cities.*

¿Cuál es mi papel en todo esto? Dentro del proyecto necesitaban a alguien que trabajara en un sistema de sondas inteligentes capaces de recolectar diversos datos, como el nivel de ruido, la polución, la temperatura y la calidad del wifi, entre otros. En el momento en el que entré en el proyecto, este se componía de un prototipo básico de sondas que se conectaban con la aplicación principal, en la que se podían visualizar los datos.

Mi trabajo durante la beca consistió en desarrollar la funcionalidad total del controlador y el *software* interno de las sondas, así como la conexión con la aplicación principal del sistema. Una vez terminado el proyecto, pude observar que existían diversas carencias en el planteamiento original. De hecho, ser el constructor del sistema me permitió analizar a través de pruebas

empíricas el funcionamiento y el rendimiento del mismo y, así, detectar estos aspectos que podían ser mejorables. Por ejemplo, aunque la herramienta había sido diseñada para ser genérica, flexible y modular, no había sido concebida para ofrecer interfaces amigables para su administración, ya que descargaba estas tareas sobre interfaces tradicionales por línea de comandos. Es por ello por lo que parecía interesante poder hacer una pequeña aportación al proyecto desde el área del desarrollo de interfaces web y contribuir, precisamente, con un panel de administración y monitorización capaz de aportar al conjunto ese carácter de producto terminado. Además, el desarrollo de este nuevo frontal brindaba también la oportunidad de incluir diversas mejoras y optimizaciones necesarias, tal y como se hizo patente durante el desarrollo del sistema inicial, así como otras funcionalidades requeridas en el sistema para poder implementar las propias del panel de administración.

A mi tutor, mi familia y mi pareja.

"La vida no debería ser un viaje hacia la tumba con
la intención de llegar a salvo con un cuerpo bonito y bien conservado,
sino más bien llegar derrapando de lado, entre una nube de humo,
completamente desgastado y destrozado,
y proclamar en voz alta: ¡Uf! ¡Vaya viajecito!"

- Hunter S. Thompson

Índice

1 Introducción ... 1

1.1 El problema ... 3

2 Antecedentes ... 6

2.1 Smart University ... 6

2.2 Controlador de sondas ... 12

2.2.1 MQTT ... 12

2.2.2 Persistencia en base de datos: MySQL ... 14

2.2.3 Backend: Controlador ... 16

2.3 Dispositivos: Sondas ... 17

2.4 Otras fuentes: Smart Meters ... 19

3 Estado del arte ... 21

3.1 Soluciones parciales ... 21

3.1.1 Freeboard.io ... 22

3.1.2 Resin.io ... 24

3.1.3 Helium ... 26

3.2 Soluciones completas ... 29

3.2.1 Thinger.io ... 29

3.2.2 TheThings.io ... 30

3.3 Conclusiones ... 32

4 Objetivos ... 33

5 Metodología ... 34

6 Análisis y especificación de requisitos ... 36

6.1 Descripción general ... 36

6.1.1 Perspectiva del producto ... 36

6.1.2 Funciones del producto ... 37

6.1.3 Características de los usuarios ... 37

6.1.4 Restricciones ... 37

6.1.5 Suposiciones y dependencias ... 38

6.2 Especificación de requisitos ... 38

6.2.1 Requisitos funcionales ... 38

6.2.2 Requisitos de diseño ... 43

6.2.3 Requisitos no funcionales ... 44

7 Diseño ... 46

7.1 Arquitectura del servidor ... 46

7.2 Arquitectura del cliente ... 47

7.3 Diseño de la base de datos ... 50

7.4 Seguridad ... 51
7.5 Diseño interfaces ... 53

8 Implementación del servidor ... 60

8.1 Servicio de autenticación ... 60
8.2 Servicio de paginación y búsqueda ... 62
8.3 Controlador para conocer el estado de las sondas ... 64
8.4 Controlador para regenerar la clave de acceso ... 65
8.5 Creación de nuevas rutas en la API ... 65

9 Implementación del cliente ... 79

9.1 Módulo base ... 79
9.2 Módulo de sondas (probeModule) ... 86
9.3 Módulo de tipos de sensores (sensorTypesModule) ... 88
9.4 Módulo de sensores (sensorsModule) ... 90
9.5 Módulo de peticiones a sensores (watcherLogsModule) ... 92
9.6 Módulo de registros (logsModule) ... 93
9.7 Módulo de aplicaciones (applicationsModule) y módulo de usuarios (usersModule) ... 94
9.8 Módulo de login (loginModule) ... 95

10 Pruebas y validación ... 96

11 Resultados ... 97

11.1 Funcionalidades no implementadas ... 97

12 Conclusiones y trabajo futuro ... 98

12.1 Trabajo futuro ... 98

13 Abreviaturas y definiciones ... 100

Bibliografía ... 102

Índice de figuras

Figura 1 - Dashboard Smart University (Fuente Smart University Alicante) 2

Figura 2 - Consola del servicio de sondas (Fuente propia) 4

Figura 3 - Arquitectura Smart University actual (Fuente propia) 6

Figura 4 - Arquitectura interna Smart University actual (Fuente propia) 7

Figura 5 - Captura de pantalla Smart University: Inicio (Fuente Smart University Alicante) 7

Figura 6 - Captura de pantalla Smart University: Mapas (Fuente Smart University Alicante) 8

Figura 7 - Captura de pantalla Smart University: Mapas. Detalle (Fuente Smart University Alicante) 8

Figura 8- Captura de pantalla Smart University: Mapas. Detalle 2 (Fuente Smart University Alicante) 9

Figura 9 - Captura de pantalla Smart University: Analíticas (Fuente Smart University Alicante) . 10

Figura 10 - Captura de pantalla Smart University: Analíticas 2 (Fuente Smart University Alicante) 11

Figura 11 - Captura de pantalla Smart University: Menu Configuración (Fuente Smart University Alicante) 11

Figura 12 - Arquitectura sistema controlador de sondas (Fuente propia) 12

Figura 13 - Arquitectura MQTT (Fuente propia) 13

Figura 14 - Jerarquía MQTT (Fuente propia) 14

Figura 15 - Esquema base de datos inicial (Fuente propia) 15

Figura 16 - Topología controlador de sondas con servicio MQTT (Fuente propia) 16

Figura 17 - Arquitectura interna del controlador de sondas (Fuente propia) 17

Figura 18 – Sonda: Raspberry Pi, Grove Pi y sensores (Fuente propia) 18

Figura 19 - Flujo Sonda – Controlador (Fuente propia) 19

Figura 20 - Gráficas de puntos de acceso inalámbricos y consumo eléctrico (Fuente Smart University Alicante) 20

Figura 21 - Logo de Freeboard.io (Fuente Freeboard.io) 22

Figura 22 - Captura de panel creado con Freeboard.io (Fuente Freeboard.io) 23
Figura 23 - Logo Resin.io (Fuente Resin.io) 24
Figura 24 - Proceso de despliegue de Resin.io (Fuente Resin.io) 25
Figura 25 - Panel de Resin.io (Fuente Resin.io) 25
Figura 26 - Logo Helium (Fuente Helium.com) 26
Figura 27 - Arquitectura de Helium.com (Fuente Helium.com) 27
Figura 28 - Logo Thinger.io (Fuente Thinger.io) 29
Figura 29 - Captura de pantalla Thinger.io (Fuente Thinger.io) 30
Figura 30 - Logo de TheThings.io (Fuente TheThings.io) 30
Figura 31 - Dashboard de TheThings.io (Fuente TheThings.io) 31
Figura 32 - Configuración de disparadores de TheThings.io (Fuente TheThings.io) 32
Figura 33 - Captura tablero Trello (Fuente propia) 35
Figura 34 - Arquitectura sistema controlador de sondas, actualizado con frontend (Fuente propia) 36
Figura 35 - Arquitectura interna del controlador (Iteración 1) (Fuente propia) 47
Figura 36 - Arquitectura básica de una aplicación Angular 2 (Fuente propia) 49
Figura 37 - Esquema base de datos (Iteración 1) (Fuente propia) 50
Figura 38 - Flujo del protocolo JWT (Fuente propia) 52
Figura 39 - Vista de login (Fuente propia) 53
Figura 40 - Plantilla de la interfaz (Fuente propia) 54
Figura 41 - Vista resumen del sistema (Fuente propia) 55
Figura 42 - Vista página de cada modelo (Fuente propia) 56
Figura 43 - Vista filtros activos (Fuente propia) 57
Figura 44 - Vista detalle y edición de modelo (Fuente propia) 58
Figura 45 - Vista nuevo elemento (Fuente propia) 59
Figura 46 - Flujo del middleware de autenticación (Fuente propia) 61
Figura 47 - NavbarComponent (Fuente propia) 79

Figura 48 - TopNavbarComponent (Fuente propia) 80
Figura 49 - DashboardComponent (Fuente propia) 80
Figura 50 - BigChartComponent (Fuente propia) 81
Figura 51 - DatatableComponent (Fuente propia) 81
Figura 52 - FiltersComponent (Fuente propia) 82
Figura 53 - FormComponent (Fuente propia) 83
Figura 54 - DetailViewComponent (Fuente propia) 84
Figura 55 - NewItemComponent (Fuente propia) 84
Figura 56 - Arquitectura cliente del módulo base (Fuente propia) 86
Figura 57 - ProbesComponent (Fuente propia) 87
Figura 58 - Vista mapa de sondas (Fuente propia) 87
Figura 59 - ProbeDetailViewComponent (Fuente propia) 88
Figura 60 - SensorTypesComponent (Fuente propia) 89
Figura 61 - Vista detalle de tipo de sensor (Fuente propia) 90
Figura 62 - SensorsComponent (Fuente propia) 91
Figura 63 - SensorDetailViewComponent (Fuente propia) 92
Figura 64 - WatcherLogsComponent (Fuente propia) 93
Figura 65 - LogsComponent (Fuente propia) 93
Figura 66 - Botón para copiar la clave de acceso al API desde la tabla (Fuente propia) 94
Figura 67 - Botón para regenerar la clave de acceso al API desde el formulario (Fuente propia) 94
Figura 68 - LoginModule (Fuente propia) 95

Índice de tablas

Tabla 1 - Requisitos funcionales (Fuente propia) 43

Tabla 2 - Requisitos de diseño (Fuente propia) 44

Tabla 3 - Requisitos no funcionales (Fuente propia) 45

Tabla 4 - Descripción tabla usuario (Fuente propia) 60

Tabla 5 - Tabla ejemplo coordenadas de sondas (Fuente propia) 64

Tabla 6 - Rutas iniciales (Fuente propia) 66

Tabla 7 - Rutas finales del servidor (Fuente propia) 78

1 Introducción

"But there's one more thing…"

Esta es la mítica frase que pronunciaba Steve Jobs al presentar nuevos productos de Apple en las conferencias de la compañía. En 1984, fue el Macintosh; en 2001, el iPod; en 2007, el iPhone; y en 2010, el iPad. Todos estos productos son bastante revolucionarios en cierta medida, pero hay uno que destaca sobre el resto, el que precisamente constituye una de las razones por las que la web creada hace ya más de 25 años por *Tim Berners-Lee* tiene muy poco que ver con la de hoy en día: el iPhone.

Han pasado 10 años ya desde el lanzamiento de este producto, que durante toda su vida ha protagonizado la batalla por llegar a los usuarios en la que ha competido contra Google, con Android, y Microsoft, aunque en menor medida, con Windows Phone. Es esta competición precisamente la que nos ha beneficiado a los usuarios y nos ha ofrecido grandes innovaciones y desarrollos que mejoran día a día la experiencia y el uso que hacemos de los teléfonos móviles.

En este sentido, es especialmente interesante destacar una parte de esta última frase: experiencia y uso. La clave del éxito del iPhone no es eliminar el teclado físico y dejar solo un botón. La clave del éxito consiste más bien en mejorar las interfaces de tal forma que sólo necesitemos un botón físico para usarlas, y esto es algo que Apple entendió muy bien. Lanzó un teléfono con una interfaz innovadora capaz de hacer que el usuario sintiera, desde la primera vez que cogiera el teléfono, que ya conocía el funcionamiento de los elementos en la pantalla.

Esta es la misma línea seguida en el proyecto del ámbito del Aprendizaje Basado en Proyectos (ABP) desarrollado por mi grupo durante el último año de carrera, proyecto que se ha basado en la creación de un producto en el que los usuarios pueden hacer intercambios de objetos o servicios. Dentro del grupo de trabajo todos hemos terminado el curso con la sensación de haber aprendido tecnologías, metodologías y paradigmas de programación que desconocíamos, pero entre los cinco ha quedado un buen sabor de boca, no por haber hecho una API funcional o por haber usado una u otra tecnología, sino por haber desarrollado un producto con una interfaz de usuario muy trabajada. Desde el momento inicial entendimos que es esencial crear interfaces y experiencias de usuario que hagan que este quiera volver a la aplicación y quedarse en ella, y en la que los flujos de movimiento de los usuarios dentro de la aplicación resulten lo más normales

posible. Por supuesto, esto implica que el equipo ha de dedicar mucho más tiempo a cuestiones como la posición de un botón o el color de un título, que pasan a ser elementos clave sobre los que discutir en las reuniones, pero también es evidente que hoy en día, y como han demostrado ya muchos antes que nosotros, cualquier sistema sin una interfaz gráfica agradable y cuidada no es un sistema completo.

Esto está relacionado con las carencias de Smart University señaladas anteriormente. Como hemos visto, Smart University es un proyecto que pretende replicar una ciudad inteligente dentro de la universidad. Teniendo en cuenta que el colectivo universitario puede superar los 40.000 usuarios, hay una cierta la similitud de la universidad con ciudades de tamaño pequeño o medio, por lo que podemos decir que este proyecto es una prueba o prototipo de la incursión de esta clase de sistemas en ciudades reales.

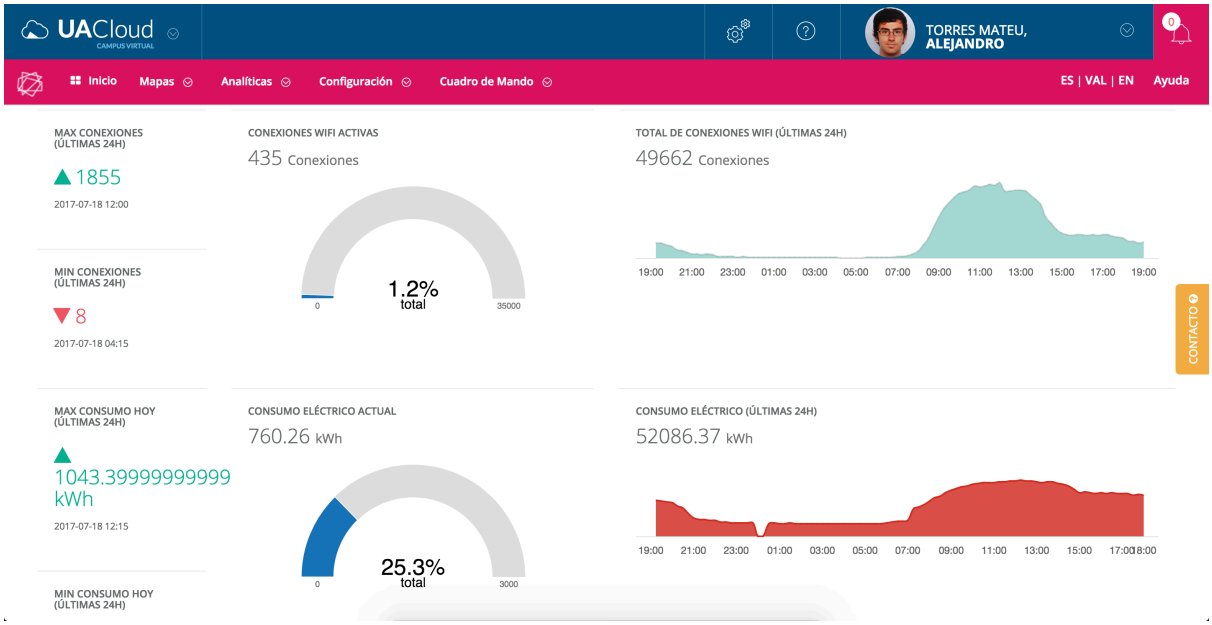


Figura 1 - Dashboard Smart University (Fuente Smart University Alicante)

El sistema de Smart University se compone de varios microservicios y cuenta con uno principal constituido por un *dashboard* (Figura 1) en el que se pueden visualizar y analizar datos y controlar el sistema. Los datos proceden de varias fuentes: por un lado, están los *smart meters*, que son datos recogidos de los sistemas propios de la universidad, como puede ser el consumo de energía procedente de los contadores de luz; por otro lado, está el sistema de sensorización, compuesto por un controlador central y una serie de sondas con unos determinados sensores que realizan lecturas y envían los datos al *dashboard*. El sistema de sensorización, se trata un sistema agnóstico de la gran cantidad de datos que recoge, capaz de asimilar un gran número de sensores y

acoplable a otros sistemas similares. Los sensores pueden ser muy variados, desde la presión atmosférica o la cantidad de luz en una habitación hasta la calidad del wifi en una determinada zona.

Mientras desarrollaba el sistema de sensorización observé que, aunque la herramienta era muy potente y eficiente, había sido concebida para funcionar como un servicio consumible por otro sistema pero no podía actuar por sí mismo, es decir, era dependiente de otra aplicación, la cual debe gestionar tanto las sondas como los sensores. Dada la necesidad hoy en día de crear aplicaciones completas, se hizo evidente que la aplicación necesitaba un frontal en el cual poder manejar el sistema en sí mismo, y recopilar datos propios de las sondas y los sensores, para así poder hacer una monitorización eficiente del sistema y eliminar esa dependencia con el sistema central de Smart University. Por lo tanto, este proyecto se centrará en el desarrollo de un panel de control para el sistema de sensorización y los cambios necesarios en el sistema original para hacer posible la inclusión de este panel sin modificar los requisitos ni servicios del sistema original. Además, también vamos a estudiar y proporcionar diversas optimizaciones en el sistema que permitan hacerlo más eficiente. El resultado del proyecto, si cumple con los objetivos y tiene cabida en el sistema inicial, quedará integrado dentro del sistema de Smart University como una aportación personal.

1.1 El problema

El proyecto Smart University tiene como herramienta central un *dashboard (Figura 1)* que permite representar la información que ha sido capturada desde diferentes fuentes de información.

A grandes rasgos, esta herramienta lo que permite es el descubrimiento de conocimiento mediante la representación de la información y la generación de servicios de valor añadido a partir del procesamiento de los datos. El proyecto tiene la opción de obtener información desde fuentes de datos como son el propio centro de proceso de datos de la UA, los contadores eléctricos digitales o incluso OPEN APIs, aunque recientemente se ha incorporado un sistema propio de generación de datos basados en sondas inteligentes y que formó parte de mi trabajo dentro de la beca en Smart University. La característica de este sistema es que está conformado por un módulo que recibe instrucciones desde el panel principal para configurar las sondas que proporcionarán datos y que se encarga de intermediar entre las sondas y el panel. Esta herramienta, aunque muy eficiente, tal como se ha dicho es un servicio a nivel de sistema que carece de una herramienta administrativa de uso sencillo. Tal como podemos ver en la Figura 2,

todas las operaciones y monitorización del sistema se deben realizar a partir de herramientas por línea de comandos, por lo que su manejo resulta poco usable y sus funcionalidades son muy limitadas.

```
mosquitto      | 1500398431: New connection from 172.26.0.5 on port 8883.
mosquitto      | 1500398431: New client connected from 172.26.0.5 as controller (c1, k60, u'smartuabroker').
probes-controller-api | MQTT: connected
probes-controller-api | info: WATCHER: Watcher fired
probes-controller-api | info: MQTT: subscribed to topic=smartUA/controller, qos=0
probes-controller-api | info: WATCHER: Lookforwatchers 2017-07-18 17:20:31
probes-controller-api | info: WATCHER: Going to fire: 2
probes-controller-api | info: WATCHER: all events fired correctly 2017-07-18 17:20:32
probes-controller-api | info: WATCHER: Calling recurse call in 30
probes-controller-api | info: WATCHER: Lookforwatchers 2017-07-18 17:21:03
probes-controller-api | info: WATCHER: Going to fire: 1
probes-controller-api | info: WATCHER: all events fired correctly 2017-07-18 17:21:03
probes-controller-api | info: WATCHER: Calling recurse call in 30
probes-controller-api | info: WATCHER: Lookforwatchers 2017-07-18 17:21:34
probes-controller-api | info: WATCHER: Going to fire: 1
probes-controller-api | info: WATCHER: all events fired correctly 2017-07-18 17:21:34
probes-controller-api | info: WATCHER: Calling recurse call in 30
probes-controller-api | info: WATCHER: Lookforwatchers 2017-07-18 17:22:05
probes-controller-api | info: WATCHER: Going to fire: 1
probes-controller-api | info: WATCHER: all events fired correctly 2017-07-18 17:22:05
probes-controller-api | info: WATCHER: Calling recurse call in 30
```

Figura 2 - Consola del servicio de sondas (Fuente propia)

Ahora mismo, en el estado actual del proyecto de sondas, la aplicación permite la gestión básica de sondas a través de una API, es decir, mediante comandos programados.

Pero ¿cómo podemos saber si una sonda está fallando o si no está conectada?, o si quizás un día necesitamos actualizar una sonda para añadir un nuevo sensor. Lo único que sabemos ahora mismo en el panel de control es que llegan una serie de datos cada x tiempo. Imaginemos que hay una sonda que se encarga de detectar fuego en una habitación concreta, pero esa sonda no está activa porque la conexión a internet ha fallado. Si se produjera un incendio, el sistema no lo sabría porque, al no conocer el estado de la sonda y al no recibir datos positivos de incendios, puede parecer que la sonda no está detectando ningún incendio y por consecuencia no se alertará a los distintos equipos de emergencia.

La problemática a la cual me enfrento en el proyecto es la necesidad por parte del equipo de Smart University de tener un panel de control con las diferentes sondas y sensores disponibles, para así poder controlarlos y administrarlos de forma precisa en cualquier momento y dispositivo, además de incorporar sistemas de alerta de fallo en sondas críticas, como la sonda para detección de incendios, por ejemplo.

Actualmente, con el controlador de sondas disponible, el equipo de Smart University puede hacer una gestión básica de las mismas, y puede darlas de alta o de baja, así como leer los datos recibidos. Mi planteamiento es focalizar el proyecto en este aspecto de dentro de la aplicación y desarrollar un panel totalmente funcional en el que poder gestionar el sistema de forma eficiente, e independientemente del dispositivo. Además, hacerlo lo suficientemente abstracto e independiente de la aplicación destino para que sea lo más portable y flexible posible. Por lo tanto, se tratará de un sistema que servirá no solo para el equipo de Smart University, sino para cualquier proyecto que quiera incorporar el sistema.

2 Antecedentes

En junio de 2017 entré a formar parte del equipo de Smart University en la Universidad de Alicante mediante unas prácticas becadas durante tres meses. Smart University lleva a cabo un ambicioso proyecto con el objetivo de crear ciudades inteligentes.

En el estado actual del proyecto, el sistema se basa en una serie de servicios y fuentes que recogen datos y los envían a un panel de control. Dentro de este panel de control se pueden analizar y obtener vistas detalladas y gráficas de los valores enviados por cada fuente de datos.

El proyecto se compone de distintas partes tal como se puede ver en la Figura 3.

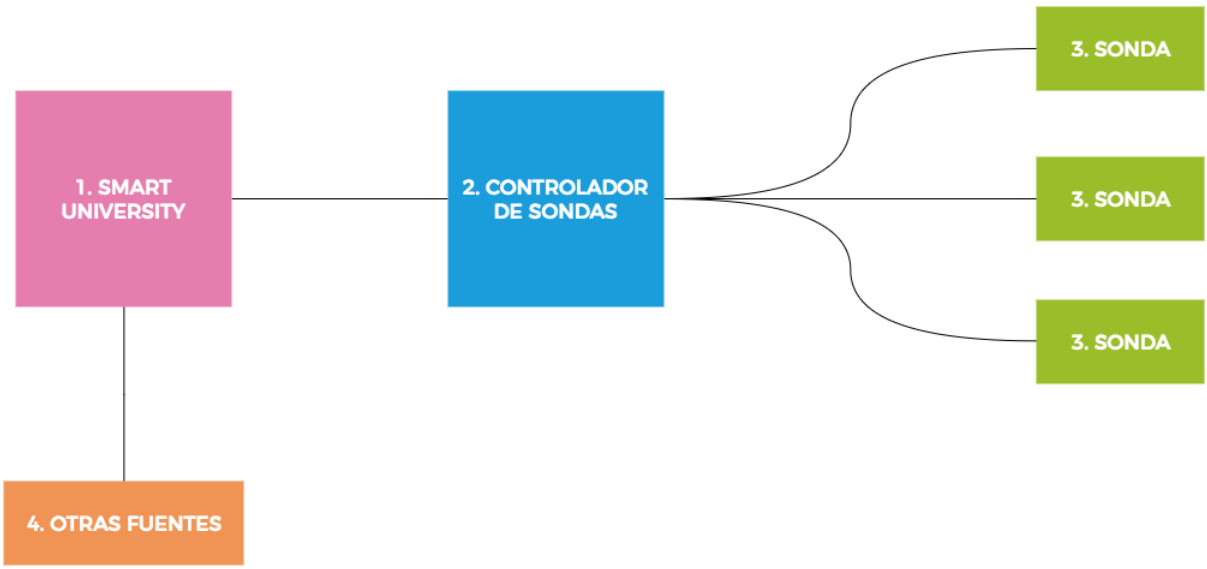


Figura 3 - Arquitectura Smart University actual (Fuente propia)

En la figura anterior se ha segmentado el proyecto a rasgos generales en 4 partes.

2.1 Smart University

Smart University es la parte central de la aplicación. En esta se hace el control y visualización tanto de las fuentes de datos como del propio sistema. Internamente está compuesto por un *backend* escrito en NodeJS y el *framework* ExpressJS, con persistencia en una base de datos no-sql mediante el motor MongoDb, y un cliente *frontend* bajo el *framework* AngularJS. Se hace uso del protocolo *REST* para la conexión entre *backend* y *frontend* mediante una *API*.

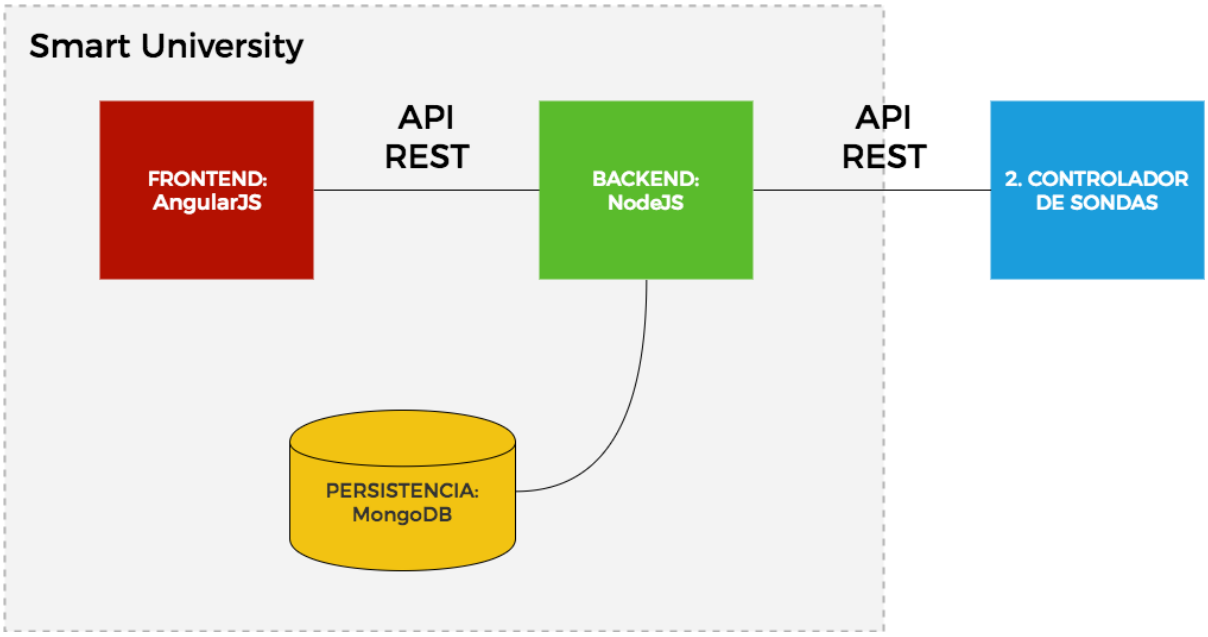


Figura 4 - Arquitectura interna Smart University actual (Fuente propia)

Además, cabe destacar que se trata de un proyecto incluido en el campus virtual de la Universidad de Alicante, por lo que cualquier persona con acceso al campus puede entrar en la aplicación.

Dentro del *dashboard* y en la parte superior tenemos un primer menú con distintas opciones que nos servirán de enlace con los distintos paneles de visualización y administración.

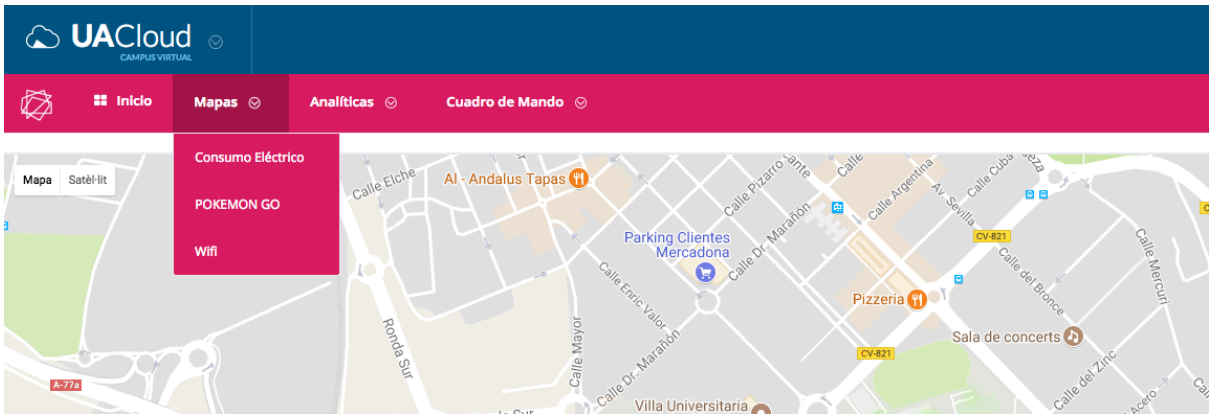


Figura 5 - Captura de pantalla Smart University: Inicio (Fuente Smart University Alicante)

En el primer menú Mapas podemos acceder a los distintos mapas en los que podemos ver representados datos procedentes de los distintos sensores.

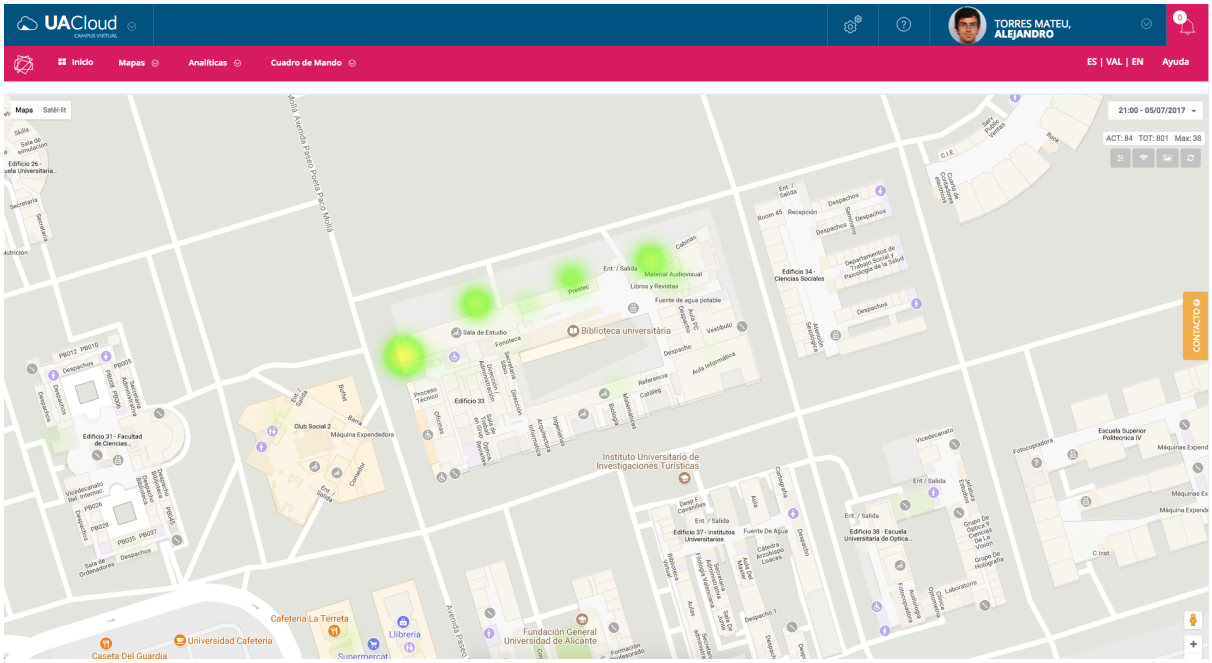


Figura 6 - Captura de pantalla Smart University: Mapas (Fuente Smart University Alicante)

Dentro del propio mapa tenemos una pequeña área de control para filtrar los datos que queremos ver representados.

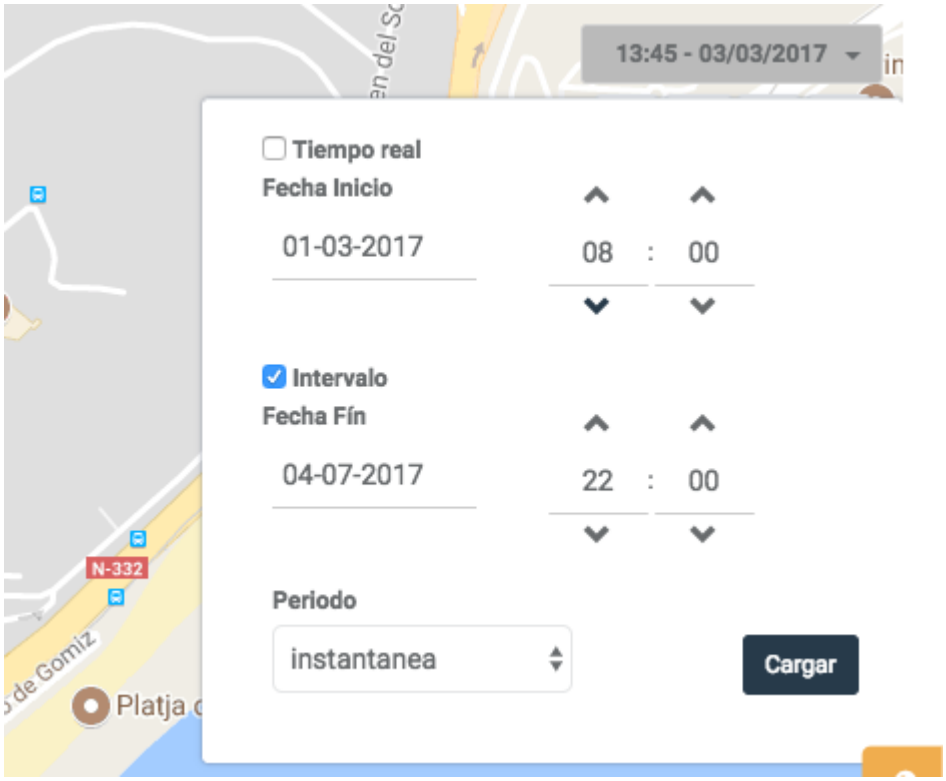


Figura 7 - Captura de pantalla Smart University: Mapas. Detalle (Fuente Smart University Alicante)

En el caso de los sensores de wifi, estos datos se ven reflejados como puntos de calor, de tal forma que podemos ver de un primer vistazo la calidad de la señal.

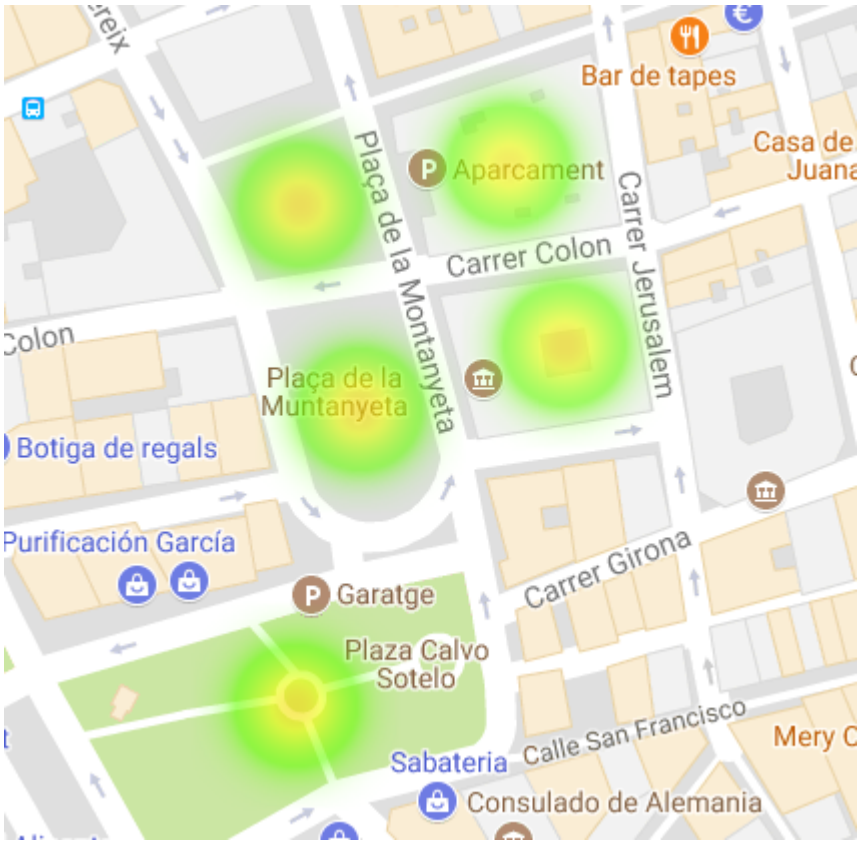


Figura 8- Captura de pantalla Smart University: Mapas. Detalle 2 (Fuente Smart University Alicante)

A continuación, en el segundo menú Analíticas tenemos una vista más detallada de los datos recogidos. Además, también disponemos de unos filtros similares a los de la vista de mapas, así como distintos formatos de gráficas para representar la información.

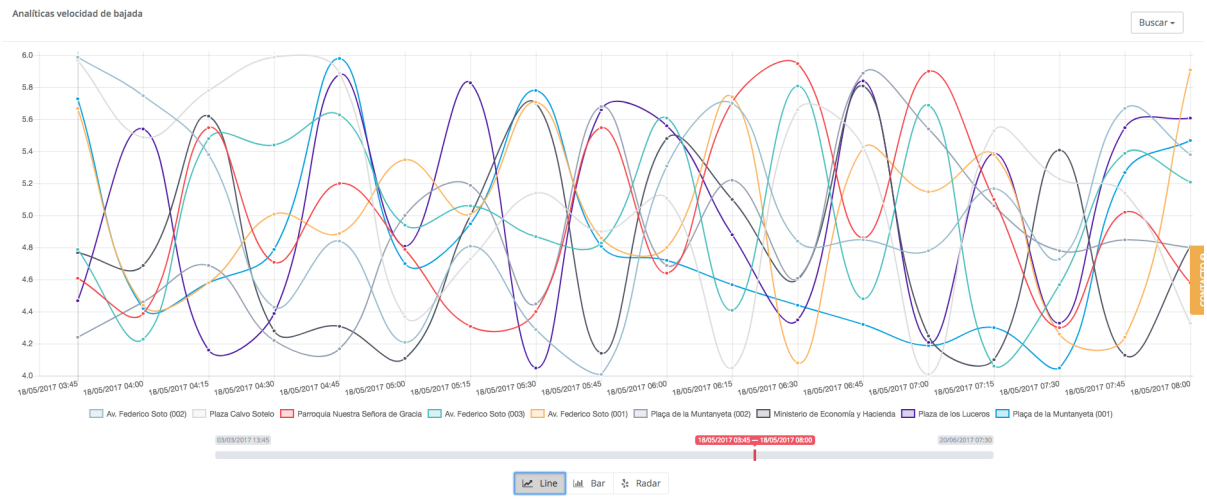


Figura 9 - Captura de pantalla Smart University: Analíticas (Fuente Smart University Alicante)

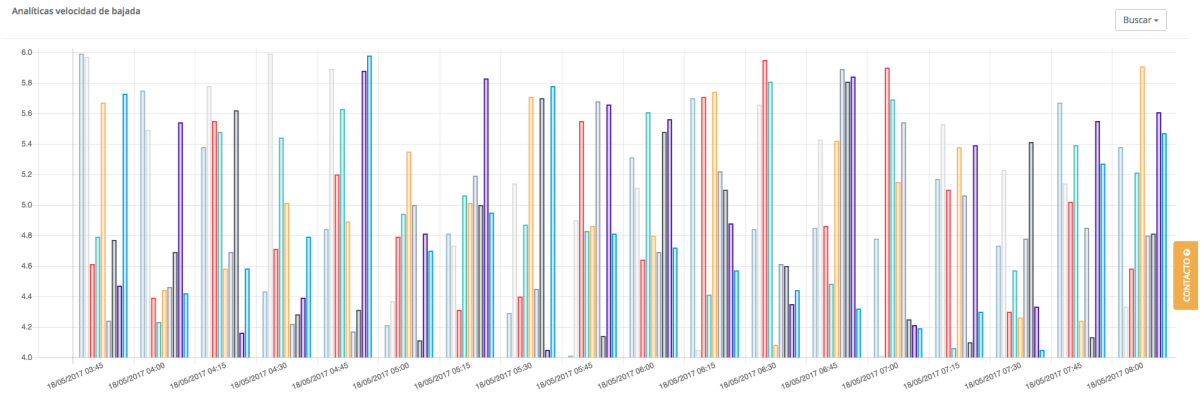


Figura 10 - Captura de pantalla Smart University: Analíticas 2 (Fuente Smart University Alicante)

Finalmente, dentro del menú Configuración podemos configurar el panel en sí, administrar las fuentes de datos y realizar tareas de administración del sistema, tales como la administración de usuarios o los roles de los mismos.

Me gustaría destacar las secciones Dispositivos, Tipo de Dispositivos y Configurador de Sensores, del menú Configuración. Como podemos observar en el esquema de la arquitectura de Smart University en la Figura 4, vemos que entre el backend y el controlador de sondas hay una conexión; esta conexión a su vez se ve reflejada en el *dashboard* en estas secciones, ya que mediante ellas podemos registrar y configurar las sondas y sensores disponibles en el controlador de sondas.

Figura 11 - Captura de pantalla Smart University: Menu Configuración (Fuente Smart University Alicante)

2.2 Controlador de sondas

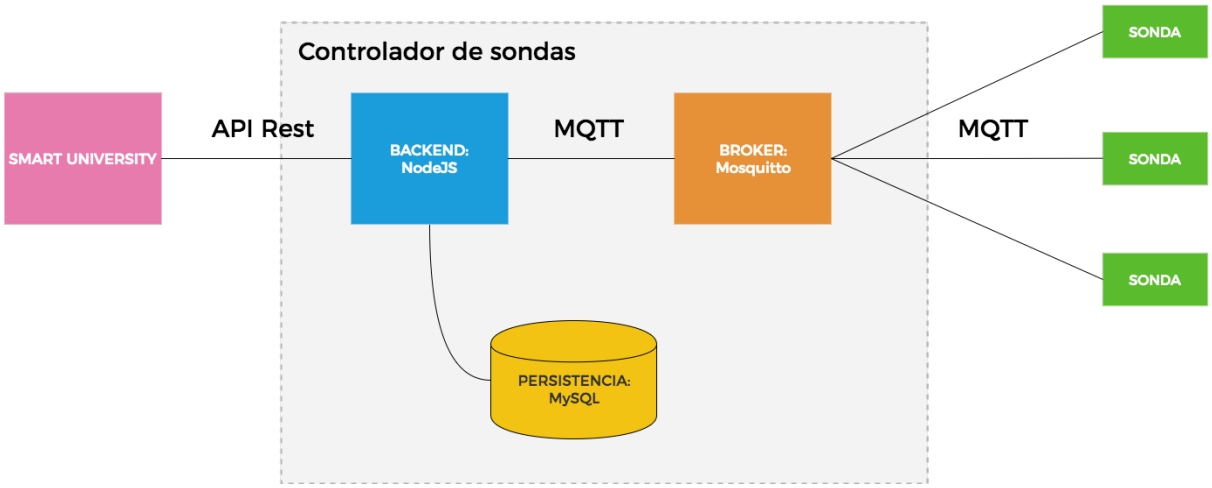


Figura 12 - Arquitectura sistema controlador de sondas (Fuente propia)

Como hemos comentado anteriormente, la parte *backend* de Smart University se conecta con el controlador de sondas. El controlador de sondas es la parte del proyecto que se encarga de gestionar íntegramente la conexión con los dispositivos *IoT*.

Como podemos ver en la arquitectura se trata de un *backend* escrito en NodeJS, Mosquitto[1] como el *broker* MQTT, y persistencia en base de datos MySql. Se conecta al backend de Smart University mediante una API REST y a las distintas sondas usando MQTT a través del *broker*.

2.2.1 MQTT

Antes de continuar con la arquitectura y la funcionalidad del controlador de sondas comentaremos las características del protocolo MQTT.

MQTT (*Message Queue Telemetry Transport)* es un protocolo muy extendido en la comunicación máquina-a-máquina, y sobre todo en el ámbito del internet de las cosas. Esto es debido a que es muy ligero, consume muy poco ancho de banda, es compatible con la gran mayoría de dispositivos con pocos recursos físicos, tiene implementación en una gran variedad de lenguajes informáticos y es altamente escalable (IBM 2017; Light s.f.).

[1] https://mosquitto.org/

La arquitectura del protocolo se representa como una topología de estrella, y es el nodo central del servidor o, siguiendo la convención terminológica de MQTT, el *broker* (Yébenes Gálvez 2016). El resto de nodos se conectan al *broker* siguiendo un sistema de publicación de mensajes y suscripción a canales (*topcis*) de interés. El *broker* es el encargado de recibir los mensajes, gestionar las colas y canales, y enviar los mensajes a los canales destino.

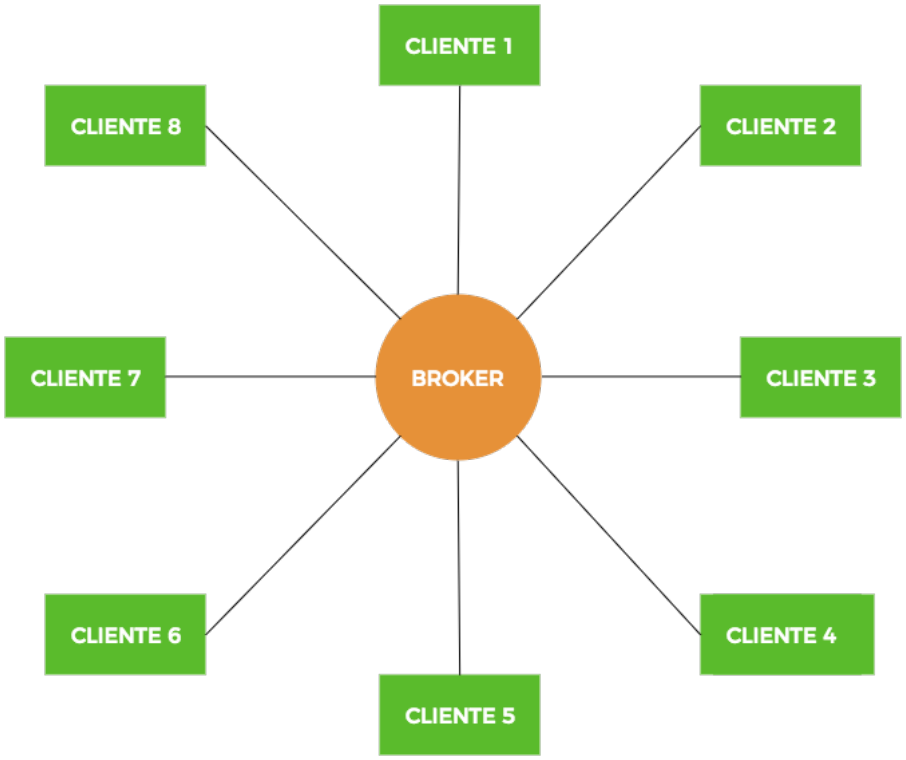


Figura 13 - Arquitectura MQTT (Fuente propia)

La arquitectura es muy similar a Twitter: los usuarios siguen a otros usuarios (canales) y cuando un usuario (publicador) publica un nuevo mensaje en su muro (canal), este mensaje se envía a todos los usuarios que le siguen (suscriptores).

Otro de los motivos que hacen de MQTT un buen protocolo para la comunicación entre dispositivos IoT es su gestión de los canales. Los canales en MQTT se organizan de forma jerárquica por niveles, es decir, podemos tener el canal `/UA/EPS1/PLANTA1/SONDA3/HUMEDAD`, que, según la convención seguida dentro del proyecto, hace referencia al sensor de humedad, de la sonda número tres, de la primera planta de la Escuela Politécnica Superior uno. Y si por otro lado tenemos `/UA/EPS1/PLANTA1/SONDA2/HUMEDAD`, siguiendo la misma convención, si quisiéramos obtener el valor de la humedad en la primera planta, podríamos enviar un mensaje a

ambas sondas usando el canal: `/UA/EPS1/PLANTA1/+/HUMEDAD`, es decir, con el uso de comodines, con lo que podemos hacer llamadas jerárquicas descendentes evitando especificar niveles en la jerarquía.

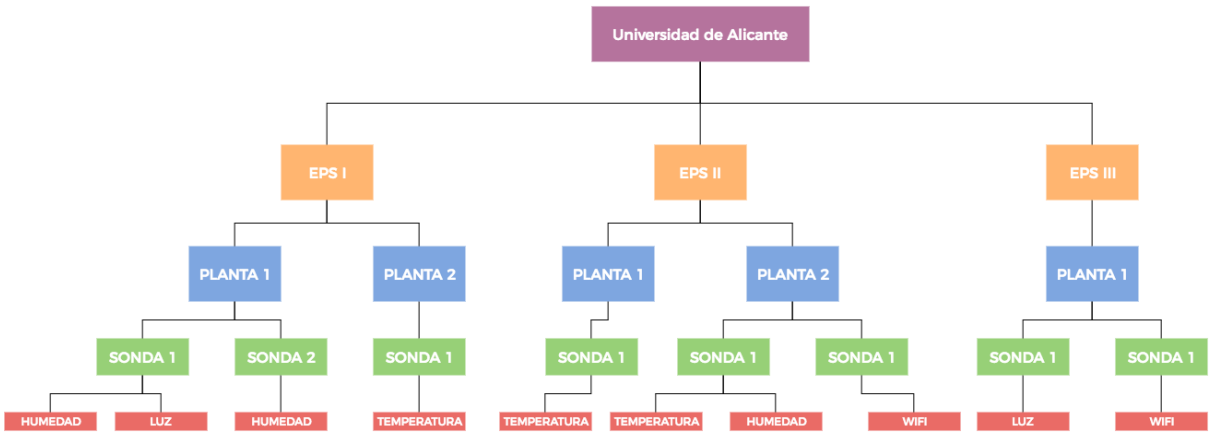


Figura 14 - Jerarquía MQTT (Fuente propia)

Finalmente, mencionar también la posibilidad de usar diferentes niveles de calidad de servicio (QoS). El publicador del mensaje puede definir 3 niveles de calidad de servicio en el momento de enviar un mensaje de tal forma que:

- QoS nivel 0, *at most once*: El mensaje se entregará como máximo una vez, es decir, no hay garantías de recepción.
- QoS nivel 1, *at least once*: El mensaje se entregará al menos una vez. El cliente emitirá el mensaje tantas veces como sea necesario hasta obtener una respuesta del *broker* que informará de que el mensaje ha sido publicado.
- QoS nivel 2, *exactly once*: La finalidad es la misma que con QoS 1, pero en este caso el cliente se asegura que solo ha enviado un mensaje igual, evitando crear duplicados.

Elegir un nivel u otro depende de las necesidades del proyecto, siendo QoS 0 el más barato en cuanto a recursos y QoS 2 el más caro. En el proyecto Smart University se ha optado por usar QoS 0, ya que se permite un margen de pérdidas en la lectura de los sensores, y beneficia la velocidad del sistema.

2.2.2 Persistencia en base de datos: MySQL

En cuanto a la persistencia de datos, el elegido para el sistema es un modelo relacional, usando el extendido sistema de gestión de base de datos MySQL.

El sistema de Smart University usa una base de datos no relacional con el gestor MongoDB, pero tras el análisis de los requisitos del controlador de sondas se observó que una base de datos relacional encajaba mucho mejor en este caso. En la Figura 15 podemos la representación del esquema de la base de datos.

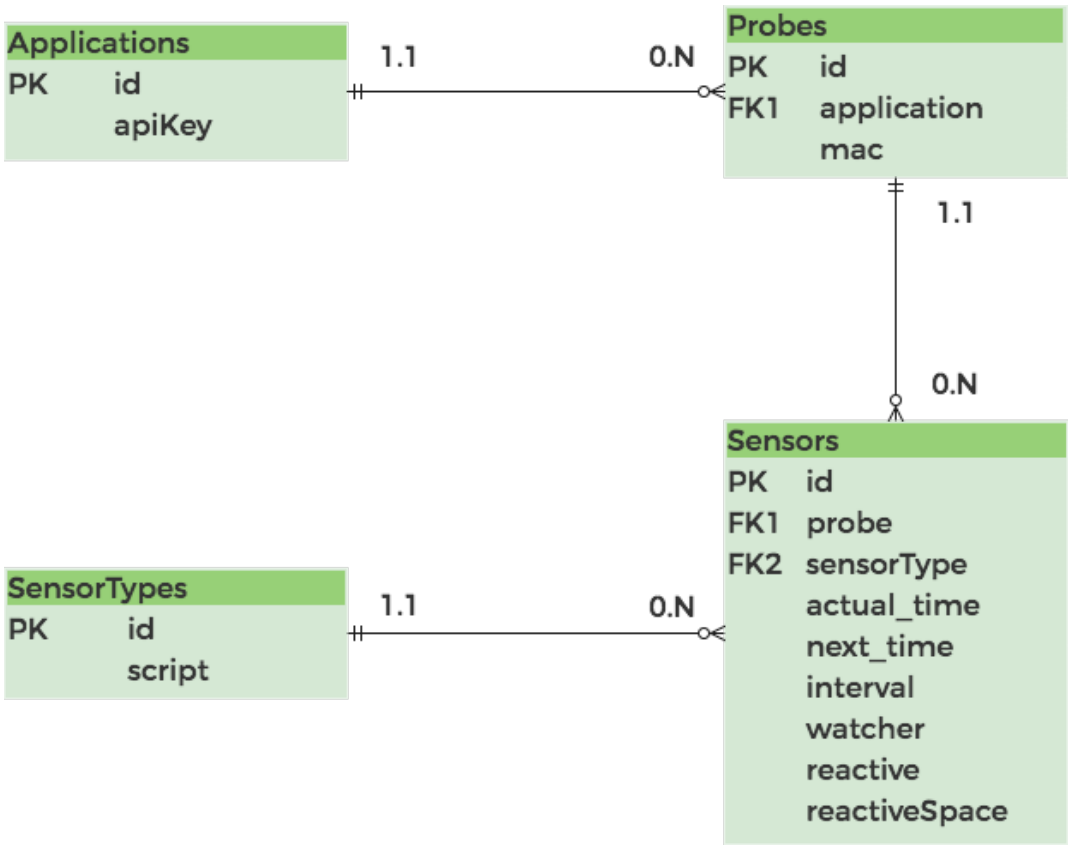


Figura 15 - Esquema base de datos inicial (Fuente propia)

2.2.3 Backend: Controlador

El *backend,* a partir de ahora controlador, se caracteriza por ser el corazón del sistema. Tal como hemos visto en la Figura 13, MQTT tiene topología de estrella, y a bajo nivel, el controlador actúa como un cliente más, pero a un nivel más alto, el controlador es realmente el centro de la estrella y el *broker* es un servicio de este, que le sirve para comunicarse con los clientes mediante el protocolo MQTT.

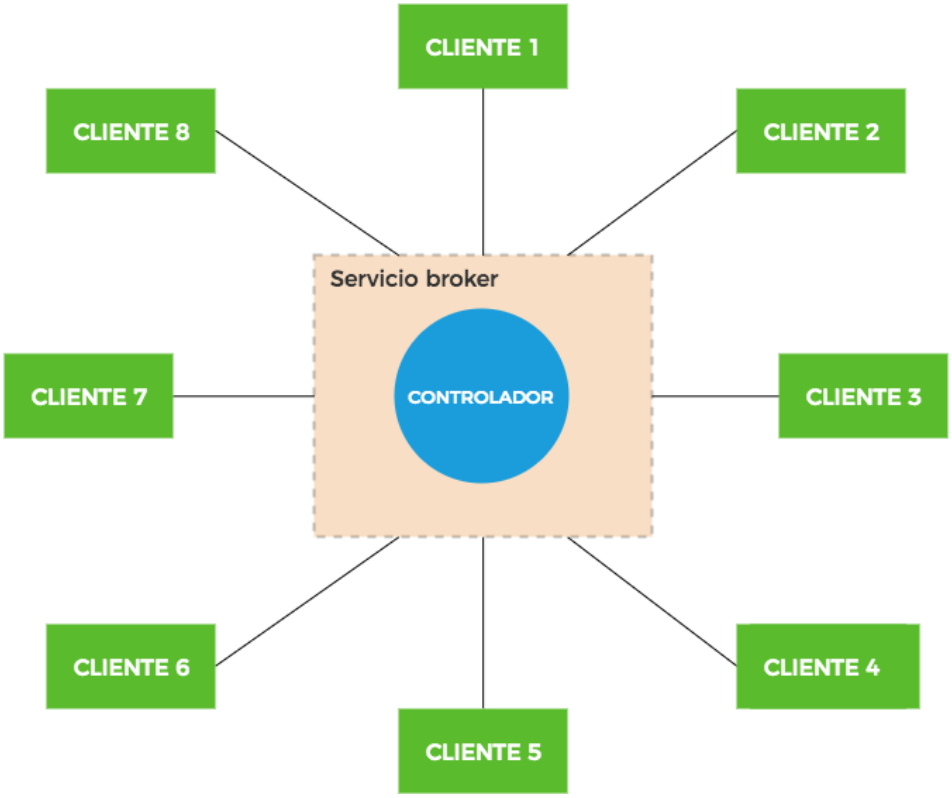


Figura 16 - Topología controlador de sondas con servicio MQTT (Fuente propia)

El controlador está compuesto por diferentes servicios, como podemos ver en la Figura 17. Por un lado, se encuentra la API REST, mediante la cual los servicios externos pueden consumir, crear, modificar y eliminar los recursos de la aplicación. También se dispone de un servicio que permite el mapeado y la construcción de consultas SQL, de tal forma que las filas que obtenemos de la base de datos tienen un formato legible nativamente por JavaScript. Por otra parte, existe un servicio que se encarga de la conexión del sistema con el *broker* MQTT que crea, en cierta medida, un nivel de independencia entre la aplicación y el protocolo para la conexión con los dispositivos.

Finalmente, y como la parte más valiosa del controlador, se encuentra el servicio de *watchers* o vigilantes. Como podemos ver en la Figura 15, donde se representa el esquema de la base de datos, cada dispositivo (*probe*) tiene una serie de sensores asociados. La manera de recoger datos de los dispositivos es enviándoles peticiones de lectura de cada sensor cada x tiempo. El servicio de *watchers* se encarga precisamente de hacer esto, como si de un *Cron* de Unix se tratara. Una vez que el controlador ha recogido los datos de los sensores, este los envía al *dashboard* de Smart University a través de una API REST, ignorando totalmente el contenido de los datos recibidos.

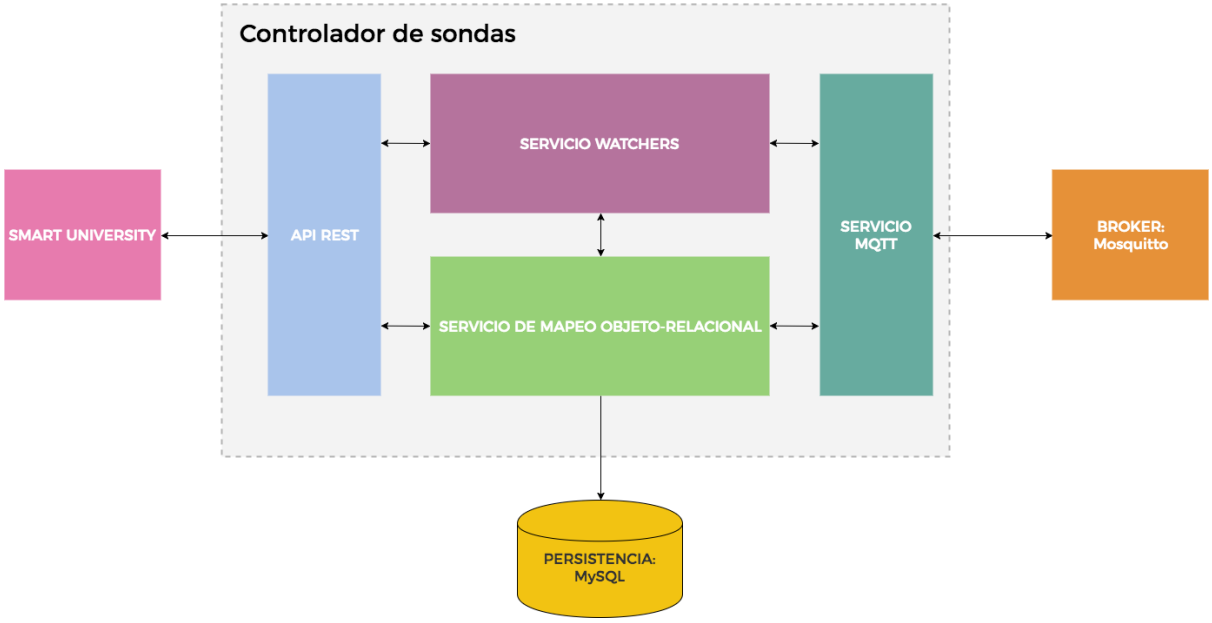


Figura 17 - Arquitectura interna del controlador de sondas (Fuente propia)

2.3 Dispositivos: Sondas

La segunda parte del sistema son las sondas. Las sondas no son más que un dispositivo conectado a internet capaz de recoger datos de sensores integrados o intercambiables. Me gustaría hacer hincapié en este detalle ya que resume la versatilidad del sistema y MQTT, ya que hoy en día las sondas son microcomputadoras Raspberry Pi con una serie de sensores conectados a los puertos GPIO de la máquina, pero en un futuro el dispositivo podrá ser desde una placa totalmente diseñada ad-hoc para el sistema, un ordenador personal o hasta un teléfono móvil.

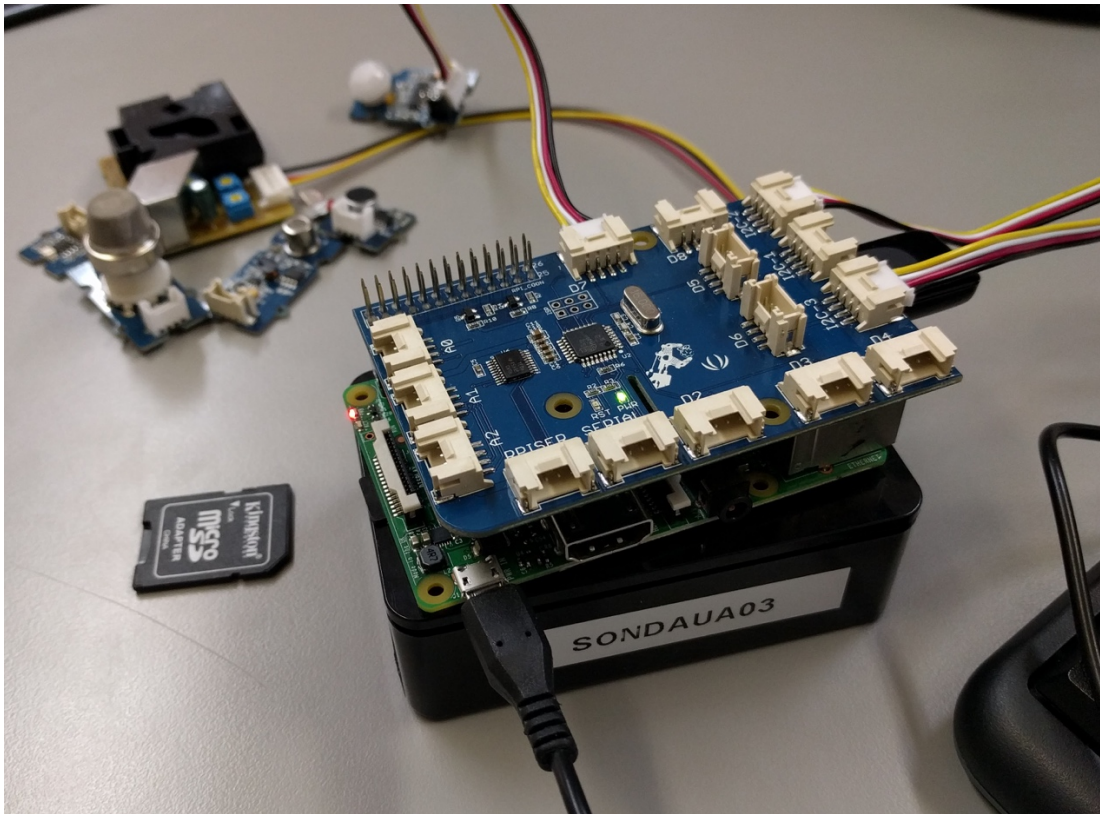


Figura 18 – Sonda: Raspberry Pi, Grove Pi y sensores (Fuente propia)

Es por eso que desde un primer momento se decidió hacer el software de las sondas lo más pasivo y ligero posible, de manera que en un momento dado, y si se añaden nuevos dispositivos, no haya que actualizar *in situ* el software de las sondas. Uno de los ejemplos más claros lo vemos a la hora de añadir sensores: en un modelo tradicional, lo primero que podríamos pensar es que la sonda es la que contiene los sensores y por eso debe conocer los procedimientos necesarios para acceder a la lectura de los mismos. En este caso, se decidió volcar esa lógica en la parte del controlador, es decir, es el controlador el que sabe cómo realizar la lectura de cada tipo de sensor y es el que le pasa ese procedimiento a la sonda.

Los pasos que sigue la sonda para configurarse son los siguientes:

1. La sonda se conecta con el controlador y pide su configuración.
2. El controlador envía la configuración a la sonda.
3. La sonda se configura y guarda los scripts necesarios para leer los sensores.
4. La sonda se queda a la espera de recibir eventos del controlador sobre qué hacer.

Figura 19 - Flujo Sonda – Controlador (Fuente propia)

2.4 Otras fuentes: Smart Meters

Finalmente, a parte de la información recibida por los sensores, el *dashobard* también se nutre de otras fuentes. Estas, a diferencia de las sondas, son particulares de la Universidad de Alicante, entre las cuales se encuentran los contadores de luz electrónicos y los puntos de acceso a internet repartidos por la universidad.

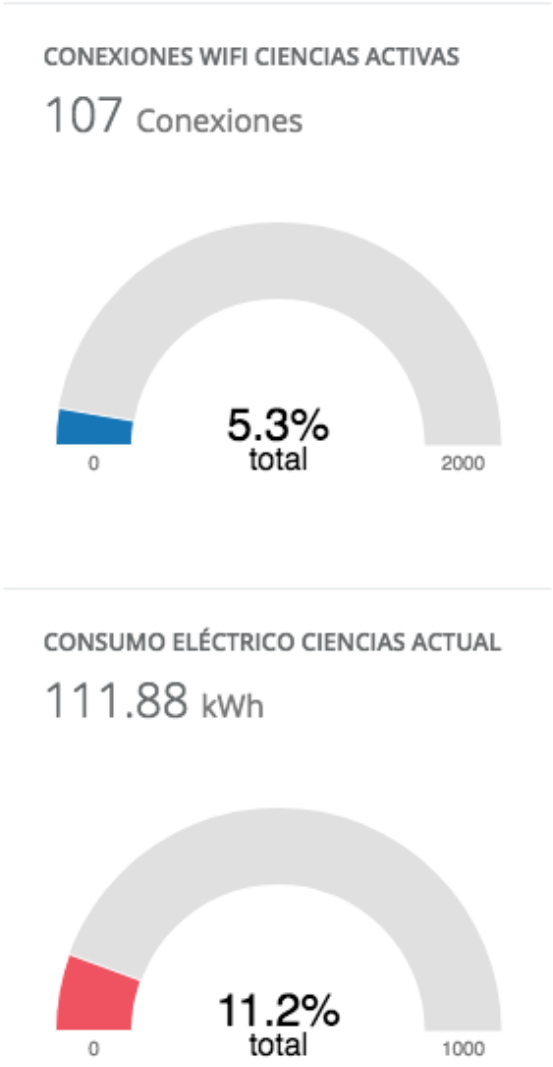


Figura 20 - Gráficas de puntos de acceso inalámbricos y consumo eléctrico (Fuente Smart University Alicante)

3 Estado del arte

Existen diversas plataformas capaces de obtener datos de sensores y crear paneles de administración de forma genérica. Haciendo un análisis de las herramientas más extendidas he podido observar que las alternativas existentes no son suficientes para abarcar los objetivos propuestos.

Por un lado, tenemos un primer segmento de aplicaciones capaces de, mediante su propio *software, hardware* o el uso del protocolo de la plataforma, recogen datos que posteriormente interpretan y analizan, y de los que generan gráficas y paneles. Por otro lado, existe un segundo segmento de aplicaciones encargadas solo de una parte del proceso, bien sea la representación o análisis de los datos, o bien la configuración del *hardware, software* o el uso de un protocolo propio.

Sin embargo, la característica que predomina en ambos segmentos es que ninguno de ellos se centra en la monitorización de los dispositivos y el sistema. Así pues, como ya hemos comentado anteriormente, este es precisamente el interés del proyecto, y no el análisis de los datos recolectados, los cuales el sistema recogerá y enviará al panel principal de Smart University sin hacer ningún tipo de procesado sobre los mismos.

En esta sección analizaremos algunas de las herramientas más populares y sus interfaces, ya que, a pesar de no tener el mismo objetivo, puede ser útil para definir nuevas funcionalidades y contemplar otras ideas que pueden servir para el producto final.

3.1 Soluciones parciales

Con soluciones parciales, nos referimos a aquellas plataformas que se encargan solo de una parte de la arquitectura, bien sea la configuración de dispositivos o la visualización de los datos recibidos de los sensores de los mismos.

3.1.1 Freeboard.io

Figura 21 - Logo de Freeboard.io (Fuente Freeboard.io)

La traducción de la descripción del sitio es "Visualiza el Internet de las Cosas", pero podríamos definir Freeboard.io[2] como un panel genérico para cualquier tipo de dato, no solo de dispositivos IoT. De hecho, el mecanismo de Freeboard.io es realmente sencillo: nada más entrar en la aplicación, nos encontramos con un espacio vacío. En este espacio podemos añadir tarjetas en las que visualizar datos. La parte más interesante de la plataforma es que la fuente de datos puede ser cualquier API REST existente, ya que Freeboard.io solo se encarga de realizar peticiones a esa API REST cada x tiempo y mostrarlas en tarjetas que pueden ser gráficas de distintos tipos: texto, mapas, etc. Además, el usuario puede usar JavaScript para convertir los datos procedentes de la API REST a un formato entendible por la aplicación.

Por ejemplo, en el laboratorio del DTIC hay una sonda conectada al sistema controlador de sondas, con un sensor de luz y otro de temperatura. Además, también disponemos de la posición GPS de la sonda y la URL de la API REST para visualizar los datos que recoge. Así que, con una simple investigación hemos podido conseguir un panel como el mostrado en la Figura 22.

[2] http://freeboard.io/

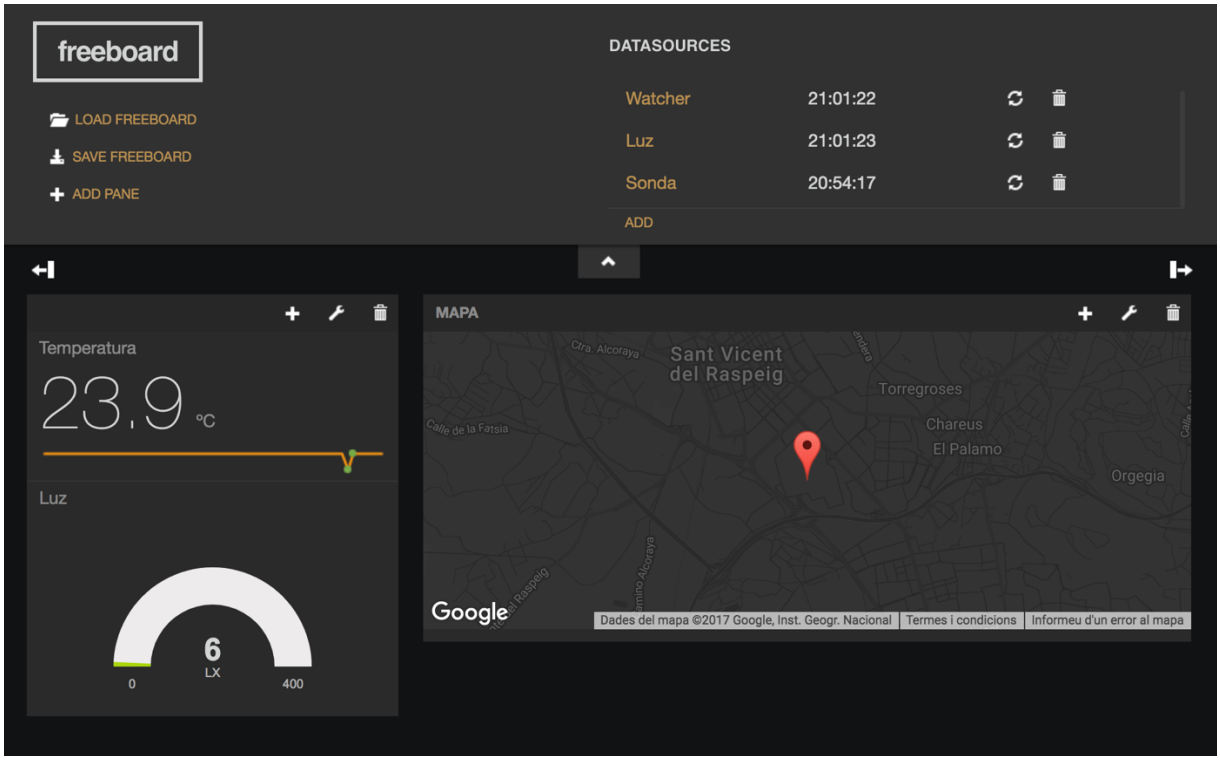


Figura 22 - Captura de panel creado con Freeboard.io (Fuente Freeboard.io)

Aunque el uso es muy sencillo y el sistema es bastante versátil, no es lo suficientemente completo, ya que solo se pueden definir tres tipos de gráficas y no existe ningún tipo de control sobre los dispositivos implicados.

3.1.2 Resin.io

Figura 23 - Logo Resin.io (Fuente Resin.io)

Resin.io[3] es una plataforma para desplegar software de forma remota para dispositivos IoT. No hace una interpretación ni procesado de los datos, ni tampoco hace una monitorización de los dispositivos en sí, pero es interesante incluirlo en el análisis, ya que, el controlador de sondas original tiene una característica que puede ser comparable al proceso de despliegue que realiza Resin.io.

En el proceso de inicialización de la sonda en el sistema de Smart University, esta actualiza su software de una forma muy simple. El código fuente del software de las sondas está alojado en un repositorio Git online, de tal forma que cuando se añade una funcionalidad al software de la sonda, este se actualiza también en el repositorio. Las sondas, antes de inicializar el programa, comprueban si existen actualizaciones de software en el repositorio; si es así, descargan la última versión y la lanzan. Este proceso también se puede realizar a distancia de forma manual mediante un comando lanzando por el controlador.

Resin.io funciona de una forma similar, pero el sistema final resulta más complejo. El código fuente del software de los dispositivos también se aloja en un repositorio Git online, así que cuando se desarrollan nuevas funcionalidades este se sube al repositorio. El repositorio está configurado con una serie de disparadores, los cuales, cuando detectan que hay un cambio en el código base construyen una imagen haciendo uso de Docker y notifican al dispositivo para que descargue la imagen más reciente. El dispositivo debe de tener instalado el sistema operativo de Resin.io para poder interactuar con el sistema.

[3] https://resin.io/

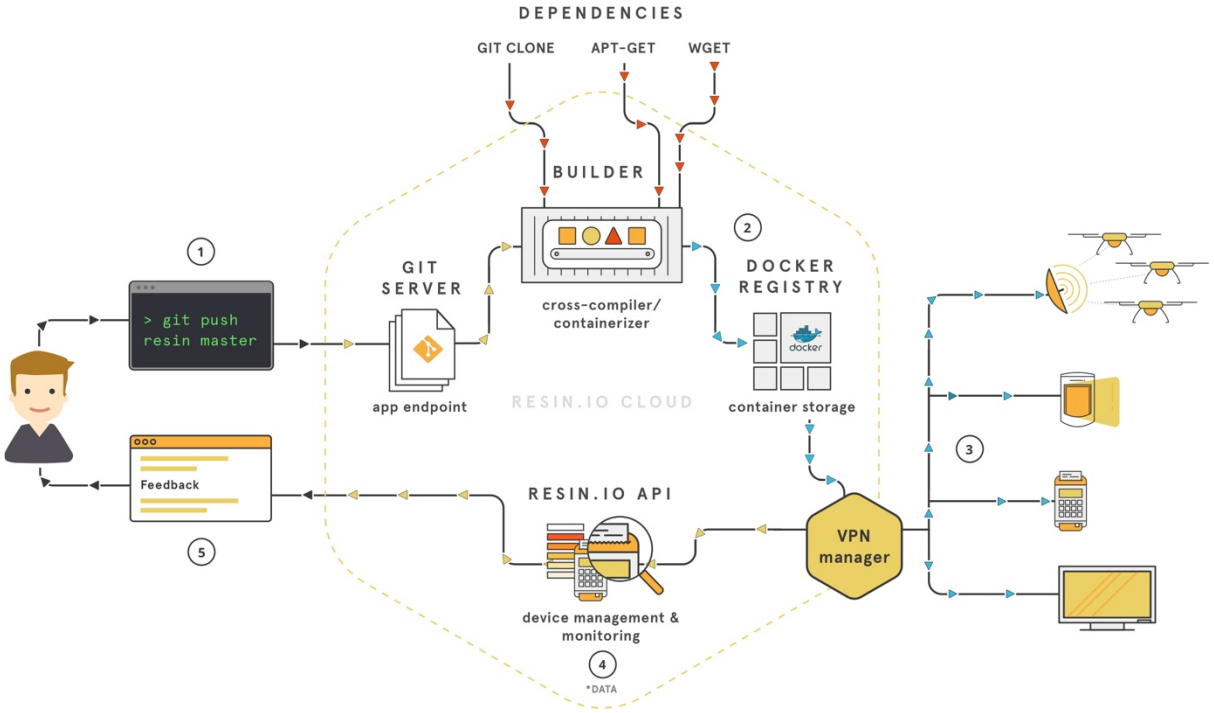


Figura 24 - Proceso de despliegue de Resin.io (Fuente Resin.io)

Como podemos ver, el proceso es bastante similar, pero se introduce un paso intermedio que es la creación de la imagen en Docker. Más allá del sistema de despliegue y la monitorización del proceso, Resin.io no aporta información adicional sobre el dispositivo en sí o los datos recogidos por el mismo.

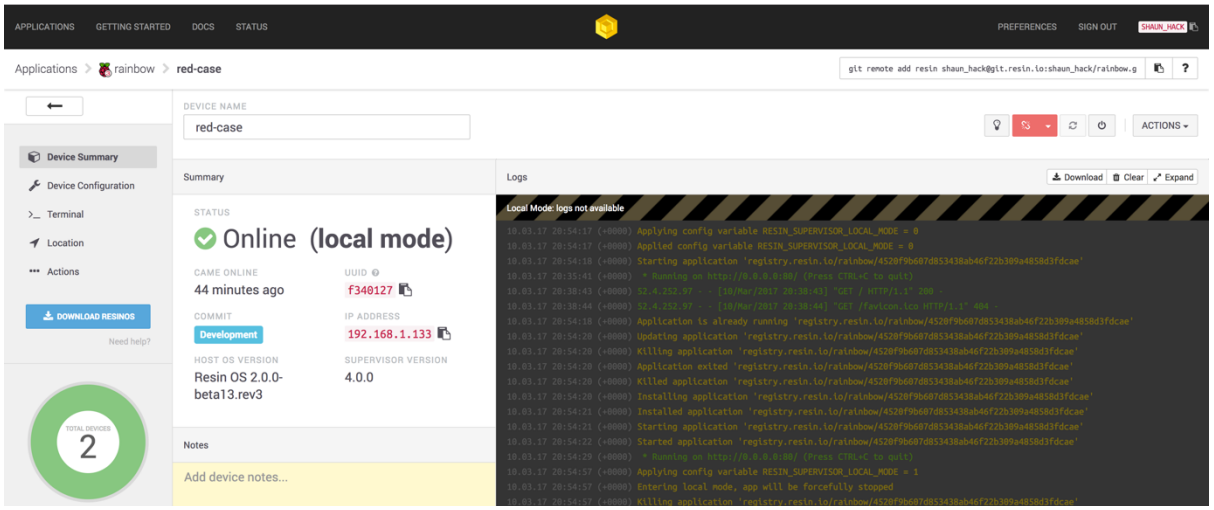


Figura 25 - Panel de Resin.io (Fuente Resin.io)

3.1.3 Helium

Figura 26 - Logo Helium (Fuente Helium.com)

Helium cubre el paso existente entre la recolección de datos por parte del dispositivo y la herramienta que procesará los datos. Es posiblemente la herramienta más parecida a la que se pretende desarrollar con este proyecto.

La arquitectura hardware se basa en dos dispositivos que funcionan en tándem. Por un lado, están los *Elements* (elementos), que son *routers* o puntos de acceso en los que se crean redes privadas conectadas a Internet bien por un cable ethernet o bien por red móvil 4G. A estos *Elements* se conectan los *Atoms* (átomos), que son los dispositivos que recogen los datos en última instancia. Los *Atoms* se componen de dos partes: por una parte, está el módulo que permite la conexión con el *Element*, que se trata de una placa XBee y por otra parte el dispositivo al que se conecta que puede ser una Raspberry Pi, un Arduino o cualquier dispositivo con un puerto USB y sistema operativo Linux (Helium s.f. (a)).

Figura 27 - Arquitectura de Helium.com (Fuente Helium.com)

En cuanto a la arquitectura software, disponen de clientes para cada una de las plataformas nombradas anteriormente para permitir la conexión con su sistema. Helium hace de intermediario entre el dispositivo y la aplicación final, por lo que desde la plataforma se debe configurar uno o más *Channels* (canales) donde enviar los datos. Estos, por lo general, son sistemas de computación en la nube como Amazon, Microsoft Azure, Google o directamente un *broker* MQTT. La entrega de valor que hace Helium reside en la seguridad a nivel de hardware y software que implementa el sistema y que hace que la conexión entre las dos partes implicadas en el proceso de recogida de datos sea lo más segura posible. Esto lo consiguen mediante una clave física que tienen los *Atoms*, que permiten solo a los dispositivos validados por el sistema conectar con los *Elements*.

Dentro del *dashboard* de la plataforma podemos ver, añadir, eliminar y actualizar *Elements, Atoms* y *Channels*. También podemos ver detalles de los *Elements y Atoms*, como el estado de la conexión y los últimos eventos recibidos, aunque no podemos hacer una monitorización de los dispositivos e interactuar con ellos (Helium s.f. (b)).

En el momento de escribir este trabajo, el sistema de Helium solo está disponible en los Estados Unidos, aunque tienen a Europa en la hoja de ruta.

3.2 Soluciones completas

Describiré aquellos sistemas que también se encargan de administrar los dispositivos desde la parte *hardware* o *software*, y que no solo se ven involucrados en una parte del proceso.

3.2.1 Thinger.io

Figura 28 - Logo Thinger.io (Fuente Thinger.io)

Thinger.io[4] es una plataforma de código abierto desarrollada por el madrileño Álvaro Luís Bustamante. Esta plataforma se caracteriza por tener librerías de código abierto para los dispositivos más populares del mercado: Arduino, Raspberry Pi, Sigfox, ARM mBed, así como su propia placa ClimaStick. La idea de las librerías es conectar y enviar datos desde los dispositivos al panel de administración de forma sencilla.

Una vez configurado el dispositivo, podemos ver desde el panel de administración el estado del mismo, así como los datos que está enviando. En el dispositivo se debe configurar las credenciales del usuario para poder enviar datos al panel.

Una vez configurado un dispositivo, podemos definir una serie de URL mediante las cuales podemos enviar datos de los sensores que se añadirán automáticamente al panel visual.

Esta solución es útil si se quiere recolectar y visualizar datos en un panel de administración, pero no dispone de suficientes opciones como para monitorizar de forma correcta los dispositivos o el sistema.

[4] https://thinger.io/

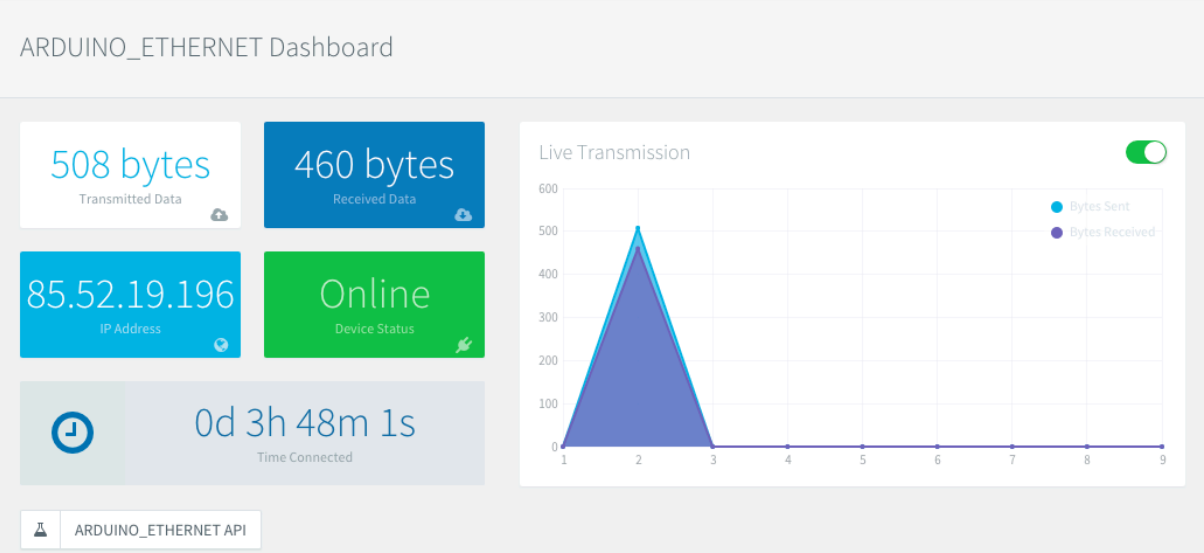


Figura 29 - Captura de pantalla Thinger.io (Fuente Thinger.io)

3.2.2 TheThings.io

Figura 30 - Logo de TheThings.io (Fuente TheThings.io)

TheThings.io[5] es una empresa barcelonesa que sigue la misma filosofía que Thinger.io, pero se diferencia en la tecnología. Mientras que Thinger.io desarrollaba software pensando en el hardware, TheThings.io desarrolla el software centrado en el lenguaje de programación (TheThings.io s.f.). Al final, el resultado es bastante similar: se usa una librería para conectar con un panel de administración donde mostrar los datos provenientes de los sensores o dispositivos asociados al usuario.

[5] https://thethings.io/

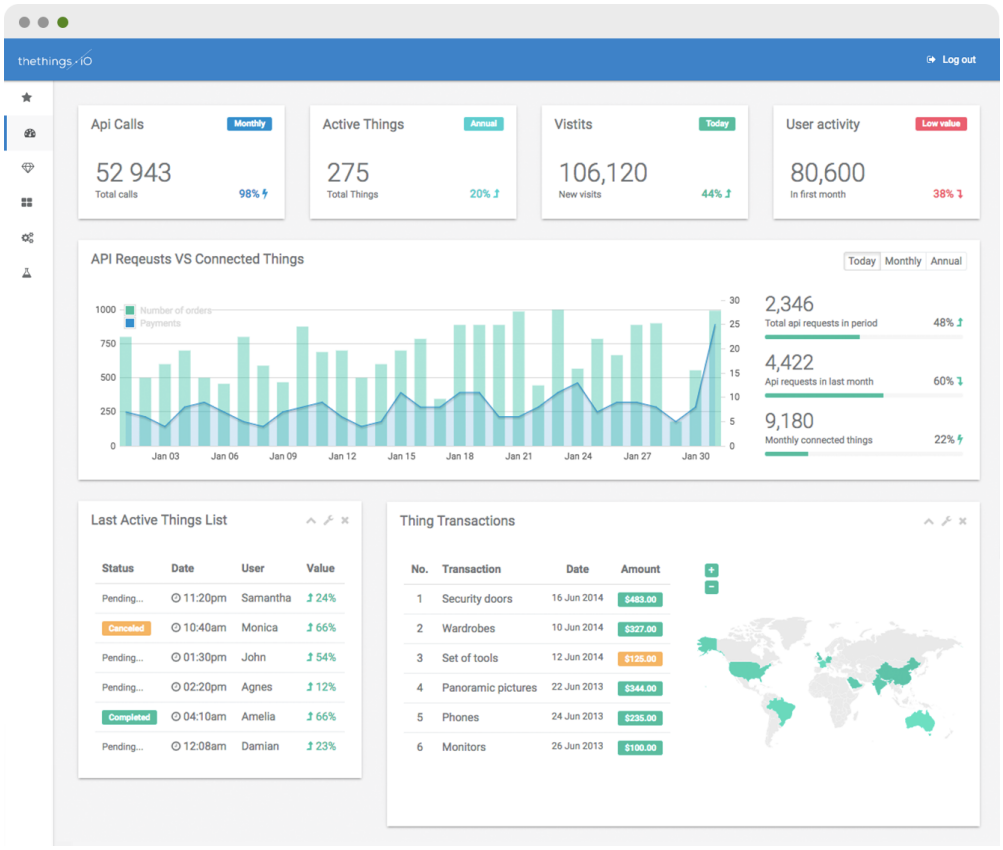


Figura 31 - Dashboard de TheThings.io (Fuente TheThings.io)

Mientras que Thinger.io solo permite mostrar los datos, TheThings.io es capaz de realizar un procesado de los mismos, y da la posibilidad de crear disparadores sobre los datos que se crean e integrar esos disparadores con servicios de terceros como IFTTT. Por ejemplo, se podría añadir un disparador que envíe un correo electrónico cuando una luz se encienda.

Aun así, la monitorización sobre los dispositivos y sistema sigue siendo limitada, por lo que, a pesar de ser una solución completa tampoco se ajusta a los objetivos del proyecto.

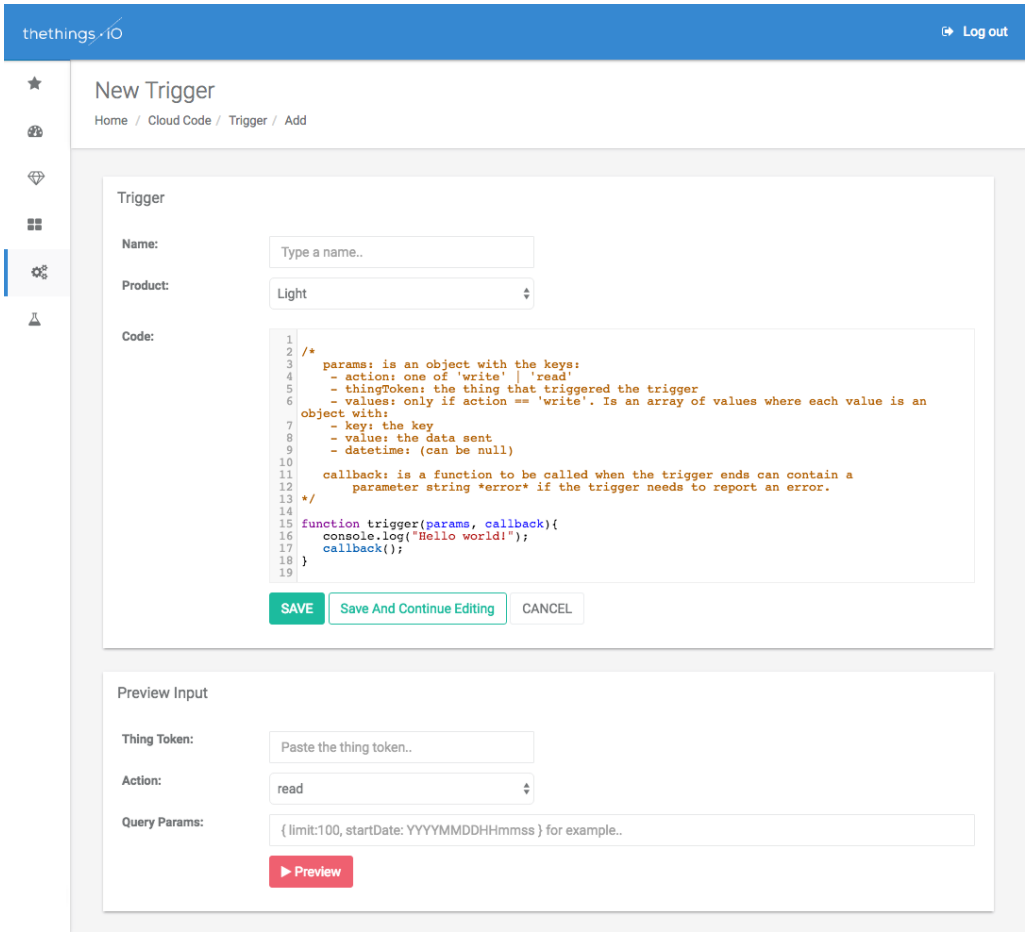


Figura 32 - Configuración de disparadores de TheThings.io (Fuente TheThings.io)

3.3 Conclusiones

Tal y como hemos comentado al inicio de esta sección, no hemos podido encontrar ninguna aplicación que se ajuste a los objetivos de este proyecto. Aun así, hemos podido analizar la forma en la que muchas de estas herramientas hacen determinados procesos, y una serie de paneles de administración, en proceso de producción, que pueden ser útiles a la hora de desarrollar nuestro propio panel.

4 Objetivos

El objetivo general de este trabajo es desarrollar una herramienta gráfica que sirva como panel de administración y monitorización para el sistema controlador de sondas. Para cumplir el objetivo principal, se plantean además los siguientes objetivos secundarios:

- Añadir herramientas de administración y gestión CRUD: posibilidad de administrar los recursos de la aplicación mediante una interfaz web.
- Añadir editor de texto para editar los scripts de configuración de los sensores.
- Añadir gráficas para monitorizar el sistema: se añadirán gráficas en tiempo real y estáticas de los procesos internos, las sondas y la monitorización general del sistema.
- Añadir elementos para una auditoría eficaz del sistema: generar y gestionar registros (logs) del sistema y herramientas para su inspección y seguimiento.
- Extender el sistema de identificación para usuarios humanos: actualmente la identificación es a nivel de aplicación, de manera que cada aplicación tiene una clave de acceso para poder realizar operaciones. Mediante la identificación para usuarios humanos se extenderá el sistema original de acceso y registro de usuarios con contraseña para acceder al panel de administración.
- Diseñar e implementar una interfaz web usable, accesible y atractiva.
- Modificación y optimización del sistema original: se deberán hacer modificaciones sobre el sistema original para permitir la conexión con el panel de administración y el uso del mismo.

5 Metodología

Tal como podemos leer en el trabajo de Rodríguez González (2008: 5-30), las metodologías tradicionales gastan mucho tiempo en analizar los requisitos iniciales, tratando de mantenerlos estables, ya que los cambios en los mismos producen riesgos, retrasos y grandes costos en el producto final, aunque el resultado del desarrollo es más predecible desde las etapas tempranas. De hecho, es el modelo usado por el resto de ingenierías, pero la del software no es comparable a estas.

Por el contrario, las metodologías ágiles surgen a raíz de hacer que el desarrollo de un producto sea lo más maleable posible, orientando la metodología sobre las personas y no sobre el proceso de desarrollo. Esta metodología se basa en unos pilares fundamentales descritos en el Manifiesto por el desarrollo ágil de software (Beedle 2001):

- Las personas y las interacciones valen más que los procesos y las herramientas.
- Un *software* que funciona vale más que una documentación extensiva: entrega continua de valor, es decir, definir iteraciones de un tiempo breve en la que se entregue siempre un producto funcional.
- La colaboración con el cliente vale más que la negociación del contrato: el cliente debe participar activamente en el desarrollo del producto.
- Saber responder ante posibles cambios vale más que seguir el plan establecido: los cambios en los requisitos no se deben de ver como un lastre, sino como una oportunidad para mejorar el sistema.

Tras este breve análisis de las distintas metodologías de software existentes y debido a la naturaleza del proyecto se ha optado por utilizar una metodología ágil. Aunque no se seguirá estrictamente ninguna metodología concreta como podría ser SCRUM o XP, ya que al tratarse de un único desarrollador estas no tienen sentido, sí que se tratará de seguir la filosofía ágil.

Para conseguirlo, se realizarán iteraciones del producto cada dos semanas, tratando de aportar valor en cada iteración e incrementando el desarrollo desde un producto mínimo viable hasta el producto final. Además, se usarán métodos de integración continua, de forma que se automatice

todo el despliegue de la aplicación, y una serie de tests que verifiquen el correcto funcionamiento de todos los elementos. Por último, se creará un tablero, con la herramienta online Trello[6], donde definiremos la planificación del proyecto por medio de historias de usuario. Estas historias de usuario se categorizarán entre desarrollo del frontal, del servidor o de las sondas, y se ordenarán según el estado de la tarea: idea, por hacer, en proceso y terminado.

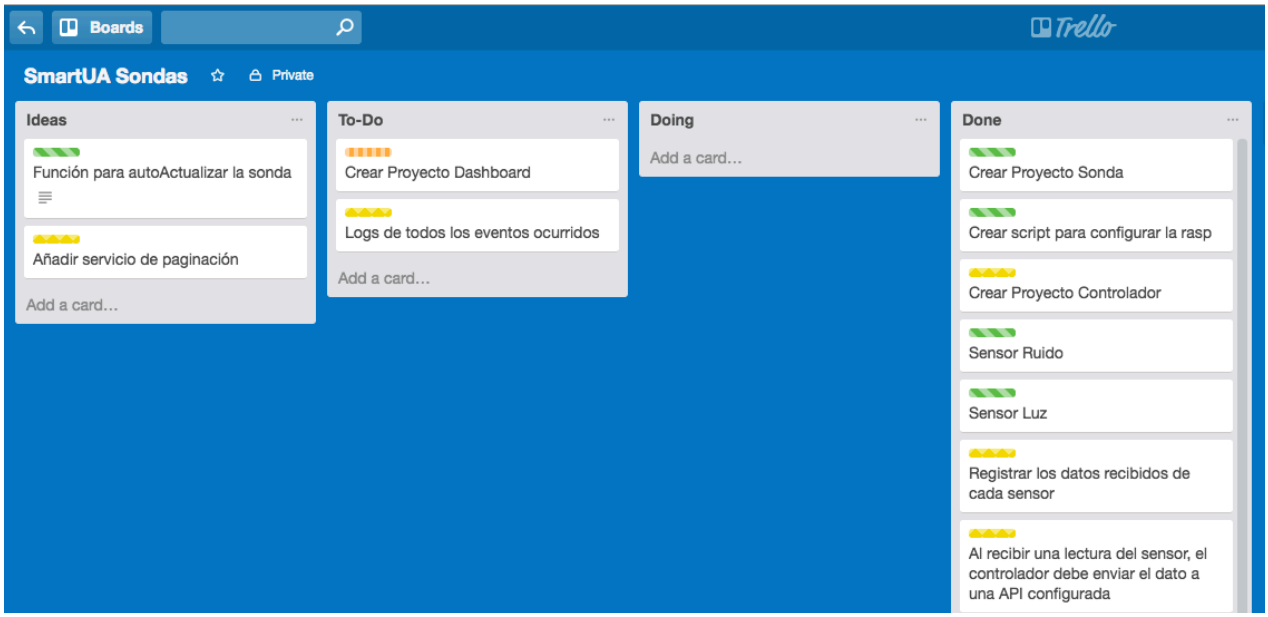


Figura 33 - Captura tablero Trello (Fuente propia)

[6] https://trello.com/

6 Análisis y especificación de requisitos

Esta sección sirve para definir y dar a conocer las funcionalidades y los requisitos del proyecto. Para la realización del documento se ha decidido utilizar el estándar IEEE 830 (Institute of Electrical and Electronics Engineers 2008 (1998)).

6.1 Descripción general

6.1.1 Perspectiva del producto

El proyecto consiste en desarrollar un panel frontal de administración para el sistema controlador de sondas dentro del proyecto Smart University de la Universidad de Alicante.

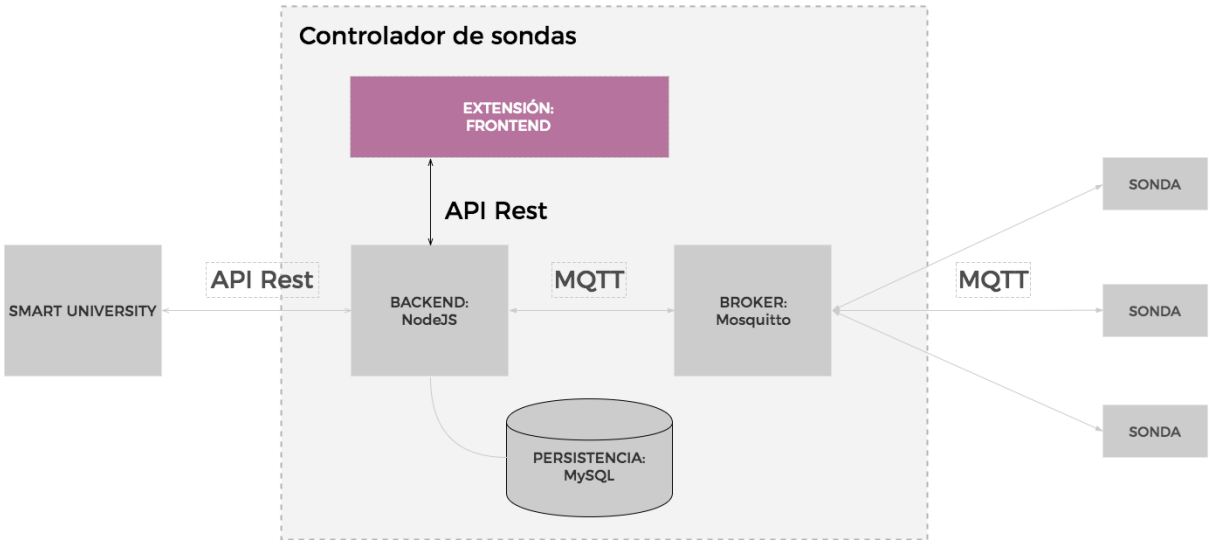


Figura 34 - Arquitectura sistema controlador de sondas, actualizado con frontend (Fuente propia)

La Figura 34 (actualización de la Figura 12, en color las partes nuevas) representa la arquitectura del sistema tras añadir el panel frontal de administración.

El panel se desarrollará con tecnologías web, siguiendo el modelo cliente-servidor distribuyendo la lógica de la aplicación entre los dos intervinientes. Al servidor existente se añadirán funcionalidades para poder cubrir todos los requisitos del panel frontal.

La aplicación será diseñada para ser lo más portable posible, por lo que, aunque actualmente se engloba dentro del proyecto Smart University, los diseños serán genéricos y se añadirá la posibilidad de ser usada por otros usuarios externos al proyecto.

6.1.2 Funciones del producto

- **Gestión de sondas**: se podrá administrar sondas y sensores con las operaciones básicas de lectura, creación, actualización y borrado de las mismas.
- **Monitorización de las sondas**: se podrá visualizar información de las sondas en paneles de monitorización mediante gráficas.
- **Gestión del código fuente de los sensores:** se podrá crear y editar el código fuente que las sondas interpretarán para hacer uso de los sensores.
- **Gestión de usuarios y aplicaciones:** se podrá dar de alta, actualizar y eliminar usuarios y aplicaciones dentro del panel.
- **Gestión de los registros del sistema:** representación de los registros del sistema mediante tablas en las que se podrá buscar y ordenar la información.
- **Creación de alarmas:** será posible crear alarmas sobre los sistemas, de tal forma que obtengamos alertas sobre ciertos estados que se definan.

6.1.3 Características de los usuarios

El panel está enfocado a ser usado por los propios usuarios de Smart University, en concreto a los usuarios administradores, los cuales son capaces de añadir sondas y sensores en el sistema original. Se presupone un nivel de conocimientos informáticos alto. La naturaleza del sistema hará que no sea necesaria la experiencia previa en el mismo, aunque ayudará si se tiene conocimiento de sistemas similares.

6.1.4 Restricciones

- La interfaz deberá ser accesible mediante Internet y desde cualquier dispositivo.
- El sistema será diseñado siguiendo el modelo cliente-servidor, y será este último el original del sistema.
- Se implementará la seguridad en dos ámbitos distintos: por un lado, la autenticación y autorización de los usuarios que acceden al panel y, por otro lado, la seguridad en la capa de transporte, es decir, la comunicación segura entre el cliente y el servidor mediante el protocolo TLS.

6.1.5 Suposiciones y dependencias

El panel dependerá en gran medida de la especificación del servidor y del correcto funcionamiento del mismo, ya que es la fuente de los datos y donde se aloja la API en la que se destinarán las acciones realizadas en el frontal.

6.2 Especificación de requisitos

En esta sección, tal como se describe en el IEE 830, dividiremos los requisitos entre aquellos de carácter funcional, no funcional y los comunes a interfaces.

6.2.1 Requisitos funcionales

En cuanto a los requisitos funcionales, serán aquellos que el sistema deberá realizar. Cada requisito tendrá un identificador único; el actor involucrado, que en nuestro caso podrá ser el usuario, el sistema o ambos (mixto); el nombre, un nombre descriptivo; la descripción del requisito; los requisitos lógicos y la prioridad.

Identificador	RF-01
Actor involucrado	Usuario
Nombre	Autenticación
Descripción	Cualquier usuario dado de alta en la aplicación podrá entrar en la aplicación mediante autenticación con usuario y contraseña.
Requisitos lógicos	Será necesario estar dado de alta previamente en el sistema para poder acceder al mismo.
Prioridad	Alta

Identificador	RF-02
Actor involucrado	Usuario
Nombre	Salir del sistema
Descripción	Los usuarios autenticados en la aplicación podrán salir de la aplicación.
Requisitos lógicos	Será necesario estar autenticado previamente en el sistema para poder acceder al mismo. Una vez cerrada la sesión se limpiarán todos los datos referentes al usuario en el navegador.

Prioridad	Alta

Identificador	RF-03
Actor involucrado	Usuario
Nombre	Visualizar resumen sistema
Descripción	El usuario podrá ver un resumen del sistema donde se monitorizarán los eventos más recientes.
Requisitos lógicos	-
Prioridad	Alta

Identificador	RF-04
Actor involucrado	Usuario
Nombre	Gestión de aplicaciones
Descripción	El usuario podrá listar, dar de alta, actualizar y eliminar las aplicaciones en el sistema.
Requisitos lógicos	Cada aplicación se compondrá por: ID, cadena de texto única, y un clave para acceder a la API. Esta clave será un identificador único autogenerado. Las aplicaciones son las propietarias de los dispositivos dados de alta en el sistema y sirven para poder hacer peticiones al sistema del tipo máquina-máquina. Por ejemplo, Smart University es una aplicación en este sistema, y realiza peticiones de alta, edición y baja de sondas usando su clave de acceso a la API. El sistema deberá disponer de una serie de *endpoints* en la API para realizar las operaciones CRUD sobre las aplicaciones.
Prioridad	Media

Identificador	RF-05
Actor involucrado	Usuario
Nombre	Regenerar API key de las aplicaciones
Descripción	El usuario podrá regenerar en cualquier momento la clave de acceso a la API de una aplicación.
Requisitos lógicos	El elemento será simplemente un botón que regenere automáticamente una clave de acceso mediante una llamada al backend.

	El sistema deberá disponer de un *endpoint* en la API para regenerar la API key de las aplicaciones.
Prioridad	Opcional

Identificador	RF-06
Actor involucrado	Usuario
Nombre	Gestión de usuarios
Descripción	El usuario podrá listar, dar de alta, actualizar y eliminar los usuarios del sistema.
Requisitos lógicos	Cada usuario se compondrá por: un ID autogenerado, nombre de usuario, contraseña y aplicación a la que pertenece. El sistema deberá disponer de una serie de *endpoints* en la API para realizar las operaciones CRUD sobre los usuarios.
Prioridad	Media

Identificador	RF-07
Actor involucrado	Usuario
Nombre	Gestión de tipos de sensores
Descripción	El usuario podrá listar, dar de alta, actualizar y eliminar los tipos de sensores en el sistema.
Requisitos lógicos	El sistema deberá disponer de una serie de *endpoints* en la API para realizar las operaciones CRUD sobre los tipos de sensores.
Prioridad	Alta

Identificador	RF-08
Actor involucrado	Usuario
Nombre	Edición de scripts de los tipos de sensores
Descripción	El usuario, a través de un editor de texto integrado, podrá editar los scripts mediante los cuales los dispositivos se configurarán para conocer el funcionamiento de los distintos sensores.
Requisitos lógicos	-
Prioridad	Media

Identificador	RF-09
Actor involucrado	Usuario
Nombre	Gestión de sensores
Descripción	El usuario podrá listar, dar de alta, actualizar y eliminar los sensores pertenecientes a las sondas en el sistema.
Requisitos lógicos	Para dar de alta un sensor será necesario definir el tipo del mismo y la sonda a la que pertenece. El sistema deberá disponer de una serie de *endpoints* en la API para realizar las operaciones CRUD sobre los sensores.
Prioridad	Alta

Identificador	RF-10
Actor involucrado	Usuario
Nombre	Gestión de dispositivos
Descripción	El usuario podrá listar, dar de alta, actualizar y eliminar los dispositivos.
Requisitos lógicos	En última instancia, los dispositivos son propiedad de la aplicación, y no de los usuarios. El sistema deberá disponer de una serie de *endpoints* en la API para realizar las operaciones CRUD sobre los dispositivos.
Prioridad	Alta

Identificador	RF-11
Actor involucrado	Usuario
Nombre	Vista detalle dispositivo
Descripción	El usuario podrá acceder a una vista detallada del dispositivo donde ver aspectos más específicos del mismo.
Requisitos lógicos	-
Prioridad	Media

Identificador	RF-12
Actor involucrado	Usuario
Nombre	Añadir sensor a dispositivos

Descripción	El usuario podrá añadir sensores a los dispositivos desde la vista detallada del dispositivo.
Requisitos lógicos	El usuario deberá estar en la vista detallada del dispositivo para poder añadir nuevos sensores.
Prioridad	Media

Identificador	RF-13
Actor involucrado	Usuario
Nombre	Conocer el estado del dispositivo
Descripción	El usuario podrá ver el estado del dispositivo, así como pedir al dispositivo que envíe un nuevo estado.
Requisitos lógicos	El sistema deberá disponer de un *endpoint* en la API para realizar la acción.
Prioridad	Media

Identificador	RF-14
Actor involucrado	Usuario
Nombre	Auditoría del sistema
Descripción	El usuario podrá ver y buscar los registros del sistema. Será la composición de cada uno de los registros generados por operaciones sobre cada modelo del sistema: dispositivo, sensor, tipo de sensor, usuarios y aplicaciones.
Requisitos lógicos	El sistema deberá disponer de una serie de *endpoints* en la API para realizar las operaciones CRUD sobre los registros.
Prioridad	Media

Identificador	RF-15
Actor involucrado	Usuario
Nombre	Búsquedas personalizadas sobre los modelos
Descripción	El usuario podrá definir la búsqueda sobre los distintos modelos de forma personalizada filtrando los datos representados.
Requisitos lógicos	-
Prioridad	Media

Identificador	RF-16

Actor involucrado	Usuario
Nombre	Visualizar mapa de dispositivos
Descripción	El usuario podrá visualizar en un mapa los dispositivos.
Requisitos lógicos	-
Prioridad	Media

Identificador	RF-17
Actor involucrado	Usuario
Nombre	Visualizar actividad de dispositivos
Descripción	El usuario podrá visualizar la actividad de los dispositivos y los sensores.
Requisitos lógicos	-
Prioridad	Media

Identificador	RF-18
Actor involucrado	Usuario
Nombre	Crear y administrar alarmas
Descripción	El usuario podrá crear alarmas sobre los dispositivos y sensores que se dispararán para avisar al usuario. Por ejemplo, se podría crear una alarma cuando una sonda no envíe su estado en un periodo mayor a una hora.
Requisitos lógicos	-
Prioridad	Opcional

Tabla 1 - Requisitos funcionales (Fuente propia)

6.2.2 Requisitos de diseño

Identificador	RD-01
Nombre	Multiplataforma
Descripción	La aplicación deberá funcionar independientemente del dispositivo con la que se acceda, aunque principalmente se contemplarán teléfonos inteligentes de un rango de tres tamaños, tabletas de tamaño medio y ordenadores.

Identificador	RD-02

Nombre	Interfaz similar a Smart University
Descripción	Para mantener la coherencia y continuidad entre los dos sistemas, se tratará de implementar una interfaz similar, dentro de las opciones disponibles, a la de Smart University. Si esto no es factible, se planteará un diseño que encaje en la mayor medida posible.

Identificador	RD-03
Nombre	Interfaz usable
Descripción	Es vital prestar atención a la usabilidad del sistema para mejor la experiencia de usuario.

Tabla 2 - Requisitos de diseño (Fuente propia)

6.2.3 Requisitos no funcionales

Identificador	RNF-01
Nombre	Seguridad
Descripción	El sistema deberá autorizar y autenticar a los usuarios cuando estos hagan peticiones sobre un recurso o acción. Además, la conexión entre cliente y servidor debe ser segura. Por otra parte, los datos sensibles del usuario deberán estar cifrados antes de almacenarlos.

Identificador	RNF-02
Nombre	Optimización
Descripción	En el momento de desarrollar el panel de administración se hará especial hincapié en mejorar el sistema original de tal forma que se mejore la optimización.

Identificador	RNF-03
Nombre	Disponibilidad
Descripción	El sistema debe garantizar una disponibilidad de este de modo que se pueda acceder a cualquier hora del día los siete días de la semana.

Identificador	RNF-04
Nombre	Mantenibilidad

Descripción	El desarrollo del sistema debe ser documentado de tal forma que, en todo momento, cualquier desarrollador pueda mantener el sistema.

Identificador	RNF-05
Nombre	Escalabilidad
Descripción	El sistema debe ser escalable de modo que en un momento dado este pueda crecer tanto en la parte hardware como software.

Tabla 3 - Requisitos no funcionales (Fuente propia)

7 Diseño

Dentro del diseño abarcaremos diferentes aspectos de la aplicación: desde la arquitectura del sistema, pasando por la seguridad y modificaciones en la base de datos, hasta el diseño de interfaces.

7.1 Arquitectura del servidor

Como ya hemos comentado anteriormente, partimos de un sistema con un servidor ya desarrollado, al cual le añadiremos un panel frontal de administración, y, además, en el camino, lo modificaremos para añadir optimizaciones y compatibilidad con el frontal. Esto lo podemos ver representado en la Figura 34.

En la sección 2.2.3 y en la Figura 17 podemos ver la arquitectura actual del servidor. Este sistema será modificado para habilitarlo para el uso con un frontal. Si observamos los requisitos descritos en la sección anterior, las modificaciones que habrá que desarrollar serán las siguientes:

- Servicio de autenticación: se creará un servicio que permitirá definir las rutas de la API que necesitarán autenticación por parte del usuario. Este servicio funcionará como una capa intermedia entre la API y los controladores, y podrá aceptar o rechazar las peticiones según el resultado de la verificación del usuario.
  Requisitos relacionados: RF-01, RNF-01
- Creación de nuevas rutas en la API: debido a la creación de nuevos modelos de datos y de la necesidad de acceso a los mismos por parte del frontal será necesario crear nuevas rutas en la API.
  Requisitos relacionados: RF-04, RF-06, RF-07, RF-09, RF-10, RF-14
- Ruta para regenerar clave de acceso: para regenerar la clave de acceso de las aplicaciones se creará una ruta específica que generará el nuevo valor, lo guardará en la base de datos y lo devolverá al cliente.
  Requisitos relacionados: RF-05
- Controlador para conocer estado de un dispositivo: será necesario crear un servicio y una ruta en la API que hará una petición al dispositivo y devolverá su estado.
  Requisitos relacionados: RF-13

- Servicio de paginación y búsqueda: será necesario crear un servicio para poder paginar y limitar las búsquedas, así como hacer búsquedas sobre los campos de los modelos.
  Requisitos relacionados: RF-15
- Servicio para generar registros: será necesario crear un sistema para registrar todas las acciones dentro de la aplicación. Dentro de estas acciones se agrupan las operaciones CRUD sobre los modelos y los accesos a métodos de la API.
  Requisitos relacionados: RF-14

Los cambios propuestos se pueden ver en los componentes coloreados de la Figura 35.

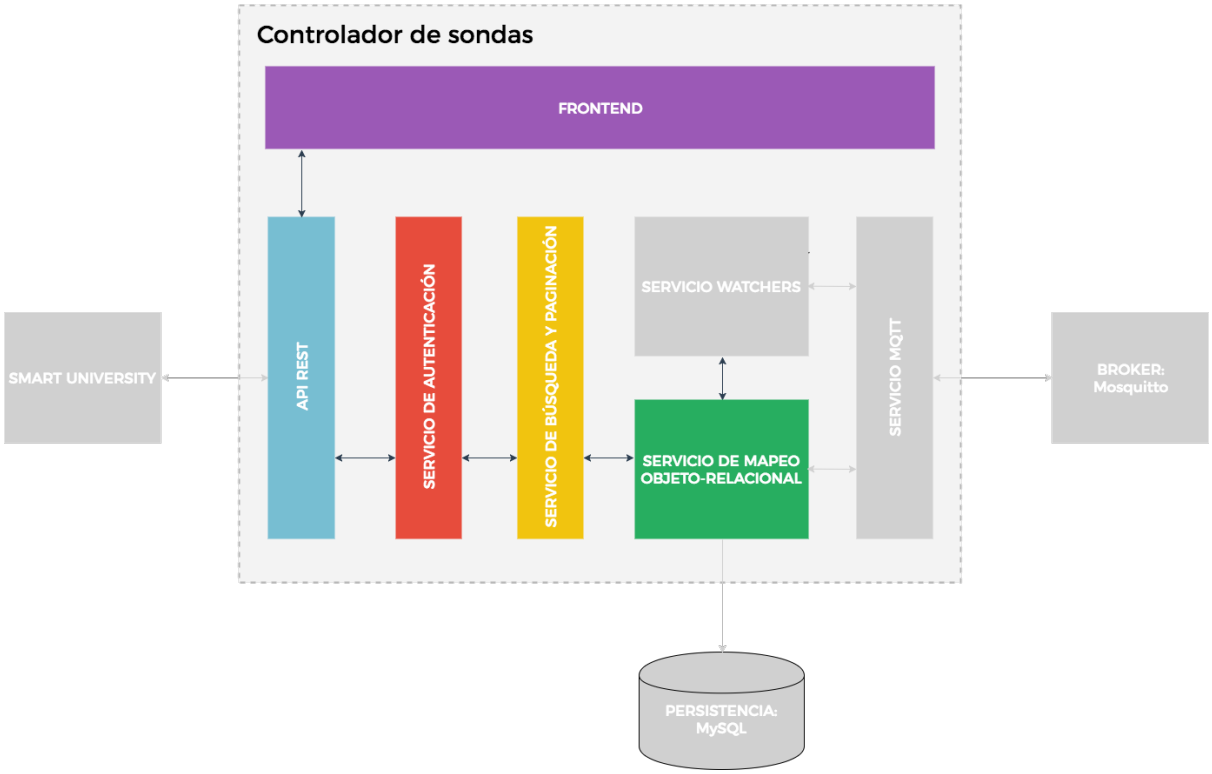


Figura 35 - Arquitectura interna del controlador (Iteración 1) (Fuente propia)

7.2 Arquitectura del cliente

El panel de administración de Smart University está desarrollado con el *framework* de Google AngularJS[7], versión 1.X. En septiembre de 2016, se lanzó la versión número 2 de la plataforma, una versión que fue totalmente reescrita desde cero y que es incompatible con las versiones

[7] https://angularjs.org/

anteriores. En un primer momento, para el desarrollo del cliente, se pensó en utilizar AngularJS para mantener la consistencia con el panel original de Smart University, pero tras conversar con el equipo se decidió dar un paso adelante y usar la versión 2.x o incluso otro *framework* distinto.

React[8] y VueJS[9] eran los otros dos candidatos que se barajaban debido a su popularidad y el crecimiento exponencial que han sufrido recientemente.

Finalmente, Angular 2[10] es la plataforma elegida, principalmente debido a mis conocimientos previos sobre el *framework*, pero también por la coherencia con el equipo de Smart University. Aunque actualmente el *dashboard* utilice Angular 1, en la hoja de ruta está planeada la actualización a Angular 2, por lo que cuando la actualización se produzca será sencillo mantener sistemas con el mismo *framework* y se abre la posibilidad a la reutilización de código entre las dos plataformas, creando un sistema globalmente más compacto y robusto. También cabe destacar que es una plataforma mucho más madura que la versión anterior, incorporando ciertas mejoras como un tipado estático y uso de clases mediante TypeScript.

Aunque no es obligatorio, las guías oficiales de Angular 2 recomiendan el uso de una arquitectura específica a la hora de desarrollar aplicaciones usando este *framework*. Obviamente es solo una guía de estilo que se deberá adaptar a las necesidades de cada proyecto. Dentro de las aplicaciones de Angular 2 podemos diferenciar distintos tipos de elementos:

- **Componentes:** son los elementos más usados, estos se componen a su vez de tres ficheros: HTML, CSS y JavaScript. Usando componentes podemos definir nuevos elementos HTML. Por ejemplo, podemos encapsular en un componente una imagen e instanciarla con una etiqueta HTML.
- **Servicios:** los servicios no afectan directamente a los elementos HTML, sino que son funciones que pueden ser usados por los componentes, directivas u otros servicios. Cuando necesitamos recoger datos de un servicio externo (una API, por ejemplo) se suele usar un servicio que funciona de intermediario entre la API y el componente que necesita el dato. Por ejemplo, en nuestro caso, podemos tener un componente que

[8] https://facebook.github.io/react/
[9] https://vuejs.org/
[10] https://angular.io/

representa una sonda, el componente sonda pide al servicio la información de la sonda y este es en última instancia quien conecta con la API.

- **Módulos:** los módulos agrupan servicios, componentes, directivas y otros módulos en un paquete que se puede importar en otros módulos. Volviendo al caso anterior, podemos tener un módulo que sea todo el paquete que representa a la sonda, desde el servicio que conecta con el API hasta el componente que representa mediante HTML la sonda en la aplicación.
- **Enrutador:** también disponemos de un enrutador capaz de leer la dirección de la URL y invocar los componentes o módulos solicitados.

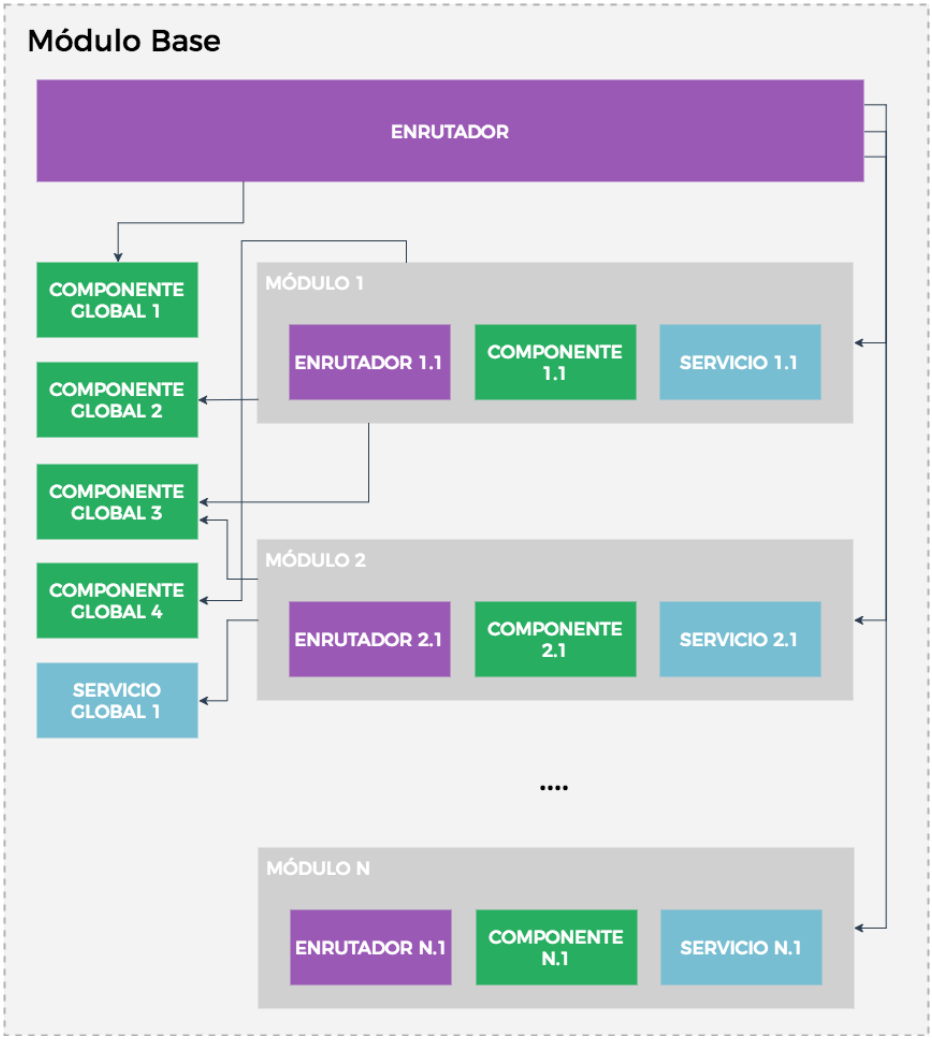


Figura 36 - Arquitectura básica de una aplicación Angular 2 (Fuente propia)

7.3 Diseño de la base de datos

En la Figura 37 podemos observar las modificaciones propuestas en cuanto al diseño de la base de datos.

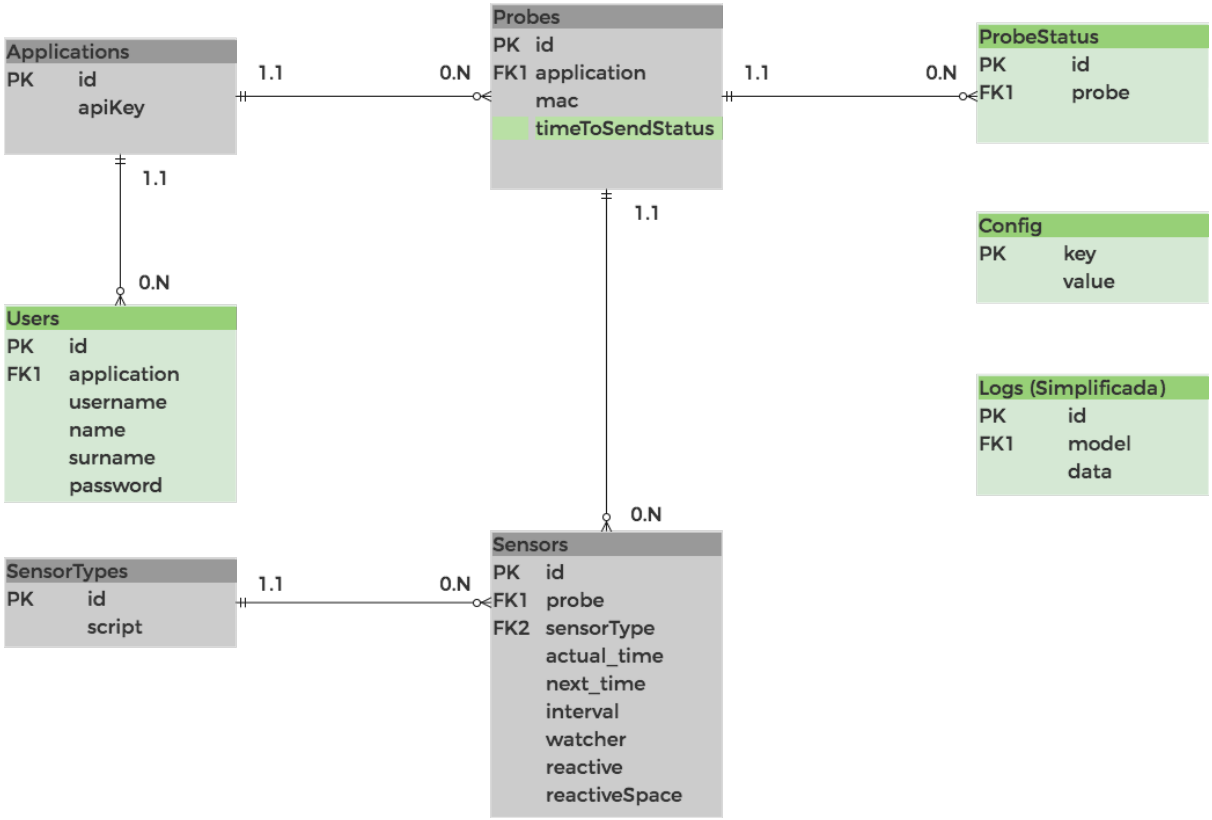


Figura 37 - Esquema base de datos (Iteración 1) (Fuente propia)

Se ha añadido la tabla **usuarios**. Una aplicación tendrá cero o muchos usuarios. Estos últimos estarán compuestos por nombre, apellido, nombre de usuario y contraseña. Además de una clave ajena a la tabla aplicaciones.

Por otro lado, se ha añadido una tabla **config** donde se guardarán las configuraciones del sistema, como podrían ser idioma por defecto, tiempo por defecto para el refresco de los sensores y otras variables de entorno. También se ha añadido una tabla **logs**, que en la Figura 37 se ve representada de una forma simplificada, ya que para cada modelo principal (usuarios, sensores, tipos de sensores, sondas y aplicaciones) habrá una tabla de registro relacionada en la que se guardarán datos de las operaciones realizadas sobre las tablas.

Finalmente, también se ha añadido una tabla **estadoSondas** para guardar los estados enviados por las sondas. Estos estados contendrán información relacionada con la propia sonda, como puede ser la temperatura del procesador, cantidad de datos enviados, tiempo en funcionamiento, versión del software, etc.

Requisitos relacionados: RF-06, RF-13, RF-14

7.4 Seguridad

Con la arquitectura elegida se ha optado por un sistema sin estado (en inglés *stateless*). Esto es debido a que, al haber separado el panel frontal del servidor, con una API REST, cada petición a la misma será independiente a las anteriores y a las posteriores, aunque procedan del mismo usuario. En el modelo tradicional se mantenía el estado y el servidor sabía en todo momento donde se encontraba el usuario y quién era dentro de la aplicación.

La gran problemática con los sistemas con estado (en inglés *stateful*) viene dada por la facilidad de dejar hilos de estado abiertos incompletos cuando no deberían, además de hacer el sistema más propenso a errores. El concepto básico es que los sistemas con estado guardan memoria del pasado, es decir, cualquier operación realizada puede desembocar en un cambio de estado, y las consecuentes operaciones pueden no dar el mismo resultado. En un sistema sin estado, una operación deberá devolver siempre el mismo valor.

Aunque no son muy obvios a primera vista, los beneficios de los modelos sin estado superan con creces a los modelos con estado, aunque estos últimos tienen cabida en ciertas aplicaciones. Una de las principales ventajas es la posibilidad de escalar, ya que si no existe estado y todas las peticiones son independientes, en un servicio web, por ejemplo, podríamos añadir más máquinas sin tener que dirigir a los usuarios a una máquina concreta en el que estuviese su estado, ya que este no existe. Otra ventaja relacionada con la escalabilidad y los sistemas sin estado es la posibilidad de poner en funcionamiento otra máquina en el caso de que la principal esté fuera de servicio sin perder la sesión de los usuarios conectados y sin que estos perciban ningún cambio.

A raíz del crecimiento del modelo RESTFul, en el que los servicios son sin estado, también surge la necesidad del uso de métodos de autenticación sin estado. Uno de los más populares y usados en la actualidad es el uso de *JSON Web Token* o JWT.

Tal y como dice Jones et al. (2015), JWT es un protocolo mediante el cual se puede conseguir la autenticación sin estado. El protocolo consiste en generar un *token* con una firma que solo conoce una de las dos partes. Una vez generado, el *token* se envía a la otra parte y esta última lo almacena. Es decir, una de las dos partes cifra y puede descifrar el *token*, mientras que la otra simplemente lo almacena y lo envía en cada petición. Una vez recibido, la parte generadora puede verificar la integridad del *token*. El *token* se compone de tres partes:

- **Cabecera:** se descompone en dos partes, el tipo de *token*, y el algoritmo usado para la encriptación.
- **Contenido:** aquí se escribe el contenido del *token,* en nuestro caso al usarlo para la autenticación escribiríamos el identificador del usuario.
- **Firma:** la firma es la parte crucial; en ella se combina la cabecera y el contenido, además de una firma que solo conoce el emisor del JWT y se cifra usando el algoritmo descrito en la cabecera.

Los tres campos se concatenan separados por puntos '.', de la forma siguiente:

```
cabecera.contenido.firma
```

Un ejemplo sería el siguiente JWT:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

Figura 38 - Flujo del protocolo JWT (Fuente propia)

Requisitos relacionados: RNF-01, RNF-03, RNF-05

7.5 Diseño interfaces

En cuanto al diseño de interfaces, vamos a realizar un planteamiento en una primera instancia lo más sencillo posible, para hacer una aproximación básica a la posición de los elementos, y su usabilidad en distintos dispositivos. Para simplificar, existen vistas que en el futuro consistirán de vistas más específicas para cada modelo. Los modelos disponibles en el sistema son los siguientes: dispositivos, sensores, peticiones a sensores, aplicaciones y usuarios.

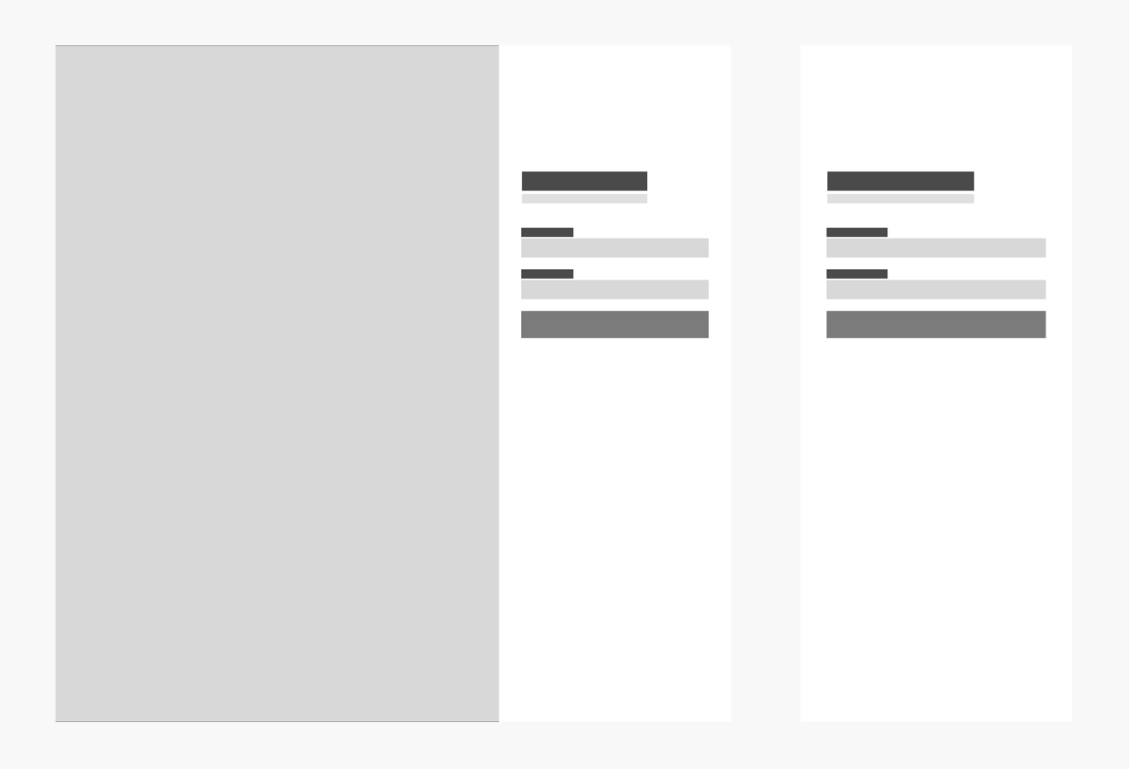


Figura 39 - Vista de login (Fuente propia)

En la Figura 39, podemos observar la vista de la página de login. Tanto en la versión escritorio como en la móvil esta vista contendrá un formulario, que en el caso de los dispositivos más pequeños ocupará toda la pantalla. El formulario tendrá dos campos: correo electrónico y contraseña, además de un botón para enviar el mismo.

Requisitos relacionados: RF-01, RNF-01

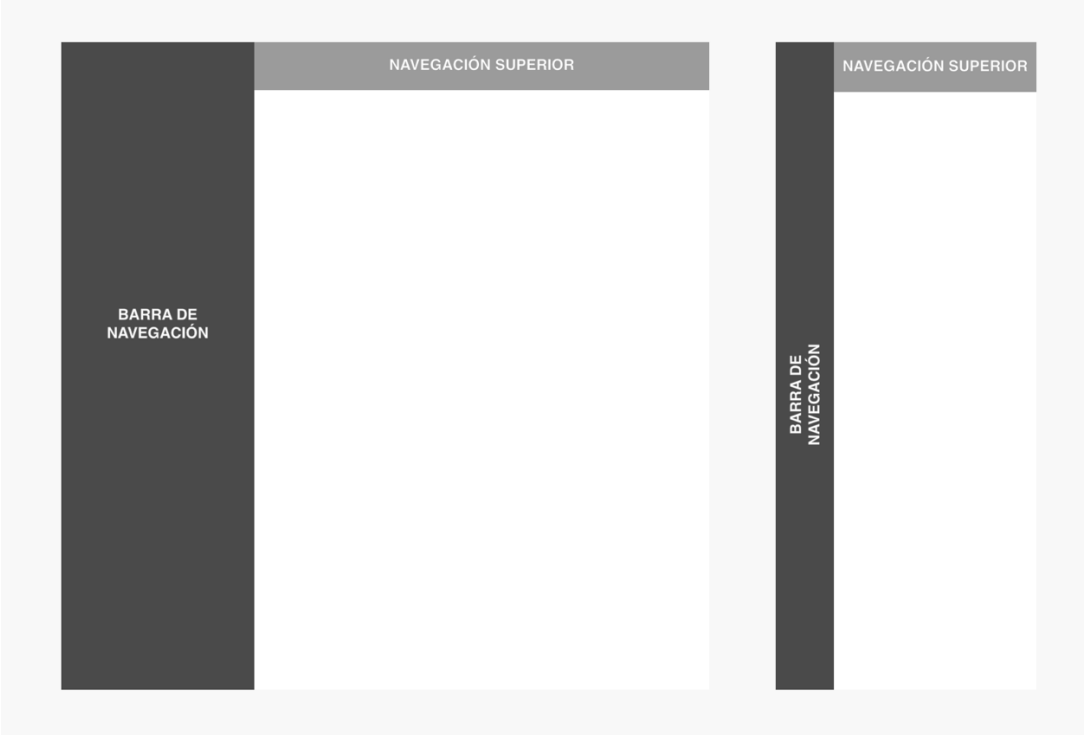


Figura 40 - Plantilla de la interfaz (Fuente propia)

En la Figura 40 podemos ver la plantilla de la aplicación que servirá como base para el resto de vistas. En esta plantilla dispondremos de un marco común que siempre será visible y un fragmento que irá cambiando según el estado de la aplicación. En este marco común se muestran dos elementos fijos, la barra de navegación lateral y superior.

- **La barra de navegación lateral** servirá para mostrar las diferentes rutas, desde esta podremos navegar a las diferentes vistas de la aplicación. En los dispositivos móviles está barra se verá reducida.
- **La barra de navegación superior** tendrá una serie de acciones inmediatas cómo será el botón de salir del sistema.

Requisitos relacionados: RF-01

Figura 41 - Vista resumen del sistema (Fuente propia)

En la Figura 41 podemos ver una aproximación de la vista que mostrará el resumen del sistema. Se mostrarán una serie de gráficas en las que se podrá analizar rápidamente el estado del sistema.

Requisitos relacionados: RF-03

Figura 42 - Vista página de cada modelo (Fuente propia)

En la Figura 42 se muestra el diseño de las páginas relativas a cada modelo: dispositivos, sensores, peticiones de lectura, aplicaciones y usuarios. En esta vista se mostrará una cabecera con el título del contenido, una gráfica resumen y una tabla con las entradas del modelo en base de datos.

Requisitos relacionados: RF-04, RF-06, RF-07, RF-09, RF-10, RF-14, RF-16

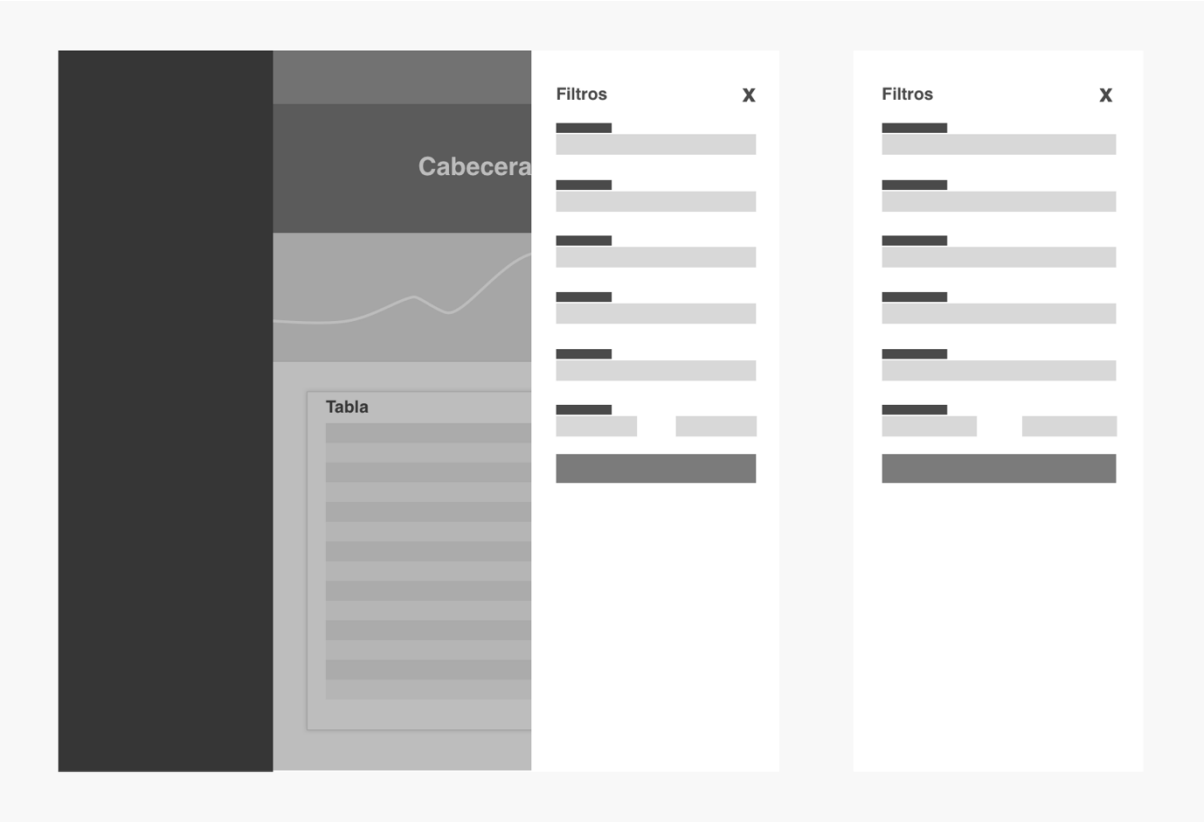


Figura 43 - Vista filtros activos (Fuente propia)

Los datos mostrados en la Figura 42 podrán ser filtrados mediante el componente mostrado en la Figura 43. El componente filtros será una ventana modal que se sobrepondrá a la vista por el lado derecho en el caso del escritorio y que ocupará toda la pantalla en el caso de un dispositivo más pequeño. En ambos casos consistirá en un formulario con los diferentes campos filtrables, un botón para resetearlo y un botón para cerrarlo.

Requisitos relacionados: RF-15

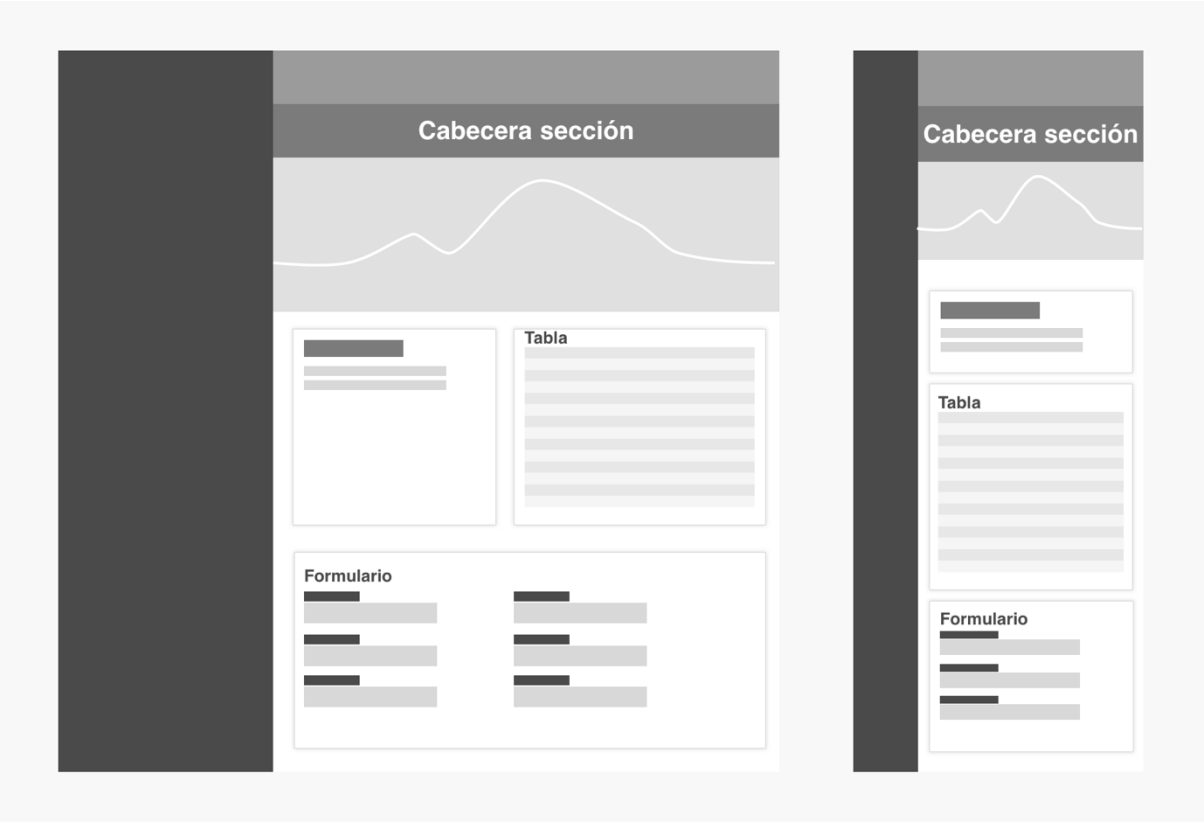


Figura 44 - Vista detalle y edición de modelo (Fuente propia)

La vista detalle y edición de modelo mostrada en la Figura 44 consistirá en una o varias gráficas que resumirán el estado del modelo, otras secciones con información del mismo y un formulario para editar los valores del elemento.

Requisitos relacionados: RF-04, RF-05, RF-07, RF-08, RF-09, RF-10, RF-11, RF-12, RF-16, RF-17

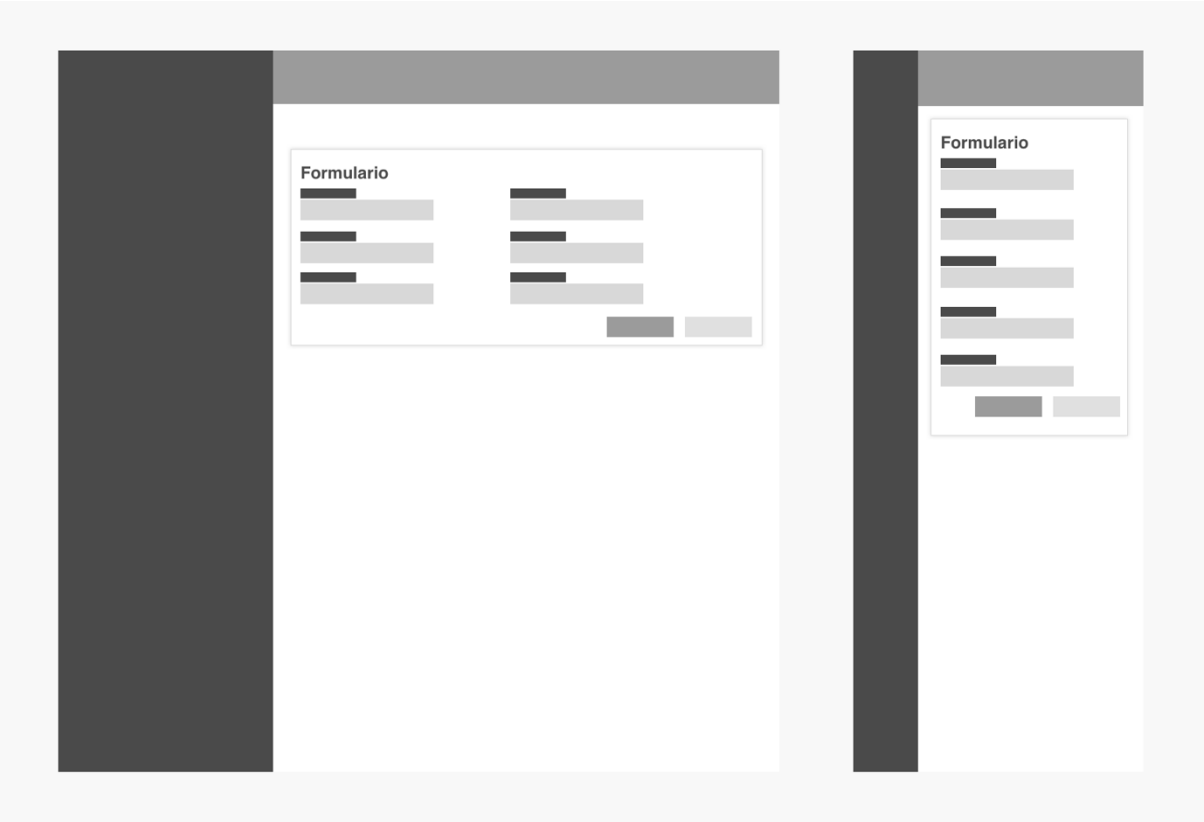


Figura 45 - Vista nuevo elemento (Fuente propia)

Finalmente, la vista para crear un nuevo elemento (Figura 45) consistirá en un formulario con los campos relativos al modelo, un botón de enviar y otro de cancelar.

Requisitos relacionados: RF-04, RF-06, RF-07, RF-09, RF-10, RF-14, RF-16

8 Implementación del servidor

Como ya comentamos en la sección de diseño, la parte del servidor a pesar de no ser nueva, era necesario modificarla de forma que pudiese funcionar con el cliente web. A continuación, detallaremos en qué han consistido estas modificaciones y como han sido implementadas.

8.1 Servicio de autenticación

El servicio de autenticación es el encargado de restringir el acceso a la API por parte de los usuarios, de tal forma que solo aquellos autorizados puedan acceder a unas determinadas rutas.

Para ello, lo primero que se ha hecho es la creación de una tabla **usuarios** en la base de datos. En la Tabla 4 se describe la estructura de la tabla **usuarios** en la base de datos:

Nombre	Tipo de dato	Descripción
ID	INT (10)	Clave primaria autoincremental.
email	VARCHAR (255)	Correo electrónico del usuario.
password	VARCHAR (255)	Contraseña del usuario encriptada.
name	VARCHAR (255)	Nombre del usuario.
surname	VARCHAR (255)	Apellido del usuario.
application	VARCHAR (255)	Clave ajena. Define la aplicación a la que pertenece el usuario.
created_at	TIMESTAMP	Fecha de creación.
updated_at	TIMESTAMP	Fecha de actualización.

Tabla 4 - Descripción tabla usuario (Fuente propia)

Una vez creada la tabla, hemos creado los controladores capaces de crear, actualizar, listar y eliminar usuarios. Hemos tenido en cuenta a la hora de la creación y actualización de usuarios que la contraseña se guardara encriptada en la base de datos.

Tras documentarnos sobre cuál es el mejor método para encriptar y guardar una contraseña, hemos llegado a la conclusión de que la mejor opción es usar el mecanismo de encriptación mediante hash *bcrypt*, explicado en la publicación de Provos y Mazières (1999: 1), y que consiste en una variante del algoritmo de encriptación *Blowfish* (Schneier 1993). El motivo de usar este mecanismo, como bien explica Coda Hale en la referencia (Coda Hale 2010), es que los métodos

utilizados tradicionalmente para encriptar contraseñas son demasiado rápidos. Es extraño pensar que si es más rápido es peor, pero en realidad cuando hablamos de contraseñas no lo es. Esto es debido a que, si alguien quisiera realizar un ataque mediante fuerza bruta, es decir, probando diferentes series de contraseñas, en el caso de una contraseña en minúsculas, con números y letras y de 6 caracteres de largo, encriptada usando el algoritmo MD5 y con una capacidad de procesamiento estándar podría probar todas las combinaciones posibles en menos de 40 segundos. En cambio, usando *bcrypt* bajo las mismas condiciones y usando un factor de trabajo de 12 tardaría alrededor de 12 años en probar todas las contraseñas. Además, podemos modificar el factor de trabajo para que la encriptación sea más rápida, aunque probablemente el usuario final no note ninguna diferencia.

Figura 46 - Flujo del middleware de autenticación (Fuente propia)

Otra de las partes importantes de este servicio de autenticación es la función que permite identificar a los usuarios. En ExpressJS se pueden implementar servicios en forma de *middleware*, es decir, por cada ruta o grupo de rutas podemos intercalar un servicio que podrá llegar a interrumpir el flujo de la petición si no se cumplen unos determinados parámetros. Precisamente esta es la forma que tiene el servicio de autenticación, un servicio que en función de la petición decide si continua el flujo o por el contrario se detiene y devuelve un mensaje de error indicando que el usuario no está autenticado. Si recordamos la Figura 38, el paso número 5 es: "Verificar firma del token y obtener datos del usuario" y es en ese momento del flujo donde intercalamos esta función. Para verificar los tókens JWT enviados por el usuario hemos hecho uso de la librería *express-jwt*[11].

```
router.use('/probe', authService.isAuthenticated, probe);
router.use('/:apiKey/probe', authService.isValidApiKey, probe);
```

[11] https://github.com/auth0/express-jwt

Las líneas anteriores extraídas del código fuente representan la forma de instanciar el *middleware* para la ruta de las sondas. En el primer caso comprobamos que se trate de un usuario autenticado, y para el segundo caso se trata de la validación de la clave de la API de una aplicación.

8.2 Servicio de paginación y búsqueda

Dentro del servicio de paginación y búsqueda hemos creado cuatro funciones, cada una encargada de un área específica de la búsqueda.

Empezaremos describiendo las fundamentales: la función encargada de ordenar y la función encargada de paginar. Cuando realizamos una consulta a la API bien podemos dejar los parámetros por defecto o configurarlos nosotros mismos en la URL. Los cuatro parámetros disponibles son: límite por página, página, columna a ordenar y dirección de la ordenación. Si dejamos los parámetros por defecto hemos definido un límite de 20 resultados por página y se ordenará por la columna ID de la tabla en orden ascendente. Si, por el contrario, decidimos configurar nosotros mismos la URL, esta será de la siguiente forma:

```
/probe?page=0&limit=10&sort=alias&order=asc
```

Con esta URL obtendremos las sondas ordenadas por la columna alias de forma ascendente, además veremos solo diez resultados de la primera página. Podemos observar los diferentes campos: *page, limit, sort* y *order*.

Tal y como hemos comentado, si no añadimos ningún parámetro obtendremos la búsqueda por defecto, por lo que hemos añadido un parámetro más por si quisiéramos obtener todos los resultados sin paginación. Es el parámetro *all*, y lo podemos usar de la siguiente forma:

```
/probe?all=true
```

A continuación, describiremos la función encargada de buscar por campos de texto usando comodines. En nuestra base de datos tenemos diferentes tipos de datos, muchos de ellos son campos de texto. Si queremos por ejemplo buscar una sonda con un alias, pero no conocemos el alias, si no usáramos comodines tendríamos que saber el alias completo para obtener el resultado esperado. Imaginemos que tenemos tres sondas con los siguientes nombres: *sonda_01*, *sonda_02*, *sonda_14*. Sin usar esta función nuestra búsqueda para obtener la *sonda_01*, debería ser:

```
/probe?alias=sonda_01
```

Pero si no sabemos cuál es el nombre de la sonda completo no podríamos buscarla. Usando la función de búsqueda de texto podríamos buscar de la forma:

```
/probe?alias=sonda_0
```

Esto, en la sentencia SQL se transformaría como:

```
SELECT * FROM probe WHERE alias LIKE %sonda_0%;
```

Con esto conseguiríamos dos resultados: *sonda_01* y *sonda_02*. Está función es una aproximación muy básica a una búsqueda por texto. En un futuro se podría hacer una función más compleja usando índices en la base de datos y aprovechando las funciones nativas de MySql, como la búsqueda *full text*.

Finalmente, comentaremos la función encargada de hacer búsquedas por rangos. Si recordamos la Figura 37, podemos ver que tenemos campos de números enteros, fechas, etc. A diferencia de los campos de tipo texto, en este tipo de campos es absurdo utilizar comodines, porque generalmente no queremos buscar el inicio o final de un número, sino que el número este dentro de un rango. Pongamos como ejemplo la latitud y longitud de una sonda. Sabemos que la latitud puede variar desde 90º a -90º y la longitud desde 180º a -180º. En la Tabla 5 tenemos tres sondas de ejemplo (las coordenadas han sido simplificadas por facilitar la lectura):

Nombre	Latitud	Longitud
sonda_01	10	50
sonda_02	-74	177
sonda_03	15	173

Tabla 5 - Tabla ejemplo coordenadas de sondas (Fuente propia)

Si quisiéramos buscar sondas dentro del rango positivo de latitud y cualquier longitud podríamos realizar la siguiente búsqueda:

```
/probe?latitude=[0,90]&longitude=[-180,180]
```

Mediante esta llamada obtendríamos dos resultados: *sonda_01* y *sonda_03*. Lo que hace la función es interpretar las búsquedas con parámetros con corchetes, y transforma la llamada a la siguiente sentencia SQL:

```
SELECT * FROM probe WHERE latitude BETWEEN 0 AND 90 AND longitude BETWEEN -180 AND
180;
```

8.3 Controlador para conocer el estado de las sondas

A causa de la naturaleza distribuida del sistema, es complicado conocer en tiempo real el estado de una sonda. Esto es debido a que las sondas funcionan con órdenes que enviamos a través de MQTT y las cuales no sabemos si han llegado al destino.

El flujo para conocer el estado de una sonda es el siguiente: Un usuario hace una llamada a la URL para conocer el estado, el controlador entonces envía una petición a la sonda y guarda la petición del usuario en una variable, que se mantiene a la espera. Una vez recibida respuesta de la sonda se finaliza la petición del usuario, enviándole el dato devuelto por la sonda. Si la sonda estuviera desconectada la petición se perdería, porque recordar que estamos usando QoS 0 en nuestro entorno MQTT, por lo que la petición del usuario jamás sería respondida.

La forma que hemos elegido para resolver el problema es la siguiente: si la respuesta de la sonda tarda más de cinco segundos marcamos la petición como no válida e indicamos al usuario que la sonda no está activa.

8.4 Controlador para regenerar la clave de acceso

Las aplicaciones necesitan una clave de acceso a la API para poder hacer peticiones del tipo máquina-máquina. Cuando una aplicación necesita hacer una petición la URL que utilizan es similar a la siguiente:

```
/:apiKey/probe
```

Siendo *apiKey* la variable que contiene la clave de acceso y *probe* el modelo a obtener.

En el nuevo panel de administración será posible regenerar esta clave de acceso de las aplicaciones. Para poder hacerlo se ha creado una ruta y una función asociada que devolverá la nueva clave de acceso. La clave de acceso es un identificador único universal, de la abreviatura en inglés: *UUID*. Un *UUID* es un código hexadecimal de treinta y dos dígitos separados en cinco grupos por guiones, de la siguiente forma: 8-4-4-4-12. En el caso de nuestra clave de acceso, una vez generado el código eliminamos los guiones intermedios. Para generar el código utilizamos la librería de código abierto: *uuid*[12].

8.5 Creación de nuevas rutas en la API

Al comienzo del desarrollo teníamos solo las rutas necesarias para la comunicación entre el controlador de sondas y el *dashboard* de Smart University. Estas rutas consistían en las mostradas en la Tabla 6:

Ruta	/:apiKey/probe
Método	POST
Parámetros	• **apiKey**: Clave de acceso de la aplicación
Cuerpo del mensaje	• **alias**: nombre de la sonda • **geo**: latitud, longitud • **_id**: id de la sonda en el *dashboard* • **citymap**: ubicación de la sonda • **wlanMac**: MAC de la sonda • **timeToSendStatus**: tiempo para el envío del estado

[12] https://www.npmjs.com/package/uuid

	• **mobile**: si es una sonda móvil o no
Descripción	Crear una nueva sonda nueva.
Devuelve	• **201:** La sonda ha sido creada • **409:** Ya existe una sonda con esa MAC o _id

Ruta	/:apiKey/probe/:id
Método	PUT
Parámetros	• **apiKey**: Clave de acceso de la aplicación • **id**: Id de la sonda
Cuerpo del mensaje	• **alias**: nombre de la sonda • **geo**: latitud, longitud • **_id**: id de la sonda en el *dashboard* • **citymap**: ubicación de la sonda • **wlanMac**: MAC de la sonda • **timeToSendStatus**: tiempo para el envío del estado • **mobile**: si es una sonda móvil o no
Descripción	Actualizar una sonda existente.
Devuelve	• **404:** No existe la sonda a actualizar • **200:** La sonda ha sido actualizada

Ruta	/:apiKey/probe/:id
Método	DELETE
Parámetros	• **apiKey**: Clave de acceso de la aplicación • **id**: Id de la sonda
Cuerpo del mensaje	-
Descripción	Eliminar una sonda existente
Devuelve	• **404:** No existe la sonda a eliminar • **204:** La sonda ha sido eliminada

Tabla 6 - Rutas iniciales (Fuente propia)

Para poder comunicar el panel frontal con el servidor hemos tenido que crear nuevas rutas, mediante las cuales se realizan las operaciones básicas sobre los modelos y otras operaciones más complejas. En la tabla número 7 se describen las nuevas rutas añadidas al sistema. Todas las rutas tienen dos variantes: la primera es el acceso mediante el *token* JWT que se incluye en la cabecera de la petición, y la segunda mediante la clave de acceso que se incluye en la ruta antes del modelo principal. Por ejemplo, la ruta para obtener las sondas tiene las siguientes variantes: `/probe` y `/:apiKey/probe`.

Por otro lado, no se han incluido en la tabla los errores del servidor, que pueden suceder en todas las rutas y devuelven un código de la serie 500 con el nombre y la descripción del error.

Ruta	/auth/local
Método	POST
Parámetros	
Cuerpo del mensaje	• **email:** correo del usuario • **password:** contraseña del usuario
Descripción	Login en la aplicación
Respuesta	• **401:** Las credenciales no son correctas • **202:** Las credenciales son correctas. Devuelve el JWT en el cuerpo.

Ruta	/user o /:apiKey/user
Método	POST
Parámetros	
Cuerpo del mensaje	• **email:** correo del usuario • **password:** contraseña del usuario • **name:** nombre del usuario • **surname:** apellido del usuario • **application:** aplicación a la que pertenece el usuario
Descripción	Crear un usuario nuevo
Respuesta	• **409:** El correo no está disponible • **201:** El usuario ha sido creado.

Ruta	/user o /:apiKey/user
Método	GET
Parámetros	• **page:** Página de resultados • **limit:** Elementos por página • **sort:** Columna a ordenar • **order:** Dirección de la ordenación
Cuerpo del mensaje	
Descripción	Obtener el listado de usuarios
Respuesta	• **200:** Devuelve el listado de usuarios

Ruta	/user/count o /:apiKey/user/count

Método	GET
Parámetros	• **page:** Página de resultados • **limit:** Elementos por página • **sort:** Columna a ordenar • **order:** Dirección de la ordenación
Cuerpo del mensaje	
Descripción	Obtiene el conteo de usuarios
Respuesta	• **200:** Devuelve el número de usuarios

Ruta	/user/:id o /:apiKey/user/:id
Método	GET
Parámetros	• **id:** id del usuario
Cuerpo del mensaje	
Descripción	Obtiene el usuario concreto
Respuesta	• **200:** Devuelve el objeto usuario

Ruta	/user/:id o /:apiKey/user/:id
Método	PUT
Parámetros	• **id:** id del usuario
Cuerpo del mensaje	• **password:** contraseña del usuario • **name:** nombre del usuario • **surname:** apellido del usuario • **application:** aplicación a la que pertenece el usuario
Descripción	Actualiza el usuario
Respuesta	• **201:** El usuario ha sido actualizado • **404:** No existe el usuario a actualizar

Ruta	/user/:id o /:apiKey/user/:id
Método	DELETE
Parámetros	• **id:** id del usuario
Cuerpo del mensaje	
Descripción	Elimina el usuario
Respuesta	• **204:** El usuario ha sido eliminado • **404:** No existe el usuario a eliminar

Ruta	/sensor o /:apiKey/sensor
Método	POST
Parámetros	
Cuerpo del mensaje	• **sensorType:** tipo de sensor • **alias:** alias del sensor • **probe:** sonda a la que pertenece el sensor • **interval:** intervalo de tiempo para realizar la petición • **watcher:** si es un sensor de tipo watcher • **reactive:** si es un sensor de tipo reactivo • **reactiveSpace:** umbral del sensor reactivo
Descripción	Crear un sensor nuevo
Respuesta	• **201:** El sensor ha sido creado.

Ruta	/sensor o /:apiKey/sensor
Método	GET
Parámetros	• **page:** Página de resultados • **limit:** Elementos por página • **sort:** Columna a ordenar • **order:** Dirección de la ordenación
Cuerpo del mensaje	
Descripción	Obtener el listado de sensores
Respuesta	• **200:** Devuelve el listado de sensores

Ruta	/sensor/count o /:apiKey/sensor/count
Método	GET
Parámetros	• **page:** Página de resultados • **limit:** Elementos por página • **sort:** Columna a ordenar • **order:** Dirección de la ordenación
Cuerpo del mensaje	
Descripción	Obtiene el conteo de sensores
Respuesta	• **200:** Devuelve el número de sensores

Ruta	/sensor/:id o /:apiKey/sensor/:id
Método	GET
Parámetros	• **id:** id del sensor
Cuerpo del mensaje	

Descripción	Obtiene el sensor concreto
Respuesta	• **200:** Devuelve el objeto sensor

Ruta	/sensor/:id o /:apiKey/sensor/:id
Método	PUT
Parámetros	• **id:** id del sensor
Cuerpo del mensaje	• **sensorType:** tipo de sensor • **alias:** alias del sensor • **probe:** sonda a la que pertenece el sensor • **interval:** intervalo de tiempo para realizar la petición • **watcher:** si es un sensor de tipo watcher • **reactive:** si es un sensor de tipo reactivo • **reactiveSpace:** umbral del sensor reactivo
Descripción	Actualiza el sensor
Respuesta	• **201:** El sensor ha sido actualizado • **404:** No existe el sensor a actualizar

Ruta	/sensor/:id o /:apiKey/sensor/:id
Método	DELETE
Parámetros	• **id:** id del sensor
Cuerpo del mensaje	
Descripción	Elimina el sensor
Respuesta	• **204:** El sensor ha sido eliminado • **404:** No existe el sensor a eliminar

Ruta	/watcher-log o /:apiKey/watcher-log
Método	GET
Parámetros	• **page:** Página de resultados • **limit:** Elementos por página • **sort:** Columna a ordenar • **order:** Dirección de la ordenación
Cuerpo del mensaje	
Descripción	Obtener el listado de peticiones a los sensores
Respuesta	• **200:** Devuelve el listado de peticiones a los sensores

Ruta	/watcher-log/count o /:apiKey/watcher-log/count
Método	GET
Parámetros	• **page:** Página de resultados • **limit:** Elementos por página • **sort:** Columna a ordenar • **order:** Dirección de la ordenación
Cuerpo del mensaje	
Descripción	Obtiene el conteo de peticiones a los sensores
Respuesta	• **200:** Devuelve el número de peticiones a los sensores

Ruta	/watcher-log/:id o /:apiKey/watcher-log/:id
Método	GET
Parámetros	• **id:** id de la petición
Cuerpo del mensaje	
Descripción	Obtiene la petición concreta
Respuesta	• **200:** Devuelve el objeto petición

Ruta	/sensor-type o /:apiKey/sensor-type
Método	POST
Parámetros	
Cuerpo del mensaje	• **id:** nombre del tipo de sensor e identificador • **isPython:** si el script del tipo de sensor está escrito en Python o JavaScript • **script:** cadena de texto con el script necesario para leer el sensor
Descripción	Crear un tipo de sensor nuevo
Respuesta	• **201:** El tipo de sensor ha sido creado

Ruta	/sensor-type o /:apiKey/sensor-type
Método	GET
Parámetros	• **page:** Página de resultados • **limit:** Elementos por página • **sort:** Columna a ordenar • **order:** Dirección de la ordenación
Cuerpo del mensaje	
Descripción	Obtener el listado de tipos de sensores

Respuesta	• **200:** Devuelve el listado de tipos de sensores

Ruta	/sensor-type/count o /:apiKey/sensor-type/count
Método	GET
Parámetros	• **page:** Página de resultados • **limit:** Elementos por página • **sort:** Columna a ordenar • **order:** Dirección de la ordenación
Cuerpo del mensaje	
Descripción	Obtiene el conteo de tipos de sensores
Respuesta	• **200:** Devuelve el número de tipos de sensores

Ruta	/sensor-type/count-by-sensor o /:apiKey/sensor-type/count-by-sensor
Método	GET
Parámetros	• **page:** Página de resultados • **limit:** Elementos por página • **sort:** Columna a ordenar • **order:** Dirección de la ordenación
Cuerpo del mensaje	
Descripción	Obtiene el conteo de sensores por cada tipo de sensor
Respuesta	• **200:** Devuelve el número de sensores por cada tipo de sensor

Ruta	/sensor-type/:id o /:apiKey/sensor-type/:id
Método	GET
Parámetros	• **id:** id del tipo de sensor
Cuerpo del mensaje	
Descripción	Obtiene el tipo de sensor concreto
Respuesta	• **200:** Devuelve el objeto del tipo de sensor

Ruta	/sensor-type/:id o /:apiKey/sensor-type/:id
Método	PUT
Parámetros	• **id:** id del tipo de sensor
Cuerpo del mensaje	• **isPython:** si el script del tipo de sensor está escrito en Python o JavaScript • **script:** cadena de texto con el script necesario para

	leer el sensor.
Descripción	Actualiza el tipo sensor
Respuesta	• **201:** El tipo de sensor ha sido actualizado • **404:** No existe el tipo de sensor a actualizar

Ruta	/sensor-type/:id o /:apiKey/sensor-type/:id
Método	DELETE
Parámetros	• **id:** id del tipo de sensor
Cuerpo del mensaje	
Descripción	Elimina el tipo de sensor
Respuesta	• **204:** El tipo de sensor ha sido eliminado • **404:** No existe el tipo de sensor a eliminar

Ruta	/application o /:apiKey/application
Método	POST
Parámetros	
Cuerpo del mensaje	• **id:** nombre de la aplicación e identificador
Descripción	Crear una aplicación nueva
Respuesta	• **201:** La aplicación ha sido creada. Devuelve la clave de acceso

Ruta	/application o /:apiKey/application
Método	GET
Parámetros	• **page:** Página de resultados • **limit:** Elementos por página • **sort:** Columna a ordenar • **order:** Dirección de la ordenación
Cuerpo del mensaje	
Descripción	Obtener el listado de aplicaciones
Respuesta	• **200:** Devuelve el listado de aplicaciones

Ruta	/application/count o /:apiKey/application/count
Método	GET
Parámetros	• **page:** Página de resultados • **limit:** Elementos por página • **sort:** Columna a ordenar

	• **order:** Dirección de la ordenación
Cuerpo del mensaje	
Descripción	Obtiene el conteo de aplicaciones
Respuesta	• **200:** Devuelve el número de aplicaciones

Ruta	/application/:id o /:apiKey/application/:id
Método	GET
Parámetros	• **id:** id de la aplicación
Cuerpo del mensaje	
Descripción	Obtiene la aplicación concreta
Respuesta	• **200:** Devuelve el objeto aplicación

Ruta	/application/:id o /:apiKey/application/:id
Método	PUT
Parámetros	• **id:** id de la aplicación
Cuerpo del mensaje	• **id:** nombre de la aplicación e identificador
Descripción	Actualiza la aplicación
Respuesta	• **201:** La aplicación ha sido actualizado • **404:** No existe la aplicación a actualizar

Ruta	/application/:id o /:apiKey/application/:id
Método	DELETE
Parámetros	• **id:** id de la aplicación
Cuerpo del mensaje	
Descripción	Elimina la aplicación
Respuesta	• **204:** La aplicación ha sido eliminado • **404:** No existe la aplicación a eliminar

Ruta	/application/:id/regenerate-key o /:apiKey/application/:id/regenerate-key
Método	GET
Parámetros	• **id:** id de la aplicación
Cuerpo del mensaje	
Descripción	Regenera la clave de acceso de la aplicación
Respuesta	• **200:** Devuelve la nueva clave de acceso • **404:** No existe una aplicación con el id solicitado

Ruta	/probe o /:apiKey/probe
Método	POST
Parámetros	
Cuerpo del mensaje	• **alias**: nombre de la sonda • **geo**: latitud, longitud • **_id**: id de la sonda en el *dashboard* • **citymap**: ubicación de la sonda • **wlanMac**: MAC de la sonda • **timeToSendStatus**: tiempo para el envío del estado • **mobile**: si es una sonda móvil o no
Descripción	Crear una sonda nueva
Respuesta	• **201:** La sonda ha sido creada

Ruta	/probe o /:apiKey/probe
Método	GET
Parámetros	• **page:** Página de resultados • **limit:** Elementos por página • **sort:** Columna a ordenar • **order:** Dirección de la ordenación
Cuerpo del mensaje	
Descripción	Obtener el listado de sondas
Respuesta	• **200:** Devuelve el listado de sondas

Ruta	/probe/count o /:apiKey/probe/count
Método	GET
Parámetros	• **page:** Página de resultados • **limit:** Elementos por página • **sort:** Columna a ordenar • **order:** Dirección de la ordenación
Cuerpo del mensaje	
Descripción	Obtiene el conteo de sondas
Respuesta	• **200:** Devuelve el número de aplicaciones

Ruta	/probe/:id o /:apiKey/probe/:id
Método	GET
Parámetros	• **id:** id de la sonda

Cuerpo del mensaje	
Descripción	Obtiene la sonda concreta
Respuesta	• **200:** Devuelve el objeto sonda

Ruta	/probe/:id o /:apiKey/probe/:id
Método	PUT
Parámetros	• **id:** id de la sonda
Cuerpo del mensaje	• **alias**: nombre de la sonda • **geo**: latitud, longitud • **_id**: id de la sonda en el *dashboard* • **citymap**: ubicación de la sonda • **wlanMac**: MAC de la sonda • **timeToSendStatus**: tiempo para el envío del estado • **mobile**: si es una sonda móvil o no
Descripción	Actualiza la sonda
Respuesta	• **201:** La sonda ha sido actualizado • **404:** No existe la sonda a actualizar

Ruta	/probe/:id o /:apiKey/probe/:id
Método	DELETE
Parámetros	• **id:** id de la sonda
Cuerpo del mensaje	
Descripción	Elimina la sonda
Respuesta	• **204:** La sonda ha sido eliminado • **404:** No existe la sonda a eliminar

Ruta	/probe/:id/probe-status o /:apiKey/probe/:id/probe-status
Método	GET
Parámetros	• **id:** id de la sonda • **page:** Página de resultados • **limit:** Elementos por página • **sort:** Columna a ordenar • **order:** Dirección de la ordenación
Cuerpo del mensaje	
Descripción	Obtiene el listado de los estados de la sonda
Respuesta	• **200:** Devuelve el listado de estados • **404:** No existe una sonda con el id solicitado

Ruta	/probe/:id/probe-status/latest o /:apiKey/probe/:id/probe-status/latest
Método	GET
Parámetros	• **id:** id de la sonda
Cuerpo del mensaje	
Descripción	Solicita un estado nuevo a la sonda
Respuesta	• **200:** Devuelve el estado actualizado a la sonda • **404:** No existe una sonda con el id solicitado • **408:** El tiempo de espera se ha agotado

Ruta	/probe/:id/probe-status/count o /:apiKey/probe/:id/probe-status/count
Método	GET
Parámetros	• **id:** id de la sonda • **page:** Página de resultados • **limit:** Elementos por página • **sort:** Columna a ordenar • **order:** Dirección de la ordenación
Cuerpo del mensaje	
Descripción	Obtiene el conteo de estados de la sonda
Respuesta	• **200:** Devuelve el conteo de estados de la sonda • **404:** No existe una sonda con el id solicitado

Ruta	/probe/:id/firmware o /:apiKey/probe/:id/firmware
Método	GET
Parámetros	• **id:** id de la sonda
Cuerpo del mensaje	
Descripción	Envía una petición de reinicio a la sonda para que actualice su software interno
Respuesta	• **200:** Petición enviada correctamente • **404:** No existe una sonda con el id solicitado

Ruta	/probe/:id/watcher-log o /:apiKey/probe/:id/watcher-log
Método	GET
Parámetros	• **id:** id de la sonda • **page:** Página de resultados • **limit:** Elementos por página

	• **sort:** Columna a ordenar • **order:** Dirección de la ordenación
Cuerpo del mensaje	
Descripción	Obtiene las peticiones enviadas a sensores de la sonda
Respuesta	• **200:** Listado de peticiones enviadas a sensores de la sonda • **404:** No existe una sonda con el id solicitado

Ruta	/probe/:id/watcher-log/count o /:apiKey/:id/watcher-log/count
Método	GET
Parámetros	• **id:** id de la sonda • **page:** Página de resultados • **limit:** Elementos por página • **sort:** Columna a ordenar • **order:** Dirección de la ordenación
Cuerpo del mensaje	
Descripción	Obtiene el conteo las peticiones enviadas a sensores de la sonda
Respuesta	• **200:** Devuelve el conteo las peticiones enviadas a sensores de la sonda • **404:** No existe una sonda con el id solicitado

Tabla 7 - Rutas finales del servidor (Fuente propia)

Con cada ruta se asocia un controlador que ejecuta las órdenes necesarias para realizar las operaciones solicitadas. A un nivel más bajo que el controlador, y como vimos en la Figura 35, existe un servicio, que utiliza las librerías *KnexJs*[13] y *BookshelfJs*[14], mediante el cual se hace un mapeado de la base de datos relacional a objetos nativos de JavaScript e igual a la inversa. De esta forma no tenemos que escribir las sentencias SQL directamente en el código, evitando errores y posibles fallos de seguridad. Aunque, en algunas ocasiones para hacer operaciones más complicadas hemos tenido que usar directamente SQL, como por ejemplo para conseguir la sentencia que devuelve el número de las peticiones enviadas a sensores de una sonda.

[13] http://knexjs.org/
[14] http://bookshelfjs.org/

9 Implementación del cliente

En el capítulo de diseño hablamos sobre la estructura recomendada por el equipo de Angular 2 para el desarrollo de proyectos, pero sin entrar en la estructura concreta que tendría nuestro proyecto. En esta sección iremos viendo los diferentes niveles dentro de nuestra aplicación final y la estructura del proyecto desde el nivel más alto al más bajo. Además, también se incluirán capturas de las interfaces implementadas.

9.1 Módulo base

El módulo base es el módulo que engloba el resto de la aplicación. Dentro de este módulo encontramos los componentes base, que podrán usar los otros módulos incluidos o funcionar por sí mismos. Además, también contiene una serie de servicios, el enrutador base y el resto de módulos.

En cuanto a componentes hemos implementado los siguientes:

- **NavbarComponent:** Se trata de la barra de navegación lateral de la plantilla base.

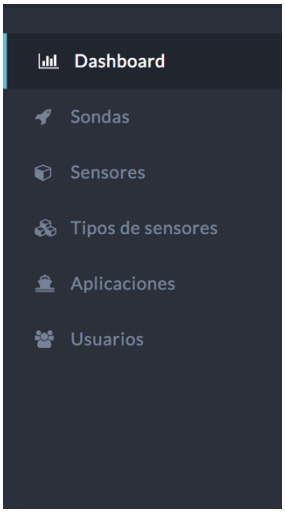


Figura 47 - NavbarComponent (Fuente propia)

- **TopNavbarComponent:** Consiste en la barra de navegación superior de la plantilla base.

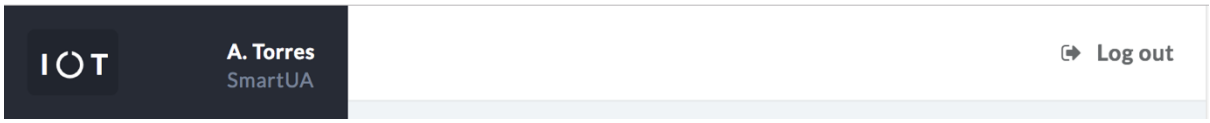


Figura 48 - TopNavbarComponent (Fuente propia)

- **LayoutComponent**: Este componente es el que engloba la plantilla base, dentro de este componente se incluyen los componentes NavbarComponent y TopNavbarComponent. Además, también se encuentra un área donde se mostrarán el resto de módulos y componentes. En la Figura 39 podemos ver una representación del componente.
- **DashboardComponent:** Es el componente que constituye el *dashboard* o página principal. Es la página en la que se puede ver el resumen del sistema. En primer lugar, disponemos de un selector de fecha inicial y final para mostrar los datos del panel comprendidos en un periodo de tiempo. Además, este componente dispone de dos secciones más, por un lado, tenemos los contadores globales y por el otro lado tres gráficas de interés. Los contadores globales nos dan información sobre el número de datos disponible de cada modelo (sondas, sensores, tipo de sensores y peticiones a los sensores). Además, incluyen una pequeña gráfica que indican la tendencia de creación de cada uno de los modelos en las fechas filtradas. Las gráficas inferiores dan información sobre los sensores que existen por cada tipo de sensor; la relación de éxito de las peticiones de lectura a los sensores, y las sondas con mayor actividad de peticiones.

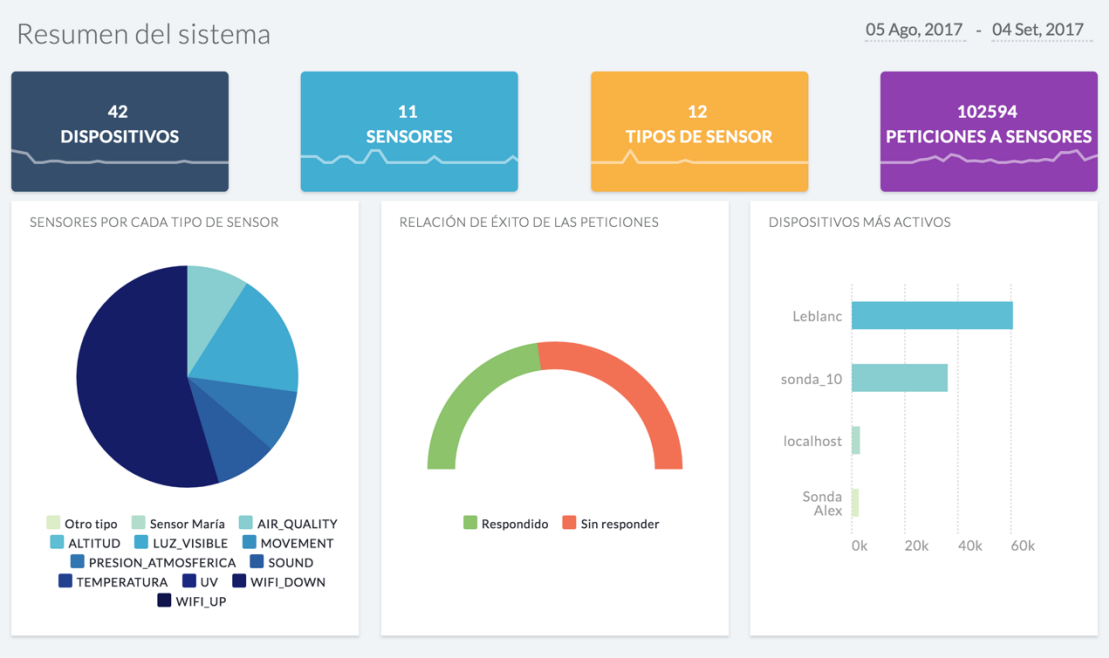


Figura 49 - DashboardComponent (Fuente propia)

- **BigChartComponent:** Este componente permite instanciar gráficas de datos mediante una serie de parámetros. Es totalmente genérico, por lo que podemos instanciar gráficas de tipo lineales, barras, etc.

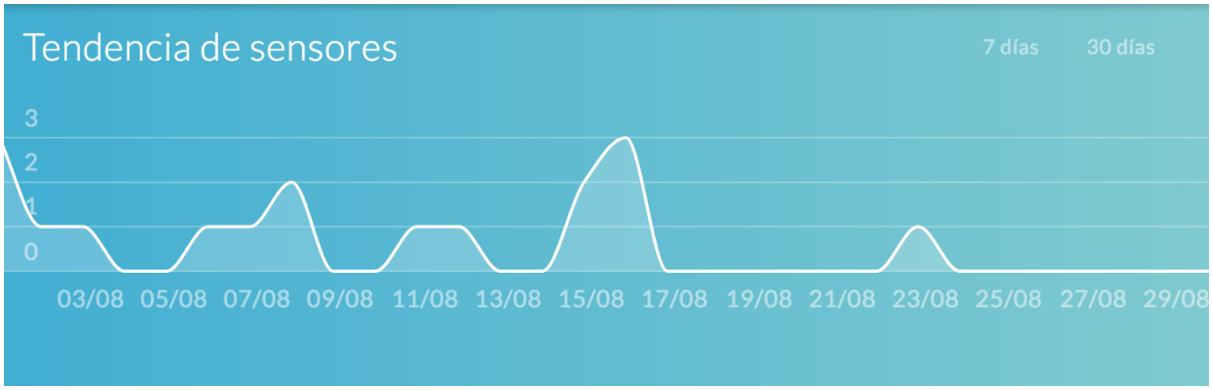
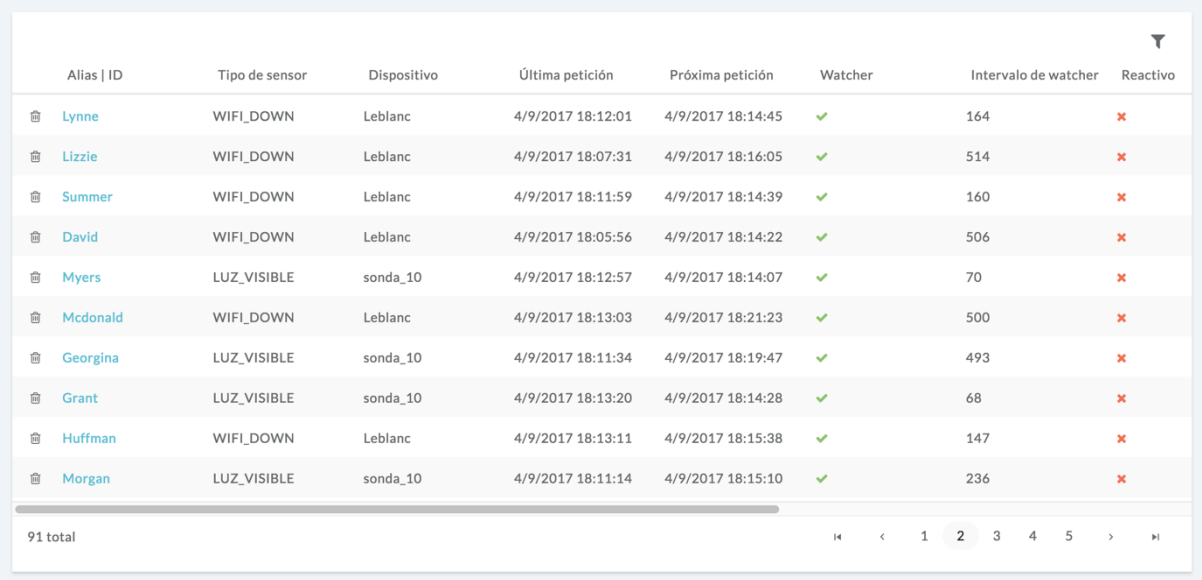


Figura 50 - BigChartComponent (Fuente propia)

- **DatatableComponent:** Es el componente encargado de mostrar datos en tablas, así como de los elementos de paginación y ordenación de la misma.



Alias | ID	Tipo de sensor	Dispositivo	Última petición	Próxima petición	Watcher	Intervalo de watcher	Reactivo
Lynne	WIFI_DOWN	Leblanc	4/9/2017 18:12:01	4/9/2017 18:14:45	✔	164	✖
Lizzie	WIFI_DOWN	Leblanc	4/9/2017 18:07:31	4/9/2017 18:16:05	✔	514	✖
Summer	WIFI_DOWN	Leblanc	4/9/2017 18:11:59	4/9/2017 18:14:39	✔	160	✖
David	WIFI_DOWN	Leblanc	4/9/2017 18:05:56	4/9/2017 18:14:22	✔	506	✖
Myers	LUZ_VISIBLE	sonda_10	4/9/2017 18:12:57	4/9/2017 18:14:07	✔	70	✖
Mcdonald	WIFI_DOWN	Leblanc	4/9/2017 18:13:03	4/9/2017 18:21:23	✔	500	✖
Georgina	LUZ_VISIBLE	sonda_10	4/9/2017 18:11:34	4/9/2017 18:19:47	✔	493	✖
Grant	LUZ_VISIBLE	sonda_10	4/9/2017 18:13:20	4/9/2017 18:14:28	✔	68	✖
Huffman	WIFI_DOWN	Leblanc	4/9/2017 18:13:11	4/9/2017 18:15:38	✔	147	✖
Morgan	LUZ_VISIBLE	sonda_10	4/9/2017 18:11:14	4/9/2017 18:15:10	✔	236	✖

91 total 1 2 3 4 5

Figura 51 - DatatableComponent (Fuente propia)

- **FiltersComponent:** Es la pantalla de filtrado de datos. Mediante este componente podemos filtrar los datos que se muestran en las tablas y gráficas.

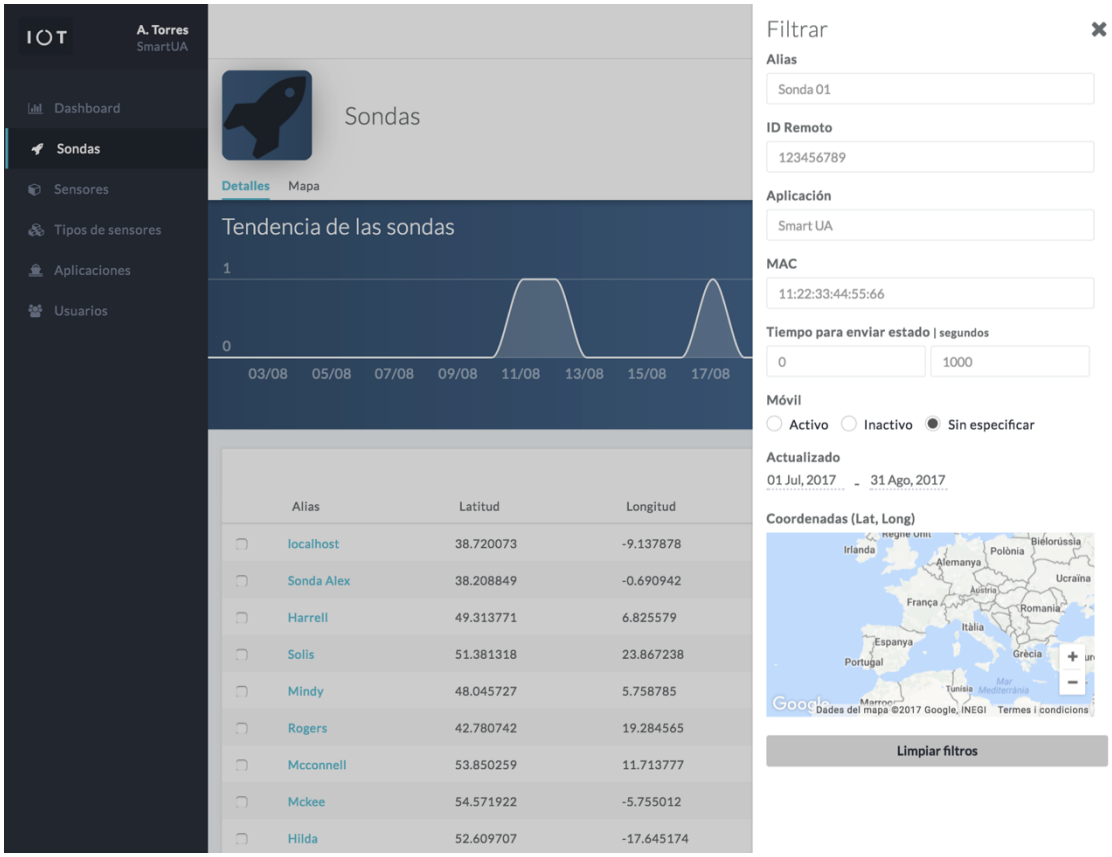


Figura 52 - FiltersComponent (Fuente propia)

- **FormComponent:** Este componente instancia formularios, y se usa en las vistas de crear nuevos elementos como en las de editar los mismos.

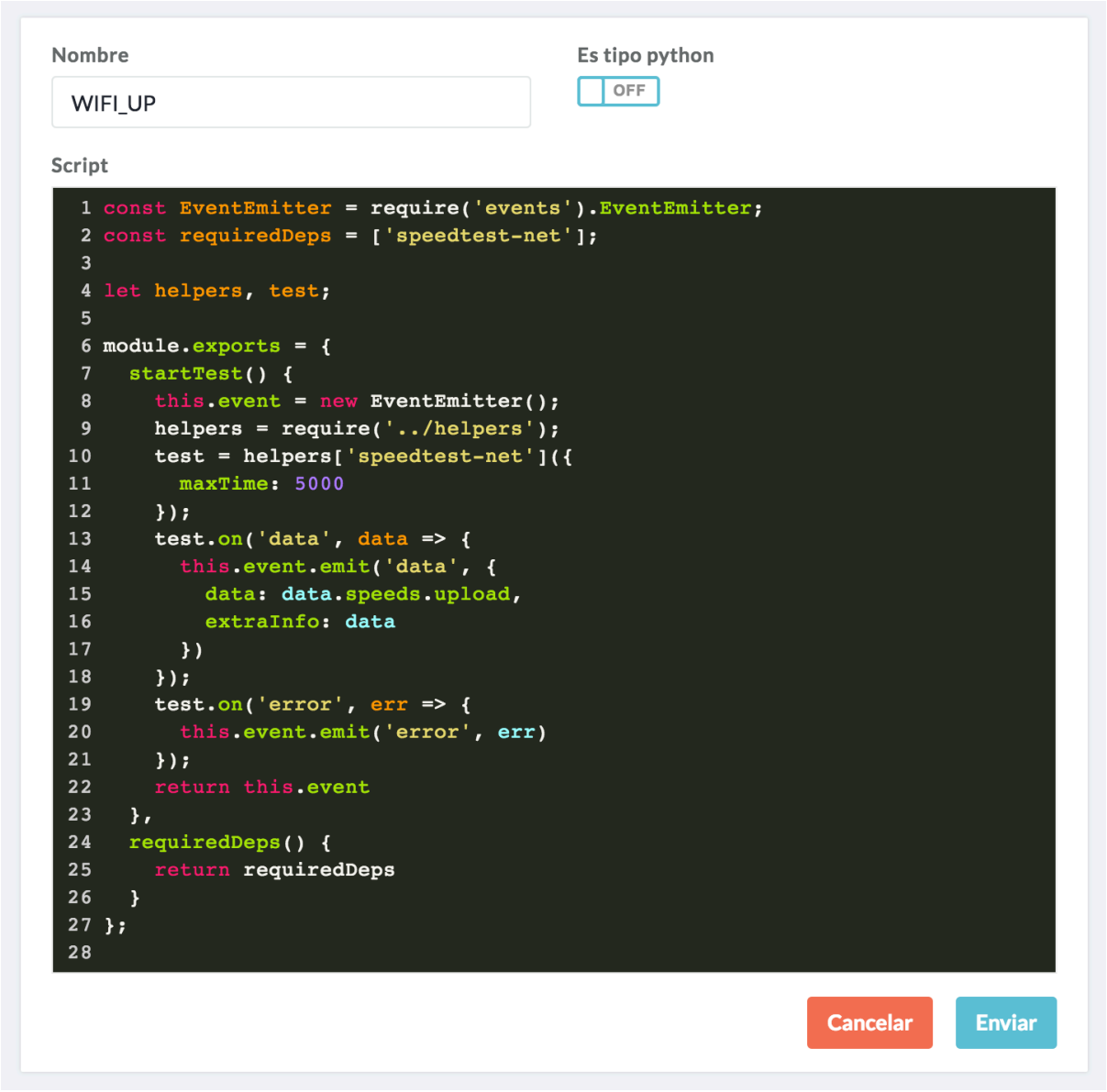


Figura 53 - FormComponent (Fuente propia)

- **DetailViewComponent:** Es el componente mediante el cual se muestra la vista de detalle de algunos modelos de forma genérica.

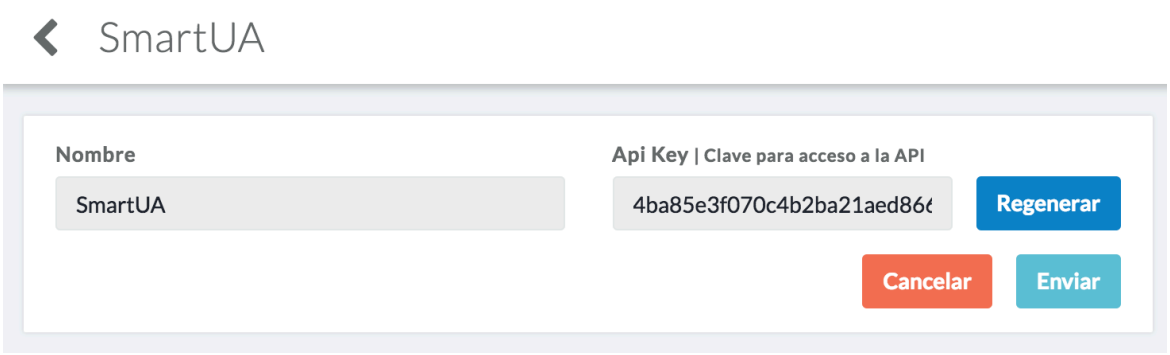


Figura 54 - DetailViewComponent (Fuente propia)

- **NewItemComponent:** Este componente usa el componente FormComponent para generar la vista para crear nuevos modelos de datos.

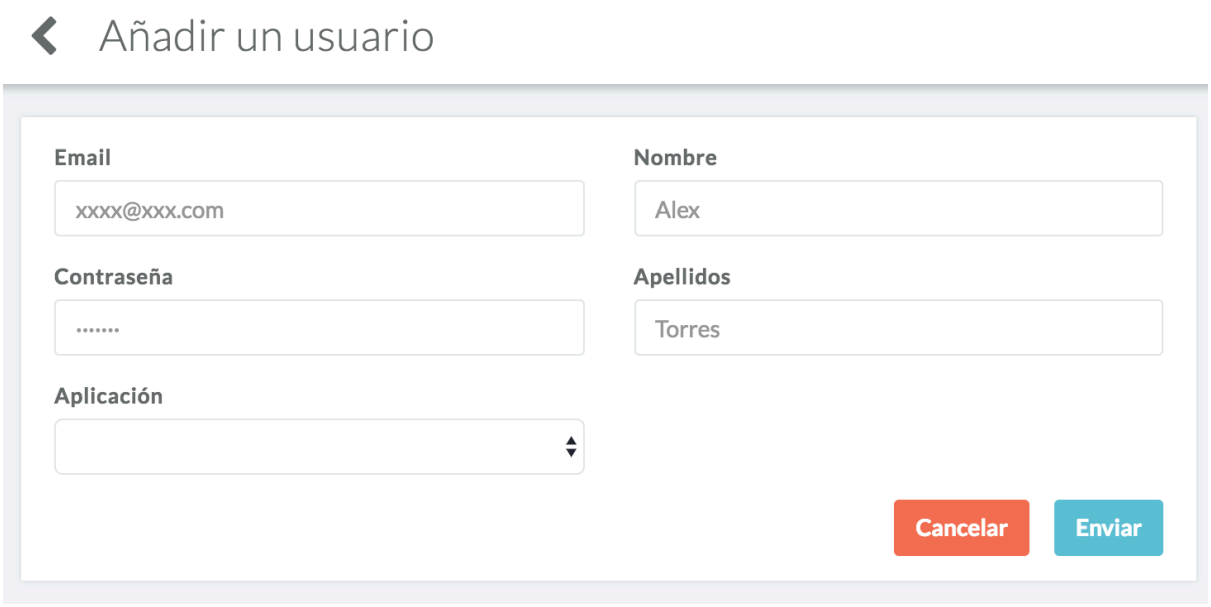


Figura 55 - NewItemComponent (Fuente propia)

En cuanto a servicios, disponemos de dos a nivel global:

- **AuthService:** Se trata del servicio encargado de toda la parte de autenticación del usuario. En él se guarda la sesión del usuario en el *sessionStorage* del navegador. También sirve de interfaz con el *sessionStorage* implementando métodos para recuperar datos como el usuario en sesión o el *token* del usuario. Además, también se usa para saber si el usuario está autenticado en el sistema, borrar los datos del usuario y que el mismo salga o se autentique en el sistema.

- **ApiService:** En cuanto a este servicio, tal como el nombre indica, se trata de una serie de métodos creados para conectar con el *backend*, y en última instancia funciona como interfaz que se comunica con el servicio nativo de Angular para hacer peticiones HTTP. En él se añaden las cabeceras necesarias para hacer peticiones autenticadas usando el *token* del usuario, así como otras cabeceras necesarias para la comunicación con nuestra API REST.

Finalmente, también en este nivel hemos implementado el enrutador a nivel global. Este enrutador nos servirá para definir las rutas bases, rutas directas a componentes y rutas que llevarán el flujo de la aplicación a un módulo interno. Las rutas definidas son las siguientes:

- `[AUTH]` "/dashboard": Mostramos el resumen del sistema y el DashboardComponent.
- `[AUTH]` `[MODULE]` "/probes": Mostramos la página de sondas.
- `[AUTH]` `[MODULE]` "/sensors": Mostramos la página de sensores.
- `[AUTH]` `[MODULE]` "/sensor-types": Mostramos la página de tipos de sensores.
- `[AUTH]` `[MODULE]` "/applications": Mostramos la página de aplicaciones.
- `[AUTH]` `[MODULE]` "/users": Mostramos la página de usuarios.
- `[MODULE]` "/login": Mostramos la página de login de usuarios.
- "*": Cualquier ruta que no coincida se redirige a "/dashboard"

Las rutas marcadas con "AUTH" indican que solo los usuarios registrados en la aplicación pueden acceder. Si se intenta entrar en cualquiera de estas rutas sin estar autenticado en el sistema el usuario será redireccionado a la página de login. Además, las rutas marcadas con "MODULE" indican que son módulos internos del módulo base.

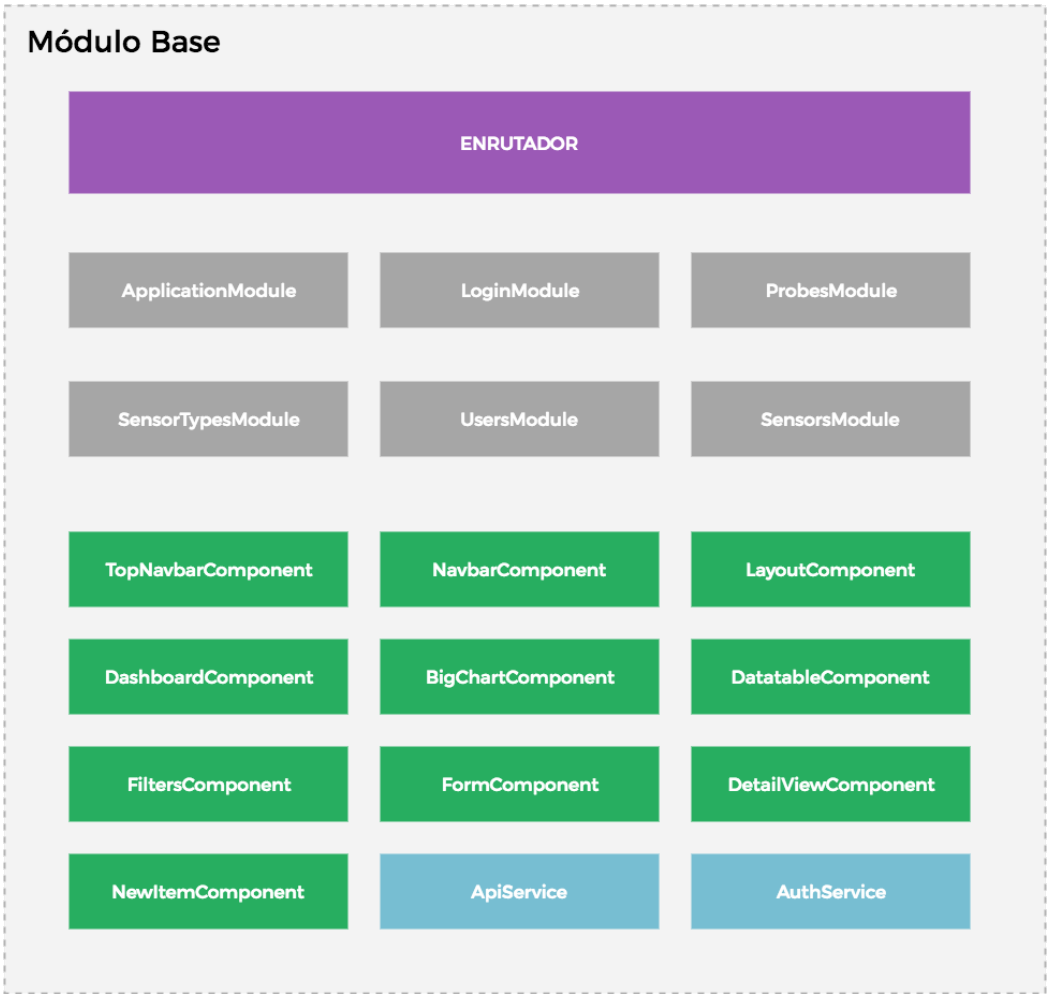


Figura 56 - Arquitectura cliente del módulo base (Fuente propia)

Todos los módulos internos del módulo base están compuestos igualmente por un enrutador propio, y una serie de componentes y servicios. En las siguientes secciones se detallarán los elementos que componen los módulos internos.

9.2 Módulo de sondas (probeModule)

Es el módulo encargado de mostrar la tabla con el listado completo de sondas, así como un mapa de las sondas y la vista detalle de cada sonda. Dentro del módulo diferenciamos los siguientes componentes:

- **ProbesComponent:** Es el componente encargado de mostrar el listado de sondas. Además del listado en una tabla también se puede mostrar una vista para ubicar fácilmente todas las sondas en un mapa.

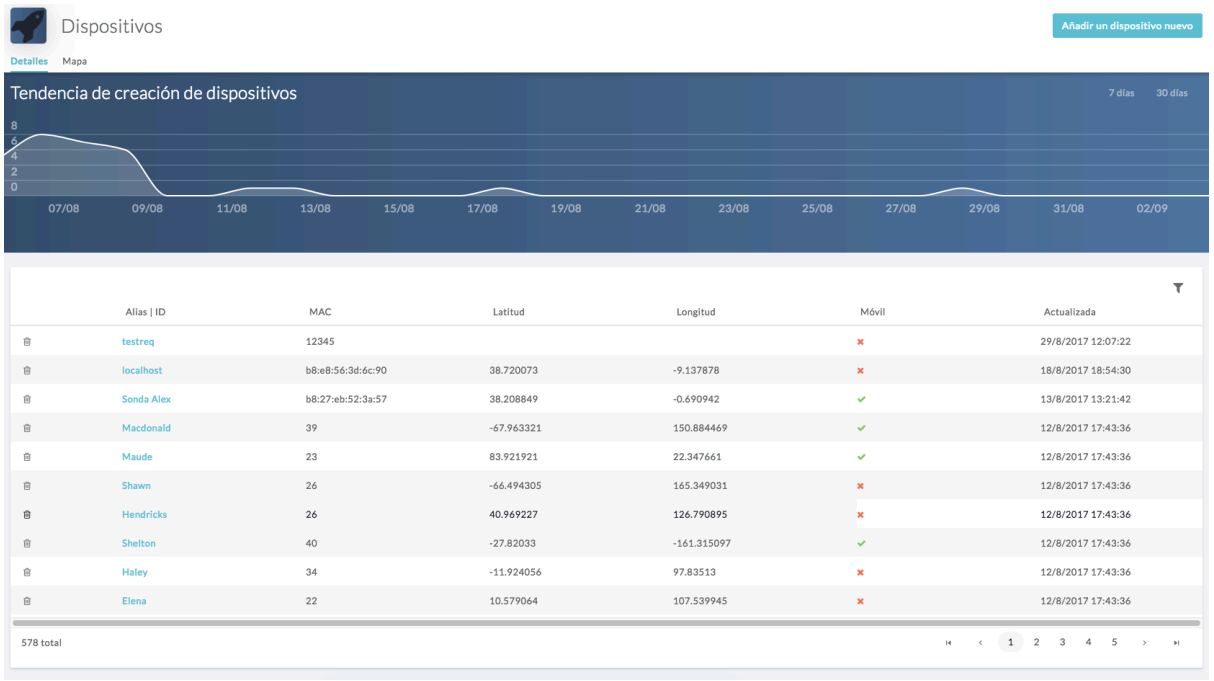


Figura 57 - ProbesComponent (Fuente propia)

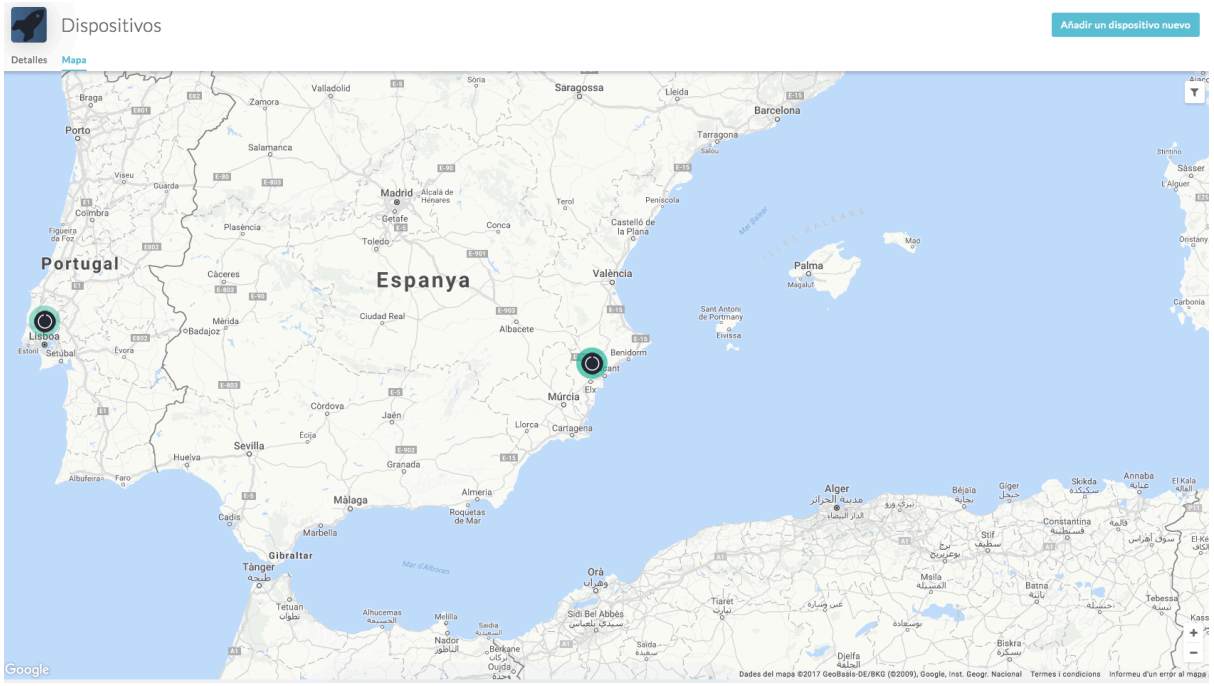


Figura 58 - Vista mapa de sondas (Fuente propia)

- **ProbeDetailViewComponent:** Se encarga de mostrar los detalles de las sondas. Es uno de los componentes más completos. En él se muestra una gráfica que representa el estado de la sonda con diferentes métricas relacionadas con la misma. Además, una tabla

de los sensores que tiene, desde donde se puede añadir un sensor nuevo a la misma, y un formulario donde editar los datos propios de la sonda.

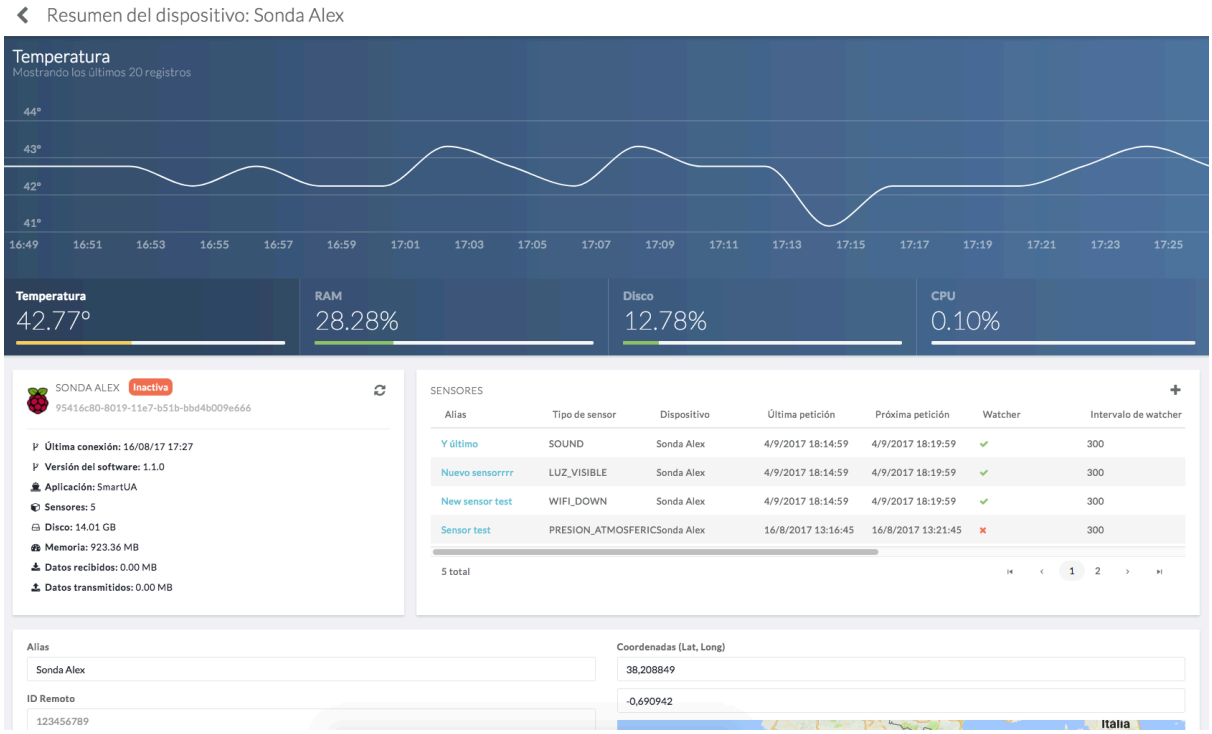


Figura 59 - ProbeDetailViewComponent (Fuente propia)

9.3 Módulo de tipos de sensores (sensorTypesModule)

En este módulo se puede gestionar todo lo relacionado con los tipos de sensores. En el caso del módulo de tipos de sensores, al igual que sucede con el de aplicaciones y usuarios, tenemos un componente principal para listar los tipos de sensores y hacemos uso del componente genérico DetailViewComponent para mostrar la vista detalle de cada tipo con más detalle.

- **SensorTypesComponent:** Este componente es el principal del módulo. En él se listan los diferentes tipos de sensores. Además, se muestra una gráfica en la que se puede observar la tendencia de creación de nuevos sensores de cada tipo de sensor existente.

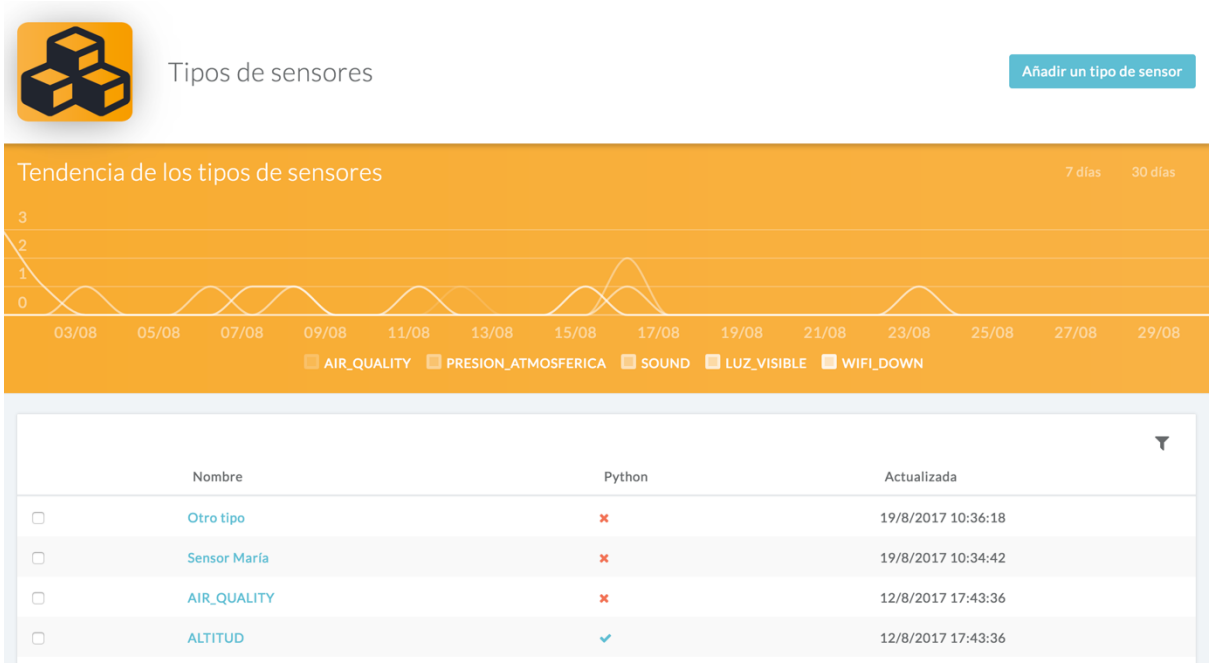


Figura 60 - SensorTypesComponent (Fuente propia)

- **DetailViewComponent:** En el caso de los tipos de sensores, hemos modificado este componente genérico para incluir un formulario en el que podemos editar los scripts de los sensores mediante un editor integrado con resaltado de sintaxis.

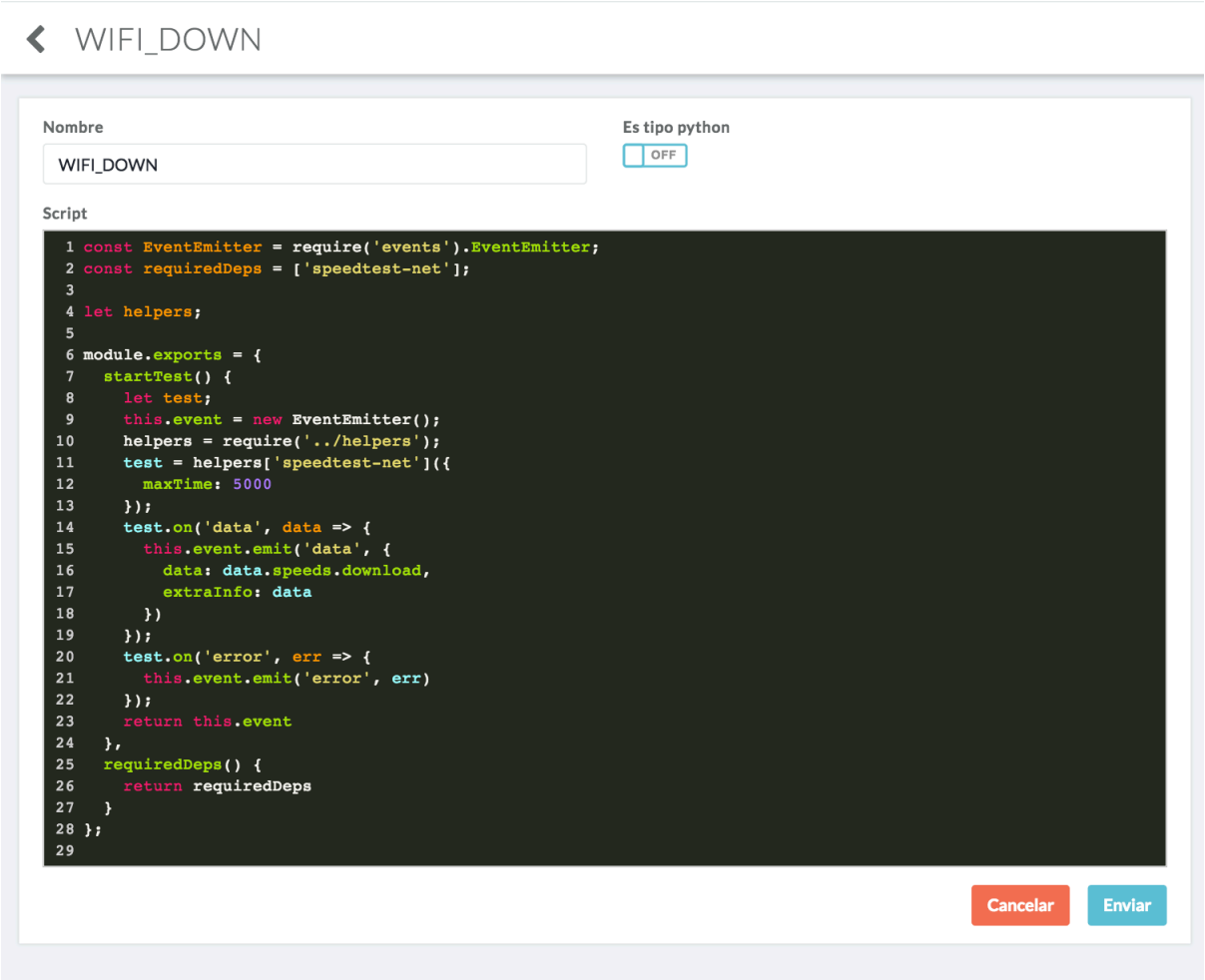


Figura 61 - Vista detalle de tipo de sensor (Fuente propia)

9.4 Módulo de sensores (sensorsModule)

El módulo de sensores nos permite analizar el desempeño global de los sensores, así como de cada sensor en particular.

- **SensorsComponent:** En esta vista se muestra una gráfica que muestra la tendencia de creación de sensores y una tabla con todos los sensores creados.

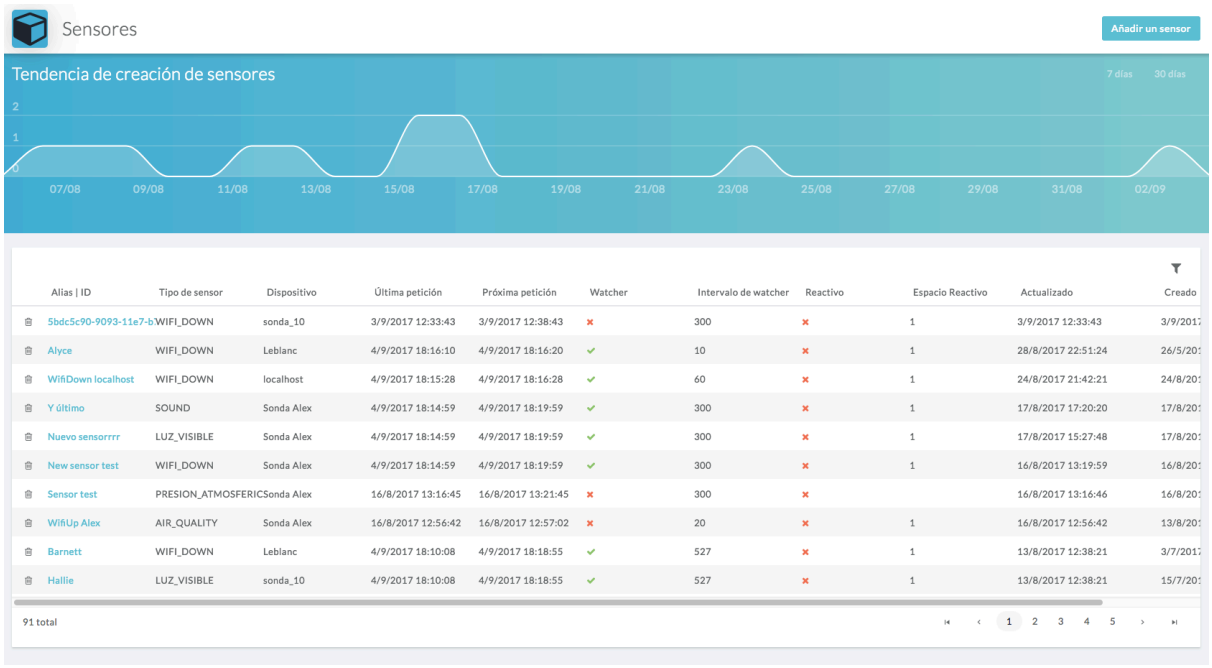


Figura 62 - SensorsComponent (Fuente propia)

- **SensorDetailViewComponent:** La vista de detalle de sensor también es una de las más completas. Permite ver el estado del sensor, y hacer una comparativa del éxito de las peticiones realizadas al sensor. Dispone de una gráfica en tiempo real en la que se pueden observar las peticiones que se realizan al sensor. También se pueden editar los detalles del sensor.

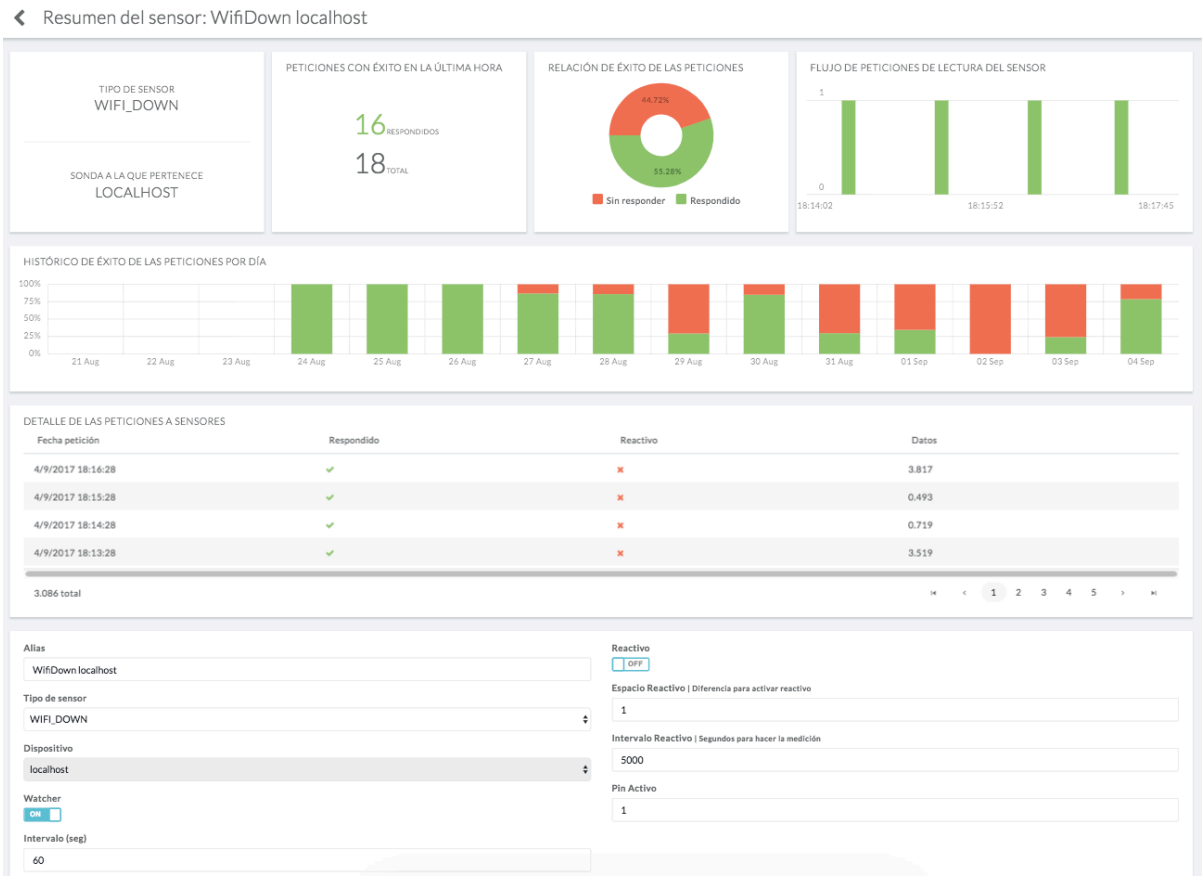


Figura 63 - SensorDetailViewComponent (Fuente propia)

9.5 Módulo de peticiones a sensores (watcherLogsModule)

En este módulo podemos ver los registros sobre las peticiones a nivel global que se realizan sobre todos los sensores.

- **WatcherLogsComponent:** La vista de peticiones es muy similar a la de sensores. En ella se muestra una gráfica y una tabla donde se muestran las peticiones de lectura a sensores. A diferencia de la vista de sensores, la gráfica que se muestra es en tiempo real, por lo que podemos ver el funcionamiento del sistema en directo.

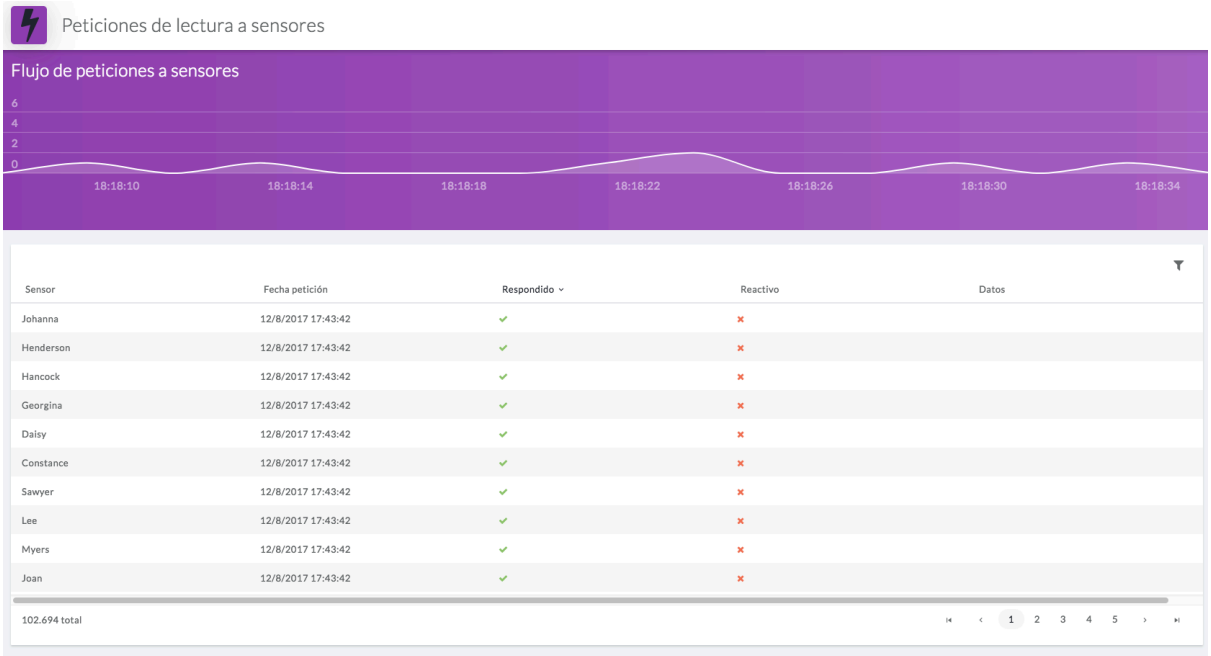


Figura 64 - WatcherLogsComponent (Fuente propia)

9.6 Módulo de registros (logsModule)

El módulo de registros, tiene relación con el requisito funcional 14. El requisito consiste en poder realizar una auditoría del sistema mediante los registros generados por el mismo.

- **LogsComponent:** Es el único componente y el principal del módulo. En él se puede ver y filtrar en una tabla los registros generados por el sistema.

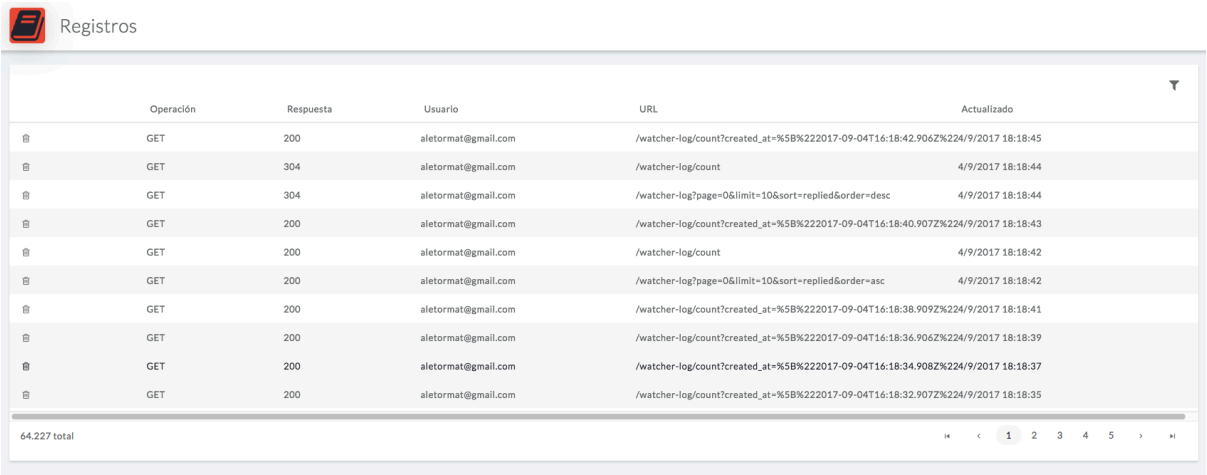


Figura 65 - LogsComponent (Fuente propia)

9.7 Módulo de aplicaciones (applicationsModule) y módulo de usuarios (usersModule)

Los módulos de aplicaciones y usuarios, son los módulos más simples ya que, el componente principal es el listado de cada uno de los tipos de modelos y utilizan el componente genérico para la vista de detalles. Aunque son módulos simples tienen algunos detalles a destacar:

- **ApplicationsComponent:** Cabe destacar la posibilidad de copiar directamente la clave de acceso a la API desde la tabla mediante un botón.

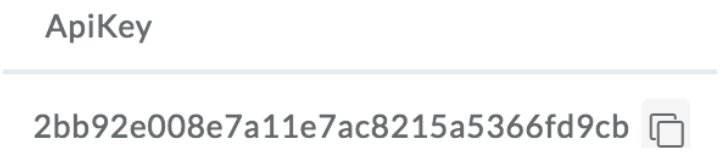


Figura 66 - Botón para copiar la clave de acceso al API desde la tabla (Fuente propia)

- **ApplicationsDetailView:** En este caso, también destacamos el botón para regenerar la clave de acceso a la API.

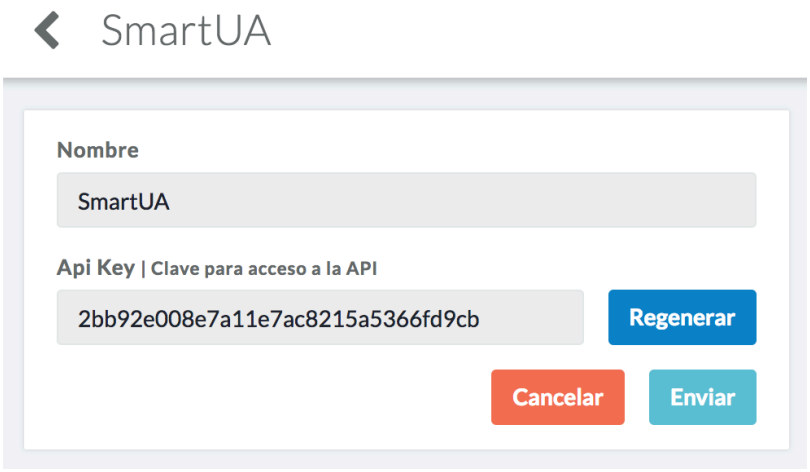


Figura 67 - Botón para regenerar la clave de acceso al API desde el formulario (Fuente propia)

9.8 Módulo de login (loginModule)

Es uno de los módulos más característicos porque no hace uso de la plantilla base que usa el resto de la aplicación. En este módulo se muestra un formulario para autenticarse y entrar en la aplicación.

Figura 68 - LoginModule (Fuente propia)

10 Pruebas y validación

El sistema ha sido probado en dos etapas distintas. La primera etapa ha sido durante la creación de las rutas de la API. En esta etapa se han creado una serie de peticiones usando el cliente de peticiones HTTP: *Postman*[15]. De esta forma se han podido probar las rutas de la API utilizando distintos parámetros para probar el correcto funcionamiento de las mismas.

Además de utilizar *Postman* también se ha podido constatar y probar el correcto funcionamiento de la API a través de la implementación de la interfaz, ya que se utilizan todas las rutas del sistema.

En la segunda etapa de la fase de pruebas se ha pedido al equipo de Smart University que utilizara el sistema en su día a día y probase las diferentes funcionalidades, además de recibir un pequeño feedback de cada miembro del equipo al finalizar las pruebas.

La validación del sistema se ve reflejada en el hecho de haber sido integrado y puesto en producción por el equipo de Smart University, usándolo en la monitorización del sistema controlador de sondas diariamente.

[15] https://www.getpostman.com/

11 Resultados

El objetivo inicial del proyecto era conseguir una interfaz web usable, atractiva y funcional para poder administrar y gestionar el sistema de controlador de sondas. En líneas generales podemos decir que estamos satisfechos con el resultado del proyecto, ya que, tras conversar con el equipo de Smart University la sensación global es que se han cumplido estos objetivos principales con un grado muy alto de satisfacción.

La interfaz es capaz de administrar de forma eficiente los diferentes tipos de modelos, así como analizar los datos de los mismos. Además, también es capaz de filtrar la información gracias a los componentes de búsqueda.

El único requisito que se ha quedado en el tintero ha sido el sistema de alarmas (RF-18), que aun siendo opcional era una funcionalidad muy interesante dentro del sistema.

Por otro lado, una de las metas del proyecto, aunque no era ni un objetivo ni un requisito, era la posibilidad de integrar el proyecto dentro del sistema original de Smart University como aportación personal. Esta meta se ha cumplido y el resultado es que el proyecto está en funcionamiento en un entorno de producción dentro de la Universidad de Alicante.

11.1 Funcionalidades no implementadas

Se ha detectado que el sistema de alarmas era un requisito demasiado grande como para incluirlo dentro del mismo proyecto, por lo que desde una etapa temprana del desarrollo se decidió darle menor prioridad, y solo tratar de implementarlo si los plazos eran lo suficientemente holgados.

12 Conclusiones y trabajo futuro

Puedo decir con firmeza que el proyecto ha sido un éxito, no solo por conseguir los resultados explicados en la sección anterior, sino porque después de todo el esfuerzo puesto tanto en este documento como en el desarrollo e implementación del proyecto, es una satisfacción personal que un trabajo de esta magnitud se pueda visualizar dentro del campus virtual de la Universidad de Alicante y sirva activamente a un equipo como Smart University en la gestión y administración de las sondas repartidas por el campus universitario.

Además, durante el diseño y desarrollo he podido ponerme en dos perspectivas diferentes, por un lado, la perspectiva del desarrollador, que ya conocía, y por otro lado la perspectiva del usuario. Muchas veces, a los desarrolladores se nos olvida que al final los usuarios de las aplicaciones son en última instancia eso mismo: usuarios. Ya tengan estos un perfil más o menos técnico, las interfaces y sistemas que creamos deben de poder ser usadas y entendidas por los mismos. Así pues, estoy también satisfecho con esta parte no tan visible del proyecto, mediante la cual he podido aprender a diseñar interfaces atractivas, pero también usables.

12.1 Trabajo futuro

Sería interesante como trabajo futuro poder desarrollar el sistema de alarmas. Creemos que sería una funcionalidad muy interesante de implementar en un futuro, ya que ahora mismo para hacer una monitorización correcta del sistema se debe navegar por la aplicación, mientras que con un sistema de alarmas se podrían definir una serie de disparadores y reglas para que el sistema nos avisara por correo de posibles fallos y anomalías del mismo.

Además de las funcionalidades no implementadas, como el sistema finalmente va a quedar integrado dentro de la plataforma de Smart University de la Universidad de Alicante, sería muy interesante seguir desarrollando funcionalidades nuevas, así como corrección de los posibles errores o problemas que surjan o se encuentren con el uso de la aplicación.

Por otra parte, una de las posibilidades que podría ofrecer el sistema y que no se ha incluido en el proyecto es poder realizar estudios sobre la eficiencia del controlador. Actualmente se muestran gráficas sobre la relación de las peticiones a dispositivos respondidas respecto a las no respondidas, pero no se ofrece ninguna gráfica sobre el índice de rendimiento del sistema en

general. Esto se podría conseguir utilizando los tiempos medios entre el envío de una petición y la respuesta de la misma, para así poder conocer el grado de funcionamiento del sistema.

Finalmente, y como comentamos en la sección 10: pruebas y validaciones, una de las etapas de pruebas fue dejar que el equipo de Smart University probase el prototipo y al finalizar la prueba recibir un pequeño feedback de cada miembro del equipo. Gracias a este feedback, aunque en todos los casos el resultado global fue positivo, pudimos constatar pequeños aspectos a mejorar o modificar dentro de la aplicación a vista de mejorar la experiencia global de usuario.

13 Abreviaturas y definiciones

- **API** *(Application Programming Interaface):* Interfaz de programación de aplicaciones, que, según Masse (2012: 5), "es la cara de un servicio web, donde se oyen y responden peticiones de los clientes".[16]
- **API Endpoint:** URI de un recurso o acción de una API. Por ejemplo: https://api.controldesondas.com/sonda/sonda01/estado
- **Arduino:** plataforma de código libre para el desarrollo de hardware y software. https://www.arduino.cc/en/Guide/Introduction
- **Backend:** el servidor dentro de una aplicación web.
- **Cron de Unix:** software administrador de procesos en segundo plano que ejecuta un proceso cada un determinado intervalo programado previamente.
- **CRUD** (*Create, Read, Update, Delete):* se refiere a las operaciones atómicas sobre modelos de persistencia: lectura, escritura, actualización y borrado.
- **Dashboard:** visual donde un usuario puede administrar y monitorizar sistemas.
- **Dispositivo o sonda**: dispositivo IoT capaz de recoger información de su entorno.
- **Docker:** software que permite crear máquinas virtuales ligeras y portables a partir de imágenes. https://www.docker.com/
- **Frontend:** el cliente dentro de una aplicación web.
- **GIT:** sistema de control de versiones. https://git-scm.com/
- **IFTTT:** Servicio que permite, mediante APIs REST, crear disparadores y realizar acciones consecuentes. Por ejemplo: subir el volumen de un dispositivo Android al conectarse a una red wifi determinada. https://ifttt.com/
- **IoT** (*Internet Of Things):* El Internet de las cosas.
- **Raspberry PI:** es un ordenador del tamaño de una tarjeta de crédito que se puede conectar a un monitor y un teclado. Puede ser usado para la gran mayoría de cosas que los ordenadores tradicionales.
- **SessionStorage:** es una nueva característica de HTML5, se trata de un pequeño almacenamiento en el navegador accesible a través del objeto *window* en JavaScript. Es

[16] Texto original: "a Web API is the face of a web service, directly listening and responding to client requests".

exactamente igual que el *localStorage* con la diferencia de que los datos almacenados en el *sessionStorage* se eliminan al cerrar el navegador y los del *localStorage* se borran al concluir una fecha de caducidad.

- **TypeScript:** se trata de un superconjunto de Javascript que añade tipado estático y objetos basados en clases. Está mantenido por Microsoft y se ha popularizado gracias a Angular 2.
- **XBee:** solución integrada que brinda un medio inalámbrico para la interconexión y comunicación entre dispositivos. Utiliza el protocolo IEEE 802.15.4. http://xbee.cl/que-es-xbee/

Bibliografía

Beedle, M. et al. 2001. *Manifiesto por el desarrollo ágil de software*. Último acceso: 1 de septiembre de 2017. http://agilemanifesto.org/iso/es/manifesto.html.

Coda Hale. 2010. *How To Safely Store A Password*. Último acceso: 1 de septiembre de 2017. https://codahale.com/how-to-safely-store-a-password/.

Helium. s.f. (a) *Helium Architecture Overview*. Último acceso: 1 de septiembre de 2017. https://www.helium.com/dev/architecture/overview.

—. s.f. (b) *Helium Dashboard*. Último acceso: 1 de septiembre de 2017. https://www.helium.com/dev/dashboard.

IBM. 2017 (actualizado). *Introducción a MQTT*. Último acceso: 1 de septiembre de 2017. https://www.ibm.com/support/knowledgecenter/es/SS9D84_1.0.0/com.ibm.mm.tc.doc/tc00000_.htm.

Institute of Electrical and Electronics Engineers. 2008 (1998). *Especificación de requisitos según el estándard de IEEE 830*. Universidad de Virginia. Último acceso: 1 de septiembre de 2017. https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf

Jones, M. et al. 2015. «JSON Web Token (JWT).» *The RFC series* (RFC Editor) (RFC 7519): 1-29. Último acceso: 1 de septiembre de 2017. https://www.rfc-editor.org/rfc/rfc7519.txt

Light, R. s.f. *mqtt — MQ Telemetry Transport*. Último acceso: 1 de septiembre de 2017. https://mosquitto.org/man/mqtt-7.html.

Masse, M. 2012. *REST API design rulebook*. Sebastopol: O'Reilly Media, Inc. Último acceso: 1 de septiembre de 2017. https://goo.gl/uDvfXm

Provos, N., y D. Mazières. 1999. «A Future-Adaptable Password Scheme.» *1999 USENIX Annual Technical Conference*. Monterey: USENIX. Último acceso: 1 de septiembre de 2017. https://www.usenix.org/legacy/events/usenix99/full_papers/provos/provos.pdf

Rodríguez González, P. 2008. *Estudio de la aplicación de metodologías ágiles para la evolución de productos software*. Madrid: Universidad Politécnica de Madrid. Último acceso: 1 de septiembre de 2017. http://oa.upm.es/1939/1/TESIS_MASTER_PILAR_RODRIGUEZ_GONZALEZ.pdf

Schneier, B. 1993. «Description of a new variable-length key, 64-bit block cipher (Blowfish).» *International Workshop on Fast Software Encryption*. Berlin: Springer. 191-204. Último acceso: 1 de septiembre de 2017. https://link.springer.com/content/pdf/10.1007/3-540-58108-1_24.pdf

TheThings.io. s.f. *App Development*. Último acceso: 1 de septiembre de 2017. https://developers.thethings.io/docs/getting-started-app-development.

Yébenes Gálvez, J. A. 2016 (actualizado). *¿Qué es MQTT?* Último acceso: 1 de septiembre de 2017. https://geekytheory.com/que-es-mqtt.