Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques
Universitat de Barcelona

CMS PARA LA MONITORIZACIÓN DE DATOS DE REDES INALÁMBRICAS DE SENSORES

Iñaki Úbeda Martínez

Director: Christos Verikoukis
Realitzat a: Centre Tecnològic de Telecomunicacions de Catalunya (CTTC)
Tutor CTTC: David Pubill

Barcelona, 30 de junio de 2016

Agradecimientos

Al Centre Tecnològic de Telecomunicacions de Catalunya por su implicación y por todas las facilidades puestas para realizar este proyecto.

A Christos Verikoukis por la oportunidad de poder realizar este proyecto.

A David Pubill y Jordi Serra por su trabajo, paciencia y consejos.

Abstract
This document explains the development of a web application to manage and monitor the content information collected by the testbed IoTWORLD, a wireless sensor network platform deployed in the Centre Tecnològic de Telecomunicacions de Catalunya (CTTC) in Castelldefels. The project consists on the design and development of a web application that meets the needs of the CTTC to manage current and future experimental use cases.

This report includes the analysis, design and development of the project and any extensions or improvements. It also explains the development of the project using the Node.js framework *Express*, *MySQL* databases, library graphical representation *C3 js* and other technologies used in the back-end (i.e. an administration section of a web application where one can modify its content) and in the front-end (i.e. the accessible section of the web application for any kind of user).

Resumen
Este documento explica el desarrollo de una aplicación web para gestionar el contenido y monitorizar la información recogida por el banco de pruebas IoTWORLD, una red inalámbrica de sensores desplegada en las oficinas del Centre Tecnològic de Telecomunicacions de Catalunya (CTTC) en Castelldefels. El proyecto consiste en el diseño y desarrollo de una aplicación web que se adapte a las necesidades del CTTC para permitir gestionar los casos de uso experimentales actuales y futuros.

Esta memoria recoge el análisis, diseño y desarrollo del proyecto, así como las posibles ampliaciones o mejoras. También se explica cómo se ha desarrollado el proyecto utilizando el *framework* de Node.js *Express*, el sistema de bases de datos *MySQL*, la librería de representación gráfica *C3 js* y el resto de tecnologías empleadas tanto en el *back-end* (sección de administración de una aplicación web desde donde se puede modificar el contenido de la misma), como en el *front-end* (parte de la aplicación web accesible para cualquier tipo de usuario).

Resum
Aquest document explica el desenvolupament d'una aplicació web per gestionar el contingut i monitoritzar la informació recollida pel banc de proves IoTWORLD, una xarxa de sensors sense fils desplegada a les oficines del Centre Tecnològic de Telecomunicacions de Catalunya (CTTC) a Castelldefels. El projecte consisteix en el disseny i desenvolupament d'una aplicació web que s'adapti a les necessitats del CTTC per permetre gestionar els casos d'ús experimentals actuals i futurs.

Aquesta memòria recull l'anàlisi, disseny i desenvolupament del projecte, així com les possibles ampliacions o millores. També s'explica com s'ha desenvolupat el projecte utilitzant el *framework* de Node.js *Express*, el sistema de bases de dades *MySQL*, la llibreria de representació gràfica *C3 js* i la resta de tecnologies emprades tant al *back-end* (secció d'administració d'una aplicació web des d'on es pot modificar el contingut) com el *front-end* (part de l'aplicació web accessible per a qualsevol tipus d'usuari).

Índice

1. Introducción
    - 1.1 Motivación
    - 1.2 Situación actual / Contexto
    - 1.3 Objetivos
    - 1.4 Planificación
    - 1.5 Distribución de la memoria

2. Análisis
    - 2.1 Análisis de requerimientos y funcionalidades
    - 2.2 Requisitos funcionales
    - 2.3 Casos de uso
        - 2.3.1 Usuario anónimo
        - 2.3.2 Administrador
    - 2.4 Requisitos no funcionales
        - 2.4.1 Accesibilidad
        - 2.4.2 Rendimiento
        - 2.4.3 Mantenibilidad
        - 2.4.4 Real time

3. Diseño
    - 3.1 Esquema de funcionamiento
    - 3.2 Modelos de datos

4. Tecnologías utilizadas
    - 4.1 GitHub
    - 4.2 Programación: Node.js vs PHP
    - 4.3 Express js
    - 4.4 Base de datos: MySQL vs MongoDB
    - 4.5 Representación: C3.js
    - 4.6 Motor de Plantillas: Jade
    - 4.7 CSS: Boostrap
    - 4.8 Socket.io

5. Implementación
    - 5.1 Arquitectura general del proyecto
    - 5.2 Servidor
        - 5.2.1 Modelos
        - 5.2.2 Controlador
        - 5.2.3 Vistas
        - 5.2.4 Routes

5.3 Cliente

6. Pruebas y resultados
6.1 Apariencia de la aplicación

7. Conclusiones
7.1 Conclusiones generales
7.2 Posibles mejoras

8. Bibliografía

9. Anexos
9.1 Manual del usuario
9.2 Guía de instalación

1. Introducción

1.1 Motivación

Este proyecto nace de la necesidad de evolucionar y rediseñar la aplicación web que presenta el *testbed* (banco de pruebas) IoTWORLD (http://iotworld.cttc.es) del CTTC. En esta web además de representar la información obtenida durante el experimento en tiempo real se deberán poder modificar ciertos parámetros que interaccionan con los dispositivos del experimento. Además, la aplicación ha de ser administrable, es decir, debe existir un panel de administración desde donde se pueda gestionar el contenido de la aplicación web.

La realización de este proyecto me permite la posibilidad de aprender y profundizar en tecnologías que están muy presentes en el mundo laboral. También el que la aplicación web esté centrada en un campo con tanto futuro como el de *Internet of Things* (IoT, Internet de las cosas) me genera una especial atención y me brinda la oportunidad de ver algunas de las múltiples aplicaciones que ofrece.

1.2 Situación actual / Contexto

El *testbed* IoTWORLD es una plataforma experimental única para IoT. Se centra en los sistemas de comunicación inalámbricos y de análisis de datos. En concreto, ha sido desarrollado por la División de Tecnologías de Comunicaciones del CTTC.

IoTWORLD se organiza en una arquitectura de cuatro capas:

1. La capa inferior está formada por sensores de bajo consumo y actuadores que permiten interactuar con el mundo real. Algunos de los sensores que integra el *testbed* son: temperatura, humedad, nivel de CO_2, consumo eléctrico, etc.

2. La segunda capa consiste en un conjunto de *gateways* que permiten reunir los datos recogidos por los dispositivos de la primera capa. Los *gateways* están implementados sobre dispositivos Raspberry Pi 2, que son mini-ordenadores de bajo coste que favorecen la flexibilidad y escalabilidad de IoTWORLD.

3. La tercera capa se encuentra en la nube y está dividida en dos: un sistema de almacenamiento local en un PC del laboratorio y otro utilizando los servicios de *Google Cloud Platform*.
4. La capa superior es la capa de usuario. Los usuarios pueden interactuar con IoTWORLD mediante una aplicación web accesible desde cualquier dispositivo, objetivo de este proyecto.

Actualmente el *testbed* IoTWORLD cuenta con una aplicación web que permite visualizar la información que recogen los sensores distribuidos por diferentes puntos de los edificios del CTTC en Castelldefels. Esta aplicación web sufre de problemas de compatibilidad con algunos navegadores, está orientada a su utilización exclusivamente desde un dispositivo táctil y los tiempos de carga son algo elevados para una aplicación que muestra información en tiempo real. Estos factores hacen que la experiencia del usuario se resienta.

Desde un punto de vista más técnico en la aplicación web actual no se utiliza ningún tipo de patrón de diseño en la aplicación más allá de la programación estructurada, lo cual afecta al mantenimiento y escalabilidad del proyecto. También se puede observar que existe mucha información que podría ser parametrizable y, sin embargo, está escrita directamente en el código, esto genera código repetido lo cual afecta a la ampliación del proyecto. Existen también una serie de cálculos que se realizan en la misma aplicación web los cuales son innecesarios. Estos cálculos se podrían emplazar en otro proceso o incluirlos sobre la propia base de datos.

1.3 Objetivos

En general, el objetivo de este trabajo final de grado (TFG) es crear una aplicación web que permita mostrar en tiempo real los datos generados por el *testbed* IoTWORLD.

La aplicación web debe ser de acceso público pero no todos los tipos de usuarios deben poder acceder a todas las funcionalidades implementadas. Existen dos modos de visualización:

- la de usuario: permite visitar la web y consultar la monitorización de datos en tiempo real.
- la de administrador: permite gestionar el contenido de la página.

Otro de los objetivos de este TFG es la compatibilidad entre dispositivos, se desea que la aplicación sea compatible con la mayor cantidad posible de navegadores del mercado tanto en ordenadores como en dispositivos portátiles. Esto implica que la aplicación debe tener un diseño *responsive* (adaptable).

El rendimiento de la aplicación tanto en los *dashboards* (páginas de la aplicación web con gráficos) como en las páginas de contenido (texto y/o video/fotografías) ha de ser el mejor posible, es decir, los tiempos de carga deben ser lo más cortos posibles.

1.4 Planificación

La distribución del tiempo durante el desarrollo de este proyecto ha sido la siguiente:

- 4 semanas de estudio e investigación de soluciones
- 9 semanas de programación
- 2 semanas de pruebas en entorno real
- 1 semana de documentación, aunque durante toda la duración del proyecto se ha redactado la presente memoria.

1.5 Distribución de la memoria

En este documento se incluyen los siguientes capítulos:

Capítulo 1 - Introducción: Se presenta una breve introducción sobre el *testbed* IoTWORLD y su diseño. Además, se explica la motivación por la cual se realiza este proyecto, los objetivos, la planificación y la estructura del documento.

Capítulo 2 - Análisis: Se explica las diferentes funcionalidades según los tipos de usuarios definidos y se analizan los requisitos del sistema.

Capítulo 3 - Diseño: Se comenta como se ha diseñado el modelo de datos, como se ha integrado sobre el ya existente y cómo se implementa la base de datos (BBDD).

Capítulo 4 - Tecnologías utilizadas: Descripción de las tecnologías utilizadas en el proyecto incluyendo algunas comparativas con otras soluciones existentes en el mercado.

Capítulo 5 - Implementación: Se comenta la implementación general del proyecto, los patrones utilizados y cómo se han utilizado las tecnologías utilizadas para obtener el resultado final.

Capítulo 6 - Pruebas y resultados: Se explican las diferentes pruebas para asegurar el correcto funcionamiento de la aplicación.

Capítulo 7 - Conclusiones: Conclusiones de la realización del proyecto y posible mejores para versiones posteriores de la aplicación.

Capítulo 8 - Bibliografía: Textos y enlaces de donde se ha extraído información para la creación del proyecto

Capítulo 9 - Anexos: Manual de usuario y guía de instalación

2. Análisis

2.1 Análisis de requerimientos y funcionalidades

Para el desarrollo de este proyecto se analizaron una serie de requisitos y funcionalidades que debía cumplir la aplicación.

Se requiere que la aplicación sea compatible con la mayor cantidad de navegadores web actuales ya sea en su versión de escritorio o en dispositivos portátiles (tablet, smartphone). Por lo tanto, la aplicación ha de ser *responsive* (adaptable según el tipo de pantalla).

La representación de la información ha de ser más clara e intuitiva para el usuario, además también se debe mejorar la experiencia del usuario optimizando los tiempos de carga en las páginas con gráficos asociados.

La actualización de datos en los gráficos cuando estos estén representando información en tiempo real debe ser los más real posible, es decir, el tiempo entre que se toma una muestra y esta es representada sobre un gráfico debe ser el menor posible. Sobre esta funcionalidad existirá alguna serie de restricciones en función del diseño final que se opte por realizar.

Por último el contenido ha de ser administrable, desde la sección de administración se debe poder editar el contenido tanto de páginas de texto, contenido informativo de los *casos de uso experimentales (implementaciones concretas de IoTWORLD)*, como de *dashboard*, las páginas con gráficos.

2.2 Requisitos funcionales

Los requisitos funcionales indican qué comportamiento debe tener la aplicación web, es decir, nos ayudan a especificar las funcionalidades del sistema.

Se definen dos grupos de usuarios para la aplicación, de esta manera podemos describir al detalle qué funcionalidades tienen cada uno de ellos.

Los dos grupos definidos son:

- Usuario anónimo
- Administrador

2.3 Casos de uso

Un caso de uso es una descripción de las diferentes acciones que puede realizar los tipos de usuarios definidos previamente. A continuación se describen los casos de uso para los grupos de usuarios definidos.

2.3.1 Usuario anónimo

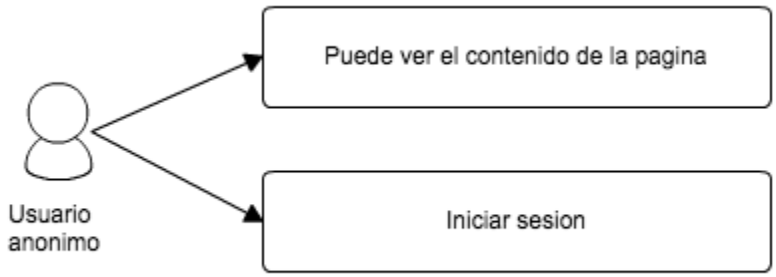


Figura 1. Casos de uso del usuario.

Consideramos como ‘usuario’ a cualquier usuario que accede a la aplicación web sin necesidad de haber iniciado sesión anteriormente. Es el tipo de usuario más básico. Derechos:

- **Visitar el contenido:** Puede visitar la página y consultar el contenido de la misma, ya sea la descripción de los *casos de uso experimentales* como los *dashboards* disponibles en ese momento.

- **Iniciar sesión:** En caso de conocer los datos de acceso un usuario anónimo puede acceder a la administración de la aplicación web.

2.3.2 Administrador

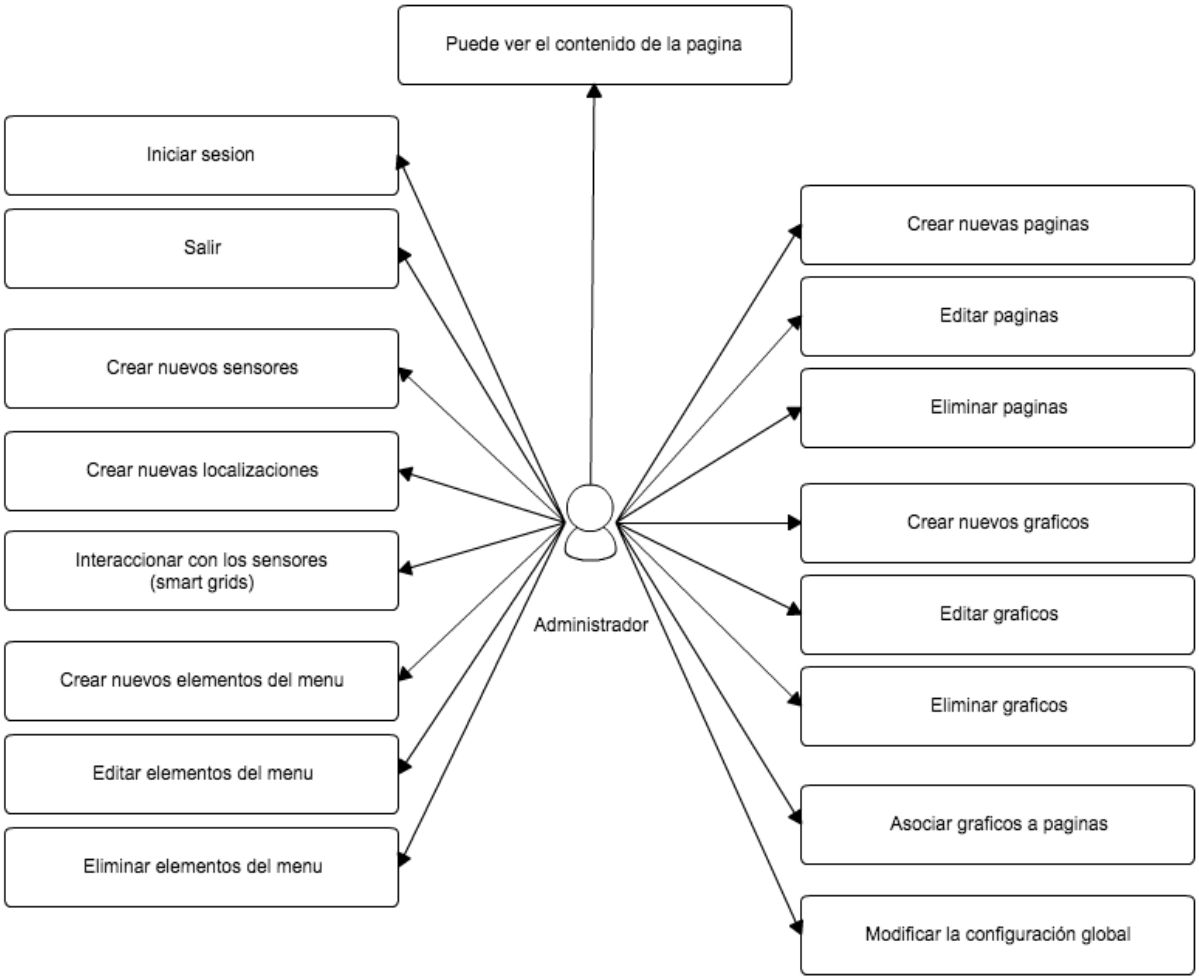


Figura 2. Casos de uso del administrador

El usuario de tipo 'administrador' tiene acceso a todas las funcionalidades de la aplicación web por lo que incluye los casos de uso del usuario anónimo. Derechos:

- **Iniciar sesión:** utilizar los datos de acceso para acceder a la sección de administración de la aplicación web.
- **Salir:** Una vez iniciada la sesión se puede salir y cancelar dicha sesión desde la sección de administración
- **Interaccionar con los sensores:** Desde el *dashboard* de *Smart Grids* (un caso de uso experimental de IoTWORLD) podrá cambiar los valores de las temperaturas de este *use case* seleccionando entre los dos tipos de algoritmos existentes.
- **Añadir sensores**: Crear nuevos sensores en las futuras instalaciones que se realicen.

- **Activar/desactivar sensores**: Se podrán activar o desactivar los sensores para que estos muestren o no información en los gráficos, esto no querrá decir que se deje de recoger la información del sensor en caso de desactivarlo.
- **Añadir localizaciones:** Crear nuevas localizaciones para la instalación de nuevos sensores.
- **Crear/editar/eliminar elementos del menú:** Se podrán añadir nuevos elementos al menú, editar los ya existentes en función de si el elemento es editable. La dirección de un elemento del menú debe ir relacionado con la dirección de una página.
- **Crear/editar/eliminar páginas:** Crear nuevas páginas de contenido o con gráficos o editar las ya existentes. En las páginas de contenido se podrá editar texto y añadir fotos o enlazar videos. En las páginas de tipo gráficos se podrán asociar los gráficos que se hayan creado previamente.
- **Crear/editar/eliminar gráficos:** Crear nuevos gráficos definiendo el tipo de datos a representar o el formato en el que se harán, editar los gráficos ya existentes y eliminar gráficos.
- **Modificar la configuración global:** modificar la configuración de la aplicación para variar la cantidad de información mostrada en los gráficos inicialmente, la cantidad de información que se enviará cuando en un gráfico se esté visualizando información en tiempo real y la frecuencia de envío de estos datos.

En el Anexo 1 (manual de usuario) se describen todas las funcionalidades aquí descritas de forma que el usuario pueda realizarlas de una forma rápida y eficiente.

2.4 Requisitos no funcionales

Los requisitos no funcionales difieren de los requisitos funcionales en que no definen ningún tipo de comportamiento o funcionalidad, son aquellas características o propiedades de funcionamiento que el sistema a desarrollar debe poseer para cumplir con las exigencias de calidad deseadas. A continuación se detallan los requisitos no funcionales observados durante el análisis.

2.4.1 Accesibilidad

La aplicación ha de ser accesible a través de diversos dispositivos, tanto para ordenadores, móviles como tabletas. Para ello se utilizará el patrón RWD (*Responsive Web Design*) que permite que las páginas se adapten al tamaño, la resolución y orientación de la pantalla, y por tanto al dispositivo del usuario. Más adelante veremos qué tecnologías se han utilizado para cubrir este requerimiento.

2.4.2 Rendimiento

La aplicación debe funcionar de una manera rápida y sencilla. El tiempo de carga en los *dashboard* ha de ser el menor posible aunque se verá condicionado por la cantidad de gráficos que se tengan en ese momento activos. Como se ha comentado en el caso de uso del administrador (punto 2.3.2) el administrador puede asociar gráficos a una página, además el administrador podrá modificar la cantidad de información que se envía en cada momento y la frecuencia con la que se hace. Por lo tanto, también es tarea del administrador mantener unos valores equilibrados para no perjudicar la experiencia del usuario.

2.4.3 Mantenibilidad

La aplicación ha de ser mantenible, el diseño de la aplicación debe permitir el desarrollo de nuevas funcionalidades o la ampliación de las ya existen de forma ágil y eficiente. Esto se puede llevar a cabo utilizando patrones de diseño que permitan la escalabilidad del proyecto y generando una documentación clara del proyecto.

2.4.4 *Real-time*

Los gráficos deben poder mostrar diferentes tipos de información (información en tiempo real y medias por minutos, horas o días), entre ellas está la opción de *real-time* la cual mostrará los últimos valores registrados en la base de datos cada cierto tiempo. Aunque es evidente que existirá un cierto *lag* (desfase) entre el guardado de esta información y su representación, este tiempo debe ser el menor posible.

3. Diseño

La aplicación web ya contaba con una estructura de datos de la versión anterior, en esta versión previa únicamente se tenía en cuenta los eventos (registro de un sensor en un instante de tiempo determinado), las medias calculadas (minutos, horas y días) de los eventos y los datos introducidos por el usuario en el caso de uso experimental de *Smart Grids*.

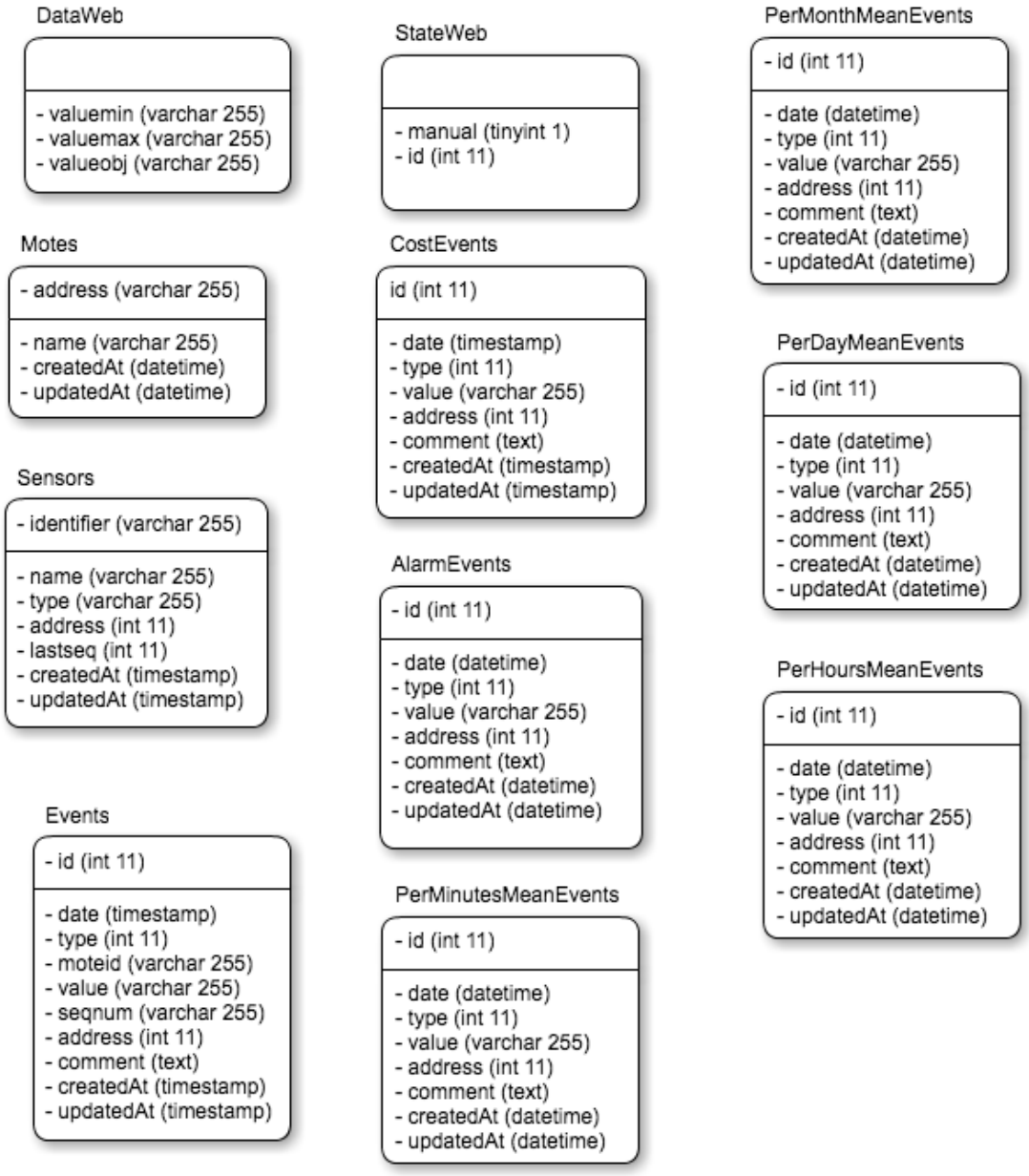


Figura 3. Modelo de datos inicial.

Como se puede ver en la Figura 3 en el modelo de datos inicial de la aplicación no existe ningún tipo de relación entre las tablas. Esto se debe a que en la mayoría de casos no era necesario, no obstante las tablas de *Sensor*, *Motes* y *Events* si están relacionadas entre ellas.

3.1 Esquema de funcionamiento

Una vez analizado el modelo de datos inicial y el funcionamiento de la plataforma anterior se procede a crear una serie de nuevas entidades para facilitar la gestión del contenido de la página y también poder generar un código más limpio y mantenible. Las nuevas entidades creadas son:

- ***SensorDetails*:** Tendrá los datos relacionados con un sensor como el tipo (campo relacionado con la tabla *Sensors*), un nombre, la unidad de medida del sensor y un identificador del tipo de evento (co2, temp, hum…).
- ***SensorLocations*:** Tendrá el nombre de la localización del sensor y un campo llamado *address* el cual está relacionado con las siguientes tablas *Sensors, Events, CostEvents, PerMinutesMeanEvents, PerHoursMeanEvents, PerDayMeanEvents, PerMonthMeanEvents*. De esta forma podremos, por ejemplo, identificar rápidamente a qué localización corresponde cada registro de la tabla *Events*.
- ***Pages*:** Contiene la información relacionada con las páginas de la aplicación web. En ella se almacenan el título de la página, la dirección para acceder a ella, el contenido (únicamente en caso de ser de tipo contenido), el tipo de página (de contenido o de gráficos) y la plantilla que se utilizará para mostrar la información.
- ***PageTypes*:** Hace referencia a los tipos de páginas disponibles inicialmente, estos tipos son: de contenido (texto y/o imágenes/videos) y de gráficos.
- ***Charts*:** Contiene la información relacionada con un gráfico como el nombre, el tipo de gráfico, el formato del gráfico y si tiene o no botones para ver información en tiempo real y ver también las medias.
- ***PageCharts*:** Relaciona una página del tipo gráficos con un gráfico.
- ***Menus*:** Elementos del menú superior, consta de información como el nombre, la dirección a la que dirige el elemento del menú, si es padre (es el elemento superior de un desplegable), el id del padre, el orden y si es editable.
- ***Users*:** En esta tabla se almacena la información de los usuarios que pueden acceder a la sección de administración para poder gestionar el contenido de la aplicación.

Una vez creadas estas entidades las adaptamos al modelo de datos que teníamos previamente, en el siguiente punto analizamos el resultado final.

3.2 Modelos de datos

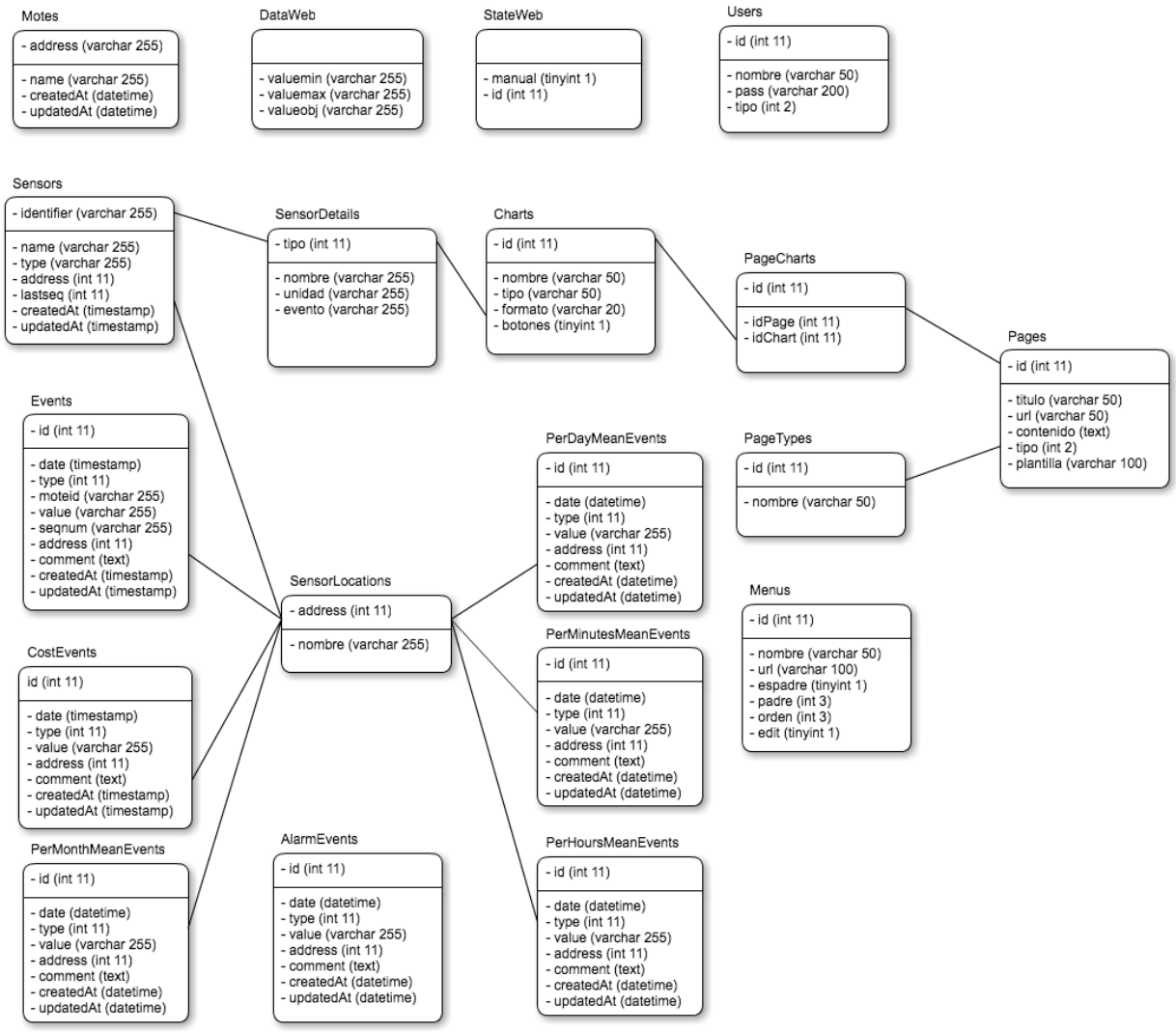


Figura 4. Modelo de la base de datos actual.

En la Figura 4 podemos visualizar las tablas que componen la base de datos final y las relaciones que existen entre ellas. Mediante este nuevo esquema podemos realizar una mejor gestión de la información de los sensores actuales y futuros y gestionar el contenido de la aplicación web para crear nuevas páginas o gráficos en función de las necesidades del usuario.

Estas relaciones están establecidas mediante *Sequelize* que es el *ORM* (*Object-Relational Mapping*) utilizado para gestionar las tablas y sus relaciones desde la aplicación, en el capítulo 5 se entra en más detalle sobre este aspecto.

4. Tecnologías utilizadas

Para el desarrollo de esta aplicación web se han considerado diferentes lenguajes de programación, sistemas de bases de datos y librerías de representación gráfica. Así como diferentes entornos de desarrollo.

Los entornos de desarrollo utilizados durante el proyecto han sido los siguientes:

- **Servidor local:** entorno de desarrollo.
    - OS X El Capitan v10.11.3
    - Node.js v4.4.0
    - MySQL v5.7.11

- **Cloud 9:** entorno de desarrollo alternativo.
    - Ubuntu 14.04.3 LTS
    - Node js v4.4.3
    - MySQL v5.5.49

- **Servidor en el CTTC:** entorno de producción.
    - Ubuntu 16.04 LTS
    - Node.js v0.10.17
    - MySQL v5.7.12

A continuación se describen algunas de las tecnologías utilizadas y sus alternativas incluyendo en algunos casos una comparativa y la decisión final.

4.1 *GitHub*

Antes de describir *GitHub* debemos hablar de *Git*.

Git es un software de control de versiones desarrollado por Linus Torvald. Un software de control de versiones es aquel que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que se puede recuperar una versión específica de estos archivos más adelante.

GitHub nos proporciona una infraestructura en internet sobre la cual realizar el control de versiones de nuestro proyecto mediante una interfaz web y también accesible por línea de comandos.

Las principales razones para utilizar *Git* y la infraestructura de *GitHub* son:

- Registro de cambios en todo el código.
- Posibilidad de deshacer cambios rápidamente.
- Trabajar desde diferentes localizaciones teniendo siempre la última versión del código.
- Guardar una copia de seguridad del código y de las versiones anteriores.

Repositorio utilizado para este proyecto: https://github.com/iubeda/TFG

4.2 Programación: Node.js vs PHP

Para la realización del proyecto se barajaron dos lenguajes de programación para el servidor, la aplicación web anterior estaba realizada con node.js. Estas dos opciones fueron ***Node.js***, que permitía una cierta continuidad sobre lo ya desarrollado, y ***PHP*** *(Pre Hypertext-Processor)*, que implicaba diseñar y reescribir todo desde cero. A continuación veremos una breve comparativa de estas dos soluciones:

- **Node.js:**

Lanzado en 2009, utiliza la máquina virtual V8 de Google, la misma que se utiliza en el navegador Google Chrome, para ejecutar código JavaScript del lado del servidor. Se basa en la programación orientada a eventos, cada vez que se realice una conexión, se reciban datos o se cierre una conexión se produce un evento.

- **PHP:**

Es un lenguaje de programación lanzado en 1995 que se ejecuta en un servidor originalmente diseñado para el desarrollo web de contenido dinámico. El código es interpretado por un servidor, Apache o Nginx por ejemplo, con un módulo que procesa el código y genera la página web resultante. Aunque inicialmente no estaba pensado como un lenguaje de POO (Programación Orientada a Objetos) en las sucesivas actualizaciones que se han hecho se ha potenciado el uso de este tipo de programación.

Comparativa Node.js vs PHP

En este apartado se comparan las capacidades de ambos lenguajes para procesar un número elevado de peticiones y su rendimiento a la hora de ejecutar la ordenación.

- Multi-Tasking

  Estos dos lenguajes tienen un enfoque diferente en cuanto a la concurrencia, mientras que Node.js utiliza un bucle de eventos no bloqueantes en un único hilo PHP utiliza múltiples hilos de eventos bloqueantes.

- Simple HTTP Request
  - *Benchmark* realizado utilizando PHP 5.6.6 con OPcache activado.
  - Test hecho utilizando la herramienta *Apache ab benchmarking*.
  - Test basado en una aplicación simple ('Hello World!").

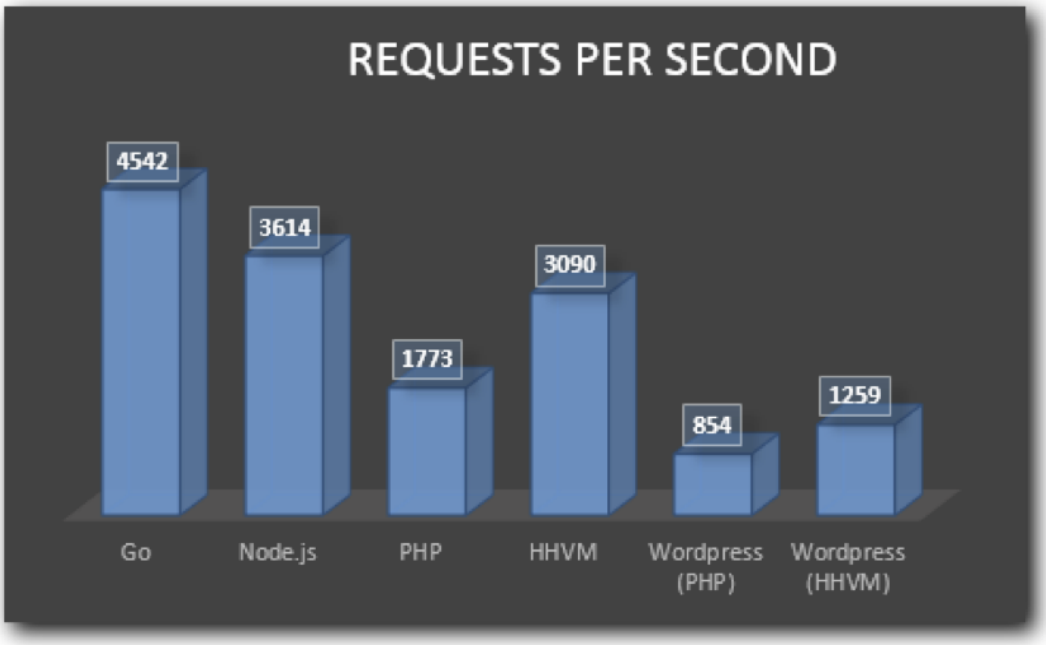


Figura 5. Peticiones por segundo soportadas por cada lenguaje.

Como podemos ver en la Figura 5 Node.js es capaz de procesar el doble de peticiones que PHP, frente a HHVM la diferencia es de tan solo de un 17%.
El rendimiento de HHVM (*HipHop Virtual Machine*) es un 74% mejor que PHP.

- HTTP + CPU task
  - *Benchmark* realizado utilizando PHP 5.6.6 con OPcache activado.
  - En este test utilizamos el algoritmo de ordenación *Bubble sort*.

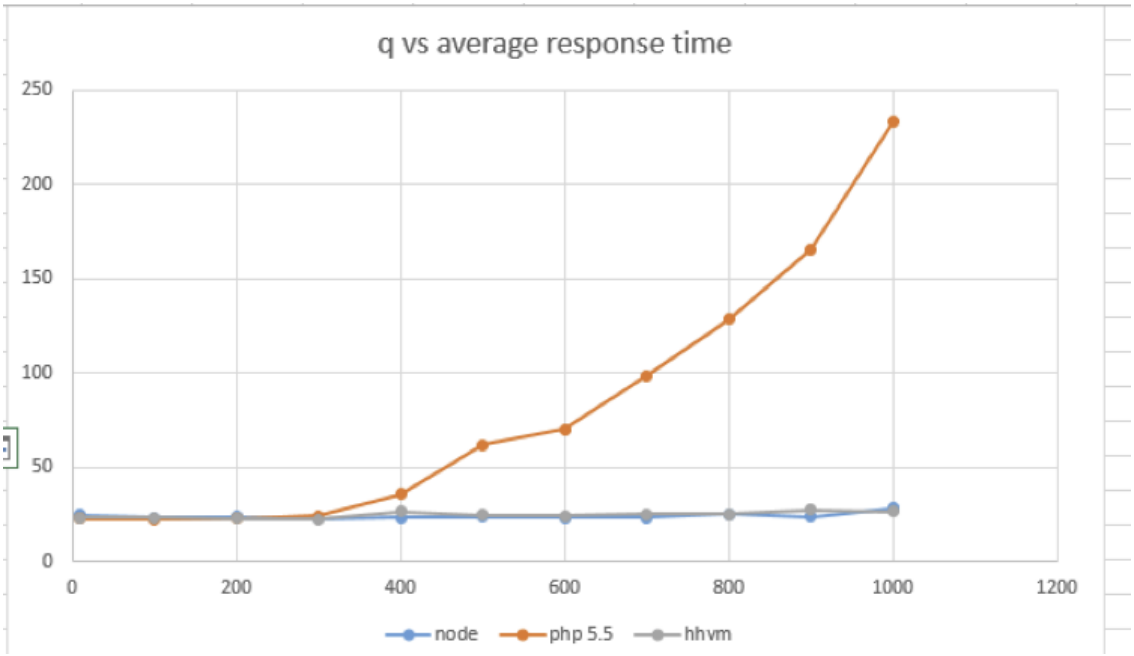


Figura 6. Ordenación Bubble sort sobre Node y PHP 5.5

Tal y como se muestra en la Figura 6 los tiempos de respuesta de PHP se degradan rápidamente después de agotar el proceso de agrupación disponible PHP (usando un estándar máximo de 250 procesos en el servidor web).

Para 1000 elementos Node.js y HHVM son bastante similares, para 10.000 elementos Node.js es el doble de rápido que HHVM (no visible en el gráfico de la Figura 6).

- *ComboSort Strict CPU Test*
  - *Test de uso estrictamente de CPU.*
  - *ComboSort [1]*

	CPU time	**System time**	**RAM**
PHP 5.6.4	102.69s	104.20s	2497508 KB
HHVM 3.5.0	12.56s	14.83s	362488 KB
Node.js v.0.10.35	2.64s	2.64s	92240 KB

Figura 8. Tabla comparativa de recursos consumidos

[1] http://kokizzu.blogspot.com.es/2015/02/numeric-combsort-benchmark-updated.html

Como podemos observar en la Figura 8 HHVM es 7 veces más rápido que PHP en tiempo de sistema, consume menos tiempo del sistema en realizar la operación, y aun así Node.js es 5 veces más rápido que HHVM.
Se puede ver más información sobre los test realizados en:
http://www.hostingadvice.com/blog/comparing-node-js-vs-php-performance/

Conclusión

Una vez vistos los diferentes tipos de test y evaluando las características de nuestra aplicación el uso de Node.js es más que recomendado. Es cierto que es poco probable que se vaya a manejar un alto número de conexiones simultáneas pero sí hemos podido ver que tanto en operaciones más costosas, como una ordenación, como en un simple "Hello Word!" los tiempos de respuesta de Node.js son notablemente mejores. Teniendo en cuenta que necesitamos un sistema lo más ágil posible para la representación de datos en tiempo real la elección de Node.js es sin duda la mejor opción.

4.3 Express js

Express es una *framework* de desarrollo de aplicaciones web minimalista y flexible para Node.js que ofrece un *router* de URLs, facilidades para integrar motores de plantilla y middleware. Con estas características, entre otras, este *framework* facilita el desarrollo de aplicaciones y/o *API's* (*Application Programming Interface*) en *Node.js*.

Se ha optado por este *framework* entre las múltiples opciones que existen para *Node.js* por su largo recorrido, por ser un proyecto estable y por haber evolucionado mucho gracias a la comunidad de usuarios que tiene.

Una de las funcionalidades que proporciona es un generador de estructura de aplicaciones, esta funcionalidad da la estructura base de un proyecto sobre la cual después se puede moldear según las necesidades del mismo.

Otra funcionalidad realmente interesante es el poder definir la carga automática de los modelos, como se explicará en la sección de implementación (capítulo 5) existe la posibilidad de crear una carpeta para gestionar estos archivos y que estos se carguen automáticamente al incluir la carpeta en los controladores que sea necesario.

4.4 Base de datos: MySQL vs MongoDB

Para la realización del proyecto se analizaron estos dos sistemas de bases de datos, inicialmente el proyecto ya contaba con una base de datos en MySQL así que esta era un de las opciones para mantener la continuidad de lo desarrollado previamente. La otra opción es MongoDB un sistema de bases de datos completamente diferente y que supondría un cambio radical tanto a nivel de representación de datos como en el guardado de información por parte de los *gateways*. A continuación se explican ventajas e inconvenientes de cada una de estas opciones, la decisión final y el porqué de esta.

- **MySQL**

MySQL (*My Structured Query Language*) es un sistema de gestión de base de datos relacional desarrollado inicialmente por MySQL AB la cual fue comprada posteriormente por Sun Microsystems en 2008 y está a su vez fue comprada por *Oracle Corporation* en 2010, es *open-source* bajo la licencia GNU (GPL) y es uno de los sistemas de bases de datos más utilizados en todo el mundo.

En un sistema de base de datos relacional el sistema almacena la información en tablas separadas en lugar de utilizar un único y gran archivo. Esto proporciona una mayor flexibilidad y velocidad. Estas tablas están relacionadas entre sí y permiten combinar los datos entre ellas.

Desde la versión 5.1 de MySQL se incluye un nuevo concepto, los eventos. Esta característica no es más que la ejecución de ciertas sentencias planificadas que pueden llamar a procedimientos almacenados o funciones de MySQL. Lo interesante de estos eventos es que podremos definir cada cuánto tiempo deben ser ejecutados. Esta funcionalidad nos puede resultar muy útil para generar vistas o cálculos cada cierto intervalo de tiempo.

- **MongoDB**

Es una sistema de bases de datos no relacional lanzado en 2009 de código abierto que opera bajo la licencia GNU AGPL v3.0.

En un sistema de base de datos no relacional la información no se almacena en tablas sino en documentos en *BSON* (*Binary JavaScript Object Notation*) que es una representación binaria de *JSON* (*JavaScript Object Notation*) con un esquema dinámico, la estructura es mutable en cualquier momento, haciendo que la integración de datos en algunos contextos sea realmente rápida y fácil.

Una de las ventajas de MongoDB reside en que las consultas se hacen utilizando una sintaxis similar a la utilización de objetos en *Javascript*. Por lo tanto, la curva de aprendizaje de este lenguaje es menor con respecto a MySQL ya que no se debe aprender una sintaxis nueva y/o compleja.

Comparativa MySQL vs MongoDB

En esta comparativa realizaremos dos test de lectura y escritura concurrente con diferentes configuraciones sobre los dos sistemas para observar su rendimiento.

Test de lectura y escritura concurrente

- **Test 1:** De 0 a 50 usuarios incrementando de 5 en 5 y manteniendo la carga durante 100 segundos.

	Muestras	Mediana	Min	Max	Desv. Est	Consultas/s	Kb/s
MySQL	373.484	**71ms**	52ms	3.152ms	27,98ms	571,1 c/s	139,91 kb/s
MongoDB	462.740	**55ms**	48ms	3.111ms	26,24ms	707,7 c/s	173,45 kb/s

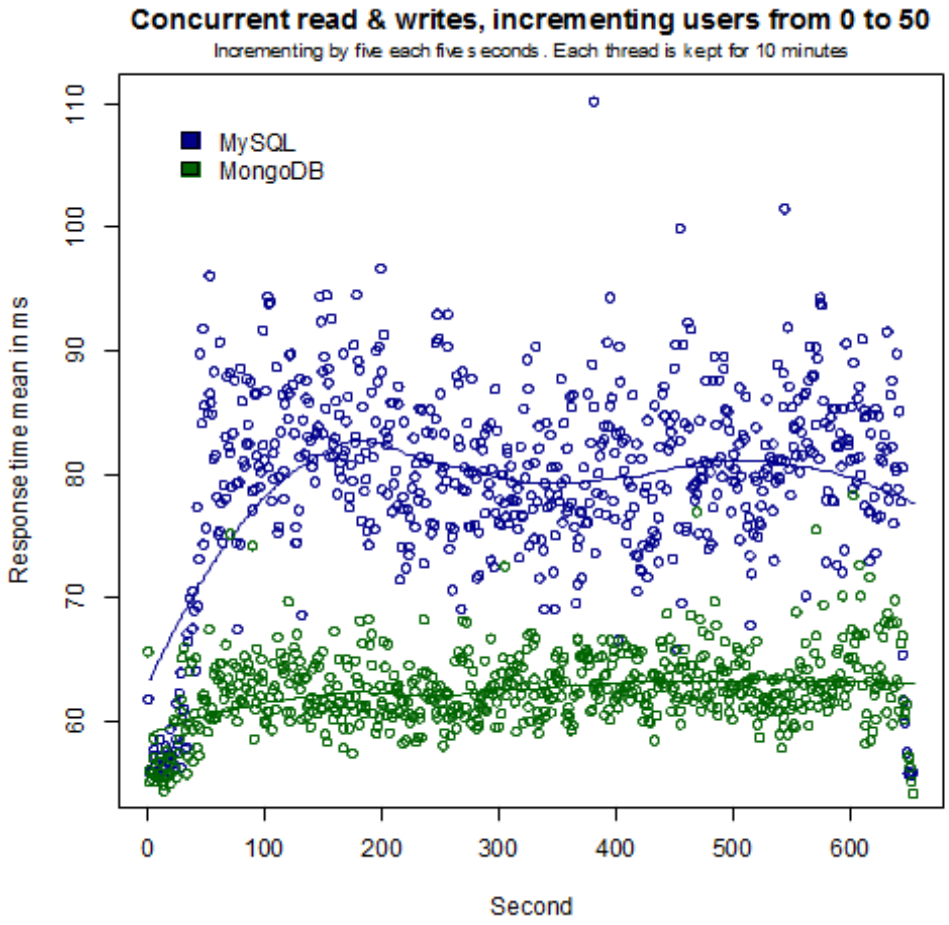


Figura 9 Lectura y escritura concurrente incrementando los usuarios.

Como podemos ver en la Figura 9 la diferencia en la mediana entre MySQL y MongoDB no es demasiado elevada, no obstante esa pequeña diferencia de velocidad hace que finalmente MongoDB sea capaz de procesar un mayor número de lecturas/escrituras totales.

- **Test 2:** 50 usuarios desde el principio y manteniendo la carga durante 50 segundos.

	Muestras	Mediana	Min	Max	Desv. Est	Consultas/s	Kb/s
MySQL	31.272	**64ms**	52ms	5.964ms	176,64ms	530,0 c/s	129,8 kb/s
MongoDB	39.096	**54ms**	50ms	4.753ms	142,72ms	665,0 c/s	162,93 kb/s

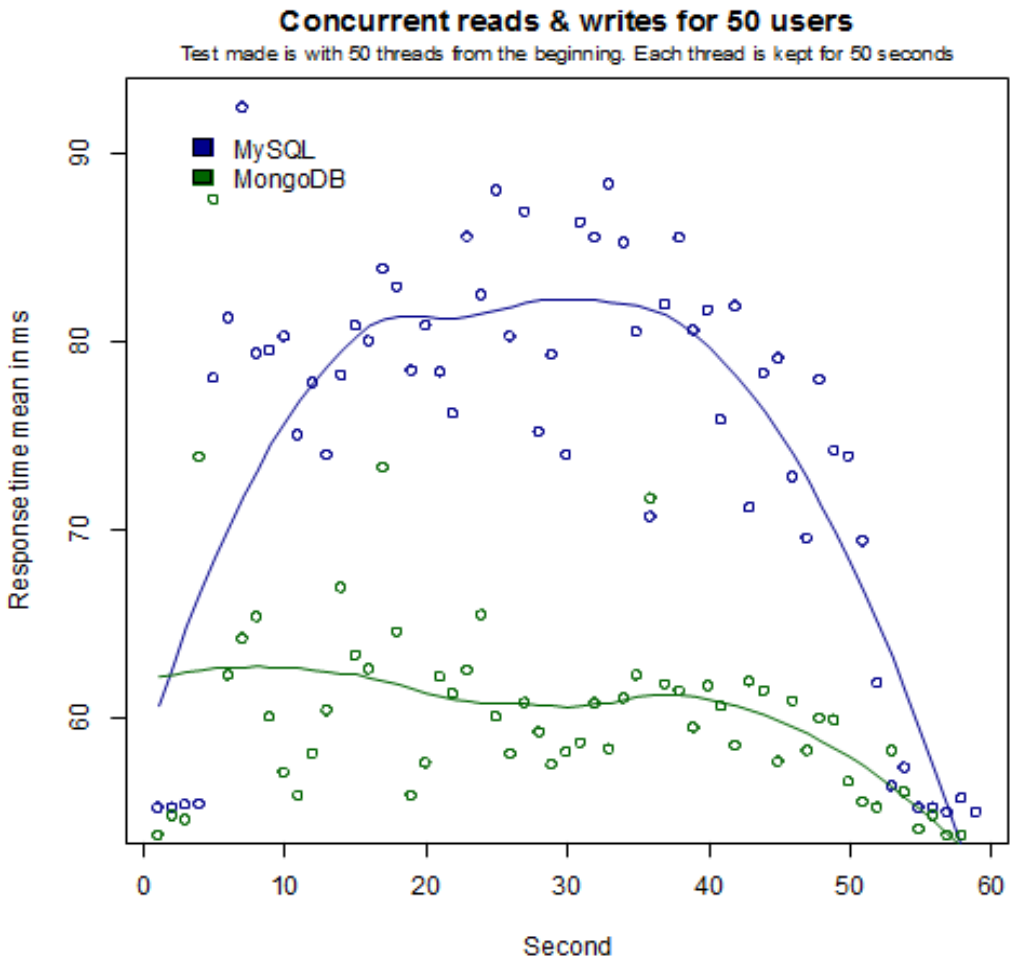


Figura 10. Lectura y escritura concurrente con una cantidad fija de usuarios.

Igual que en el test anterior MongoDB obtiene unos resultados ligeramente superiores en cuanto a rendimiento.

Más información sobre los tests y sus resultados en:
http://www.ciges.net/mongodb-vs-mysql-tests-de-lectura-y-escritura-combinados

Conclusión

Pese al rendimiento de MongoDB, según hemos podido comprobar en la comparativa anterior, se ha optado por MySQL. Uno de los motivos para tomar esta decisión es el rediseño a nivel de todo el testbed IoTWORLD que supondría el cambio del sistema de bases de datos. Además MySQL proporciona una funcionalidad llamada *Events* que es similar a las tareas *CRON* (tareas que se ejecutan cada cierto tiempo) y que pueden resultar muy interesantes a implementar en un futuro para el testbed IoTWORLD.

Por último, y no por ello menos importante, destacar que MongoDB no implementa las propiedades *ACID* (*Atomicity, Consistency, Isolation and Durability*) lo cual no asegura la durabilidad, integridad y consistencia necesarias.

4.5 Representación: C3.js

Es una librería basada en D3 (*Data Driven Document*) que facilita la integración de gráficas en aplicaciones web. Dispone de varias propiedades que se le pueden asignar a las gráficas para personalizarlas sin la necesidad de profundizar tanto en la programación como en D3 acelerando la curva de aprendizaje, si se desea se pueden extender algunas estructuras directamente de D3 para realizar gráficos más complejos.

Facilita varias APIs y *callbacks* para acceder al estado de los gráficos, así se pueden actualizar o modificar funcionalidades inicialmente definidas o reaccionar de forma diferente según la interacción del usuario con los gráficos.

A continuación, explicamos brevemente en qué consiste D3.js:

D3.js

Es una librería de JavaScript que permite crear visualizaciones complejas y gráficos interactivos a partir de datos lanzada en 2011. Básicamente, esta librería permite manipular documentos basados en datos usando estándares abiertos como SVG, HTML5, CSS, etc. evitando así depender de software propietario. Al ser un proyecto de código abierto permite que cualquier desarrollador pueda implementar o mejorar funcionalidades.

4.6 Motor de Plantillas: Jade

Jade es un lenguaje de plantillas HTML desarrollado por el creador de Express.js. Este lenguaje sustituye el lenguaje de marcado HTML en el que se tienen que cerrar las etiquetas HTML por la indentación (sangrado), esto da lugar a archivos mucho más limpios y entendibles, además añade funcionalidades a la hora de generar plantillas HTML para ser rellenadas con contenido dinámico, algunas de estas funcionalidades son:

- Reutilizar bloques de código (*mixins*).
- Herencia, permitiendo definir por ejemplo una plantilla base donde solo cambiara el contenido.
- Ejecutar código *Javascript* en las plantillas para evitar escribir código extra.

En definitiva podríamos definir Jade como un pre-procesador de código HTML.

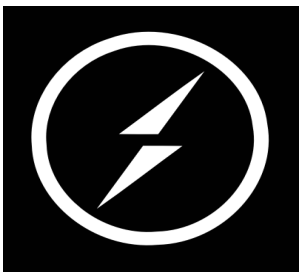
4.7 *CSS: Bootstrap*

Bootstrap es uno de los frameworks CSS, HTML y *Javascript* más famosos y más utilizados del momento para diseño de aplicaciones o sitios web. Inicialmente fue creado por trabajadores de Twitter para fomentar la consistencia entre sus diferentes aplicaciones y finalmente en 2011 se liberó el código a todo el mundo convirtiéndose pocos meses después en el proyecto más popular de *GitHub*.

Está formado por una serie de plantillas de diseño base sobre las cuales existen múltiples variaciones que añaden o modifican funcionalidades. Incluye también múltiples tipografías, botones, formularios (incluyendo validación de formulario), menús y prácticamente cualquier otro elemento genérico utilizado dentro de una aplicación web.

Una de sus mayores características es el hecho de que adapta el contenido de la aplicación web al tamaño del dispositivo en el que se esté visualizando. Es decir, la aplicación web se adapta automáticamente al tamaño, resolución y escala de los múltiples dispositivos desde donde se puede acceder a ella.

4.8 Socket.io



Socket.io es una librería de *Javascript* para aplicaciones en tiempo real. Esta librería habilita la comunicación bilateral entre el cliente y el servidor.

Está formada por dos partes, la parte que se ejecuta en el navegador del usuario, es decir, la del lado del cliente y la del lado del servidor Node.js.

Esta comunicación bidireccional se basa en los *websockets* definidos por la W3C para HTML5, tecnología que proporciona un canal de comunicación entre el cliente y el servidor mediante un socket TCP. Este canal de comunicación bidireccional también permite crear “salas” o aislar ciertos tipos de mensajes para que solamente sean enviados a los clientes que deseemos.

5. Implementación

5.1 Arquitectura general del proyecto

El proyecto está divido en dos partes, la parte del cliente que está desarrollada en *Javascript* sobre jQuery y la parte del servidor que también está desarrollada en *Javascript* pero esta vez sobre *Express* y con partes de *Javascript* puro.

Para el servidor utilizamos el modelo de interfaz *API* (*Application Programming Interfaces*), una interfaz orientada a desarrolladores para servir los recursos del servidor mediante un conjunto de funciones o procedimientos. Al implementar esta interfaz obtenemos la posibilidad de que en un futuro otra aplicación pueda acceder a gran parte de las funcionalidades de la aplicación sin necesidad de volver a programarlas.

En cuanto al cliente se ha basado en jQuery y su capacidad de interactuar con el código *HTML*, manipular el *DOM (Document Object Model),* manejar eventos y desarrollar animaciones. Con todo esto el cliente es capaz de llamar a las funciones de la *API* necesarias para visualizar el contenido deseado.

5.2 Servidor

El servidor es el encargado de gestionar toda la información de la aplicación, el uso del *framework* Express nos facilita mucha de las tareas a realizar, a continuación se detalla la estructura interna del proyecto.

Por la propia estructura del *framework* y la flexibilidad que proporciona resulta algo complicado implementar el patrón *MVC* (Modelo Vista Controlador). Las peticiones en Express están servidas por los archivos que se encuentran en la carpeta de *routes*, en estos archivos se definen las posibles rutas disponibles en la aplicación y las posibles restricciones que se podrían aplicar como en el caso de la administración. Más adelante se describe más en detalle su funcionamiento y que peticiones gestiona cada uno de los archivos.

5.2.1 Modelos

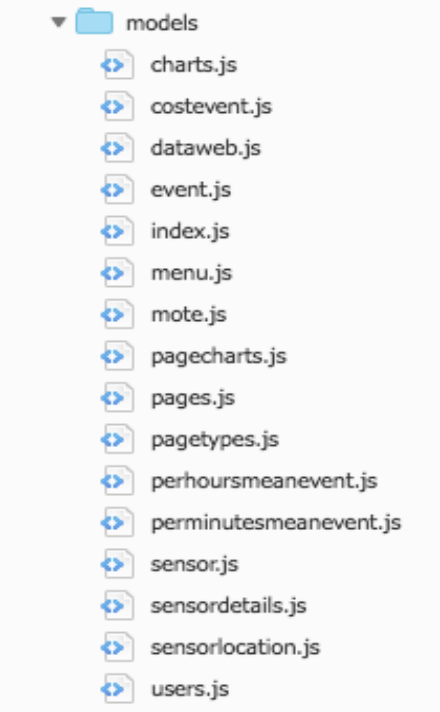


Figura 11. Estructura de la carpeta *models*

Para gestionar las tablas desde Express hemos recurrido a un *ORM* (*Object-Relational Mapping*) llamado *Sequelize*, esto nos permite gestionar las tablas de la base de datos como si se tratase de objetos. Los modelos utilizados en la aplicación se encuentran en la carpeta *models* y en cada archivo de esta carpeta se encuentra una clase que mapea una tabla de la base de datos incluyendo sus relaciones con otras tablas.

El archivo *index.js* es el encargado de conectar a la base de datos y establecer las relaciones entre las tablas de la misma y los archivos con sus clases de nuestro proyecto. Por lo tanto, cada vez que necesitemos acceder a la base de datos en cualquier controlador de nuestra aplicación solo deberemos incluir la carpeta *models* mediante la siguiente directiva:

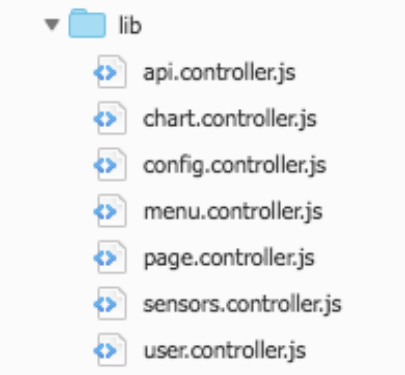
var models = require('../models');

En caso de querer generar un nuevo modelo simplemente será necesario incluir un archivo en la carpeta *models* con la clase de la tabla a relacionar y sus posibles relaciones con otras tablas.

Los datos de acceso a la base de datos de los diferentes entornos están definidos en un archivo de formato *JSON* en la carpeta *config*. En caso de cambiar el entorno sobre el que trabajaremos debemos modificar la variable que define en qué entorno estamos.

5.2.2 Controlador

En el directorio *lib* (ver Figura 12) se encuentran los controladores, en ellos se realiza todas las consultas y validaciones antes de obtener o enviar información a los modelos. Los controladores son llamados por los archivos de la carpeta *routes* que se describen más adelante.



Cada uno de los archivos localizados en la carpeta de controladores se encarga de gestionar una parte de la información de la web. Estos controladores pueden ser cargados individualmente a diferencia del caso de los modelos, esto es así porque no en todos los casos será necesario cargar todos los modelos y al tratarse de archivos relativamente grandes es un gasto de recursos del que podemos prescindir.

Figura 12. Estructura de la carpeta *lib*

A continuación una breve descripción de la información que gestiona cada controlador:

- **api.controller.js:** Gestiona todas las operaciones relacionadas con información de los sensores, tanto la carga inicial de un gráfico como la obtención de los datos en tiempo real en los gráficos que lo requieran. Obtiene los valores de cantidad de elementos a enviar y elementos a actualizar del *config.controller*.

- **chart.controller.js:** Gestiona las operaciones de listado y guarda la actualización o eliminación de un gráfico. Cuando se asocia un gráfico con una página también se recurre a este controlador para que se encargue de la operación.

- **config.controller.js:** Se encarga de cargar y actualizar la configuración global de la aplicación. Esta configuración no se almacena en base de datos sino en un archivo de formato *JSON* que gestiona este controlador.

- **menu.controller.js:** Controla todas las operaciones que se realizan sobre los elementos del menú, desde su carga en la vista del cliente, su listado en la sección de administración correspondiente a la creación o actualización de elementos.

- **page.controller.js:** Controla las operaciones realizadas sobre las páginas de la aplicación web como la búsqueda de una página en función de su dirección para servir las peticiones del cliente hasta el listado, creación o actualización de páginas. En caso de no encontrar un elemento por su dirección el *router* será el encargado de desviar la petición hasta la página de error 404.

- **sensors.controller.js:** Gestiona las operaciones sobre sensores que se realizan en la página principal de la administración.

- **user.controller.js:** Se encarga de comprobar que los datos introducidos por el usuario para acceder a la sección de administración son correctos.

5.2.3 Vistas

Las vistas de la aplicación están almacenadas en la carpeta *views* (ver Figura 13) y están generadas mediante Jade, que nos permite crear un *template* (plantilla) base que define la estructura de la página sobre el cual únicamente modificar el contenido de la página. Se ha dividido los archivos de la administración y las plantillas de las páginas de gráficos para organizar mejor el sistema de archivos y hacerlo más entendible. A continuación se describen los archivos más relevantes de esta sección:

- **layout.jade:** Plantilla base sobre la que se estructura toda la página, se compone de cabecera, menú superior y *footer*.

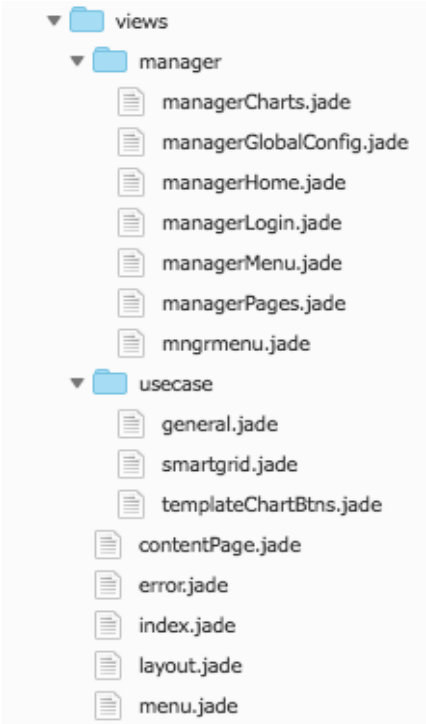


Figura 13. Estructura de la carpeta de vistas

- **menu.jade:** Menú de la aplicación, recibe la información del controlador de menú y carga la estructura del menú sobre la plantilla base *layout*.

- **contentPage.jade:** plantilla sobre la que se cargan las páginas de tipo contenido, cuando se procesa una petición esta plantilla recibe la información de la página y la carga sobre ella y esta, a su vez, sobre la plantilla base *layout*.

- **/manager:** En esta carpeta se alojan todas las vistas correspondientes a la administración, cada página de la administración consta de una vista separada.

- **/usecase:** En esta carpeta se almacenan las plantillas de las páginas de tipo gráficos hasta el momento desarrolladas. Estas plantillas cargan la información sobre las peticiones realizadas y cargan también los recursos necesarios para la visualización de los gráficos.

5.2.4 *Routes*

Los archivos alojados en este directorio son los encargados de gestionar las diferentes peticiones que se realizarán sobre la página. Estas peticiones están divididas en función de la dirección sobre la que se haga la petición. De esta manera, cada archivo se encarga de gestionar un tipo determinado de peticiones. Dicha gestión consiste en recibir una petición, obtener la información necesaria para procesar la petición de un controlador determinado y, por último, devolver los resultados de la petición al cliente.

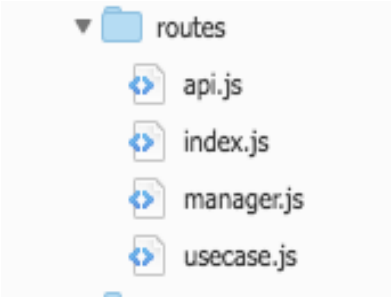


Figura 14. Estructura de la carpeta routes

- **api.js:** Gestiona todas las peticiones relacionadas con las llamadas a la API. Las peticiones que únicamente devuelve información en formato *JSON* son procesadas por el método *GET*, mientras que las peticiones que comportan enviar información al servidor son procesada con el método *POST*.

- **index.js:** Gestiona las peticiones sobre la URL base como puede ser la home, la página de contacto o la galería de fotos. En resumen se encargara de resolver cualquier petición en que la dirección no comience por "*/use-cases/*" o "*/manager*".

- **manager.js:** Gestiona todas las peticiones relacionadas con la administración de la aplicación. Como en la gestión de la API, las peticiones que únicamente devuelvan información se realizan mediante el método *GET*, mientras que las peticiones que impliquen enviar información al servidor son procesadas con el método *POST*.

- **usecase.js:** Gestiona las peticiones bajo la URL base "*/use-cases/*", también controla el envío de información según el periodo de tiempo definido en la configuración global de la aplicación mediante el uso de *socket.io.*

En cualquiera de los documentos antes mencionados en caso de intentar procesar una petición y que esta sea errónea, ya sea porque la página no existe o este mal escrita la dirección, el sistema redirigirá al usuario hacia una pantalla mostrando el error 404.

5.3 Cliente

Para la creación del cliente se ha utilizado la librería *jQuery* para facilitar la interacción con los elementos HTML de la página y C3.js como librería de representación gráfica.

La parte del cliente se divide en dos, las funcionalidades que se ejecutarán cuando un usuario anónimo este utilizando la aplicación y las funcionalidades que el administrador podrá utilizar una vez haya iniciado sesión.

Los archivos relacionados con el cliente se encuentran en la carpeta: **/public/javascript/**

5.3.1 Usuario anónimo

Cuando un usuario realiza una petición sobre una página con gráficos, la aplicación recupera todos los gráficos relacionados con esta página. Una vez recuperada esta información se carga la página y se realiza una serie de peticiones asíncronas a la API para cargar la información necesaria para inicializar los gráficos.

Los archivos sobre los que están implementadas las funcionalidades del usuario anónimo son los siguientes:

- **scripts.js:** Al cargar la página se encarga de llamar a la función necesaria del archivo c3-chart.js para iniciar los gráficos que correspondan con la página. Además implementa las funcionalidades de los botones relacionados con los gráficos.
- **c3-chart.js:** Contiene las funciones encargadas de iniciar los gráficos con las propiedades definidas y realizar la actualización de estos gráficos una vez se recibe la información desde el socket en las gráficas en tiempo real.
- **Plantillas predefinidas:** Sobre las plantillas predefinidas de los gráficos también se ejecuta código *JavaScript*, que está relacionado con la actualización de datos en tiempo real y con la identificación de gráficos según la página actual.

5.3.2 Administrador

La sección de administración está puramente basada en *jQuery* que nos permite manipular eventos y trabajar con elementos HTML. La implementación se basa en el lanzamiento de eventos que realizan una acción sobre el servidor y una vez finalizada esta acción manipular elementos de la página para informar al usuario del resultado.
Los archivos que se encargan de implementar las funcionalidades del administrador son:

- **adminscripts.js:** Este archivo es el encargado de gestionar todas las acciones posibles del administrador sobre la aplicación web.
- **ckeditor.js:** Es un plugin de *JavaScript* que implementa un editor de texto completo y personalizable para hacer la entrada de contenido de texto en una aplicación más fácil y rápida.

6. Pruebas y resultados

En este capítulo resumimos los resultados finales de la aplicación web. Para acceder a la aplicación se debe visitar la siguiente dirección:

http://monitoring.iotworld.cttc.es

Para acceder a la administración se debe iniciar sesión con los datos de administrador por defecto:

- **Dirección:** http://monitoring.iotworld.cttc.es/manager
- **Usuario/password:** admin/admin

Se ha comprobado que la aplicación es compatible con los siguientes dispositivos y navegadores:

- PC (Windows):
  - Chrome 51.0.2704
  - Internet Explorer 11
  - Mozilla Firefox 47.0.1
- Mac:
  - Chrome 51.0.2704
  - Safari 9.1
  - Firefox 47.0
  - Opera 37.0
- Smartphone
  - Android 5.1 Chrome
  - Iphone 5, Iphone 6 Safari
  - Iphone 5, Iphone 6 – Chrome
  - Iphone 6 – Firefox
- Tableta
  - Ipad 2 - Safari
  - Ipad Air 2 - Safari

Durante los test realizados entre navegadores se ha observado que existen algunas incompatibilidades entre funciones de *Javascript* y algunos navegadores como es el caso de Internet Explorer 11. Para resolver esta incidencia se ha optado por implementar la función incompatible utilizando otro método similar que si es compatible y que está disponible gracias al uso de la librería jQuery. El método que ha generado la incompatibilidad consiste en una función de Javascript que permite la búsqueda sobre arrays de objetos nativamente, esta implementación es relativamente nueva y esto hace que se incompatible con ciertos navegadores.

6.1 Apariencia de la aplicación

A continuación se muestra una serie de imágenes que muestran la apariencia final de la aplicación en sus visualizaciones para móvil y tableta:

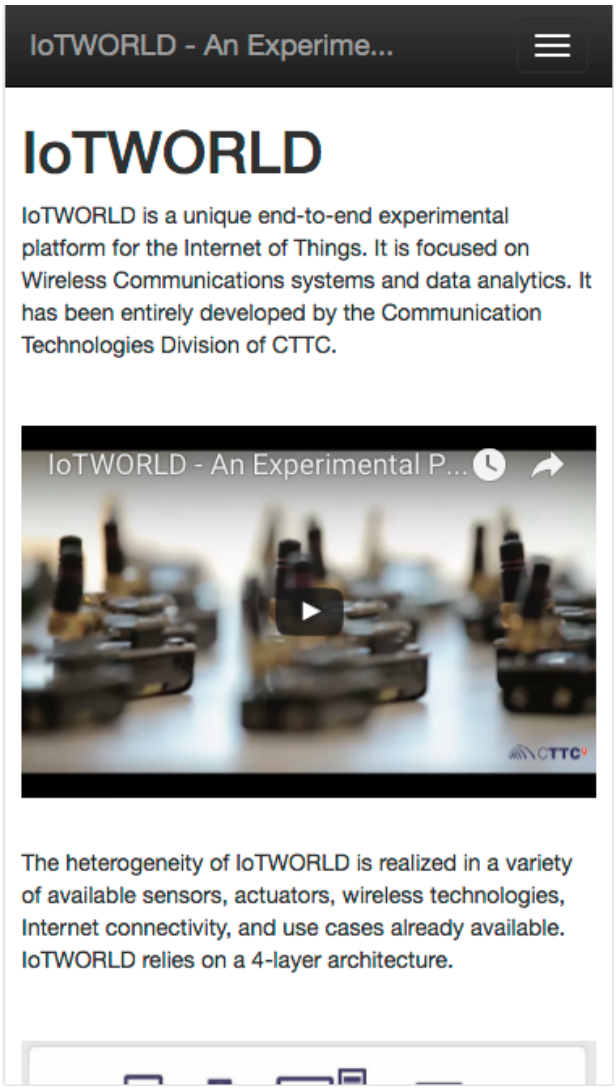


Figura 15. Pagina de inicio

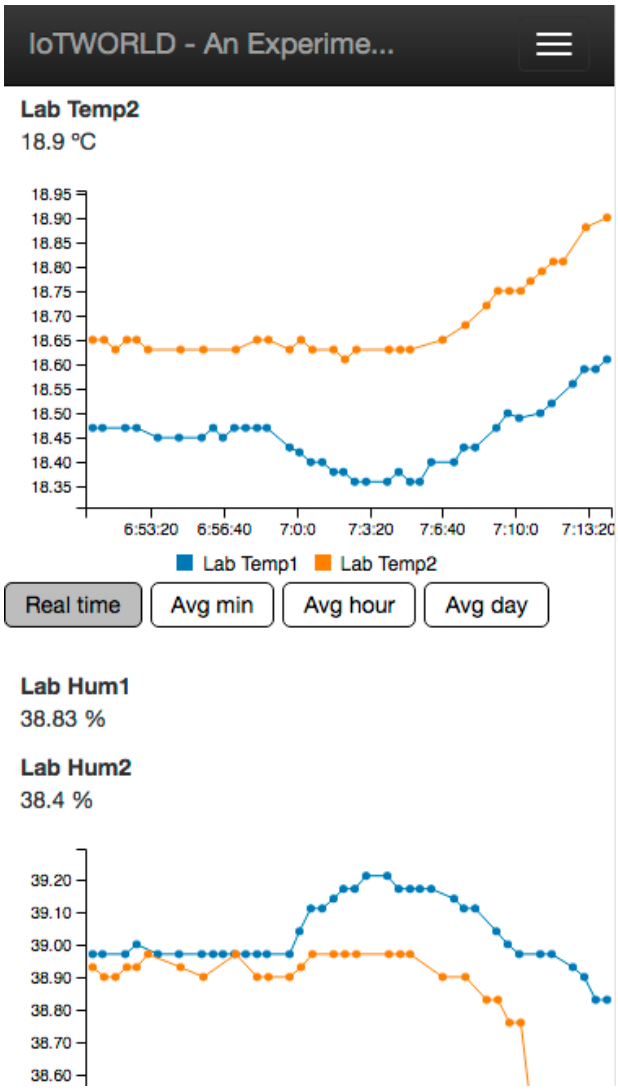


Figura 16. Use case Air Quality

En las Figuras 15 y 16 se observa como la visualización del contenido de la aplicación se adapta en ambos casos al tamaño de la pantalla del dispositivo, en este caso un *smartphone.*

En las Figuras 17 y 18 podemos observar lo mismo pero en este caso las visualizaciones corresponden a un dispositivo tipo tableta.

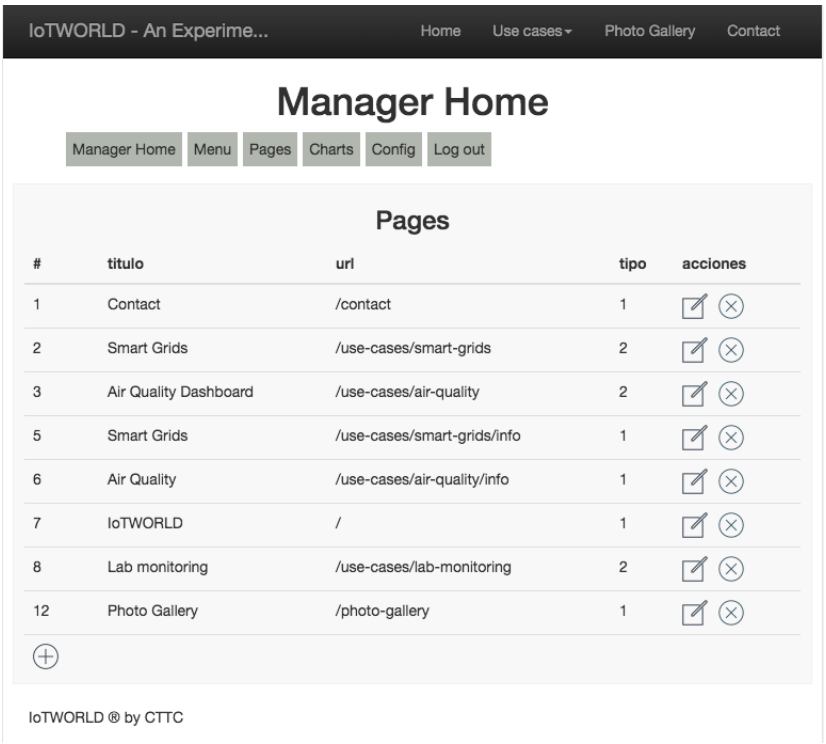


Figura 17. Sección de administrador versión tableta.

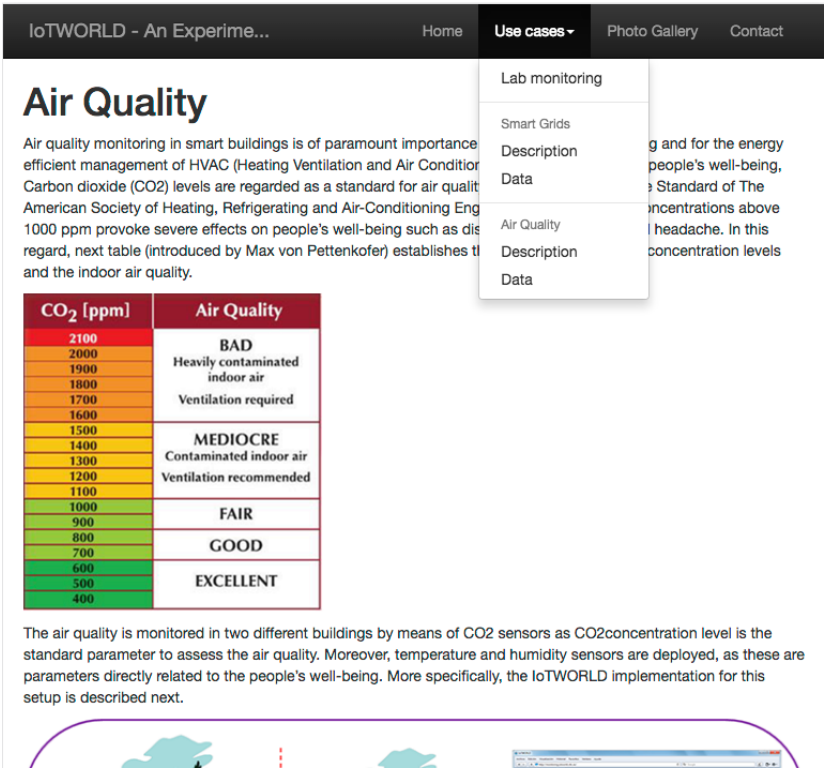


Figura 18. Página de contenido versión tableta.

6.2 Resultados

Después de las pruebas de compatibilidad de navegadores, tanto de funcionalidad como de visualización, se considera que se ha conseguido el objetivo de hacer la aplicación compatible con una gran cantidad de navegadores tanto en sus versiones para escritorio como para dispositivos portátiles.

El estilo final de la aplicación esta altamente influenciado por el uso de *Bootstrap*, los colores y los iconos utilizados hacen que que sea fácil entender el funcionamiento de la aplicación.

En cuanto a las pruebas realizadas los tiempos de carga de la aplicación han disminuido notablemente con respecto a la versión anterior de la aplicación, si bien es cierto que estos tiempos se ven altamente influenciados por la configuración de la aplicación y por el estado de la red en el momento de hacer las peticiones

7. Conclusiones

7.1 Conclusiones generales

En el proyecto se proponía el diseño e implementación de una aplicación web que tuviese un buen rendimiento en cuanto a tiempos de carga y fluidez en la navegación, que fuese compatible con el mayor número de navegadores posibles y que permitiese gestionar los datos generados por el *testbed* IoTWORLD del CTTC.

Mi implicación en este Trabajo de Final de Grado me ha permitido conocer un poco más en profundidad el mundo del IoT, sus capacidades y su increíble potencial. A nivel técnico me ha permitido adquirir una serie de conocimientos que me resulta muy enriquecedores y los cuales me gustaría poder seguir potenciando en un futuro.

El estar involucrado en todas las fases de desarrollo del proyecto también me ha hecho ver que los trabajos de análisis y documentación son tareas complejas que resultan altamente útiles para agilizar el desarrollo.

En conclusión, gracias a este trabajo he podido descubrir una pequeña parte del mundo del IoT y he podido ampliar mis conocimientos de las aplicaciones web.

7.2 Posibles mejoras y trabajo futuro

Este proyecto se ha creado como un primer paso de una aplicación la cual podría tener un largo recorrido. El haber creado una API permite que se puedan integrar otras aplicaciones a esta misma, como podrían ser aplicaciones nativas para dispositivos móviles.

Algunas de las mejoras que se podrían implementar en un futuro en la aplicación serían:

- Migración de cálculos de medias a MySQL:

  Desde su versión 5.1 MySQL cuenta con una funcionalidad llamada Eventos. Estos eventos permiten ejecutar procedimientos almacenados o funciones de MySQL cada cierto tiempo. Ya que los datos de las medias actualmente se calculan gracias a tareas *CRON* que se ejecutan cada cierto tiempo, se podría aprovechar esta funcionalidad de MySQL para incluir estos cálculos dentro de la base de datos.

- Replicación automática entre la BBDD local y la disponible en Google Cloud Platform:

  Al configurar esta replicación automática de la base de datos conseguiríamos tener la información segura ya que siempre se contaría con un respaldo en caso de fallo del servidor local asegurando la persistencia de la información.

- Añadir funcionalidades al administrador:

  Se podrían incluir nuevas funcionalidades para el administrador, estas consistirían en dotarlo de mayor capacidad de personalización de la plataforma como, por ejemplo, añadir la funcionalidad de crear galerías fotográficas, crear páginas que mezclen contenido y gráficos a la vez, mayor libertad a la hora de crear nuevos elementos de menú y un largo etcétera.

8. Bibliografía

- GitHub
  https://github.com/

- Node.js
  https://nodejs.org/en/
  https://www.ibm.com/developerworks/ssa/opensource/library/os-nodejs/

- NPM
  https://www.npmjs.com/

- PHP
  http://php.net/

- MySQL
  https://www.mysql.com/

- MongoDB
  https://www.mongodb.com/es

- Express
  http://expressjs.com/

- C3.js
  http://c3js.org/

- D3.js
  https://d3js.org/

- Jade
  http://jade-lang.com/

- Bootstrap
  http://getbootstrap.com/

- Socket.io
  http://socket.io/

- Cloud 9
  https://c9.io/

9. Anexos

9.1 Manual de usuario

Manual de usuario
CMS IoTWORLD

Autor: Iñaki Úbeda Martínez

Índice

1. Introducción

2. Acceso

3. *Home*
    a. Crear una nueva localización
    b. Crear un nuevo sensor
    c. Activar/desactivar sensores

4. *Menu*
    a. Crear un nuevo ítem
    b. Editar un ítem
    c. Eliminar un ítem

5. *Pages*
    a. Crear una nueva página
    b. Editar una página
    c. Asociar gráficos a una página
    d. Eliminar una página

6. Gráficos
    a. Crear un nuevo gráfico
    b. Editar un grafico
    c. Eliminar una grafico

7. Configuración global

1. Introducción

En este manual se explica todo lo necesario para utilizar la administración del CMS IoTWORLD.

2. Acceso

Para acceder a la administración se deberá disponer de los datos de acceso del usuario, actualmente solo existe un usuario con el nombre “admin”. Las siguientes URLs nos permitirán acceder a la administración:

http://monitoring.iotworld.cttc.es/manager
o
http://monitoring.iotworld.cttc.es/manager/login

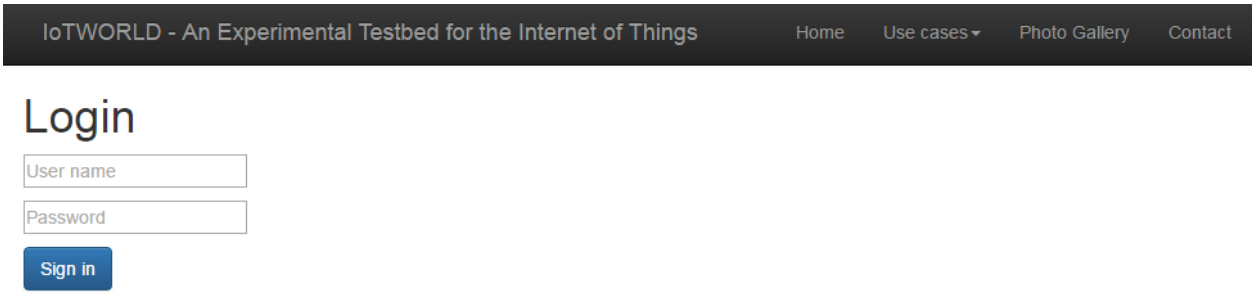


Figura 1. Página de login

En caso de estar logueado y que se finalice la sesión (después de un largo periodo de inactividad desde la entrada) el sistema redirigirá al usuario a la página de *login*.

Una vez el usuario haya introducido correctamente los datos de acceso el sistema redirigirá al usuario a la *home* de administración.

3. *Home*

Desde la *Home* de la administración se pueden realizar las siguientes acciones:

a) Crear una nueva localización.
b) Crear un nuevo sensor.
c) Activar/desactivar un sensor.

Algunos requisitos a tener en cuenta en el momento de realizar las acciones anteriores:

- Un sensor debe estar relacionado con una localización.
- Una localización sólo puede tener un sensor asociado.
- En caso de tener varios sensores en una misma localización se deberán crear con nombres diferentes (por ejemplo,. Lab temp1, Lab temp2,...).
- Activar/desactivar un sensor solo afecta a la visualización de los gráficos, es decir, si el sensor está desactivado pero está recogiendo datos no se mostrará y viceversa. Por lo tanto, se puede dar la situación de tener un sensor representado en los gráficos (activo) pero que no tenga información.

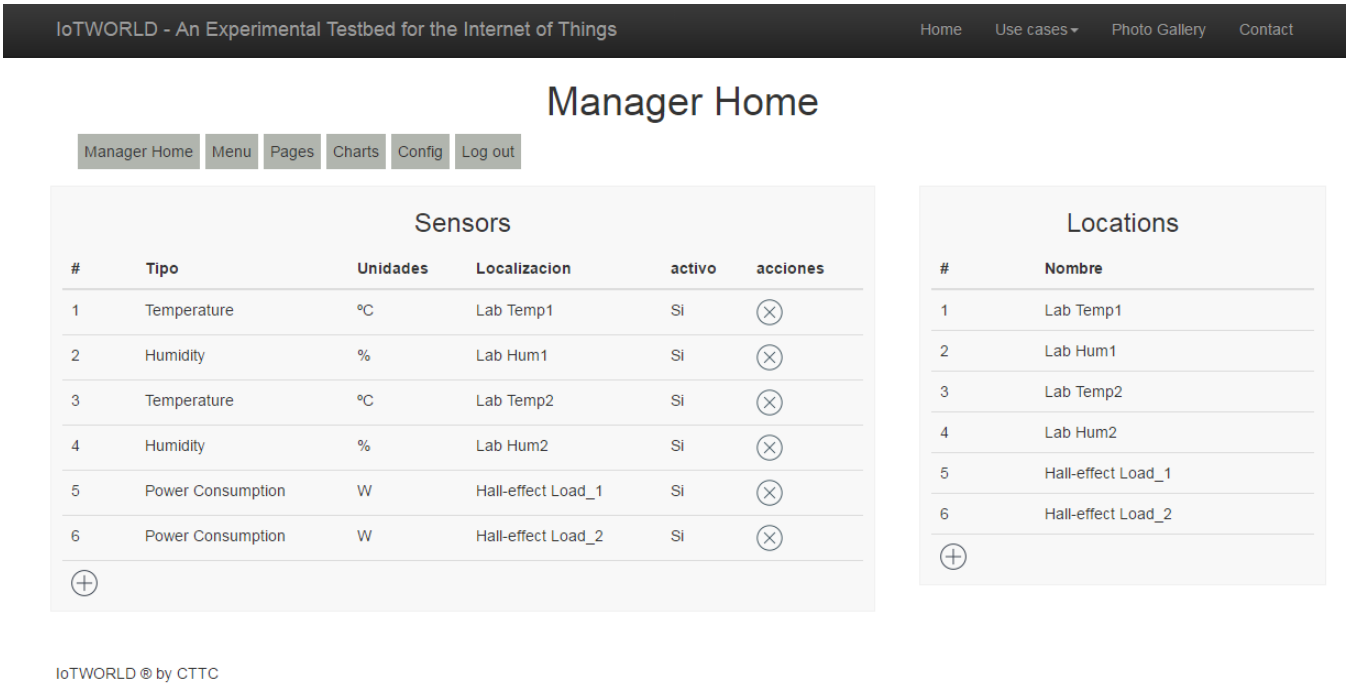


Figura 2. Vista de la *Home* de administración

a. Crear una nueva localización.

Para crear una localización se debe pulsar en el botón con el símbolo más (+) en el panel de *Locations* de la *home* de administración. Una vez clicado se abrirá un cuadro de texto para introducir el nombre de la localización a crear.

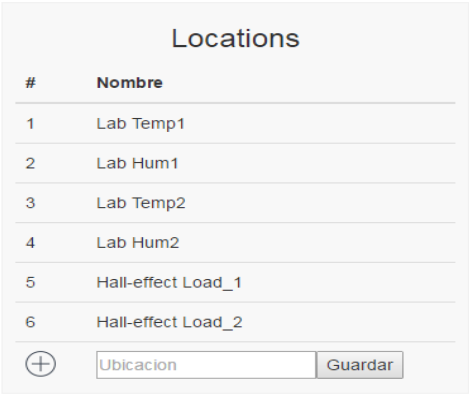


Figura 3. Formulario nueva localización.

Una vez introducido el nombre pulsamos en el botón guardar y se añadirá directamente al listado.

b. Crear un nuevo sensor.

Para crear un sensor se debe pulsar en el botón con el símbolo más (+) en el panel de *Sensors* de la *home* de administración. Una vez clicado se abrirán dos desplegables para definir el tipo de sensor y la localización.

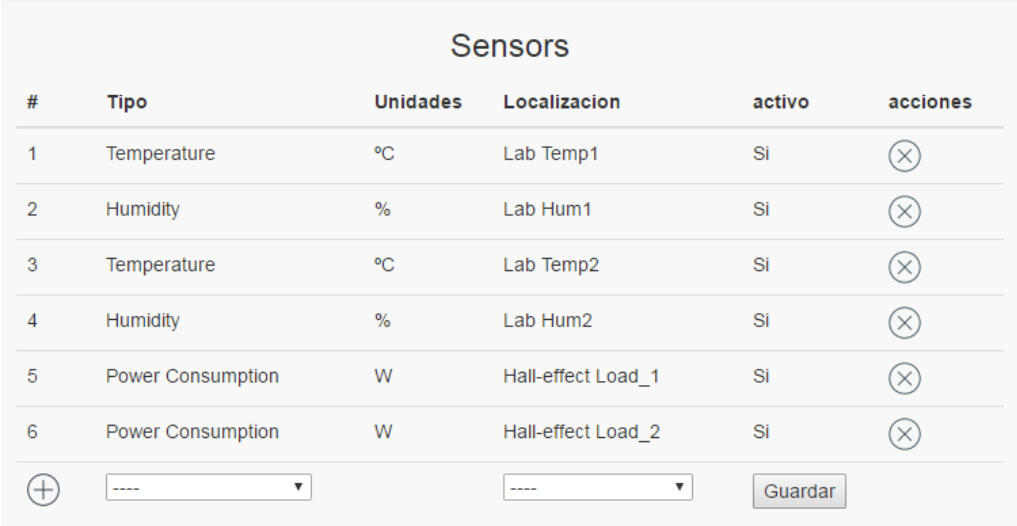


Figura 4. Formulario nuevo sensor.

Una vez seleccionado el tipo de sensor y la localización pulsamos en el botón “guardar” y el nuevo sensor se añadirá directamente al listado.

c. Activar/desactivar un sensor.

Desde el listado de sensores podemos activar, si el sensor no esta activo, o desactivar un sensor. Para ello disponemos de los botones en la columna de acciones de cada sensor.

Figura 5 y 6. Iconos para activar y desactivar un sensor, respectivamente.

4. *Menu*

Desde la sección de *Menu* se pueden realizar las siguientes acciones:

a) Crear un nuevo ítem.
b) Editar un ítem.
c) Eliminar un ítem.

Aclaraciones a tener en cuenta a la hora de gestionar los ítems de menú:

- Existen elementos no editables, estos solo se pueden modificar desde la base de datos.
- Los nuevos ítems creados irán siempre como hijos del ítem padre “use-cases”.
- La URL introducida debe empezar siempre por “/use-cases/” excepto si se quiere crear una cabecera de menú.
- Al introducir un ítem de menú con la URL “#” este pasará a formar una cabecera de menú como en los ítems “Smart Grids” y “Air Quality”.
- Un ítem de menú debe ir relacionado con una página, en caso de no coincidir las URLs introducidas no se podrá ver el contenido de la página.

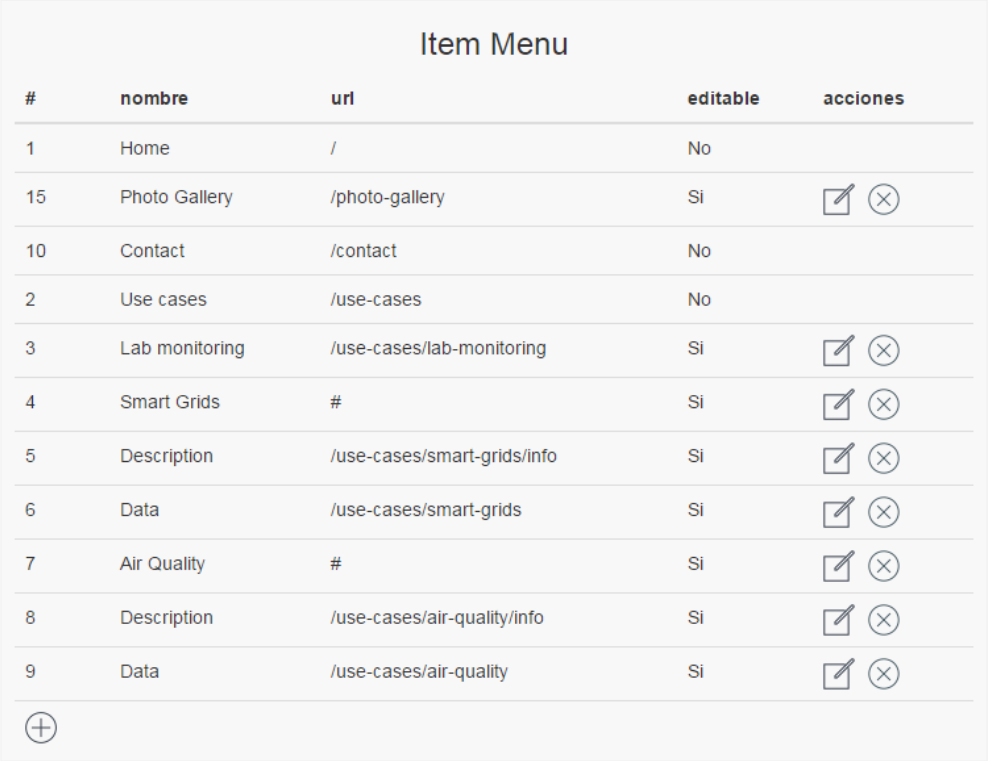


Figura 7. Panel de administración de los ítems.

a. Crear un ítem de menú.

Para crear un ítem se debe pulsar en el botón con el símbolo más (+) en el panel de Ítem Menú de la sección Menú. Una vez clicado se abrirá un panel en la parte inferior de la pantalla desde donde podremos definir el nombre del ítem, la URL del ítem y su padre (por defecto “use cases”), una vez pulsemos en ‘Guardar’ se recargará la página con el elemento añadido en el listado.

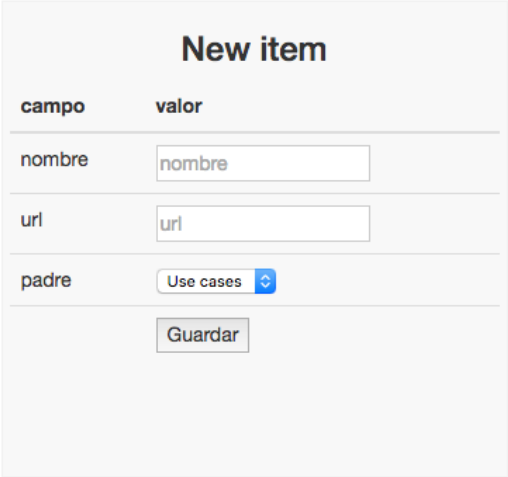


Figura 8. Formulario de nuevo ítem de menú.

b. Editar un ítem.

Para editar un ítem ya creado debemos pulsar en el botón de la columna de acciones siguiente.

Figura 9. Botón de editar.

Una vez pulsado se abrirá el panel inferior, el mismo que al crear un nuevo elemento, con los datos del ítem para poder modificarlos. Una vez editamos los datos pulsamos en guardar y estos se actualizarán al momento.

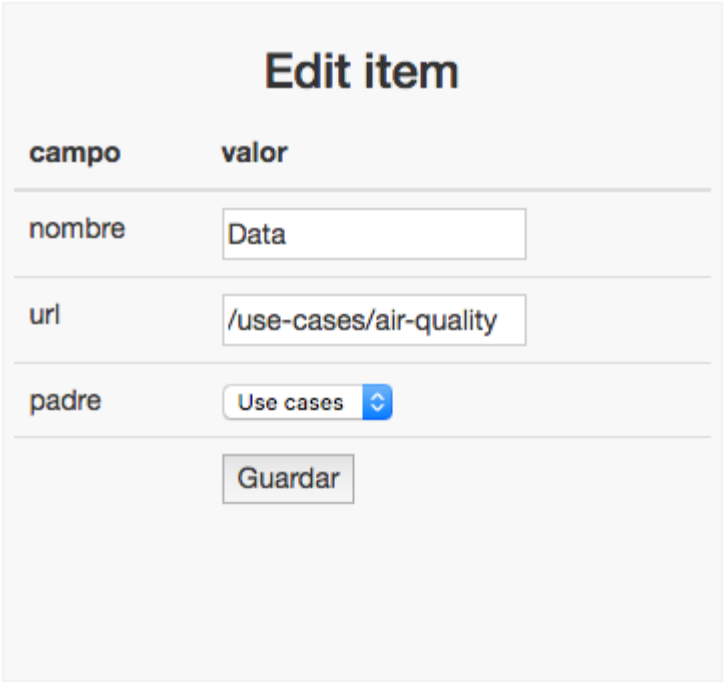


Figura 10. Formulario de edición de ítem de menú.

c. Eliminar un ítem.

Para eliminar un ítem de menú debemos pulsar en el botón de la columna de acciones siguiente:

Figura 11. Botón de eliminar.

Una vez pulsado se actualizará el listado automáticamente eliminando el ítem.

5. *Pages*

Desde la sección de Pages se pueden realizar las siguientes acciones:

a) Crear una nueva página
b) Editar una página
c) Asociar gráficos a una página
d) Eliminar una página

Aclaraciones a tener en cuenta a la hora de gestionar las páginas:

- Existen dos tipos de páginas: de contenido y de gráficos.
- En las páginas de contenido para incluir imágenes se deben haber subido al servidor previamente y se debe indicar la URL en el editor.
- Al crear una página de gráficos se debe seleccionar un tipo de plantilla.
- Solo se pueden asociar gráficos a una página de tipo gráficos al editar.

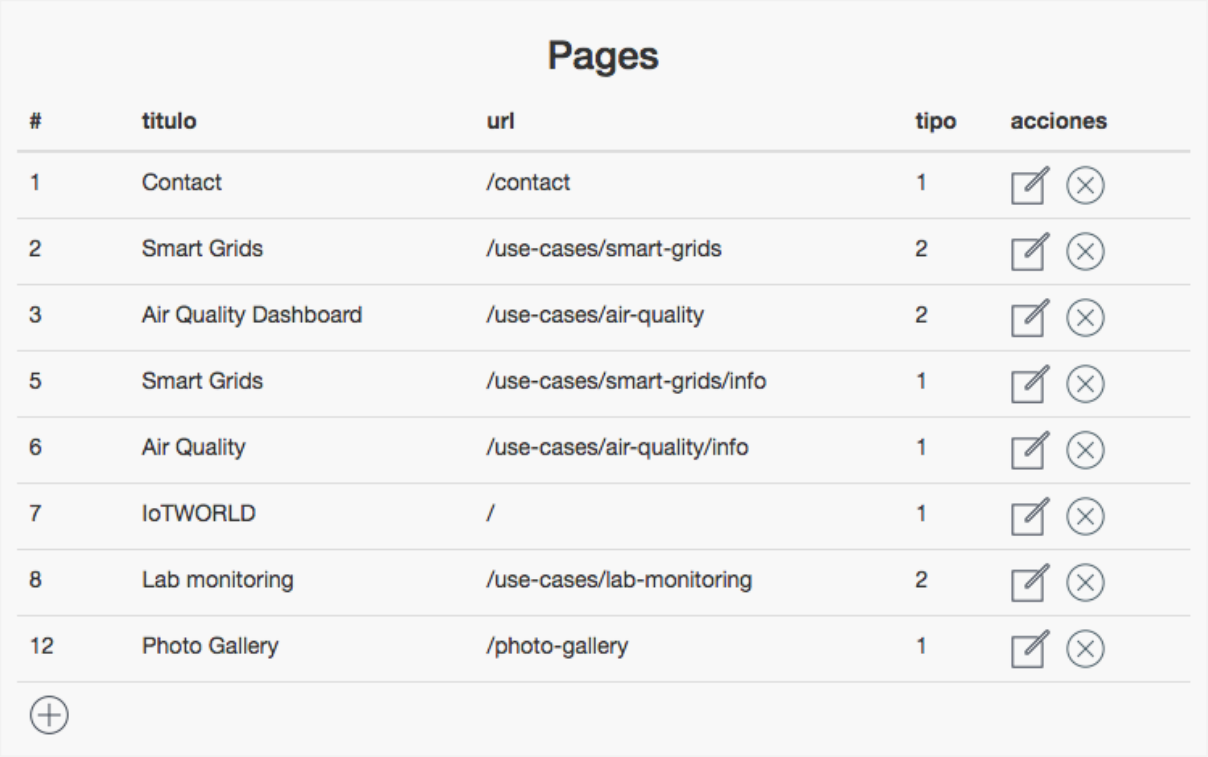


Figura 11. Panel de administración de páginas.

a. Crear una nueva página.

Para crear una nueva página se debe pulsar en el botón con el símbolo más (+) en el panel de *Pages* de la sección Paginas. Una vez clicado el botón se abrirá el panel de la Figura 12 con los campos para rellenar, en función del tipo de página seleccionado la información a introducir se ampliara como podemos ver en las Figura 13 (pagina de tipo contenido) y en la Figura 14 (pagina del tipo gráficos)

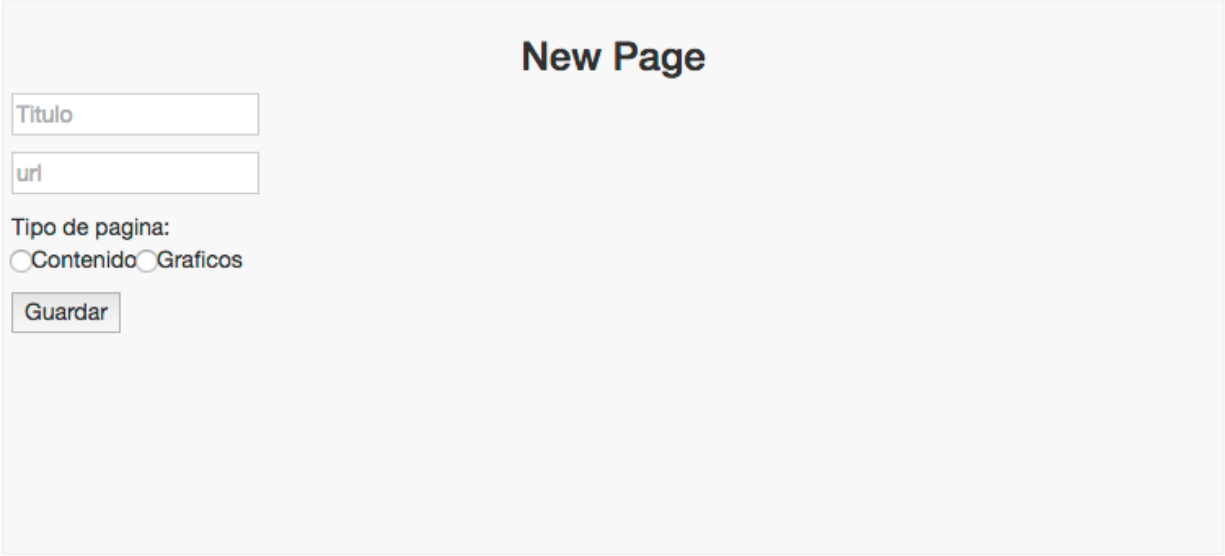


Figura 12. Formulario de nueva página de tipo contenido.

En el caso de las páginas del tipo contenido (Figura 13) se abrirá un cuadro de texto sobre el cual podremos editar el contenido que se quiera.

Para las páginas de tipo gráficos (Figura 14) se muestra un desplegable sobre el cual deberemos seleccionar la plantilla con la que queremos representar los gráficos que asociemos posteriormente a la página.

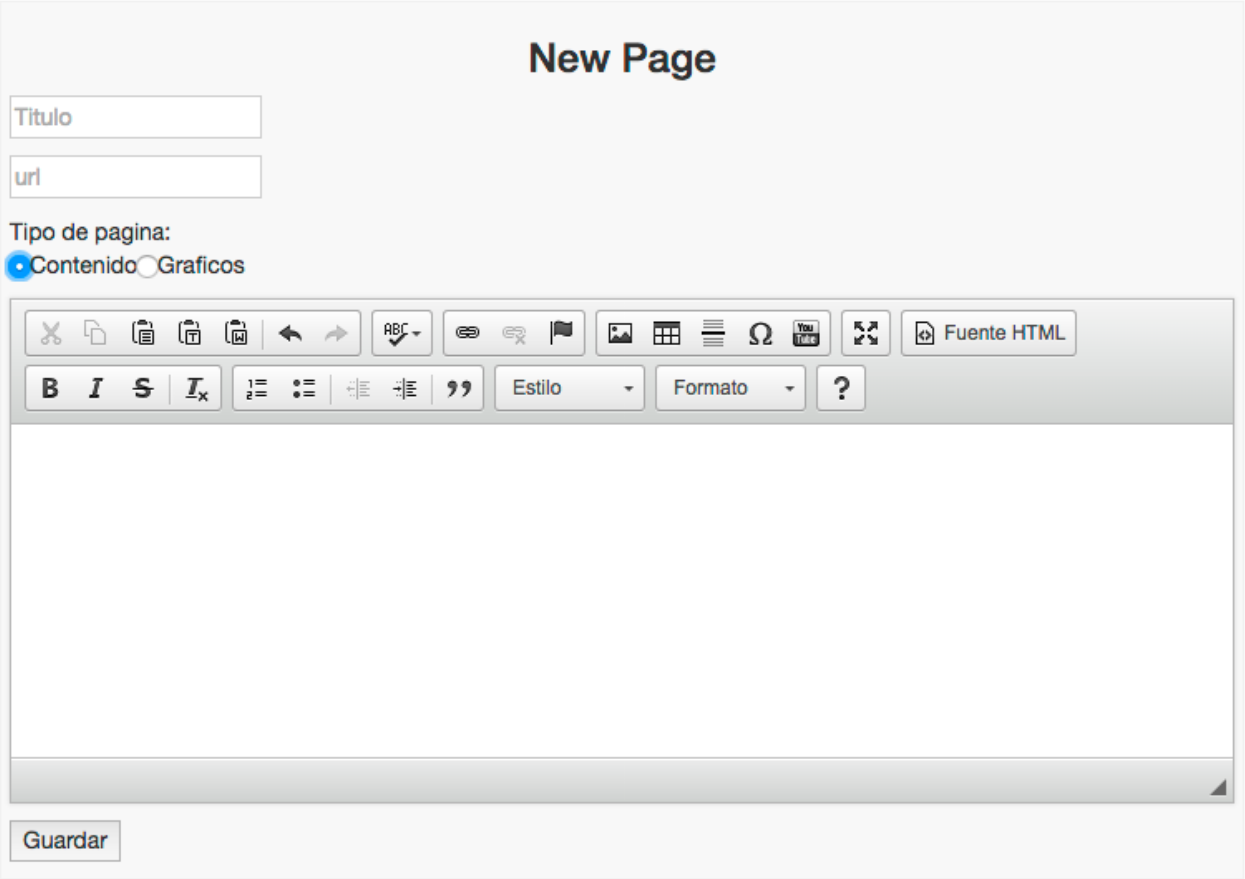


Figura 13. Formulario de nueva página de tipo contenido.

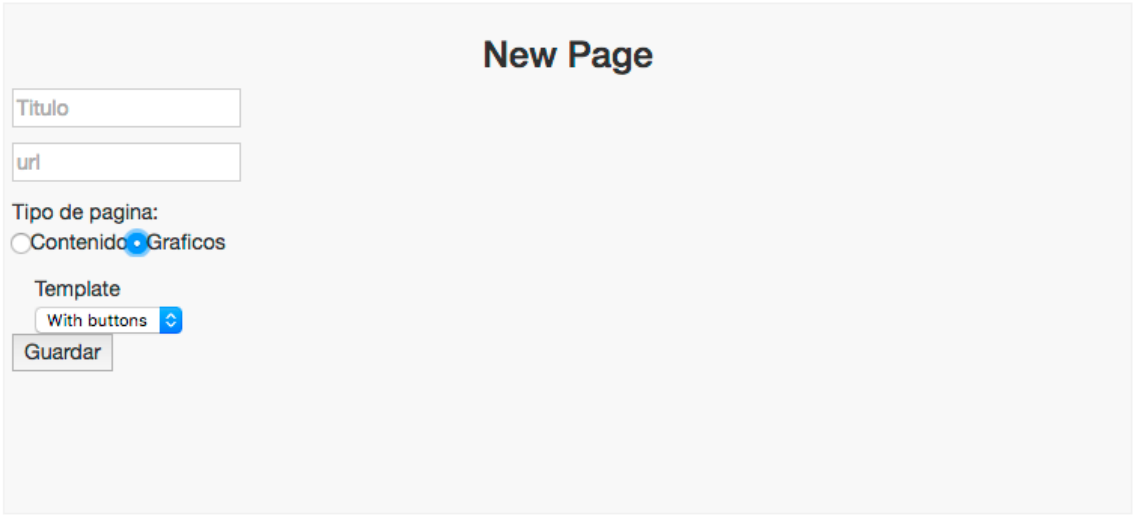


Figura 14. Formulario de nueva página de tipo gráficos.

Una vez pulsemos el botón Guardar se recargara la página con el listado actualizado de las páginas.

b. Editar una página.

Para editar una página ya creada debemos pulsar en el botón de la columna de acciones siguiente.

Figura 15. Botón de editar.

En función del tipo de página se cargará un formulario u otro en el panel inferior, como anteriormente, con los datos de la página cargados. En el caso de las páginas de gráficos se muestra un botón para añadir gráficos y un listado (si tiene gráficos ya asociados).

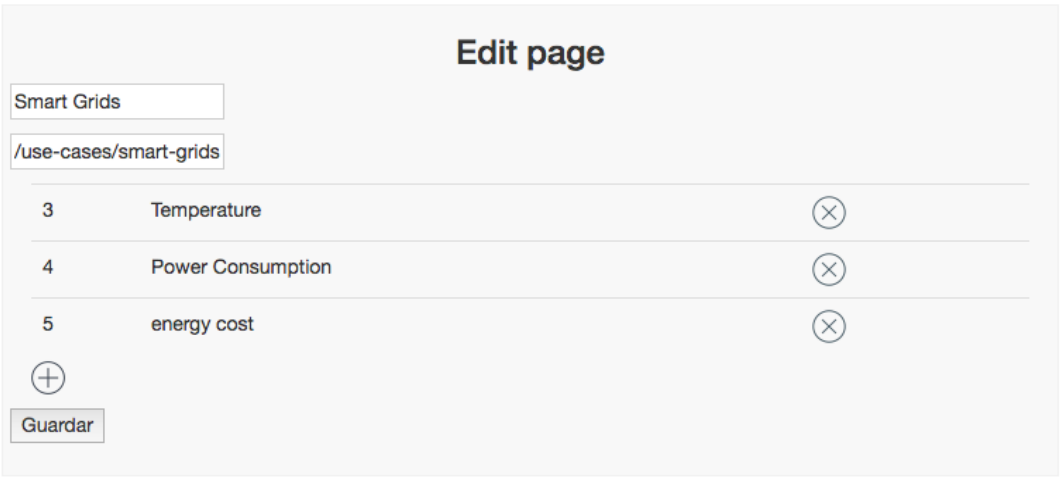


Figura 16. Formulario de editar página de tipo gráficos.

Una vez guardemos se actualizará la información automáticamente y se cierra el panel inferior.

c. Asociar gráficos a una página.

Para asociar gráficos a una página debemos editar una página de tipo gráficos. Una vez hayamos pulsado la opción de editar una página deberemos pulsar el botón con el símbolo más (+) en el panel inferior.

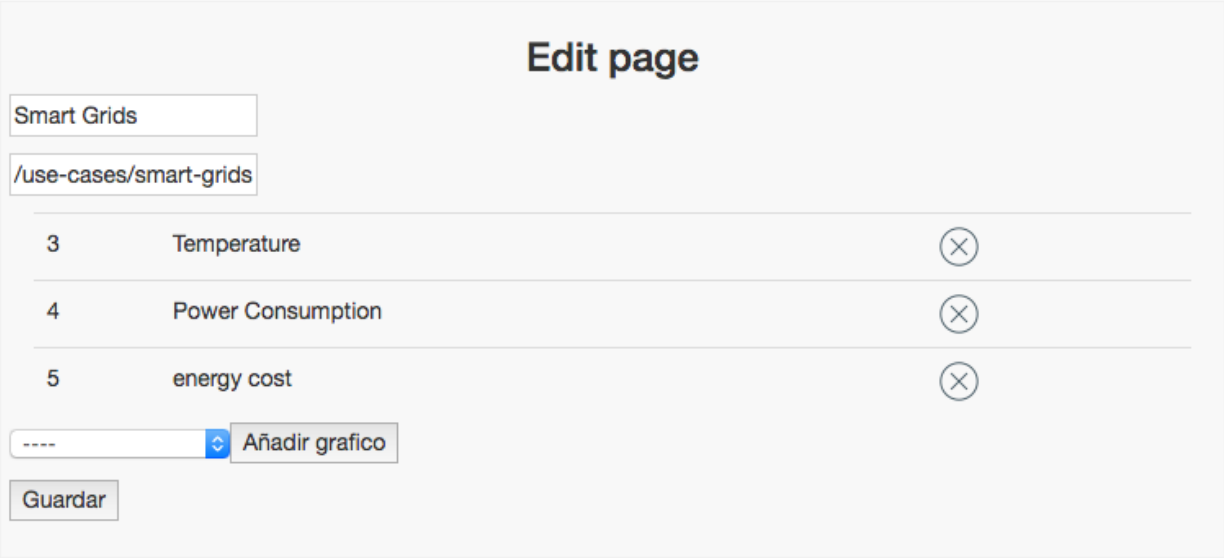


Figura 17. Formulario para asociar gráficos a una página.

Se mostrará un listado con los gráficos disponibles, una vez seleccionado pulsamos el botón “Añadir grafico” y automáticamente éste se asociará a la página. En caso de querer eliminar un gráfico deberemos pulsar en el símbolo(x) de la columna de acciones del listado de gráficos de la página.

d. Eliminar una página.

Para eliminar una página debemos pulsar en el botón de la columna de acciones siguiente:

Figura 18. Botón de eliminar.

Una vez pulsado se actualizará el listado automáticamente eliminando la página.

6. Gráficos

Desde la sección de Gráficos se pueden realizar las siguientes acciones:

a) Crear un nuevo gráfico.
b) Editar un gráfico.
c) Eliminar un gráfico.

Aclaraciones a tener en cuenta a la hora de gestionar los gráficos:

- Se deben definir todos los campos, el único opcional es si se quieren o no botones.
- Los gráficos del tipo *gauge* y *numeric* son exclusivos para la plantilla general y no pueden incluir botones.
- Pueden existir múltiples gráficos para un mismo tipo pero es recomendable ponerle un nombre descriptivo para que al asociar un gráfico a una página no haya confusiones.

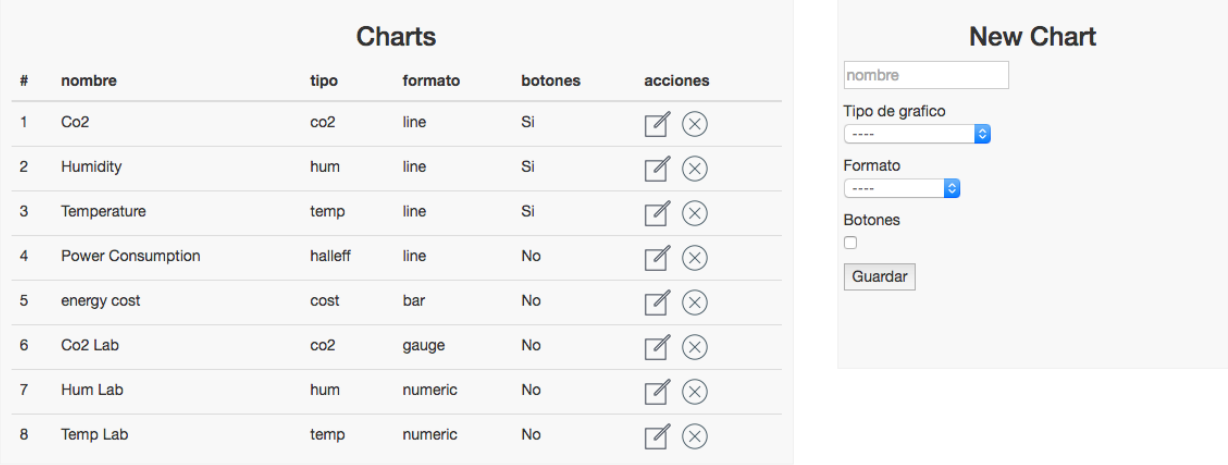


Figura 19. Panel de administración de gráficos.

a. Crear un nuevo gráfico.

Para crear un nuevo gráfico utilizaremos en este caso el formulario disponible en el panel lateral de la Figura 19. Una vez pulsemos el botón guardar se recargara la página con el listado de gráficos actualizado.

b. Editar un gráfico.

Para editar un gráfico debemos pulsar en el botón de la columna de acciones siguiente.

Figura 20. Boton de editar.

Una vez pulsado se cargará toda la información del gráfico en el panel lateral.

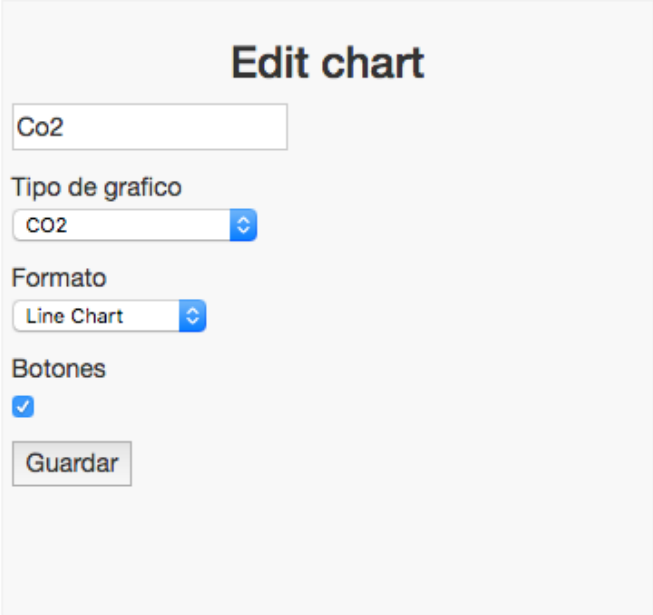


Figura 21. Formulario de edición de un gráfico.

Una vez finalizada la edición pulsaremos el botón guardar y se actualizará la información automáticamente.

c. Eliminar una gráfico.

Para eliminar un gráfico debemos pulsar en el botón de la columna de acciones siguiente:

Figura 22. Botón de eliminar.

Una vez pulsado se actualizará el listado automáticamente eliminando el gráfico.

7. Configuración global

En la sección de configuración global podemos controlar diferentes parámetros de la aplicación, los nombres de estas variables son bastante descriptivos pero a continuación entraremos un poco más en detalle.

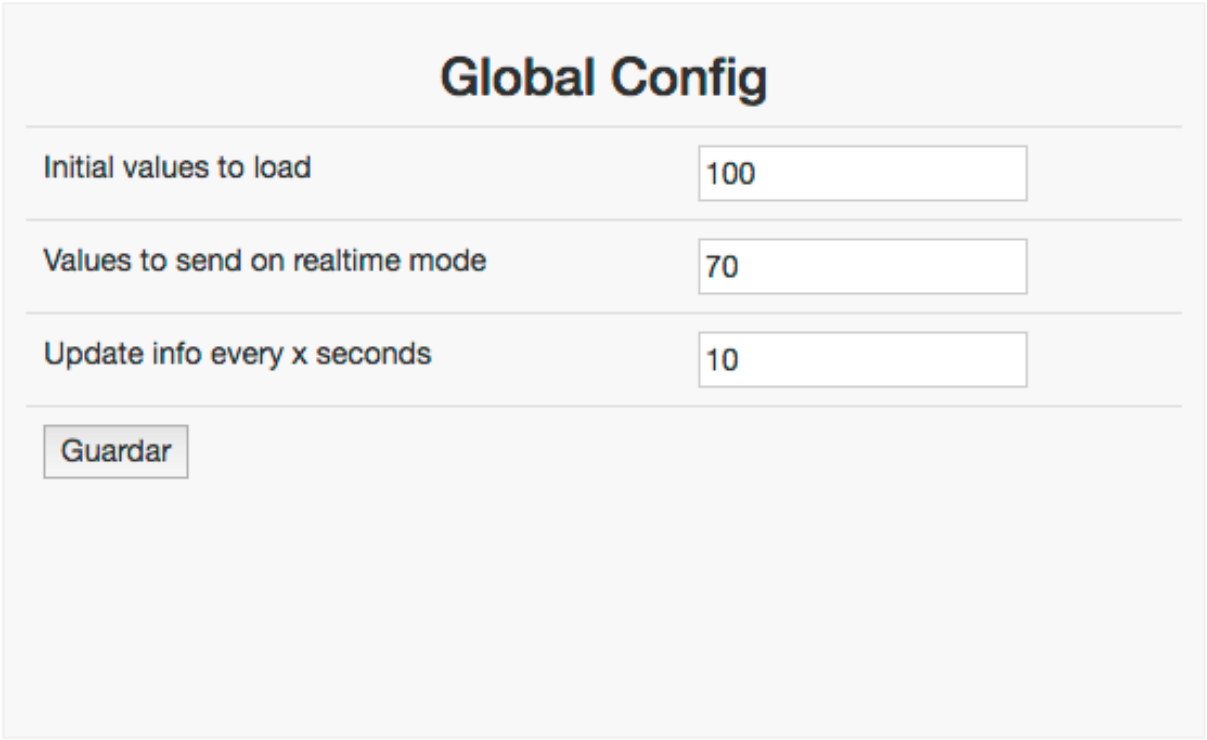


Figura 23. Panel de administración de la configuración global.

- **Número de valores iniciales:** Número de valores que se cargaran en los gráficos de toda la página al entrar a un *dashboard*, cuanto mayor sea el número más específico será el gráfico pero también más tiempo tardará en cargar.

- **Número de valores en real time:** Número de valores que se enviarán cada vez que se actualice la información de un gráfico, como antes, cuanto mayor sea el número más específico será el gráfico pero también tardará más tiempo en cargar.

- **Actualizar información cada x segundos:** Define cada cuantos segundos se actualiza la información en los gráficos de la página, cuanto menor sea el número mayor carga tendrá el servidor y el cliente. El valor recomendado es de 10 segundos.

9.2 Guía de instalación

Para la instalación se parte de la premisa de que están instalados los siguientes recursos:

- npm
- Node.js
- MySQL

1. Descargar el código desde *GitHub*

Ejecutamos el siguiente comando:

$ git clone https://github.com/iubeda/TFG.git iotworld

Esto nos pedirá nuestro usuario y contraseña de GitHub y nos descargara el código de la aplicación en la carpeta *iotworld*

2. Descargar las dependencias necesarias.

Ejecutar el comando para instalar dependencias:

$ cd iotworld
$ npm install

3. Iniciar la aplicación

Ejecutar el comando para iniciar la aplicación:

$ DEBUG=iotworld:3000 npm start

O directamente el comando

npm start