Universidad de Oviedo

Memoria del Trabajo Fin de Máster realizado por

Victoria Villa Valle

para la obtención del título de

Máster en Ingeniería de Automatización e Informática Industrial

Configuración y Visualización con Interfaz Web para infraestructura de computación distribuida

Mayo 2017

Índice

Índice de figuras

1. Glosario

- API *Application Programming Interface* : Conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software.
- Ajax : Se trata de una tecnología asíncrona. Los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la aplicación.
- BackboneJS: Uno de los primero frameworks que implementaron el modelo vista controlador. Permite crear aplicaciones rápidas pero resulta complejo de utilizar.
- Back-end: Tecnologías que corren del lado del servidor (PHP, Python, Java), la parte encargada de interactuar con la base de datos, montar la página en un servidor y otras funcionalidades.
- BootStrap : Es un framework originalmente creado por Twitter, que permite crear interfaces web con CSS y JavaScript cuya particularidad es adaptar la interfaz del sitio web a la del tamaño del dispositivo que se quiera visualizar.
- BDD : Behaviour Driven Development es un proceso de Desarrollo del Software que combina técnicas generales y principios del DGP (Digital Generator Protection) junto con ideas del diseño guiado por el dominio y análisis de diseño de programas orientados a Software.
- DOM *Document Object Model*: Document Object Model. Es una librería API que manipula el documento HTML cargado en el navegador. Permite la gestión de eventos, insertar y eliminar elementos entre otras funcionalidades.Proporciona un conjunto estándar de objetos para representar HTML, XHTML y XML.
- CSS textitCascading Style Sheets : Se trata del lenguaje utilizado para asignar estilos y como se va a presentar la información (en cuanto a decisiones visuales, como puede ser el color) en documentos HTML o XML.
- Full-Stack: Ambos sistemas, front-end [Glosario 1] y back-end [Glosario 1]. Además de los componentes encargados de la integración entre ellos y los distintos sistemas operativos.
- Front-end: Tecnologías que corren del lado del cliente, es decir, del lado del navegador (HTML, CSS, JavaScript)

- MVVM (Model-view-view model): : Un tipo de arquitectura. Separa la interfaz de usuario de la lógica que hay en ella para implementar funcionalidades.. Facilita la separación entre el código de interfaz de usuario con aquel que no sea código de interfaz de usuario

- Plugin : Aplicación que añade una funcionalidad adicional o una característica al software. Un complemento.

- Spaguetti Code : Es un término peyorativo para los programas que están formados por una estructura de control de flujo compleja e incomprensible. Se le llama así por su similitud a un plato de espaguetis : hilos intrincados y anudados.

- URI : Identificador de recursos uniforme. Es una cadena de caracteres que identifica los recursos de una red de forma unívoca. La diferencia respecto a un localizador de recursos uniforme (URL) es que estos hacen referencia a recursos que pueden variar con el tiempo.

2. Introducción

2.1. Motivación

Hoy en día, cada vez es más necesario el uso de las tecnologías web tanto en el mundo empresarial como a la hora de compartir recursos para que otras personas puedan acceder a ellos. Está demostrado que una página web aumenta el incremento de ventas, además de la productividad, siendo por tanto necesario conocer el campo de las tecnologías web para su correcto desarrollo. [1]

Las tecnologías web conectan un conjunto de archivos entrelazados entre sí a los cuales se puede acceder a partir de una interfaz capaz de conectar dichos ficheros con los usuarios, haciendo posible que millones de personas sean capaces de conectarse simultáneamente para compartir información y conocimientos.

A su vez, se deberá presentar la información de forma más ordenada e intuitiva, ya que una página bien diseñada será más atractiva para el usuario. Existen varias tecnologías dentro de este campo, destacando desde los últimos años aquellas denominadas "Front-end"[Glosario 1].

El desarrollo Front-end es una especialidad para el desarrollo web que trabaja la interfaz y permite que el usuario pueda interactuar con la web. Utiliza principalmente lenguajes de marcado y programación. Este último ejecuta las operaciones en el cliente sin necesidad del uso de servidores externos, es decir, manipula los resultados obtenidos para una visión más intuitiva de los datos, evitando de esta forma sobrecargas en el servidor.

Cabe destacar que no hay que confundirlo con el desarrollo Back-end[Glosario1], ya que este está enfocado únicamente a los lenguajes de programación, es decir, trabajar con datos internos creando aplicaciones que sean capaces de controlar los datos y mostrar la información de manera que el usuario pueda comprender.

2.2. Objetivo del trabajo

En el proyecto Sentinel2 E2ESPM (End-To-End System Performance Monitoring) de la empresa Elecnor Deimos se tiene una interfaz desarrollada con tecnologías anticuadas de desarrollo de aplicaciones web [2.2,2.3,2.4], siendo en este caso PHP [2.3.3]. Aunque PHP

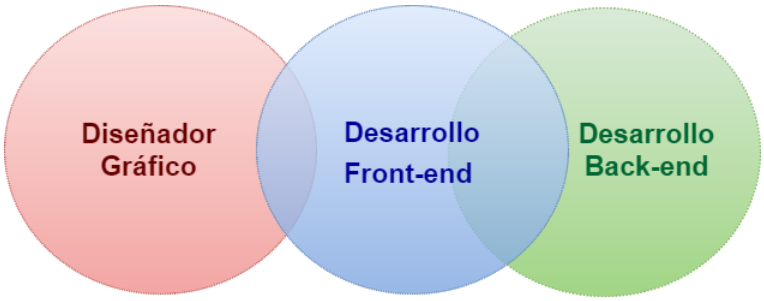

Figura 2.1: Ciencias que influyen en el Desarrollo Web

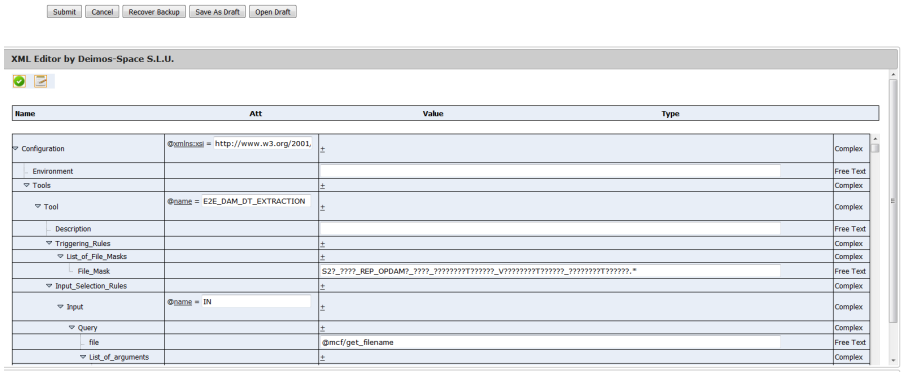

Figura 2.2: Editor XML actual (I)

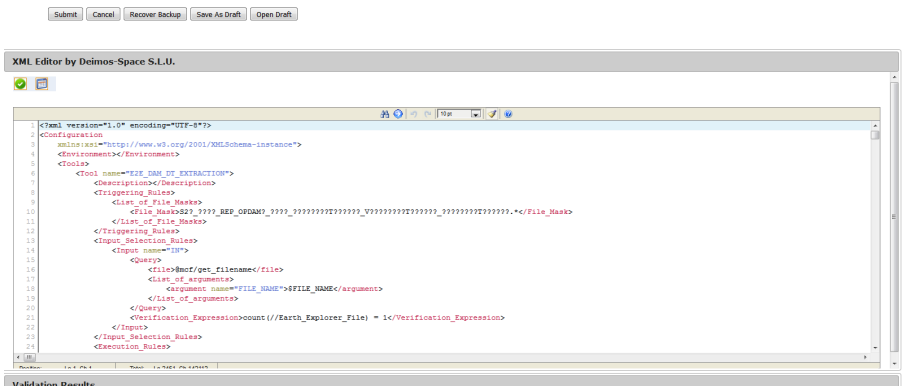

Figura 2.3: Editor XML actual (II)

sea un lenguaje multiplataforma, cuya programación es segura y fiable, también es un lenguaje del lado del servidor siendo necesario que este ejecute el código ante la petición de un cliente, necesitando de esta forma que el servidor envíe la página web a visualizar (lenguaje del lado del servidor [10.2]). De esta forma se produce, al contrario que las tecnologías del lado del cliente, una carga innecesaria en los recursos del servidor, siendo este uno de los motivos por los que se ha optado a migrar la aplicación a una tecnología Front-End como Angular JS. [4]

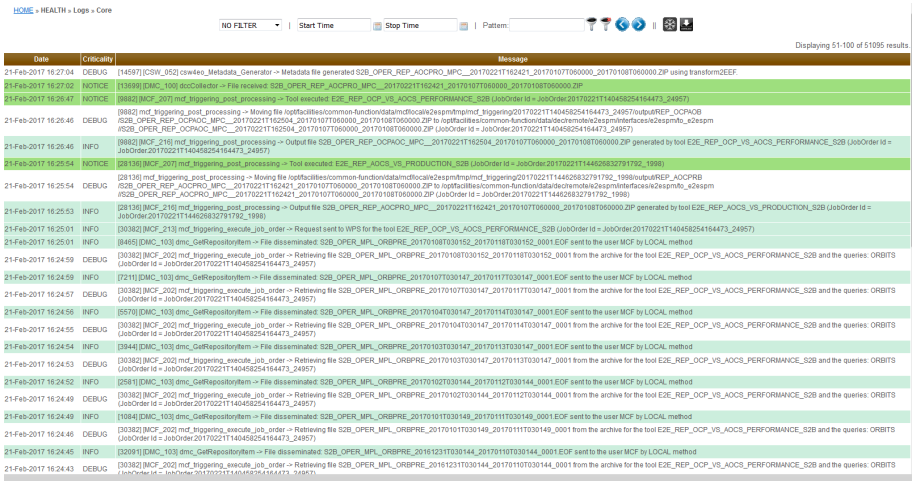

Figura 2.4: Visor de archivos actual

Angular JS es un framework de JavaScript reciente de código abierto creado y mantenido por Google. Se trata de una tecnología que logra fluidez debido a que existe una comunicación entre cliente y servidor transparente, logrando una buena experiencia de usuario, además de que permite la comunicación con tecnologías back-end [Glosario 1] con facilidad (bases de datos, por ejemplo), teniendo de esta forma una integración Full-Stack [Glosario 1] .Como ejemplo de uso de Angular JS se tienen páginas conocidas como Gmail o Amazon.

El objetivo de este proyecto es realizar dos aplicaciones para sustituir a las existentes :

- Un editor de XML interactivo: con funciones adicionales, como selección de elementos de array a visualizar o capacidad de añadir nuevos objetos.
- Visualización de archivos: un visor de archivos “logs” con posibilidad de filtrado

Además, se construirá una API REST (Integrada con Angular JS y llamadas AJAX[Glosario 1) capaz de actualizar los archivos del servidor y enviar aquellos en los cuales se haya realizado cambios.

Se ha elegido el protocolo REST ya que es útil y sencillo de aplicar en sistemas que usen HTTP para obtener datos o realizar operaciones sobre esos datos en todos los formatos posibles, como, por ejemplo, XML y JSON, siendo una alternativa a protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol. SOAP es un protocolo web pesado (hay que montarlo por completo), mientras que REST es un protocolo web ligeros (de fácil implementación porque no requieren definir un estándar al usar HTTP) disponiendo de una gran capacidad pero también mucha complejidad.

Entre las características del protocolo REST destacan:

- Arquitectura cliente/servidor sin estado: las peticiones HTTP contienen toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar un estado anterior para ejecutarla.
- Posee cuatro operaciones importantes relacionadas con los datos y su uso en HTTP:
  - POST, que permite crear un archivo de datos.
  - GET, que permite leer o consultar un archivo.
  - PUT, que permite editar un archivo
  - DELETE, que permite eliminar un archivo.
- Los objetos REST se manipulan a través de la URI [*Uniform Resource Identifier*] [Glosario 1]. La URI es el único elemento identificador de cada recurso de un sistema REST, permitiendo acceder a la información para su modificación o borrado, o para compartir la ubicación con terceros.
- Interfaz uniforme: se utiliza para la transferencia de datos, aplicando las acciones correspondientes (POST,GET,PUT o DELETE) sobre los recursos. De esta forma se facilita la existencia de una interfaz uniforme que sistematiza el proceso con la información.
- Sistema de capas a través de una arquitectura jerárquica entre los componentes, esto quiere que el sistema permite tener una arquitectura compuesta por capas limitando el comportamiento de los componentes, ya que estos no pueden ver más allá de la capa con la que está interactuando.

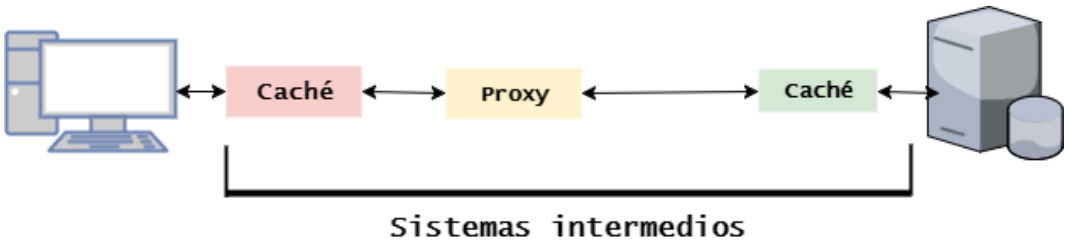

Figura 2.5: Sistema de capas

- Hipermedios: este término hace referencia a la extensión del concepto de hipertexto. Es lo que permite al usuario navegar por los objetos a través de enlaces HTML. En el caso de una API REST, el concepto se refiere a la capacidad de una interfaz de desarrollo de aplicaciones de proporcionar al cliente los enlaces para ejecutar acciones concretas sobre los datos.

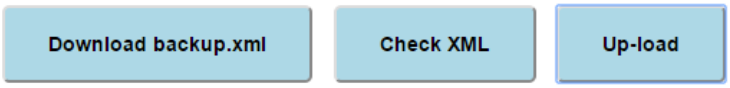

Figura 2.6: Ejemplo de Hipermedios en la aplicación: Upload sería un POST y Download sería un GET

Todo ello, a su vez, se deberá unificar en un menú fácil de utilizar por el usuario que permita iteraciones sin necesidad del uso del servidor, siendo así una aplicación más eficiente, mejorando a su vez la usabilidad y la eficiencia.

2.3. Estado del arte

El desarrollo de aplicaciones web ha evolucionado rápidamente, cada vez es más necesario este tipo de aplicaciones para diversos usos [2.1] Dicha evolución se ha desarrollado tanto desde el punto de vista del desarrollo de software a través de nuevas tecnologías, bibliotecas y arquitecturas, como del punto de vista de administración de sistema a través de sistemas de alojamiento, monitorización o gestores de datos.

Debido a ello, se dispone de numerosas tecnologías para elegir, siendo indispensable un enfoque de alto nivel para seleccionar la tecnología que mejor se ajuste. Los enfoques a tener en cuenta a la hora de realizar una aplicación web son: creación a través de tecnologías de desarrollo y creación con sistemas gestores de contenido.

Cuando se habla de creación a través de tecnologías de desarrollo, dependerán del tipo de aplicación a diseñar, ya que cada una tiene diferentes arquitecturas [Anexo 10], de ello dependerá que se escoja una de las siguientes tecnologías a usar:

- Tecnologías de cliente: Permiten crear interfaces llamativas para el usuario, además de la interacción con el servidor. Esto quiere decir que el cliente tiene que interpretar la respuesta que le da el servidor a su petición. Están basadas en HTML [2.3.1], CSS [2.3.1] y JavaScript [2.3.1] como pueden ser Angular JS y Backbone entre otras.
- Tecnologías de servidor: Permiten implementar el comportamiento de la aplicación en el servidor, siendo este el que genera la información y se la manda al navegador. Se usa cuando hay que compartir información entre usuarios o enviar correos, por ejemplo. En la mayoría de los casos, se utilizan estas tecnologías para ocultar al usuario la lógica del servicio, como pueden ser Django [2.3.3] o Flask [2.3.3] entre otras.

Sin embargo, la creación de webs con sistemas gestores de contenido se realiza a través de un software ya desarrollado que crea páginas web que únicamente varían su aspecto y contenido (siendo la lógica la misma) teniendo como funcionalidad realizar publicidad del contenido a publicar. Ejemplos de ello son las páginas de empresas, blogs u organismos públicos. Algunos ejemplos actuales de gestores de contenidos pueden ser Simple PHP o Wordpress para blogs y Apache Lenya o Mambo para portales.

Otra división a la hora de tomar decisiones para el correcto desarrollo de aplicaciones web, es elegir el lenguaje a utilizar. Dicha elección es una de las más complejas a la hora llevar a cabo el servicio, ya que no todos los lenguajes de programación tienen las mismas funcionalidades. Dentro de este grupo se pueden diferenciar dos grandes subgrupos:

- Tecnologías estandarizadas: El cliente por excelencia en una aplicación web es el propio navegador, existen por tanto un conjunto de estándares definidos por W3C (World Wide Web Consortium) [2] siendo este una comunidad internacional que desarrolla estándares abiertos para el crecimiento de la web, que todo navegador debería implementar, existiendo a su vez algunas tecnologías no estándar usadas para aplicaciones más avanzadas que mejoran la experiencia de usuario. Dentro de este grupo destacan: HTML, CSS, JavaScript y Flash entre otras.
- Tecnologías no estandarizadas : Se utilizan tecnologías propietarias o abiertas, por lo que no será necesario el uso de estándares, ya que cada organización desarrollará la tecnologías que considere adecuada en cada aplicación. Ejemplos de tecnologías no estándar son: Adobe Flash o Microsoft Silverlight.

2.3.1. Tecnologías estándares en el lado del cliente

Dentro de los estándares web, destacan sobre el resto HTML (Hypertext Markup Language) y CSS (Cascading Style Sheets), pero también existen otras tecnologías como JavaScript que se usan como apoyo para el desarrollo de aplicaciones, siendo un complemento a HTML desde los últimos años gracias a su flexibilidad y a una extensa biblioteca de funcionalidades.

- HTML: Proporciona la información estructurada en secciones, párrafos, títulos o imágenes. Supuso una revolución para el dinamismo cliente-servidor porque ofrece muchas librerías y tecnologías avanzadas como las comunicaciones con websockets o la concurrencia con los mismos. Cabe destacar que la versión actual de HTML se trata de la versión 5, habiendo sufrido varios cambios desde sus inicios. Entre los cambios que destacan en cada versión, son el añadido de nuevas etiquetas, atributos y API's[Glosario 1], siendo por tanto una versión mejorada y depurada.

Figura 2.7: Logotipo de HTML

- CSS: Proporciona la distribución de los elementos y su estilo (colores, tipos de letra, fondos y efectos entre otras características). Se usa para definir la presentación de un documento estructurado escrito en HTML, XML y SVG o incluso interfaces de usuario de otras tecnologías como JavaFX.

Figura 2.8: Logotipo de CSS

Actualmente su versión es la CSS3 aunque no esté finalizada.

- JavaScript: Es un lenguaje de scripting que permite que se modifique la página cuando el usuario interactúe con ella (a través del modelo de objetos de documento DOM [Glosario 1]) mediante peticiones al servidor web en segundo plano (AJAX) [Glosario 1], siendo a su vez un lenguaje de tipado dinámico (no es necesario declarar el tipo de variable) y orientado a objetos. Está basado en el lenguaje de programación EC-

Figura 2.9: Logotipo de JavaScript

MAScript de ECMA (organización diferente al W3C). Inicialmente era un lenguaje interpretado pero actualmente se ejecuta con máquinas virtuales en los navegadores. Cabe destacar que JavaScript es un lenguaje optimizado en los navegadores, como por ejemplo en Chrome con su motor v8 [24], lo que hace que sea un lenguaje potente y con un amplio catálogo de recursos, siendo una de las mayores características de JavaScript la multitud de bibliotecas que existen para el desarrollo de aplicaciones, siendo algunas de las más utilizadas las siguientes:

  - jQuery: Es un recubrimiento de la API DOM [Glosario1,1] que aporta facilidad de uso, potencia y compatibilidad entre navegadores. Gestiona la interfaz y las peticiones Ajax [Glosario 1] de la misma.

- Underscore.js: Es una librería para trabajar con estructuras de datos con un enfoque funcional. También gestiona plantillas capaces de generar código HTML partiendo de datos.

A la vez que librerías, también existen *frameworks* de alto nivel que estructuran la aplicación de forma completa, las más populares son las siguientes:

- Angular.js
- Backbone.js
- Ember

En este trabajo, se tratarán de abordar las tecnologías de Angular.js.

2.3.2. Tecnologías no estándares en el lado del cliente

El diseño de webs ha avanzado y evolucionado debido a tecnologías incluidas en los navegadores llamados *plugins* [Glosario 1], convirtiéndose muchos de ellos en estándares. Gracias a la estandarización de HTML5 dichas tecnologías siguen siendo compatibles, aunque con el tiempo entran en desuso ya que suelen ser sustituidas por estándares web o librerías con dicha funcionalidad. Algunas de las más conocidas son:

- Adobe Flash: Aunque se usa en varias aplicaciones, no es eficiente ni de código abierto, por lo cual no se considera una tecnología estándar.
- Java Applets : Precursores de Flash, quedó en desuso por problemas legales de Microsoft además de problemas de seguridad, ya que debido a un fallo de diseño de Java en algunos casos se permitía el acceso de estos al sistema pudiendo realizar tareas maliciosas sin ninguna información de ellos.
- Microsoft Silverlight : Apuesta de Microsoft para competir con Adobe Flash con soporte limitado, con Windows 8 cayó en desuso debido a problemas de incompatibilidad.

2.3.3. Tecnologías del lado del servidor

Existen multitud de tecnologías de construcción de aplicaciones en el servidor, siendo las más usadas PHP, Java EE, ASP.NET, node.js, DJango o Flask.

- Java EE: Es una tecnologías basada en Java, desarrollada por empresas lideradas por Oracle, IBM y Red Hat entre otras, siendo la mayoría de sus implementaciones y herramientas de software libre.

  Una de sus ventajas es que existe una comunidad muy grande de desarrolladores y empresas realizando complementos, bibliotecas y herramientas con Java EE. Debido a ello, tiene una organización de estandarización propia llamada Java Community Process (JCP), en ella se definen estándares abiertos que se pueden implementar tanto con licencia libre como propietaria. A su vez, Java EE tiene un conjunto de frameworks y bibliotecas muy amplio, entre los que destacan Spring, Hibernate, GWT y Vaadin.

  Toda aplicación Java EE debe ejecutarse con un servidor de aplicaciones Java, existiendo muchos tipos. Entre ellos destacan : Glassfish (Oracle), Tomcat (Apache), Jetty (Eclipse), JBoss (RedHat), WebSphere (IBM) ó WbLogic (Oracle).

- PHP [3]: Es la tecnología del lado del servidor con la cual se han implementado más servidores de Internet, siendo ésta multiplataforma. Normalmente se integra con Apache y MySQL en entornos Linux.

  Las páginas diseñadas con PHP contienen HTML con código incrustado, siendo un código muy fácil de implementar que a su vez permite realizar características avanzadas para programadores profesionales.

  Igual que Java EE, una de sus grandes ventajas es que existen multitud de frameworks para el desarrollo de PHP, como por ejemplo, CakePHP.

- ASP.NET: Es un framework comercializado por MicroSoft, siendo la sucesora de ASP (Active Server Pages). Permite escribir código usando cualquier lenguaje admitido por el .Net Framework. Suele usarse para el desarrollo de formularios.

  Tiene una comunidad más limitada, siendo la mayoría de sus librerías proporcionadas por MicroSoft. Entre ellas destacan: Web Pages, Web Forms ó Data Access Layer (DAL).

- Node.js [5]: es un entorno de ejecución para JavaScript construido con el motor de JavaScript versión 8 de Chrome. Fue creado para ser útil con programas de red altamente escalables, como por ejemplo, servidores web. Node.js usa un modelo de opciones entrada/salida sin bloqueo orientado a eventos, de esta forma logra ser liviano y eficiente. A su vez, destaca a su gran abanico de librerías de código abierto, siendo la más grande del mundo.

- Django [6]: Es un framework para aplicaciones web gratuito y con código abierto desarrollado en Python. Su principal característica es facilitar la creación de sitios

web complejos, poniendo énfasis en el re-uso, la conectividad y la extensibilidad de componentes para desarrollar de manera ágil y sin repetición de código. Algunas de las páginas conocidas que usan Django son [7]: Disqus, Instagram, Mozilla, National Geographic o Pinterest

- Flask [8]: Es un framework, al igual que Django, escrito en Python. Está basado el *Werkzeug* [25], que permite crear aplicaciones web rápidamente y con un número mínimo de líneas de código, ya que incluye un servidor web de desarrollo de manera que se pueden probar las aplicaciones desarrolladas sin necesidad de instalar Apache[26] o Nginx [27]. Sus principales características son:
  - Propio servidor web
  - Debbuging
  - Fácil desarrollo
  - Gran cantidad de documentación
  - Compatibilidad con WSGI 1.0
  - Soporte integrado para hacer test unitarios.

3. Integración del sistema

La función de la aplicación actual es realizar una serie de transformaciones en imágenes recibidas por un satélite, representando la información de estas a través de diferentes cálculos para que el usuario pueda observar la información. Existen diferentes métodos para representar dicha información, que se muestra a través de módulos denominados “reports”[3.1].

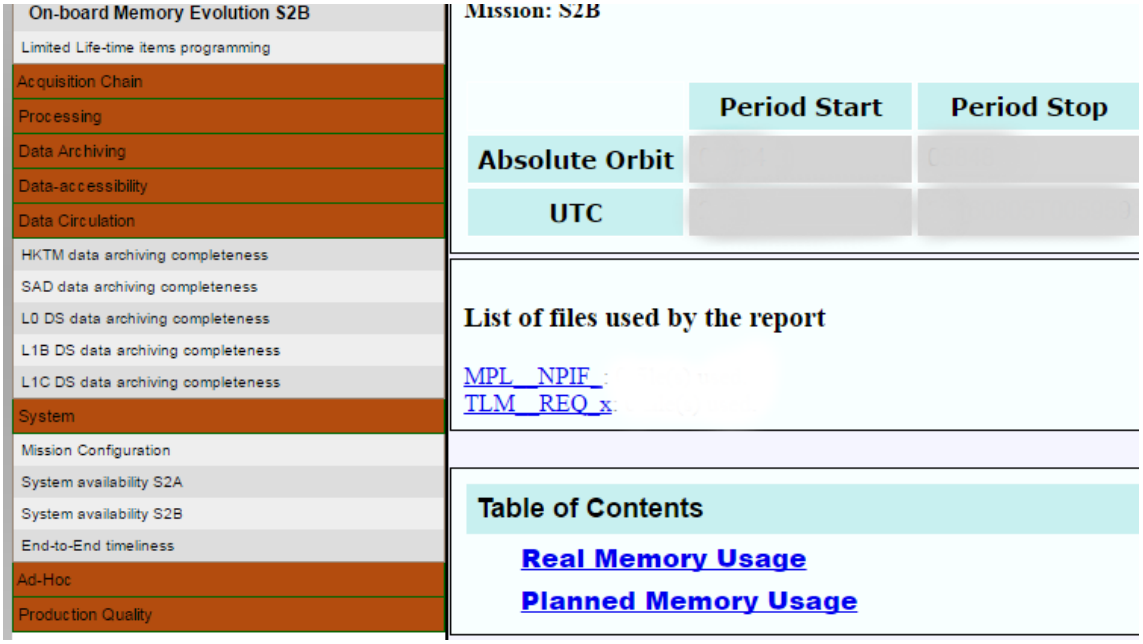

Figura 3.1: Ejemplos de “reports”

El desarrollo de este trabajo, tal y como se ha nombrado anteriormente, tiene como objetivo principal sustituir a la aplicación actual, donde las peticiones que realiza el cliente generan que el servidor tenga que realizar a su vez, en cada interacción, la página web para enviársela al cliente.[12.1]

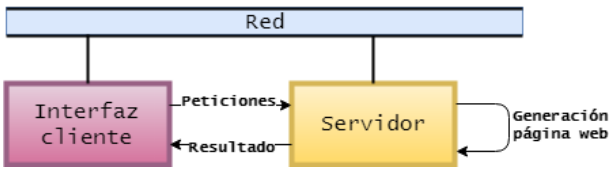

Figura 3.2: Aplicación con PHP

Los motivos por los cuales se desea este cambio de tecnología, se pueden resumir en los siguientes puntos:

- Aplicación del lado del cliente para reducir consumo en el servidor. Mejorando de esta forma el rendimiento.[12.1]

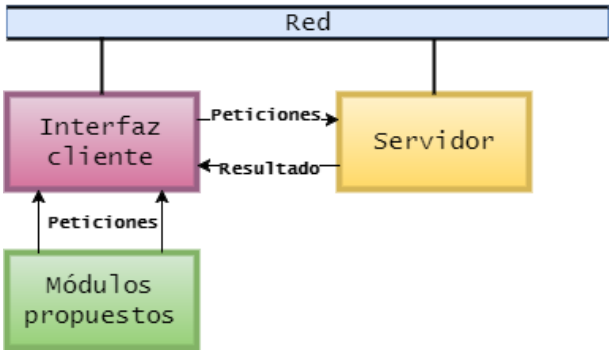

Figura 3.3: Aplicación con JavaScript

- Añadir funcionalidades:
  - Actualmente en el sistema no se permite la descarga de los archivos en un sistema local. A través de los nuevos módulos, el usuario podrá descargar una copia de estos.
  - Chequeo del archivo XML que se genera en el servidor, por si este tiene algún error. A su vez, en caso de que haya errores, mostrará la línea de estos.
  - Filtrado por fecha de los archivos en el visor.
  - La posibilidad de "comprimir" el editor XML, ya que si se trata de un archivo grande puede ser incómodo para el usuario su visualización.
- Actualmente el sistema se actualiza a través de un temporizador, que refrescará la información, creando sobrecarga en el servidor. Se cambiará dicha funcionalidad por una que permita al usuario actualizar la información a través de una petición pulsando un botón.
- Comprimir los menús, ya que en "reports" grandes, ocupan espacio de pantalla necesario para visualizar el contenido correctamente.

4. Angular JS

Los frameworks de JavaScript en general sirven para solucionar necesidades actuales en el área del desarrollo multiplataforma de aplicaciones grandes.

AngularJS permite crear aplicaciones SPA (*Single-Page Applications*) dentro de la familia de frameworks como BackboneJS [28] [Glosario 1] o EmberJS [29].

Anteriormente, solo se tenía jQuery como opción para aplicaciones Front-End (además de librerías menos conocidas como Mootools o Prototype). Se manipulaba el DOM [Glosario 1] de una forma sencilla, añadiendo efectos y llamadas AJAX [Glosario 1], pero no se seguía ningún patrón de diseño, convirtiendo aplicaciones más complejas en código difícil de manejar llegando a ser el "Spaguetti Code" [Glosario 1]

Angular JS es un framework Javascript que se puede añadir como un script adicional a cualquier página HTML. Utiliza los 'módulos' para definir una aplicación, y el concepto de controlador para gestionar dicha aplicación. Además, extiende los atributos HTML mediante directivas, y relaciona código HTML con modelos de datos mediante expresiones incrustadas.

A su vez, uno de los mayores problemas que soluciona AngularJS son los diferentes enfoques a la hora de diseñar la arquitectura de una aplicación, pudiendo implementar el modelo MVC (Modelo-Vista-Controlador) [11] separando la visualización de la lógica de negocio, o implementando MVVM [Glosario 4]] (Model+View-View+Model)

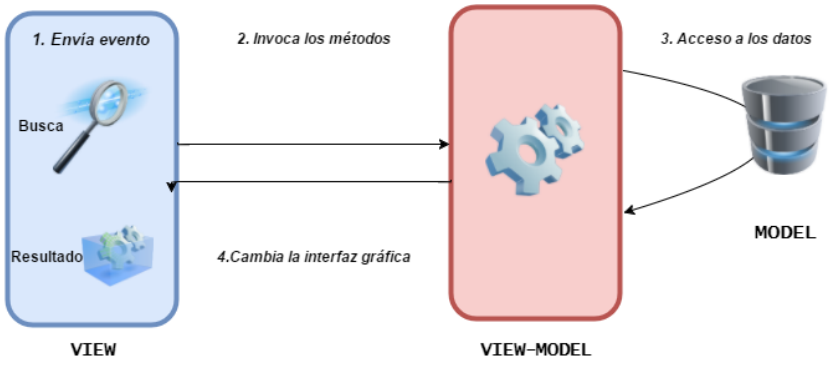

Figura 4.1: Modelo MVC

Por tanto, Angular JS es una solución completa que incluye prácticamente todo lo necesario para crear una aplicación cliente en JavaScript (generación de vistas, uso de databinding, las rutas, organización de componentes en módulos o comunicación con el servidor). Tratándose a su vez de un lenguaje que se está haciendo más conocido desde finales de 2012, y que

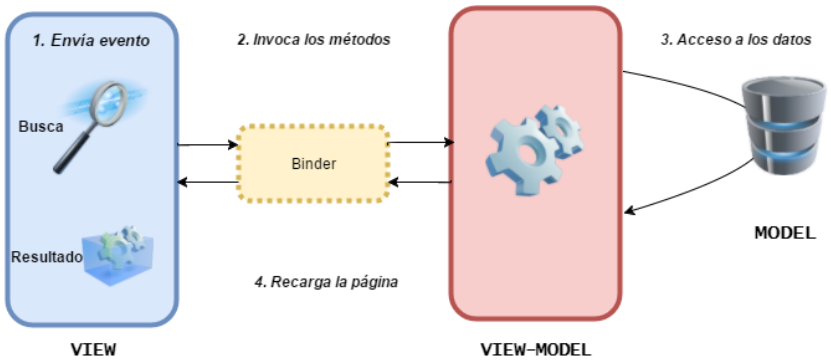

Figura 4.2: Modelo MVVM

se encuentra en pleno auge durante el último año debido a su nueva versión (AngularJS 2), siendo candidato para una nueva tecnología "stack" llamada MEAN (MongoDB/Mongoose +ExpressJS +AngularJS +NodeJS) como antes lo era LAMP (Linux +Apache +MySQL +PHP).

Además se trata de un framework mantenido por Google, lo que asegura su continuo desarrollo y evolución durante los próximos años

4.1. Arquitectura de Angular JS

La arquitectura de Angular JS está formada por diversos elementos entre los que se encuentran: módulos, directivas, rutas, vista, controlador, *scope*, y las **Factorías*.

Figura 4.3: Arquitectura de Angular JS

- *Module* (Módulos): Permite enlazar el valor de un campo de entrada HTML (como los botones o los selectores entre otros) con variables de una aplicación AngularJS. Este enlace va en ambos sentidos, es decir, modificaciones en el HTML provocarán cambios en variables de angular y viceversa. Los módulos se crean a través de un nombre y, de manera opcional, un array conteniendo nombres de otros módulos de los que dependa el nuestro (Angular JS Inyecta automáticamente los módulos al crear el nuestro), tal y como se muestra a continuación:

```
angular.module = ('app',['module1,'module2'] )
```

Se podrían ver los módulos como un paquete de Java, donde se agrupan funcionalidades de Angular JS. Se pueden agrupar de esta forma directivas [4.1] comunes en un módulo y luego crear las aplicaciones utilizando dicho módulo.

Es importante, a la hora de crear un módulo, tener en cuenta la sintaxis ya que no existen nombres de funciones distintos para crear o referenciar módulos, sino que en ambos casos se utiliza el método *module()* . Si se incluye un segundo parámetro con los corchetes, se indica que se está creando, y en caso de que se ponga un segundo parámetro, significaría que se hace referencia a un módulo ya existente, tal y como se muestra a continuación:

```
angular.module = ('module1',[ ])
```

En los módulos, se pueden definir diversos artefactos que permitirán configurar nuestra aplicación. Un artefacto es, por ejemplo, una directiva o un controlador, que se deben definir y asociar a módulos de AngularJS. Los elementos habituales dentro de un módulo serán:

- Controladores
- Servicios (cinco tipos)
- Bloques *config/run*
- Filtros
- Directivas
- Constantes
- Providers
- Factorías

- *Scope* : Es la pieza más importante del motor de AngularJS, ya que en el *Scope* se encuentran los datos que se tienen que manejar dentro de la parte de presentación. Se encarga de contener los datos, transportar y hacer visible la información necesaria para implementar la aplicación: desde el controlador a la vista y desde la vista al controlador.

  A la hora de declarar una directiva, se pueden asignar distintas propiedades: *restrict* para delimitar el ámbito de declaración de la directiva, *template* que especifica el código HTML que se compilará o *templateUrl* que apuntala dirección donde puede encontrarse un *template*, o *Scope*.

Todas las directivas tienen asociado un ámbito que por defecto es el de su padre, lo normal es que sea un *controller*. Este atributo se puede modificar con diferentes valores:

- False : Es el comportamiento por defecto, hereda el ámbito. Cualquier modificación tanto dentro como fuera de una propiedad del modelo se verá reflejada en ambos sentidos.
- True : Crea un nuevo ámbito haciendo una copia del campo del padre de modo que, al ser un nuevo ámbito, las modificaciones en el modelo que se realicen dentro de la directiva no se verán reflejadas hacia el exterior. Sin embargo, las modificaciones en el modelo del ámbito del padre sí tendrán reflejo en el ámbito de la directiva.
- { } : Se creará un ámbito nuevo independiente del padre. Este es el ámbito recomendado para qué realmente una directiva pueda ser reutilizada y no dependa ni herede todo el ámbito de su padre. Dentro de la declaración de ámbito se pueden pasar valores especificando una serie de prefijos que van a permitir no solo pasar propiedades del modelo del ámbito del padre, sino vincularlas en uno o ambos sentidos. Las propiedades pueden ser :
  - Vinculación en un solo sentido (@) de manera que al crearse el ámbito de la directiva el valor de esta propiedad se asigna de nuevo de forma que las modificaciones no afectan al ámbito del padre.
  - Vinculación en ambos sentidos (=) de forma que el valor de la propiedad sea una referencia al modelo, no una expresión evaluable como en el caso anterior.
  - Vinculación de métodos que permite invocar desde la directiva a métodos declarados en el ámbito del padre *&*

- *Controller* (Controladores): Los controladores se utilizan principalmente para dos funcionalidades :
  - Inicializar el estado del *Scope* : De esta forma la aplicación tendrá los datos necesarios para empezar a funcionar, pudiendo presentar la información correcta al usuario en la vista.
  - Escribir código que añada funcionalidades o comportamientos (métodos o funciones) al *Scope*

  Con el controlador, no se debe manipular el DOM [Glosario 1] de la página, ni formatear la entrada de datos.

Para crear un controlador, simplemente hay que crear una función de JavaScript con el nombre de dicho controlador, aunque esta no es la forma adecuada, se deberían crear de la siguiente forma:

```
var app=angular.module ('module1',[ ]);
  app.controller('controller1',function ($scope))
  $scope.mensaje='Hola Mundo!';
});
```

En el ejemplo, la línea 1 contiene la variable "app" con una referencia al módulo que acabamos de crear llamado "app". Esta variable contiene un método llamado "controller" que permite añadir controladores al módulo. Y la línea 3 define un controlador llamado "controller1" al cuál se añadirá el módulo "app" definido anteriormente. El método *controller* acepta dos parámetros, el primero el nombre del controlador y el segundo la función con el código JavaScript del controlador.

- *Directives* (Directivas): Las directivas de Angular JS sirven para encapsular código HTML de manera que las aplicaciones puedan tener elementos reutilizables, también conocidos como componentes. Se trata de marcas en los elementos del árbol DOM [Glosario 1] en los nodos del HTML que indican al compilador que debe asignar cierto comportamiento a dichos elementos o transformarlos según corresponda, pudiendo de esta forma añadir comportamiento dinámico al árbol DOM haciendo uso de las nativas o usando directivas propias permitiendo crear elementos visuales con el método *directive* del módulo, tal y como se puede observar en el siguiente ejemplo:

```
app.directive('user-info',function() {
return {
    restrict: 'E',
   template:'Nombre: {{user.name}}, email: <a href="mailto
      :{{user.email}}">{{user.email}}</a>'
});
```

Pudiéndose hacer uso de la misma a través de:

```
<div np-app=''module1' ng-controller='controller1'>
<user-info></user-info>
</div>
```

De hecho una de las recomendaciones de Angular JS es que se utilicen las directivas como único sitio para manipular el árbol DOM.

Como se ha dicho anteriormente, existen directivas nativas de Angular que permiten la vinculación del modelo a la vista, exponiéndolo a través de un controlador. Existen las suficientes directivas nativas para soportar cualquier aplicación SPA (*Single Page Application* teniendo incluso formularios de HTML5 que permiten asociar una propiedad del modelo al componente del formulario. Algunas directivas nativas que destacan son las siguientes [4]:

- ng-app (ngApp): Directiva de Angular JS que arranca la aplicación estableciendo el elemento raíz de la misma.
- ng-controller (ngController): Directiva nativa que asocia a una vista el controlador que mantiene la vinculación del modelo y gestiona los eventos de control de la misma.
- ng-model (ngModel) : Es la directiva que vincula una propiedad en concreto del modelo declarado en el controlador a un componente de la vista; normalmente a un formulario para obtener la información de este.
- btn.radio (btnRadio): Se trata de una directiva de angular-bootstrap que transforma una etiqueta en un componente de tipo *radio button* de Bootstrap [Glosario 1] Lo normal es que las directivas que empiecen por "ng*"sean de Angular.

A la hora de invocar directivas personalizadas, se puede hacer mediante diversos métodos, que se identifican a través de una letra:

- Mediante la declaración de un atributo (“A”) en cualquier elemento del DOM.

```
<div customixe-it></div>
```

- Mediante la declaración de un elemento en cualquier bloque de código comentado (“M”)

```
<!-- directive: customize-it -->
```

- Mediante la declaración de un elemento del propio árbol DOM, es decir, con el nombre de la directiva (“E”):

```
<customize-it></customize-it>
```

- Mediante una clase (“C”) :

```
<div class="nombre"></div>
```

- Mediante combinación, como por ejemplo un tipo “EA” o añadirle también un comentario siendo “EAC”.

Lo normal es disponer de declaraciones de directivas a partir del nivel de elemento y atributo.

- *Factory* (Factoría) : Se trata de contenedores de código, un tipo de servicio de Angular JS con los que se pueden implementar librerías de funciones o almacenar datos. Cuando se usa una factoría, devuelve un dato de cualquier tipo, normalmente un objeto JavaScript donde se pueden encontrar propiedades o métodos, teniendo como diferencia con los controladores que únicamente pueden ser instanciados una vez dentro de las aplicaciones, asegurándose así de no perder su estado, por lo que son una solución para la pérdida de datos de los controladores cuando cambiamos de vista. Ejemplo de factoría en Angular JS :

```
  .factory("descargasFactory", function(){
   var descargas = ["JavaScript", "jQuery", "AngularJS"];

   var interfaz = {
      nombre: "Victoria",
      getDescargas: function(){
         return descargasRealizadas;
      },
      nuevaDescarga: function(descarga){
         descargas.push(descarga);
      }
   }
   return interfaz;
})
```

Con esta función, se ha definido una factoría llamada "descargasFactory"devuelve un objeto, siendo a su vez la variable "descargas"privada a la factoría, ya que esta no se devuelve en la interfaz siendo por tanto una variable no accesible desde fuera de la factoría.

Para usar la factoría creada anteriormente, se puede ejecutar el siguiente código:

```
.controller("appCtrl", function(descargasFactory){
   var vm = this;

   vm.name = descargasFactory.name;
```

```
   vm.descargas_realizadas = descargasFactory.getDescargas();
   vm.funciones = {
      guardarNombre: function(){
         descargasFactory.name = vm.name;
      },
      agregarDescarga: function(){
         descargasFactory.NewDescarga(vm.nameNuevaDescarga);
         vm.msg = "Descarga nueva agregada";
      },
      borrarMensaje: function(){
         vm.msg = "";
      }
   }
});
```

En dicho código recibimos como parámetro la variable "descargasFactory"que contiene los datos y funciones. Para poder acceder a los datos de la factoría se deberá volcar ese dato en el *Scope* a través del siguiente código :

```
vm.nombre = descargasFactory.nombre;
vm.descargas = descargasFactory.getDescargas();
```

- Routing (Rutas) : Para navegar por diferentes páginas en una aplicación sin necesidad de recargar la página, se utiliza el módulo *ngRoute*.

  Para poder enlazar distintas páginas dentro de una aplicación, solo es necesario usar dicho módulo tal y como se muestra a continuación:

```
<body ng-app="myApp">

<p><a href="#/">Main</a></p>

<a href="#red">Red</a>
<a href="#green">Green</a>
<a href="#blue">Blue</a>

<div ng-view></div>

<script>
var app = angular.module("myApp", ["ngRoute"]);
app.config(function($routeProvider) {
```

```
   $routeProvider
   .when("/", {
      templateUrl : "main.html"
   })
   .when("/red", {
      templateUrl : "red.html"
   })
   .when("/green", {
      templateUrl : "green.html"
   })
   .when("/blue", {
      templateUrl : "blue.html"
   });
});
</script>
```

[Referencia [9]]

A través del módulo se crea la dependencia de ngRoute, usando el “routeProvider” se configuran diferentes rutas en la aplicación de manera que, en el ejemplo, se puede acceder a las páginas “green” ,“blue” ,“red” o “main” a través de un click en “Red”, “Green” o “Blue”. Para ello, es necesario que el código se encuentre dentro de un controlador con la directiva “ng-view”

5. Desarrollo de los módulos propuestos

Tal y como se nombra en el apartado 2.2 ,se tratará de realizar dos módulos: un visor de archivos y un editor de xml.

Dichos módulos sustituirán a los actuales (Imágenes: [2.2], [2.3]) tal y como se ha mencionado en [3].

Para ello, se ha dividido el código en diferentes partes para una correcta organización, quedando de la siguiente forma:

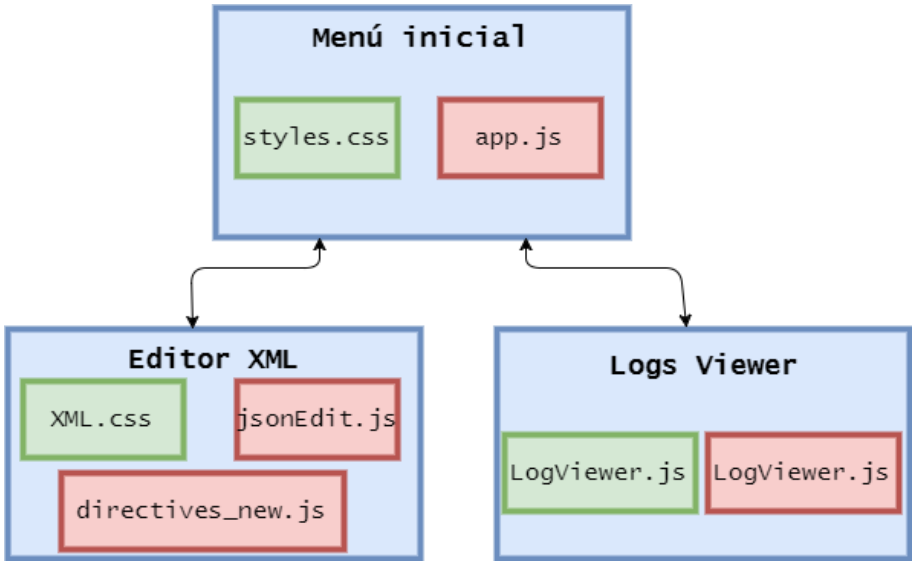

Figura 5.1: Distribución del código fuente de la aplicación

- El menú inicial estará compuesto por una hoja de estilos (*styles.css*) y un módulo de JavaScript (*app.js*).
- El editor XML tendrá su propia hoja de estilos (*XML.css*) y dos módulos JavaScript:
  - Archivo Javascript *jsonEdit.js*: Su función será validar el código XML.
  - Archivo JavaScript *directives_new.js*: Cuya función es realizar el resto de funcionalidades del XML entre las que destacan la visualización, el borrado, el editado, la descarga y la subida de archivos.
- El visor de archivos está compuesto por una hoja de estilos (*LogViewer.css*) y un archivo JavaScript (*LogViewer.js*).

El menú será el encargado de hacer las llamadas necesarias a los métodos correspondientes. De esta forma, el diseño será modular teniendo funcionalidades escalables y reutilizables además de autónomas.

5.1. Arquitectura de red

Para el desarrollo del trabajo, es necesario que los módulos adquieran los datos de un servidor [12] para después representarlos.

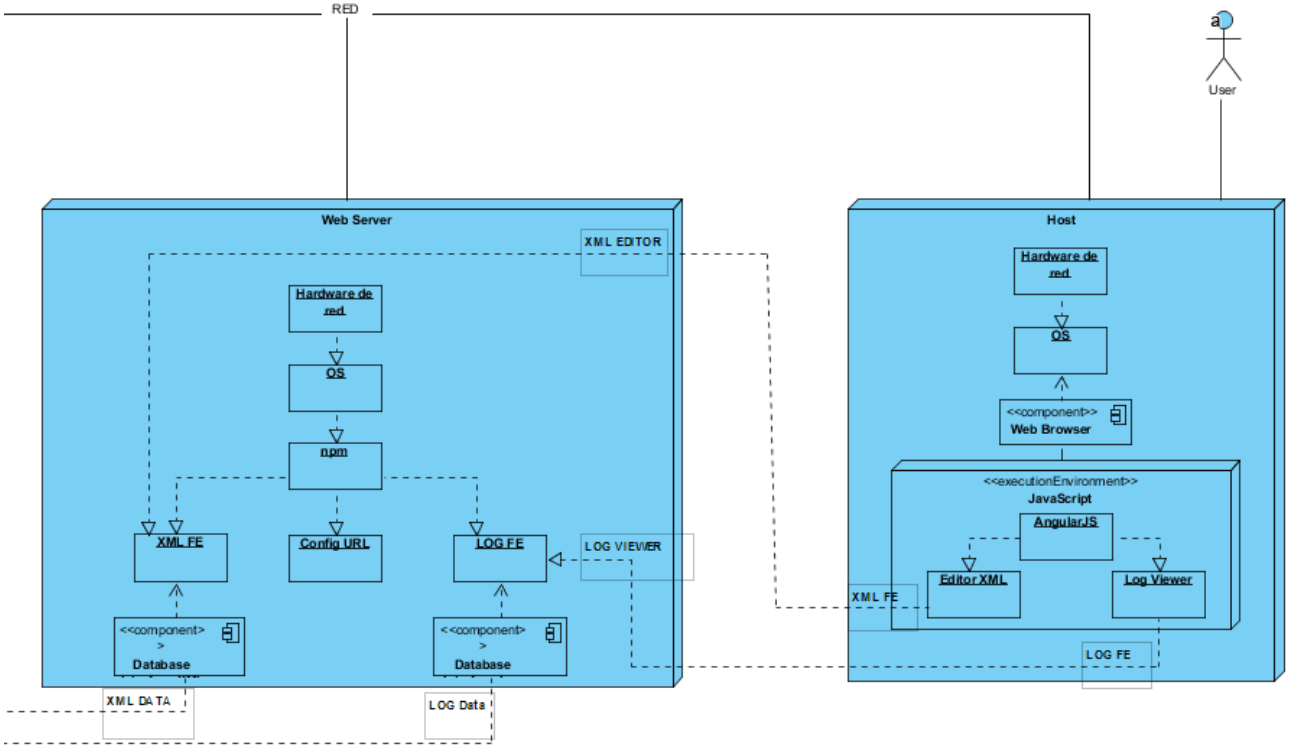

Figura 5.2: Arquitectura de red del sistema II

La arquitectura de red del sistema [5.3] está compuesta por los siguientes elementos:

- Host: el host representa un cliente que desea conectarse al servidor web. Está compuesto por las siguientes instancias:
  - Hardware de red.
  - Sistema Operativo (OS).
  - Web Browser: será el componente activo del sistema operativo, en él, el usuario realizará las acciones.

- JavaScript: será el entorno de ejecución donde se tendrá:
  - La aplicación de Angular JS que ejecutará la aplicación web
  - Los módulos de edición XML y visor de archivos

- Web Server: representa el servidor. Éste tendrá los recursos necesarios para generar la interfaz Web utilizando las bases de datos que se encuentran en el *Node Server*. Está compuesto de las siguientes instancias:
  - Hardware de red
  - Sistema Operativo (OS)
  - Npm: gestor de paquetes de node.js [2.3.3] que permite descargar librerías y enlazarlas, o, en su defecto, descargar programas de JavaScript para su ejecución.
    - XML FE : *Front-end* del editor XML
      - Database Interface XML
    - Configuración de la URL
    - LOG FE:*Front-end* del visor de archivos
      - Database Interface Archives

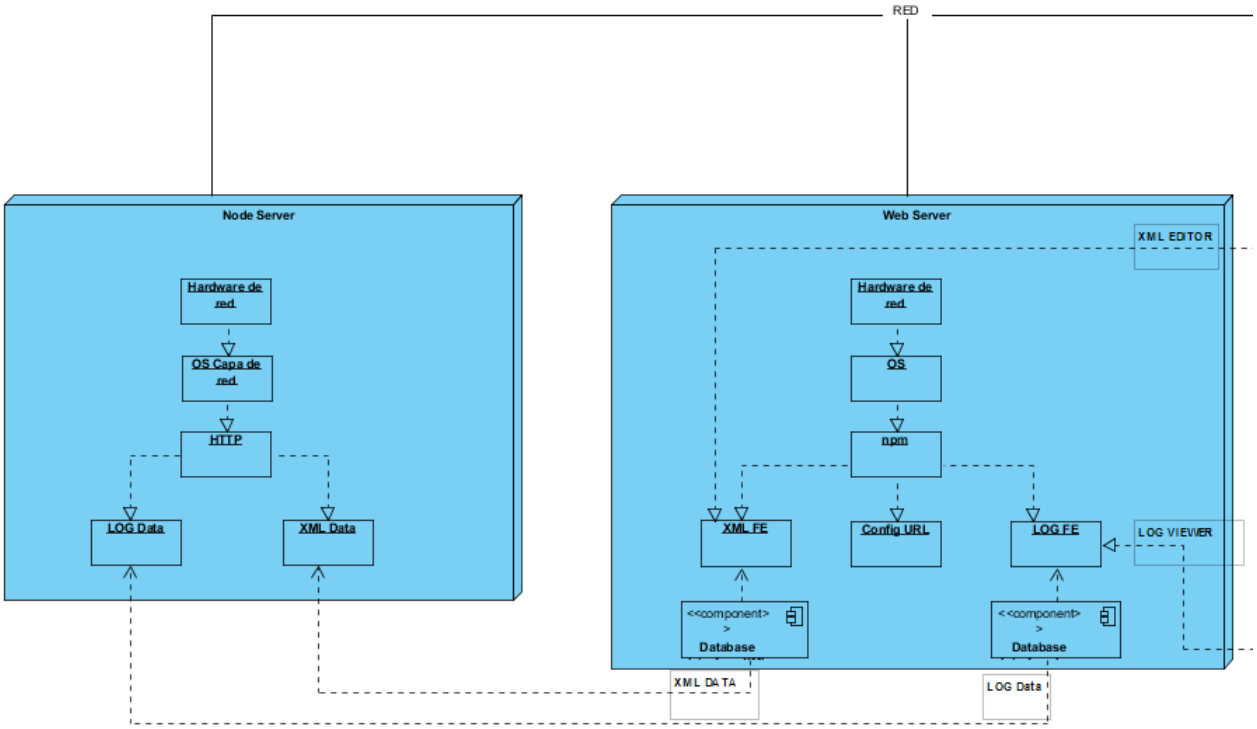

Figura 5.3: Arquitectura de red del sistema I

- Node Server: el *Node Server* representa el servidor donde se encuentran los archivos que se quieren representar en la *Interface* web (explicado en el trabajo [12]). Está compuesto por las siguientes instancias:
  - Hardware de red
  - Sistema operativo (OS)
  - HTTP
    - Log Data: archivo de datos *Log*
    - XML Data: archivo de datos del XML

5.2. Visor de archivos

El visor de archivos, tiene como objetivo representar un archivo de *log* de un sistema con el fin de hacer más fácil su compresión.

Un ejemplo de archivo *log* es el siguiente :

```
[Sep 14 15:19:07 2016] [notice] [Dispatcher-Main] Unix socket
   created
[Sep 14 15:19:08 2016] [notice] [Dispatcher-Main] Listening for
[Sep 15 14:49:10 2016] [warn] [StatusManager] Number of subscribers
    too high. It is possible that processes are not being executed
[Sep 15 14:52:13 2016] [notice] [StatusManager] Job ccc3f272-7b42
   -11e6-a770-000c2940a46a has subscribed.
[Sep 15 14:52:19 2016] [warn] [StatusManager] Number of subscribers
    too high. It is possible that processes are not being executed
```

Donde se puede observar que el primer campo es la fecha del evento (mes:día:hora:año), el segundo campo es el tipo de alerta, el tercero la entidad que ha notificado el evento y por último el mensaje.

Los tipos de alerta pueden ser los siguientes:

- *notice* para las notificaciones del sistema. Son mensajes de información del estado
- *error* para los errores del sistema. Informan de que uno de los procesos ha fallado y el por qué.

- *warn* para los avisos. Informan de un posible error que no ha hecho fallar al sistema, tienen menor prioridad que los errores ya que se ha podido ejecutar la orden en el servicio.

5.2.1. Adaptación del formato

Uno de los problemas a la hora de representar un archivo del servidor, es que este no viene en el formato que se desea para la implementación de un visor en el cual se puedan aplicar metodologías de filtrado o visualización eficiente.

La primera transformación que se debe realizar del formato, es pasarlo a un archivo del tipo JSON, ya que este formato permitirá realizar acciones más sencillas y eficaces. Para ello se hace uso de la siguiente función:

```
function CSV2JSON(csv) { //Funcion que se encarga de transformar
   el archivo en un JSON
        var array = CSVToArray(csv); //Llama a una funcion
           encargada de realizar una conversion del formato a una
           matriz
        var objArray = [];
        for (var i = 1; i < array.length; i++) { //Para cada linea
            del documento...
           objArray[i - 1] = {};
           for (var k = 0; k < array[0].length && k < array[i].
              length; k++) { //coge los campos separados por una
              tabulacion del CSV, asignandolos a un array. Dicho
              array tiene la segunda componente como el nombre del
               campo, asociado cada linea al nombre
              var key = array[0][k];
              objArray[i - 1][key] = array[i][k]
           }
        }

        var json = JSON.stringify(objArray); //Convierte el array
           en un archivo del tipo JSON
        var str = json.replace(/},/g, "},\r\n"); //Remplaza,por
           temas de formato,algunos elementos
        return str;
     };
```

Dicha función se encarga de llamar, en primera instancia, a la función que realizará los cambios en el archivo, denominada *CSVToArray*:

```
function CSVToArray(strData, strDelimiter) {
         //Chequea si el delimitador esta definido, en caso de no
            ser asi, sera por defecto una coma
         strDelimiter = (strDelimiter || "\t");
         //Parsea los valores de CSV en los deseados para su
            tratamiento.
         var objPattern = new RegExp((
         // Delimitadores
         "(\\" + strDelimiter + "|\\r?\\n|\\r|^)" +
         // Quoted fields.
         "(?:\"([^\"]*(?:\"\"[^\"]*)*)\"|" +
         // Standard fields.
         "([^\"\\" + strDelimiter + "\\r\\n]*))"), "gi");
         // Crear una matriz para almacenar nuestros datos. Da a la
             matriz una primera fila vacia por defecto.
         var arrData = [[]];
         // Crear una matriz para mantener nuestro patron
            individual
         var arrMatches = null;
        //Bucle sobre los coincidencias de expresion regular
         // hasta que no se pueda encontrar una coincidencia.
         while (arrMatches = objPattern.exec(strData)) {
            // Coger el limitador encontrado
            var strMatchedDelimiter = arrMatches[1];
         // Se comprueba si el delimitador dado tiene una longitud
            // (sino es es el comienzo de la cadena) y si coincide
            // con el delimitador de campo establecido.Si no lo
               hace, entonces sabemos
            // que este delimitador es un delimitador de filas.
            if (strMatchedDelimiter.length && (strMatchedDelimiter
               != strDelimiter)) {
           // Puesto que hemos alcanzado una nueva fila de datos,
               // agrega una fila vacia a nuestra matriz de datos.
               arrData.push([]);
            }
         // Ahora que tenemos nuestro delimitador fuera,
            // vamos a comprobar para ver que tipo de valor que
```

```
            // capturamos
            if (arrMatches[2]) {
           // Encontramos un valor. Cuando capturamos
               // este valor, buscamos las comillas dobles que
                  indican que ya se ha recorrido el valor
               var strMatchedValue = arrMatches[2].replace(
               new RegExp("\"\"", "g"), "\"");
            } else {
           // Encontramos un valor no esperado
               var strMatchedValue = arrMatches[3];
            }

        // Ahora que se tiene la cadena de valores en el formato
           deseado, se agrega a una matriz de datos.
           arrData[arrData.length - 1].push(strMatchedValue);
        }
        // Devuelves el valor final
        return (arrData);
    };
```

Esta función, como se puede observar en los comentarios, realiza los siguientes pasos:

- Chequea si el limitador esta definido, en caso de no estarlo pondrá por defecto una coma.
- Sustituye los elementos del CSV a través de una serie de transformaciones para obtener el formato deseado(tal y como se vio en [5.2]). Para ello comprueba la longitud de los elementos a través de los parámetros que lo encierran (en este caso comillas)
- Devuelve el archivo con el formato deseado una vez se ha añadido a una matriz que recoge todo el documento ya modificado.

El formato final, será el siguiente:

Figura 5.4: Representación final del archivo

Donde el archivo se representará en cuatro columnas diferentes:

- *Date* : La fecha en la cual se generó el archivo, vendrá en formato: Mes[3 Dígitos] Día[2 Dígitos] Hora[2 Dígitos]:Minutos[2 Dígitos]:Segundos[2 Dígitos] Año[4 Dígitos]
- *Type*: define el tipo del mensaje que se genera, pudiendo ser:
  - *notice*: notificación de un servicio del servidor.
  - *error*: notificación de un error en un servicio del servidor.
  - *warm*: notificación de un aviso de un servicio del servidor, la diferencia entre un error y un aviso es que uno puede causar problemas en un archivo mientras que el otro puede ser un aviso del sistema ante un cambio.
- *HostProcess*: indica cuál es el servicio que genera el mensaje.
- *Message*: el contenido del mensaje, la información a representar al usuario.

5.2.2. Filtros

En el archivo a visualizar se realizan una serie de filtros para diferenciar el contenido.

Uno de los filtros aplicados es la clasificación de los elementos en colores dependiendo de su importancia, pudiendo ser: verde, amarillo y rojo. El rojo ha sido asignado a los mensajes de “error”, ya que por convención se relaciona con peligro, el amarillo serán aquellos avisos del sistema (“warnings”), y el verde serán aquellos mensajes informativos, siguiendo así los colores de un semáforo.

Figura 5.5: Filtros de colores aplicados

Para ello, se han usado cuatro directivas de Angular JS para tablas:

- ng-table: permite el uso dinámico de tablas, pudiéndoles aplicar filtros automáticos, paginado, estilos y cabeceras entre otras características.

```
<table ng-table="usersTable" show-filter="true" class="table
    table-striped "> //Se usara el estilo striped cuando se
    muestre la tabla
```

Uno de sus usos serán los distinto filtrados, entre ellos estará el filtrado por página, que mostrará los elementos dependiendo del paginado que elija el usuario, pudiendo representar 10, 25, 50 ó 100 elementos. Se inicializa por defecto a mostrar 50 elementos en la primera página.

```
$scope.usersTable = new NgTableParams({ //Elementos a
   visualizar,inicializamos en la primera pagina mostrando 50.
   page: 1,
   count: 50,
   sorting: {
      datetime: 'desc'
   },
```

- ng-repeat: Permite repetir una serie de elementos del HTML en función de un array. Su mayor utilidad es a la hora de realizar tablas o estructuras que se repiten. Un ejemplo de uso puede ser el siguiente, donde representará la cadena de números que contiene "n":

```
<div ng-repeat="n en [42, 42, 43, 43] representar (n)">
  {{n}}
</div>
```

En este caso se usará para recorrer los datos del archivo.

```
<tr data-ng-repeat="user in data | dateRangeFilter:dt1:dt2"
   class="animate"> //El dt1 y dt2 es para el filtrado de los
   calendarios, explicado a continuacion
```

- ng-if: permite que exista o no un elemento en la página, la diferencia con las directivas *ng-show* o *ng-hide* es que los elimina del DOM. Combinada con *ng-class* seleccionará el color que le corresponde dependiendo del tipo.

```
<td data-title ="'Date'" ng-if="true" sortable="'Date'" filter
   ="{ 'Date': 'text' }" ng-class="{'notice-class': user.Type
   === 'notice' , 'error-class': user.Type=='error', 'warn-
   class': user.Type=='warn'}" >
{{user.Date}}
</td> //Clasificara los elementos del tipo Date, y dependiendo
    de su tipo en "Type" los pondra del color correspondiente
   (rojo,verde o amarillo)
```

```
<td data-title="'Type'" sortable="'Type'" filter="{ 'Type': '
   select' }" filter-data="Tipo" ng-class="{'notice-class':
   user.Type === 'notice' , 'error-class': user.Type=='error',
    'warn-class': user.Type=='warn'}" >
{{user.Type}}
</td> //Clasificara los elementos del tipo Type, y dependiendo
    de su tipo los pondra del color correspondiente (rojo,
   verde o amarillo)

<td data-title="'HostProcess'" filter="{ 'HostProcess': 'text'
    }" class="default-color" ng-class="{'notice-class': user.
   Type === 'notice' , 'error-class': user.Type=='error', '
   warn-class': user.Type=='warn'}">
{{user.HostProcess}}
</td> //Clasificara los elementos del tipo HostProcess, y
   dependiendo de su tipo en "Type" los pondra del color
   correspondiente (rojo,verde o amarillo)

<td data-title="'Message'" filter="{ 'Message': 'text' }" ng-
   class="{'notice2-class': user.Type === 'notice' , 'error2-
   class': user.Type=='error', 'warn2-class': user.Type=='warn
   '}">
{{user.Message}}
 </td> //Clasificara los elementos del tipo Message, y
    dependiendo de su tipo en "Type" los pondra del color
    correspondiente (rojo,verde o amarillo)
```

Se han realizado también una serie de filtros dentro de los campos de selección de los elementos, para ello se ha utilizado el parámetro “filter” en el HTML (como se puede ver en el código anterior) y la siguiente línea en el JavaScript:

```
$scope.data = params.filter() ? $filter('filter')($scope.data,
   params.filter()) : $scope.data;
```

Existiendo así una vinculación que permitirá al usuario filtrar los campos que desee. Siendo posible filtrar a través de una lista (en el caso de tipos de mensajes) o por cadena de texto, además de poder ordenar las fechas de mayor a menos o viceversa si se desea. [5.6,5.7]
También se ha creado un calendario capaz de filtrar, dependiendo de la fecha seleccionada

Figura 5.6: Ejemplo de filtrado en una lista

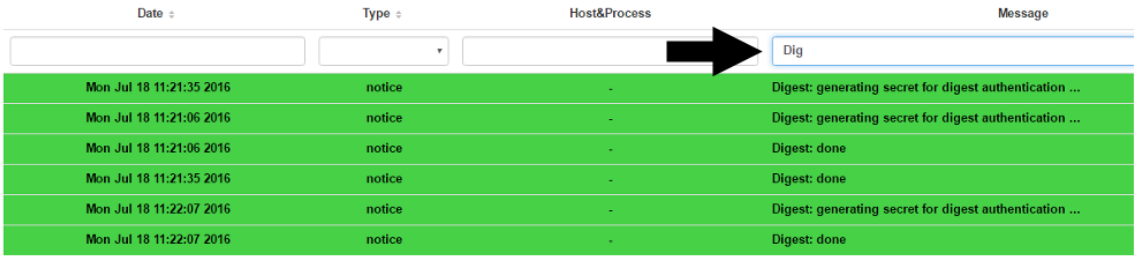

Figura 5.7: Ejemplo de filtrado por campo de texto

los mensajes. Se ha tenido en cuenta como requisito de la empresa, que el segundo campo no tenga que ser necesariamente rellenado.

Para dicho filtrado, se han tenido que hacer una serie de operaciones capaces de filtrar los datos dependiendo de la fecha mínima elegida y la máxima.

```
$scope.today(); //Selecciona como fecha recomendada el dia actual
$scope.toggleMin = function() { //Escoge el valor minimo
    selecionado por el usuario
  $scope.minDate = $scope.minDate ? null : new Date();
};
$scope.toggleMin();
$scope.maxDate = new Date(2020, 5, 22); //Escoge por defecto el
    anio 2020

$scope.open1 = function($event) { //En caso de elegir algo en el
    primer cuadrado de filtrado (el primer calendario) cogera el
    nuevo valor
    $scope.status1.opened = true;
};

$scope.open2 = function($event) { //En caso de elegir algo en el
    segundo cuadrado de filtrado (segundo calendario) cogera el
    nuevo valor
```

```
    $scope.status2.opened = true;
};

$scope.setDate = function(year, month, day) { //Aplica la nueva
    fecha en ambos campos seleccionados
    $scope.dt1 = new Date(year, month, day);
    $scope.dt2 = new Date(year, month, day);
};

var formatDate = function(dateString) { //Parsea al formato
    correcto en las fechas,para que concuerden con las que poseen
    los filtros
    var monthMap = {"Jan":"01","Feb":"02","Mar":"03","Apr":"04","May
        ":"05","Jun":"06","Jul":"07","Aug":"08","Sep":"09","Oct":"10"
        ,"Nov":"11","Dec":"12"};
    var fields = dateString.split(' ');
    var completeDay = [fields[3], monthMap[fields[0]], fields[1]];
    var dayandTime = [completeDay.join('-'),fields[2]];
    return dayandTime.join(' ');
};
```

Finalmente, se filtra los elementos seleccionados en el calendario entre los elementos a representar en la tabla a través de esta función:

```
{
    total: $scope.users.length, //Funcion que filtra los elementos
        por los campos seleccionados del calendario
    getData: function (params) {
        for (var l=0; l<$scope.users.length;l++){
                //console.log(test["1"].Date)
            $scope.users[l].datetime=(getDateFromFormat($scope.users[l
                ].Date,"MMM d hh:mm:ss yyyy"));
        }
```

5.2.3. Actualizar contenido

Se ha implementado un sistema para poder cargar el formato del archivo cuando se desee a través de la pulsación en un botón, siendo útil para actualizar su contenido.

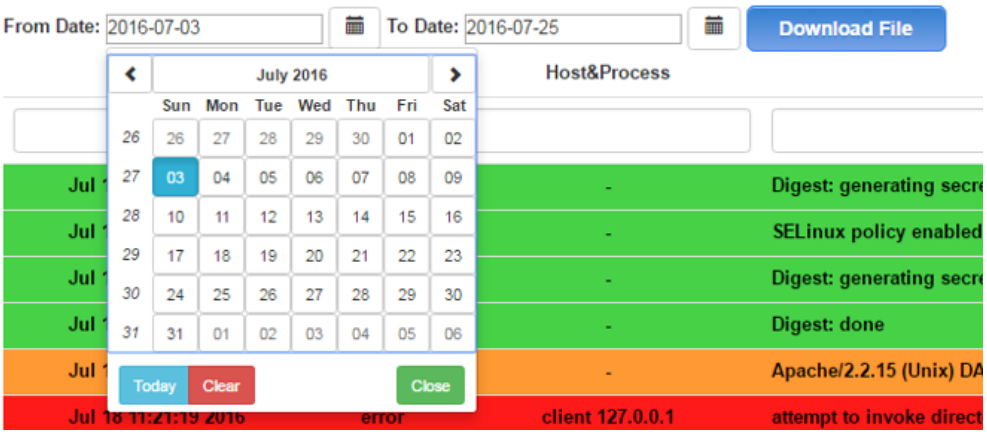

Figura 5.8: Ejemplo del calendario

En la versión del sistema en PHP, la actualización se hacía a través de un temporizador que cada vez que pasaba el tiempo establecido refrescaba la pantalla del usuario, teniendo las siguientes desventajas:

- Sobrecarga del servidor: al recibir una petición cada cierto tiempo, el servidor deberá atenderla. De esta forma, tendrá que enviar la página generada (al ser PHP) y los datos cada intervalo de tiempo que se cumpla, consumiendo memoria y ancho de banda de la comunicación, que dependiendo del tamaño del archivo será más o menos importante.
- Si el usuario está visualizando un dato en concreto, podrá perder la vista de él, ya que podrá ser desplazado hacia abajo u otras páginas debido a que el nuevo contenido se pondrá al principio, aunque puede ser solucionado accionando el pulsador de parada de peticiones.
- Necesidad de parar las peticiones de los archivos, teniendo que dejar al servidor en espera si estaba en medio de una actualización, perdiendo datos que ya haya podido enviar con anterioridad.

Por dichas razones, se ha decidido cambiar el sistema a un método de actualizado de archivos cuando el usuario quiera. Para ello, se ha implementado una función denominada *loadData* cuyo funcionamiento es sencillo, ya que invocará a una llamada REST del tipo GET para su actualización.

La función *loadData* será accionada al realizar una pulsación al botón, realizando tras la llamada GET las transformaciones ya descritas en el archivo para su visualización

```
$scope.loadData=function(){
```

```
$http({ //Configuramos la peticion HTTP

   url:"http://172.23.33.126/data/dispatcher_log"
   method:'GET',
   cache:false,
   headers: { 'Cache-Control' : 'no-store,no-cache,must-
      revalidate,max-age=0','Pragma': 'no-cache'} ,
   transformResponse: function (cnv) {
   //Codigo anterior para el archivo error_log:
      var newString= cnv.replace(/] /g,"\t");
      var final= newString.replace(/\[/g,"");

      var lines= final.split('\n');

      for (var i=0; i<lines.length;i++){
```

5.2.4. Gráfico

Se ha realizado un pequeño gráfico a partir de la librería D3.js de JavaScript. Esta librería produce a través de datos infogramas dinámicos e interactivos en aplicaciones web, destacando porque permite tener control completo sobre el resultado visual final. [15]

Se basa en funciones predefinidas de JavaScript según las cuales se pueden seleccionar elementos, crear objetos SVG, darles estilo, añadir transiciones, efectos dinámicos o agregar información.

En este trabajo se ha usado como base un trabajo de Michael Bostock publicado en su web [16] para la representación de elementos en formato de gráfico de barras.

La función del gráfico es representar en cada página el número de eventos de cada tipo que hay, siendo útil a la hora de visualizar páginas con varios elementos, pudiendo así hacerse una idea de la media de errores.

Para ello, se ha utilizado una función denominada *InitChart* .

```
function InitChart(result) {
   //Datos:
   var error=0;
   var warn=0;
   var notice=0;
```

```
for (var l=0; l<result.length;l++) //Bucle que recorre los datos
{
   if (result[l].Type=="warn"){ //Tipo warn
       warn=warn+1;
   }
   if (result[l].Type=="notice"){ //Tipo notice
      notice=notice+1;
   }
   if (result[l].Type=="error"){ //Tipo error
      error=error+1;
   }
}
var Types= ['notice','error','warn']; //Titulos
var Prueba=[notice,error,warn]; //Numero de casos

d3.selectAll("svg>*").remove(); //Comenzamos el grafico D3
   var barData = [{ //Primera barra de la grafica
     'x': Types[0],
     'y': Prueba[0]
   }, {
     'x': Types[1], //Segunda barra de la grafica
     'y': Prueba[1]
     }, {
     'x': Types[2], //Tercera barra de la grafica
     'y': Prueba[2]
   }];

   //Grafico D3
   var vis = d3.select('#visualisation'), //espacio para la
      visualizacion y sus caracteristicas
    WIDTH = 300,
    HEIGHT = 200,
    MARGINS = {
      top: 20,
      right: 20,
      bottom: 20,
      left: 50
    },
   xRange = d3.scale.ordinal().rangeRoundBands([MARGINS.left,
      WIDTH - MARGINS.right], 0.1).domain(barData.map(function (
```

```
         d) { //Eje x
         return d.x;
      })),

      yRange = d3.scale.linear().range([HEIGHT - MARGINS.top,
         MARGINS.bottom]).domain([0,d3.max(barData, function (d) {
         //Eje y
         return d.y;
      })
      ]),
//Realizamos el scale
      xAxis = d3.svg.axis()
         .scale(xRange)
         .tickSize(5)
         .tickSubdivide(true),
      yAxis = d3.svg.axis()
         .scale(yRange)
         .tickSize(5)
         .orient("left")
         .tickSubdivide(true);

      vis.append('svg:g') //Generamos los ejes x
         .attr('class', 'x axis')
         .attr('transform', 'translate(0,' + (HEIGHT - MARGINS.
            bottom) + ')')
         .call(xAxis);

      vis.append('svg:g') //Generamos los ejes y
         .attr('class', 'y axis')
         .attr('transform', 'translate(' + (MARGINS.left) + ',0)')
         .call(yAxis);

      vis.selectAll('rect') //Calculamos el rectangulo
         .data(barData)
         .enter()
         .append('rect')
         .attr('x', function (d) {
      return xRange(d.x);
      })
         .attr('y', function (d) {
```

```
    return yRange(d.y);
    })
       .attr('width', xRange.rangeBand()) //Calculamos el rango y
           lo devolvemos
       .attr('height', function (d) {
    return ((HEIGHT - MARGINS.bottom) - yRange(d.y));
    })
       .attr('fill', 'grey') //Rellenamos el color
       .on('mouseover',function(d){ //Cuando el raton pase por
          encima
          d3.select(this)
          .attr('fill','darkblue'); //Lo ponemos azul oscuro
          tip.show(d);
        })
       .on('mouseout',function(d){ //Cuando el raton se retire
          tip.hide(d);
          d3.select(this)
          .attr('fill','grey'); //Lo ponemos gris
       });
}
```

En dicha función, lo primero que se realiza es calcular, teniendo el resultado de los datos, cuántos son de un tipo y cuales de otro dependiendo de su *Type*. Esta acción se realizará a través de un bucle.

Una vez se tiene el número de casos de cada tipo, se inicia la unión de los datos a través de la selección de ellos (a través de la función *selectAll*) , definiendo los ejes y los efectos que se quieren añadir a ellos. Un ejemplo sería el uso del ratón al pasar por la representación (que en este caso mostrará el valor), además del cambio de color que se produce.

Otra herramienta orientadas a la representación podría ser *Nvd3* [17], siendo esta una librería que ofrece componentes reutilizables construidos sobre *D3*. Pero se ha decidido usar D3 ya que se puede adaptar más fácilmente a la implementación que se quiere realizar, ya que esta funcionalidad no será compleja de implementar.

Otras herramientas orientada a la visualización de datos que se han tenido en cuenta son:

- *Charts.js* [18]: muestra gráficos de forma sencilla, siendo una herramienta de software libre. Se ha descartado porque es menos completo que D3.

- *Zoomdata* [19]: software que permite el analisis y la visualización de datos, ofreciendo servicios para los móviles. Se ha descartado debido a que es de pago

5.3. Editor XML

El editor XML tiene como objetivo principal representar un archivo para que sea más sencilla su visualización.

Para ello, ha sido necesario realizar una conversión del XML a un formato aceptado por Angular JS, además de realizar el código correspondiente para sus funcionalidades.

5.3.1. Conversión XML a JSON

Los archivos del servidor, se reciben en formato XML, por lo que si se quieren usar directivas de Angular JS será necesario pasar el XML a formato JSON, debido a que Angular JS es un *framework* de JavaScript para el desarrollo de aplicaciones, heredando de JavaScript el uso nativo de JSON.

Para ello existen dos posibilidades:

- Pasar el archivo de XML a JSON en el servidor.
- Convertir el XML a JSON con una librería en el cliente, realizando la misma operación para reenviar el archivo al servidor.

En este trabajo se ha decidido realizar la segunda opción ya que es más sencillo realizar un cambio en el cliente, que generar archivos con otro formato diferente (y menos común) en el servidor. Para ello se va a utilizar el paquete *x2js* [13] instalado vía npm [11], que permitirá pasar del formato XML a JSON y viceversa de manera sencilla.

Un ejemplo de uso es el siguiente:

- XML a JSON: se crea una instancia x2js con la configuración por defecto aplicando a continuación la función correspondiente:

```
var x2js = new X2JS();
var xmlText = "<MyRoot><test>Success</test><test2><item>val1</
   item><item>val2</item></test2></MyRoot>";
```

```
var jsonObj = x2js.xml_str2json( xmlText );
```

En el código será:

```
transformResponse: function (cnv) {
            var x2js = new X2JS();
            var aftCnv = x2js.xml_str2json(cnv);
            return aftCnv;
}
```

Donde *cnv* es el archivo xml a transformar

- JSON a XML: primero se crea una instancia con los parámetros por defecto, a continuación se aplica la función correspondiente.

  Un ejemplo de uso es:

```
var x2js = new X2JS();
var jsonObj = {
    MyRoot : {
            test: 'success',
            test2 : {
               item : [ 'val1', 'val2' ]
            }
     }
};
var xmlAsStr = x2js.json2xml_str( jsonObj );
}
```

  En el código vendrá implementado de la siguiente forma:

```
$scope.jsonData = JSON.parse(json);
         $scope.wellFormed = true;
         xmlAsStr = x2js2.json2xml_str($scope.jsonData);
}
```

  Donde *json* corresponde al archivo JSON a pasar a XML, será necesario usarlo mediante *$scope* por si se edita el archivo y se producen actualizaciones en este.

5.3.2. Representación del archivo

Para representar el archivo, es necesario pasar primero el archivo a formato JSON tal y como se explicó anteriormente [5.3.1]. Una vez se tiene en dicho formato, se clasificará la información de este en diferentes tipos:

- String: Cadena de texto.
- Array: Lista de elementos, puede ser de cualquier tipo.
- Boolean: Para elementos de tipo booleano (verdadero o falso).
- Number: Para elementos de tipo numérico.
- Attribute: Elementos dentro de la etiqueta XML que no son valores. En el siguiente ejemplo *name* es un atributo de la etiqueta *text*:

```
<ogc:text name='Ejemplo'> </ogc:text>
```

- Text: Es como un *String* pero permite longitudes mayores y separación por párrafos.
- Prefix: Son los *namespaces* de XML. En el ejemplo anterior *ogc* sería un *prefix*. Se utilizan para tener etiquetas del mismo nombre diferenciadas para evitar conflicto.
- Object: Tipo genérico. Corresponde a todos los objetos, sirve para aquellos que no se han clasificado anteriormente

Este proceso, se realiza a través de una directiva de Angular JS [4.1], donde se lleva un proceso para seleccionar el tipo de elemento que es a través de una función denominada *getType*, además de las funciones de borrado, añadido o editado de elementos [5.3.3].

```
var getType = function(obj) { //Funcion que ayuda a definir la
   clase del elemento
        var type = Object.prototype.toString.call(obj);
        if (type === "[object Object]") {
           return "Object";
        } else if(type === "[object Array]"){
           return "Array";
        } else if(type === "[object Boolean]"){
           return "Boolean";
        } else if(type === "[object Number]"){
           return "Number";
```

```
        } else {
            return "Literal";
        }
    };
```

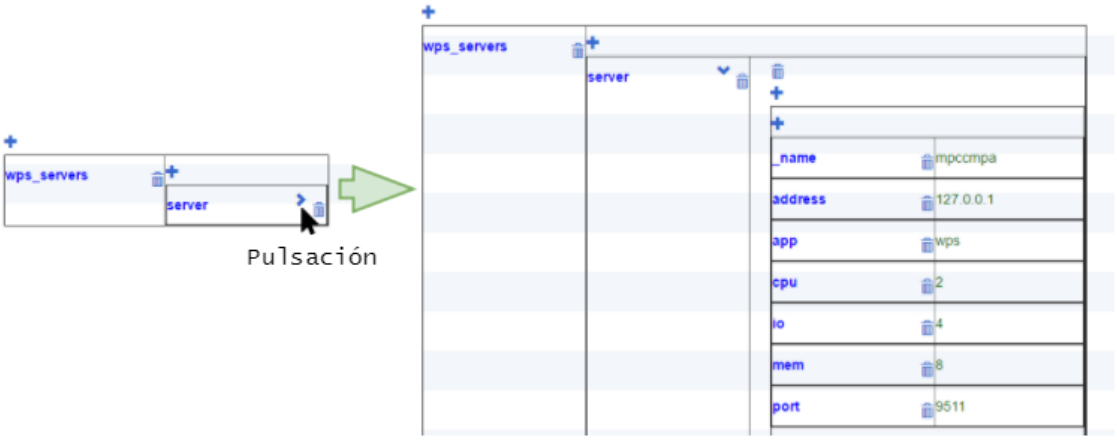

Figura 5.9: Ejemplo de objeto comprimido

Inicialmente, el documento XML se trata como un elemento *Object* o *Array* (en caso de que sea un conjunto de elementos sin agrupar), decidiendo si se comprime o no su visualización.[Figura 5.9] Dentro de dicho elemento se va a iterar en busca de elementos que podrán ser cualesquiera de los tipos citados anteriormente. Dicha iteración se realiza a través de la función *switchTemplate*.

```
    // start template : Comienza la representacion de los
       elementos, comenzara con un tipo objeto. Dentro de ella se
        podra
    //diferenciar como Object o Array, haciendo a su vez llamadas
        a la funcion recursiva que se encarga de aniadir los
       elementos
    if (scope.type == "object"){
       var template ='<div>'
          // funcion recursiva (se repite)
         + '<table border="1" ng-repeat="(key, val) in child " >
            '
          + '<tbody ng-show="(key != \''+"__prefix"+'\')">'
             // object llave
             + '<td valign= top >'
                + '<input id="scroll" class="keyinput" type="text
                   " ng-model="newkey" ng-init="newkey=key" '
                   + 'ng-blur="moveKey(child, key, newkey)"/>'
```

```
                    + '<i class="deleteKeyBtn glyphicon
                       glyphicon-trash" ng-click="deleteKey(
                       child, key)"></i>'

            + '</td>'
            // object value
            + '<td valign=top>' + switchTemplate + '</td>'
         // fin recursiva
         +'</tbody>'
         +addItemTemplate //Add button and function
         +'</table>'

      + '</div>';
   } else if (scope.type == "array") {
      var template ='<i ng-click="toggleCollapse()" class="
         glyphicon" ng-class="chevron"></i>'+
         '<span class="jsonItemDesc">'+'</span>'+
         '<div class="jsonContents" ng-hide="collapsed">'+
         //
         '<table border="1" ui-sortable="sortableOptions" ng-
            model="child"><tbody >'
            // funcion recursiva (se repite )
            + '<tr ng-repeat="(key, val) in child track by
               $index " >'
               // delete button
               + '<i class="deleteKeyBtn glyphicon glyphicon-
                  trash" ng-click="deleteKey(child, $index)"></i
                  >'
               + '<td >' + switchTemplate + '</td>'
            + '</tr>'
            // repeat end
         + '</tbody>'
         + addItemTemplate //Add button and function
                     '</table>'
      + '</div>';
```

Función recursiva que añade los elementos:

```
var switchTemplate =
      '<span ng-switch on="getType(val)" >'
```

```
            + '<json ng-switch-when="Object" child="val" type="
               object" default-collapsed="defaultCollapsed" ></json
               >'
            + '<json ng-switch-when="Array" child="val" type="array
               " default-collapsed="defaultCollapsed"></json>'
            + '<span ng-switch-when="Boolean" type="boolean">'
               + '<input type="checkbox" ng-model="val" ng-model-
                  onblur ng-change="child[key] = val">'
            + '</span>'
            + '<span ng-switch-when="Number" type="number"><input
               type="text" ng-model="val" '
               + 'placeholder="0" ng-model-onblur ng-change="child[
                  key] = possibleNumber(val)"/>'
            + '</span>'
            + '<span ng-switch-default class="jsonLiteral"><input
               type="text" ng-model="val" '
               + 'placeholder="Empty" ng-model-onblur ng-change="
                  child[key] = val"/>'
            + '</span>'
         + '</span>';
```

En los elementos finales (es decir, que no tienen “hijos” sino únicamente “padres”) que añade la función *AddItemTemplate* cuya función es dibujar la clave y el valor asignado con los botones de Editar, Añadir y Borrar.

Como se puede observar, dentro de esta función se está aplicando métodos característicos de Angular JS [14]:

- ng-switch: Es similar a *ng-if*. Realiza condiciones dentro de lo elementos del archivo, de manera que actúa como filtro para aquellos que no lo cumplan, a su vez, puede tener una condición por defecto para cuando se anida condiciones y no se cumple ninguna.

```
<ANY ng-switch="expresion">
  <ANY ng-switch-when="Valor1">...</ANY>
  <ANY ng-switch-when="Valor2">...</ANY>
  <ANY ng-switch-default>...</ANY>
</ANY>
```

- ng-model [4.1]
- ng-change: Evalúa inmediatamente una expresión asignada cuando el usuario cambia la entrada, a diferencia del evento de JavaScript *onChange* que lo evalúa al final del elemento (cuando el usuario a dejado de interactuar con él)

```
<ng-change
  ng-change="expression">
...
</ng-change>
```

- ng-show: Permite que un elemento de la página se haga visible o invisible en función de cualquier valor del *$Scope* . Es similar a la directiva *ng-disabled* pero en vez de deshabilitar lo que hace es mostrar el elemento si la expresión es verdadera o falsa.

```
<ng-show
  ng-show="expression">
...
</ng-show>
```

- ng-init: Sirve para inicializar datos en la aplicación por medio de expresiones que se evaluarán en el contexto donde hayan sido definidas, es decir, permite cargar elementos en el modelo al inicializarse la aplicación. En el siguiente ejemplo, se consigue que la aplicación inicialice el *$Scope* con un dato llamado "miArrayDatos"que contiene un array vacío:

```
<div ng-app ng-init="miArrayDatos = [];">
```

- ng-click: Reacciona ante eventos del tipo “click”. En ella se debe especificar el código que se pretende evaluar cuando se produzca un evento sobre él. Son normalmente usadas para botones o elementos visuales dentro de la aplicación.

```
<input type="button" value="Haz Clic" ng-click="procesarClic()
   ">
```

- ng-class: Asigna dinámicamente una clase específica de CSS [1] para representarla en el HTML. Funciona de tres formas diferentes:
  - Si la expresión se evalúa como una cadena, esta tiene que tener uno o más elementos delimitados por espacios.

- Si la expresión se evalúa como un objeto, cada *key* es diferenciada como una clase. Pudiendo diferenciar diferentes clases dentro de un objeto.
- Si la expresión se evalúa como una matriz, cada elemento de la matriz debe ser cualquiera de lo dos casos, esto significa que se pueden mezclar secuencias de cadenas y objetos diferenciándose a través del CSS asociado.

```
<ANY
  ng-class="expresion">
...
</ANY>
```

5.3.3. Funcionalidades

El usuario podrá realizar varias acciones aparte de la visualización del archivo:

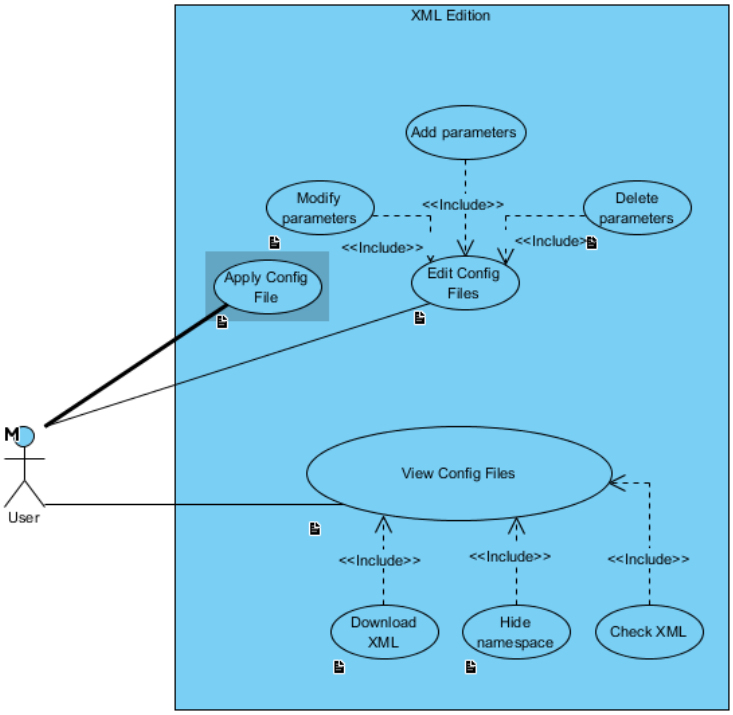

Figura 5.10: XML Casos de uso

- Actualizar la configuración del Archivo a través de un botón de *Upload*.

Figura 5.11: Botón de Upload

```
<button id="button" type="button" ng-click="uploadFile()">
   Up-load</button>
```

Para ello se ha utilizado la función que se muestra a continuación:

```
$scope.uploadFile = function() { //nombre de la funcion
   var fd = new FormData();
 //Coge el primer archivo seleccionado, ya que sobre el se
    han realizado los cambios
   var file=new File([blob2],"servers.xml");
```

```
      fd.append("file", file);
    //Llamada POST con la IP correspondiente, por ejemplo:
      $http.post("http://172.23.33.126/upload.php", fd, {
       withCredentials: false,
       headers: {'Content-Type':undefined},
       transformRequest: angular.identity
      });
```

Donde a través de una llamada *POST* enviará el archivo a la dirección IP correspondiente, actualizando el archivo o generando uno nuevo (si el nombre que se le asigna no es el mismo que el del archivo inicial, se creará un archivo nuevo)

La llamada a dicha función se realizará mediante el método *ng-click* de Angular JS. Se trata de una directiva capaz de especificar el comportamiento de una variable cuando es pulsado. En este caso, llamará al método *uploadFile*. Su uso es muy sencillo, solo es necesario asignar la expresión que se quiere ejecutar:

```
<ng-click
 ng-click="expression">
...
</ng-click>
```

En este caso, la llamada al método se produce cuando el usuario realiza una pulsación sobre un botón generado en el *index* :

```
<button id="button" type="button" ng-click="uploadFile()">Up-
   load</button>
```

- Editar el archivo XML: el usuario podrá añadir, borrar y modificar elementos dentro del XML.

  Para ello, hace uso de una directiva de Angular JS creada para realizar todo tipo de acciones correspondientes con la visualización [5.3.2], editado y borrado de los elementos.

  - Añadir elementos: Para añadir elementos, se utilizará la función *addItem*. Dicha función coge el valor del *$Scope* [4.1] que se está generando y comprueba el tipo al que pertenece. Para ello, realiza una serie de comprobaciones (reglas impuestas por el diseño del sistema) como son: que el nombre no empiece con un " $" ni con un "_" entre otras.

Una vez se han realizado las comprobaciones, se añadirá el elemento dependiendo de su tipo en el "*Object*" o "*Array*" correspondiente.

```
scope.addItem = function(obj) { //Funcion encargada
   de aniadir nuevos elementos
   if (getType(obj) == "Object") { //En caso de que
      sea del tipo Objeto
      //Checkeamos si contiene estos valores
         iniciales no permitidos
      if (scope.valueType==textName){
         scope.keyName = "__text";
      }
      if (scope.valueType==prefixName){
         scope.keyName = "__prefix";
      }

      if ((scope.keyName == undefined || scope.
         keyName.length == 0 )){
         alert("Please fill in a name");
      } else if (scope.keyName.indexOf("$") == 0){
         alert("The name may not start with $ (the
            dollar sign)");
      } else if (scope.keyName.indexOf("_") == 0 &&
         scope.valueType!=textName){
         alert("The name may not start with _ (the
            underscore)");
      }
```

En caso de que ya exista el nombre que se quiera generar, el usuario tendrá que confirmarlo por pantalla, ya que esto puede ser debido a un error.

```
      else {
         if (obj[scope.keyName]) { //Si existe ya el
             nombre que se quiere generar
            if( !confirm('An item with the name "'+
               scope.keyName
               +'" exists already. Do you really
                  want to replace it?') ) {
               return;
            }
         }
```

En caso de ser tipo numérico, comprobará que el valor con el que se rellena, sea un número y no otro tipo.

```
            if (scope.valueType == numberName && !
               isNumber(scope.valueName)){
               alert("Please fill in a number");
               return;
            }
```

```
      scope.possibleNumber = function(val) { //Funcion que
          comprueba si es un numero o string en el caso de
          Number
         return isNumber(val) ? parseFloat(val) : val;
      };
```

Una vez definido el tipo de elemento, los añade al Object dependiendo del tipo definido.

```
            switch(scope.valueType) {
               case stringName: obj[scope.keyName] =
                  scope.valueName ? scope.valueName : "
                  ";
                           break;
               case numberName: obj[scope.keyName] =
                  scope.possibleNumber(scope.valueName)
                  ;
                            break;
               case objectName: obj[scope.keyName] =
                  {};
                           break;
               case arrayName: obj[scope.keyName] = [];
                           break;
               case atributeName: obj["_"+scope.keyName
                  ]=scope.valueName ? scope.valueName :
                   "";
                           break;
               case textName: obj["__text"]=scope.
                  valueName ? scope.valueName : "";
```

```
                        break;
                    case prefixName: obj["__prefix"]=scope.
                        valueName ? scope.valueName : "";
                        break;

                    case boolName: obj[scope.keyName] =
                        false;
                        break;
                }
                //clean-up
                scope.keyName = "";
                scope.valueName = "";
                scope.showAddKey = false;
            }
        } else if (getType(obj) == "Array") { //Aniade
            elementos al Array dependiendo del tipo
            if (scope.valueType == numberName && !isNumber
                (scope.valueName)){
                alert("Please fill in a number");
                return;
            }
            // add item to array
            switch(scope.valueType) {
                case stringName: obj.push(scope.valueName ?
                    scope.valueName : "");
                        break;
                case numberName: obj.push(scope.
                    possibleNumber(scope.valueName));
                        break;
                case objectName: obj.push({});
                        break;
                case arrayName: obj.push([]);
                        break;
                case boolName: obj.push(false);
                        break;
                case atributeName: obj.push(scope.valueName
                    ? scope.valueName : "");
                        break;
                case textName: obj.push([scope.valueName ?
                    scope.valueName : ""]);
```

```
                        break;
                case prefixName: obj.push([scope.valueName
                    ? scope.valueName : ""]);
                        break;

            }
            scope.valueName = "";
            scope.showAddKey = false;
        } else {
            console.error("object to add to was " + obj);
        }
    };
```

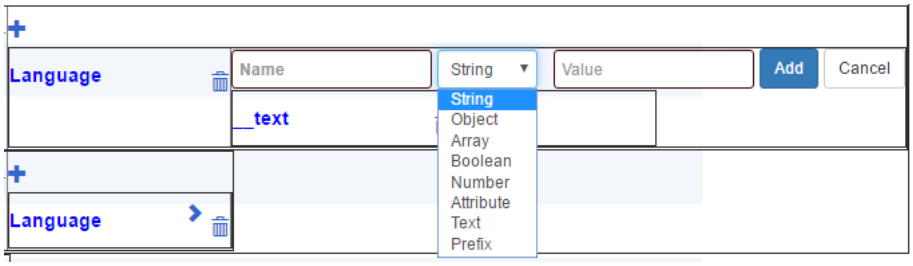

Figura 5.12: Funcionalidad de añadir un nuevo elemento

- Borrar un elemento: Para borrar un elemento se utiliza una función dentro de la directiva [4.1] nombrada anteriormente.

```
    scope.deleteKey = function(obj, key) {
        if (getType(obj) == "Object") {
            if( confirm('Delete "'+key+'" and all it
                contains?') ) {
                delete obj[key];
            }
        } else if (getType(obj) == "Array") {
            if( confirm('Delete "'+obj[key]+'"?') ) {
                obj.splice(key, 1);
            }
        } else {
            console.error("object to delete from was " +
                obj);
        }
    };
```

En ella se realizará un chequeo del tipo del elemento antes de su borrado (ya que, si es de tipo *Object* o *Array* borrará también su contenido) asegurándose así que el usuario ha realizado la acción de manera intencionada.

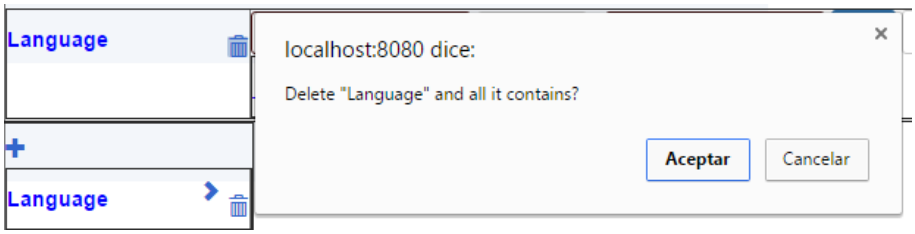

Figura 5.13: Ejemplo de borrado

- Descarga del archivo: a través del botón superior *Download*, el usuario podrá descargarse una copia del archivo en formato XML. Para ello, se hace uso de una función

Figura 5.14: Botón de Download de la interfaz

similar a la definida en *uploadFile*, donde se realiza una conversión del archivo de JSON a XML para mantener el formato del archivo.

Para ello, se captura el evento del botón denominado "divButton"del HTML.

```
<div id="divButton" class="button">
```

Por medio de la función *"appendChild"* podemos incluir en un nodo un nuevo hijo. El nuevo nodo se incluirá inmediatamente después de los hijos ya existentes (si existen) y el nodo padre contará con una nueva rama.

Esta función se encontrará dentro del controlador.

```
var x2js2 = new X2JS();
var b = document.createElement('a');
b.download = "backup.xml";
//b.textContent = "Download backup.xml";
document.getElementById('divButton').appendChild(b);

var c = document.createElement('button');
c.textContent = "Download backup.xml";
c.type="button";
c.id="button";
b.appendChild(c);
```

- Chequeo del formato del XML: se ha implementado otro botón adicional para chequear si el formato del XML se ha generado en el formato correcto.

```
<input id="button" type="button" value="Check XML" onclick="
   validateXML()">
```

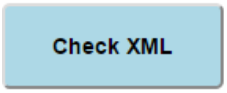

Figura 5.15: Botón de Chequeo del formato XML

Para ello, se hace uso de una función denominada *"validateXML"* que actuar al pulsar sobre el botón. Dicha función chequea si el formato es correcto a través de unas reglas llamando a *"checkXML"* que devolverá un mensaje con el resultado, si dicho resultado es negativo indicará la línea del error.

```
function validateXML()//Comprueba si el XML esta bien generado
{
  try
  {
     var text=xmlAsStr;
     var parser=new DOMParser();
     var xmlDoc=parser.parseFromString(text,"application/xml");
     }
  catch(err)
  {
     alert(err.message) //Muestra a traves de una alerta en el
        navegador el resultado del error
  }
if (xmlDoc.getElementsByTagName("parsererror").length>0)
   {
      checkErrorXML(xmlDoc.getElementsByTagName("parsererror")
         [0]);
      alert(xt) //Muestra las lineas del error
   }
 else
   {
      alert("No errors found"); //Mensaje de todo correcto
   }
 }
```

```
var xt="",h3OK=1 //Inicializa la variable de numero de linea
   ademas de donde se comienza a leer
function checkErrorXML(x)
{
   xt=""
   h3OK=1
   checkXML(x)
}

function checkXML(n) //Funcion con las normas para comprobar
   el XML
{
   var l,i,nam
   nam=n.nodeName
   if (nam=="h3")
      {
      if (h3OK==0)
         {
         return;
         }
      h3OK=0
      }
   if (nam=="#text")
      {
      xt=xt + n.nodeValue + "\n"
      }
   l=n.childNodes.length
   for (i=0;i<l;i++)
      {
      checkXML(n.childNodes[i])
      }
}
```

Cabe destacar, que la funcionalidad principal recae en comprobar si el archivo que se recibe del servidor tiene un formato XML correcto.

Dentro de dichas funcionalidad, se pueden encontrar más funcionalidades características de Angular JS además de las citadas anteriormente:

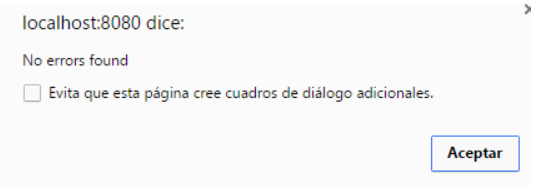

Figura 5.16: Ejemplo de mensaje sin errores

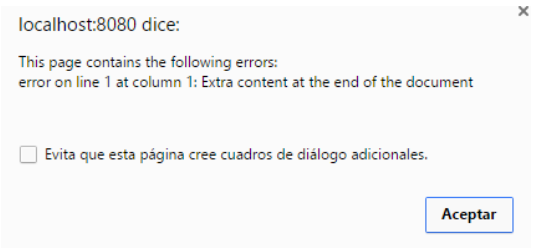

Figura 5.17: Ejemplo de mensaje con errores

- ng-options: Es similar al *ng-repeat*. Rellena las opciones del *ng-model* automáticamente a partir de los datos del *$Scope* [4.1]. Por ejemplo, si tenemos una lista de elementos:

```
$scope.items = [{
  id: 1,
  label: 'Item1',
}, {
  id: 2,
  label: 'Item2',
}];
```

  Se podrán seleccionar de esta forma:

```
<select ng-options="item as item.label for item in items track
    by item.id" ng-model="selected"></select> //Seleccionara
   los elementos dentro de items por el id
$scope.selected = $scope.items[0]; //Representara el elemento
   0 de los items
```

- ng-blur: Indica a Angular JS qué hacer cuando un elemento HTML pierde el foco. Su sintaxis es la siguiente:

```
< element ng-blur=" expression "></ element >
```

  Un ejemplo puede ser:

```
<html>

<input ng-blur="contador = contador + 1" ng-init="contador=0"
   />

<h1>{{contador}}</h1>
</html>
```

De esta forma, cada vez que se genere un evento en el campo de entrada, se sumará una unidad al contador.

Figura 5.18: Ejemplo ng-blur

5.4. Menú de selección

El menú actual , no es un menú que se pueda ocultar al usuario a la hora de representar elementos,ocupando espacio a la hora de representar elementos haciendo las páginas más grandes y menos visuales.

Por ello, se ha decidido realizar un menú más sencillo que cumpla con las siguientes características:

- Que no ocupe espacio en la pantalla cuando se están representando elementos.
- Desplegable dinámicamente.
- Que se pueda ejecutar desde cualquier módulo en cualquier momento.
- Sencillo, que no sea complejo su utilización.
- Visual, que el usuario pueda acceder a los componentes.

- Modular, permitiendo tener los archivos correspondientes a cada módulo diferente en carpetas separadas para una organización mejor, pudiendo ampliarse en el futuro con más funcionalidades.

Para ello, se ha implementado un menú que a través de un botón se muestre:

```
</h4>  <button ng-click="showLeft($event)">Show Left Menu!</
   button>
```

Figura 5.19: Botón para mostrar el menu

El menú será una directiva de Angular JS que tomará dos argumentos:

- Visibilidad, encargada de indicar qué variable del controlador debe ser responsable de mostrar el menú. En este caso será un botón asociado a la pulsación del usuario.

  A través de una función *ng-click* [5.3.2] se realizará una llamada a la función *showLeft*, encargada de mostrar el menú si la variable *leftVisible* se pone a verdadero.

  ```
  $scope.showLeft = function(e) {
                $scope.leftVisible = true;
                e.stopPropagation();
            };
  ```

- Alineación, que determinará si el menú se desliza desde el lado derecho o izquierdo de la pantalla. Siendo en este caso el lado izquierdo.

```
   <menu visible="leftVisible" alignment="left">
```

En este caso, el código HTML indicará el estado de la directiva del menú establecido y proporcionará su uso, en el JavaScript irá el código de la directiva así como el vínculo del controlador de Angular JS en la propia directiva, y el CSS se especificarán los efectos del deslizamiento del menú.

El uso de *ng-transclude* es convertir los elementos secundarios en la directiva, indicando el número de elementos en el menú, es decir, el número de nodos que se colocarán en el menú.

A su vez, en JavaScript, se describen el menú principal y los menús anidados a este.

Separando el código, se pueden observar las distintas partes con sus funcionalidades:

- Inicializamos la aplicación de los menús, conectando eventos que permitan ocultar los menús una vez se revelan, de forma que cuando el usuario realiza un clic en cualquier parte del documento, estos deben ocultarse. Angular JS, utilizará la variable *$rootScope* como emisor de eventos, ocupándose de reaccionar a este tipo de eventos.

```
app.run(function($rootScope) {
   document.addEventListener("keyup", function(e) {
      if (e.keyCode === 27)
         $rootScope.$broadcast("escapePressed", e.
            target);
   });

   document.addEventListener("click", function(e) {
      $rootScope.$broadcast("documentClicked", e.
         target);
   });
});
```

- El controlador manipula los elementos para mostrarlos y ocultarlos. A través de la opción de *showRight* mostrará el menú a la izquierda. Estará a su vez vinculado al emisor de eventos nombrado anteriormente (*$rootScope*) para cerrar los menús cuando se produzca un evento.

```
app.controller("modalDemo", function($scope,
   $rootScope,$state) {

   $scope.content = ['red', 'blue'] //Paginas que
      contiene

   $scope.setPage = function (page) { //Transicion
      $state.transitionTo(page);
   };
   $scope.leftVisible = false;
   $scope.rightVisible = false;

   $scope.close = function() { //Cerrar el menu
```

```
            $scope.leftVisible = false;
            $scope.rightVisible = false;
        }
        $scope.showLeft = function(e) { //Funcion de
            ventana a la derecha para enseniar el menu
            $scope.leftVisible = true;
            e.stopPropagation();
        };
        $rootScope.$on("documentClicked", _close);
        $rootScope.$on("escapePressed", _close);

        function _close() { //Cerrar el menu
            $scope.$apply(function() {
                $scope.close();
            });
        }
    });
```

- El siguiente código, es el encargado de controlar las dos directivas del menú:

```
    app.directive("menu", function() {
        return {
            restrict: "E",
            template: "<div ng-class='{ show: visible, left
                : alignment === \"left\", right: alignment
                === \"right\" }' ng-transclude></div>",
            transclude: true,
            scope: {
                visible: "=",
                alignment: "@"
            }
        };
    });

    app.directive("menuItem", function() {
        return {
            restrict: "E",
            template: "<div ng-click='navigate()' ng-
                transclude></div>",
            transclude: true,
```

```
            scope: {
               hash: "@"
            },
            link: function($scope) {
               $scope.navigate = function() {
                  window.location.hash = $scope.hash;
               }
            }
         }
      });
```

La restricción "E" significa que restringe los elementos, de esta forma transfiere el contenido interno de ambos a cada directiva [4.1]. De esta forma el contenido interno que tiene cada módulo se mostrará dependiendo de la etiqueta seleccionada, siendo esto posible a través de la directiva de Angular *ng-transclude.* Se ha creado a su vez un método bidireccional que permite leer el valor de la variable visible. De esta forma se podrá navegar entre menús, a través del clic que realice el usuario (registrado a través de la directiva *ng-click*).

6. Rendimiento del sistema

Una de los motivos para el cambio de tecnología en la implementación del sistema, ha sido la sobrecarga del servidor a la hora de generar la aplicación web. Como se ha visto, el primer paso ha sido el cambio de arquitectura para evitar carga innecesaria en el servidor.

Al tratarse de una aplicación encargada de procesar archivos con tamaño variable (dependiendo de la función que se quiera generar), se ha hecho un estudio de cuál sería el tiempo y cuánto crecería dependiendo del tamaño del archivo. Para ello se ha tenido en cuenta cuánto tardaría en generar un archivo de un tamaño considerable para el sistema (ya que, suelen rondar en torno a los 30 Mb).

Para medir el tiempo se usa una herramienta de desarrollo de Chrome que a través del navegador permite ver los segundos que tarda una aplicación y la asignación de memoria, pudiendo detectar pérdidas de ésta si las hubiese y analizar cuánta memoria necesita la página.

Tamaño archivo	Tiempo GET	Generación total
936 Kb	103 ms	1,10 s
10,28 Mb	909 ms	2,2 s
50 Mb	1,43 s	5,28 s
102 Mb	2,07 s	13 s

Tabla 6.1: Tabla de rendimientos

Como se puede observar en la tabla, a partir de archivos en torno a los 50 Mb, su generación no es inmediata (ya que, en los casos anteriores, su generación es bastante imperceptible)

Se han realizado pruebas a su vez comparando en ambos sistemas con tamaños de archivos diferentes, obteniendo que Angular es ligeramente más rápida.

Cabe destacar, que en Angular JS hay varias formas de mejorar el rendimiento del sistema, una relacionada con la directiva *ng-repeat* y otra con el filtrado. Algunas de las técnicas a tener en cuenta son las siguientes:

- La directiva *ng-repeat* se vuelve lenta por encima de 2.500 enlaces de datos bidireccionales. La mejor manera de solucionar esto, es aplicar el método de paginación de los datos, ya que no será necesario mostrar todos los elementos en una primera interacción. [21].

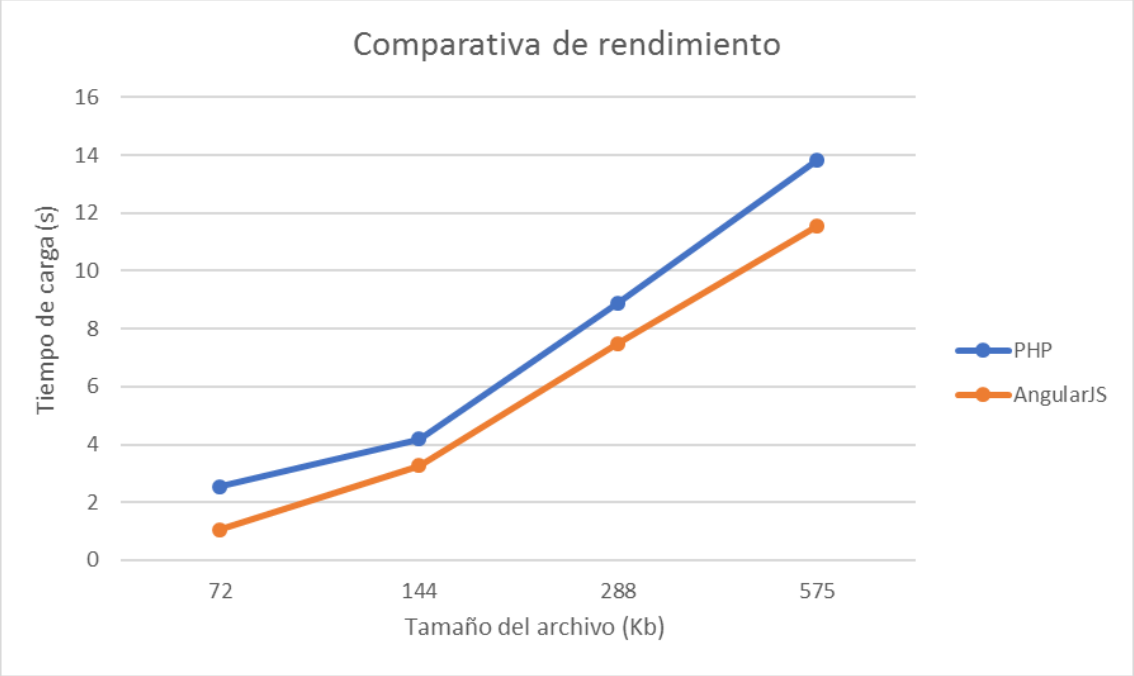

Figura 6.1: Rendimiento en función del tamaño de PHP vs AngularJS

- El método *ng-show* puede sobrecargar la aplicación, se pueden usar directivas o plantillas adicionales para mostrar elementos de un sistema, como puede ser *ng-if*, evaluando los elementos adicionales del DOM y los enlaces de datos bajo demanda únicamente.
- Es recomendable no usar directivas como *ng-mouseenter* (que detecta el paso del ratón), ya que crean sobrecargas en la aplicación. Es mejor utilizar jQuery para crear animaciones o efectos.
- El uso de filtros en listas largas puede llevar a problemas de rendimientos. Es útil prefiltrar los datos si es posible. Para ello se puede filtrar antes en la directiva *ng-repeat*, devolviendo desde su uso un conjunto más acotado de elementos (eliminandolos así del DOM, y del *Scope*). Si no, se puede realizar este filtrado previo con el uso de directivas como *ng-show* o *ng-hide*.
- Otra forma de evitar el sobre-exceso al filtrar, es aplicar los filtros una vez el usuario deje de escribir.

7. Resultados

Se ha realizado de manera satisfactoria una aplicación capaz de sustituir a la actual realizando nuevas funciones además de las actuales, además de las siguientes características:

- Se ha realizado una aplicación dividida en un conjunto de módulos funcionales, pudiendo acoplarse a un menú desplegable como dictaban las especificaciones de la empresa.
- Se ha comprobado el uso de nuevas tecnologías en el área de aplicaciones web comprobando que su eficiencia es mayor a tecnologías anticuadas como PHP. Cabe destacar, que el uso de una tecnología del lado del cliente aprovecha mejor los recursos de los navegadores de hoy en día, mejorando de esta forma su rendimiento.
- El editor utilizado, ha resultado ser dinámico y con nuevas funcionalidades.[7.1]

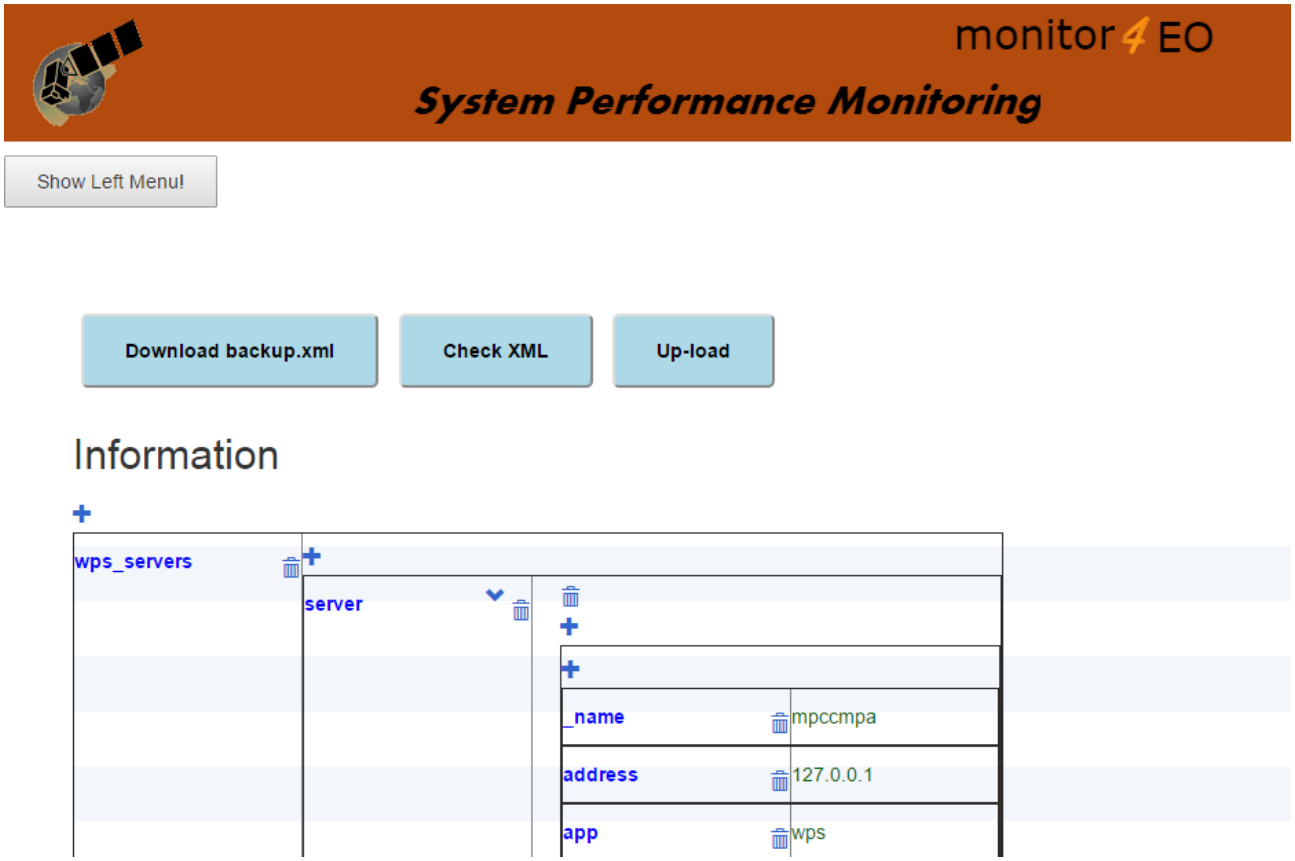

Figura 7.1: Editor de archivos XML final

A su vez,destaca sobre el editor la carga optimizada de archivos, la posibilidad de editar y borrar de manera dinámica y visual (no en formato XML como en el caso

anterior) y el hecho de poder minimizar archivos que contengan muchos elementos.

- Se ha añadido un botón que permite al usuario descargar el archivo, útil en el caso de que se quiera guardar una configuración, ya que en la herramienta en PHP, el usuario no podía guardar configuraciones siendo necesario el uso de NotePad o herramientas de pegado para guardar archivos.

- En el caso del visor de archivos, se ha realizado una herramienta dinámica y visual, donde el usuario puede filtrar de diferentes maneras el contenido de la aplicación, siendo a su vez más rápida que la anterior y eficiente.

  Entre las funcionalidades añadidas destaca la descarga de archivos, igual que en el caso del editor, y el uso de un sistema diferente para la actualización de archivos. Dicho sistema destaca en que en el diseño antiguo se actualizaban los archivos en función de un temporizador, mientras que aquí se actualizan cuando el usuario lo considere necesario. [7.2]

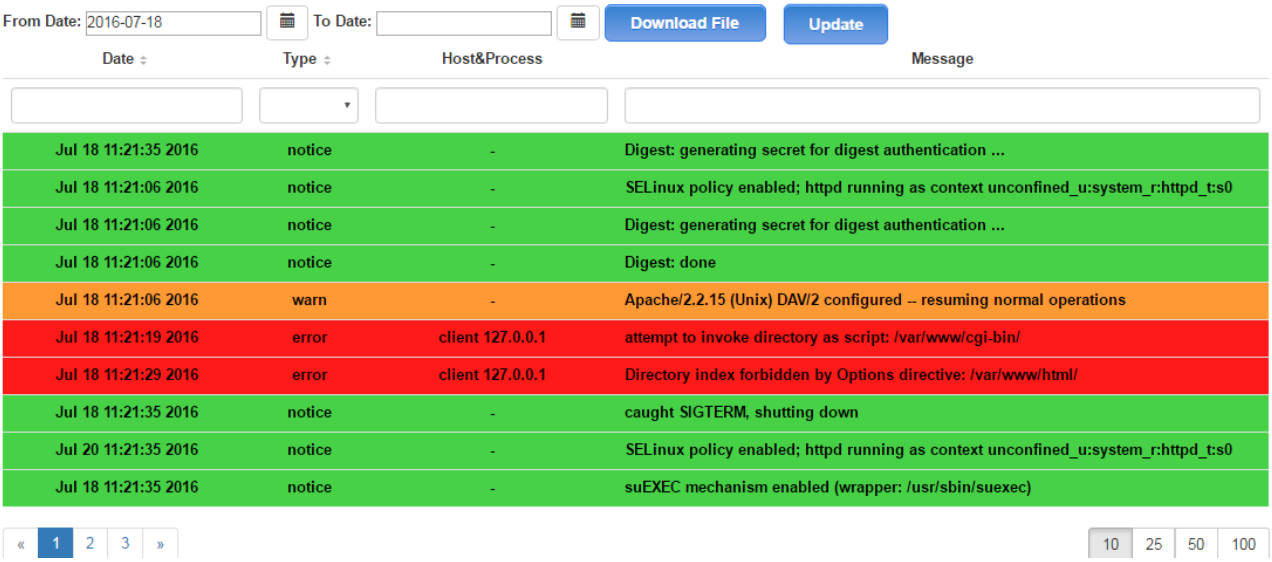

Figura 7.2: Visor de archivos XML

- Se han cumplido las especificaciones de la empresa, añadiendo a su vez elementos adicionales, como un pequeño gráfico donde se puede contabilizar el número de elementos de cada tipo de manera más intuitiva (ya que no es lo mismo perder dos paquetes de información que un número mucho mayor).
- A su vez, destaca el uso de un menú desplegable. Las especificaciones requerían el uso de un menú que no invadiera el espacio de visualización como el anterior, logrando de esta forma más espacio para archivos de gran tamaño. [12.1]

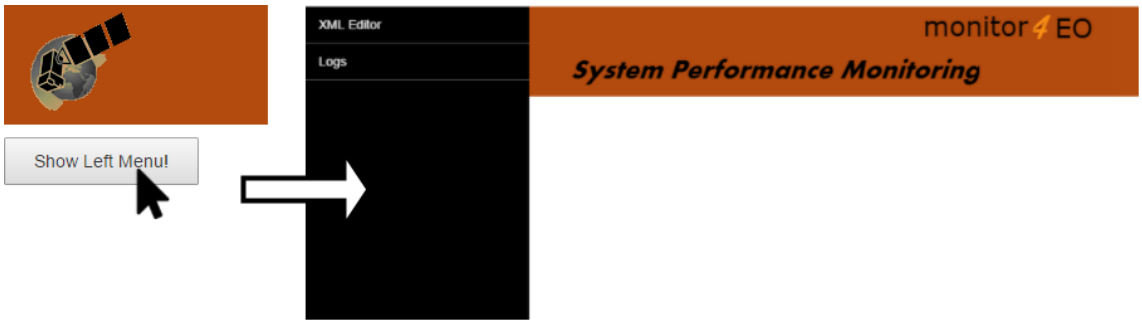

Figura 7.3: Menú desplegable

8. Discusión

Se ha desarrollado una aplicación en Angular JS cuya principal característica es que es capaz de realizar una interfaz más intuitiva para el usuario de manera sencilla.

Mediante el uso de elementos visuales (como los colores en el visor de archivos) o elementos capaces de interactuar con el propio cliente (como por ejemplo los botones y el calendario) se ha comprobado la agilidad que posee esta herramienta a la hora de representar información. Gracias al modelo MVVM[Glosario 4] en el que se basa, se puede destacar sobre el resto de funcionalidades la facilidad de realizar filtros sobre conjuntos de datos (con la directiva *filter* aplicada en un *ng-repeat*[5.2.2]), o la forma de aplicar estilos (con la directiva *ng-class* [5.3.2]).

Además de las ventajas que se observan para visualizar la información, se ha podido comprobar a través del desarrollo de estos módulos, las ventajas y desventajas de Angular JS.

Entre las ventajas, cabe destacar el uso de Angular en los sistemas o *frameworks*. Ya que usa un sistema en el que tanto la vista como el modelo están en relación constante (lo que Google llama *"Two way data binding"*), logrando de esta forma que los cambios visuales se actualicen en tiempo real en el modelo (y viceversa) evitando que el desarrollador sea el encargado de lograr esta sincronía.

Pero el uso de ello hace que el sistema tenga que evaluar cada variable de las usadas por Angular en cada ciclo de la aplicación, suponiendo un problema a la hora de desarrollar, ya que se puede llegar a tener estructuras de datos complejos a evaluar, consumiendo más tiempo de ejecución.

También destaca en Angular el uso de directivas indican al compilador de Angular el comportamiento específico que tienen los elementos, pudiendo trabajar de esta forma a nivel de componentes, además de permitir que estos sean reutilizables en toda la aplicación.

A su vez, posee una gran comunidad de desarrolladores que dan soporte a este *framework* teniendo varios ejemplos y módulos creados, facilitando su desarrollo y uso para aplicaciones. Esto puede facilitar el desarrollo de módulos reutilizando código, pero teniendo cierto cuidado ya que al inyectar dependencias se cargan los módulos necesarios para cargar la aplicación, implicando un volumen de tráfico que será mayor conforme más dependencias se usen.

Aunque su mayor desventaja, recae en el uso de uno de sus elementos más importantes, los *scopes*. Aunque estos permiten heredar propiedades de otros elementos, también permite que un *scope* pueda influir en el estado de otro, haciendo que Angular comparta elementos. Para evitar esto se suelen usar directivas con sus propios *scopes*, de manera que no comparten su estado con los elementos que se encuentren fuera de esta.

Como conclusión, Angular es un buen *framework* para el desarrollo de aplicaciones web, ya que logra interfaces fluidas y buena experiencia de usuario (donde no se aprecia la interacción entre la aplicación y el servidor). También destaca el diseño por componentes teniendo cada uno su propia lógica a través de módulos independientes donde se evitan variables compartidas, pudiendo reutilizar módulos en su futuro. Se trata por tanto de un *framework* equilibrado que proporciona el uso de una arquitectura estándar que combina el uso de *templates* como el *databinding*, que necesita pocos años para estar implantado en las empresa, ya que usa un lenguaje de programación poco convencional y aceptado como es JavaScript.

9. Trabajo futuro

Existen una serie de mejoras que se pueden aplicar a los módulos realizados:

- Realizar un sistema capaz de escoger entre archivos el que se desee visualizar o editar. Para ello se podrá hacer uso de un menú capaz de enseñar que archivos tiene el servidor y seleccionar entre ellos, pudiendo de esta forma tener varios archivos de diferentes tamaños.
- El uso de animaciones para realizar una interfaz más amigable con el usuario. Se pueden usar animaciones para hacer transiciones entre módulos y funcionalidades, para ello existen paquetes ya desarrollados como puede ser *angular-animate*[22] que permite realizar animaciones de forma sencilla. Un ejemplo podría ser el siguiente:

```
app.animation('.my-repeat-animation', function() {
    return {
      enter : function(element, done) {
        jQuery(element).css({
          position:'relative',
          left:-10,
          opacity:0
        });
        jQuery(element).animate({
          left:0,
          opacity:1
        }, done);
      },

      leave : function(element, done) {
        jQuery(element).css({
          position:'relative',
          left:0,
          opacity:1
        });
        jQuery(element).animate({
          left:-10,
          opacity:0
        }, done);
      },
```

```
    move : function(element, done) {
      jQuery(element).css({
       opacity:0.5
      });
      jQuery(element).animate({
       opacity:1
      }, done);
    }
   };
});
```

Que permitiría realizar animaciones a la hora de pasar por encima el ratón de elementos.

- Permitir búsquedas en el editor, de manera que el usuario pueda localizar rápidamente el archivo que quiere editar o borrar.

A su vez, desde Septiembre de 2016 ha salido Angular JS 2, tratándose de una versión totalmente diferente a Angular JS, que combinada con *TypeScript* puede ser el *framework* que vaya a triunfar en el desarrollo web. Las diferencias más notables de esta nueva versión son las siguientes:

- Controladores y Componentes: Se ha simplificado el uso de componentes, realizando llamadas más directas y en código aparte, siendo necesario el uso de *imports* para su invocación.
- Bootstrapping : En Angular 1, es necesario invocar el método *ng-app* en el HTML para inicializar la aplicación. En Angular 2 se realizará a través de *imports* del *AppComponent*, teniendo en dicho componente la inicialización de la aplicación.
- Directivas: El *ng-repeat* será sustituido por **ngFor*, consumiendo menos recursos.
- Servicios: En Angular 1, había muchas formas de crear servicios, en Angular 2 podrá ser únicamente a través de una clase Injectable.

```
@Injectable()
export class MiServicio {
 public MiServicio = () => [
  //Funciones a realizar
```

```
  ];
}
```

- Sintaxis en lo referente a los inputs de los eventos. En Angular 2 se escribe el evento al que se quiere referenciar entre paréntesis. Por ejemplo, en Angular 1:

```
<button ng-click="thing.submit(item)" type="submit">
```

Angular 2:

```
<button (click)="submit(item)" type="submit">
```

- El *Scope* que era útil en Angular 1, pero daba problemas de asociaciones, se ha sustituido por el *Controller AS* siendo uno de los cambios más notables.

Por ejemplo, en Angular 1 :

```
angular.module(''example'').controller(''ExampleCtrl'',
   function(\$scope) {

   \$scope.name = ''Victoria Villa'';

});
```

En Angular 2:

```
(function() {

var AppComponent = ng
.Component({
selector: 'mi-aplicacion',
template: '<h1>Mi aplicacion Angular 2</h1>'
})
.Class({
constructor: function () {
   this.name = "Victoria Villa";
}
})();
```

A su vez, la aparición de *TypeScript* hace que sea más sencillo realizar aplicaciones en Angular JS 2. *TypeScript* se trata de un editor que incluye todas las funcionalidades de

ECMAScript6 (el editor usado comúnmente para desarrollar en AngularJS) además de una capa por encima de funcionalidades extra, permitiendo mezclar ambos tipos de códigos.

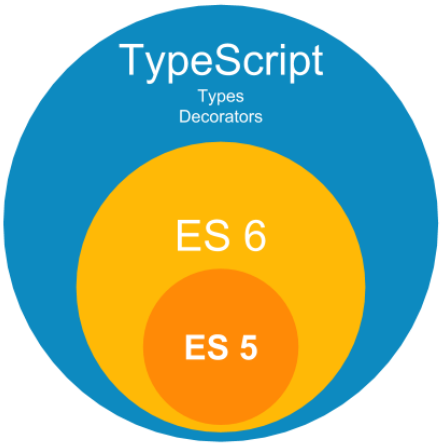

Figura 9.1: TypeScript

La mayor diferencia de TypeScript es que permite definir de qué tipo son las variables que se van a utilizar, además de que podrá ayudar al usuario a la hora de solucionar errores de compilación, ya que al poderse definir el tipo de variable, puede detectar errores de asignaciones y tipado, siendo por tanto una opción futura a considerar.

10. ANEXO I : Arquitectura básica de una aplicación web

Una arquitectura básica de una aplicación web está formada por los siguientes elementos:

- Un navegador: el navegador será la interfaz a manejar por el usuario, hará de cliente, solicitando los recursos a los servidores web.
- Servidor web: recibe las peticiones realizadas por el usuario, enviando los recursos necesarios o notificando errores en caso de que dicho recurso no exista o no se encuentre disponible.
- Protocolo HTTP: es el protocolo basado en TCP/IP que se utiliza para que el cliente pueda realizar peticiones y el servidor responda. Dicho protocolo garantiza el envío de los paquetes.
- HTML: es el formato usado para los documentos de las webs. Es un estándar que define una estructura básica y un código para la definición del contenido.

Dentro de los tipos de arquitectura, se pueden diferenciar por dinámicas en el cliente y/o servidor.

10.1. Arquitecturas con cliente y servidor estáticos

Realizan peticiones a través de HTTP. Dicho servidor transforma las URL a ruta en disco, devolviendo un fichero disco al navegador. El navegador será el encargado de visualizar la página con estilos CSS e imágenes (sin JavaScript). Cada vez que el usuario realiza una opción en el navegador, éste repetirá el proceso y recargará de nuevo la página web.

Con esta arquitectura el servidor siempre devuelve los mismos recursos. Es un tipo de aplicación que puede sobrecargar los servidores debido a las peticiones, haciendo menos eficiente el uso de recursos.

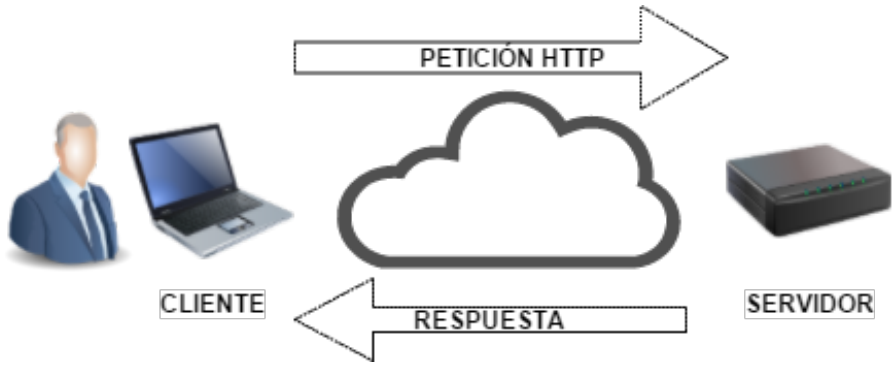

Figura 10.1: Arquitectura cliente y servidor estáticos

10.2. Arquitectura con cliente estático y servidor dinámico

En esta arquitectura se podría diferenciar un tercer elemento que sería la base de datos. Cuando el servidor recibe una petición por parte del cliente, consulta la base de datos para adquirir la información, generándose de esta forma la página web de forma dinámica.

Igual que en el caso anterior, recarga la página al completo, ejecutándose el código en el servidor, generando problemas de tiempos de carga lentos llevando a una mala experiencia de usuario.

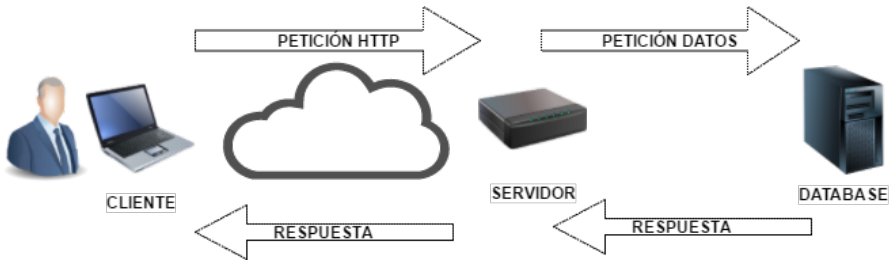

Figura 10.2: Arquitectura cliente estático y servidor dinámico

10.3. Arquitectura con cliente dinámico y servidor estático

El contenido de la página está alojada en el disco duro del servidor (estático). Las páginas web incluyen código JavaScript ejecutable en el navegador, realizando efectos gráficos que permiten realizar una experiencia más llamativa al usuario, además de mostrar u ocultar información dependiendo de las acciones y ser adaptable para móviles.

Figura 10.3: Arquitectura cliente dinámico y servidor estático

10.4. Arquitectura con cliente y servidor dinámicos

Se utiliza en la mayoría de aplicaciones de hoy en día debido a su rápida respuesta y a su poco consumo de recursos en el servidor. El cliente ejecuta una aplicación web JavaScript, que puede ser de diferentes tipos:

- JavaScript para efectos gráficos: El dinamismo del cliente es el caso anterior. Su mayor utilidad son efectos gráficos. Son las aplicaciones mas usadas
- JavaScript con peticiones en segundo plano (AJAX): Se utiliza para no tener que recargar la página completa al pulsar un link. De esta forma se realiza una petición oculta al usuario (en segundo plano), cuando llega la respuesta del servidor, JavaScript actualiza las partes de la página referentes a dicha información, mejorando así la experiencia del usuario. Al usar AJAX [Glosario 1], el servidor no genera un HTML, sino que devuelve la información en formato XML o JSON implementando normalmente una API REST.
- Single Page Application con API REST: Cuando todo el contenido dinámico se carga únicamente con JavaScript usando AJAX [Glosario 1], siendo de esta forma la aplicación un conjunto de recursos HTML,CSS y JavaScript estáticos que se cargan en el navegador. El cliente se descarga la aplicación al conectarse al URL y se comunica con el servidor a través de REST, habiendo una sola página que interactúa con el usuario a través de los datos recibidos del servidor.Un ejemplo de uso es Gmail o Google Maps.

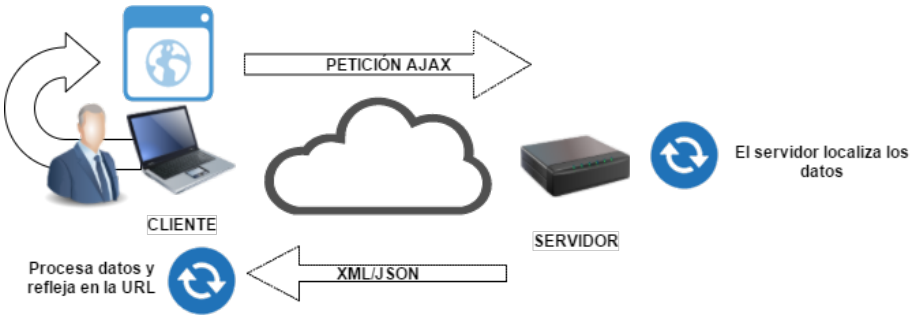

Figura 10.4: Arquitectura con cliente y servidor dinámico

11. ANEXO II : Instalación del entorno

Para que sea posible realizar pruebas con Angular JS, es necesario un conjunto de herramientas tal y como se detalla en los siguientes sub-apartados.

11.1. Instalación de node.js

Node.js es un ambiente de ejecución para JavaScript construido sobre el motor JavaScript V8 de Chrome, esto significa que su uso ya no queda restringido solamente al navegador web, sino que también es posible usar este popular lenguaje de programación para desarrollar aplicaciones de todo tipo, desde servidores web hasta aplicaciones de escritorio. De esta manera, se tendrán muchos recursos utilizando JavaScript del lado del servidor.

Por otra parte, "npm" es el administrador de paquetes de node.js y funciona de manera similar a pip en el entorno Python. Con él podemos instalar cualquier paquete node js que lleguemos a necesitar para nuestros proyectos.

Existen dos formas de instalar node.js dependiendo del sistema operativo del usuario:

- Instalación de node.js usando el instalador oficial: para instalarlo se deberá descargar el instalador apropiado para el sistema operativo de la máquina a usar, dicho instalador se podrá conseguir su página oficial.

  Una vez descargado el instalador necesario dependiendo del sistema operativo que se este utilizando, se ejecutará siguiendo las instrucciones que se van indicando en cada paso de este.

- Instalación de node.js usando un gestor de paquetes: si el sistema operativo es de tipo UNIX (Linux, Mac), es posible instalar node.js utilizando el gestor de paquetes correspondiente. Para ello hay que seguir los siguientes pasos:

  - ArchLinux: Utilizar el siguiente comando

    ```
    sudo pacman -S nodejs npm
    ```

  - Fedora

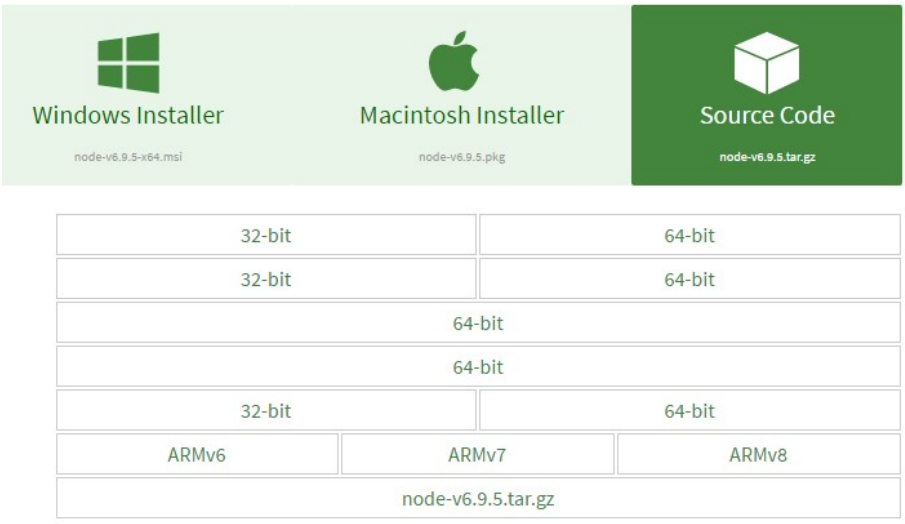

Figura 11.1: Página oficial node.js

```
sudo dnf groupinstall 'Development Tools'
sudo dnf install nodejs
```

- Debian, Ubuntu

```
sudo apt-get install build-essential
sudo apt-get install nodejs
```

- Mac (con Homebrew)

```
brew install nodejs
```

Una vez finalizado cualquiera de los comandos anteriores dependiendo del gestor de paquetes, se habrán instalado node js y npm en la máquina, y ya se podrán desarrollar aplicaciones en JavaScript.

11.2. Instalación de npm

Npm (node package manager)[10] es un gestor de paquetes de node.js que permite descargar librerías y enlazarlas o descargar programas de js. Permite empezar a desarro-

llar en node js con TDD *(Test-driven development*) o BDD (*Behaviour Driven Development*)[Glosario 1].

Con npm se pueden manejar dependencias o módulos en node.js, teniendo como usos comunes los siguientes:

- Publicar
- Descubrir
- Instalar
- Desarrollo de programas en node.js

Para su instalación, desde la versión 0.6.3 de npm, solamente es necesario instalar node.js como se indicó en [11.1].Si se está en una versión anterior, es recomendable actualizar a través de un administrador de versiones, pudiendo tener múltiples versiones de node instaladas en la máquina de desarrollo.

11.3. Instalación de Bower

Bower es una solución de gestión de paquetes para los paquetes de front-end, tales como bibliotecas JavaScript y CSS.

El método de instalación es muy sencillo, una vez se tiene instalado npm, simplemente se deberá realizar el siguiente comando, con ello se descargará la mayoría de los paquetes de Git necesarios.

```
npm install -g bower
```

Se puede encontrar una lista de todos los paquetes que se pueden instalar usando Bower en la línea de comandos.

11.3.1. Diferencias entre npm y bower

Tanto bower como npm son administradores de paquetes cuyo uso principal es facilitar el desarrollo de sistemas.

Bower es exclusivamente para librerías de front-end, mientras que npm se suele utilizar para nodejs (necesario para utilizar Angular JS y JQuery entre otros proyectos).A su vez, npm usa dependencias anidadas mientras que bower utiliza dependencias planas, por lo que este último intenta que las aplicaciones sean menos pesadas pero a diferencia de npm no permite utilizar muchas herramientas sin darle importancia a las dependencias.

Es decir, las dependencias de npm (cuyo objetivo es la estabilidad) tendrían la siguiente forma:

```
NPM project root
[node_modules] // default directory for dependencies
 -> dependency A
 -> dependency B
    [node_modules]
    -> dependency A

 -> dependency C
    [node_modules]
    -> dependency B
      [node_modules]
       -> dependency A
    -> dependency D
```

Pudiéndose observar las dependencias de manera recursiva, en formato de árbol, estando anidadas unas con otras.

Sin embargo, en Bower (cuya principal característica es tener la mínima carga de recursos) tendría la siguiente forma:

```
Bower project root
[bower_components] // default directory for dependencies
 -> dependency A
 -> dependency B // needs A
 -> dependency C // needs B and D
 -> dependency D
```

Observando las dependencias al mismo nivel.

Si observamos ambos casos, parece más sencillo el uso de Bower, pero se utiliza Npm por las dependencias entre versiones. Ya que una dependencia "B"puede requerir una versión

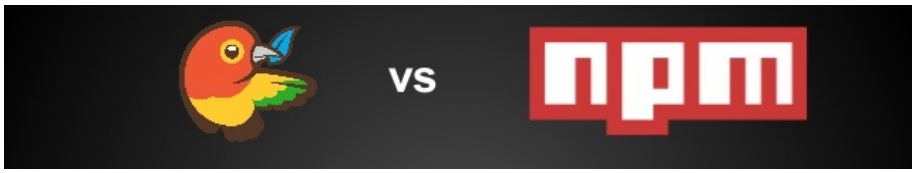

Figura 11.2: Bower vs Npm

diferente de la dependencia “A” ya instalada, haciendo que funcione de cualquier manera.

Por ello, lo común es usar ambos como administración de dependencias: Bower para los paquetes que se quieran publicar (evitando duplicidad de dependencias y una aplicación más liviana) y npm para el testeo, la construcción y la optimización de la aplicación (normalmente, para su desarrollo)

12. Anexo III : Cronograma del proyecto

En el siguiente cronograma se puede ver el tiempo dedicado a cada una de las partes.

2016/2017	OCTUBRE				NOVIEMBRE				DICIEMBRE				ENERO				FEBRERO				MARZO				ABRIL			
Tareas	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4
INICIO																												
Documentación : Lenguajes web y arquitecturas																												
Documentación : Angular JS																												
Documentación y primeras pruebas Angular JS																												
Tutorial de iniciación de Angular JS																												
Objetivos del trabajo																												
DESARROLLO																												
Desarrollo de módulo 1																												
Pruebas con datos ejemplo																												
Desarrollo de módulo 2																												
Pruebas con datos ejemplo																												
Desarrollo menú interactivo																												
Unión de aplicaciones en un archivo																												
Testeo con ejemplos																												
DOCUMENTACIÓN																												
Desarrollo de estructura																												
Redacción del documento																												
Correciones del documento																												

Figura 12.1: Cronograma del proyecto

13. Anexo IV : Presupuesto del proyecto

A continuación se puede observar el presupuesto hipotético del proyecto conforme al personal y a su tiempo correspondiente:

Descripción	Precio/Unidad	Cantidad	Importe
Coste personal			
Ingeniero Técnico (€/h)	15	500	7500
Material			
Material bibliografico	70	1	70
Material empresa	10	10	100
Asesoria			
Ingeniero Doctorado (€/h)	60	50	3000
Personal empresa	40	10	400
		Base importe	11070
		IVA %21	2324,7
		TOTAL	13394,7

Figura 13.1: Presupuesto del proyecto

Siendo por tanto el presupuesto de TRECE MIL TRESCIENTOS NOVENTA Y CUATRO EUROS CON SIETE CÉNTIMOS.

Referencias

[1] Frontend and Backend Web Technologies in Social Networking Sites: Facebook as an Example.
Hanin M. Abdullah and Ahmed M.Zeki
AdvancedComputer Science Applications and Technologies (ACSAT), 29 Diciembre 2004 .
http://ieeexplore.ieee.org/document/7076874/

[2] World Wide Web Consortium.
https:www.w3.org

[3] The Research of PHP Development Framework Based on MVC Pattern
Wei Cui and Lin Huang
Computer Sciences and Convergence Information Technology, 24 Noviembre del 2009 .
http://ieeexplore.ieee.org/abstract/document/5369976/

[4] Directive components in ng
Google 2010-2017
https://docs.angularjs.org/api/ng/directive/

[5] Node.js oficial page
2017 Node.js Foundation
https://nodejs.org/es/

[6] Django oficial page
2015-2017 Django Software Foundation
https://www.djangoproject.com/

[7] 5 of the most popular python and django websites
Shuup
https://www.shuup.com/en/blog/25-of-the-most-popular-python-and-djan

[8] Flask web development, one drop at a time
2017 by Armin Ronacher
http://flask.pocoo.org/

[9] AngularJS Tutorial
Google 2010-2017
https://www.w3schools.com/angular/default.asp

[10] Installing node.js and updating npm
October,2016 https://docs.npmjs.com/getting-started/installing-node

[11] Detecting Inconsistencies in JavaScript MVC Applications
Frolin S.Ocariza, Karthik Pattabiraman & Ali Mesbah
Software Engineering,May 2015 http://ieeexplore.ieee.org/abstract/document/7194585/?section=abstract

[12] Título del trabajo de Héctor
Héctor Rodríguez Campo, Mayo 2017

[13] Simple JavaScript Object to XML string converter: x2js https://github.com/abdmob/x2js

[14] ng directives
Google 2010-2017 https://docs.angularjs.org/api/ng

[15] D3, Data-Driven Documents
Michael Bostock, Vadim Ogievetsky & Jeffrey Heer (2011)
IEEE Transactions on Visualization and Computer Graphics http://vis.stanford.edu/files/2011-D3-InfoVis.pdf

[16] Mike Bostock Oficial page
Mike Bostock October 5,2015 https://bost.ocks.org/mike/

[17] NvD3 oficial page http://nvd3.org/

[18] Charts.js oficial website http://www.chartjs.org/

[19] Zoom oficial website https://www.zoomdata.com/

[20] AngularJS Perfomance Tuning for Long Lists
Sebastian Fröstl
Setember 10, 2013 https://tech.small-improvements.com/2013/09/10/angularjs-performance-with-large-lists/

[21] Stackoverflow Misko Hevery angular performance http://stackoverflow.com/questions/9682092/databinding-in-angularjs/9693933#9693933

[22] Angular JS Animations
Google 2010-2017 https://docs.angularjs.org/guide/animations

[23] TypeScript oficial site
MicroSoft 2012-2017 `https://www.typescriptlang.org/`

[24] Chrome motor v8 oficial site
Chrome 2012-2017 `https://developers.google.com/v8/`

[25] Werkzeug oficial site `http://werkzeug.pocoo.org/`

[26] Apache oficial site `https://httpd.apache.org/`

[27] Nginx oficial site `https://www.nginx.com/resources/wiki/`

[28] BackBone oficial site `http://backbonejs.org/`

[29] Ember oficial site `https://www.emberjs.com/`