



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

**GRADO EN INGENIERÍA INFORMÁTICA
INGENIERÍA DEL SOFTWARE**

**DISEÑO E IMPLEMENTACIÓN DE UNA API REST
Y SU CONEXIÓN CON MONGODB**

Realizado por

JUAN MANUEL LÓPEZ PAZOS

Asignatura

COMPLEMENTOS DE BASES DE DATOS

Departamento

LENGUAJES Y SISTEMAS INFORMÁTICOS

Sevilla, a 22 de junio de 2014

Tabla de contenidos

1. INTRODUCCIÓN.....	5
2. JUSTIFICACIÓN.....	7
3. TECNOLOGÍAS.....	9
3.1 PLATAFORMA PARA EL DESPLIEGUE: HEROKU.....	11
3.2 BASE DE DATOS: MONGODB CON MONGO HQ.....	12
Introducción.....	12
Comparación entre MongoDB y MySQL.....	13
MongoDB en Heroku.....	16
3.3 LENGUAJE EN EL SERVIDOR: NODE.JS.....	17
Introducción a Node.js.....	17
Comparación entre Node.js y Java con GSON.....	18
3.4 TECNOLOGÍA EN EL CLIENTE: ANGULARJS Y JADE.....	22
Introducción.....	22
Comparación entre AngularJS y jQuery.....	22
Comparación entre JADE y HTML.....	27
4. API REST.....	30
5. APLICACIÓN WEB.....	32
6. CONCLUSIONES.....	35
7. WEBGRAFÍA.....	37

1. INTRODUCCIÓN

En este trabajo he desarrollado una aplicación web basada en una API REST, la cual accede a una base de datos no relacional, y envían al cliente los datos solicitados de forma muy rápida para mostrarlos de forma dinámica en el navegador.

Como base de datos se usará MongoDB, la cual está orientada a documentos; programaremos el servidor con el lenguaje Node.js y en la parte del cliente se usará AngularJS para consumir la API REST diseñada, aprovechando esta ocasión para introducir nuevas tecnologías que cada día son usadas con mayor frecuencia y mostrando una de las posibles aplicaciones de una base de datos no relacional en las aplicaciones web.

La aplicación consistirá en un sistema en el que los alumnos de un colegio puedan ver las excursiones programadas, apuntarse a las que quieran acudir y poder realizar comentarios, siempre y cuando ya se hayan realizado las mismas.

De esta forma, veremos cómo se realizan consultas sobre la base de datos y cómo se usa con el framework AngularJS para mostrarla en nuestra aplicación, de modo que simplificaremos nuestras plantillas HTML dotándolas de gran dinamismo.

2. JUSTIFICACIÓN

En la titulación que estoy cursando, el grado de ingeniería del software, se hace especial hincapié en las bases de datos relacionales, en las que los datos están conectados entre sí; pero no se han mencionado apenas las bases de datos no relacionales. Por este motivo, para realizar este trabajo he decidido usar una base de datos no relacional como es ***MongoDB***, la cual está orientada a documentos y que hemos estudiado en clase, aunque ya la conocía anteriormente al haberla usado en mi trabajo de fin de grado y en otras asignaturas.

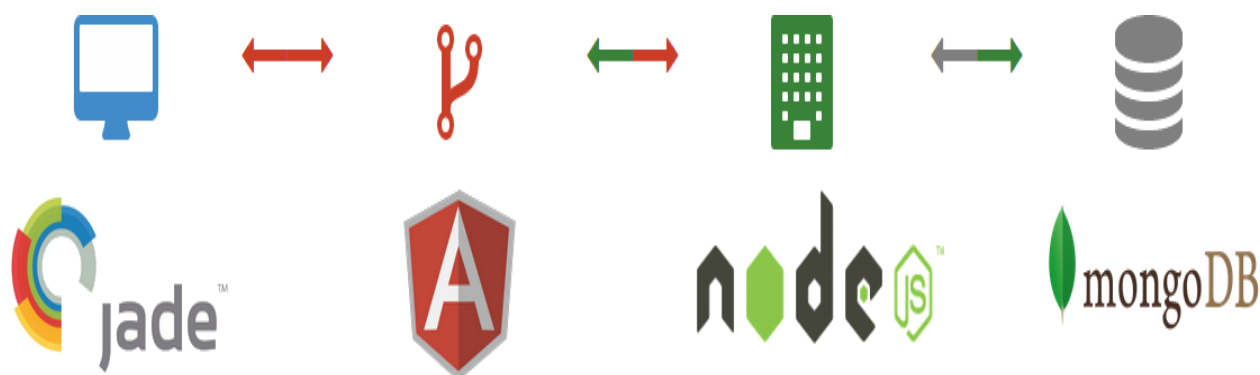
Así pues, ***MongoDB*** va a ser el núcleo de este trabajo al ser la base de datos que proporcionará la información, pero dentro de las tecnologías que voy a introducir, como ***Node.js***, ***JADE*** o ***AngularJS***, considero que ésta última es la más importante. Con este framework desarrollado por Google podremos realizar peticiones al servidor, obtener información y mostrarla en nuestra web con muy pocas líneas de código, ya que el nivel de abstracción es bastante superior si lo comparamos con otros frameworks de JavaScript; y otras muchas cosas que veremos más adelante.

El motivo por el que he seleccionado este conjunto de tecnologías se debe a que son las mismas que he utilizado para mi trabajo de fin de grado. Todas y cada una de ellas me resultan muy interesantes y aprovechables, y pienso que ésta era una buena ocasión para mostrarlas ante mis compañeros, además de servirme para realizar comparativas con otras tecnologías y, así, darme cuenta de las fortalezas y debilidades de estas herramientas.

Lo única tecnología que no utilizo en mi trabajo de fin de grado es ***JADE***. En este trabajo he usado ***JADE*** ya que he usado como base de mi aplicación web un proyecto publicado en <https://github.com>, el cual se puede consultar en la webgrafía. Por tanto, he pensado que estaría bien mantener esas plantillas y añadir las mías en ese lenguaje, con el único fin de aprender algo que no conocía de antes.

3. TECNOLOGÍAS

En esta sección hablaremos sobre la arquitectura de la aplicación web y de las tecnologías que se han usado en cada capa.



Empezaremos por la base de datos, luego hablaremos sobre el servidor, más tarde sobre **AngularJS**, el cual se encargará de gestionar toda la información que fluya por la aplicación; y para finalizar hablaremos de la parte estática de la aplicación web.

En nuestra base de datos **MongoDB** se encontrará toda la información necesaria, en este caso las excursiones y las cuentas de los usuarios. Dicha información será consultada desde el servidor, programado con **Node.js**, con la ayuda del driver oficial para **MongoDB**. Dicha implementación ya venía realizada en el proyecto que he tomado como base. Sólo he tenido que modificarla y ampliarla según las necesidades de mi aplicación. Además, la implementación de dicho proyecto se hace de otra forma a la que yo había seguido para mi trabajo de fin de grado, por lo que he comprobado el gran número de alternativas que tenemos con **Node.js**.

A continuación viene, en mi opinión, una de las partes más importantes de este proyecto, si no la más importante. **AngularJS**, un framework de JavaScript de código abierto, hace uso del **MVC** (Modelo Vista Controlador) y permite la inyección de dependencias al igual que Spring, un framework para Java que estudiamos los alumnos de mi titulación en el tercer curso en la asignatura **DP** (Diseño y Pruebas). De esta forma, **AngularJS** proporciona a las plantillas **HTML** la información necesaria en el momento de mostrarla, de modo que **AngularJS** hace de intermediario constantemente entre el navegador y el servidor con un gran dinamismo.

Complementos de Bases de Datos

Para finalizar tenemos las vistas, las cuales están diseñadas mediante plantillas en lenguaje ***JADE***, un motor de plantillas para ***HTML*** que permite reducir el número de líneas de código, facilitando su implementación, y reutilizar las plantillas, entre otras cosas.

3.1 Plataforma para el despliegue: Heroku



En esta plataforma tenemos un plan gratuito siempre y cuando no sobrepasemos lo que se llama “dyno hour”. Los dynos son contenedores alojados en Heroku, los cuales podemos configurar, para que nuestra explicación se ejecute correctamente. Dichos dynos constan de cpus y memoria, los cuales son factores que determinan el precio de las tarifas.

La principal ventaja es que Heroku proporciona a los usuarios 750 “dyno-hours” gratis por aplicación. Por tanto, alojar esta aplicación en Heroku no me va a suponer coste alguno, ya que si tengo sólo un dyno, estaría funcionando 24 horas al día durante, como máximo, 31 días por mes. En ese caso, habríamos consumido 744 “dyno-hours”, por lo que en ningún momento pasaremos el límite.

Heroku no sólo soporta **Node.js**, el lenguaje que ha sido utilizado en la aplicación web que he desarrollado, ya que también se puede implementar el servidor con lenguajes como **Python**, **PHP**, **Ruby** o **Java**, entre otros.

3.2 Base de datos: MongoDB con MongoHQ

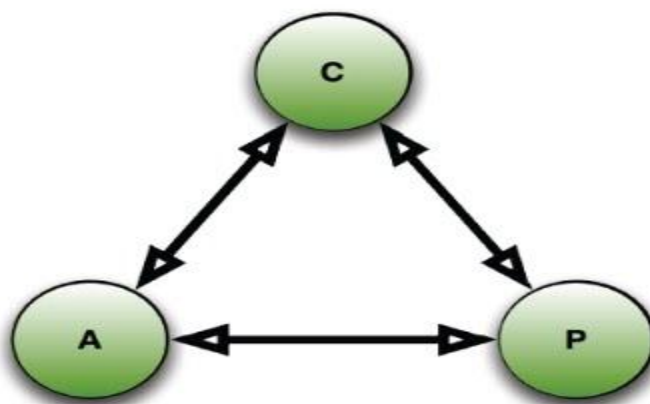


En este apartado se hablará de la base de datos estudiada y utilizada para este trabajo. También se explicará qué es *MongoHQ* y cómo interactúa con la aplicación web.

Introducción

MongoDB es una base de datos **NoSQL** orientada a documentos. Los datos se guardan en colecciones en formato **JSON** aunque internamente se almacenan en formato **BSON**, una variante de **JSON** con más detalles utilizada por *MongoDB*.

Está basada en el teorema **CAP**, según el cual se define que de las 3 propiedades (Consistencia, Disponibilidad y Tolerancia a fallos) sólo se pueden tener 2.



En el caso de *MongoDB* se dispone por defecto de Consistencia y Tolerancia a fallos, pero se puede cambiar la configuración del nivel de consistencia, de modo que podemos sacrificar consistencia para ganar disponibilidad.

Comparación entre MongoDB y MySQL

En este caso vamos a realizar una comparativa entre **MongoDB** y **MySQL**, de modo que podremos ver las principales diferencias que supondría realizar esta aplicación con una base de datos relacional.

	MongoDB	MySQL
Plataformas	Windows, Linux, OS X y Solaris	Windows, Linux, Mac, Solaris y muchos más
Implementación	C++	C y C++
Sistema de almacenamiento de datos	Sistema de archivos	Transaccionales y no transaccionales
Formato de los datos	Colecciones > Documentos > Campos	Tablas > Filas > Columnas
Replicación	Sí	Sí, pero con muchas limitaciones
Balanceo de carga	Escalación horizontal mediante shards	Mediante extensiones software (HAProxy) o hardware
Instalación	Fácil (mediante el terminal)	Dificultad media (mediante asistente de instalación)
Lenguajes soportados	C, C++, Java, PHP, Python, Node.js, Ruby, y muchos más	C, C++, Java, PHP, Ruby y algunos más
Búsquedas full-text	No soportadas (disponibles en versión beta)	Sí

MongoDB soporta diversas plataformas como Windows (a partir de Windows 7), Linux, Mac y Solaris, pero **MySQL** soporta muchísimas más plataformas.

Ambas bases de datos están implementadas en el lenguaje **C++**, pero **MySQL** también tiene algunas implementaciones realizadas en el lenguaje **C**.

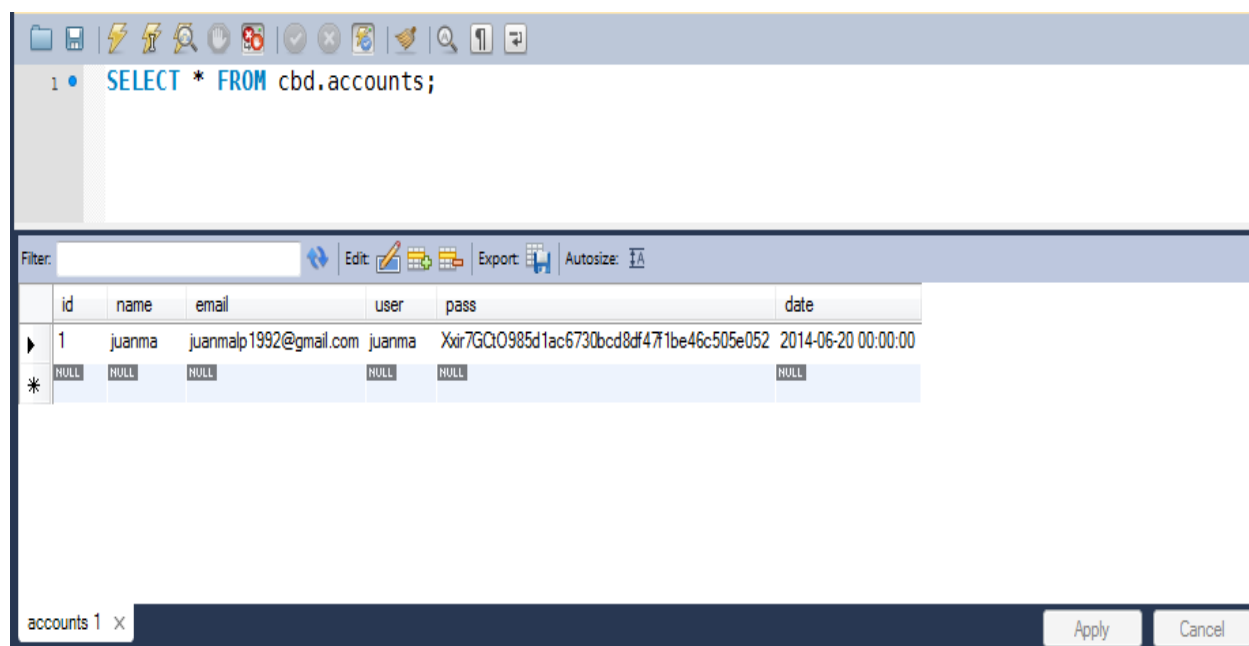
A la hora de clasificar estas bases de datos podemos decir que **MongoDB** es una base de datos basada en sistemas de archivos, mientras que **MySQL** puede estar basadas en procesos transaccionales o en procesos no transaccionales.

Complementos de Bases de Datos

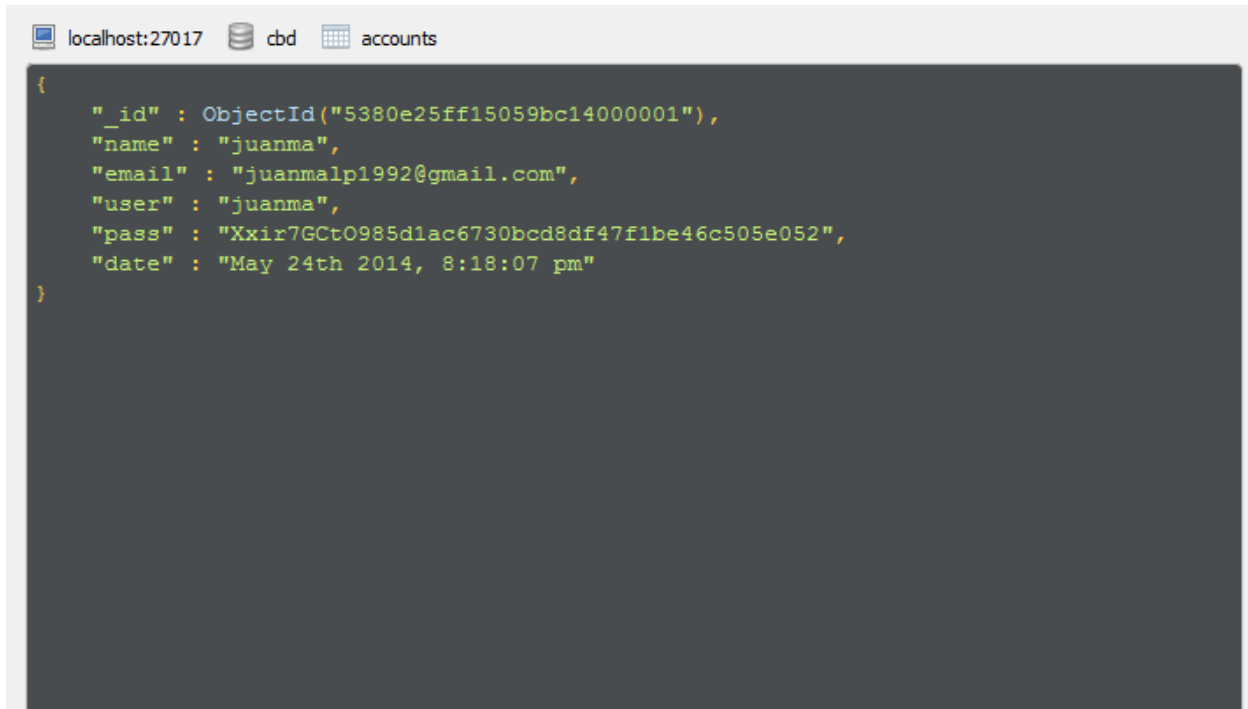
En caso de que usáramos **MySQL** con procesos transaccionales, en caso de conflicto en la base de datos se volvería a un estado estable anterior, lo que se denomina rollback. Sin embargo, eso no se puede realizar con **MongoDB**. No obstante, según la documentación oficial de **MongoDB**, es posible simular una transacción. Sin embargo, hay que realizar a mano una serie de colecciones y configurarlas de una forma determinada. A pesar de todo, pienso que esa solución no es demasiado técnica.

En **MongoDB** la información se almacena en ficheros, mientras que en **MySQL** se guarda en tablas. Podemos decir que una colección en **MongoDB** es equivalente a una tabla en **MySQL**, que un documento en **MongoDB** es equivalente a una fila en **MySQL** y que un campo de **MongoDB** es equivalente a una columna en **MySQL**.

En la siguiente imagen del workbench de **MySQL** podemos ver en la práctica cómo se almacena la información:



Sin embargo, en esta otra imagen de la interfaz de **Robomongo** (una herramienta para visualización y edición rápida para **MongoDB**) podemos ver cómo se almacena en **MongoDB**:

A screenshot of a MongoDB shell window. The title bar shows 'localhost:27017', a database icon, 'cbd', a collection icon, and 'accounts'. The main area displays a JSON document with the following fields: '_id' (ObjectId), 'name' (juanma), 'email' (juanmalp1992@gmail.com), 'user' (juanma), 'pass' (Xxir7GCtO985dlac6730bcd8df47f1be46c505e052), and 'date' (May 24th 2014, 8:18:07 pm).

```
{
  "_id" : ObjectId("5380e25ff15059bc14000001"),
  "name" : "juanma",
  "email" : "juanmalp1992@gmail.com",
  "user" : "juanma",
  "pass" : "Xxir7GCtO985dlac6730bcd8df47f1be46c505e052",
  "date" : "May 24th 2014, 8:18:07 pm"
}
```

La replicación se contempla en **MongoDB** como una relación Maestro-Esclavo. En **MySQL** también se puede replicar la información, pero es muy costoso debido a la forma en que se almacena y distribuye la información en los servidores.

MongoDB se puede escalar de forma horizontal, de modo que se puede repartir la información de un documento entre distintos servidores. Esto mismo no se puede hacer de forma tan sencilla con **MySQL**, ya que las limitaciones tecnológicas son grandes.

A la hora de instalar **MongoDB** no tenemos más que seguir la documentación oficial. La instalación se realiza mediante el terminal y en pocos pasos. En cuanto a **MySQL**, la instalación no llega a ser compleja ya que se realiza mediante un wizzard de instalación. Sin embargo, hay que realizar muchas configuraciones, por lo que para algún principiante puede convertirse en una tarea tediosa.

En cuanto a los lenguajes soportados, **MongoDB** cobra ventaja ya que consta de muchos drivers oficiales para los lenguajes mencionados en la tabla y otros como **Scala**, **JavaScript** o **Haskell**, entre otros. **MySQL** también soporta diversos lenguajes, pero en menor cantidad que **MongoDB**.

Otro aspecto que me ha parecido interesante analizar ha sido las búsquedas full-text. Como ya he comentado anteriormente, estoy utilizando **MongoDB** para mi trabajo de fin de grado y para otra asignatura, en concreto para **ISPP** (Ingeniería del Software y Práctica Profesional). En la aplicación web que he desarrollado para esa asignatura junto con otros compañeros necesitábamos realizar búsquedas full-text, pero nos dimos cuenta de que **MongoDB** no la soportaba. Sin embargo, esta característica sí la proporciona **MySQL**.

MongoDB en Heroku

Para subir nuestra base de datos a la nube usaremos **MongoHQ**, un servicio de base de datos para **MongoDB**. Desde **Heroku** podemos añadir un add-on de **MongoHQ** y conectarlo con nuestra aplicación para consultar los datos. Un add-on no es más que una extensión para nuestra aplicación, en este caso para la base de datos.



Sin embargo, **Heroku** nos provee de muchos otros add-ons para la base de datos. Si queremos usar **MongoDB** podemos usar **MongoLab**, el cual es muy similar a **MongoHQ**. Si por el contrario usamos una base de datos relacional, tenemos un add-on para **Postgres** o **MySQL**.

3.3 Lenguaje del servidor: Node.js



En esta sección se explicará todo lo relacionado con la programación del servidor, la cual se ha realizado con el lenguaje **Node.js**. Se explicarán brevemente las principales características y se realizará una comparación con Java haciendo uso de la librería **GSON**.

Introducción

Node.js es un lenguaje **JavaScript** asíncrono y basado en eventos para el lado del servidor. Se puede considerar muy reciente ya que se inventó a finales de 2009. **Node.js** está formado por módulos básicos para su funcionamiento, pero hay muchísimos otros oficiales o de terceros que no están incluidos. Estos módulos se pueden instalar mediante **NPM** (Node Package Manager), el gestor de paquetes para **Node.js**. A diferencia de otros lenguajes de servidor, utiliza un único hilo común para todas las peticiones. Esto implica mayor escalabilidad pero también que sólo se pueda utilizar una única CPU.

También podríamos haber escogido otra plataforma como **GAE** (Google App Engine) y haber desarrollado nuestra aplicación en **Java** con la ayuda de la librería **GSON**, algo bastante similar a lo que ya estudiamos en una asignatura de 2º curso. Esta opción también es gratuita para aplicaciones pequeñas.

Comparación entre Node.js y Java con GSON

A continuación vamos a realizar una tabla por cada lenguaje, de modo que analizaremos de forma individual sus ventajas e inconvenientes.

Node.js

Ventajas	Desventajas
Es un lenguaje JavaScript	La documentación oficial no está clara
Con pocas líneas se puede crear un servidor HTTP	Por sí sólo no es seguro. Necesita de librerías
Es capaz de manipular datos JSON sin conversiones intermedias	No existe un IDE específico
Existe una comunidad muy activa	Tiene las mismas limitaciones que JavaScript
Es fácil conectarse a una base de datos	No tiene frameworks que simplifiquen las implementaciones
Las librerías se instalan con un solo comando en el terminal	No hay un patrón común en la estructura de los proyectos

Node.js es un lenguaje **JavaScript** para servidor, por tanto podríamos desarrollar una aplicación sólo con **JavaScript** o casi al completo, como sucede con la que yo propongo.

Poner en funcionamiento un proyecto **Node.js** es muy sencillo y rápido. Se puede realizar un “¡Hola, mundo!” en pocos minutos.

Otro punto fuerte de **Node.js** es que es capaz de trabajar con archivos **JSON** directamente sin necesidad de realizar conversiones para tratar los datos.

También es reseñable la comunidad tan activa acerca de **Node.js**. En páginas como **GitHub** podemos encontrar una gran cantidad de proyectos muy bien documentados y muy útiles para aprender.

A la hora de conectar el servidor con una base de datos como **MongoDB** también tenemos muchas facilidades, ya que es una tarea muy sencilla y se puede

realizar de muchas formas. En el proyecto que he usado como base, la gestión de la conexión a la base de datos se realiza mediante el driver de **MongoDB**, pero también se podría haber realizado con el framework **Mongoose**.

No podemos olvidar la forma de instalar las librerías. Todas y cada una de las librerías se instalan mediante **NPM** (Node Package Manager), el gestor oficial de paquetes de **Noje.js**. Basta con escribir en el terminal “npm install” seguido del nombre exacto de la librería a instalar, y ésta quedará instalada en la carpeta “node_modules” de nuestro proyecto.

Sin embargo, también existen una serie de desventajas. La primera es que la documentación oficial de **Node.js** no está clasificada de una forma muy clara. No está pensada para que la lea un principiante en **Node.js**.

Otra desventaja es que **Node.js**, por sí sólo, no es seguro. Debido a que la implementación del servidor es totalmente manual, debemos cuidar cada uno de los detalles. Otra opción es usar alguna de las diversas librería que hay para proteger nuestras aplicaciones en **Node.js**. Sin embargo, es difícil conseguir proteger al máximo nuestra aplicación si no tenemos conocimientos previos.

También hay que mencionar que no existe un **IDE** específico para **Node.js**. En mi caso he utilizado el **IDE WebStorm**, el cual sirve para aplicaciones desarrolladas con **HTML** y **JavaScript** y, desde hace poco, cuenta con un intérprete estupendo para **AngularJS**. Sin embargo, no está pensado para **Node.js** ya que, a pesar de ser un lenguaje **JavaScript**, tiene muchas características y funcionalidades propias de **Node.js**.

Podemos deducir que si **Node.js** es un lenguaje **JavaScript** para servidor, tendrá prácticamente las mismas limitaciones que **JavaScript**. Por ejemplo, en JavaScript no existen clases como sí ocurre en Java.

Otro inconveniente muy destacable es que no existe ningún framework para **Node.js** que proporcione una plantilla ya implementada con las funciones básicas, como sí sucede con otros lenguajes como **Java**, que cuenta con el framework **Spring**.

Un último inconveniente es que no existe un patrón común en los proyectos **Node.js**. Si exploramos un poco los proyectos alojados en **GitHub**, veremos que en

Complementos de Bases de Datos

cada proyecto se clasifican los proyectos de una forma, los nombres de las carpetas cambian, etc. Por tanto, eso dificulta el aprendizaje de este lenguaje.

Pasemos ahora con **Java**.

Java + GSON

Ventajas	Desventajas
Es un lenguaje que hemos usado muchos años	Es necesario convertir los datos para tratarlos en el servidor
Existen frameworks con plantillas iniciales	Requiere de muchas configuraciones
Se puede conseguir una aplicación muy segura	Pueden producirse problemas de compatibilidades con las dependencias
Estructura común en todos los proyectos	A veces las trazas de errores no son muy precisas

En cuanto a las ventajas algunas son ya conocidas. En primer lugar es el lenguaje que más hemos utilizado los alumnos de la primera promoción del grado de ingeniería del software. Por lo tanto, es un lenguaje que conocemos todos los alumnos.

En segundo lugar hay que destacar que existen muchos frameworks que proporcionan plantillas iniciales para los proyectos **Java**. Como he mencionado anteriormente, **Spring** nos proporciona plantillas muy completas en las que sólo tenemos que implementar los aspectos específicos de nuestra aplicación.

También hay que destacar que se pueden proteger las aplicaciones desarrolladas en Java de muchas formas. Algunas son más fáciles que otras, pero en muchos frameworks para Java se facilita la seguridad de la aplicación web.

Como última ventaja destacable hay que decir que existe una estructura común en todos los proyectos **Java**. En todo proyecto **Java** se separan en paquetes el modelo, los controladores, los servicios, los recursos estáticos, etc.

La principal desventaja aplicable a la aplicación que he desarrollado es que, en caso de usar una base de datos como MongoDB, hay que convertir los archivos JSON

a objetos mediante la librería **GSON** para tratarlos en el servidor. La conversión no es compleja, ya que son apenas dos líneas de código. Sin embargo, se pierde el tiempo suficiente en la conversión como para que las características que ofrece *AngularJS* no se puedan explotar al máximo.

Otro inconveniente es que requiere de muchas configuraciones. Hay frameworks que vienen preconfigurados, pero según vamos añadiendo funcionalidades hay que configurar más ficheros, lo cual puede resultar una tarea muy tediosa.

También se suele dar un problema muy habitual, y es que algunas librerías pueden entrar en conflicto si no están instaladas las versiones adecuadas. Este problema puede llegar a ser difícil de solucionar, ya que muchas veces hay que desinstalar las librerías a mano, refrescar el proyecto, etc.

Para finalizar, el último inconveniente de los proyectos Java es que los logs que se muestran en el terminal no suelen ser muy precisos. En caso de que usemos un framework se muestran todas y cada una de las clases en las que han surgido conflictos, pero hay veces en las que no se indica con exactitud cuál puede ser la causa del fallo ocasionado.

3.4 Tecnología en el cliente: AngularJS y JADE



En el cliente usaremos el framework *AngularJS*, desarrollado por **Google** desde el 2009, con el que podremos obtener los datos de la *API REST* mediante peticiones **HTTP** y mostrarlos de forma muy dinámica en nuestras plantillas *HTML*, las cuales se generarán mediante *JADE*, un motor de plantillas **HTML**.

En cuanto a las alternativas se encuentran potentes frameworks o librerías JavaScript como **Ember**, **jQuery** o **Backbone**.

Comparación entre AngularJS y jQuery

Las principales ventajas de *AngularJS* son: la inyección de dependencia entre sus componentes (bastante similar al funcionamiento de Spring), la posibilidad de definir los controladores o servicios como módulos distintos o el alto nivel de abstracción frente a otros frameworks **JavaScript** basados en el modelo **MVC**.

Por tanto, para la aplicación que queremos desarrollar, considero *AngularJS* como la mejor opción teniendo en cuenta todas sus ventajas.

Todas las tecnologías que he seleccionado son muy recientes, de grandes prestaciones y se están haciendo un hueco en el mundo del software.

AngularJS

Ventajas	Inconvenientes
Basado en el MVC + Inyección de dependencia	La documentación no está pensada para principiantes
Conexión entre controlador y plantillas mediante directivas	Usado junto con jQuery puede provocar conflictos
Data-binding y evaluación de expresiones en plantillas HTML	Es difícil seguir la traza de una aplicación
Mayor nivel de abstracción	Hay muchas diferencias entre versiones
Ofrece funcionalidades que permiten diseñar una web estéticamente impecable	Angular incorpora una versión de Bootstrap internamente

Una de las principales ventajas de **AngularJS** es que está basado en el patrón **MVC** (Modelo-Vista-Controlador), de modo que las vistas son totalmente independientes de la aplicación y son utilizadas por el controlador asociado en el momento de mostrar dicha vista. Esto se consigue mediante inyección de dependencia, de forma muy similar a la utilizada en el framework para **Java Spring**.

Otra ventaja es la conexión entre las vistas y los controladores, lo cual se hace mediante directivas. Éstas están muy bien documentadas en la página web oficial de **AngularJS** y tienen una sintaxis muy parecida a **JavaScript** y **Java**.

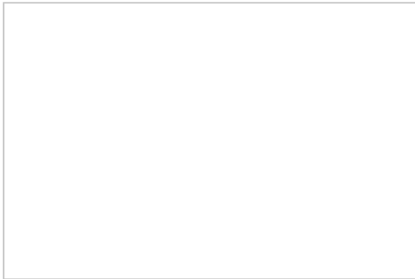
La siguiente ventaja va muy unida a la anterior. En las plantillas **HTML** podemos evaluar expresiones o mostrar datos después de iterar sobre una colección. Esto se puede hacer gracias a los data-bindings, que no son más que expresiones encerradas entre corchetes dobles (“{{” para abrir y “}}” para cerrar la expresión).

AngularJS posee un nivel de abstracción muy alto, ya que realiza tareas de muy bajo nivel de las que el programador no es plenamente consciente a la hora de desarrollar, por ejemplo: las peticiones **AJAX** al servidor.

Como última ventaja podemos destacar la estética que se puede llegar a conseguir en nuestra aplicación. Un ejemplo de ello es la directiva `ng-bind-template`.

Excursiones disponibles

```
{{ $index + 1 }}. {{trip.title}} ({{trip.moment |  
date:'dd/MM/yyyy'}})"
```



```
{{trip.description}}
```

Así se vería durante unos instantes la aplicación si tenemos una conexión a Internet lenta. Esto sucede porque los data-bindings están añadidos como “value” de las etiquetas. Sin embargo, si los añadimos dentro de la directiva ng-bind-template, los data-bindings no se ven y la información a mostrar se carga directamente quedando este resultado:

Excursiones disponibles

1. Excursión al parque del Alamillo (10/06/2014)



Si eres de fuera y no conoces el parque del Alamillo o eres de Sevilla y nunca has tenido la oportunidad de visitarlo esta es tu ocasión perfecta. Por sólo 5 € podrás explorar todo el parque con tus compañeros, jugar y disfrutar de uno de los mejores parques de Sevilla.

Dicho todo esto, pasemos con los inconvenientes. La documentación de **AngularJS** está muy bien detallada. Sin embargo, es fácil de leer cuando ya se tiene cierta experiencia. Mi forma de aprender **AngularJS** no fue leyendo la documentación, ya que hace 3 meses no entendía nada. Yo conseguí aprender explorando una gran cantidad de proyectos en **GitHub**, al igual que hice con **Node.js**.

Otro inconveniente es que usar **AngularJS** y **jQuery** simultáneamente puede provocar conflictos, ya que AngularJS usa una versión de **jQuery** internamente. Además de esto, los posibles fallos se deben a que en ciertas ocasiones los cambios que hagamos en el **DOM** mediante **jQuery** no son detectados por **AngularJS**. Esto se puede solucionar usando el método **\$apply** del **\$scope** que tengamos definido en el controlador, sin embargo hay que saber cómo funciona dicha función para evitar estos conflictos.

También hay que mencionar que no es fácil seguir la traza de la aplicación, ya que determinados fallos no son notificados en la consola de los navegadores. Por tanto, si cometemos algún fallo como no importar una librería o escribir incorrectamente el nombre de una variable podemos perder muchas horas hasta encontrar el fallo.

El siguiente inconveniente también es de suma importancia, y es que las versiones de **AngularJS** distan mucho entre sí, ya que cambian nombres de métodos, directivas, implementaciones, etc. Por tanto, hay que tener cuidado a la hora de elegir la versión de **AngularJS**, ya que puede ser muy determinante.

Lo último puede ser un inconveniente si no se es consciente de ello. **AngularJS** utiliza por defecto cierta versión de la librería **Bootstrap**, una librería para diseño y maquetación web muy famosa. El problema puede surgir si la versión que incorpora es antigua y queremos añadir una nueva. Hay que tener nuevamente mucho cuidado porque añadir nuevas funcionalidades de **Bootstrap** puede implicar que dejen de funcionar otras de la versión antigua.

A continuación vamos a realizar una comparativa similar para **jQuery**. Como **AngularJS** usa **jQuery** de fondo, las diferencias fundamentales van a radicar en el nivel de abstracción de una librería y otra.

jQuery

Ventajas	Inconvenientes
Selección de elementos DOM	Alta frecuencia de publicación de nuevas versiones
Manipulación de hojas de estilo	Librería de mucho tamaño
Peticiones AJAX al servidor	Ciertas funcionalidades son de muy bajo nivel
Efectos y animaciones	

Empezando por las ventajas, podemos decir que **jQuery** permite la selección de elementos DOM mediante id, class o name de las etiquetas.

También se pueden manipular las hojas de estilo de forma dinámica según el comportamiento de nuestra web. Yo no sabía que se podía realizar esto hasta que examiné el proyecto base que he usado para mi aplicación en la que se volvían a añadir ciertas líneas en el css aunque las hubiera comentado previamente.

Una de las características más importantes es la capacidad de realización de peticiones **AJAX** a un servidor. Estas mismas peticiones se realizan con **AngularJS**, sólo que a más alto nivel.

Y la característica que sea quizás más importe es el hecho de poder realizar efectos y animaciones. Hoy en día se pueden hacer auténticas florituras en una página web.

En cuanto a inconvenientes son pocos los que tiene **jQuery**. Uno de ellos es que se publican muchas versiones, algo similar a lo que sucede con **AngularJS**, y eso puede causar muchos despistes y conflictos.

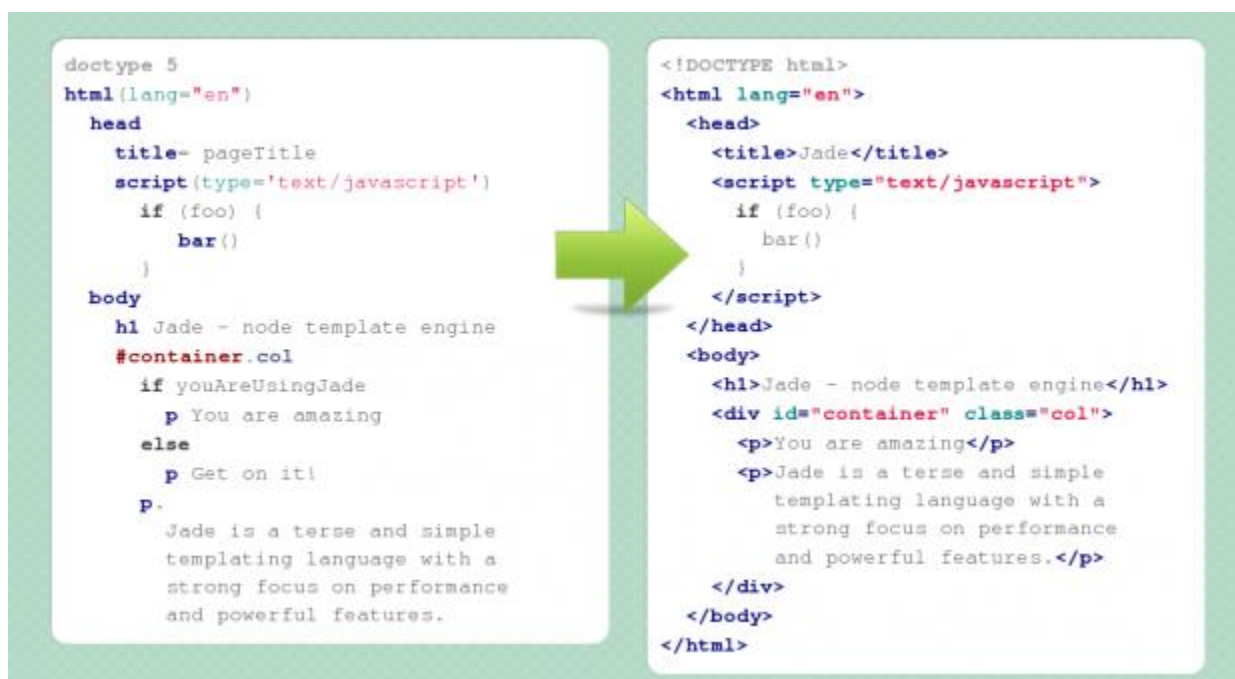
Un segundo inconveniente es el tamaño de la librería, ya que ocupa alrededor de unos 85 KB de carga. Por tanto, en ordenadores antiguos o con conexión a Internet lenta, es posible que las características asociadas a **jQuery** no funcionen del todo bien.

Y un último inconveniente según mi criterio, ya que para otra persona podría ser una ventaja, es que hay características que se usan a un nivel muy bajo. Por

ejemplo, las peticiones **AJAX** hay que realizarlas con todo detalle, mientras que en **AngularJS** se pueden realizar con una única línea gracias al servicio **\$http**.

Comparación entre JADE y HTML

Para comparar estos lenguajes vamos a utilizar dos imágenes, ya que las diferencias son pocas aunque muy significativas.



En **JADE** se programa mediante indentación, tal y como sucede en **Python**, ya que las etiquetas se abren pero no se cierran; los atributos como los id, class, etc se indican pegados al nombre de la etiqueta con el selector css correspondiente (“#” para los ids, “.” para los class, etc), los values se indican con un espacio respecto a la etiqueta y el resto de atributos se indican entre paréntesis y separados por “,”.

La principal ventaja de **JADE** sobre **HTML** es que se pueden reutilizar plantillas. Por ejemplo, si queremos usar una barra de navegación en nuestra aplicación web, en lugar de escribir el código en todas las plantillas, podemos

Complementos de Bases de Datos

implementar una plantilla que contenga sólo la barra de navegación e importarla en cada plantilla con la cláusula “extends”.

4. API REST

En primer lugar vamos a explicar qué es una **API**. Una **API** (Application Programming Interface) es un conjunto de funciones y procedimientos ofrecidos por una biblioteca.

Por otro lado tenemos el término **REST** (REpresentational State Transfer), un tipo de arquitectura de desarrollo web que se basa en el estándar **HTTP** y que está orientada a recursos. Por tanto, se implementarán los métodos **CRUD** (Create, Read, Update y Delete).

Así pues, si unimos ambos conceptos, podemos definir una **API REST** como una biblioteca de recursos de nuestro sistema y que están basados en el estándar **HTTP**. Cada recurso debe estar identificado por una url única. Además, ninguna url debe implicar ninguna acción, por lo que no debe contener un verbo en la estructura.

Método	Recurso	Descripción
GET	/api/trips	Devuelve todas las excursiones registradas en el sistema.
GET	/api/trips/[id]	Devuelve la excursión con el id = [id] en caso de que exista.
GET	/api/trips/[id]/users	Devuelve la lista de alumnos apuntados a la excursión con id = [id] en caso de que la excursión exista.
POST	/api/trips/[id]/users	El alumno logueado se apuntará a la excursión con el id proporcionado.
POST	/api/trips/[id]/comments	El alumno logueado comentará la excursión con el id proporcionado.

5. APLICACIÓN WEB

La aplicación web se puede probar en la siguiente dirección: <http://cbd.herokuapp.com> (la primera vez que se accede tarda un poco debido a que se usa el plan gratuito de **Heroku**). El código de dicha aplicación se encuentra en el enlace: <https://github.com/jualoppaz/cbd>.

En esta sección se indicarán las partes fundamentales de cada tecnología.

NodeJS

De Node.js me gustaría destacar el archivo “router.js”. El archivo principal de la aplicación y el que se ejecuta en el servidor es el “app.js”. Sin embargo, en el archivo “router.js” es donde se establecen todas las rutas de nuestra aplicación.

```
app.get('/index', function(req, res) {  
  if (req.session.user == null){  
    // if user is not logged-in redirect back to login page //  
    res.redirect('/');  
  } else{  
    res.render('index', {  
      title : 'CBD',  
      countries : CT,  
      udata : req.session.user  
    });  
  }  
});
```

Éste es el código que carga la vista inicial de la página, donde se muestran las excursiones. Sin embargo, aquí sólo se carga la parte estática. La información no viaja en esta petición.

```
app.get('/api/trips', function(req, res) {  
  DBM.findAllTrips(function(err, excursiones){  
    if(err) {  
      res.send(err);  
    }else{  
      res.json(excursiones);  
    }  
  });  
});
```


En este método sí se envía la información asociada a las excursiones existentes en la base de datos. Como vemos, se envía la respuesta tal como la envía la base de datos: en formato **JSON**.

AngularJS

De AngularJS me gustaría destacar un controlador. Para seguir el ejemplo mostraré el controlador de la vista “index”, en la que se listan todas las excursiones:

```
var angularIndex = angular.module('angularIndex', []);

function indexController($scope, $http) {
    $scope.trips = {};
    $scope.username = "";

    // Cuando se cargue la página, pide del API todas las excursiones
    $http.get('/api/trips')
        .success(function(data) {
            $scope.trips = data;
            console.log(data);
        })
        .error(function(data) {
            console.log('Error: ' + data);
        });

    $http.get('/api/user')
        .success(function(data) {
            $scope.username = data;
        })
        .error(function(data) {
        });
}
```

Una vez que se ha cargado la parte estática de la página (primera imagen de **Node.js**) se ejecuta el controlador y se consultan todas las excursiones existentes (segunda imagen de **Node.js**).

En caso de que todo haya ido bien, se almacenan los datos en la variable “trips”, sobre la que se iterará en la vista.

JADE

```
div.panel.panel-body.terms-and-conditions.col-md-8.col-md-offset-2
  h1(style="text-align: center").red Excursiones disponibles
div.panel.panel-body.terms-and-conditions.col-md-8.col-md-offset-2(ng-repeat="trip in trips", style="text-align:center")
  div(style="text-align:left")
    h1(style="margin-bottom:30px").green
    a(ng-href="/trips/{{trip._id}}", ng-bind-template="{{index + 1}}. {{trip.title}} ({{trip.moment | date:'dd/MM/yyyy'}})")
  div(style="float:left; margin-right: 20px")
    img(ng-src="/img/{{trip.image}}", height="200", width="300", style="margin-left: 10px; margin-bottom: 10px")
  div(style="margin-left: 10px;")
    h3(style="text-align:justify; margin-top: 0px", ng-bind-template="{{trip.description}}").blue
```

Como podemos observar, se hace uso de la variable `trips` definida en el controlador, se itera sobre ella, y para cada excursión se crea una tarjeta en la que se muestran el título, la imagen y la descripción.

6. CONCLUSIONES

Después de realizar este trabajo sólo puedo destacar cosas positivas. Realizar este trabajo me ha ayudado de cara a mi trabajo de fin de grado, ya que muchas implementaciones me pueden ser de utilidad.

También he aprendido una serie de cosas sobre *AngularJS* y *Node.js* que desconocía. Por ejemplo, de *AngularJS* no conocía la directiva ng-bind-template, la cual es muy útil para aumentar la estética de nuestra aplicación; y de *Node.js* no sabía cómo se usaba la variable session. Era algo que necesitaba para mi trabajo de fin de grado, y haciendo este trabajo lo he descubierto.

Y uno de los aspectos más fundamentales de este trabajo es que el día de la presentación mostraré ante mis compañeros una serie de tecnologías que nadie conoce, o que ha oído hablar de ellas pero nunca las ha usado.

Sólo espero que este trabajo sirva para los profesores de la asignatura y para los compañeros que lo soliciten. Todo sea por aprender.

7. WEBGRAFÍA

En esta sección se detallan todos y cada uno de los enlaces consultados tanto para la realización de este trabajo como para el desarrollo de la aplicación web. Para mayor comprensión, se indican los enlaces clasificados por tecnologías.

MongoDB

- <http://docs.mongodb.org/>
- <http://es.wikipedia.org/wiki/MongoDB>
- <http://www.genbetadev.com/bases-de-datos/mongodb-que-es-como-funciona-y-cuando-podemos-usarlo-o-no>.
- <http://planetubuntu.es/post/mongodb-lider-de-las-alternativas-a-las-tradicionales-bases-de-datos-sqls>.
- <http://www.genbetadev.com/bases-de-datos/nosql-clasificacion-de-las-bases-de-datos-segun-el-teorema-cap>

MySQL

- <http://dev.mysql.com/doc/refman/5.0/es/features.html>.
- <http://es.wikipedia.org/wiki/MySQL>.

Node.js

- <https://github.com/braitsch/node-login> (**Proyecto base utilizado**)
- <http://nodejs.org/api/>.
- <http://www.rmuno.net/introduccion-a-node-js.html>
- <http://blog.beeva.com/las-ventajas-de-apostar-por-node-js/>
- <http://blog.cballesterosvelasco.es/2012/03/presento-nodenet-alternativa-real.html>

AngularJS

- <https://docs.angularjs.org>
- <http://tutorials.jenkov.com/angularjs/ajax.html>

jQuery

- <http://es.wikipedia.org/wiki/JQuery>.
- <http://blog.capacityacademy.com/2013/03/16/jquery-que-es-origenes-ventajas-desventajas/>.
- <http://www.etnassoft.com/2011/03/28/alternativas-a-jquery/>
- <http://api.jquery.com/jquery.ajax/>

Jade

- <http://jade-lang.com/>