UNIVERSIDAD DEL BÍO-BÍO

FACULTAD DE CIENCIAS EMPRESARIALES

Departamento de Ciencias de la Computación y Tecnologías de la Información

ESTUDIO DEL FRAMEWORK ANGULARJS Y SU USO EN EL DESARROLLO WEB

Memoria para optar al título de Ingeniero Civil en Informática

AUTOR:

CAMILO ANDRES STUARDO MÜLCHI

Profesor Guía: Rodrigo Ariel Torres Avilés

Chillán, Diciembre de 2016

AGRADECIMIENTOS

A mis padres

Por siempre apoyarme y motivarme en este proceso, por la paciencia y cariño que tuvieron, ya que no fue corto.

A mis compañeros de Generación

Por su amistad y compartir tantas experiencias conmigo, en especial a German, Guillermo, Francisca, Christian, Naaman y Alejandra. Gracias a todos.

Al profesor Luis Gajardo

Por ayudarme en mi primera memoria, la cual no cumplió con los objetivos, pero a pesar de mis reiterados errores siempre estuvo ahí para apoyarme.

Al profesor Rodrigo Torres

Por ayudarme en esta memoria, y estar presente cada vez que necesitaba de su ayuda.

Camilo Andres Stuardo Mülchi

RESUMEN

Actualmente, los framework basados en JavaScript se usan en el desarrollo de la mayoría de las aplicaciones web y siempre están apareciendo herramientas novedosas que buscan liderar el mercado. Además, con la contaste evolución de las tecnologías web, se pueden materializar ideas que años atrás sería impensable realizar, debido a que no existía los recursos para construirlas.

El objetivo de este proyecto es el estudio del framework AngularJS uno de los más usados del mercado. Se realiza un análisis comparativo que permite dar a conocer al lector las características que proporciona AngularJS y que lo ha llevado a ser uno de los líderes del mercado. Es un estudio que, además, incluye el desarrollo de un caso práctico para mostrar el funcionamiento de esta herramienta.

Para realizar el estudio, se ha decidido analizar tres frameworks basados en JavaScript: AngularJS, BackboneJS, Ember. Se detallan las características de cada una, su arquitectura y tipos de aplicaciones que se puede realizar con ellas.

El estudio determinó que AngularJS es la más completa en comparación a BackboneJS y Ember, debido a que presenta más y/o mejores funcionalidades con respecto a las otras herramientas. Además, el estudio permitió observar todas las principales características que brinda AngularJS al mundo de las aplicaciones web.

SUMMARY

Currently, JavaScript-based frameworks are used in the development of most web applications and new tools are always emerging that seek to lead the market. In addition, with the evolution of the web technologies, you can materialize ideas that years ago would be unthinkable to do, because there were no resources to build them.

The objective of this project is to study the AngularJS framework one of the most used on the market. A comparative analysis is carried out that allows the reader to know the characteristics that AngularJS provides and that has led him to be one of the leaders in the market. It is a study that, in addition, includes the development of a practical case to show the operation of this tool.

To perform the study, it has been decided to analyze three frameworks based on JavaScript: AngularJS, BackboneJS, Ember. They detail the characteristics of each one, its architecture and types of applications that can be realized with them.

The study determined that AngularJS is the most complete in comparison to BackboneJS and Ember, because it presents more and/or better functionalities with respect to the other tools. In addition, the study allowed to observe all the main features that AngularJS provides to the world of web applications.

INDICE GENERAL

INDICE DE TABLAS

INDICE DE IMÁGENES

1. Introducción

1.1. Introducción General

Este proyecto tiene como propósito obtener el título profesional de ingeniero civil en informática, demostrando que el autor tiene las aptitudes y conocimientos necesarios para optar a dicho título a través de la investigación llamada "ESTUDIO DEL FRAMEWORK ANGULARJS Y SU USO EN EL DESARROLLO WEB", la cual consiste aprender el funcionamiento y las ventajas de uno de los framework más usados del mercado para luego compararla con diferentes herramientas de desarrollo de este tipo de tecnología en plataforma web para posteriormente implementar un prototipo. AngularJS es un framework tipo cliente MVC, el proyecto fue puesto en marcha en 2009 y es patrocinado por Google desde entonces, en esencia esta tecnología entrega al desarrollador muchas herramientas que facilitan y disminuyen la codificación de las aplicaciones web.

La investigación se centra en describir las características de las herramientas de desarrollo con AngularJS que existen actualmente, además de indicar los requerimientos técnicos necesarios para utilizar las herramientas estudiadas.

Para realizar la comparación entre las herramientas se seleccionan distintos parámetros que fueron escogidos durante el proceso de investigación y se justifica el "¿por qué?" se eligió esa medida.

Al final de la investigación se presentan las conclusiones extraídas del estudio y se sugiere al lector qué herramienta utilizar en las diferentes situaciones que se puedan presentar al momento de implementación de una aplicación con AngularJS.

Finalmente, se desarrolla un prototipo con AngularJS, de manera de proporcionar una guía de desarrollo para los futuros programadores. El prototipo es una aplicación web que permite a los alumnos inscribirse en los talleres extra programáticos de la carrera (Robótica y Videojuegos) y posteriormente ver noticias y programaciones de estos.

Esta memoria consta de siete capítulos, en el primero se especifican los objetivos de este informe, además de mencionar la metodología usada en el estudio comparativo y por último definir las limitaciones del éste.

El segundo capítulo consta con el marco teórico de la tecnología y una explicación de cómo funciona cada atributo de ésta para una mejor compresión del lector.

El tercer capítulo corresponde al estudio comparativo entre AngularJS y otros dos framework de características similares.

En el capítulo cuatro se muestran los resultados del caso práctico desarrollado con AngularJS.

El capítulo cinco habla de las conclusiones del estudio.

El capítulo seis presenta una serie de recomendaciones a la hora de elegir una herramienta de este tipo para desarrollar una aplicación web. Y por último el capítulo siete se encuentra la bibliografía usada para llevar a cabo el estudio comparativo e implementación del prototipo.

1.2. Especificaciones del estudio.

Primero se investigan todas las funcionalidades de AngularJS y cómo el framework hace uso de estas.

Al momento de tener un entendimiento acabado del framework y su funcionamiento, se pasan a reconocer las características que lo potencian y diferencian del resto de los frameworks del mercado.

Se estudian las características principales de AngularJS y se identifican en qué afectan directamente a las aplicaciones web en términos de: rendimiento, tiempos de carga, experiencia de usuario, agilidad de desarrollo, etc…

Por último, con todas las características estudiadas, se da paso a una comparación en los diferentes tópicos estudiados con respecto a los frameworks más usados del mercado.

1.3. Especificaciones del prototipo.

El prototipo es una aplicación web, muy parecida en funcionalidad a WordPress, la cual tiene un administrador(es) que puede publicar noticias, eventos y les permite a los alumnos que pertenezcan a estos talleres, publicar noticias, comentarlas y evaluarlas para mantener una constante comunicación entre ellos.

1.4. Objetivos.

a) Generales:

Estudiar las principales capacidades del framework AngularJS, para luego compararlo con otros dos frameworks de características similares.

b) Específicos:

- Descubrir las ventajas del desarrollo web con AngularJS y qué lo diferencia del resto de los frameworks del mercado.
- Dar a conocer una nueva tecnología poco usada en el ámbito nacional, pero de gran uso en el mundo.
- Mostrar una forma opcional de generar aplicaciones web, la cual se centra en el front-end developer.
- Descubrir si la tecnología puede usarse en temas ajenos al ámbito web.
- Brindar a través de un prototipo una página a los alumnos que participen en los talleres extra programáticos de la carrera.

1.5. Metodología

La metodología a utilizar en este proyecto es la investigación experimental sin hipótesis previa, la cual se emplea en aquellos casos en donde la indagación tiene por objetivo dar a conocer tecnologías nuevas y que puede ser un importante avance en el área. Constituye un método de investigación muy importante, tanto por su amplitud como por su utilización en el campo tecnológico. (Sampieri, 1998)

Las etapas de la metodología elegida son:

- **Delimitar y definir el objeto de la investigación:** Consiste en determinar claramente los objetivos de la investigación y, además, decidir el número de herramientas a estudiar.
- **Elaborar el diseño experimental:** Ya conocido los objetos de estudios, la precisión deseada y el equipo adecuado, se debe analizar si el estudio va a ser la interpretación de una gráfica, un valor o una relación empírica; esto señala el procedimiento experimental, es decir ¿cómo medir?, ¿en qué orden?, y ¿qué precauciones tomar al hacerlo?

- **Realizar el experimento:** Consiste en realizar prototipos con la tecnología y observar el comportamiento de ellos según los parámetros establecidos en el punto anterior.
- **El análisis o interpretación de resultados:** Ya sean valores, gráficas, tabulaciones, etcétera, debe contestar lo más claramente posible la o las preguntas planteadas por la investigación discriminando entre los modelos y los resultados deben permitir hacer la discriminación en forma tajante y proporcionar los motivos para aceptar uno y rechazar otro.
- **Obtener conclusiones:** Ya logrados los resultados del experimento el investigador debe aplicar su criterio científico para recomendar qué modelo ocupar o descartar en las situaciones vividas a lo largo del estudio.
  (Sampieri, 1998)

1.6. Alcances y Limitaciones

Los alcances que tiene esta memoria se centran en el uso del framework AngularJS y sus aplicaciones en el ámbito web.

Angular consta de variados servicios adicionales creados por la comunidad para ser usado en conjunto con el framework, estas extensiones solo se nombran en el caso de ser usadas, pero no son descritas en profundidad ya que el objetivo del estudio es evaluar el framework en su forma nativa, con las especificaciones básicas.

AngularJS permite el uso integrado de Testing, este se nombra en el estudio y se describen las distintas herramientas que hacen posible el trabajo con este, pero no se profundiza más allá en el tema.

1.7. Definiciones y abreviaturas

- **API REST:** Es una API, o librería de funciones, a la que se accede por el protocolo HTTP. Una REST API, por tanto, se accede a través de direcciones web o URLs en las que enviamos los datos de nuestra consulta. Como respuesta a la consulta sobre el REST API se obtienen datos en diferentes formatos, como pueden ser texto plano, XML, JSON, etc. (Marques, 2013)
- **Aplicación web:** En la ingeniería de software se denomina aplicación web a aquellas herramientas que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una intranet mediante un navegador. (PortalProgramas, 2015)

- **Backend:** El backend es aquel que se encuentra del lado del servidor, es decir, esta persona se encarga de lenguajes como PHP, Python, .Net, Java, etc., es aquel que se encarga de interactuar con bases de datos, verificar manejos de sesiones de usuarios, montar la página en un servidor, y desde este "servir" todas las vistas que el Frontend crea, es decir, uno como backend se encarga más que nada de la manipulación de los datos. (Alvarado, 2014)
- **Bootstrap:** es un framework originalmente creado por Twitter, que permite crear interfaces web con CSS y JavaScript, cuya particularidad es la de adaptar la interfaz del sitio web al tamaño del dispositivo en que se visualice. (Solis, 2014)
- **Compilador:** Un compilador es un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje diferente. Usualmente el segundo lenguaje es lenguaje de máquina, pero también puede ser un código intermedio (bytecode), o simplemente texto. Este proceso de traducción se conoce como compilación. (SISDOMA, 2011)
- **CSS:** Hojas de Estilo en Cascada (Cascading Style Sheets) es el lenguaje utilizado para describir la presentación de documentos HTML o XML, esto incluye varios lenguajes basados en XML como son XHTML o SVG. (Mozilla, 2016)
- **DNS:** El sistema de nombres de dominio (DNS, por sus siglas en inglés, Domain Name System) es un sistema de nomenclatura jerárquico descentralizado para dispositivos conectados a redes IP como Internet o una red privada. Este sistema asocia información variada con nombre de dominio asignado a cada uno de los participantes. (Lanzarote, 2015)
- **DOM:** Document Object Model o DOM ('Modelo de Objetos del Documento' o 'Modelo en Objetos para la Representación de Documentos') es esencialmente una interfaz de plataforma que proporciona un conjunto estándar de objetos para representar documentos HTML, XHTML y XML,1 un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos. A través del DOM, los programas pueden acceder y modificar el contenido, estructura y estilo de los documentos HTML y XML, que es para lo que se diseñó principalmente. (Le Hégaret, 2005)

- **Frontend:** El frontend son todas aquellas tecnologías que corren del lado del cliente, es decir, todas aquellas tecnologías que corren del lado del navegador web, generalizándose más que nada en tres lenguajes, HTML, CSS Y JavaScript. (Alvarado, 2014)
- **HTML:** sigla en inglés de HyperText Markup Language (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web. (Mozilla, 2016)
- **HTTP XML:** XMLHttpRequest (XHR), también referida como XMLHTTP (Extensible Markup Language / Hypertext Transfer Protocol), es una interfaz empleada para realizar peticiones HTTP y HTTPS a servidores Web. Para los datos transferidos se usa cualquier codificación basada en texto, incluyendo: texto plano, XML, JSON, HTML y codificaciones particulares específicas. La interfaz se implementa como una clase de la que una aplicación cliente puede generar tantas instancias como necesite para manejar el diálogo con el servidor. (Microsoft, 2015)
- **JQueryUI:** jQuery UI es una biblioteca de componentes para el framework jQuery que le añaden un conjunto de plug-ins, widgets y efectos visuales para la creación de aplicaciones web. Cada componente o módulo se desarrolla de acuerdo a la filosofía de jQuery5 (find something, manipulate it: encuentra algo, manipúlalo). (JQueryUI, 2016)
- **RDBMS:** Un sistema de gestión de bases de datos relacionales (RDBMS) es un programa que te permite crear, actualizar y administrar una base de datos relacional. La mayoría de los RDBMS comerciales utilizan el lenguaje de consultas estructuradas (SQL) para acceder a la base de datos, aunque SQL fue inventado después del desarrollo del modelo relacional y no es necesario para su uso. (Rouse, 2015)
- **SPA:** Un single-page application (SPA), o aplicación de página única es una aplicación web o es un sitio web que cabe en una sola página con el propósito de dar una experiencia más fluida a los usuarios como una aplicación de escritorio. (CodeSchool, 2014)
- **WordPress:** WordPress es un sistema de gestión de contenidos o CMS (por sus siglas en inglés, Content Management System) enfocado a la creación de cualquier tipo de sitio. Originalmente alcanzó una gran relevancia usado para la creación de blogs, para convertirse con el tiempo en una de las principales herramientas para la creación de páginas web comerciales. (Tagle, 2014)

2. Marco Teórico

En este capítulo se presentan las tecnologías predominantes a la hora de desarrollar una aplicación web, más en específico las que hace uso el framework AngularJS, para tener una idea global de la estructura y metodologías que esta ocupa.

2.1. Tecnología JavaScript

2.1.1. Introducción

JavaScript es el lenguaje de programación interpretado más usado actualmente en el desarrollo de aplicaciones Web, contiene una sintaxis muy similar a Java y a C, pero no guarda relación directa con ninguno de estos lenguajes de programación, puesto que no es un lenguaje orientado a objetos, ya que está basado en prototipos, ósea las clases nuevas se generan clonando las clases base y de esta forma se extiende su funcionalidad. (Flanagan, 2007).

Al ser un lenguaje de programación interpretado no necesita que los programas sean compilados para que se ejecuten ya que el lenguaje funciona del lado del cliente, cualquier código escrito en JavaScript es directamente trasparente y visible por cualquier navegador web, puesto que él es el encargado de interpretar dicho código sin la necesidad de procesos intermedios. (Pérez, 2007).

2.1.2. Como identificar JavaScript dentro de las aplicaciones web.

El código JavaScript se puede encontrar dentro de las etiquetas <body></body> en las páginas web.

Por lo general se insertan entre: <script></script>.

También pueden estar ubicados en ficheros externos como se ve en la figura 1:

```
1
2  <script type="text/javascript" src="mificheroexterno.js"></script>
3
```

Figura.1. Ejemplo uso de ficheros externos en JavaScript.

Fuente: Stuardo, 2016.

Entre algunos de los servicios que se encuentran realizados con JavaScript en Internet se encuentran:

- Correo
- Chat
- Buscadores de Información
- Sitos de consulta
- Páginas de comunicación
- Tiendas virtuales.

Es decir que se puede encontrar código JavaScript en la gran mayoría de servicios y aplicaciones web publicado en internet o en redes privadas.

2.1.3. Ventajas

Entre las ventajas más destacables podemos mencionar las siguientes:

- Útil para el desarrollo de páginas web dinámicas.
- Excelente intermediario para la validación de datos de un formulario en el lado del cliente.
- Es un lenguaje sencillo y liviano.
- Consume poca memoria.
- Tiene gran cantidad de efectos visuales.
- Compatible con la gran mayoría de navegadores modernos y fácil manejo de datos.
- Se recomienda para la creación de aplicaciones web.
- El código se ejecuta en el lado del cliente, por lo que el servidor no es solicitado más de lo debido.
- No necesita de un compilador para ejecutarse, sino es el navegador el que interpreta el código.

2.1.4. Desventajas

Una de las desventajas más notable de JavaScript es la seguridad, el código es visible y puede ser leído por cualquiera. Es verdad que hoy en día existen varios estándares de seguridad que restringen la ejecución de código por parte de los navegadores, pero aun así se puede ejecutar código malicioso que dañe, robe o destruya información del lado del cliente.

Los <script> tienen capacidades limitadas, por razones de seguridad, por lo cual no es posible hacer todo con JavaScript, sino que es necesario usarlo conjuntamente con otros lenguajes evolucionados, posiblemente más seguros, como Java. Existe un determinado número de etiquetas, por lo que no se pueden crear más.

2.2. Arquitectura MVC

2.2.1. Patrón de arquitectura Modelo Vista Controlador (MVC)

Es un patrón de arquitectura de las aplicaciones software, separa la lógica de negocio de la interfaz de usuario:

- Facilita la evolución por separado de ambos aspectos
- Incrementa reutilización y flexibilidad

Modelo-Vista-Controlador

- Un modelo
- Varias vistas
- Varios controladores

Las vistas y los controladores suelen estar muy relacionados, los controladores tratan los eventos que se producen en la interfaz gráfica (vista).

Esta separación de aspectos de una aplicación da mucha flexibilidad al desarrollador.

El patrón de arquitectura "modelo vista controlador", es una filosofía de diseño de aplicaciones compuesta por tres capas:

2.2.2. Capa Modelo

Esta capa del modelo define la lógica de negocio (la base de datos pertenece a esta capa). En esta sección estará la estructura de datos, junto con las clases relacionadas a la base de datos y los métodos respectivos.

2.2.3. Capa Vista

La vista es lo que se muestra al usuario para que pueda interactuar con aplicación (A esta capa pertenecen los templates o plantillas). Proporcionará al cliente múltiples páginas web dinámicas, visualizándose para él como simples páginas HTML. Esto permite que personas con conocimientos de HTML puedan desarrollar sus aplicaciones de forma sencilla.

2.2.4. Capa Controlador

El controlador es un bloque de código que realiza llamadas al modelo para obtener los datos y se los pasa a la vista para que los muestre al usuario. (Pressman, 2005)

En la figura 2, se representa la arquitectura MVC gráficamente.

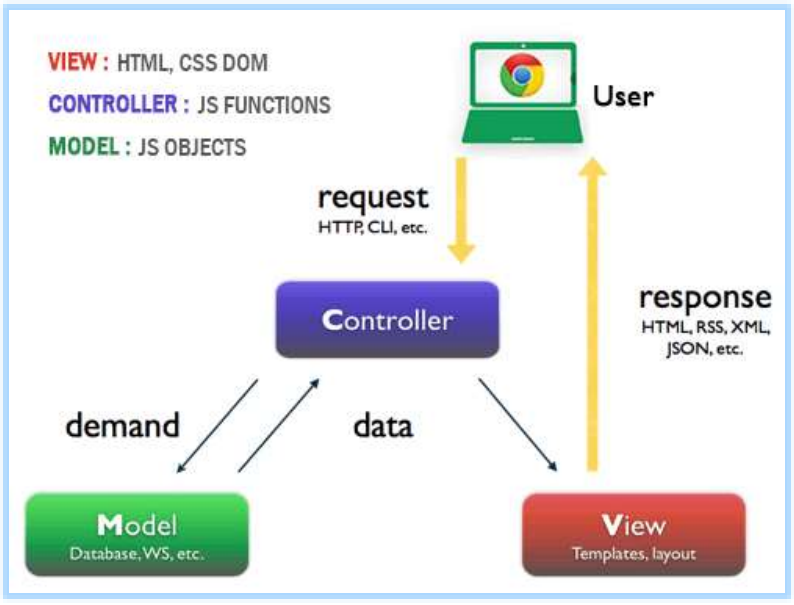


Figura.2. Arquitectura MVC.

Fuente: http://ustutorials.com/angularjs-tutorials/angularjs-mvc.php, 2014.

2.3. Framework AngularJS

2.3.1. Introducción

Diseñado para aplicaciones web dinámicas, mantenido por Google, utiliza HTML como lenguaje de plantillas el que extiende para expresar los componentes de su aplicación, luego AngularJS enseña al navegador la nueva sintaxis a lo que llamamos directivas.

Los conceptos de enlace de datos e inyección de dependencias permiten reducir líneas de código y su ejecución es realizada dentro del navegador.

Angular es 100% JavaScript del lado del cliente, no es un simple sistema de plantillas, la razón se debe a "bidirectional data binding", enlace de datos bidireccional, que posee y se ejecuta de forma automática. La plantilla se compila en el navegador y el paso de compilación da como resultado una vista en vivo. Ya no se necesita que los desarrolladores deban sincronizar constantemente la vista con el modelo o viceversa.

Además, tampoco es necesario dejar de lado jQueryUI o Bootstrap. Angular trabaja muy bien con librerías de componentes de terceros. Ayuda a la gestión de lo que se conoce como aplicaciones de una sola página (SPA). (docs.angularjs.org, 2016, https://docs.angularjs.org).

Angular permite el uso de cualquier tipo de backend, desde Java, Python, Ruby, C# entre otros.

Al trabajar con Angular es necesario utilizar una forma de comunicación de ida y vuelta con el servidor, peticiones HTTP XML o JSON.

2.3.2. Cronología de AngularJS

AngularJS nació en 2009 como parte de un producto comercial más grande, llamado GetAngular. Poco después, Misko Hevery, uno de los ingenieros que fundaron GetAngular, en 3 semanas logro regresar una aplicación web que constaba de 17mil líneas de código utilizando GetAngular, algo que normalmente se tarda 6 meses, este producto logro reducir el tamaño de la aplicación a unas 1000 líneas de código lo que logro convencer a Google para que patrocine este proyecto. (AngularZone, 2015)

2.3.3. Instalación

Angular está desarrollado en JavaScript, por lo tanto, basta con agregar el script en el proyecto desde algún CDN que lo contenga y está listo para ser usado, la figura 3 lo ejemplifica:

```
<html ng-app="myApp">
<head>
    <title>Hola Mundo</title>
    <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.min.js"></script>
</head>
<body>
    <h1>Esta es una suma: {{ 2 + 2 }}</h1>
</body>
</html>
```

Figura.3. Instalación AngularJS utilizando el CDN de Google.

Fuente: Stuardo, 2016.

La directiva ng- app, marca el inicio de la utilización de AngularJS, al colocarlo junto a la etiqueta HTML permite que AngularJS controle la aplicación desde esta etiqueta, también se puede colocar dentro de cualquier otro elemento, y todo lo que sea hijo de éste también se maneja con AngularJS.

Las expresiones encerradas entre llaves dobles es una sintaxis de AngularJS: si es una variable, la interfaz de usuario se mantiene actualizada con los cambios que se puedan dar, y si se trata de una expresión, AngularJS lo evalúa y mantiene actualizado la interfaz con los cambios de valor de la expresión. (Seshadri, S., Green, B., 2014, pp.11-11).

De estar todo correcto el resultado en el navegador debe ser presentado en la figura 4:

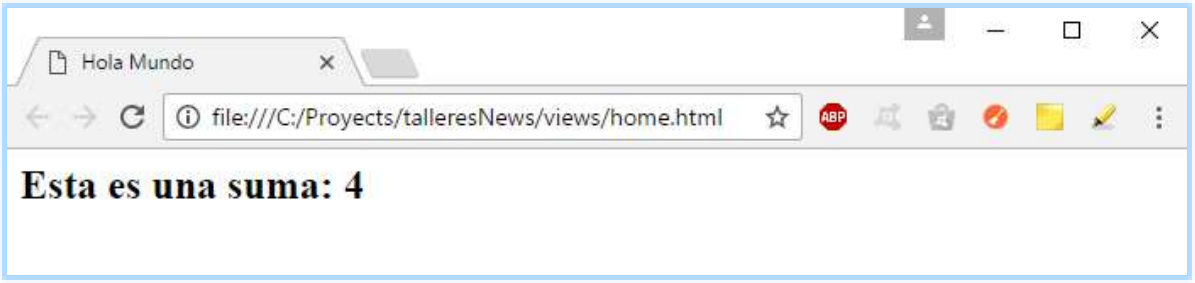


Figura.4. Ejemplo Framework AngularJS.

Fuente: Stuardo, 2016.

2.3.4. Conceptos Generales

En la tabla 1 se presentan los conceptos más importantes de AngularJS:

Concepto	Descripción
Template	HTML con marcado adicional
Directivas	HTML extendido con atributos y elementos personalizados
Model	Datos que se muestran al usuario en la vista
Scope	En este se almacena el modelo para que los controladores, directivas y expresiones puedan acceder a él
Expressions	Variables de acceso y funciones del ámbito
Compilador	Analiza la plantilla y crea una instancia, directivas y expresiones
Filtro	Formatea el valor de una expresión para la visualización al usuario
View	Lo que ve el usuario (DOM)
Data Binding	Datos de sincronización entre el modelo y la vista
Controller	Lógica de negocio detrás de las vistas
Dependency Injection	Patrón de diseño de como los componentes se apoderan de sus dependencias
Injector	Contenedor de inyección de dependencias
Module	Contenedor para las diferentes partes de una aplicación, controladores, servicios, filtros, directivas que configura el inyector
Service	Lógica de negocio reutilizable independiente de vistas

Tabla.1. Conceptos importantes AngularJS.

Fuente: https://docs.angularjs.org/guide/concepts, 2016.

2.3.5. Características de AngularJS

A continuación, nombraremos las características más destacadas de Angular.

2.3.5.1. $scope

El $scope, es uno de los puntos fundamentales de AngularJS, dentro de este se tiene acceso a las variables y los métodos que enlazan al controlador con la vista. Es decir, todo controlador debe poseer un $scope que le permita interactuar con la vista, en la estructura Model-View-Controller, este objeto se convierte en el modelo, en la figura 5 se ilustra la interacción entre sus componentes. (Ahmed, 2014)

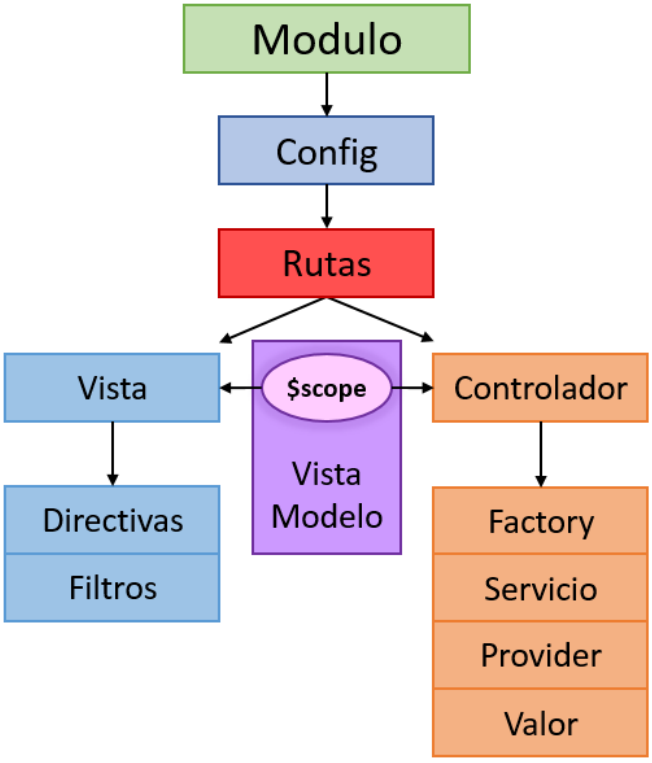


Figura.5. $scope en Angular.

Fuente: http://conceptf1.blogspot.cl/2014/05/learning-angularjs-part5.html, 2014.

En la figura 6 se muestra un ejemplo del funcionamiento del $scope entre la vista y el controlador.

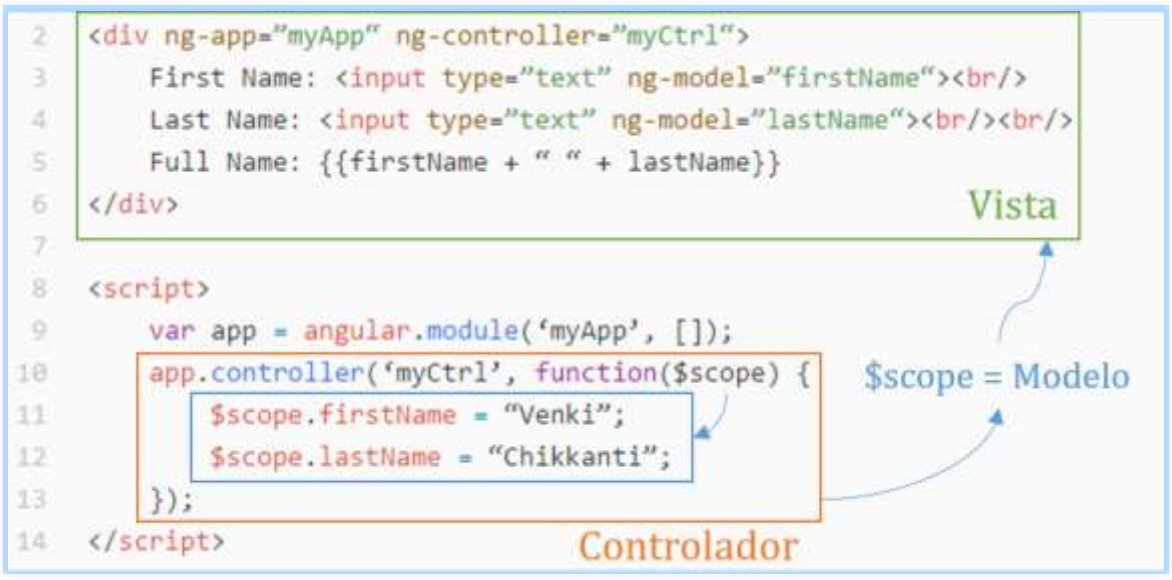


Figura.6. Ejemplo funcionamiento $scope entre vista y controlador.

Fuente: https://chikkanti.files.wordpress.com/2016/03/mvc-architecture-using-angularjs, 2016.

2.3.5.2. Data Binding

"Enlace de Datos", Sin duda uno de los puntos fuertes de AngularJS.

El binding es enlazar los datos del $scope con lo que mostramos en el HTML de forma automática, producida en dos sentidos.

- One-way binding:

Los datos van únicamente desde el $scope hacia la parte visual, (modelo hacia la vista). Se consigue mediante la sintaxis de llaves dobles.

{{Datos}}

Si se modifica el dato del modelo ($scope), se actualiza automáticamente la vista, la figura 7 muestra el funcionamiento del One-way Data Binding.

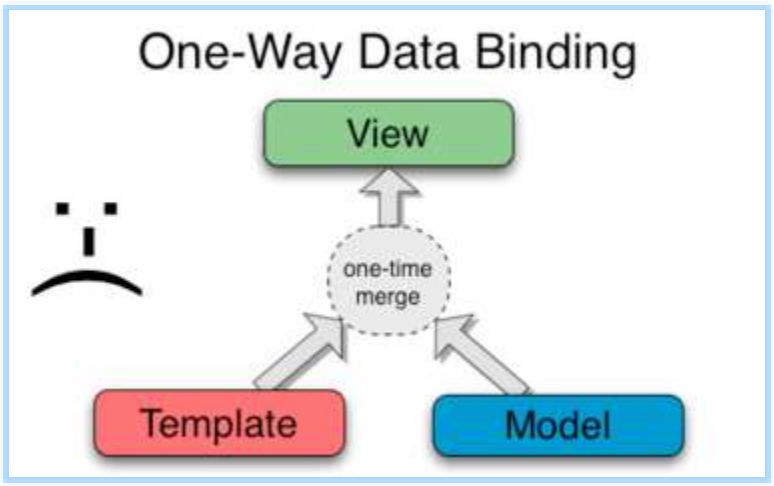


Figura.7. One-Way Data Binding.

Fuente: https://docs.angularjs.org/guide/databinding, 2016.

- Two-way binding:

Los datos fluyen no solo desde el $scope hacia la vista, sino también desde la vista hacia el $scope, se implementa mediante la directiva ngModel, la figura 8 muestra el uso de ng-model.

```
<input type="text" ng-model="miDato"/>
```

Figura.8. Ejemplo de uso de ng-model.

Fuente: Stuardo, 2016.

Cuando el modelo cambia, el dato que está dentro de la caja de texto cambia y cuando el usuario cambie el valor de la caja de texto el $scope se actualiza automáticamente. La figura 9 presenta el funcionamiento del enlace de datos bidireccional de AngularJS.

```
<div ng-app>
  <input type="text" ng-model="valor" />  <!--Doble enlace -->
  {{ valor }}          <!--Enlace simple -->
</div>
```

Figura.9. Enlace de datos Bidireccional AngularJS.

Fuente: Stuardo, 2016.

La figura 10 muestra el funcionamiento del Two-way Data Binding.

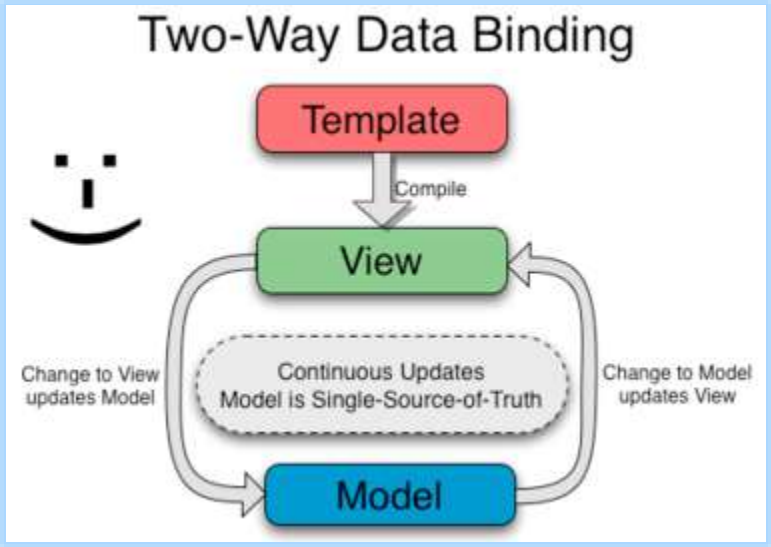


Figura.10. Two-Way Data Binding.

Fuente: https://docs.angularjs.org/guide/databinding, 2016.

La plantilla, que consta de HTML sin compilar, junto con las directivas adicionales, se compila en el navegador. Al realizarse el paso de compilación genera una vista en vivo. Todos los cambios dados en la vista se reflejan al instante en el modelo ($scope) y viceversa.

El término "compilar" una plantilla HTML, para Angular significa agregar los detectores de eventos al HTML. En otras palabras, es agregar listeners (escuchadores) de eventos al HTML.

2.3.5.3. Directivas

Son marcadores en un elemento DOM (atributo, comentario o clase CSS) estos dan aviso al compilador HTML de angular ($compile), para que sea adjuntado el comportamiento especificado para ese elemento DOM o hasta se podría transformar el elemento DOM y sus hijos. Es la manera en la que se puede extender el HTML, permitiéndole agregar código.

- Tipos de Directivas

Las directivas pueden ser basadas en nombres de elementos, atributos, nombres de clase, así como los comentarios, se pueden dar en el nombre de atributo, nombre de la etiqueta, comentarios, o el nombre de la clase de atributos, la figura 11 muestra los tipos de directivas en AngularJS.

```
2  <my-dir></my-dir>
3  <span my-dir="exp"></span>
4  <!-- directive: my-dir exp -->
5  <span class="my-dir: exp;"></span>
```

Figura.11. Tipos de Directivas en AngularJS.

Fuente: Stuardo, 2016.

Aunque puede ser utilizado de distintas maneras, es preferible utilizar las directivas en el nombre de la etiqueta y atributos. (docs.angularjs.org, 2016, https://docs.angularjs.org).

- Principales directivas de AngularJS.

La tabla 2 presenta una breve descripción de las directivas nativas de AngularJS, hay que decir que los nombres de las directivas se crean en camelcase (ngApp), pero se separa por un guion (ng-app) en la vista.

Directiva	Descripción
ngApp (ng-app)	Permite auto arrancar una aplicación Angular, indica el elemento raíz, se coloca como atributo en la etiqueta que quieres que sea la raíz de la aplicación.
ngController (ng-controller)	Permite indicarle a la vista donde trabajará nuestro controlador y enlazar un $scope, todo modelo que este dentro del ámbito de la directiva podrá ser accedido desde el controlador asignado.
ngModel (ng-model)	Representa el modelo o dato, permite obtener la información ingresada por el usuario en algún elemento del formulario, basta con asociarle un modelo y éste podrá ser accedido tanto en el controlador como la vista mediante el nombre del modelo.
ngClick (ng-click)	Trabaja relacionado al evento click, se le puede asociar alguna funcionalidad en cuanto el usuario haga click sobre algún elemento.
ngInit (ng-init)	Permite evaluar una expresión en el scope donde se está trabajando.
ngRepeat (ng-repeat)	Permite iterar una colección de datos, generar un template por cada elemento de la colección y pintarlo en la vista, cada template o plantilla recibe su propio ámbito ($scope).
ngChange (ng-change)	Detecta los cambios que se produzcan dentro de una etiqueta de entrada, inputs, checkbox.
ngShow (ng-show) | ngHide (ng-hide)	Permiten mostrar y ocultar alguna parte de la vista.
ngBind (ng-bind)	Cumple la misma funcionalidad que las llaves **{{}}**, sin embargo, **ng-bind** tiene una mejor performance en cuanto a tiempo.

Tabla.2. Principales Directivas en AngularJS.

Fuente: https://docs.angularjs.org/guide/directive, 2016.

Además de estas directivas angular permite crear directivas personalizadas que permiten ahorrar muchas líneas de código y permiten estructurar la aplicación de maneras más modular, en la figura 12 un ejemplo.

```
var myApp = angular.module('MiModulo', []);
app.controller('MiControlador', function($scope) {
    $scope.cliente = {
        nombre: 'Juan',
        direccion: 'Av. Ecuador 481'
    };
});
app.directive('miCliente', function() {
    return {
        templateUrl: 'cliente.html'
    };
});
```

Figura.12. Ejemplo de Directivas personalizada en AngularJS.

Fuente: Stuardo, 2016.

Mientras tanto en la vista quedaría representada como se muestra en la figura 13:

```
<body>
    <div ng-controller="MiControlador">
        <div mi-cliente></div>
    </div>
</body>
```

Figura.13. Ejemplo de Directivas personalizada en AngularJS (Vista).

Fuente: Stuardo, 2016.

Y lo que se muestra en el navegador es la plantilla cliente.html, la plantilla retornada en la figura 14:

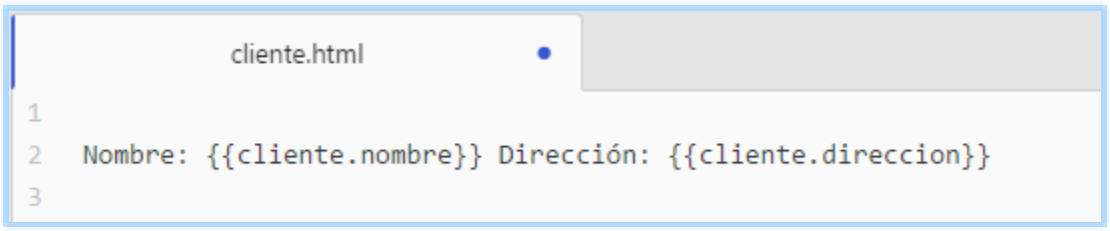


Figura.14. Plantilla que retorna la directiva personalizada.

Fuente: Stuardo, 2016.

2.3.5.4. Expresiones

Se utilizan para representar datos dentro de cualquier parte de un documento HTML, estos pueden ser cadenas de texto, números, operaciones matemáticas, variables o funciones, su sintaxis es: {{Dato}}.

Ejemplo de expresiones válidas en angular:

- **Expresiones numéricas**

En la figura 15 se muestra la expresión numérica.

Figura.15. Ejemplo de expresión numérica aceptada por Angular.

Fuente: Stuardo, 2016.

- **Expresiones de cadena de texto**

En la figura 16 se muestra la expresión de cadena de texto.

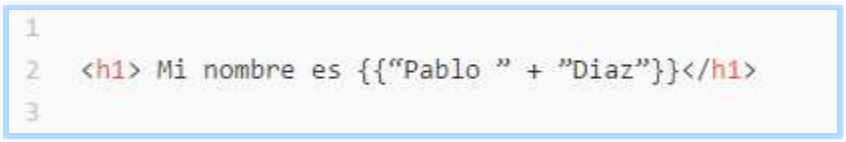


Figura.16. Ejemplo de expresión en cadena de texto aceptada por Angular.

Fuente: Stuardo, 2016.

- **Expresiones con Arrays**

En la figura 17 se muestra la expresión de Arrays.

```
1
2   <h1> Usuario:{{ usuario[1] }}</h1>
3
```

Figura.17. Ejemplo de expresión con Arrays aceptada por Angular.

Fuente: Stuardo, 2016.

- **Expresiones con Objetos**

En la figura 18 se muestra la expresión de objetos.

```
1
2   <h1> Usuario:{{ usuario.nombre }}</h1>
3
```

Figura.18. Ejemplo de expresión con Objetos aceptada por Angular.

Fuente: Stuardo, 2016.

2.3.5.5. Módulos

Contenedores de distintas partes de la aplicación. Se puede crear la cantidad de módulos que se vaya a utilizar, la idea es desarticular el código y tratar de agruparlo ya sea por características, por funcionalidades, por componentes reusables. El desacoplar el código facilita mantenerlo y escalarlo. La forma de declarar un módulo se puede observar en la figura 19.

```
1
2   angular.module('Nombre_modulo', []);
3
```

Figura.19. Declaración de un módulo en Angular.

Fuente: Stuardo, 2016.

Se utiliza el método module que es nativo de angular, se asigna el nombre que lo identificará y luego se declara las dependencias dentro de los corchetes, en el caso de que se necesite utilizar múltiples dependencias basta con separarlos por comas. La forma de declarar un módulo con sus dependencias se puede observar en la figura 20.

```
1
2  angular.module('Nombre_modulo', ['dependencia1', 'dependencia2', 'dependenciaN']);
3
```

Figura.20. Declaración de un módulo con sus dependencias en Angular.

Fuente: Stuardo, 2016.

Se ha declarado el módulo en la parte de JavaScript, para que funcione hay que enlazarlo a la vista y para ello hay que usar la directiva ng-app. La figura 21 muestra el módulo enlazado con la vista mediante ng-app.

```
2  <html lang="es" ng-app="Nombre_modulo">
3  <body>
4      <h1>AngularJS</h1>
5      <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.min.js">
6      </script>
7      <script src="app.js">
8      </script>
9  </body>
10 </html>
```

Figura.21. Módulo enlazado en la vista mediante la directiva ng-app.

Fuente: Stuardo, 2016.

Se añade la directiva ng-app en la etiqueta HTML y le indicamos el módulo que trabaja en la vista (Nombre_modulo), se importa el archivo angular.min.js desde el DNS de google y añadimos el archivo donde se tiene guardado el módulo, en este caso es app.js.

2.3.5.6. Controladores

De manera general es el encargado de gestionar los eventos, este objeto permite desarrollar la lógica de la aplicación, permite el enlace entre el scope con la vista, permitiendo tener control total de los datos.

Como declarar un controlador, se puede observar en la figura 22.

```
2  var app = angular.module('Nombre_App', []);
3  app.controller('Nombre_Controlador', function($scope) {
4      $scope.usuario = "Pedro";
5  });
```

Figura.22. Ejemplo de declaración del controlador.

Fuente: Stuardo, 2016.

Se da un nombre e inyecta las dependencias, todo controlador tiene un $scope asociado, se puede inyectar otras dependencias nativas de AngularJS o personalizados, si se desea inyectar múltiples dependencias hay que separarlos por comas, en la figura 23 un ejemplo de controlador declarado con dependencias.

```
var app = angular.module('Nombre_App', []);
app.controller('Nombre_Controlador', function($scope, dependencia1, dependencia2) {
    $scope.usuario = "Pedro";
});
```

Figura.23. Ejemplo de declaración del controlador con dependencias.

Fuente: Stuardo, 2016.

Los controladores se enlazan con la vista a través de la directiva ng-controller, como se muestra en la figura 24.

```
<body>
    <div ng-controller="nombreControlador">
        <h1>Usuario: {{usuario}}</h1>
    <div/>
</body>
```

Figura.24. Ejemplo de controlador enlazado a la vista.

Fuente: Stuardo, 2016.

2.3.5.7. Servicios

Son los objetos que permite obtener información, un servicio no interactúa con la propia página, sino con otros servicios, con un servidor que pueda estar en otro host. Se caracteriza principalmente porque solo hay una única instancia, pero puede ser utilizado más de una vez, en otras palabras, es singleton.

Existen 2 tipos principales de Servicios.

- Service: Se le pasa una clase, AngularJS crea internamente una instancia de la clase, todo lo que se establece en la instancia de nuestro servicio utilizando "this" es público y estará expuesto a cualquier otro controlador o la Directiva donde inyectemos este servicio.

- Factory: Se le pasa una función para que ésta retorne el valor del servicio, devuelve selectivamente lo que queremos exponer. Esto permite tener variables privadas que no puede ser accedido desde el interior de un controlador.

En la figura 25 se muestra un ejemplo del uso de Factory y Service en AngularJS.

```
var app = angular.module("app", []);
app.factory('miFactory', function() { <!--creamos la factory-->
    return {
        datosFactory: [{
            id: 0,
            nombre: "Israel Parra",
            edad: "32 años"
        }]
    };
});
app.service('miServicio', function() { <!--creamos el servicio-->
    this.datosServicio = function() {
        return [{
            id: 0,
            nombre: "Israel Parra",
            edad: "32 años"
        }]
    }
});
<!--creamos el controlador e injectamos tanto la factory como el servicio-->
app.controller("factoriaController", function($scope, miFactory, miServicio) {
    <!--devolvemos una factory con scope-->
    $scope.factorys = miFactory.datosFactory;
    <!--devolvemos un servicio, vemos como los dos hacen lo mismo, pero de distinta forma-->
    $scope.servicios = miServicio.datosServicio();
});
```

Figura.25. Ejemplo de uso de Factory y Service.

Fuente:https://www.uno-de-piera.com/consumir-datos-con-factorias-y-servicios-en-angularjs/, 2016.

2.3.5.8. Routing

En la inicialización de la aplicación definimos las rutas con los pares Controlador-Vista. De esta manera podremos definir la ruta que muestra la aplicación mediante #/'Ruta'.

En la figura 26 se ve el archivo app.js que muestra el uso de Routing en AngularJS.

```
var app = angular.module('app', []);
<!-- Definimos las rutas de la 'app' e inyectamos las dependencias de los objetos necesarios, en este caso el enrutador-->
app.config(['$routeProvider', function($routes) {
    <!--Indica la funcionalidad a ejecutar al seleccionar cada ruta-->
    $routes.
    when('/nuevaOportunidad', {
        <!--indica la vista a cargar-->
        templateUrl: './nuevaOp.html', <!--indica el controlador de dicha vista controller: NuevaOp-->
    }).
    <!--Mediante dos puntos (:) definimos un parámetro que será enviado por el enrutador al controlador indicado-->
    when('/ops/:opId', {
        templateUrl: './op-detail.html',
        controller: OpsDetailCtrl
    }).
    <!--Se establece la ruta por defecto-->
    otherwise({
        redirectTo: '/index'
    });
}]);
```

Figura.26. Ejemplo de uso de Routing.

Fuente: Stuardo, 2016.

Mediante la llamada $routes.when definimos la URL y el par controlador-vista a ejecutar.

Como se puede apreciar mediante dos puntos ':' podemos pasarle parámetros. En este caso al controlador encargado de mostrar los detalles de una oportunidad le pasamos como parámetro el identificador de la oportunidad: opId.

Con otherwise podemos definir una ruta por defecto haciendo redirect en cualquier ruta que no corresponda a las ya definidas. En el ejemplo si la ruta no la encuentra irá al home que es lo mismo que ir a #/index.

3. Estudio Comparativo

3.1. Introducción

JavaScript uno de los lenguajes más utilizados en la actualidad, debido a que es un lenguaje interpretado, esto hace que el código no necesite ser compilado, sino simplemente puede ser ejecutado en un navegador sin la necesidad de procesos intermedios.

La utilización de JavaScript, potencia las páginas haciéndolas dinámicas, en otras palabras, extiende HTML para realizar funciones que no se podían realizar comúnmente, es por ello que se han creado varios Frameworks JavaScript para la ayuda del desarrollo de aplicaciones web y de esta manera facilitar su utilización.

En el presente capítulo se realizó el análisis del rendimiento de AngularJS, para luego hacer una comparación con algunos de los frameworks basados en JavaScript más usados del momento, más específicamente a los que se ajustan al modelo Cliente MVC.

Para tener más claro a que se refiere Cliente MVC una breve descripción:

Cliente MVC: frameworks JavaScript que cumplen con el patrón modelo-vista-controlador. Se ejecutan en el navegador y se comunican con una capa de servicios para obtener los datos. Ejemplos: Angular, Backbone, Ember, ExtJS.

Sabiendo esto para nuestro estudio que busca comparar AngularJS con los otros framework del mercado tomaremos como sujetos de prueba Backbone y Ember ya que fueron los que más se acercaban en características a AngularJS.

3.2. Presentación de los Framework

En esta sección se muestra una breve descripción de cada framework y las principales ventajas y desventajas de cada uno.

3.2.1. AngularJS

AngularJS es un framework de JavaScript de código abierto, mantenido por Google, que ayuda con la gestión de lo que se conoce como aplicaciones de una sola página (SPA), que extiende el tradicional HTML con etiquetas propias. **Fuente:** https://angularjs.org/.

3.2.1.1. Logo

La figura 27 muestra el logo usado por AngularJS.

Figura.27. Logo AngularJS.

Fuente: https://docs.angularjs.org/, 2016.

3.2.1.2. Ventajas

- Ligero y con buena gestión de dependencias.
- Potente sistema de plantillas, extendiendo vocabulario de HTML básico.
- El concepto de directivas, que permite crear nuevos tags customizados que incorporan tanto funcionalidad como capa visual.
- Posee un potente enlace de interfaces (UI-Binding).
- Posee buenas herramientas para hacer debug.
- Inyección automática de dependencias.
- Desacoplamiento del DOM de JavaScript
- Internacionalización i18n y l10n.

3.2.1.3. Desventajas

- Curva de aprendizaje muy elevada.
- La escritura de directivas es compleja, es la parte más difícil de escribir código en Angular.

3.2.1.4. Resumen Gráfico

En el gráfico de la figura 28, se muestra una foto de las áreas en las que destaca frente a las que es más débil. Hay que tener en cuenta que no todas las áreas tienen la misma importancia, como se puede ver en los pesos asignados a los criterios de la comparación.

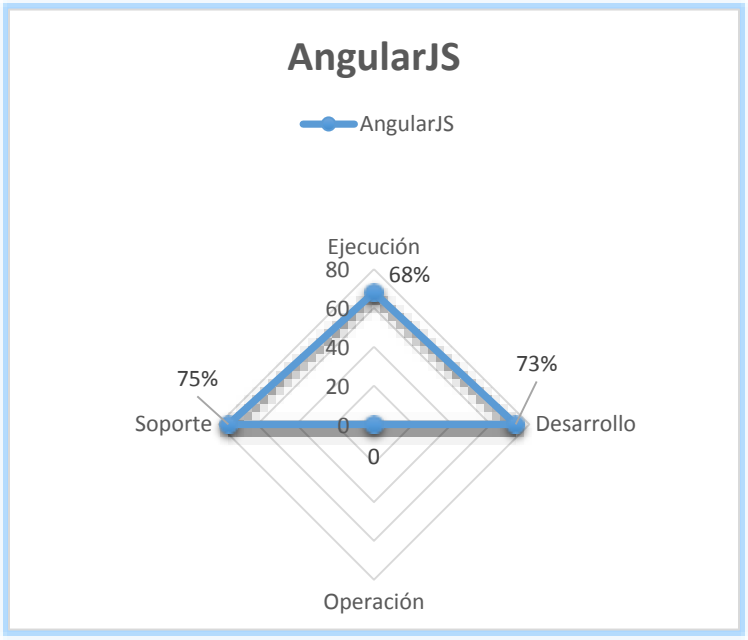


Figura.28. Gráfico resumen de áreas en evaluación para AngularJS.

Fuente: Stuardo, 2016.

3.2.2. BackboneJS

Backbone es una herramienta de desarrollo/API para el lenguaje de programación JavaScript con un interfaz RESTful por JSON, basada en el paradigma de diseño de aplicaciones Modelo Vista Controlador. Está diseñada para desarrollar aplicaciones de una única página (SPA) y para mantener las diferentes partes de las aplicaciones web (p.e. múltiples clientes y un servidor) sincronizadas.

Es de gran utilidad para tener la capa de presentación correctamente modulada y con una estructura claramente definida que hace que la aplicación sea mucho más fácil de mantener, mejorar y reutilizar.

Además de la fuerte comunidad que respalda esta tecnología, BackboneJS puede presumir de haber sido utilizado en muchos sitios web de renombre, como son LinkedIn, WordPress.com o Foursquare. **Fuente:** http://backbonejs.org/.

3.2.2.1. Logo

La figura 29 muestra el logo usado por BackboneJS.

Figura.29. Logo BackboneJS.

Fuente: https://backbonejs.org/, 2016.

3.2.2.2. Ventajas

- Posee una comunidad muy fuerte.
- Existe mucha documentación de calidad.
- Clara estructura
- Permite modularizar la capa de presentación
- Simplicidad, sólo posee 4 componentes básicos (Collection, Model, View, Router).
- Es muy fácil inicializarse en él.
- Es muy customizable.

3.2.2.3. Desventajas

- Debido a la ligereza del framework, hay que repetir tareas similares en multitud de ocasiones.
- No posee una estructura rígida y definida.

3.2.2.4. Resumen Gráfico

La gráfica de la figura 30, presenta las áreas en las que destaca frente a las más débiles, siempre teniendo en cuenta los pesos asociados:

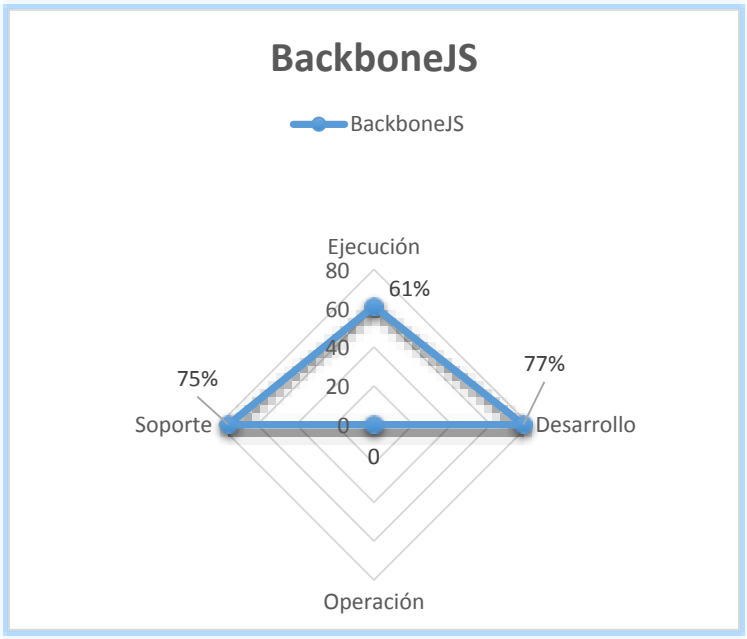


Figura.30. Gráfico resumen de áreas en evaluación para BackboneJS.

Fuente: Stuardo, 2016.

3.2.3. Ember

Ember.js está catalogado como unos de los principales framework en el mundo de JavaScript ya que permite a los desarrolladores crear aplicaciones de una sola página (single-page) escalables. **Fuente:** http://emberjs.com/.

3.2.3.1. Logo

La figura 29 muestra el logo usado por ember.

Figura.31. Logo ember.

Fuente: https://emberjs.org/, 2016.

3.2.3.2. Ventajas

- Intuitiva separación entre interfaz y controlador.
- Complejas funciones con código simple.
- Estructura determinada y consistente.
- Sistema de plantillas extremadamente rico, con vistas compuestas y enlace de interfaces (UI-Binding).

3.2.3.3. Desventajas

- Su versión 2.10 estable es relativamente nueva (28/11/2016).
- Dificultad para encontrar buena documentación, debido a que los radicales cambios de la versión 2.10.

3.2.3.4. Resumen Gráfico

La gráfica de la figura 32, presenta las áreas en las que destaca frente a las más débiles, siempre teniendo en cuenta los pesos asociados:

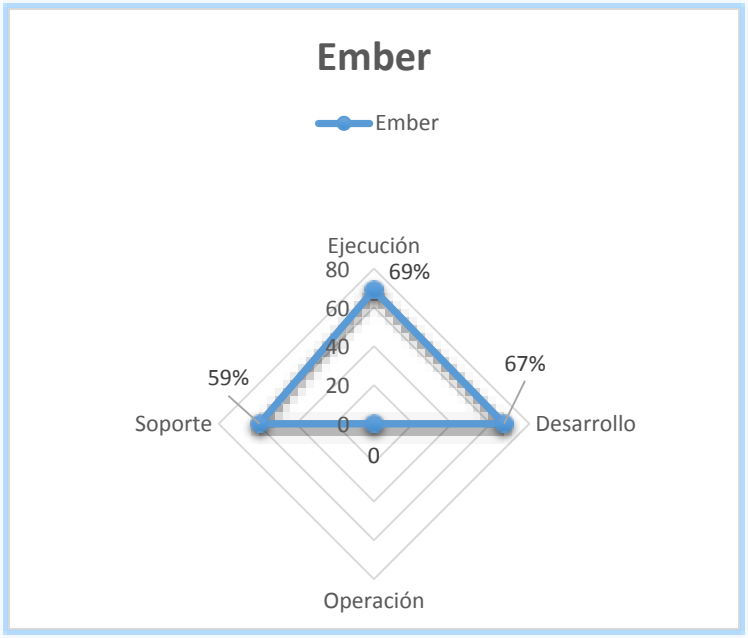


Figura.32. Gráfico resumen de áreas en evaluación para Ember.

Fuente: Stuardo, 2016.

3.3. Análisis Comparativo

A continuación, se detalla la metodología que se ha utilizado para llevar a cabo la comparativa entre AngularJS, Backbone y Ember, que fueron los frameworks seleccionados para este estudio, así como una explicación del funcionamiento de las tablas que contienen los datos de las comparaciones.

En primer lugar, se han seleccionado los frameworks más relevantes en la categoría frameworks Cliente MVC.

Una vez seleccionados los frameworks sobre los que realizar la comparativa, se han definido los criterios a analizar y se han seleccionado las características más relevantes para cada tipología. Estas características han sido agrupadas en cuatro áreas:

- **Ejecución:** criterios fundamentados en las capacidades del framework.
- **Desarrollo:** criterios que facilitan el trabajo del desarrollador de la aplicación, desde la escritura del código hasta las pruebas, automatización de tareas o tecnologías a emplear.
- **Operación:** atiende a criterios para el mantenimiento de la aplicación.
- **Soporte:** criterios que hacen referencia a la calidad de la documentación, soporte del proveedor, soporte de la comunidad de usuarios, existencia de certificados, licencia del producto y evolución histórica del framework.

Para cada tipo de framework a analizar se han definido las siguientes tablas:

- **Análisis:** existe una tabla de este tipo por cada framework que corresponden a la tabla 5, tabla 6 y tabla 7, en la que se detalla la puntuación obtenida en cada criterio junto a una explicación aclaratoria de dicha nota.
- **Pesos:** en esta tabla se pondera la puntuación de cada criterio con el peso asociado a este, el detalle se muestra en la tabla 8.
- **Resultados:** tabla resumen con la puntuación final de cada framework en cada criterio, junto a la nota final del framework, la cual se representa en la tabla 9.

La nota asignada a cada framework para cada una de las características analizadas está definida por los siguientes niveles, tal como se presenta en la tabla 3.

Cobertura		**Nivel**
N	0%	Ninguna
B	25%	Bajo
M	50%	Medio
A	75%	Alto
T	100%	Total

Tabla.3. Notas para las características analizadas.

Fuente: Stuardo, 2016.

Una vez asignado el nivel, este será multiplicado por el peso asociado al criterio de comparación en cuestión, obteniendo como resultado una nota.

Cada peso está asignado tomando en cuenta cómo funciona cada framework y la diferentes formas y características que usa para la implementación de las aplicaciones web.

Los pesos están comprendidos en un rango del 1 al 3, tal como se presenta en la tabla 4.

Coeficiente de Peso	Valor
Imprescindible	3
Importante	2
Aconsejable	1

Tabla.4. Valores para los distintos pesos.

Fuente: Stuardo, 2016.

Una vez realizado este proceso para cada criterio, se calcula la media ponderada de cada framework, sumando las notas obtenidas y dividiendo el resultado entre la suma total de los pesos. Ésta será la nota final del framework en la comparativa.

3.3.1. Definición de parámetros de comparación

A continuación, se expone un análisis más exhaustivo de los frameworks atendiendo a los siguientes criterios:

Dentro del área de ejecución los criterios son:

- **Componentización:** riqueza de componentes ofrecidos por el framework, funcionalidad de los componentes y capacidad de personalización.
- **Navegación:** capacidad del framework para enrutar servicios.
- **Integración con terceros:** capacidad para integrar frameworks de terceros.
- **Manejo de Eventos:** eventos ofrecidos por el framework al interaccionar con la aplicación.
- **Seguridad:** gestión de la autenticación y autorización de los usuarios de la aplicación. La autenticación permite comprobar que el usuario es quien dice ser, mientras que con la autorización se comprueba que el usuario tenga permiso para acceder al recurso que solicita.
- **Gestores comunes:** gestión que ofrece el framework relativo a captura de errores, gestión de sesiones y gestión de la configuración.
- **Multidispositivo/Multinavegador:** soporte de los diferentes navegadores y dispositivos.

Dentro del área de desarrollo los criterios son:

- **Herramientas de desarrollo:** herramientas que facilitan el desarrollo como plugins, toolkits y compatibilidad con los diferentes IDE's.
- **Herramientas de test:** capacidades que permiten acciones de debug, trazado, medición de rendimiento.
- **Herramientas de despliegue:** facilidad para el despliegue de la aplicación que utiliza el framework. Analizando el tamaño de la librería, las dependencias de terceros que tenga.
- **Facilitadores:** plantillas, proyectos demo, para acelerar la fase inicial.
- **Análisis de la calidad:** análisis automático de la calidad del sistema.
- **Independencia del servidor:** capacidad de la capa del cliente para poder trabajar con un servidor de forma transparente a la tecnología de este.

Dentro del área de operación los criterios son:

- **Monitorización de la aplicación:** herramientas ofrecidas por el framework para la monitorización de aplicación.
- **Tareas de mantenimiento:** herramientas que proporciona el framework para realizar backups, ejecución de jobs y explotación de logs.

Dentro del área de soporte los criterios son:

- **Soporte Proveedor:** documentación disponible del framework, certificaciones disponibles, soporte de la comunidad y foros.
- **Licencia del producto:** condiciones económicas del uso de las herramientas.
- **Evolución:** grado de actualización, incorporación de nuevas funcionalidades y continuidad.

Todas las notas asignadas a cada punto en la comparación son basadas en la investigación de cada framework y su respuesta correspondiente en cada área, cabe destacar que las notas son una mezcla de los datos obtenidos y el criterio empleado por el investigador. (Porto, 2013)

3.3.2. Análisis de parámetros y variables de comparación

Ahora se analiza cada framework mediante los parámetros antes señalados:

3.3.2.1. AngularJS

El análisis para el framework AngularJS es presentado en la tabla 5.

		AngularJS		
Área	**Característica**	**Cobertura**		**Comentarios**
Ejecución	**Componentización**	M	**50%**	Proporciona componentes propios mediante extensión de etiquetas HTML y permite crear sus propios componentes mediante directivas.
	Navegación	A	**75%**	Dispone de un potente enrutador.
	Integración con terceros	M	**50%**	Proporciona gran flexibilidad para beneficiarse de otros frameworks o librerías como jquery, ui-angular o Bootstrap.
	Manejo de Eventos	A	**75%**	Dispone de un escuchador de eventos.
	Seguridad	A	**75%**	Existen innumerables librerías que se pueden integrar como: lOauth/2, Passport y google account. Se puede extender su comportamiento usando directivas de angular.
	Gestores comunes (errores, sesiones, configuración)	A	**75%**	Dispone de un manejador de errores. También dispone de un servicio de llamadas HTTP para consumir servicios, internacionalización i18n.
	Multidispositivo/Multinavegador	A	**75%**	Soportado por todos los navegadores modernos de escritorio y móvil.
Desarrollo	**Herramientas de desarrollo**	A	**75%**	Existen multitud de plugins para eclipse y otros IDE's como WebStorm, Atom y SublimeText. Para debugger existen plugins como Batarang.
	Herramientas de test	A	**75%**	Buena integración con herramientas de Test como: Karma, Jasmine y PhantomJS.
	Herramientas de despliegue	A	**75%**	Fácil integración con la herramienta de despliegue Grunt.
	Facilitadores (Plantillas, proyectos demo)	A	**75%**	Fácil integración con Yeoman y con el plugin generator AngularJS, una de las mejores herramientas para acelerar las primeras fases de desarrollo de la aplicación.
	Análisis de la calidad (automática)	B	**25%**	No proporciona una funcionalidad específica, se debe desarrollar reglas específicas.
	Independencia del servidor	T	**100%**	Al comunicarse a través de servicios Rest y a través de Json, la tecnología del servidor no influye.
Operación	**Monitorización de aplicación**	N	**0%**	Ninguna.
	Tareas de mantenimiento (Jobs, backups y explotación de logs)	N	**0%**	Ninguna.
Soporte	**Soporte proveedor**	A	**75%**	Tiene el respaldo de un gigante como Google, la cual usa en algunos de sus productos.
	Licencia producto	A	**75%**	Licencia MIT, permite usar, copiar, modificar, publicar, sublicenciar o vender el código, incluyendo el copyright.
	Evolución	A	**75%**	Es uno de los frameworks basados en JavaScript más populares y su comunidad es la que mayor crecimiento está experimentando.

Tabla.5. Análisis de los parámetros a comparar en AngularJS.

Fuente: Stuardo, 2016.

3.3.2.2. BackboneJS

El análisis para el framework BackboneJS es presentado en la tabla 6.

		Backbone		
Área	**Característica**	**Cobertura**		**Comentarios**
Ejecución	**Componentización**	M	**50%**	No provee componentes, necesita apoyarse en frameworks complementarios como jquery, Bootstrap. Aunque permite la creación y reutilización de componentes.
	Navegación	A	**75%**	Dispone de un enrutador.
	Integración con terceros	M	**50%**	Proporciona gran flexibilidad para beneficiarse de otros frameworks o librerías como jquery, ui-angular o Bootstrap.
	Manejo de Eventos	A	**75%**	Dispone de un potente manejador de eventos.
	Seguridad	M	**50%**	No proporciona una funcionalidad específica, aunque permite la integración con los principales servicios de login.
	Gestores comunes (errores, sesiones, configuración)	M	**50%**	Proporciona una gran variedad de servicios, con los cuales puedes construir tus propios servicios.
	Multidispositivo/Multinavegador	A	**75%**	Soportado por todos los navegadores modernos. La única restricción es la dependencia con jquery.
Desarrollo	**Herramientas de desarrollo**	A	**75%**	Existen multitud de plugins para eclipse y otros IDE's como WebStorm y SublimeText. Para debugger existe la herramienta Backbone Eye and debugger.
	Herramientas de test	A	**75%**	Permite la integración con los Frameworks de test: Jasmine, karma.
	Herramientas de despliegue	A	**75%**	Fácil integración con la herramienta de despliegue Grunt.
	Facilitadores (Plantillas, proyectos demo)	A	**75%**	Fácil integración con Yeoman, una de las mejores herramientas para acelerar las primeras fases de desarrollo de la aplicación.
	Análisis de la calidad (automática)	M	**50%**	No proporciona una funcionalidad específica, se debe desarrollar reglas específicas.
	Independencia del servidor	T	**100%**	Al comunicarse a través de servicios Rest y a través de Json, la tecnología del servidor no influye.
Operación	**Monitorización de aplicación**	N	**0%**	Ninguna.
	Tareas de mantenimiento (Jobs, backups y explotación de logs)	N	**0%**	Ninguna.
Soporte	**Soporte proveedor**	A	**75%**	DocumentCloud la empresa detrás de Backbone y su creador Jeremy Ashkenas, participan activamente en el proyecto.
	Licencia producto	A	**75%**	Licencia MIT, permite usar, copiar, modificar, publicar, sublicenciar o vender el código, incluyendo el copyright.
	Evolución	A	**75%**	Se trata de uno de los Frameworks MVC JavaScript más populares, lo utilizan grandes compañías y tiene una gran comunidad de usuarios.

Tabla.6. Análisis de los parámetros a comparar en BackboneJS.

Fuente: Stuardo, 2016.

3.3.2.3. Ember

El análisis para el framework ember es presentado en la tabla 7.

		Ember		
Área	**Característica**	**Cobertura**		**Comentarios**
Ejecución	**Componentización**	A	**75%**	Proporciona componentes visuales y permite crear tus propios componentes.
	Navegación	A	**75%**	Dispone de un servicio de enrutado.
	Integración con terceros	M	**50%**	Se puede integrar con otros Frameworks mediante div's e iFrame's
	Manejo de Eventos	A	**75%**	Proporciona un manejador de eventos.
	Seguridad	M	**50%**	Compatible con otras medidas de seguridad como Content Security Policy (CSP), HTTPS (SSL/TLS).
	Gestores comunes (errores, sesiones, configuración)	A	**75%**	Dispone de un gestor de errores , invocador de servicios Rest, i18n y proporciona servicios customizables.
	Multidispositivo/Multinavegador	A	**75%**	Soportado por la mayoría de navegadores.
Desarrollo	**Herramientas de desarrollo**	M	**50%**	No existen plugins para eclipse, se pueden utilizar otros editores como SublimeText y WebStorm.
	Herramientas de test	A	**75%**	Fácil integración con herramientas de Test como: Karma, Jasmine y PhantomJS.
	Herramientas de despliegue	A	**75%**	Fácil integración con la herramienta de despliegue Grunt.
	Facilitadores (Plantillas, proyectos demo)	M	**50%**	Fácil integración con Yeoman, una de las mejores herramientas para acelerar las primeras fases de desarrollo de la aplicación.
	Análisis de la calidad (automática)	M	**50%**	No proporciona una funcionalidad específica, se debe desarrollar reglas específicas.
	Independencia del servidor	T	**100%**	Al comunicarse a través de servicios Rest y a través de Json, la tecnología del servidor no influye.
Operación	**Monitorización de aplicación**	N	**0%**	Ninguna.
	Tareas de mantenimiento (Jobs, backups y explotación de logs)	N	**0%**	Ninguna.
Soporte	**Soporte proveedor**	M	**50%**	El desarrollo de este framework es muy activo debido a la gran cantidad de contribuidores que posee, se desarrollan versiones estables con una alta frecuencia.
	Licencia producto	A	**75%**	Licencia MIT, permite usar, copiar, modificar, publicar, sublicenciar o vender el código, incluyendo el copyright.
	Evolución	M	**50%**	Se trata de uno de los frameworks que más evolución ha experimentado en los últimos años junto con AngularJS y Backbone.

Tabla.7. Análisis de los parámetros a comparar en Ember.

Fuente: Stuardo, 2016.

3.3.3. Tabla de Pesos

En la tabla 8 se presentan las puntuaciones de cada framework en los distintos parámetros de estudio tomando en cuenta el coeficiente de peso de cada una.

Área	Característica	Coeficiente de Peso	AngularJS Puntuación		Backbone Puntuación		Ember Puntuación	
Ejecución	Componentización	3	50%	1,5	50%	1,5	75%	2,3
	Navegación	3	75%	2,3	75%	2,3	75%	2,3
	Integración otros frontales	2	50%	1,0	50%	1,0	50%	1,0
	Manejo de Eventos	2	75%	1,5	75%	1,5	75%	1,5
	Seguridad	2	75%	1,5	50%	1,0	50%	1,0
	Gestores comunes (errores, sesiones, configuración)	3	75%	2,3	50%	1,5	75%	2,3
	Multidispositivo/Multinavegador	3	75%	2,3	75%	2,3	75%	2,3
			68%		**61%**		**69%**	
Área	Característica	Coeficiente de Peso	AngularJS Puntuación		Backbone Puntuación		Ember Puntuación	
Desarrollo	Herramientas de desarrollo	3	75%	2,3	75%	2,3	50%	1,5
	Herramientas de test	2	75%	1,5	75%	1,5	75%	1,5
	Herramientas de despliegue	1	75%	0,8	75%	0,8	75%	0,8
	Facilitadores (Plantillas, proyectos demo)	2	75%	1,5	75%	1,5	50%	1,0
	Análisis de la calidad (automática)	2	25%	0,5	50%	1,0	50%	1,0
	Independencia del servidor	3	100%	3,0	100%	3,0	100%	3,0
			73%		**77%**		**67%**	
Área	Característica	Coeficiente de Peso	AngularJS Puntuación		Backbone Puntuación		Ember Puntuación	
Operación	Monitorización de aplicación	1	0%	0,0	0%	0,0	0%	0,0
	Tareas de mantenimiento (Jobs, backups y explotación de logs)	1	0%	0,0	0%	0,0	0%	0,0
			0%		**0%**		**0%**	
Área	Característica	Coeficiente de Peso	AngularJS Puntuación		Backbone Puntuación		Ember Puntuación	
Soporte	Soporte proveedor	2	75%	1,5	75%	1,5	50%	1,0
	Licencia producto	3	75%	2,3	75%	2,3	75%	2,3
	Evolución	3	75%	2,3	75%	2,3	50%	1,5
			75%		**75%**		**59%**	

Tabla.8. Tabla de pesos con la nota resultado por cada característica.

Fuente: Stuardo, 2016.

3.3.4. Tabla comparativa resumen frameworks Cliente MVC

La tabla 9 muestra el resumen de los resultados obtenidos en la comparación realizada.

Área	AngularJ	Backbon	Ember
Ejecución	**68%**	**61%**	**69%**
Componentización	50%	50%	75%
Navegación	75%	75%	75%
Integración otros frontales	50%	50%	50%
Manejo de Eventos	75%	75%	75%
Seguridad	75%	50%	50%
Gestores comunes (errores, sesiones, configuración)	75%	50%	75%
Multidispositivo/Multinavegador	75%	75%	75%
Desarrollo	**73%**	**77%**	**67%**
Herramientas de desarrollo	75%	75%	50%
Herramientas de test	75%	75%	75%
Herramientas de despliegue	75%	75%	75%
Facilitadores (Plantillas, proyectos demo)	75%	75%	50%
Análisis de la calidad (automática)	25%	50%	50%
Independencia del servidor	100%	100%	100%
Operación	**0%**	**0%**	**0%**
Monitorización de aplicación	0%	0%	0%
Tareas de mantenimiento (Jobs, backups y explotación de	0%	0%	0%
Soporte	**75%**	**75%**	**59%**
Soporte proveedor	75%	75%	50%
Licencia producto	75%	75%	75%
Evolución	75%	75%	50%
Media	**54,03%**	**53,26%**	**49,03**

Tabla.9. Tabla resumen con los resultados obtenidos.

Fuente: Stuardo, 2016.

3.4. Conclusiones de la comparación entre frameworks

Respecto a los frameworks de ejecución estructurales la conclusión me lleva a la recomendación de los frameworks AngularJS y Backbone utilizando un enfoque Cliente MVC. Esta propuesta se basa en:

- Los beneficios identificados en la prueba de concepto. (Pag. 38 y 40)
- La gran funcionalidad que incorporan o la gran cantidad de herramientas desarrolladas para estos frameworks. (Pag. 38 y 40)
- La popularidad de los frameworks, lo que se traduce en abundante documentación de calidad y soporte por parte de la comunidad.
- Estabilidad y madurez de los frameworks.
- Al tratarse de los frameworks más relevantes del mercado, existen multitud de soluciones en el ámbito del desarrollo como puede ser el debug de la aplicación, herramientas de desarrollo, análisis de calidad de código.

4. Caso Práctico

4.1. Introducción

Una vez realizado el estudio comparativo de los framework basado en la estructura Cliente MVC del capítulo 3, es el momento de exponer la implementación de una pequeña aplicación web utilizando AngularJS.

Habiendo escogido AngularJS que fue el framework mejor evaluado, se crea este ejemplo para mostrar las principales características de la herramienta, como el uso de directivas, la implementación con el uso de los variados servicios que están disponibles para el framework, las ventajas que trae el two-way data binding y así también establecer el modo de trabajo de esta tecnología.

4.2. Caso Práctico: Prototipo página de noticias Talleres ICI

4.2.1. Objetivo

El objetivo general de este caso práctico es mostrar las ventajas del desarrollo con el framework AngularJS mediante una pequeña aplicación web dirigida a los talleres de la carrera de Ingeniería Civil en Informática (Robótica y Videojuegos), que les sirve como una página de noticias, la cual les permite crear su propia cuenta lo que les otorga permisos para publicar noticias, comentarios, y a su vez poder calificarlos, esto los ayuda a organizarse y motivarse en las actividades que los talleres realizan, además de permitirle al profesor encargado tener el conocimiento de quienes participan activamente en el taller.

4.2.2. Descripción

El prototipo fue desarrollado usando el stack MEAN.JS, el cual es open source y nos permite implementar de forma fácil y rápida una aplicación, este stack junta cuatro poderosas herramientas que se complementan de muy buena manera, **M**ongoDB, **E**xpressJS, **A**ngularJS y **N**ode.JS, a continuación, se explica la función de cada una:

4.2.2.1. MongoDB

En el lado del almacenamiento se han utilizado tradicionalmente bases de datos relacionales. Sin embargo, actualmente los tipos de información que suelen requerir las aplicaciones web demandan mayor flexibilidad, menos coherencia y sobre todo mayor capacidad de escalar. Para dar respuesta a todo esto, surge la tendencia tecnológica en almacenes de datos que se denomina NoSQL.

- **NoSQL**

NoSQL es usada según el caso. No existe una solución estándar que cumpla con todos los tipos de requerimientos. Dependiendo del tipo de uso, una solución NoSQL está a la mano. Sin dudar las bases de datos relacionales (RDBMS) lideran el mercado en términos de posicionamiento, pero hay ciertos casos, donde NoSQL prueba ser beneficioso. Cuando hablamos de la cantidad o el tipo de los datos, hay ciertos casos en los que las RDBMS pueden verse como la base de datos a usar, pero se debe optar por una solución más acorde a los requerimientos del proyecto.

La mejor razón para elegir NoSQL es por su escalabilidad. Hay instancias donde las RDBMS no escalan y una solución NoSQL lo hace mejor. Las RDBMS no manejan bien el Data Sharding o partición de bases de datos ya que manejan su carga distribuida horizontalmente, ahí es cuando NoSQL tiene la ventaja ya que no distribuyen entidades lógicas en múltiples tablas; guardan todo en un solo lugar y no forzar la integridad referencial entre estas entidades, sino asegura que la consistencia esté presente dentro de una sola unidad de entidad. La figura 33 muestra la comparación entre el funcionamiento entre NoSQL y RDBMS.

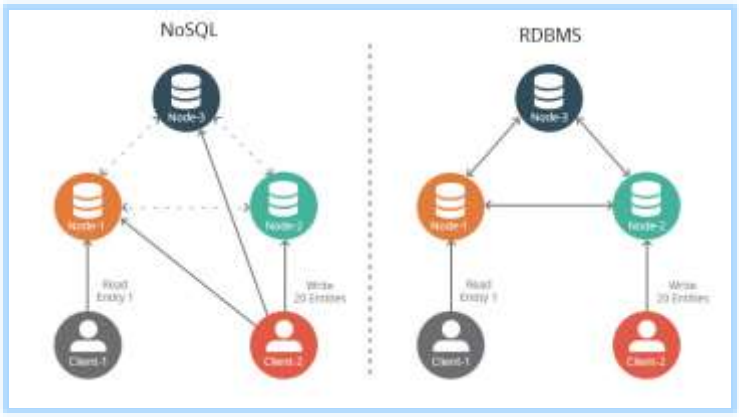


Figura.33. Comparación entre el funcionamiento de NoSQL y RDBMS.

Fuente: http://blog.spec-india.com/nosql-creating-waves-in-the-database-community, 2015.

Estos almacenes de datos NoSQL pueden ser de diversos tipos, pero en muchos casos utilizan JavaScript para representar la información, recibiendo, enviando y almacenando datos usando la notación JSON. **Fuente:** http://blog.spec-india.com/nosql-creating-waves-in-the-database-community

- **JSON**

JSON (JavaScript Object Notation) es un formato para el intercambio de datos, básicamente JSON describe los datos con una sintaxis dedicada que se usa para identificar y gestionar los datos. JSON nació como una alternativa a XML, el fácil uso en JavaScript ha generado un gran número de seguidores de esta alternativa. Una de las mayores ventajas que tiene el uso de JSON es que puede ser leído por cualquier lenguaje de programación. Por lo tanto, puede ser usado para el intercambio de información entre distintas tecnologías.

Para tener una mejor idea tenemos este ejemplo:

Se tiene una frutería y que queremos obtener el nombre y la cantidad de fruta y verdura se tiene. En un principio se suponer que se tiene lo siguiente:

- Fruta:
  - 10 manzanas
  - 20 Peras
  - 30 Naranjas
- Verduras
  - 80 lechugas
  - 15 tomates
  - 50 pepinos

Para empezar, nos tenemos que familiarizar con la sintaxis de JSON:

- JSON Nombre/Par de Valores

Para asignar a un nombre un valor debemos usar los dos puntos ':' este separador es el equivalente al igual ('=') de cualquier lenguaje. La figura 34 muestra la sintaxis JSON.

```
1
2   "Nombre" : "Geeky Theory"
3
```

Figura.34. Sintaxis JSON.

Fuente: https://geekytheory.com/json-i-que-es-y-para-que-sirve-json, 2016.

- Valores Json

Los tipos de valores que podemos encontrar en Json son los siguientes:

- Un número (entero o float)
- Un string (entre comillas simples)
- Un booleano (true o false)
- Un array o arreglo (entre corchetes [])
- Un objeto (entre llaves {})
- Null

- Objetos JSON

Los objetos JSON se identifican entre llaves, un objeto puede ser en nuestro caso una fruta o una verdura, La figura 35 presenta un ejemplo de objeto en JSON.

```
1
2   { "NombreFruta":"Manzana" , "Cantidad":20 }
3
```

Figura.35. Ejemplo Objeto en JSON.

Fuente: https://geekytheory.com/json-i-que-es-y-para-que-sirve-json, 2016.

- Arreglo JSON

En un Json puedes incluir arreglos, para ellos el contenido del arreglo debe ir entre corchetes [], La figura 36 muestra un ejemplo de Arreglo en JSON.

```
2   {
3       "Frutas": [
4           { "NombreFruta":"Manzana" , "cantidad":10 },
5           { "NombreFruta":"Pera" , "cantidad":20 },
6           { "NombreFruta":"Naranja" , "cantidad":30 }
7       ]
8   }
```

Figura.36. Ejemplo Arreglo en JSON.

Fuente: https://geekytheory.com/json-i-que-es-y-para-que-sirve-json, 2016.

Una vez explicado el funcionamiento de la sintaxis JSON, se aplica el ejemplo de la frutería, en la figura 37 se presenta la frutería como un Arreglo conteniendo los objetos "Fruta" y "Verdura".

```
{"Fruteria":
    [
        {"Fruta":
            [
                {"Nombre":"Manzana","Cantidad":10},
                {"Nombre":"Pera","Cantidad":20},
                {"Nombre":"Naranja","Cantidad":30}
            ]
        },
        {"Verdura":
            [
                {"Nombre":"Lechuga","Cantidad":80},
                {"Nombre":"Tomate","Cantidad":15},
                {"Nombre":"Pepino","Cantidad":50}
            ]
        }
    ]
}
```

Figura.37. Ejemplo JSON.

Fuente: https://geekytheory.com/json-i-que-es-y-para-que-sirve-json, 2016.

Como se puede observar, se ha creado un objeto llamado frutería y, dentro de ese objeto se ha almacenado un arreglo de dos elementos. El primer elemento del arreglo contiene un objeto llamado fruta y el segundo elemento del arreglo contiene otro objeto llamado verdura. Estos objetos a su vez contienen un arreglo cuyo contenido es el nombre y la cantidad de cada fruta o verdura.

Luego se quiere saber la cantidad de manzanas que se tienen. El path de este arreglo sería el observado en la figura 38.

```
Path: json['Fruteria'][0]['Fruta'][0]['Cantidad']
```

Figura.38. Ejemplo Path en JSON.

Fuente: https://geekytheory.com/json-i-que-es-y-para-que-sirve-json, 2016.

Se puede observar que la cantidad de manzanas se almacena dentro del primer elemento del array que contiene el objeto Frutería, y a su vez dentro del primer elemento del array que contiene el objeto Fruta. **Fuente:** https://geekytheory.com/json-i-que-es-y-para-que-sirve-json, 2016.

4.2.2.2. ExpressJS

Para ayudar a crear aplicaciones web más fácilmente nació Express. Este framework está escrito en JavaScript para Node.js. Su objetivo es que no se tenga que crear cada vez que se desee crear una aplicación web, ofreciéndonos soporte para las principales necesidades en este tipo de aplicaciones: gestión de peticiones y respuestas, cabeceras, rutas, vistas.

4.2.2.3. AngularJS

Algunos han definido a AngularJS como "lo que HTML debería haber sido si se hubiese diseñado para crear aplicaciones". Angular.js va mucho más allá de ser una simple biblioteca de funciones: es un completo framework que brinda todo tipo de funcionalidades avanzadas, extendiendo de hecho HTML.

4.2.2.4. Node.js

Node.js utiliza por debajo el motor de JavaScript de Google, denominado V8, y provee de una arquitectura orientada a eventos (como la de los navegadores) así como una serie de APIs no-bloqueantes (asíncronas) que le proporcionan un rendimiento y una escalabilidad muy elevadas. Se puede utilizar para crear cualquier tipo de lógica de aplicación, pero dado que incorpora un módulo para poder actuar como un servidor web, es especialmente popular para crear aplicaciones web.

Fuente: http://www.campusmvp.es/recursos/post/Que-es-el-stack-MEAN-y-como-escoger-el-mejor-para-ti.aspx

Lo que permite estas cuatro tecnologías es trabajar con JavaScript en todas las capas de la aplicación, haciendo el aprendizaje mucho más rápido y el desarrollo más ágil.

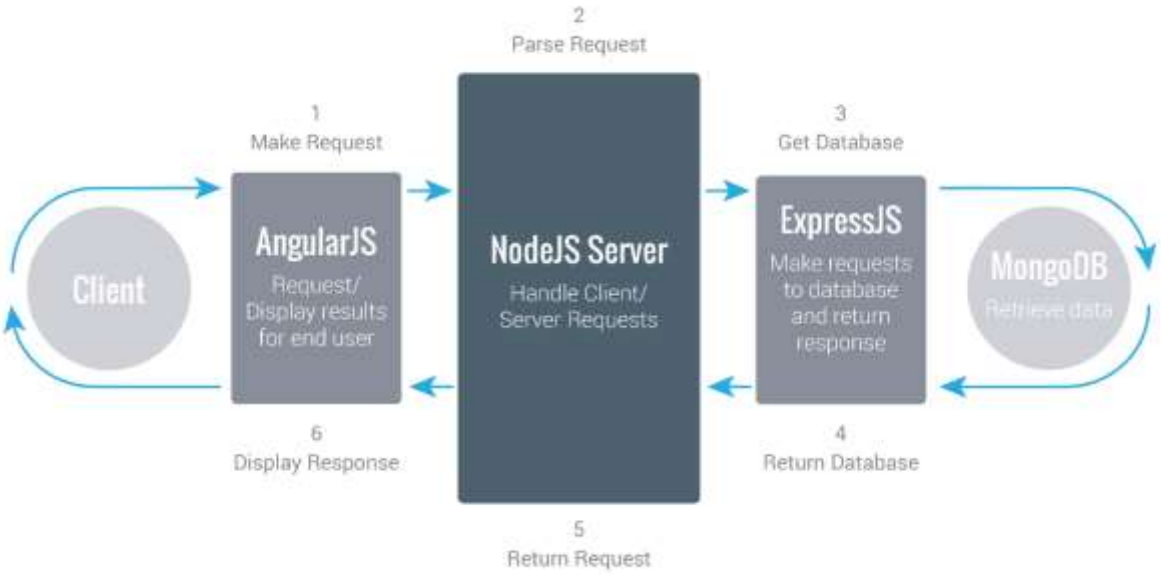


Figura.39. Ilustración de la interacción de los elementos de MEAN.js.

Fuente: http://www.flexitech.io/news/dev-tools/2015/09/03/21-introduction-to-mean-stack, 2016.

4.2.3. Implementación

Al momento de la primera implementación de MEAN.js, ésta contiene todo lo básico de un proyecto lo que da un punto de inicio sólido para aplicaciones basadas en MongoDB, Node.js, Express y AngularJS. Antes de iniciar una aplicación se deben tener las siguientes herramientas instaladas y luego seguir los siguientes pasos:

- **Git:** El control de versiones libre y open-source más usado, nos ayuda a clonar desde el repositorio local de MEAN, se puede descargar desde https://git-scm.com/downloads/.
- **Node.js:** Además de Node.js el instalador tiene incluido NPM que es el manejador de paquetes por defecto para Node.js se puede descargar desde https://nodejs.org/en/download/.
- **MongoDB:** Se puede descargar desde https://www.mongodb.com/download-center.

El primer paso es descargar MEAN.js la forma más sencilla es clonando el repositorio desde GitHub esto se hace usando el comando en cmd de windows presentado en la figura 40.

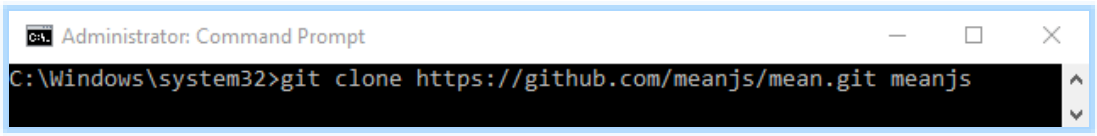


Figura.40. Clonación del repositorio de MEAN.js.

Fuente: Stuardo, 2016.

Esto clona la última versión del repositorio de MEAN.js en una carpeta meanjs.

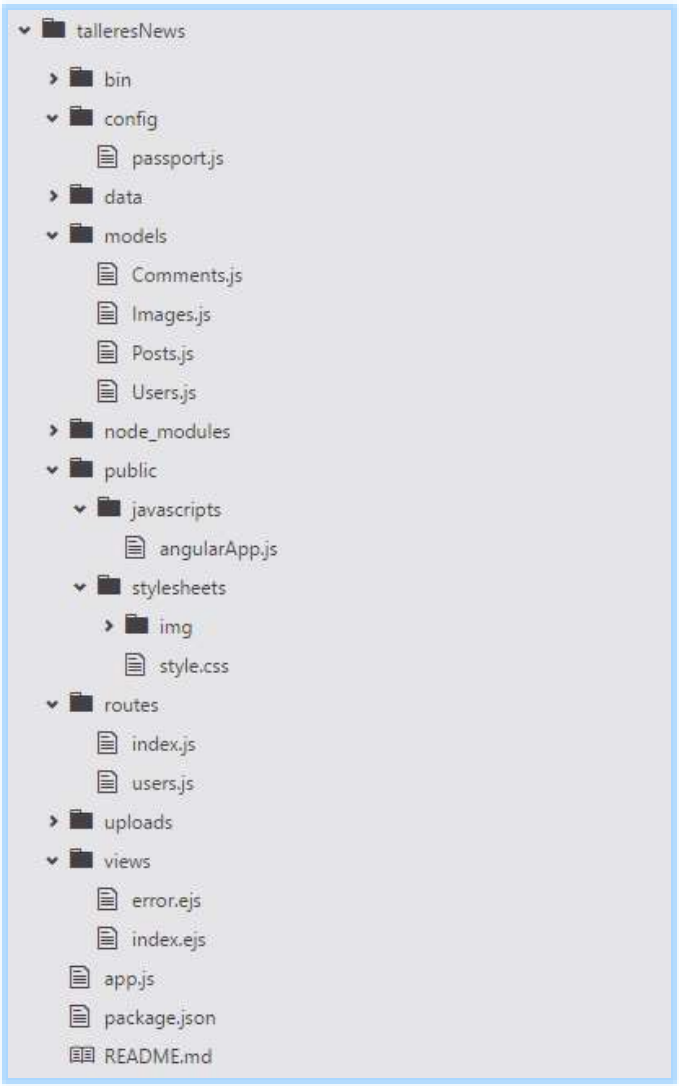


Figura.41. Estructura del prototipo.

Fuente: Stuardo, 2016.

En la figura 41 se observa cual va a ser la estructura del proyecto que se va a desarrollar. En ella hay una carpeta “config” que tiene el fichero del servicio de autentificación de usuarios (“passport.js”), en la carpeta “models” se tienen los modelos que se usan para guardar información en la base de datos, en la carpeta “node_modules” se instalan todos los módulos necesarios para node.js, express y las dependencias que se instalan más adelante, en la carpeta “javascript”, se tiene el fichero que será el encargado de hacer la peticiones al API REST y de tener actualizados los modelos de datos para que los muestre la web (“angularApp”), en la carpeta “routes” se definen los endpoints de la API REST y sus acciones (“index.js” y “users.js”), en la carpeta “uploads” se guardan las imágenes que los usuarios suban para las noticias, en la carpeta “views” tiene los ficheros del frontend ("index.ejs") con una Single-Page-Application y el fichero que muestra los errores (“error.ejs”).

Por último se tienen los ficheros "package.json" para ver las dependencias que se necesitaran y luego el fichero "app.js" en el que esta la configuración del servidor. Ahora se muestra el código fuente de estos ficheros.

En primer lugar, el fichero "Package.json" que aparte de la información relacionada con el proyecto nos indica las dependencias que se necesitan para el proyecto. En este caso se necesitan las librerías de "mongoose" y "express". Express ya explicamos lo que es y "mongoose" permite definir el modelo de datos que se guarda en MongoDB. Otra cosa muy importante (y de esto depende que funcione o no el proyecto) es que se ponga la versión correcta tanto de express como de mongoose que se vaya a utilizar, la librería “passport” permite usar la validación de usuarios, estas son las más importantes. El código de este fichero se presenta en la figura 42.

package.json

```
{
  "name": "talleres-news",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "body-parser": "~1.8.1",
    "cookie-parser": "~1.3.3",
    "debug": "~2.0.0",
    "ejs": "~0.8.5",
    "express": "~4.9.0",
    "express-jwt": "^3.1.0",
    "jsonwebtoken": "^5.4.0",
    "mongoose": "^3.8.17",
    "morgan": "~1.3.0",
    "passport": "^0.3.0",
    "passport-local": "^1.0.0",
    "serve-favicon": "~2.1.3"
  }
}
```

Figura.42. Archivo package.json incluido en el paquete de MEAN.js.

Fuente: Stuardo, 2016.

Una vez instalado los prerrequisitos y clonado el repositorio el paso siguiente es instalar las dependencias que vienen incluidas en el archivo package.json.

Para instalar estas dependencias se usa nuevamente la línea de comandos en el directorio donde se clono "meanjs" y usar el siguiente comando presentado en la figura 43.

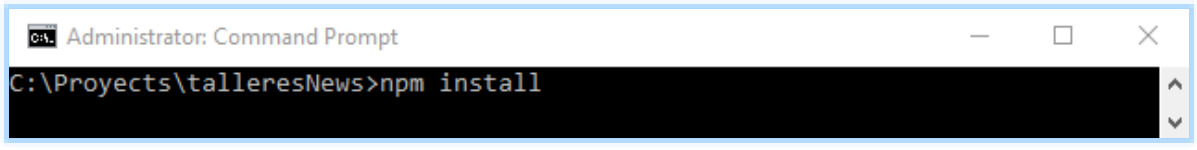


Figura.43. Comando **"npm install"** genera la instalación de las dependencias.

Fuente: Stuardo, 2016.

Este comando hace algunas cosas:

- Primero instala las dependencias necesarias para correr la aplicación.
- Si se está trabajando en un ambiente de desarrollador, instala las dependencias necesarias para testear y correr la aplicación.
- Si cualquier dependencia saca una nueva actualización solo basta con usar el comando **"npm update"** para que se actualicen automáticamente.

Ahora para correr nuestra aplicación usamos el comando indicado en la figura 44.

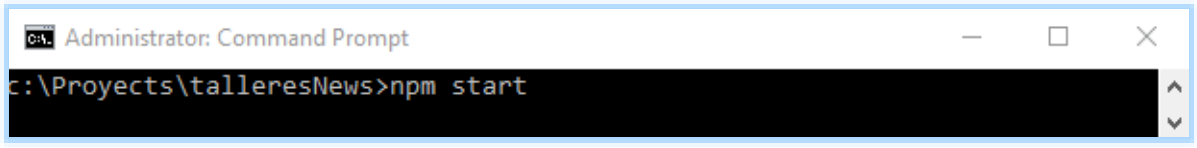


Figura.44. Comando **"npm start"** inicia la aplicación.

Fuente: Stuardo, 2016.

La aplicación debe correr en el puerto 3000 con la configuración del ambiente de desarrollador, solo basta entrar en el buscador http://localhost:3000.

Fuente: https://github.com/meanjs/mean.

El siguiente fichero que se tiene es el "app.js" en el cual se configura el servidor. En él se define las librerías a utilizar (express y mongoose), además de las otras librerías segundarias que utiliza, también se hace la conexión a la base de datos Mongo ("mongodb://localhost:27017/news"), luego de esto, se cargan los modelos que se usan en la base de datos y por último se cargan las rutas o endpoints de la aplicación ("index.js" y "users.js").

app.js

```
var express = require('express');     // Utilizamos express
var mongoose = require('mongoose');   // mongoose para mongodb
var path = require('path');
var favicon = require('serve-favicon');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

var app = express();
var passport = require('passport');   // passport para autenficacion

// Hacemos la conexión a la base de datos de Mongo con nombre "news"
mongoose.connect('mongodb://localhost/news', function(err, db) {
    if (!err) {
        console.log('Connected to /TalleresICINews!');
    } else {
        console.dir(err);
    }
});
// Cargamos los modelos
require('./models/Posts');
require('./models/Comments');
require('./models/Users');
require('./models/Images');
require('./config/passport');

// Cargamos los endpoints
var routes = require('./routes/index');
var users = require('./routes/users');
```

Figura.45. Fichero app.js.

Fuente: Stuardo, 2016.

Ahora se explica cómo se ha hecho el API REST con Node.js. En primer lugar, se muestran los modelos de datos que utiliza. Eso está en los ficheros "Comments.js", “Images.js", "Posts.js" y "Users.js”.

Para el modelo “comments” (comentario) este contiene; body (cuerpo del comentario), author (autor del comentario), date (fecha de creación del comentario), upvotes (votos totales del comentario), post (post al que pertenece la imagen). Su contenido es el indicado en la figura 46.

Comments.js

```
var mongoose = require('mongoose');

var CommentSchema = new mongoose.Schema({
    body: String,
    author: String,
    date: {
        type: Date,
        default: Date.now
    },
    upvotes: {
        type: Number,
        default: 0
    },
    post: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Post'
    }
});
// funcion que suma un voto
CommentSchema.methods.upvote = function(cb) {
    this.upvotes += 1;
    this.save(cb);
};
// funcion que resta un voto
CommentSchema.methods.downvote = function(cb) {
    this.upvotes -= 1;
    this.save(cb);
};
// nombre del cual se puede tener acceso al modelo
// desde cualquier lugar donde se use mongoose
mongoose.model('Comment', CommentSchema);
```

Figura.46. Fichero Comments.js.

Fuente: Stuardo, 2016.

Para el modelo “images” (imágenes) este contiene; imgname (nombre de la imagen), author (autor de la imagen), date (fecha de creación de la imagen), originalname (nombre original de la imagen), post (post al que pertenece la imagen). Su contenido es el indicado en la figura 47.

Images.js

```
var mongoose = require('mongoose');

var ImageSchema = new mongoose.Schema({
    path: {
        type: String,
        required: true,
        trim: true
    },
    imgname: {
        type: String,
        required: true,
        trim: true
    },
    author: String,
    date: {
        type: Date,
        default: Date.now
    },
    originalname: {
        type: String,
        required: true
    },
    post: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Post'
    }]
});
// nombre del cual se puede tener acceso al modelo
// desde cualquier lugar donde se use mongoose
mongoose.model('Image', ImageSchema);
```

Figura.47. Fichero Images.js.

Fuente: Stuardo, 2016.

Para el modelo “posts” (posts) este contiene; title (título del post), link (link del post), body (cuerpo del post), author (autor del post), images (imágenes subidas al post), date (fecha de creación del post), upvotes (votos totales del post), comments (comentarios ingresados al post). Su contenido es el indicado en la figura 48.

Posts.js

```
var mongoose = require('mongoose');

var PostSchema = new mongoose.Schema({
    title: String,
    link: String,
    body: String,
    author: String,
    images: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Image'
    }],
    date: {
        type: Date,
        default: Date.now
    },
    upvotes: {
        type: Number,
        default: 0
    },
    comments: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Comment'
    }]
});
// funcion que suma un voto
PostSchema.methods.upvote = function(cb) {
    this.upvotes += 1;
    this.save(cb);
};
// funcion que resta un voto
PostSchema.methods.downvote = function(cb) {
    this.upvotes -= 1;
    this.save(cb);
};
// nombre del cual se puede tener acceso al modelo
// desde cualquier lugar donde se use mongoose
mongoose.model('Post', PostSchema);
```

Figura.48. Fichero Posts.js.

Fuente: Stuardo, 2016.

Para el modelo “users” (usuarios) este contiene; username (nombre del usuario), hash y salt (se usan para encriptar la password). Su contenido es el indicado en la figura 49.

Users.js

```
var mongoose = require('mongoose');
var crypto = require('crypto'); // Servicio de encriptacion
var jwt = require('jsonwebtoken');

var UserSchema = new mongoose.Schema({
    username: {
        type: String,
        lowercase: true,
        unique: true
    },
    hash: String,
    salt: String
});

UserSchema.methods.generateJWT = function() {
    // expira en 60 days
    var today = new Date();
    var exp = new Date(today);
    exp.setDate(today.getDate() + 60);

    return jwt.sign({
        _id: this._id,
        username: this.username,
        exp: parseInt(exp.getTime() / 1000),
    }, 'SECRET');
};
// funcion que fija la password
UserSchema.methods.setPassword = function(password) {
    this.salt = crypto.randomBytes(16).toString('hex');
    this.hash = crypto.pbkdf2Sync(password, this.salt, 1000, 64).toString('hex');
};
// funcion que valida la password
UserSchema.methods.validPassword = function(password) {
    var hash = crypto.pbkdf2Sync(password, this.salt, 1000, 64).toString('hex');
    return this.hash === hash;
};
// nombre del cual se puede tener acceso al modelo
// desde cualquier lugar donde se use mongoose
mongoose.model('User', UserSchema);
```

Figura.49. Fichero Users.js.

Fuente: Stuardo, 2016.

En el fichero "index.js" se definen los endpoints de la API REST y sus acciones. Para luego escribir las funciones de cada llamada de la API REST, por lo extenso del código se presenta en el capítulo ocho, desde la figura 57 a la 62, correspondiente a los anexos.

Llegados a este punto ya tenemos el API REST con Node.js desarrollado y ahora se "une" con AngularJS; es decir el frontend.

Dentro de Angular los ficheros "angularApp.js" e "index.ejs". El fichero "angularApp.js" será el encargado de hacer las peticiones al API REST y de tener actualizados los modelos de datos para que los muestre la web. El contenido del fichero "angularApp.js" se puede revisar en el capítulo ocho, desde la figura 63 a la 66, correspondiente a los anexos.

Por último, se muestra el fichero "index.ejs" de AngularJS (Para esta página se usó los estilos CSS de Bootstrap). El contenido del fichero "index.ejs" se puede revisar en el capítulo ocho, desde la figura 67 a la 70, correspondiente a los anexos.

Ahora después de mostrar el fichero index.ejs se explican los fragmentos más destacables para comprender la forma como angular interactúa con HTML.

Lo primero que se muestra son todos los datos de la base de datos en una tabla. En el siguiente fragmento de código referente a la tabla, se tiene la etiqueta "ng-repeat" y le indica que nos vaya cogiendo un objeto "post" del array "posts" que está definido en el "angularApp.js" y que los ordene de mayor a menor por la cantidad de "upvotes". Una vez que el objeto "post" lo mostramos en la tabla accediendo a sus atributos ({{ post.upvotes }}, {{ post.title }},...). En definitiva, este "ng-repeat" funciona igual que un bucle "foreach". Con la etiqueta "ng-click" se ejecuta la función "upvote(post)" y downvote(post) que están definida en el "angularApp.js" y que suma o resta un voto al objeto post que seleccione al pulsar una fila de la tabla para mostrarla en el formulario.

La directiva “ng-show” hará visible en el navegador el tag al cual está ligada si el valor de la variable es diferente de Null, en cambio la directiva “ng-hide” oculta el contenido si el valor de la variable es distinto de Null, lo último que se rescata son filtros que permiten utilizar angular nativamente, los cuales son usados después del signo “|”, lo que ahorra mucho código que tendría que desarrollar en el backend de la aplicación, en este caso se pueden ver: el “orderBy” que permite ordenar la tabla con respecto al valor que toma el atributo “upvotes” y el filtro “date” que permite dar a una fecha cualquier formato, lo que se presenta en la figura 50.

```
<div ng-repeat="post in posts | orderBy: '-upvotes'">
    <span class="glyphicon glyphicon-thumbs-up" ng-click="upvote(post)">
    </span> {{post.upvotes}}
    <span class="glyphicon glyphicon-thumbs-down" ng-click="downvote(post)"></span>

    <span style="font-size:20px;margin-left:10px">
      <a ng-show="post.link" href="#/posts/{{post._id}}">{{post.title}}</a>
      <span ng-show="post.author">publicado por <a>{{post.author}}</a> |</span>
    </span>
    <span ng-show="post.date">{{post.date | date:"dd/MM/yyyy 'a las' h:mma"}} |</span>
    <span>
      <a href="#/posts/{{post._id}}">Comentarios</a>
        ({{post.comments.length}})
    </span>
</div>
```

Figura.50. Extracto del fichero index.ejs.

Fuente: Stuardo, 2016.

4.2.4. Capturas de la aplicación

Se muestra capturas de la aplicación funcional, en la figura 51 muestra la aplicación sin usuario logeado.

Figura.51. Home de la aplicación sin usuario logeado.

Fuente: Stuardo, 2016.

En la figura 52 muestra la aplicación con usuario logeado.

Figura.52. Home de la aplicación con usuario logeado.

Fuente: Stuardo, 2016.

En la figura 53 muestra el Segundo Post con comentario agregado sin usuario logeado.

Figura.53. Segundo Post con comentario agregado sin usuario logeado.

Fuente: Stuardo, 2016.

En la figura 54 muestra el Segundo Post con comentario agregado con usuario logeado.

Figura.54. Segundo Post con comentario agregado con usuario logeado.

Fuente: Stuardo, 2016.

En la figura 55 muestra la pantalla de log in.

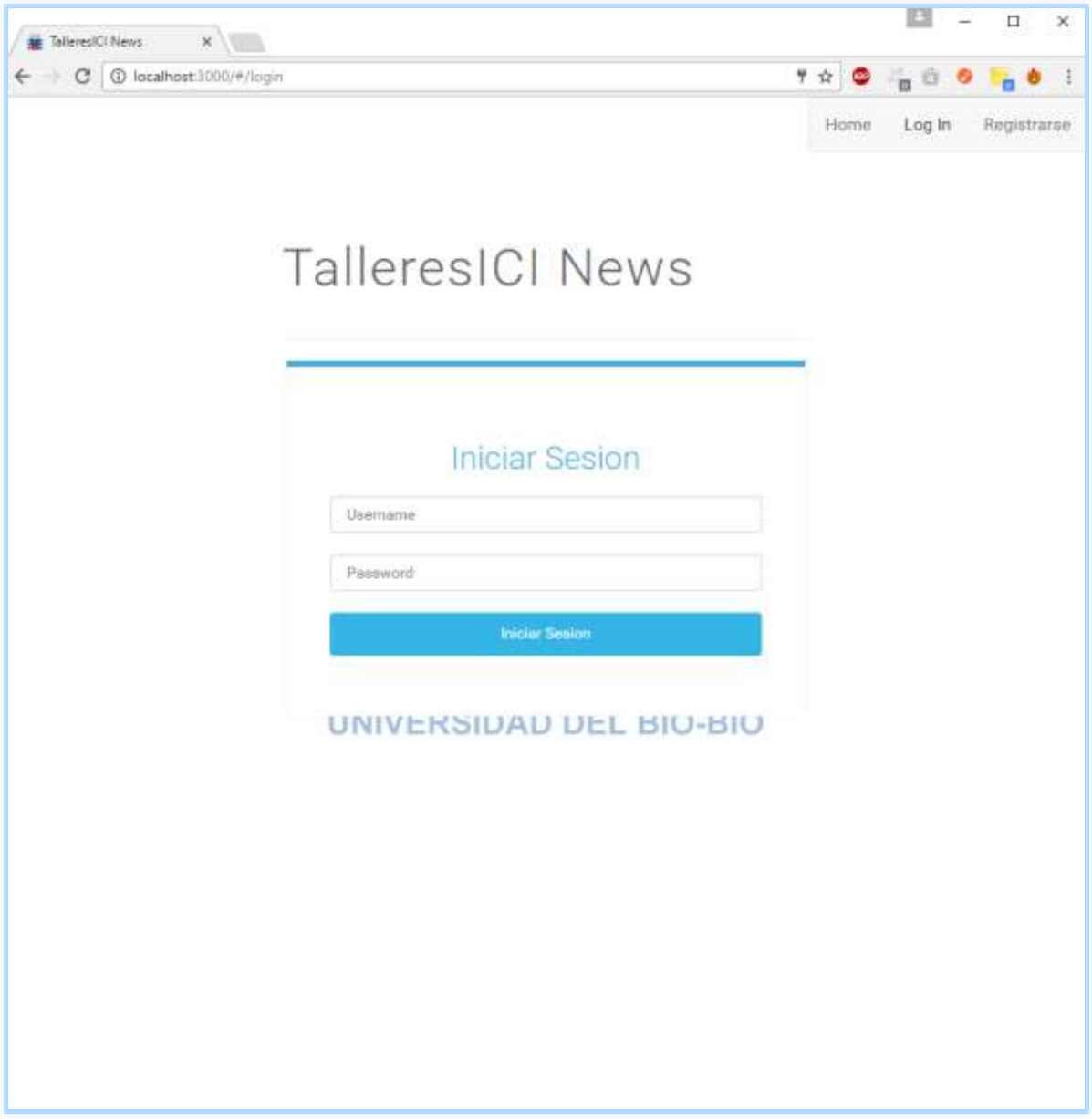


Figura.55. Pantalla de log in.

Fuente: Stuardo, 2016.

En la figura 56 muestra la pantalla de formulario de registro.

Figura.56. Pantalla de formulario de registro.

Fuente: Stuardo, 2016.

4.2.5. Resultado

Luego de estudiar y desarrollar en profundidad el framework AngularJS se concluye que la curva de aprendizaje es bastante grande. Al comienzo no es un framework intuitivo como para usar todas las variadas características que éste ofrece.

Su escalabilidad y modularidad permite que otras herramientas del mercado no tengan conflicto al funcionar con AngularJS.

La comunidad online de este framework es bastante grande, entonces para un desarrollador que se esté recién iniciando en esta herramienta, siempre va a haber variados ejemplos que le permite un aprendizaje constante.

Para finalizar, el desarrollar del prototipo de este estudio, fue rápido en un comienzo y a medida que se descubrían nuevas características la velocidad de desarrollo fue disminuyendo. La cantidad de código es muy inferior a otras herramientas, ya que los servicios que entrega AngularJS disminuyen la cantidad de código necesario.

5. Conclusiones

Al terminar este viaje en que me adentro en el desarrollo web, más específico en los frameworks cliente MVC basados en javascript puedo decir que el crecimiento que han tenido estas tecnologías en la última década ha sido muy grande, esto debido a las grandes herramientas que le entregan al desarrollador y por ende la gran comunidad que se genera entorno a ellas.

Más y más desarrolladores están aprovechando las ventajas de estos frameworks, en especial AngularJS, como se pudo observar, en el estudio realizado recopilando numerables experiencias de desarrolladores con distintos niveles de experiencia, este fue el mejor evaluado entre los tres frameworks estudiados y con muchas razones, su gran escalabilidad y modularidad ayudan al desarrollador a generar aplicación rápidamente y con la menor cantidad de código.

Es cierto que la curva de aprendizaje es pronunciada, pero los resultados obtenidos después de usar la herramienta valen la pena la espera, por último, no queda más que decir que por algo AngularJS fue patrocinado por el gigante Google y lo usa en varias de sus aplicaciones.

6. Recomendaciones

1. Al realizar un análisis comparativo entre Framework de desarrollo, se debe elegir cuidadosamente los Framework involucrados, los cuales deben tener funcionalidad y características semejantes para obtener resultados certeros sin inconvenientes.
2. El patrón de arquitectura de software Modelo, Vista, Controlador MVC puede ser usado e implementado para el desarrollo de cualquier proyecto de aplicación web, en especial aquellos que manejan una gran cantidad de datos y transacciones complejas.
3. Realizar un estudio previo al uso e implementación de cualquier Framework de desarrollo, para poder seleccionar el que mejor se acople a los requerimientos y necesidades de la aplicación web.
4. Para el desarrollo de aplicaciones web es importante que el desarrollador esté actualizando diariamente su conocimiento en nuevas mejoras o modificaciones que se le añadan al Framework de su elección.

7. Bibliografía

1. Ahmed, Z. (2014). Learning AngularJS Part 4: Modular Design of AngularJS Application. conceptf1 Recuperado de: http://conceptf1.blogspot.cl/2014/05/learning-angularjs-part5.html.
2. AngularJS. (2016). AngularJS by Google. Recuperado de: https://angularjs.org/.
3. AngularZone. (2015). AngularJs History. AngularZone Recuperado de: https://angularzone.wordpress.com/angularjs-history/.
4. BackboneJS. (2016). Backbone.JS. Recuperado de: http://backbonejs.org/.
5. Bootstrap. (2016). Designed for everyone, everywhere. Recuperado de: http://getbootstrap.com/.
6. Brad, Green; Shyam, Seshadri. (2013). *Angular JS*. Estados Unidos: O'Reilly Media Inc.
7. Branas, Rodrigo. (2014). *AngularJS Essentials*. Birmingham: Packt Publishing Ltd.
8. CodeSchool. (2014). Single-page Applications. codeschool.com Recuperado de: https://www.codeschool.com/beginners-guide-to-web-development/single-page-applications.
9. Darwin, Peter Bacon; Kozlowski, Pawel. (2013). *Mastering Web Application Development with AngularJS.* Birmingham: Packt Publishing Ltd.
10. Ember. (2016). A framework for creating ambitious web applications. Tilde INC. Recuperado de: http://emberjs.com/.
11. Express (2016). Infraestructura web rápida, minimalista y flexible para Node.js. StrongLoop INC. Recuperado de: http://expressjs.com/es/.
12. Freeman, Adam (2014). *Pro AngularJS*. Estados Unidos: Apress.
13. Ivan Alvarado. (2014). ¿Qué es Frontend Y Backend en la programación web?. serprogramador.es. Recuperado de: https://serprogramador.es/que-es-frontend-y-backend-en-la-programacion-web/.
14. Johanny Solis. (2014). ¿Qué es Bootstrap y cómo funciona en el diseño web?. arweb.com Recuperado de: http://www.arweb.com/chucherias/editorial/¿que-es-bootstrap-y-como-funciona-en-el-diseno-web.htm/.
15. JQueryUI. (2016). JQueryUI user interface. JQueryUI Recuperado de: http://jqueryui.com/.
16. Lanzarote, G. (2015). Dirección DNS Definición, Para Que Sirve. todohostingweb Recuperado de: http://www.todohostingweb.com/dns-definicion/.

17. Le Hégaret, P. (2005). Document Object Model (DOM). W3C Recuperado de: https://www.w3.org/DOM/#what.
18. Lerner, Ari (2013). *ng-book*. Estados Unidos: FULLSTACK.io.
19. Lundiak, A. (2014). History'n'Evolution of JS MV* frameworks. Work'n'Me. Recuperado: de https://worknme.wordpress.com/2014/09/25/history-and-evolution-of-js-mvc-mvv-frameworks/comment-page-1/.
20. Marques, A. (2013). Conceptos sobre APIs REST. [online] asiermarques.com. Recuperado de: http://asiermarques.com/2013/conceptos-sobre-apis-rest/.
21. Microsoft. (2015). About Native XMLHTTP. Microsoft Recuperado de: https://msdn.microsoft.com/en-us/library/ms537505(VS.85).aspx.
22. MongoDB (2016). Building on the Best of Relational with the Innovations of NoSQL. MongoDB INC. Recuperado de: https://www.mongodb.com/.
23. Mozilla. (2016). HTML. Mozilla.org Recuperado de: https://developer.mozilla.org/es/docs/Web/HTML.
24. Mozilla.org. (2016). CSS. mozilla.org Recuperado de: https://developer.mozilla.org/es/docs/Web/CSS.
25. Node.js. (2016). Node.js. Node.js Foundation. Recuperado de: https://nodejs.org/.
26. Osmani, A. (2012). Journey Through The JavaScript MVC Jungle. Smashing Magazine. Recuperado de: https://www.smashingmagazine.com/2012/07/journey-through-the-javascript-mvc-jungle/.
27. PortalProgramas. (2015). Programas online - Ayuda para programas. PortalProgramas. Recuperado de: http://www.portalprogramas.com/ayuda/c19/programas-online.
28. Porto, S. (2013). A Comparison of Angular, Backbone, CanJS and Ember. sporto.github.io Recuperado de: http://sporto.github.io/blog/2013/04/12/comparison-angular-backbone-can-ember/.
29. Pressman, R. S. (2005). Software engineering: a practitioner's approach. Palgrave Macmillan.
30. Rouse, M. (2015). Sistema de gestión de bases de datos relacionales (RDBMS). searchdatacenter.techtarget.com Recuperado de: http://searchdatacenter.techtarget.com/es/definicion/Sistema-de-gestion-de-bases-de-datos-relacionales-RDBMS.

31. Sampieri, R. H., Collado, C. F., Lucio, P. B., & Pérez, M. D. L. L. C. (1998). Metodología de la investigación (Vol. 1). México: McGraw-Hill.
32. SISDOMA. (2011). Que es un compilador. ingsistemascompilador.blogspot.cl Recuperado de: http://ingsistemascompilador.blogspot.cl/p/conceptos-basicos-sobre-compiladores.html.
33. Tagle, J. (2014). ¿Qué es Wordpress y para qué funciona?. wpavanzado.com Recuperado de: http://wpavanzado.com/que-es-wordpress/.

8. Anexo: Código fuente del caso práctico

A continuación, se presenta el código utilizado para construir la aplicación con AngularJS. En la figura 57 se presenta la primera parte del fichero index.js.

index.js

```
var mongoose = require('mongoose');// Utilizamos express
var express = require('express');// mongoose para mongodb
var router = express.Router();
var multer = require('multer');
var passport = require('passport');// passport para autenficacion
var jwt = require('express-jwt');

// Creamos las instancias de los modelos
var Post = mongoose.model('Post');
var Comment = mongoose.model('Comment');
var User = mongoose.model('User');
var Image = mongoose.model('Image');

var auth = jwt({
    secret: 'SECRET',
    userProperty: 'payload'
});

/* GET home page. */
router.get('/', function(req, res) {
    res.render('index');
});
// Obtenet todos los posts
router.get('/posts', function(req, res, next) {
    Post.find(function(err, posts) {
        if (err) {
            return next(err);
        }

        res.json(posts);
    });
});
```

Figura.57. Fichero index.js (parte 1).

Fuente: Stuardo, 2016.

En la figura 58 se presenta la segunda parte del fichero index.js.

index.js

```
// Obtener todos los posts y autentificar al usuario
router.post('/posts', auth, function(req, res, next) {
    var post = new Post(req.body);
    post.author = req.payload.username;

    post.save(function(err, post) {
        if (err) {
            return next(err);
        }

        res.json(post);
    });
});
// Define el parametro 'post'
router.param('post', function(req, res, next, id) {
    var query = Post.findById(id);

    query.exec(function(err, post) {
        if (err) {
            return next(err);
        }
        if (!post) {
            return next(new Error("Publicacion no encontrada"));
        }

        req.post = post;
        return next();
    });
});
```

Figura.58. Fichero index.js (parte 2).

Fuente: Stuardo, 2016.

En la figura 59 se presenta la tercera parte del fichero index.js.

index.js

```
// Define el parametro 'comment'
router.param('comment', function(req, res, next, id) {
    var query = Comment.findById(id);

    query.exec(function(err, comment) {
        if (err) {
            return next(err);
        }
        if (!comment) {
            return next(new Error("Comentario no encontrado"));
        }

        req.comment = comment;
        return next();
    });
});
// Define el parametro 'image'
router.param('image', function(req, res, next, id) {
    var query = Image.findById(id);

    query.exec(function(err, image) {
        if (err) {
            return next(err);
        }
        if (!image) {
            return next(new Error("Imagen no encontrada"));
        }
        req.image = image;
        return next();
    });
});
// Obtiene el post segun su id ':post'
router.get('/posts/:post', function(req, res, next) {
    req.post.populate('comments', function(err, post) {
        res.json(post);
    });
});
```

Figura.59. Fichero index.js (parte 3).

Fuente: Stuardo, C, 2016.

En la figura 60 se presenta la cuarta parte del fichero index.js.

index.js

```
// Suma un voto al post
router.put('/posts/:post/upvote', auth, function(req, res, next) {
    req.post.upvote(function(err, post) {
        if (err) {
            return next(err);
        }
        res.json(post);
    });
});
// Resta un voto al post
router.put('/posts/:post/downvote', auth, function(req, res, next) {
    req.post.downvote(function(err, post) {
        if (err) {
            return next(err);
        }
        res.json(post);
    });
});
// Agrega un comentario al post
router.post('/posts/:post/comments', auth, function(req, res, next) {
    var comment = new Comment(req.body);
    comment.post = req.post;
    comment.author = req.payload.username;

    comment.save(function(err, comment) {
        if (err) {
            return next(err);
        }
        req.post.comments.push(comment);
        req.post.save(function(err, post) {
            if (err) {
                return next(err);
            }
            res.json(comment);
        });
    });
});
```

Figura.60. Fichero index.js (parte 4).

Fuente: Stuardo, 2016.

En la figura 61 se presenta la quinta parte del fichero index.js.

index.js

```
// Suma un voto al comentario segun su id ':comment'
router.put('/posts/:post/comments/:comment/upvote', auth, function(req, res, next) {
    req.comment.upvote(function(err, comment) {
        if (err) {
            return next(err);
        }
        res.json(comment);
    });
});
// Resta un voto al comentario segun su id ':comment'
router.put('/posts/:post/comments/:comment/downvote', auth, function(req, res, next) {
    req.comment.downvote(function(err, comment) {
        if (err) {
            return next(err);
        }
        res.json(comment);
    });
});
// Agrega la imagen subida al post
router.post('/posts/:post/images', auth, function(req, res, next) {
    var image = new Image(req.path);
    image.post = req.post;
    image.author = req.payload.username;

    image.save(function(err, image) {
        if (err) {
            return next(err);
        }
        req.post.images.push(image);
        req.post.save(function(err, post) {
            if (err) {
                return next(err);
            }
            res.json(image);
        });
    });
});
```

Figura.61. Fichero index.js (parte 5).

Fuente: Stuardo, 2016.

En la figura 62 se presenta la sexta parte del fichero index.js.

index.js

```
// Regista a un usuario con su nombre y contraseña
router.post('/register', function(req, res, next) {
    if (!req.body.username || !req.body.password) {
        return res.status(400).json({
            message: 'Please fill out all fields'
        });
    }
    var user = new User();

    user.username = req.body.username;
    user.setPassword(req.body.password)

    user.save(function(err) {
        if (err) {
            return next(err);
        }
        return res.json({
            token: user.generateJWT()
        })
    });
});
// Ingresa al usuario en su cuenta
router.post('/login', function(req, res, next) {
    if (!req.body.username || !req.body.password) {
        return res.status(400).json({
            message: 'Please fill out all fields'
        });
    }
    console.log('calling passport)');
    passport.authenticate('local', function(err, user, info) {
        if (err) {
            return next(err);
        }
        if (user) {
            return res.json({
                token: user.generateJWT()
            });
        } else {
            return res.status(401).json(info);
        }
    })(req, res, next);
});
```

Figura.62. Fichero index.js (parte 6).

Fuente: Stuardo, 2016.

En la figura 63 se presenta la primera parte del fichero angularApp.js.

angularApp.js

```
var app = angular.module('talleresNews', ['ui.router']);
// Define los estados o rutas principales
// dependiento de la ruta se mostrara en el navegador
// una parte del codigo de index.ejs
app.config(['$stateProvider', '$urlRouterProvider',
    function($stateProvider, $urlRouterProvider) {
        $stateProvider.state('home', {
            url: '/posts/:id',
            templateUrl: '/posts.html',
            controller: 'PostsCtrl',
            resolve: {
                post: ['$stateParams', 'posts',
                    function($stateParams, posts) {
                        return posts.get($stateParams.id);
                    }
                ]
            }
        }).state('login', {
            url: '/login',
            templateUrl: '/login.html',
            controller: 'AuthCtrl',
            onEnter: ['$state', 'auth',
                function($state, auth) {
                    if (auth.isLoggedIn()) {
                        $state.go('home');
                    }
                }
            ]
        }).state('register', {
            url: '/register',
            templateUrl: '/register.html',
            controller: 'AuthCtrl',
            onEnter: ['$state', 'auth',
                function($state, auth) {
                    if (auth.isLoggedIn()) {
                        $state.go('home');
                    }
                }
            ]
        });
        $urlRouterProvider.otherwise('home');
    }
]);
```

Figura.63. Fichero angularApp.js (parte 1).

Fuente: Stuardo, 2016.

En la figura 64 se presenta la segunda parte del fichero angularApp.js.

angularApp.js

```
55  // Factory que se encarga de la autentificacion
56  app.factory('auth', ['$http', '$window',
108 // Factory que se encarga del CRUD del post
109 app.factory('posts', ['$http', 'auth',
204 // Controlador principal encargado de crear el posts
205 // con los datos obtenidos del usuario
206 app.controller('MainCtrl', ['$scope', 'posts', 'auth',
207     function($scope, posts, auth) {
208
209         $scope.posts = posts.posts;
210         $scope.isLoggedIn = auth.isLoggedIn;
211         $scope.title = '';
212
213         $scope.addPost = function() {
214             if ($scope.title === '') {
215                 return;
216             }
217             posts.create({
218                 title: $scope.title,
219                 link: $scope.title.trim().toLowerCase(),
220                 body: $scope.body,
221                 author: $scope.currentUser,
222             });
223             //limpiamos los valores
224             $scope.title = '';
225             $scope.body = '';
226         };
227         $scope.upvote = function(post) {
228             console.log('Upvoting:' + post.title + "votes before:" + post.upvotes);
229             posts.upvote(post);
230         };
231         $scope.downvote = function(post) {
232             posts.downvote(post);
233         };
234     }
235 ]);
```

Figura.64. Fichero angularApp.js (parte 2).

Fuente: Stuardo, 2016.

En la figura 65 se presenta la tercera parte del fichero angularApp.js.

angularApp.js

```
// Controlador encargado de creor los comentarios
// con los datos obtenidos del usuario y subir las imagenes
// a la base de datos
app.controller('PostsCtrl', ['$scope', 'posts', 'post', 'auth',
    function($scope, posts, post, auth) {
        $scope.post = post;
        $scope.isLoggedIn = auth.isLoggedIn;

        $scope.addComment = function() {
            if ($scope.body === '') {
                return;
            }
            posts.addComment(post._id, {
                body: $scope.body,
                author: 'user'
            }).success(function(comment) {
                $scope.post.comments.push(comment);
            });
            $scope.body = '';
        };
        $scope.upvote = function(comment) {
            posts.upvoteComment(post, comment);
        };
        $scope.downvote = function(comment) {
            posts.downvoteComment(post, comment);
        };
        $scope.addImage = function() {
            if ($scope.imgname === '') {
                return;
            }
            posts.addImage(post._id, {
                imgname: $scope.imgname,
                author: 'user',
                path: $scope.path
            }).success(function(image) {
                $scope.post.images.push(image);
            });
            $scope.imgname = '';
            $scope.path = '';
        };
    }
]);
```

Figura.65. Fichero angularApp.js (parte 3).

Fuente: Stuardo, 2016.

En la figura 66 se presenta la cuarta parte del fichero angularApp.js.

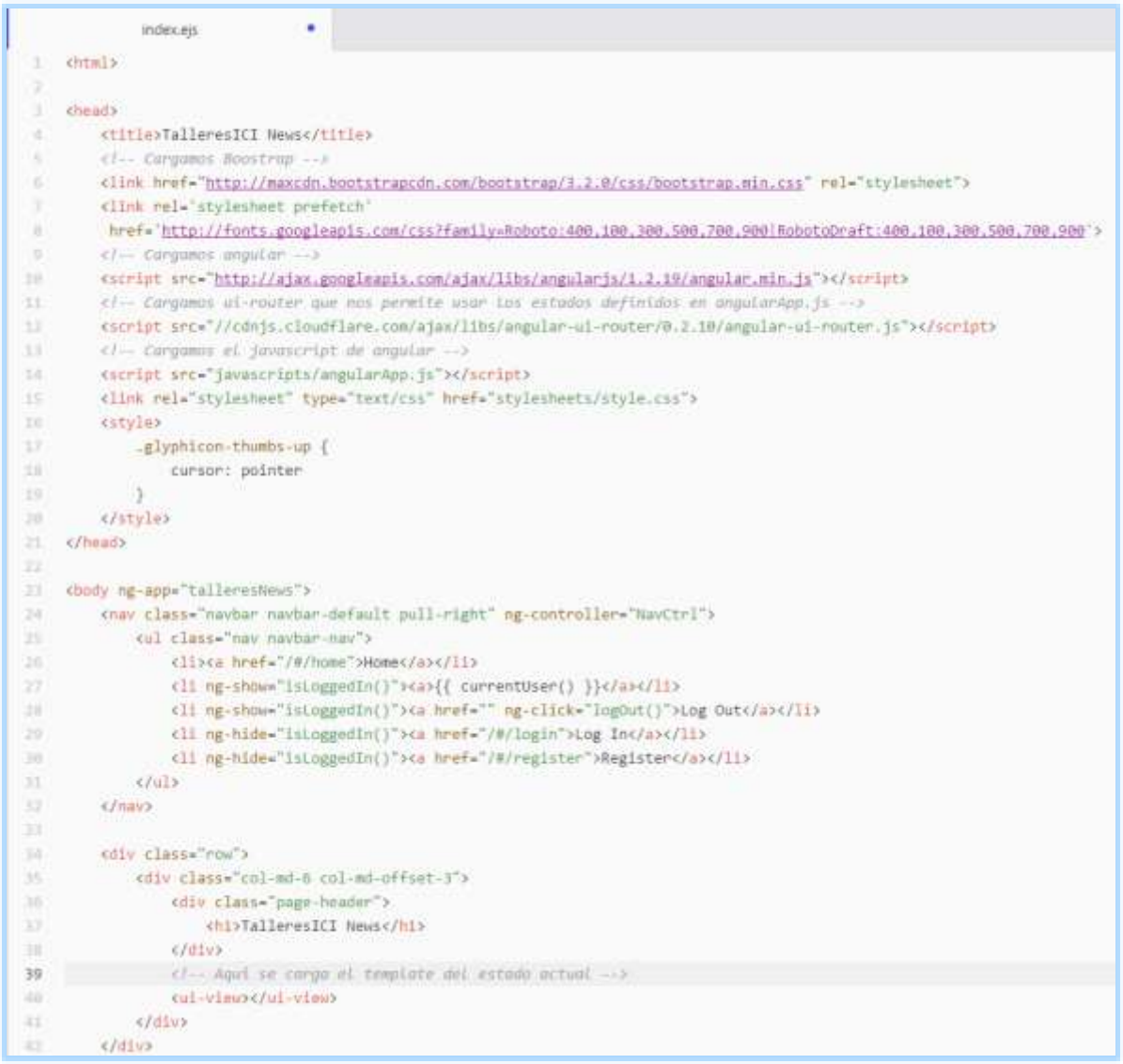
angularApp.js

```
// Controlador encargado de del registro y log in del usuario
app.controller('AuthCtrl', ['$scope', '$state', 'auth',
    function($scope, $state, auth) {
        $scope.user = {};

        $scope.register = function() {
            auth.register($scope.user).error(function(error) {
                $scope.error = error;
            }).then(function() {
                $state.go('home');
            });
        };

        $scope.logIn = function() {
            auth.logIn($scope.user).error(function(error) {
                $scope.error = error;
            }).then(function() {
                $state.go('home');
            });
        };
    }
]);

app.controller('NavCtrl', ['$scope', 'auth',
    function($scope, auth) {
        $scope.isLoggedIn = auth.isLoggedIn;
        $scope.currentUser = auth.currentUser;
        $scope.logOut = auth.logOut;
    }
]);
```

Figura.66. Fichero angularApp.js (parte 4).

Fuente: Stuardo, 2016.

En la figura 67 se presenta la primera parte del fichero index.ejs.



```
index.ejs
<html>

<head>
    <title>TalleresICI News</title>
    <!-- Cargamos Boostrap -->
    <link href="http://maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css" rel="stylesheet">
    <link rel='stylesheet prefetch'
     href='http://fonts.googleapis.com/css?family=Roboto:400,100,300,500,700,900|RobotoDraft:400,100,300,500,700,900'>
    <!-- Cargamos angular -->
    <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.min.js"></script>
    <!-- Cargamos ui-router que nos permite usar los estados definidos en angularApp.js -->
    <script src="//cdnjs.cloudflare.com/ajax/libs/angular-ui-router/0.2.10/angular-ui-router.js"></script>
    <!-- Cargamos el javascript de angular -->
    <script src="javascripts/angularApp.js"></script>
    <link rel="stylesheet" type="text/css" href="stylesheets/style.css">
    <style>
        .glyphicon-thumbs-up {
            cursor: pointer
        }
    </style>
</head>

<body ng-app="talleresNews">
    <nav class="navbar navbar-default pull-right" ng-controller="NavCtrl">
        <ul class="nav navbar-nav">
            <li><a href="/#/home">Home</a></li>
            <li ng-show="isLoggedIn()"><a>{{ currentUser() }}</a></li>
            <li ng-show="isLoggedIn()"><a href="" ng-click="logOut()">Log Out</a></li>
            <li ng-hide="isLoggedIn()"><a href="/#/login">Log In</a></li>
            <li ng-hide="isLoggedIn()"><a href="/#/register">Register</a></li>
        </ul>
    </nav>

    <div class="row">
        <div class="col-md-6 col-md-offset-3">
            <div class="page-header">
                <h1>TalleresICI News</h1>
            </div>
            <!-- Aqui se cargo el template del estado actual -->
            <ui-view></ui-view>
        </div>
    </div>
```

Figura.67. Fichero index.ejs (parte 1).

Fuente: Stuardo, 2016.

En la figura 68 se presenta la segunda parte del fichero index.ejs.

index.ejs

```
<!-- Template del estado home -->
<script type="text/ng-template" id="/home.html">
    <div ng-repeat="post in posts | orderBy: '-upvotes'">
        <span class="glyphicon glyphicon-thumbs-up" ng-click="upvote(post)">
        </span> {{post.upvotes}}
        <span class="glyphicon glyphicon-thumbs-down" ng-click="downvote(post)"></span>

        <span style="font-size:20px;margin-left:10px">
          <a ng-show="post.link" href="#/posts/{{post._id}}">{{post.title}}</a>
          <span ng-show="post.author">publicado por <a>{{post.author}}</a> |</span>
        </span>
        <span ng-show="post.date">{{post.date | date:"dd/MM/yyyy 'a las' h:mma"}} |</span>
        <span>
          <a href="#/posts/{{post._id}}">Comentarios</a>
            ({{post.comments.length}})
        </span>
    </div>
    <hr />
    <form ng-submit="addPost()" ng-show="isLoggedIn()" style="margin-top:30px">
        <h3>Agrega un nuevo post:</h3>

        <div class="form-group">
            <input type="text" class="form-control" placeholder="Titulo" ng-model="title">
        </div>

        <div class="form-group">
            <input type="textarea" class="form-control" rows="5"
            placeholder="Escriba aqui su Post" ng-model="body">
        </div>

        <div class="form-group">
            <span>Para agregar una imagen al post ingrese a este una vez creado.</span>
        </div>

        <button type="submit" class="btn btn-primary">Publicar</button>
    </form>
    <div ng-hide="isLoggedIn()">
        <h3>You need to <a href="/#/login">Log In</a> or <a href="/#/register">
          Register</a> before you can add a post.</h3>
    </div>
</script>
```

Figura.68. Fichero index.ejs (parte 2).

Fuente: Stuardo, 2016.

En la figura 69 se presenta la tercera parte del fichero index.ejs.

index.ejs

```
<!-- Template del estado post -->
<script type="text/ng-template" id="/posts.html">
    <div class="page-header">
        <h2>
          <a ng-show="post.title">{{post.title}}</a>
          <span style="font-size:18px;font-style: italic;">     publicado por <a>{{post.author}}</a></span>
        </h2>
    </div>
    <div ng-repeat="image in post.images">
        <img ng-src="image.path" alt="image.originalname" width="500">
        <span style="font-size:20px; margin-left:10px;">{{image.originalname}}</span>
    </div>
    <form ng-submit="addImage()" action="/" method="POST" enctype="multipart/form-data" style="margin-top:30px;">
        <h3>Agregar una imagen</h3>
        <input type="text" class="form-control" placeholder="Nombre Imagen" ng-model="imgname">
        <br />
        <input type="file" name="myimage" ng-model="images">
        <br />
        <button type="submit" class="btn btn-primary">Publicar</button>
    </form>
    <div>
        <hr />
        <span ng-show="post.body" style="font-size:20px;margin-left:20px">{{post.body}}</span>
        <hr />
    </div>
    <div ng-repeat="comment in post.comments | orderBy:'-upvotes'">
        <span class="glyphicon glyphicon-thumbs-up" ng-click="upvote(comment)">
        </span> {{comment.upvotes}}
        <span class="glyphicon glyphicon-thumbs-down" ng-click="downvote(comment)">
        </span> - by <a>{{comment.author}}</a>
        <span style="font-size:20px; margin-left:10px;">{{comment.body}}</span>
    </div>
    <form ng-submit="addComment()" ng-show="isLoggedIn()" style="margin-top:30px;">
        <h3>Agrega un comentario</h3>
        <div class="form-group">
            <input type="text" class="form-control" placeholder="Comentario" ng-model="body"></input>
        </div>
        <button type="submit" class="btn btn-primary">Publicar</button>
    </form>
    <div ng-hide="isLoggedIn()">
        <h3>You need to <a href="/#/login">Log In</a> or <a href="/#/register">
          Register</a> before you can comment.</h3>
    </div>
</script>
```

Figura.69. Fichero index.ejs (parte 3).

Fuente: Stuardo, 2016.

En la figura 70 se presenta la cuarta parte del fichero index.ejs.

index.ejs

```
128     <!-- Template del estado register -->
129     <script type="text/ng-template" id="/register.html">
130         <div ng-show="error" class="alert alert-danger row">
131             <span>{{ error.message }}</span>
132         </div>
133
134         <form ng-submit="register()" style="margin-top:30px;">
135             <h3>Registro</h3>
136
137             <div class="form-group">
138                 <input type="text" class="form-control" placeholder="Username"
139                 ng-model="user.username"></input>
140             </div>
141             <div class="form-group">
142                 <input type="password" class="form-control" placeholder="Password"
143                 ng-model="user.password"></input>
144             </div>
145             <button type="submit" class="btn btn-primary">Registrarse</button>
146         </form>
147     </script>
148     <!-- Template del estado login -->
149     <script type="text/ng-template" id="/login.html">
150         <div ng-show="error" class="alert alert-danger row">
151             <span>{{ error.message }}</span>
152         </div>
153         <div class="module form-module">
154             <div class="form">
155                 <form ng-submit="logIn()" style="margin-top:30px;">
156                     <h3>Iniciar Sesion</h3>
157
158                     <input type="text" class="form-control" placeholder="Username"
159                     ng-model="user.username"></input>
160                     <input type="password" class="form-control" placeholder="Password"
161                     ng-model="user.password"></input>
162
163                     <button type="submit" class="btn btn-primary">Iniciar Sesion</button>
164                 </form>
165             </div>
166         </div>
167     </script>
168 </body>
169
170 </html>
```

Figura.70. Fichero index.ejs (parte 4).

Fuente: Stuardo, 2016.