UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática
Universitat Politècnica de València

Aplicación web para el seguimiento online de lecturas basadas en el stack MEAN

Proyecto Final de Carrera

Ingeniería Técnica en Informática de Gestión

Autor: Melanie Durá Gil

Director: José Vicente Busquets Mataix

Septiembre 2015

Resumen

Este proyecto consiste en una aplicación web para la gestión personal de lecturas que permite a un usuario registrado consultar la base de datos de libros de Google y realizar ciertas acciones sobre los libros, como por ejemplo, escribir comentarios con la opinión personal para la posterior consulta de otros usuarios de la aplicación, marcar un libro como favorito o pendiente para una posterior lectura y crear listas personalizadas, entre otras cosas. Se desarrollará sobre el sistema operativo Windows 7 y a priori, podrá ser utilizado desde cualquier sistema operativo con navegador web.

Palabras clave: MEAN, NodeJS, AngularJS, JavaScript, PFC, MongoDB, ExpressJS, aplicación web, single page, JavaScript

Tabla de contenidos

I. - Introducción

1. - Introducción

El principal objetivo de este proyecto es la implementación de un sistema para el seguimiento detallado de lecturas de un usuario desarrollado bajo el conocido Stack MEAN.

El stack MEAN se basa en la utilización de cuatro componentes de software: MongoDB, ExpressJS, AngularJS y NodeJS. El uso de estas cuatro herramientas juntas permite a los desarrolladores crear aplicaciones interactivas, eficientes y bien organizadas de manera rápida.

Todos los componentes del stack usan JavaScript, con lo que podremos hacer cosas como:

– Usar JavaScript del lado del servidor (NodeJS y ExpressJS)

– Usar JavaScript en el lado del cliente (AngularJS)

– Almacenar objetos JSON en MongoDB

Para la elaboración de la memoria se ha utilizado el siguiente software:

-Procesador de textos: Microsoft Word

-Para la generación de los diagramas UML, se ha hecho uso de la web http://www.lucidchart.com, que permite generar los diagramas online y guardarlos como una imagen sin la necesidad de descargar ningún software.

2. - Motivación y objetivos

Este proyecto se propone la realización de una aplicación web que, utilizando la API de Google Books, permita al usuario mantener un registro de libros o revistas leídos. Nos ofrecerá la posibilidad de buscar un libro por título, autor o ISBN en la base de datos de Google, disponiendo así, de una enorme cantidad de información de partida.

3. - Entorno de desarrollo

Este proyecto se ha desarrollado sobre Windows 7 y puede ser utilizado desde cualquier navegador web, ya que cumple los estándares bajo los que éstos trabajan.

II. - Especificación de requisitos

1. - Introducción

1.1. - Propósito

El propósito de esta especificación de requisitos es establecer, formalizar y definir con exactitud la funcionalidad de la aplicación web que se va a desarrollar. Esto incluye objetos de los que se compone, así como la relación entre ellos y sus restricciones.

Esta especificación de requisitos está redactada siguiendo el patrón del estándar IEEE-STD-830-1998 [5].

1.2. - Alcance

Desarrollaremos una aplicación llamada Mi Biblioteca utilizando el stack MEAN, el cual se basa en el uso de JavaScript para el desarrollo de toda la aplicación, tanto la parte de front-end como la parte de back-end.

El software permitirá al usuario llevar un seguimiento de sus lecturas, consultar reseñas de libros, opinar sobre lecturas, crear listas de libros personalizadas, entre otras cosas.

Los usuarios se darán de alta en el sistema y tendrán acceso a un buscador que conecta con la API de Google Books. Los usuarios podrán buscar libros por autor, por título o por ISBN. Una vez encontrado el libro que se estaba buscando, se podrá ver toda su información en una ficha, marcar el libro como leído, favorito, pendiente o añadirlo a una o más listas. Además, el usuario podrá compartir su opinión sobre un libro de modo que el resto de usuarios de la aplicación puedan tener estos comentarios en cuenta a la hora de elegir su próxima lectura.

El objetivo de esta aplicación es ofrecer a los usuarios la posibilidad de mejorar el seguimiento de sus lecturas, así como consultar opiniones de otros usuarios.

1.3. - Definiciones, siglas y abreviaciones

PFC: Proyecto final de carrera.

Framework: es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, que puede servir de base para la organización y desarrollo de software. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

MVC: Modelo-Vista-Controlador, es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

API: Interfaz de programación de aplicaciones (*Application Programming Interface*), es el conjunto de subrutinas, funciones y procedimientos o métodos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Git: es un software de control de versiones diseñado por Linus Torvalds, pensado en la eficiencia y confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.

XML: *eXtensible Markup Language* (Lenguaje de marcas extensible), es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos de forma legible.

JSON: *JavaScript Object Notation,* es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de JavaScript que no requiere el uso de XML.

SQL: *Structured Query Language,* es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas. Una de sus características es el manejo del álgebra y el cálculo relacional que permiten efectuar consultas con el fin de recuperar, de forma sencilla, información de bases de datos, así como hacer cambios en ellas.

NoSQL: es una amplia clase de sistemas de gestión de bases de datos que difieren del modelo clásico del sistema de gestión de bases de datos relacionales en aspectos importantes, el más destacado es que no usan SQL como el principal lenguaje de consultas. Los datos almacenados no requieren estructuras fijas como tablas, normalmente no soportan operaciones JOIN, ni garantizan completamente ACID (atomicidad, consistencia, aislamiento y durabilidad), y habitualmente escalan bien horizontalmente. Los sistemas NoSQL se denominan a veces "no solo SQL" para subrayar el hecho de que también pueden soportar lenguajes de consulta de tipo SQL.

JavaScript: es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.

HTML: *HyperText Markup Language* (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia para la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, videos, entre otros. Es un estándar a cargo de la W3C, organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web, sobre todo en lo referente a su escritura e interpretación.

CSS: *Cascading style sheet* (Hoja de estilo en cascada) es un lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML o XML. La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.

AngularJS: Angular es un framework basado en JavaScript para la parte cliente o front-end de una aplicación web, que respeta el paradigma MVC y permite crear Single-Page Applications(Aplicaciones web que no necesitan recargar la página), de manera más o

menos sencilla. Es un proyecto mantenido por Google y que actualmente está muy en auge.

NodeJS: Node es un entorno de programación en JavaScript para el back-end basado en el motor V8 de JavaScript del navegador Google Chrome y orientado a eventos, no bloqueante, lo que lo hace muy rápido a la hora de crear servidores web y emplear tiempo real. Fue creado en 2009 y aunque aún es joven, las últimas versiones lo hacen más robusto además de la gran comunidad de desarrolladores que posee. No solo se utiliza en servidor, se ha extendido tanto que se emplea en Stylus, un preprocesador CSS, en Grunt un gestor de tareas basado en JavaScript y en varias cosas más como Test, etc.

ExpressJS: Express es un framework por encima de Node.js que permite crear servidores web y recibir peticiones HTTP de una manera sencilla, lo que permite también crear APIs REST de forma rápida.

MongoDB: es un sistema de base de datos NoSQL orientado a documentos, desarrollado bajo el concepto de código abierto. Mongo guarda los datos en documentos tipo JSON (*JavaScript Object Notation*) pero en forma binaria (BSON) para hacer la integración de una manera más rápida. Se pueden ejecutar operaciones en JavaScript en su consola en lugar de consultas SQL. Además tiene una gran integración con Node.js con los driver propios y con Mongoose. Debido a su flexibilidad es muy escalable y ayuda al desarrollo ágil de proyectos web.

1.4. - Referencias

-IEEE-STD-830-1998

-Wikipedia

-https://nodejs.org

-https://www.mongodb.org

-http://expressjs.com/es

-https://angularjs.org

1.5. - Visión global

Esta especificación de requisitos seguirá la siguiente estructura: a continuación se muestra una descripción global de la aplicación, tras la cual se especificarán en profundidad todas las funcionalidades de la aplicación. Para ello utilizaremos una estructura definida por los distintos objetos que representan el producto a desarrollar, así como las distintas funcionalidades que afectan a éstos.

2. - Descripción general

El producto que se va a desarrollar es una aplicación web para el seguimiento personal de lecturas. La aplicación permitirá al usuario mantener un registro de libros leídos. Ofrecerá la posibilidad de buscar un libro por título, autor o ISBN a través de consultas a la API de Google Books y acceder así a una ficha con información detallada sobre el volumen. Un usuario podrá hacer comentarios sobre un libro y puntuarlo.

2.1. - Funciones del producto

En esta sección se enumeran las funcionalidades más importantes de la aplicación.

- Buscador de libros: el usuario podrá buscar libros de forma fácil e intuitiva. Según el criterio de búsqueda (autor, título, ISBN) la aplicación devolverá un listado de libros que cumplan con dicho criterio.

- Crear listas: los usuarios podrán crear listas personalizadas de libros.

- Información de un libro: la aplicación proporcionará información básica sobre un libro como su descripción, título, autor, puntuación de los lectores y comentarios de éstos.

- Marcar libros favoritos: los usuarios podrán marcar sus libros favoritos para tenerlos más accesibles.

- Asignar estado a libros: desde la aplicación, también se podrá asignar el estado pendiente o leído a los libros.

- Añadir amigos: los usuarios podrán añadir amigos para intercambiar información sobre sus lecturas.

3. - Requisitos específicos

3.1. - Interfaces externas

Como toda interfaz de usuario, ésta debe ser intuitiva, rápida y fácil de usar en la medida de lo posible.

Para hacerla visualmente atractiva y ágil, emplearemos un módulo de AngularJS para aplicar el paradigma de Material Design. Este diseño está basado en objetos materiales y es una manera de intentar aproximarse a la realidad jugando con la luz y las sombras. Es un diseño con una tipografía clara y colores e imágenes llamativos.

En la siguiente imagen, podemos apreciar el diseño de Material utilizado en Inbox, el nuevo servicio de correo electrónico de Google.

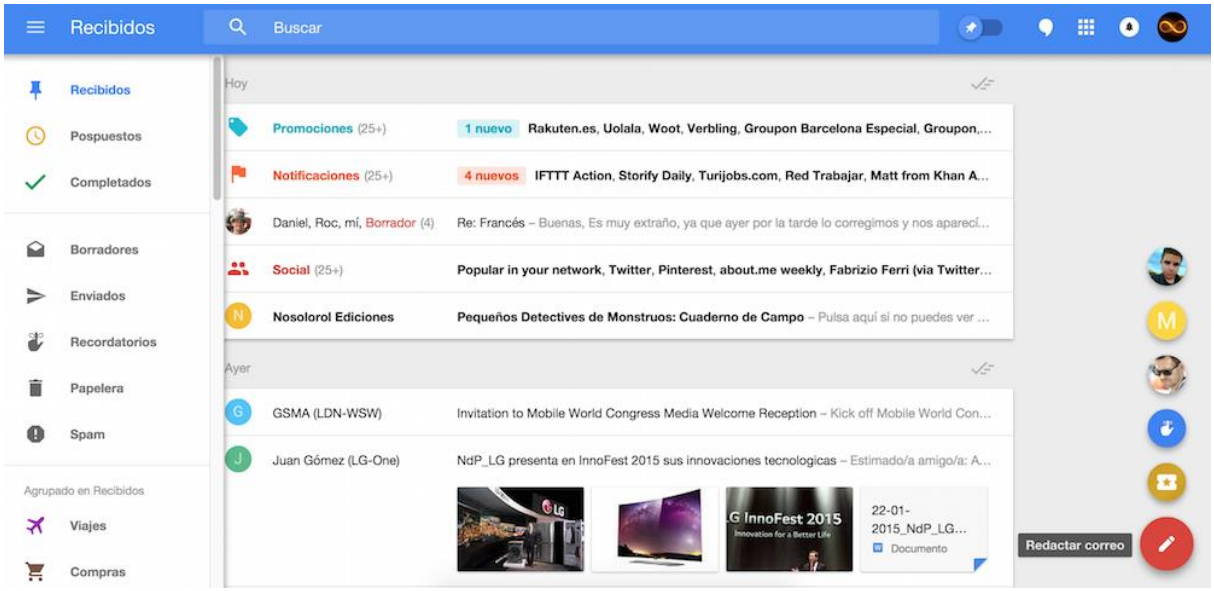


Figura 2.1.- Inbox. Ejemplo de uso de Material Design.

3.2. - Requisitos funcionales

Estos requisitos describen las funcionalidades de la aplicación. Cada funcionalidad consta de un título, una introducción, unas entradas, un proceso y sus salidas.

Registro de usuario	
Introducción	Al iniciar la aplicación, podremos registrarnos como nuevo usuario, o hacer login con un usuario existente.
Entradas	Se requerirá la siguiente información para el registro de un usuario: • nombre de usuario o *nickname* • nombre • apellidos • contraseña • confirmación de contraseña • email • avatar o imagen de perfil Todos los campos mencionados son obligatorios excepto la imagen para el perfil.
Proceso	La aplicación mostrará un formulario con los campos a rellenar y un botón de envío del formulario. El usuario deberá rellenar los datos y hacer clic sobre el botón de enviar el formulario para crear la cuenta.
Salidas	Si los datos introducidos son correctos y el usuario no existe en nuestra base de datos, iremos a una pantalla de confirmación de registro. Si los datos introducidos son incorrectos o el usuario ya existe, se mostrará un mensaje de error.

Buscador de libros	
Introducción	Este buscador realizará una llamada a la API de Google Books proporcionándole una cadena de texto y el criterio de búsqueda.
Entradas	Cadena de texto para la búsqueda y criterio (búsqueda por autor, título, editorial).
Proceso	Al hacer clic sobre el cuadro de búsqueda, el usuario podrá introducir el texto con el que se realizará la llamada a la API de Google Books. El usuario deberá introducir también el criterio de búsqueda, de modo que el filtrado se hará por dicho criterio. Si el usuario no selecciona ningún criterio de búsqueda, no se aplica ningún criterio de búsqueda, con lo que la búsqueda se realiza en todos los campos de un libro.
Salidas	Listado de libros que cumplen con los criterios de búsqueda.

Ficha de libro	
Introducción	Se mostrará en la pantalla toda la información del libro seleccionado. • Título • Autor/es • Descripción • Fecha de publicación • Nota media • Estado del libro • Es o no favorito • Imagen de la portada • Comentarios
Entradas	Clic sobre el título del volumen.
Proceso	Se mostrará en la pantalla la siguiente información: el título, una imagen que será una miniatura de la portada del libro, autor o autores, una breve descripción, la fecha de publicación, la nota media de los usuarios de la aplicación, el estado actual del libro por el usuario, si es o no es favorito y los comentarios de todos los usuarios de la aplicación con respecto al libro.
Salidas	Página con toda la información del libro solicitada.

Compartir	
Introducción	Compartir acción en redes sociales
Entradas	Clic en botón de compartir.
Proceso	Al hacer clic sobre el botón de compartir, abriremos un modal solicitando acceso a la red social seleccionada (Google+ o Twitter)
Salidas	Tras la publicación en la red social, volvemos a la ficha de libro.

Marcar libro como favorito	
Introducción	Cuando el usuario encuentre un libro de su agrado, la aplicación le permitirá marcarlo como favorito.
Entradas	Clic en el botón "Añadir a favoritos"

Proceso	Al pulsar el botón de "Añadir a favoritos", marcaremos el libro como favorito.
Salidas	El botón "Añadir a favoritos" pasará ahora a llamarse "Eliminar de favoritos".

Eliminar libro de favoritos	
Introducción	Eliminar libro de libros favoritos
Entradas	Clic en el botón "Eliminar de favoritos"
Proceso	Al pulsar el botón de "Eliminar de favoritos", desmarcaremos el libro como favorito.
Salidas	El botón "Eliminar de favoritos" pasará ahora a llamarse "Añadir a favoritos".

Login	
Introducción	Acceso a la aplicación como usuario ya registrado.
Entradas	Nombre de usuario y contraseña.
Proceso	Validación del usuario al hacer clic en el botón de login.
Salidas	Si el usuario está registrado y la contraseña es correcta, la aplicación nos redirigirá al buscador de libros. En caso contrario, permanecerá en la pantalla de login con un mensaje indicando que los datos introducidos son incorrectos.

Comentar un libro	
Introducción	Comentario personal del usuario sobre un libro.
Entradas	Título del comentario y texto.
Proceso	Un comentario constará de un título y el texto. El usuario debe rellenar estos campos y pulsar el botón de enviar para publicar el comentario.

Salidas	Si los campos no están vacíos, iremos al final de la ficha del libro donde aparecerá el comentario.

Eliminar un comentario	
Introducción	Un usuario podrá eliminar un comentario sobre un libro siempre y cuando sea él el autor del comentario.
Entradas	Botón de eliminar comentario.
Proceso	Eliminar comentario al hacer clic sobre el botón.
Salidas	Ficha de libro.

Asignar estado a libro.	
Introducción	Un usuario podrá asignar cualquiera de los siguientes estados a un libro: leído, pendiente o ninguno.
Entradas	Selector con los tres estados.
Proceso	Clic sobre uno de los estados y clic en guardar.
Salidas	Ficha de libro.

Puntuar libro	
Introducción	El usuario podrá dar una nota al libro entre 0 y 5
Entradas	Selector con números.
Proceso	Clic en número y botón de guardar.
Salidas	Ficha de libro.

Crear una lista	
Introducción	Creación de listas personalizadas.
Entradas	Clic en botón de crear lista.
Proceso	Al hacer clic en el botón de crear lista, iremos a un formulario con el campo de texto "Título" y un botón de guardar.
Salidas	Cuando hacemos clic en guardar lista, si el campo título no está vacío, se guarda la lista y volvemos a la página anterior.

Añadir un libro a una lista.	
Introducción	Añadir libros a las listas personalizadas.
Entradas	En la ficha de un libro, clic sobre botón de añadir libro a lista.
Proceso	Al hacer clic sobre el botón de añadir libro a lista, desplegaremos un selector con todas las listas del usuario. Para añadir el libro a una lista, haremos clic sobre el nombre de ésta.
Salidas	Ficha de libro.

Listas personalizadas de un usuario	
Introducción	Catálogo de listas de un usuario.
Entradas	Clic en "Ver mis listas"
Proceso	Al hacer clic sobre el botón "Ver mis listas" accedemos a un catálogo con todas las listas del usuario.
Salidas	Catálogo de listas del usuario. De cada lista veremos el nombre, el número de libros que contiene y un botón de eliminar lista.

Lista	
Introducción	El usuario podrá ver en todo momento los libros que contiene una determinada lista.
Entradas	Clic sobre el nombre de la lista.

Proceso	Al hacer clic sobre el nombre de la lista navegaremos a una página con todos los libros de la lista.
Salidas	Tabla de libros pertenecientes a la lista conformado por los siguientes campos: • Imagen de portada • Título • Autor/es • Botón para eliminar libro de la lista

Eliminar libro de una lista	
Introducción	Eliminar libro de una lista.
Entradas	Botón de eliminar libro de lista.
Proceso	Al hacer clic sobre el botón de eliminar libro de lista, se actualizará la información de la tabla.
Salidas	Tabla los libros de la lista.

Eliminar lista	
Introducción	Eliminar lista personalizada de usuario
Entradas	Clic en el botón de eliminar lista del catálogo.
Proceso	Al hacer clic sobre el botón de eliminar lista, actualizaremos la información del catálogo.
Salidas	Catálogo de listas actualizado.

Perfil de usuario	
Introducción	Datos personales e información del usuario
Entradas	Clic sobre "Mi perfil"
Proceso	Al hacer clic sobre "Mi perfil" navegamos a una nueva página con la información del usuario logueado.

Salidas	Página con la siguiente información sobre el usuario: • Nombre • Apellidos • Nombre de usuario • Imagen de perfil • Email • Fecha de registro La página constará también de un botón de cerrar sesión, un botón de editar perfil y otro de eliminar cuenta.

Editar perfil de usuario	
Introducción	Editar datos de un usuario.
Entradas	Formulario con la siguiente información: • Nombre • Apellidos • Nombre de usuario • Imagen de perfil • Email Botón de guardar cambios.
Proceso	Al hacer clic sobre el botón de guardar cambios, se validarán los campos.
Salidas	Si los datos son correctos volveremos a la página de perfil. Si los datos no son correctos, marcaremos sobre el formulario los campos incorrectos y permaneceremos en la misma página.

Baja usuario	
Introducción	Eliminar usuario del sistema.
Entradas	Clic en el botón "Baja de usuario"
Proceso	Al hacer clic en el botón de baja, eliminaremos al usuario de nuestra base de datos.
Salidas	Pantalla de bienvenida para hacer login o registrarse.

Listado de usuarios de la aplicación	
Introducción	Listado de todos los usuarios registrados en la aplicación
Entradas	Clic sobre "Ver otros usuarios"
Proceso	Al hacer clic sobre el botón de "Ver otros usuarios" navegaremos a un listado con todos los usuarios de la aplicación.
Salidas	Tabla con todos los usuarios de la aplicación. De cada usuario mostraremos su imagen de perfil y su nombre. Para cada usuario tendremos un botón para añadirlo a nuestros amigos.

Añadir amigo	
Introducción	El usuario tiene la opción de añadir como amigos a otros usuarios de la aplicación.
Entradas	Botón de añadir amigo.
Proceso	Cuando hacemos clic sobre añadir amigo, se añade el usuario a nuestra lista de amigos pendientes de confirmación.
Salidas	Listado de usuarios de la aplicación.

Listado de peticiones de amistad	
Introducción	Listado de peticiones de amistad de otros usuarios.
Entradas	Clic en botón de "Ver peticiones de amistad"
Proceso	Al hacer clic sobre el botón de ver peticiones de amistad, navegamos a una página con un listado de usuarios que han solicitado nuestra amistad.
Salidas	Listado de solicitudes de amistad

Aceptar amigo	
Introducción	Un usuario debe decidir si quiere ser amigo de otro o no.
Entradas	Listado de peticiones de amistad. Clic en “aceptar amigo”.
Proceso	Para cada elemento del listado, es decir, para cada petición, tendremos dos botones. Un botón para aceptar la petición y otro para rechazarla.
Salidas	Listado de amigos.

Rechazar amigo	
Introducción	Rechazar la petición de amistad de otro usuario de la aplicación
Entradas	Listado de peticiones de amistad. Clic en rechazar petición de amistad.
Proceso	Para cada elemento del listado, es decir, para cada petición, tendremos dos botones. Un botón para aceptar la petición y otro para rechazarla.
Salidas	Listado de amigos.

Eliminar amigo	
Introducción	Eliminar amigo previamente aceptado.
Entradas	Clic en botón de eliminar amigo del listado de amigos.
Proceso	Al hacer clic sobre el botón de eliminar amigo, eliminamos al usuario de nuestro listado de amigos y mostramos el listado actualizado.
Salidas	Listado de amigos actualizado.

Listado de amigos	
Introducción	Tabla con los usuarios amigos.
Entradas	Clic sobre "Ver amigos"
Proceso	Cuando hacemos clic sobre el botón de "Ver amigos" navegamos a una página con un listado de todos nuestros amigos.
Salidas	Listado con los amigos del usuario registrado. Para cada amigo, tendremos un botón de eliminar amigo.

Cerrar sesión	
Introducción	Salir de la aplicación.
Entradas	Clic en botón de "Cerrar sesión"
Proceso	Al hacer clic sobre Cerrar sesión, borraremos las cookies del navegador.
Salidas	Pantalla de bienvenida.

3.3. - Rendimiento

El rendimiento de la aplicación dependerá de varios factores. Uno de ellos será el servidor donde esté alojada la aplicación. Otro factor será el rendimiento que promedie el equipo del usuario. Y por último, el tiempo de respuesta de las consultas a la API de Google Books.

Los esfuerzos requeridos por el servidor ante las peticiones de los usuarios no serán de alto nivel debido a la arquitectura de la aplicación.

Por otro lado, los usuarios experimentarán un mejor rendimiento cuanto mayor sea la potencia de la máquina con la que interactúen con la aplicación.

La aplicación ha sido probada en varios equipos de distintas prestaciones y en distintos navegadores y versiones de éstos y su rendimiento ha sido óptimo.

No obstante, es posible que bajo navegadores Internet Explorer inferiores a la versión 10, la aplicación presente ciertos problemas, tanto de lentitud como de renderización de algunos elementos debido a que estas versiones del navegador no soportan algunas especificaciones tanto de Bootstrap como de JavaScript.

3.4. - Portabilidad

La aplicación presenta una portabilidad absoluta, esto es debido a que se va a desarrollar una aplicación web y por tanto, puede ejecutarse en cualquier equipo que disponga de navegador web y cumpla con los requisitos básicos.

El uso de la aplicación desde dispositivos móviles no supone ningún problema puesto que se ha implementado bajo *responsive design*.

III. - Análisis

1. - Introducción

En esta sección de la memoria, se utilizan modelos que muestran los aspectos generales de la aplicación a desarrollar basándonos en la especificación de requisitos del capítulo anterior.

Concretamente, expondremos algunos de los diagramas UML más importantes de la aplicación.

2. - Diagrama de clases

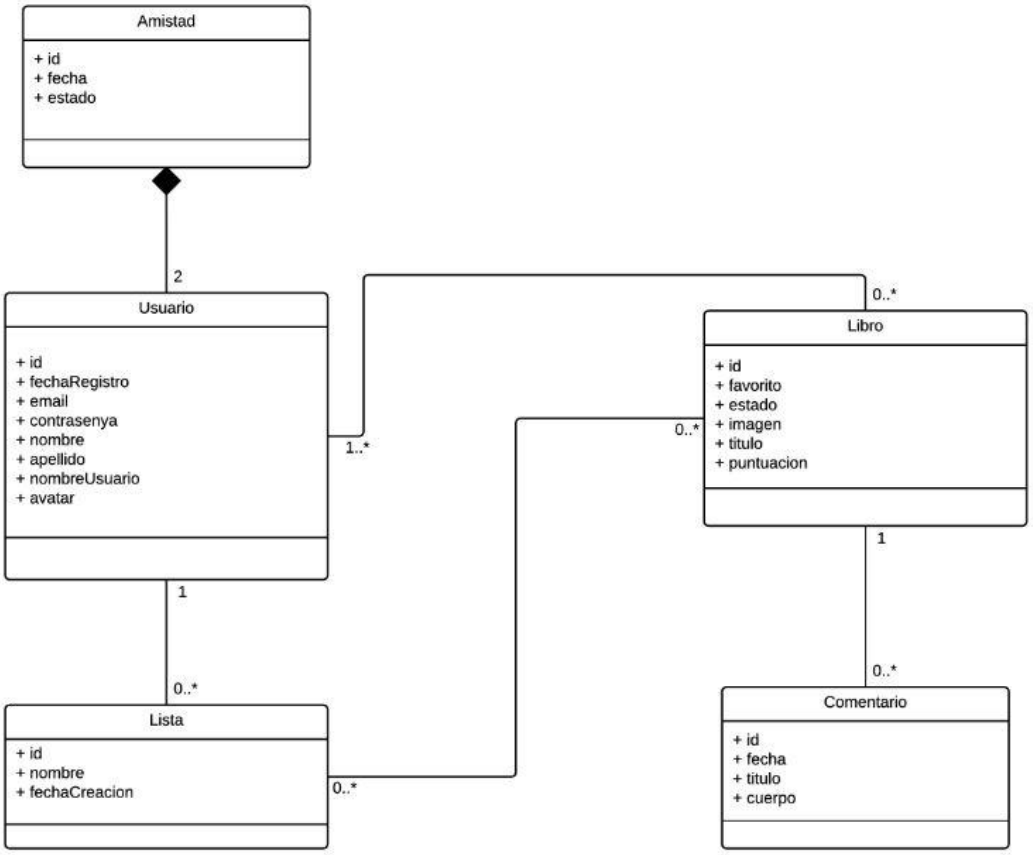


Figura 3.1.- Diagrama de clases de la aplicación Mi Biblioteca.

3. - Diagramas UML

3.1. - Actores de la aplicación

En la siguiente figura, vemos el diagrama correspondiente a los actores o usuarios de la aplicación. Existen dos tipos de actores. El primer actor es un usuario no registrado, y el segundo actor, el usuario registrado.

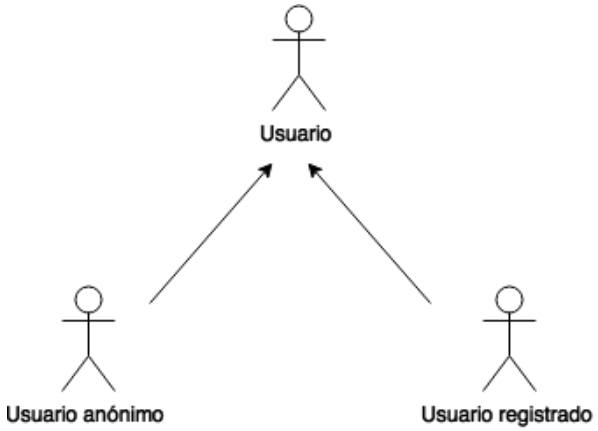


Figura 3.2. – Actores de la aplicación.

3.1.1. - Usuario anónimo

Para poder hacer uso de nuestra aplicación web, un usuario ha de registrarse en el sistema. Si un usuario no ha efectuado el registro, se le considera usuario anónimo, y por tanto, la única acción que podrá realizar en la aplicación será el registro.

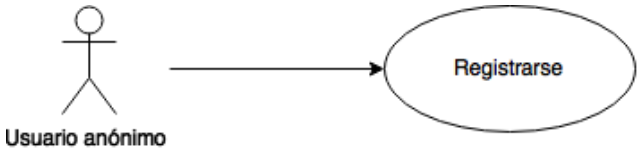


Figura 3.3. – Usuario anónimo.

3.1.2. - Usuario registrado

A continuación se muestra un diagrama general de casos de uso de un usuario registrado:

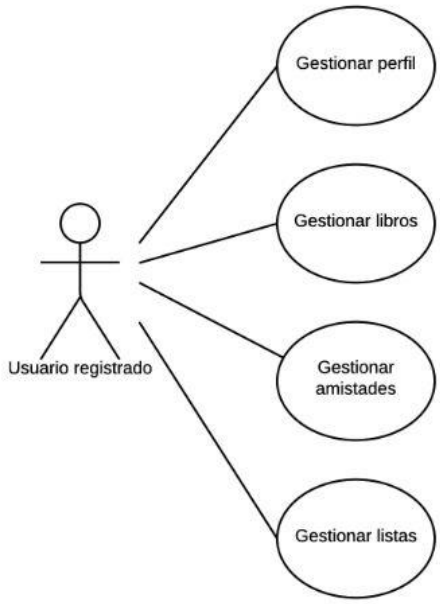


Figura 3.4. – Usuario registrado.

3.2. - Diagramas de casos de uso

3.2.1. - Login

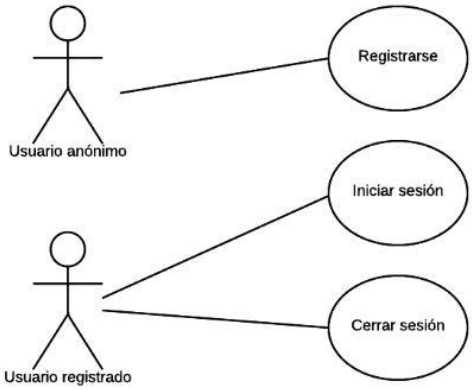


Figura 3.5. – Usuarios de la aplicación. Login y registro.

3.2.2. - Perfil

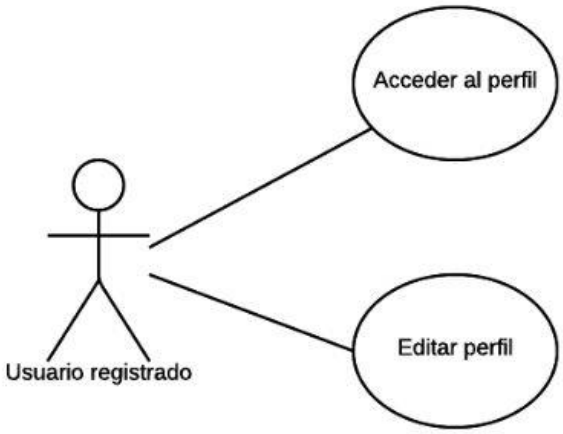


Figura 3.6. – Usuario registrado. Casos de uso. Perfil.

3.2.3. - Libros

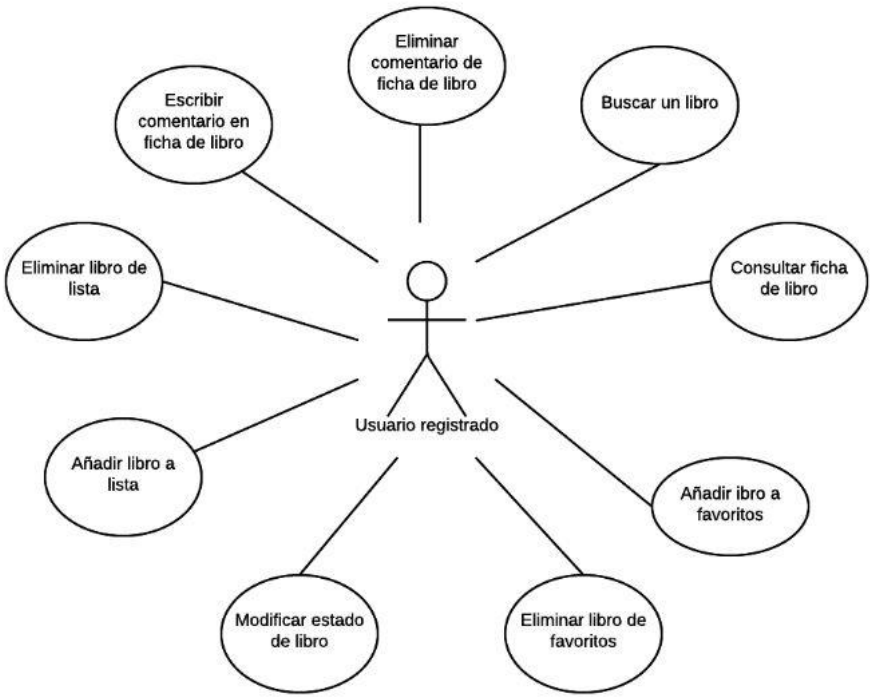
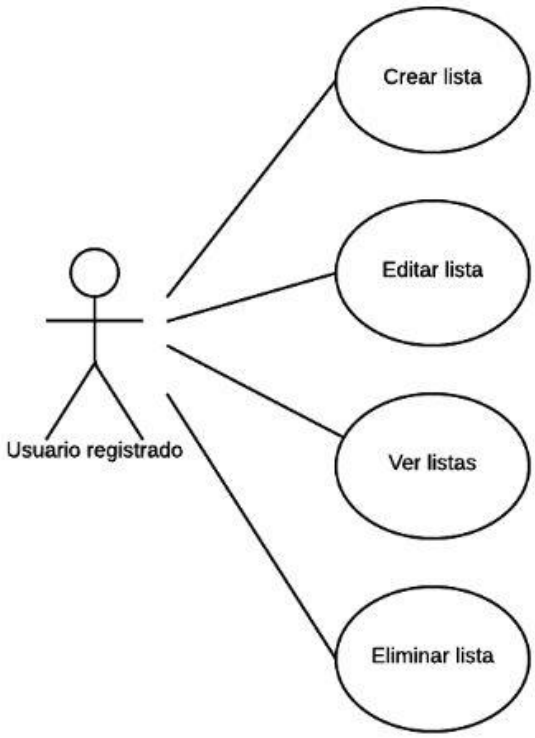


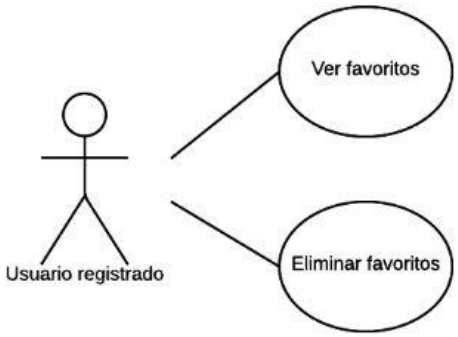
Figura 3.7. – Usuario registrado. Casos de uso. Libros.

3.2.4. - Listas



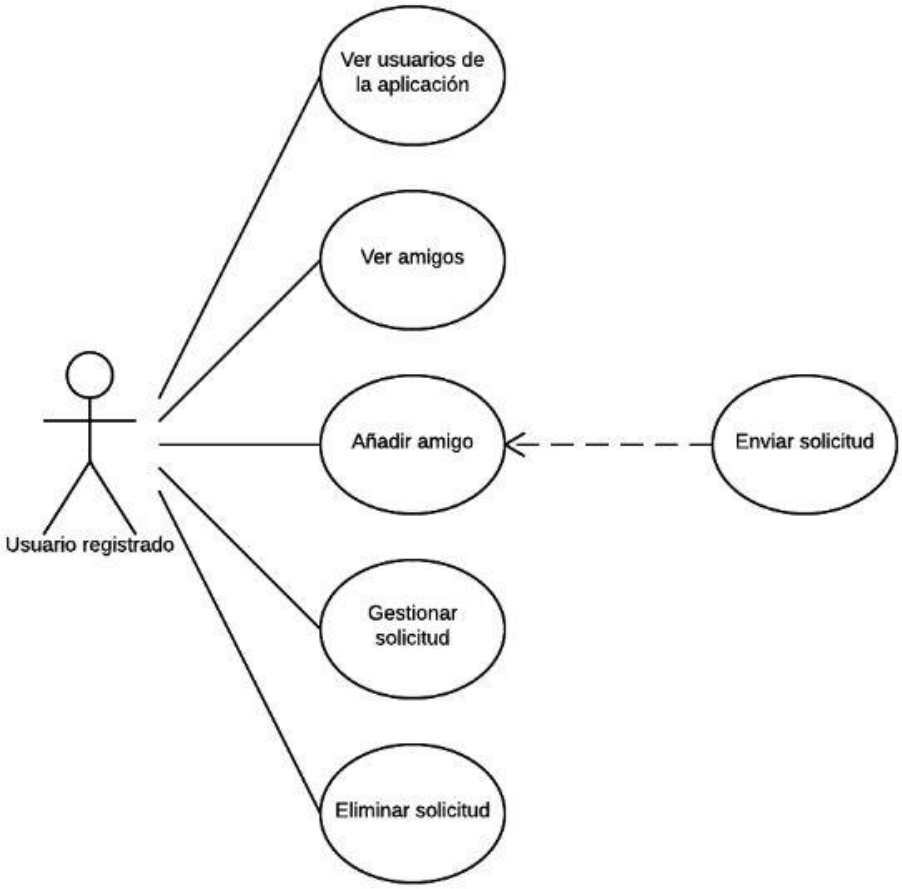
3.8. – Usuario registrado. Casos de uso. Listas.

3.2.5. - Favoritos



3.9. – Usuario registrado. Casos de uso. Favoritos.

3.2.6. - Usuarios

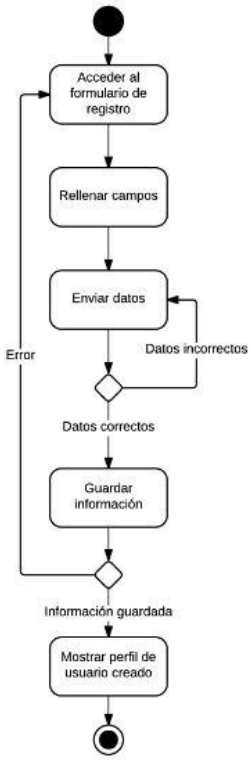


3.10. – Usuario registrado. Casos de uso. Acciones sobre usuarios.

3.3. - Diagramas de actividad

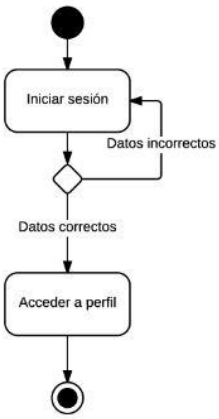
A continuación mostraremos los diagramas de actividad más importantes de la aplicación.

3.3.1. - Registro de usuario



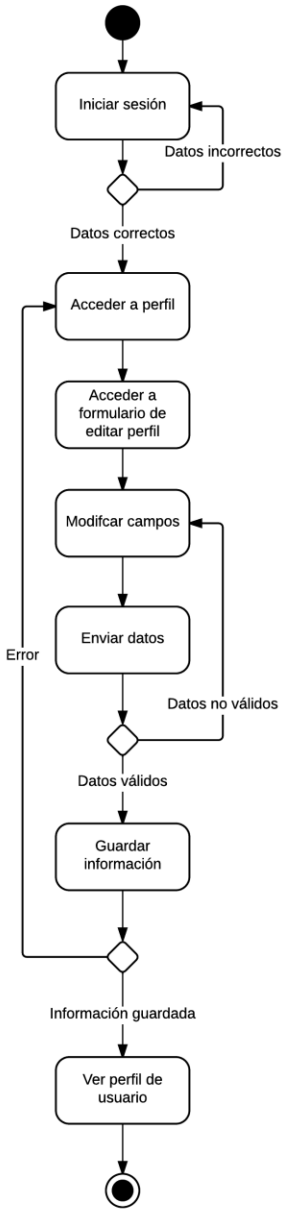
3.11.- Diagrama de actividad. Registro de usuario.

3.3.2. - Ver perfil



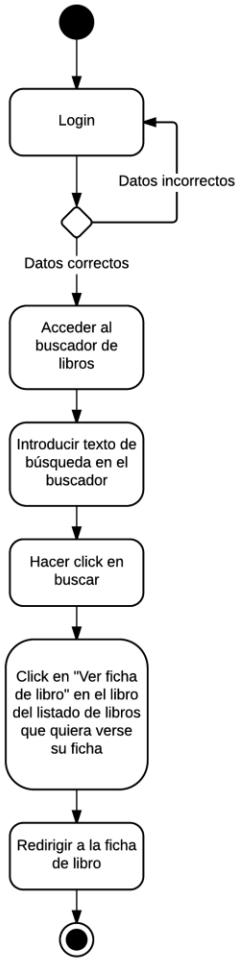
3.12.- Diagrama de actividad. Ver perfil.

3.3.3. - Editar perfil



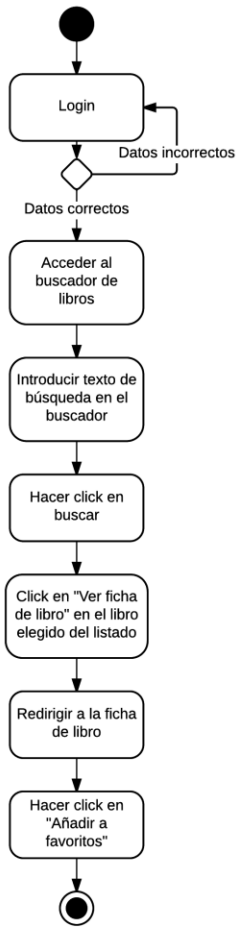
3.13.- Diagrama de actividad. Ver perfil de usuario.

3.3.4. - Ver ficha de libro



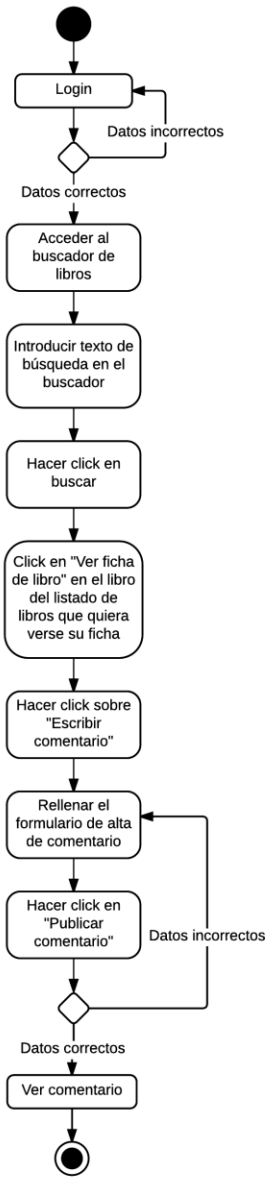
3.14.- Diagrama de actividad. Ver ficha de libro.

3.3.5. - Marcar libro como favorito



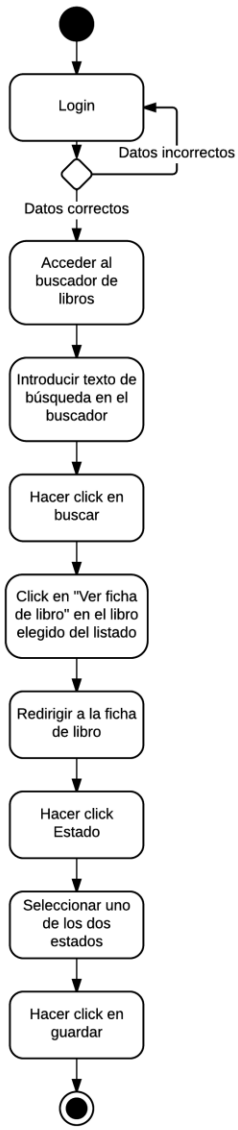
3.15.- Diagrama de actividad. Marcar libro como favorito.

3.3.6. - Escribir comentario sobre un libro



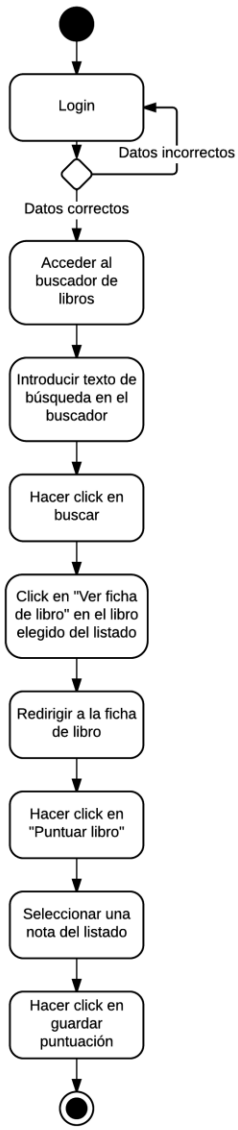
3.16.- Diagrama de actividad. Escribir comentario sobre un libro.

3.3.7. - Asignar estado a libro



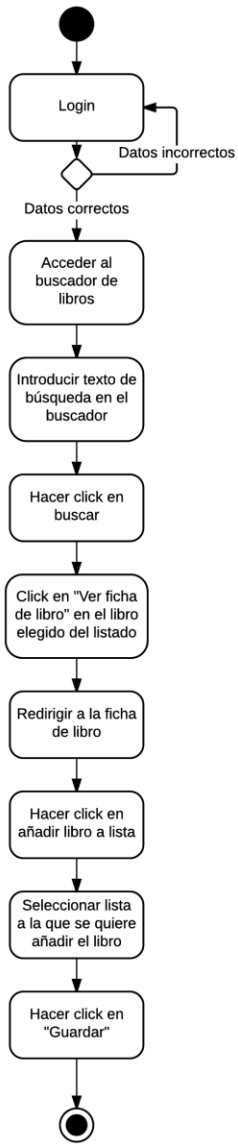
3.17.- Diagrama de actividad. Asignar estado a libro.

3.3.8. - Puntuar un libro



3.18.- Diagrama de actividad. Puntuar un libro.

3.3.9. - Añadir libro a lista

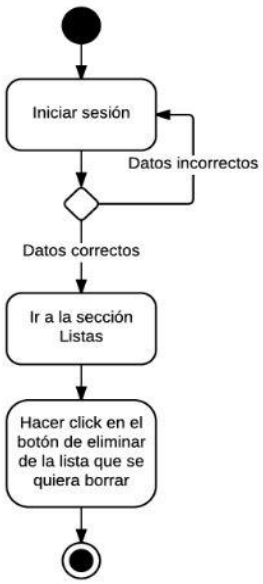


3.19.- Diagrama de actividad. Añadir libro a lista.

3.3.10. - Editar una lista

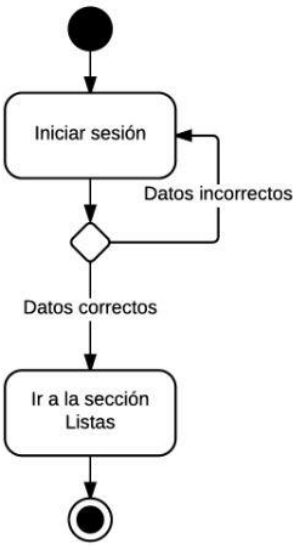
3.20. – Diagrama de actividad. Editar una lista.

3.3.11. - Eliminar una lista



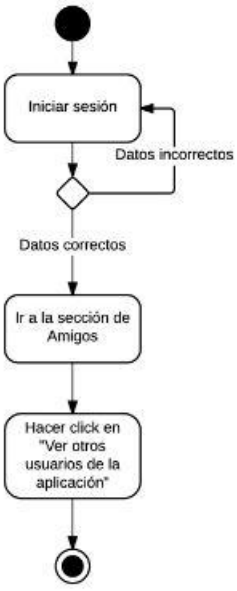
3.21. – Diagrama de actividad. Eliminar una lista.

3.3.12. - Ver listado de listas



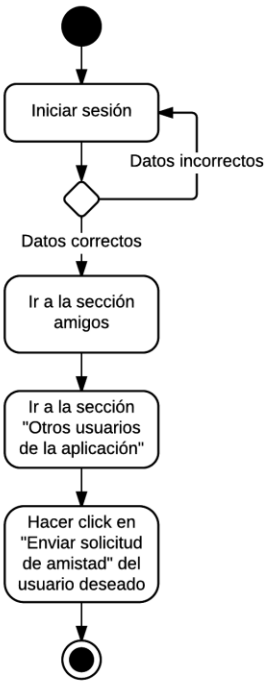
3.22. – Diagrama de actividad. Ver listas.

3.3.13. - Ver usuarios de la aplicación



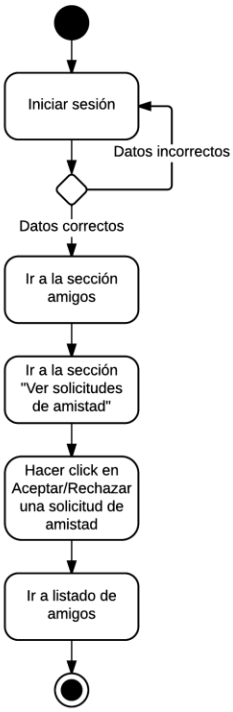
3.23. – Diagrama de actividad. Ver usuarios de la aplicación.

3.3.14. - Añadir amigos



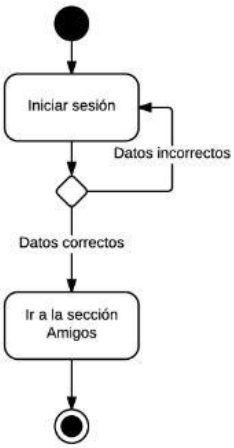
3.24. – Diagrama de actividad. Añadir amigos.

3.3.15. - Aceptar/Rechazar petición de amistad



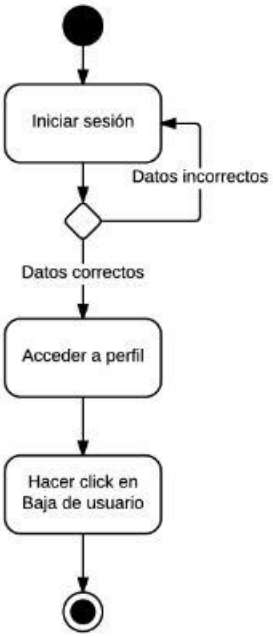
3.25. – Diagrama de actividad. Aceptar/Rechazar peticiones de amistad.

3.3.16. - Ver listado de amigos



3.26. – Diagrama de actividad. Ver listado de amigos.

3.3.17. - Baja de usuario



3.27. – Diagrama de actividad. Darse de baja en el sistema.

IV. - Diseño

1. - Introducción

Uno de los patrones de diseño más utilizados y conocidos en el mundo del desarrollo es el patrón MVC (Modelo-Vista-Controlador). Los principios de este patrón de arquitectura son la reutilización de código y la separación de conceptos.

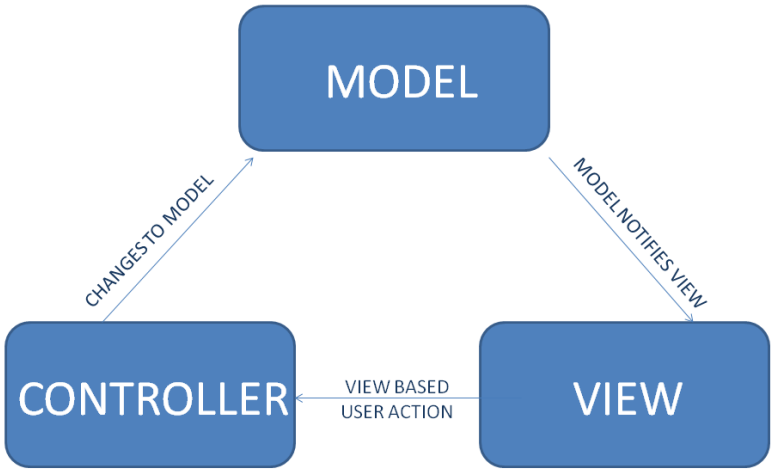
2. - Patrón de diseño MVC

El patrón de diseño MVC separa la estructura de una aplicación en tres tipos de elementos o capas: el modelo, las vistas y los controladores.

- Modelo: es la representación de los datos en el sistema.

- Controlador: se encarga de responder a eventos y realizar peticiones al modelo. Hace de enlace entre la capa de datos y la interfaz de usuario.

- Vista: la interfaz de usuario.



4.1. – Esquema Modelo-Vista-Controlador.

Este patrón fue en un principio desarrollado por los creadores de Smalltalk para aplicaciones de escritorio, pero poco a poco ha sido adoptado como arquitectura para el diseño y la implementación de aplicaciones web.

Para comprender bien el funcionamiento de una arquitectura MVC se debe entender esta división por capas y la comunicación entre éstas.

2.1. - Modelo

El modelo es la información y las reglas que se aplican a dicha información, los cuales representan los conceptos que la aplicación maneja. El modelo le proporciona al controlador una representación de los datos que el usuario solicita. Contiene la parte más importante de la lógica de una aplicación, la lógica que se aplica al problema que estamos intentando resolver.

Los distintos modelos que forman la aplicación se van distribuyendo según la necesidad, esto quiere decir que no todos los modelos deben estar inyectados en todas las partes de la aplicación, lo cual fomenta un mejor rendimiento de ésta.

2.2. - Vista

La vista proporciona formas distintas de presentar la información recibida desde el modelo. Este patrón nos permite utilizar una misma vista con distintos controladores y viceversa.

En el caso de este proyecto, cada vista está asociada a un controlador. Éste se encarga de gestionar la información requerida por la vista. Para ello se sirve de los distintos modelos que se necesiten y a partir de ellos realiza las peticiones oportunas al back-end.

2.3. - Controlador

El controlador se encarga de manejar las peticiones del usuario. Su principal función es llamar y coordinar los recursos y objetos necesarios para llevar a cabo la acción del usuario. Normalmente, el controlador llama al modelo adecuado para realizar la tarea y entonces, selecciona la vista asociada.

Debido a que el back-end del proyecto presenta una arquitectura API RESTful, los controladores se encargan de:

- Recoger la información de los modelos.
- Presentar la información a las vistas.
- Interactuar mediante servicios con el back-end de la aplicación, o en caso necesario, con aplicaciones externas.
- Capturar eventos para que puedan ser gestionados dentro del flujo de la aplicación. Un ejemplo de ello serían los posibles errores que se puedan producir al intentar interactuar con el back-end.

2.3.1. - Recoger la información de los modelos

Debido a la característica de la bidireccionalidad (*Double binding*) de AngularJS, la información es recogida en los modelos en dos direcciones, cuando la vista actualiza la información o cuando el código es el encargado de realizar dicha acción. Esto permite que el modelo esté siempre actualizado.

2.3.2. - Presentar la información a las vistas

Igual que en el apartado anterior, la bidireccionalidad nos permite realizar esta acción sin esfuerzo alguno, no obstante en ocasiones debemos ayudarnos de algún evento para que dicha actualización se lleve a cabo.

2.3.3. - Interactuar mediante servicios con el back-end de la aplicación

Los controladores se apoyan en clases llamadas servicios, para este caso los servicios realizan peticiones http para obtener y enviar la información de los modelos al back-end. Además, se encargan de gestionar tanto las peticiones como las respuestas. Esto nos obliga a tener en cuenta tanto los casos de error como de éxito al realizar dichas llamadas.

2.3.4. - Capturar eventos para que puedan ser gestionados dentro del flujo de la aplicación

En ocasiones debemos redirigir al usuario a ciertos lugares de la aplicación debido a nuevos permisos. En este caso utilizamos una extensión de los controladores llamada interceptor, el cual nos permite redirigir el flujo de la aplicación a nuestro controlador.

3. - Beneficios e inconvenientes

3.1. - Beneficios

Los principales beneficios del patrón MVC son los siguientes:

- Desacoplamiento total entre capas.
- Reutilización de código.
- Mejor escalabilidad.
- La información mostrada en las vistas está actualizada siempre.

3.2. - Inconvenientes

Los principales inconvenientes del patrón MVC son los siguientes:

- Mayor dedicación en el proceso de planificación inicial, debido a que se debe dejar clara desde un principio la distribución y contenido de sus componentes.
- Cuando se pretende dar soporte a distintos tipos de bases de datos, se deben realizar las implementaciones oportunas para que ambos puedan ser soportados.
- MVC es un patrón de diseño orientado a objetos, por lo que su implementación es sumamente costosa y difícil en lenguajes que no siguen este paradigma.

V. - Implementación

1. - Introducción

A continuación se describen las tecnologías, frameworks, herramientas y entornos de trabajo que se han utilizado para el desarrollo de la aplicación, así como una descripción de los temas abordados que se consideran de mayor importancia o dificultad.

2. - MEAN Stack

2.1. - ¿Qué es MEAN?

El stack MEAN [3] es una congregación de cuatro tecnologías de programación, en lenguaje JavaScript, las cuales son:

- MongoDB: gestor de bases de datos no relacionales.
- ExpressJS: proporciona la funcionalidad necesaria para atender y recibir peticiones en un servidor.
- AngularJS: framework para el desarrollo front-end que nos permite gestionar las vistas de la aplicación, así como interactuar con el servidor.
- NodeJS: aporta la funcionalidad necesaria para poder ejecutar código JavaScript en el servidor.

Todas ellas combinadas, permiten la creación de aplicaciones web, tanto del lado del servidor como del cliente, interactivas, eficientes y bien organizadas. Más adelante ampliaremos la información de las cuatro tecnologías.

5.1. – Logo MEAN Stack.

Entre las características más destacadas de esta asociación encontramos:

- Usar JavaScript del lado del servidor (NodeJS y ExpressJS)
- Usar JavaScript en el lado del cliente (AngularJS)
- Almacenar objetos JSON en MongoDB

- Usar los objetos JSON para transferir datos fácilmente de la base de datos al servidor y del servidor al cliente

Debido al uso de JavaScript en ambas partes de la aplicación, los tiempos de desarrollo de la aplicación mejoran notablemente, ya que los desarrolladores pueden trabajar tanto en el back-end como en el front-end sin necesidad de tener que especializarse en una única parte.

Otro detalle importante, es el almacenamiento de la información en formato JSON. Esto nos permite enviar datos en las peticiones con la estructura que se va a persistir en la base de datos, lo cual nos ahorra tareas de parseo para tener que adaptar los datos al formato exigido por la capa de persistencia.

2.2. - ¿Cuándo utilizar MEAN?

La columna vertebral de MEAN es NodeJS, gracias a su robustez a los métodos que expone su API. Podemos usarlo para transferir datos para aplicaciones como chats, actualización de estados, o cualquier otra situación que requiera mostrar datos rápidamente en tiempo real:

- Actualización de estados en tiempo real por el usuario (como Twitter)
- Canales RSS
- Tienda online
- Aplicaciones para votaciones

2.3.- ¿Quién trabaja con MEAN?

A continuación se presenta una lista de empresas que ya han integrado NodeJS en varias de sus operaciones:

5.2. – Logo Walmart.

Walmart: Walmart empezó usando Node.js en 2012 para dar a los usuarios de móvil una moderna experiencia front-end. Haciendo uso de la plataforma JavaScript, fueron capaces de integrar de forma rápida y sencilla sus APIs existentes con su aplicación NodeJS.

5.3. – Logo Yahoo!

Yahoo!: Yahoo empezó experimentando con NodeJS en 2010. Al principio lo usaban para pequeñas tareas como subir archivos, y ahora usan NodeJS para controlar casi 2 millones de peticiones por minuto. Han notado un incremento en la velocidad y una simplificación en el proceso de desarrollo.

5.4. – Logo LinkedIn.

LinkedIn: LinkedIn empezó a desarrollar el lado del servidor de su app móvil totalmente con NodeJS. Antes usaban Ruby, pero desde que cambiaron han visto un enorme incremento en rendimiento, en un rango de 2 a 10 veces más rápido.

5.5. – Logo PayPal.

Paypal: Paypal recientemente se ha subido a bordo de NodeJS y empieza a migrar su código Java a NodeJS. Empezaron experimentando solo con la página de Account Overview, pero una vez vieron que la velocidad se incrementó en un 35% y el coste de desarrollo se redujo a la mitad, empezaron a pasar todo el sitio a NodeJS.

2.4. - Los pilares de MEAN Stack

5.6. – Logo MongoDB.

Mongo es un sistema de base de datos no relacional (NoSQL) de código abierto que guarda los datos en documentos tipo JSON (JavaScript Object Notation) pero en forma binaria (BSON) para hacer la integración de una manera más rápida. Se pueden ejecutar operaciones en JavaScript en su consola en lugar de consultas SQL. Debido a su flexibilidad, es muy escalable y ayuda al desarrollo ágil de proyectos web.

5.7. – Logo ExpressJS.

Express es un framework por encima de Node.js que permite crear servidores web y recibir peticiones HTTP de una manera sencilla, lo que permite también crear APIs REST de forma rápida.

5.8. – Logo AngularJS.

Angular es un framework basado en JavaScript para la parte cliente o front-end de una aplicación web, que respeta el paradigma MVC y permite crear Single-Page Applications (Aplicaciones web que no necesitan recargar la página), de manera más o menos sencilla. Es un proyecto mantenido por Google y que actualmente está muy en auge.

5.9. – Logo NodeJS.

NodeJS es un entorno de programación en JavaScript para el back-end basado en el motor V8 de JavaScript del navegador Google Chrome y orientado a eventos, no bloqueante, lo que lo hace muy rápido a la hora de crear servidores web y emplear tiempo real. Fue creado en 2009 y aunque aún es joven, las últimas versiones lo hacen más robusto además de la gran comunidad de desarrolladores que posee. No sólo se utiliza del lado del servidor, se ha extendido tanto que se emplea en Stylus, un preprocesador CSS, en Grunt un gestor de tareas basado en JavaScript, etc.

3. - Requisitos y herramientas. Instalación.

3.1. - Introducción

En esta sección describiremos brevemente las herramientas que necesitaremos a la hora de desarrollar nuestra aplicación.

3.2. - Paquete MEAN

Para poder desarrollar una aplicación con MEAN, necesitaremos una serie de frameworks y librerías, a los que se les denomina comunmente paquetes o dependencias.

Podemos instalar todas las dependencias que necesitamos para ejecutar una aplicación con MEAN Stack de varias formas. A continuación, explicaremos cómo instalar el paquete MEAN entero.

Lo primero que haremos será instalar NodeJs y su gestor de paquetes npm. Esta instalación es muy sencilla, aún así, hablaremos sobre ella un poco más adelante, ya que hay un par de aspectos muy importantes a tener en cuenta a la hora de configurar Node.

Cuando hayamos terminado de instalar Node, pasaremos a instalar MongoDB. La instalación de este gestor de bases de datos no relacionales también es muy sencilla.

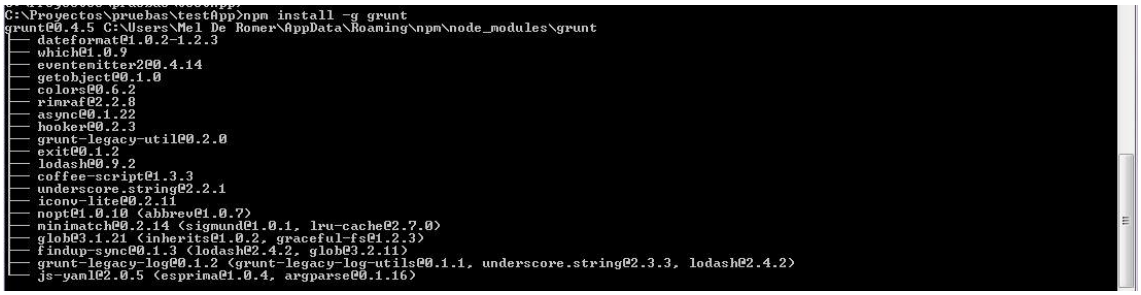
Para instalar MongoDb, iremos a su página web oficial https://www.mongodb.org y haremos click en Descargar MongoDb. Antes de que empiece la descarga, deberemos seleccionar nuestro sistema operativo, y su versión para descargar el instalador correcto, y seguidamente, haremos click en Descargar(MSI). Con esto descargaremos un archivo de tipo .msi, y lo instalaremos siguiendo las instrucciones. Para más detalle de cómo instalar MongoDb, podemos consultar la documentación oficial, para Windows en nuestro caso, en http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/.

Una vez instalado MongoDb, procederemos a instalar Git, que es la herramienta de control de versiones que utilizaremos en este proyecto. Para descargar el instalador de Git, iremos a su web oficial(https://git-scm.com/downloads), y descargaremos el paquete asociado a nuestro sistema operativo. La instalación de Git es igual de simple que las anteriores y no haremos hincapié en ella.

Una vez instalados Node, MongoDb y Git, instalaremos Grunt en nuestro equipo.

Grunt es un sistema que permite automatizar tareas comunes de desarrollo.

Ahora que tenemos Node y su gestor de paquetes npm instalados, podemos ejecutar la siguiente instrucción que instalará Grunt de forma global en nuestro equipo.



5.10 .- Comando para instalar Grunt de manera global.

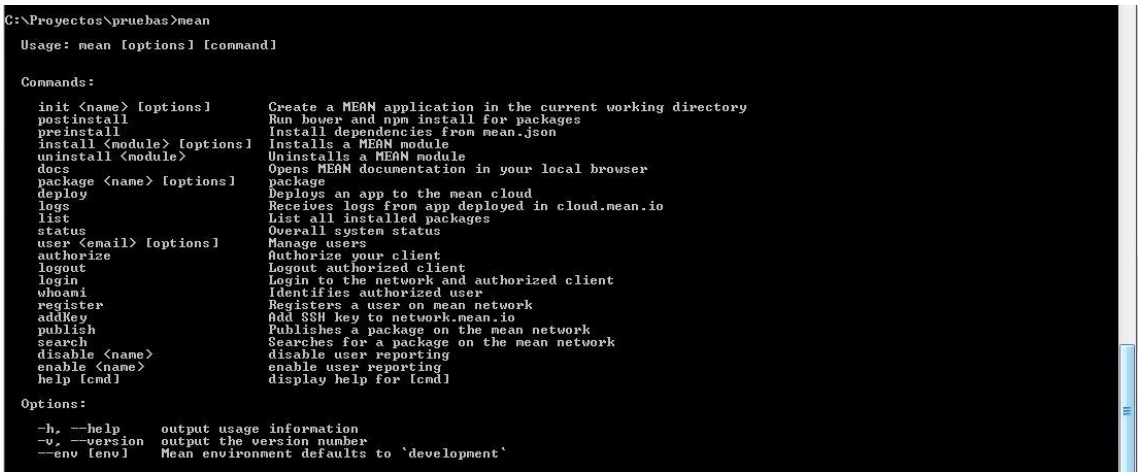
Una vez haya finalizado la instalación de Grunt, procederemos a instalar Bower, que es un gestor de paquetes principalmente enfocado al front-end, y del cual hablaremos más adelante.

5.11 .- Comando para instalar Bower de manera global.

Por último, deberemos instalar el paquete mean-cli con npm.

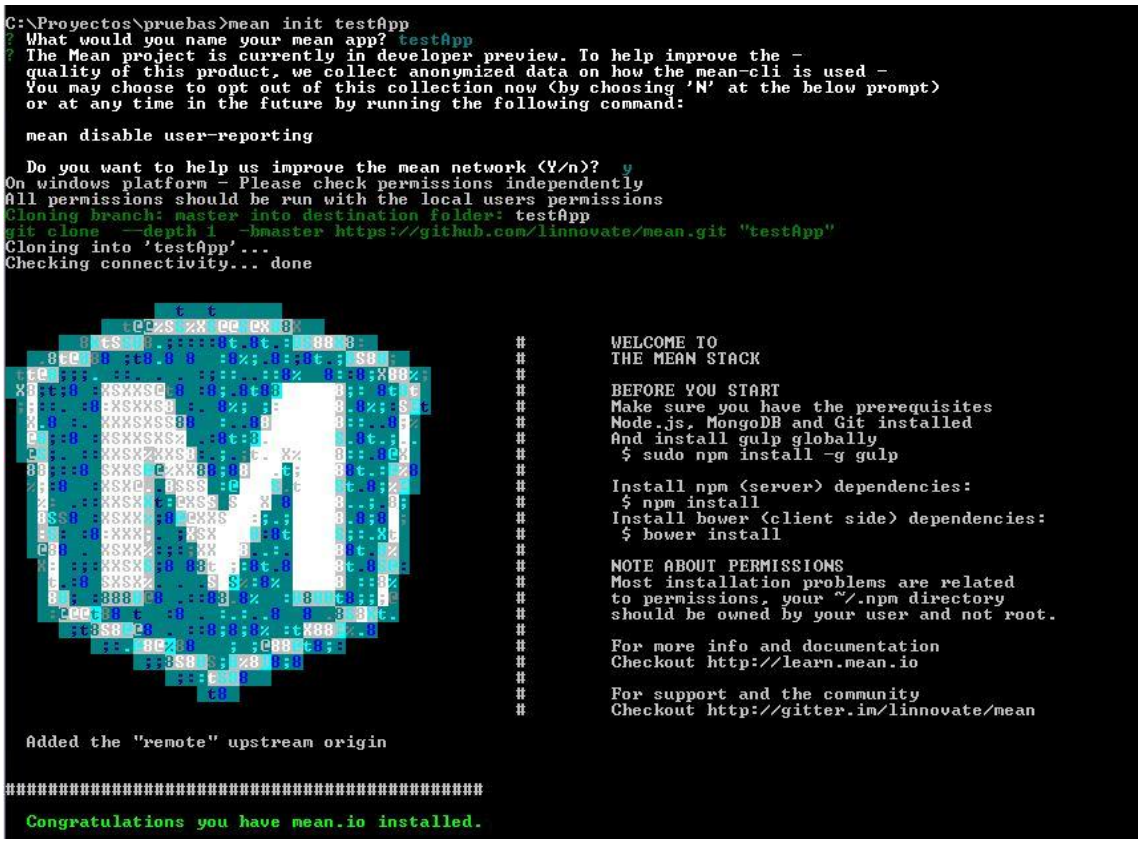
5.12 .- Comando para instalar mean-cli de manera global.

Esta instrucción hará que podamos ejecutar esta serie de comandos desde nuestra consola:



5.13 .- Comandos que ofrece mean-cli

Por último, para crear un nuevo proyecto con MEAN, lanzaremos el comando `mean init <nombreDeLaAplicacion>`, lo cual nos creará un proyecto vacío para empezar a desarrollar nuestra aplicación.



5.14 .- Resultado en consola tras crear una nueva aplicacion MEAN

La ventaja de instalar MEAN de esta forma, es que nos crea ya un proyecto con sus respectivos package.json y bower.json con todas las dependencias básicas, como pueden ser ExpressJS, Mongoose, AngularJs, Bootstrap, etc.

El principal inconveniente es que tanto bower.json como package.json, contienen dependencias que probablemente no vayamos a utilizar, y si no las eliminamos manualmente, tendremos un montón de archivos que no vamos a necesitar.

Más adelante, hablaremos de estos dos ficheros, package.json y bower.json, sobre qué son y para qué los vamos a necesitar.

Para el desarrollo de este proyecto, se ha optado por la instalación componente a componente de MEAN.

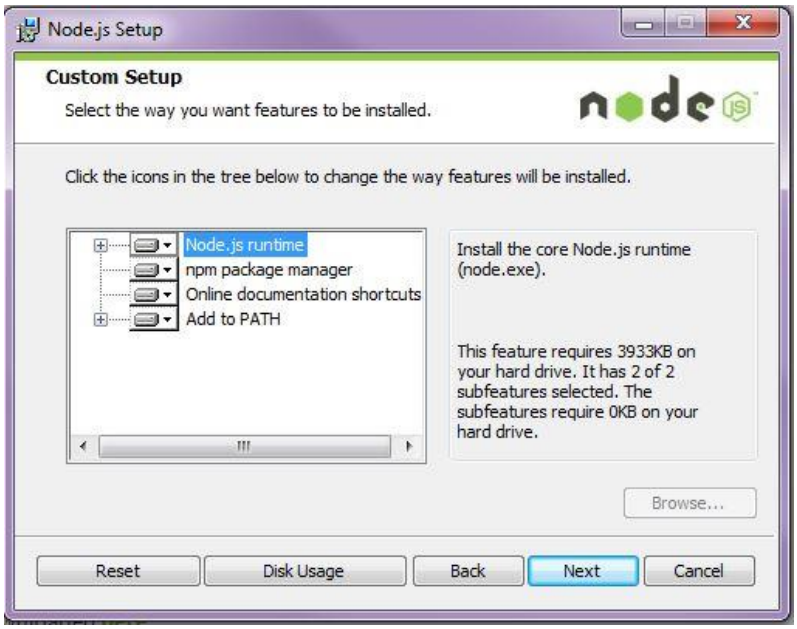
3.3. - Node

Node [6] es un entorno de programación en JavaScript para el back-end.

Para instalar NodeJS, visitaremos su página web http://nodejs.org y descargaremos el instalador para nuestro sistema operativo. En este caso descargaremos el archivo .msi para Windows (64-bit).

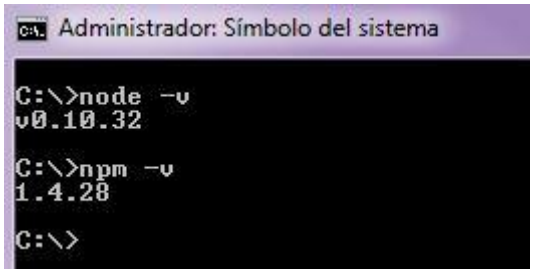
Para instalarlo, simplemente haremos doble clic sobre el archivo descargado y seguiremos las instrucciones.

Debemos asegurarnos que en el penúltimo paso seleccionamos los componentes que vamos a necesitar, que son Node.js runtime y npm package manager. Además, aquí debemos asegurarnos que tenemos marcada la opción de añadir al *PATH,* sino deberemos hacerlo manualmente más tarde para poder ejecutar los comandos de node en la consola.



5.15. – Instalación Node.

Cuando haya terminado de instalarse NodeJS en nuestro equipo comprobaremos que todo ha ido correctamente ejecutando los siguientes comandos en la consola:



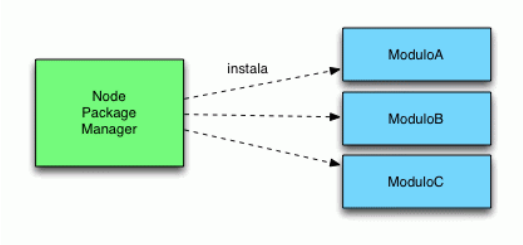
5.15. – Captura consola de verificación de la instalación de Node.

Si todo ha ido bien, veremos el número de la versión de node y del gestor de paquetes de node (npm).

3.4. - npm

A lo largo del desarrollo de una aplicación con Node, nos veremos obligados a utilizar módulos de terceros debido a que el propósito principal de éste es utilizar JavaScript del lado del servidor y delegar otras funcionalidades en su principio de modularidad.

La instalación de nuevos módulos con Node se realiza de manera muy sencilla con la herramienta npm (Node Package Manager). Esta herramienta es la encargada de añadir los nuevos módulos que necesitemos.



5.17. – Esquema NPM.

Para añadir un nuevo módulo, simplemente escribimos el siguiente comando en la consola:

```
npm install <nombre_módulo>
```

Vamos a ver un ejemplo de cómo instalar un nuevo módulo. En este caso instalaremos el módulo *validator*. Este módulo sirve, entre otras cosas, para validar Strings.

```
C:\ejemplos_pfc>npm install validator
validator@3.41.2 node_modules\validator

C:\ejemplos_pfc>
```

5.18. – Instalación validator.

Una vez instalado, si abrimos el directorio elegido para la instalación, podremos ver que npm ha generado una carpeta llamada *node_modules*, donde se instalarán todos los módulos que añadamos de ahora en adelante a nuestro proyecto con npm.

A partir de ahora, para utilizar el módulo que acabamos de instalar, escribiremos la siguiente instrucción en nuestro código Node:

```
1  var validator = require('validator');
```

5.19. – Inclusión del módulo validator en Node.

Para ver más módulos disponibles podemos consultar la web oficial de npm: http://www.npmjs.com/

3.5. - Bower

5.20. –Logo Bower.

Bower [13] es un gestor de paquetes principalmente para front-end. Con Bower podemos gestionar cómodamente las dependencias del lado del cliente desde la consola. Por dependencia se entiende cualquier biblioteca, framework o conjunto de archivos susceptible de ser encapsulado como paquete.

3.5.1. - Instalación de Bower

Para poder instalar Bower, deberemos instalar Node.js y Git previamente.

La instalación de Bower se hace mediante el siguiente comando:

```
C:\>npm install -g bower
```

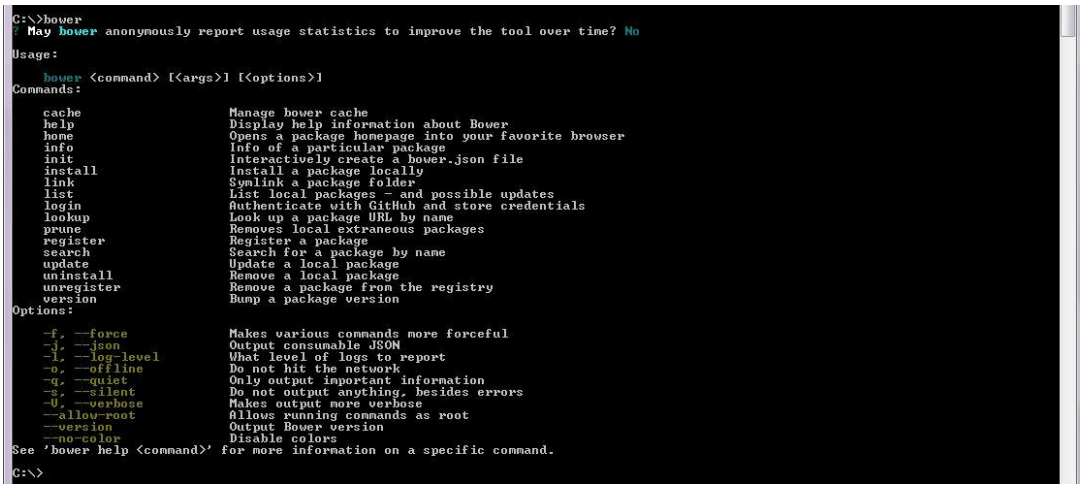
5.21. –Instalación Bower.

Cuando finalice el proceso de instalación, podemos comprobar que bower se ha instalado correctamente lanzando el siguiente comando:

```
C:\>bower --v
1.4.1
```

5.22. –Comprobación instalación de Bower.

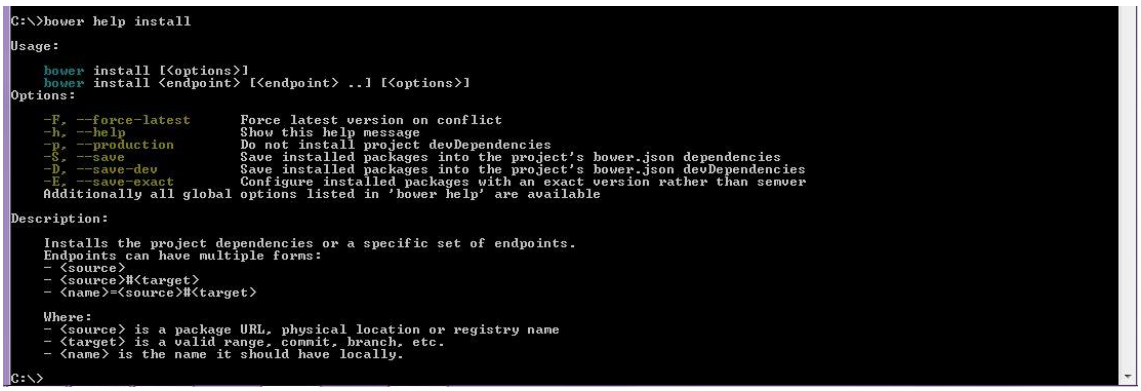
Bower dispone de una serie de comandos que podemos ver tecleando la palabra bower en la consola.



5.23. –Comandos disponibles de Bower.

Para ver más detalles sobre la utilización de un determinado comando, podemos escribir

`bower help <comando>`



5.24. –Ayuda sobre comandos en Bower.

3.5.2. - Instalación de un paquete con Bower

Hay dos formas de instalar un paquete con Bower. Una opción es indicando solamente el nombre del paquete que queremos instalar, y otra, indicando, además del nombre, la versión específica del paquete.

Si no indicamos el número de versión, el comando `bower install` instala automáticamente la última versión estable del paquete que hayamos seleccionado. Por ejemplo, si instalamos Bootstrap:

5.25. –Instalación de Bootstrap con Bower.

Como vemos, Bower ha creado un nuevo directorio llamado *bower_components*, donde se descargan todas las dependencias. Además de la última versión de Bootstrap, Bower ha instalado también jQuery, ya que es una dependencia directa de Bootstrap. Esta función de Bower es realmente útil, ya que nos evita tener que ir buscando dependencias "a mano".

Una de las ventajas de Bower es que permite especificar todas las dependencias front-end de un proyecto en un archivo especial llamado *bower.json*.

De esta forma, cada vez que despleguemos nuestro proyecto, sólo tendremos que lanzar el comando `bower install` para que se instalen automáticamente todas las dependencias indicadas en nuestro *bower.json*.

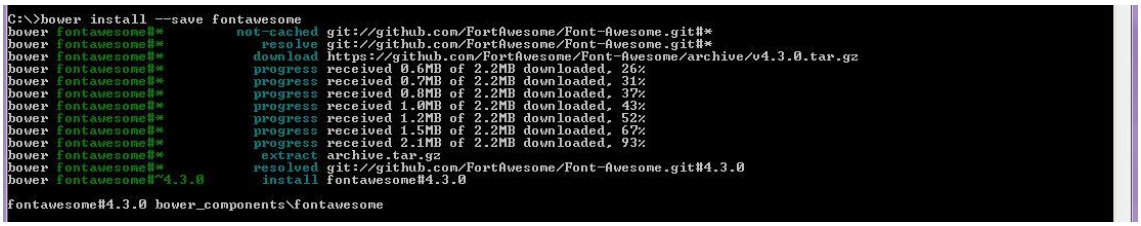
3.5.3. - Creación del fichero bower.json

La forma más sencilla de generar el archivo *bower.json* es con el comando `bower init`

Este comando lanza una utilidad que, a través de una serie de preguntas, genera automáticamente el archivo.

5.26. –Inicialización del fichero bower.json.

Para instalar paquetes, y que además se añadan como dependencia en *bower.json*, podemos utilizar la opción `--save` del comando install de Bower.



5.27. –Instalación de paquete fontawesome con Bower.

Si abrimos ahora el archivo *bower.json*, veremos que automáticamente se ha creado la dependencia de *fontawesome*.

5.28. – Captura de bower.json.

3.6. - Yeoman

5.29. –Logo Yeoman.

Tal y como explican en su web oficial, Yeoman es un "*robusto conjunto para el lado del cliente compuesto por herramientas y frameworks que pueden ayudar a los desarrolladores a crear rápidamente aplicaciones web*".

Entre otras cosas, Yeoman permite la creación de esqueletos para aplicaciones proporcionándote una serie de plantillas predefinidas como HTML5, Twitter Bootstrap o creando las propias.

Para poder utilizar Yeoman deberemos tener instalado Node.js y npm.

Para instalar Yeoman, abrimos la consola y escribimos el siguiente comando:

```
C:\>npm install -g yo bower grunt-cli

> spawn-sync@1.0.11 postinstall C:\Users\elena\AppData\Roaming\npm\node_modules\yo\node_modules\cross-spawn\node_modules\spawn-sync
> node postinstall

C:\Users\elena\AppData\Roaming\npm\grunt -> C:\Users\elena\AppData\Roaming\npm\node_modules\grunt-cli\bin\grunt
C:\Users\elena\AppData\Roaming\npm\yo -> C:\Users\elena\AppData\Roaming\npm\node_modules\yo\lib\cli.js

> yo@1.4.7 postinstall C:\Users\elena\AppData\Roaming\npm\node_modules\yo
> yodoctor

C:\Users\elena\AppData\Roaming\npm\bower -> C:\Users\elena\AppData\Roaming\npm\node_modules\bower\bin\bower

Yeoman Doctor
Running sanity checks on your system

√ Global configuration file is valid
√ NODE_PATH matches the npm root
√ Node.js version
√ No .bowerrc file in home directory
√ No .yo-rc.json file in home directory
√ npm version

Everything looks all right!
grunt-cli@0.1.13 C:\Users\elena\AppData\Roaming\npm\node_modules\grunt-cli
├── resolve@0.3.1
├── nopt@1.0.10 (abbrev@1.0.7)
└── findup-sync@0.1.3 (glob@3.2.11, lodash@2.4.2)

bower@1.4.1 C:\Users\elena\AppData\Roaming\npm\node_modules\bower
├── is-root@1.0.0
```

5.30. – Instalación de Yeoman.

Este comando, además de instalar Yeoman, instalará Bower y Grunt en caso de que no los tuviéramos instalados antes.

Esta utilidad se basa en el uso de generadores, los cuales permiten crear arquitecturas predefinidas además de tipos de archivos. Por ejemplo, si estamos trabajando en un proyecto con AngularJS, Yeoman nos permitirá añadir el esqueleto de archivos como directivas o controladores. Yeoman dispone de muchos generadores y gracias a que es una herramienta open source, éstos aumentan cada día. Además, es totalmente

extensible ya que podemos crear nuestros propios generadores. Para consultar sus posibilidades podemos consultar la página web del proyecto o su repositorio en GitHub.

4. - Creación del proyecto Node

La configuración de las aplicaciones de Node se especifica en un archivo llamado *package.json*.

Este archivo se utiliza para dar la información necesaria a npm y permite identificar el proyecto, su autor/es y sus dependencias, entre otras cosas.

También contiene otra información sobre el proyecto como su descripción, la versión del proyecto en una distribución particular, información de licencia, etc.

Normalmente, este archivo está ubicado en el directorio raíz del proyecto y se genera automáticamente tras ejecutar el comando `npm init` en la consola.

Para ello abriremos la consola y crearemos un directorio para nuestro proyecto.

Sobre el nuevo directorio, ejecutaremos el comando `npm init`. Con este comando ejecutará un generador que formulará una serie de preguntas para ayudar a crear nuestro *package.json*.

Muchos de los campos los podremos dejar en blanco, y en caso de que sean obligatorios, la herramienta nos proporcionará una opción por defecto.

En primer lugar nos pregunta el nombre del proyecto *name.* Este campo será el nombre del proyecto. Por defecto, npm le pone el nombre del directorio del proyecto.

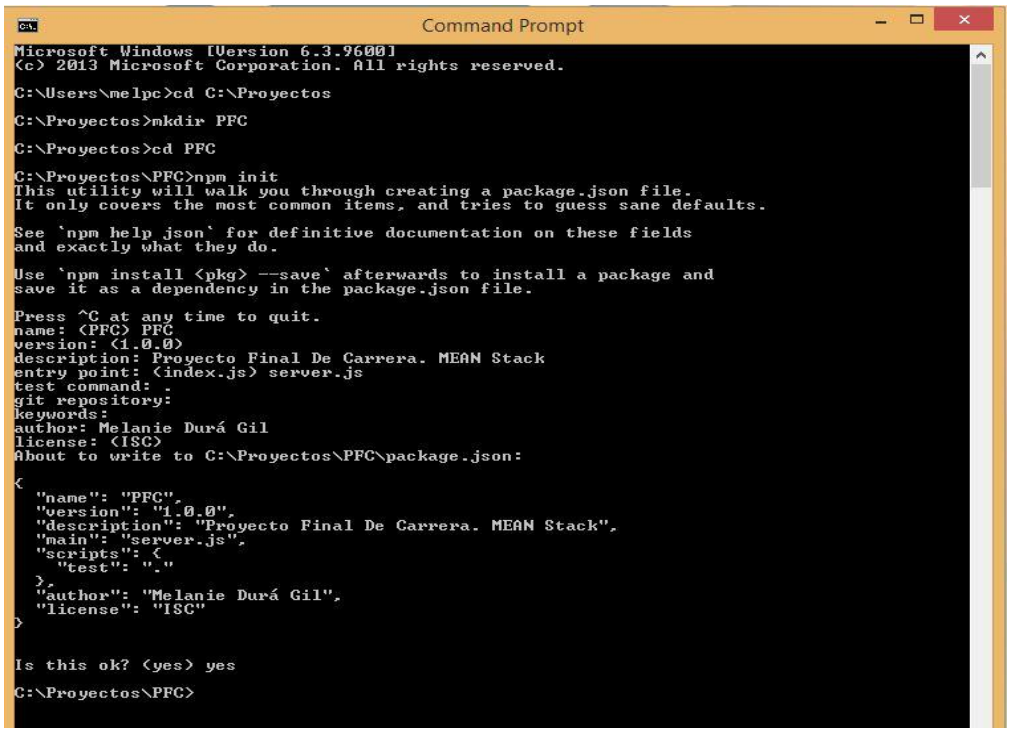
La siguiente pregunta es la versión del código *version.* Como estamos inicializando el proyecto, dejaremos el valor por defecto que es 1.0.0. Este valor irá incrementando conforme vayamos haciendo mejoras e implementando nuevas funcionalidades en nuestro código.

En el campo *description* introduciremos una pequeña descripción del proyecto.

El siguiente campo es *main o entry point.* Éste será el punto de entrada a nuestra aplicación.

De momento, los campos *test command, git repository* y *keywords* los dejaremos en blanco.

Cuando hayamos terminado de rellenar los campos, el generador nos mostrará el *package.json* resultante y nos preguntará si todo es correcto. Si contestamos que sí, se generará el archivo en el directorio raíz del proyecto.



5.31. – Creación de package.json.

5. - Creación del servidor

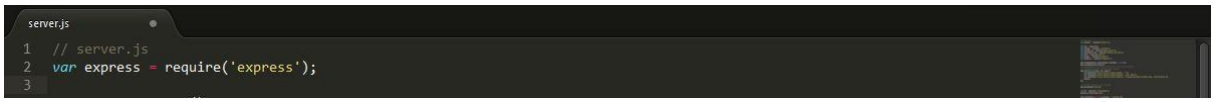
En esta sección veremos cómo crear un servidor que atienda peticiones REST. Para ello nos serviremos de NodeJS combinado con ExpressJS [9].

ExpressJS es un micro framework que funciona mediante Node.js y que nos permite declarar y gestionar peticiones a nuestro back-end desde nuestra aplicación single page.

Para instalar ExpressJS de manera global, utilizaremos el siguiente comando:

```
npm install -g express
```

Para poder utilizar ExpressJS en nuestro proyecto, deberemos incluirlo en nuestro archivo "*server.js*" mediante la siguiente instrucción:



5.32. – Inclusión de ExpressJS en el proyecto.

En primer lugar indicaremos a nuestra aplicación los parámetros de configuración para gestionar estas peticiones.

5.33. – Configuraciones de las peticiones al servidor.

En la primera línea indicamos que el formato en el que vamos a parsear la respuesta será JSON. Por su parte las siguientes líneas indican que cabeceras contendrán las respuestas de la aplicación.

5.34. – Conexión con la base de datos.

Por otro lado indicaremos a nuestro server la conexión que debe establecer con la capa de base de datos. Para ello utilizamos Mongoose, que es una librería para gestionar los accesos a la base de datos, entre otras cosas.

A pesar de que el back-end de nuestra aplicación se trata de una API REST, debemos indicar de alguna forma un endpoint al que apuntar las respuestas a las peticiones. Gracias a AngularJS este endpoint será un solo fichero, ya que nuestro front-end es

single page. Para ello, tal y como vemos en la imagen, nuestro server enviará el contenido de las respuestas a "*index.html*", y AngularJS será el encargado de seleccionar y renderizar la vista que solicita el usuario.

```
33
34  app.get('*', function(req, res) {
35      res.sendfile(path.join(__dirname + '/public/views/index.html'));
36  });
37
```

5.35. – Vista inicial de la aplicación.

En la siguiente imagen podemos apreciar la inicialización de nuestro servidor (línea 40), así como la publicación de éste (línea 42) para que pueda ser accesible.

```
39
40  app.listen(config.port);
41
42  exports = module.exports = app;
```

5.36. – Inicialización del servidor.

6. - Herramientas de control de versiones y creación del repositorio de código con Git

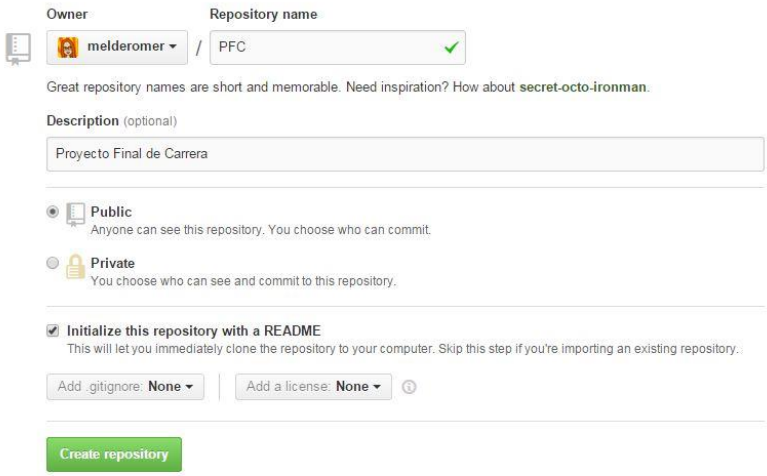
Un sistema de control de versiones es una herramienta que registra todos los cambios hechos en un proyecto, guardando así versiones del producto en todas sus fases del desarrollo. Es especialmente útil cuando hay más de un desarrollador trabajando en el proyecto.

Para este proyecto, hemos hecho uso de Git [31], que es un software de control de versiones muy potente diseñado por Linus Torvalds.

Aunque aquí sólo hagamos uso de él para controlar nuestro código fuente, Git puede registrar los cambios realizados sobre un archivo o conjunto de archivos de casi cualquier tipo. Git es open source y su utilización es bastante sencilla.

A continuación vamos a ver cómo generar un repositorio, y cómo subir archivos a él. Para el alojamiento de nuestro código hemos utilizado GitHub [32].

Una vez registrados en GitHub, para crear un nuevo repositorio, haremos clic en el botón de “New repository” y rellenaremos el siguiente formulario:



5.37. – Formulario para la creación de un repositorio en GitHub.

Una vez creado el repositorio en GitHub, abriremos la consola y en el directorio raíz de nuestro proyecto ejecutaremos el comando `git init`

```
C:\>cd Proyectos

C:\Proyectos>cd PFC

C:\Proyectos\PFC>git init
Initialized empty Git repository in C:/Proyectos/PFC/.git/

C:\Proyectos\PFC>
```

5.38. – Inicialización de un repositorio.

Ahora deberemos indicar a qué repositorio de GitHub hace referencia nuestro directorio, de manera que ejecutaremos el siguiente comando:

```
git remote add PFC https://github.com/melderomer/PFC.git
```

Con esto, ya podremos subir nuestros archivos al repositorio. Primero deberíamos comprobar el estado del repositorio con `git status`

```
C:\Proyectos\PFC>git status
# On branch master
# Changes not staged for commit:
#   (use "git add/rm <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   app/routes.js
#       modified:   public/js/app.js
#       modified:   public/js/controllers/UserBookInfoCtrl.js
#       deleted:    public/js/services/NerdService.js
#       modified:   public/views/index.html
#       deleted:    public/views/nerd.html
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       .gitignore
#       public/js/controllers/MainCtrl.js
#       public/js/services/BookService.js
#       public/views/book.html
no changes added to commit (use "git add" and/or "git commit -a")
```

5.39. – Ejecución del comando git status.

Como bien indica git en la consola, antes de poder subir los archivos al repositorio hay que ejecutar los siguientes comandos:

```
git add .
```

```
git commit –m “Mensaje descriptivo de los cambios realizados”
```

Una vez hemos lanzado los comandos anteriores, ya podemos subir los cambios al repositorio con el comando `git push`. Por seguridad, Git nos solicitará el nombre de usuario y contraseña.

```
C:\Proyectos\PFC>git commit -m "First commit"
[master 4363146] First commit
 4 files changed, 47 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 public/js/controllers/MainCtrl.js
 create mode 100644 public/js/services/BookService.js
 create mode 100644 public/views/book.html

C:\Proyectos\PFC>git push
Username for 'https://github.com': melderomer
Password for 'https://melderomer@github.com':
Counting objects: 17, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (10/10), done.
Writing objects: 100% (11/11), 1.48 KiB | 0 bytes/s, done.
Total 11 (delta 1), reused 0 (delta 0)
To https://github.com/melderomer/PFC.git
   455342c..4363146  master -> master
```

5.40. – Ejecución de los comandos git commit y git push.

Para más información de cómo utilizar Git, además de su web oficial, podemos consultar esta sencilla guía: http://rogerdudler.github.io/git-guide/index.es.html.

7. - Enrutamiento

Para el manejo de rutas en nuestra aplicación utilizaremos Express Router.

Express Router es una instancia aislada de un interceptor que se encarga de manejar las rutas de la aplicación. Podemos pensar en ello como una mini aplicación que sólo es capaz de ejecutar dicho interceptor y sus funciones de enrutamiento asociadas.

Este objeto router viene por defecto en ExpressJS.

Hay muchas formas distintas de utilizar la función router. En nuestra aplicación, primero crearemos una instancia de router.

```
10
11  var apiRouter = express.Router();
12
```

5.41. – Creación de una instancia de Express Router.

Después aplicaremos las rutas a dicha instancia:

```
90  apiRouter.route('/users')
91      .post(function(req, res){
92
93          var user = new User();
94
95          user.username = req.body.username;
96          user.name = req.body.name;
97          user.surname = req.body.surname;
98          user.password = req.body.password;
99          user.email = req.body.email;
100         user.avatar = req.body.avatar;
101         user.registerDate = new Date();
102
103         user.save(function (err) {
104             if (err){
105                 if(err.code == 11000)
106                     return res.json({ success: false, message: 'Este usuario ya existe' });
107                 else
108                     return res.send(err);
109             }
110             res.json({ message: 'Usuario creado!' });
111         });
112
113     })
114
115     .get(function(req, res){
116
117         User.find(function (err, users) {
118             if (err) res.send(err);
119
120             res.json(users);
121         });
122     });
123
```

5.42. – Rutas para POST y GET de usuarios.

Y por último añadiremos estas rutas a nuestra aplicación de la siguiente manera:

```
require('./app/routes')(app);
```

5.43. – Instrucción para importar las rutas de ficheros externos a la aplicación.

8. - Construcción de la base de datos

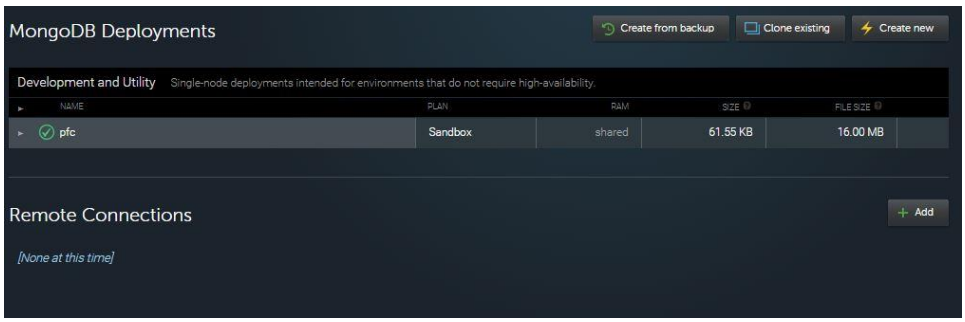
Para la gestión de la base de datos, vamos a utilizar una herramienta visual llamada MongoLab [37]. Esto es debido a que de esta forma podremos simular un entorno real.

8.1. - Creación de la base de datos

MongoDB es un sistema de base de datos NoSQL orientado a documentos, desarrollado bajo el concepto de código abierto. MongoDB guarda los datos en documentos tipo JSON (*JavaScript Object Notation*) pero en forma binaria (BSON) para hacer la integración de una manera más rápida.

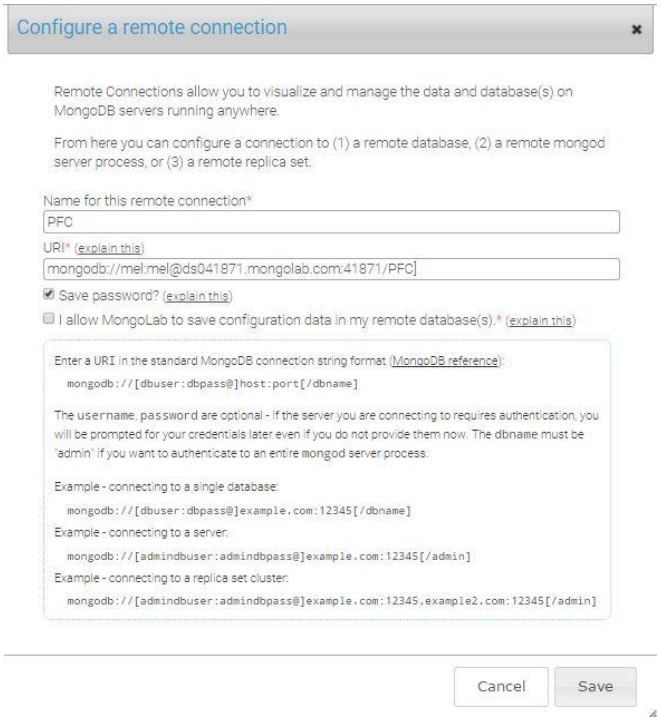
MongoLab nos da el servicio de base de datos como servicio SaaS (Software As A Service) De esta manera sólo tenemos que preocuparnos de dar de alta la base de datos y crear el usuario con el que nos vamos a conectar.

Primero habrá que darse de alta en MongoLab. Una vez hecho esto, para crear nuestra base de datos haremos clic en el botón “+ Add”.



5.44. – Interfaz gráfica de MongoLab.

Cuando hagamos clic en el botón, se nos mostrará un formulario para la creación de la base de datos con conexión remota:



5.45. – Configuración de una conexión remota a una base de datos en MongoLab.

Ahora en nuestro código tendremos que hacer referencia a la base de datos. Para esto, crearemos un fichero de configuración con la información de la conexión a la base de datos:

```
1  //config/db.js
2
3  module.exports = {
4      url : 'mongodb://mel:mel@ds041871.mongolab.com:41871/PFC'
5  }
```

5.46. – Fichero de configuración de la conexión a la base de datos de MongoLab.

Seguidamente, en nuestro server.js, importaremos el archivo que acabamos de crear:

```
12
13  var db = require('./config/db');
14
```

5.47. – Importación del fichero de configuración de la conexión a la base de datos de MongoLab.

8.2. - Creación de modelos

Para la creación de los modelos de nuestra aplicación, utilizaremos Mongoose.js [7].

Mongoose es un módulo que ejerce de puente entre nuestra base de datos MongoDB y Node.js, proporcionando la funcionalidad necesaria para llevar a cabo las interacciones con la capa de persistencia.

Para instalar Mongoose, ejecutamos el siguiente comando en la consola en el directorio raíz de nuestro proyecto:

```
npm install mongoose
```

Una vez finalizada la instalación, importamos el nuevo módulo que acabamos de instalar y conectamos con la base de datos a través del fichero de configuración de ésta.

```
14  var mongoose = require('mongoose');
15  var db = require('./config/db');
16  mongoose.connect(db.url);
```

5.48. – Importación Mongoose y conexión con la base de datos.

Ahora que ya podemos hacer uso de Mongoose, veremos cómo se ha generado el modelo lista.

```
1   var mongoose = require('mongoose');
2   var Book = require('./book');
3   var User = require('./user');
4
5   var Schema = mongoose.Schema;
6
7   var ListSchema = new Schema({
8       name: { type: String, required: true },
9       creationDate: { type: Date, required: true },
10      userId: { type: Schema.Types.ObjectId, ref: 'User' },
11      books: [{ type: Schema.Types.ObjectId, ref: 'Book' }]
12      });
13
14  module.exports = mongoose.model('List', ListSchema);
15
```

5.49. – Modelo Lista generado con Mongoose.

Lo que estamos haciendo aquí es crear un objeto de Mongoose de tipo Schema donde cada documento tiene los campos *name, creationDate, userId y books*, junto a sus respectivos tipos. En la última línea, exportamos el modelo para poder hacer uso de él fuera de este archivo.

9. - API REST. Implementación de un CRUD.

Una API REST es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP.

9.1. - Uso correcto de URIs

Las URL, Uniform Resource Locator, son un tipo de URI, Uniform Resource Identifier, que además de permitir identificar de forma única el recurso, nos permite localizarlo para poder acceder a él o compartir su ubicación.

La definición correcta de una URI sería:

```
{protocolo}://{dominio o hostname}[:puerto (opcional)]/{ruta del recurso}?
{consulta de filtrado}
```

5.50. – Estructura de una URI

9.2. - Acciones y métodos. CRUD.

En la filosofía REST, cada uno de los métodos del protocolo HTTP debe encargarse de realizar un tipo de acción:

- GET: su objetivo es obtener el recurso o recursos solicitados. Por lo general, estas llamadas no deben enviar ningún tipo de información en el cuerpo de la petición. Su estructura sería la siguiente: `/users` y con esta llamada, obtendríamos los usuarios de la aplicación.

```
apiRouter.route('/users')
    .get(function(req, res){

        User.find(function (err, users) {
            if (err) res.send(err);

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

5.51. – GET de usuarios.

Para obtener un usuario concreto de la aplicación utilizaríamos: `/users/:user_id`

Con esta llamada obtendríamos un solo usuario de la aplicación, siendo éste determinado por el valor que diéramos a userId. Por ejemplo, /users/38 nos proporcionaría la información del usuario con identificador 38.

- POST: su objetivo es guardar la información que enviamos en el cuerpo de una petición HTTP. En este caso, su estructura comparte la llamada a /users, pero al tratarse de una petición de tipo post, el back-end la gestionará con el método asociado a dicho tipo de petición.

```
apiRouter.route('/users')
    .post(function(req, res){

        var user = new User();

        user.username = req.body.username;
        user.name = req.body.name;
        user.surname = req.body.surname;
        user.password = req.body.password;
        user.email = req.body.email;
        user.avatar = req.body.avatar;
        user.registerDate = new Date();

        user.save(function (err) {
            if (err){
                if(err.code == 11000)
                    return res.json({ success: false, message: 'Este usuario ya existe' });
                else
                    return res.send(err);
            }
            res.json({ message: 'Usuario creado!' });
        });

    })
```

5.52. – POST de usuarios.

- PUT: en caso de que queramos actualizar un recurso de la aplicación, utilizaremos dicho método. Su estructura de URL sería:

  /users/:user_id

  A diferencia del GET, esta llamada acepta parámetros en el cuerpo de la petición.

- DELETE: este método se utiliza para eliminar un recurso de la aplicación.

  Su estructura de URL sería:

  /users/:user_id

```
apiRouter.route('/users/:user_id')
    .get(function (req, res){
        User.findById(req.params.user_id, function (err, user) {
            if (err) res.send(err);
            res.json(user);
        });
    })
    .put(function (req, res){
        User.findById(req.params.user_id, function (err, user){
            if (err) res.send(err);
            if (req.body.username) user.username = req.body.username;
            if (req.body.name) user.name = req.body.name;
            if (req.body.surname) user.surname = req.body.surname;
            if (req.body.password) user.password = req.body.password;
            if (req.body.email) user.email = req.body.email;
            if (req.body.avatar) user.avatar = req.body.avatar;

            user.save(function (err){
                if (err) res.send(err);

                res.json({ message: 'Usuario actualizado!'});
            });
        });
    })
    .delete( function (req, res){
        User.remove({
            _id: req.params.user_id
        }, function (err, user) {
            if (err) return res.send(err);
            res.json({message: 'Usuario eliminado'});
        });
    });
```

5.53. – GET de un usuario. PUT de un usuario. DELETE de un usuario.

10. - Gestionando vistas con AngularJS

10.1. - Creación de una vista

Como se ha comentado en capítulos anteriores, AngularJS [12] es un framework MVC front-end de JavaScript para el desarrollo de aplicaciones web que permite crear aplicaciones SPA (*Single-Page Applications*). Angular separa muy bien la responsabilidad de cada tecnología en su ámbito: CSS, HTML, JavaScript.

Las ventajas en la utilización de AngularJS para el desarrollo de aplicaciones web son muchas. Nombraremos las más importantes:

- **Reusabilidad:** Permite crear componentes (directivas) reutilizables.
- **Testeo:** Al tener componentes aislados, podemos testear su comportamiento de manera independiente.
- **Inyección de dependencias** a la hora de definir cualquier componente, ya sea un Controller, un Service, etc. debemos indicar de qué otros componentes depende y AngularJS se encargará de proporcionárselos a través de la función constructora.

En este apartado, vamos a explicar cómo se ha desarrollado el login de la aplicación. En un principio necesitaremos:

- Crear la ruta de login
- Crear la vista de login
- Crear la función doLogin() en el controlador asociado

Para crear la ruta de login, iremos a nuestro app.js donde indicamos la ruta en el navegador y el controlador asociado (líneas 8 a 11).

```
1  angular.module('miBiblioteca', ['ngRoute', 'ngResource', 'ngCookies', 'ngMaterial','infinite-scroll'])
2      .config([ '$routeProvider', '$locationProvider', function ($routeProvider, $locationProvider) {
3          $routeProvider
4              .when('/', {
5                  templateUrl: 'views/home.html',
6                  controller: 'MainController'
7              })
8              .when('/login', {
9                  templateUrl: 'views/login.html',
10                 controller: 'MainController'
11             })
```

5.54. – Creación de rutas con AngularJS.

Una vez creada la ruta, crearemos el fichero HTML asociado: login.html

```
<div class="md-whiteframe-z3 row col-sm-6 col-sm-offset-3">
        <h3>Login</h3>
        <form name="loginForm">
            <md-input-container>
                <label>Nombre de usuario</label>
                <input type="text" ng-model="username" required>
            </md-input-container>
            <md-input-container>
                <label>Contraseña</label>
                <input type="password" ng-model="password" required>
            </md-input-container>

            <md-button class="btn-login md-raised md-primary" type="submit" ng-click="doLogin()">Iniciar
            sesión</md-button>
            <md-button class="btn-login md-raised md-primary" ng-href="/users">Quiero registrarme!</md-
            button>
            <br />
        </form>
</div>
```

5.55. – HTML de Login.

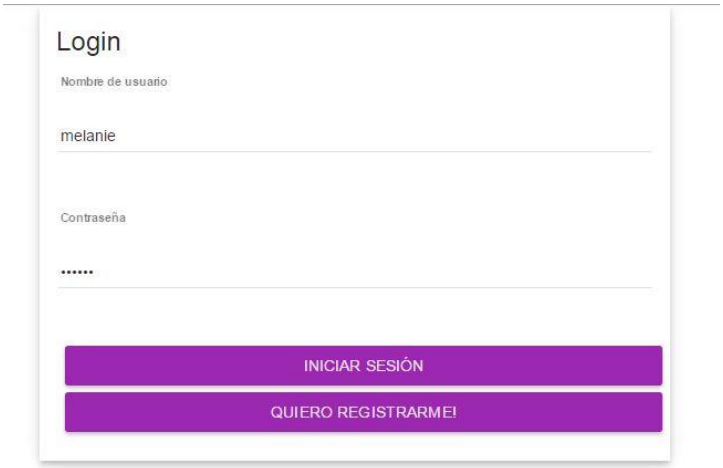
Para acabar, crearemos la función *"doLogin()"* en nuestro controlador:

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

            Auth.login($scope.username, $scope.password)
            .then( function(data){
                UserSrv.getUserId({'username' : $scope.username}).$promise
                    .then( function (user){
                        saveCookie(user.id);
                        $location.path('/books');
                    });
            });
        };
```

5.56. – Método doLogin() de MainController.

El resultado sería la siguiente vista:



5.57. – Página de Login que utiliza Material Design.

10.2. - Angular y su interacción con HTML

Angular nos permite asociar un valor a una variable desde la vista y desde el controlador. Tomemos como ejemplo la variable *username.*

Para utilizar esta bidireccionalidad declararemos la variable de la siguiente forma:

$scope.username

Con esto conseguimos que al modificar el valor del campo de texto asociado a esta variable, ésta adquiera inmediatamente el valor indicado en la vista. Esto es posible gracias a que en el código HTML del campo de texto, hemos indicado como parámetro de la directiva *ng-model* la variable *username.*

De la misma forma que acabamos de setear el valor de los modelos, podemos vincular otro tipo de funcionalidades mediante las directivas que aporta AngularJS (*ng-hide, ng-clic, ng-repeat,*etc.).

Aunque se trata de un tema de gran profundidad, destacaremos que el ciclo de Angular hace posible ver, sin necesidad de recargar la página, reflejados de forma inmediata los distintos cambios que se van produciendo en los modelos.

11. - Autenticación

Uno de los procesos más importantes de la aplicación es la autenticación. A lo largo de este apartado hablaremos de cómo se realiza dicho proceso recorriendo las distintas partes de la aplicación. Para ello, el flujo que seguiremos intentará emular los procesos que realiza el usuario de la aplicación.

11.1. - Front-end

11.1.1. - Comprobación y gestión del login

Tal y como se ve en la imagen, definimos una ruta que gestionará la pantalla de login asociándole un controlador. En este caso se ha asociado el controlador principal de la aplicación.

```
    .when('/login', {
        templateUrl: 'views/login.html',
        controller: 'MainController'
    })
```

5.58. – Ruta de login.

Lo primero que realizará dicho controlador es lanzar un evento cada vez que la URL cambie. Éste se encargará de evaluar si el usuario está registrado o no. Éste es el principal motivo por el cual hemos utilizado el controlador principal, debido a que éste siempre está cargado en la aplicación, y así podemos lanzar el evento en cada cambio de URL.

```
$rootScope.$on('$routeChangeStart', function() {

    $scope.loggedIn = Auth.isLoggedIn();

    if($scope.loggedIn === false) {
        $location.path('/login');
    } else {
        Auth.getUser()
        .then(function(data) {
            $scope.user = data;
        });
    }
});
```

5.59. – Evento que se encarga de controlar el cambio de rutas.

Si observamos la imagen veremos que dicha comprobación se lleva a cabo mediante el objeto Auth. Hablaremos más adelante de él.

Por otro lado tenemos la función “doLogin()”, que es llamada cuando el usuario envía la información para acceder a la aplicación a través del formulario de login. Esta función se encarga de procesar la información para que sea enviada al back-end mediante Auth.

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

    Auth.login($scope.username, $scope.password)
    .then( function(data){
        UserSrv.getUserId({'username' : $scope.username}).$promise
            .then( function (user){
                saveCookie(user.id);
                $location.path('/books');
            });
    });
};
```

5.60. – Método doLogin().

11.1.2. - Servicio de autenticación Auth

El servicio de autenticación Auth es el que se encarga de gestionar las llamadas al back-end relacionadas con la autenticación de los usuarios.

11.1.2.1. - Proceso de login

Después de que el usuario introduzca la información en el formulario, y que el controlador asociado a éste solicite a Auth que envíe la información, éste realizará una llamada al back-end enviando los datos, tal y como podemos apreciar en la imagen.

```
.factory('Auth', function($http, $q, AuthToken) {

    // create auth factory object
    var authFactory = {};

    // log a user in
    authFactory.login = function(username, password) {

        // return the promise object and its data
        return $http.post('/api/authenticate', {
            'username': username,
            'password': password
        })
            .success(function(data) {
                AuthToken.setToken(data.token);
                return data;
            });
    };
```

5.61. – Servicio de autenticación.

Se realiza un POST con los datos y si todo es correcto, el back-end nos devolverá un token de seguridad que setearemos de la siguiente manera.

```
.factory('AuthToken', function($window) {

    var authTokenFactory = {};

    // get the token out of local storage
    authTokenFactory.getToken = function() {
        return $window.localStorage.getItem('token');
    };

    // function to set token or clear token
    // if a token is passed, set the token
    // if there is no token, clear it from local storage
    authTokenFactory.setToken = function(token) {
        if (token)
            $window.localStorage.setItem('token', token);
        else
            $window.localStorage.removeItem('token');
    };

    return authTokenFactory;

})
```

5.62. – Servicio que setea un token para la autenticación.

11.1.2.2.-Comprobación de usuario logueado

Como vemos en la imagen del apartado anterior, existe un método llamado *getToken()* que utilizaremos para comprobar si un usuario ya está dentro la aplicación. Este método será llamado cuando el controlador principal dispare el evento asociado al cambio de ruta, tal y como hemos explicado antes.

```
authFactory.isLoggedIn = function() {
    if (AuthToken.getToken())
        return true;
    else
        return false;
};
```

5.63. – Método encargado de comprobar que un usuario está logueado.

11.2.- Back-end

11.2.1- Obtención del token

Cuando se realiza la petición POST para la autenticación, el sistema realiza un simple algoritmo. Para ello, comprueba en primer lugar si existe el usuario en la base de datos y en caso de no existir devuelve un error. Si el usuario existe, comprueba que su *password* sea correcto, ante lo cual, enviará como respuesta a dicha petición el token de usuario correspondiente. En caso contrario devolverá un error.

```
apiRouter.post('/authenticate', function (req, res) {
    User.findOne({
        username: req.body.username
        }).select('name username password').exec(function(err, user){
            if (err) throw err;

            if (!user) {
                res.json({
                    success: false,
                    message: 'Auth failed. User not found.'
                });
            } else if (user) {
                var validPassword = user.comparePassword(req.body.password);
                if (!validPassword) {
                    res.json({
                        success: false,
                        message: 'Authentication failed. Wrong password.'
                    });
                } else {
                    var token = jwt.sign({
                        name: user.name,
                        username: user.username
                    }, superSecret, {
                        expiresInMinutes: 1440 //24 hours
                    });

                    res.json({
                        success: true,
                        message: 'Enjoy your token!',
                        token: token
                    });
                }

            }
        });
```

5.64. – POST para la autenticación.

11.2.2.- Generación del token

Este proceso se lleva a cabo por la librería jsonwebtoken [14]. Ésta se encarga de recibir la información del usuario y codificarla junto con un tiempo de vida del token que queremos generar. Para más información podemos visitar su web oficial (http://jwt.io/).

VI. - Pruebas

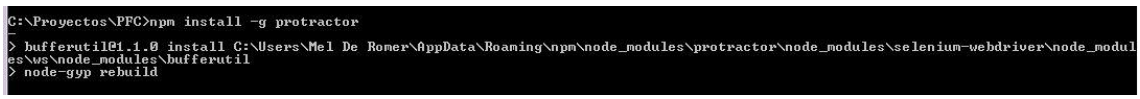
1. - Test funcionales con Protractor

Se han desarrollado una serie de test "*end to end*" para comprobar que la funcionalidad del sistema es la correcta. Para ello se ha utilizado la herramienta protractor [33], la cual nos permite ejecutar test en un servidor local. Para poder realizar dicha acción nos serviremos de Selenium [35], que es un *plugin* que permite simular acciones en un navegador.

1.1. - Preparación del entorno

A continuación se indican los pasos a realizar para montar el entorno sobre el que lanzaremos nuestro test. En primer lugar debemos instalar protractor:

```
npm install -g protractor
```



6.1. – Instalación de Protractor con npm.

Al ejecutar este comando se nos instala un webdriver manager, que es la instancia de Selenium que hemos citado anteriormente. Lo primero que debemos hacer es actualizarlo, ya que así nos aseguramos la total compatibilidad con los navegadores web:

```
webdriver-manager update
```

Una vez actualizado podemos lanzar ya el *manager*:

```
webdriver-manager start
```

Con este comando lanzamos una instancia de un servidor el cual se mantiene a la escucha en la siguiente URL: http://127.0.0.1:4444/wd/hub

6.2. – Servidor a la escucha en http://127.0.0.1:4444/wd/hub

De esta forma cuando lancemos un test y nuestro servidor este activo, será aquí donde se ejecuten.

Por otro lado, debemos configurar Protractor, para lo cual basta con crear un fichero *conf.js* como el siguiente:

6.3. – Fichero de configuración para test con Protractor.

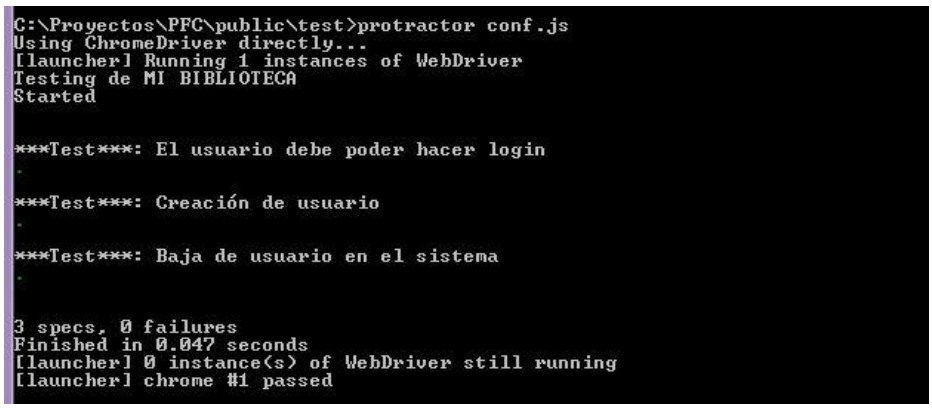
Como apreciamos en la imagen, uno de los parámetros que debemos indicar es dónde se va a estar ejecutando Selenium. Otro aspecto importante, es el atributo specs, en el cual debemos indicar el nombre del fichero o ficheros donde van a estar nuestros test. Es importante destacar que la ruta que se debe indicar aquí es relativa al lugar desde donde ejecutemos Protractor.

1.2. - Lanzando los test

Para lanzar los Test debemos ejecutar el siguiente comando

```
protractor conf.js
```

Tras esto veremos cómo se abre una ventana nueva del navegador y empieza a realizar aquellas acciones que le hayamos indicado en nuestros test. Mientras esto sucede, en la consola podemos ver el resultado de cada uno de ellos.



6.4. – Ejecución de los test con Protractor.

1.3. - Entendiendo los test

Las diferentes pruebas se han realizado con el framework Jasmine [34], que nos proporciona una sintaxis y una estructura para nuestras pruebas.

En primer lugar, debemos llamar a la función *describe*, la cual nos sirve como marco para ubicar nuestros test. Dicha función recibe como parámetro un string que suele ser el título general de los test que vamos a lanzar. Como segundo parámetro recibe un *callback*, que será la función que contendrá la ejecución pormenorizada de nuestras pruebas.

Dentro de dicha función llamaremos a la función "*it*()" tantas veces como test vayamos a realizar. Al igual que *describe*, recibe como primer parámetro un título y como segundo un *callback*, salvo que en este caso el *callback* ya contendrá el código de nuestro test.

Por último, cabe destacar la existencia de la función *beforeEach*(), la cual nos permite ejecutar el código que contenga ésta, al inicio de cada test.

2. - Pruebas con JSHint

JSHint [36] es una herramienta para detectar errores y problemas potenciales en código JavaScript.

JSHint se puede utilizar de varias formas. Podemos hacer uso de su herramienta online, o como es nuestro caso, instalar el paquete de node y ejecutar los test desde nuestra consola.

Para instalar JSHint lanzaremos el siguiente comando:

```
npm install -g jshint
```

```
C:\Proyectos\PFC>npm install -g jshint
C:\Users\Mel De Romer\AppData\Roaming\npm\jshint -> C:\Users\Mel De Romer\AppData\Roaming\npm\node_modules\jshint\bin\jshint
jshint@2.8.0 C:\Users\Mel De Romer\AppData\Roaming\npm\node_modules\jshint
├── strip-json-comments@1.0.2
├── exit@0.1.2
├── shelljs@0.3.0
├── console-browserify@1.1.0 (date-now@0.1.4)
├── minimatch@2.0.10 (brace-expansion@1.1.0)
├── htmlparser2@3.8.3 (domelementtype@1.3.0, entities@1.0.0, domhandler@2.3.0, domutils@1.5.1, readable-stream@1.1.13)
├── cli@0.6.6 (glob@3.2.11)
└── lodash@3.7.0
```

6.5. – Instalación de JSHint con npm.

Una vez instalado el módulo de JSHint, haremos una prueba sobre uno de nuestros archivos. Para esta prueba hemos elegido el controlador de listas, ListController.js.

```
1  'use strict';
2
3  angular.module('myLappbrary')
4      .controller('ListController', function($scope, $location, $cookies, ListSrv) {
5
6          $scope.createList = function(){
7
8              ListSrv.save({'name': $scope.name , 'userId' : $cookies.get('userId')}).$promise
9              .then(function() {
10                 console.log("DONE!!");
11             })
12         };
13
14         $scope.addBookToList = function(){
15             ListSrv.addBookToList({'books' : $scope.book})
16         };
17     });
18
```

6.6. – ListController.js con errores de sintaxis.

Ahora, en la consola ejecutamos el comando para testear el archivo:

```
jshint /public/js/controllers/ListController.js
```

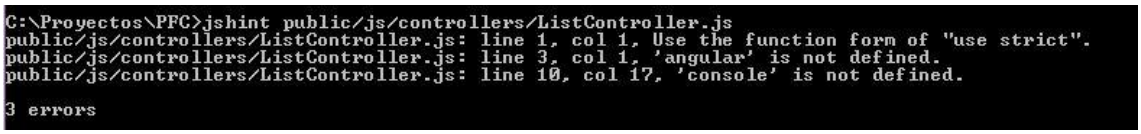
```
C:\Proyectos\PFC>jshint public/js/controllers/ListController.js
public/js/controllers/ListController.js: line 1, col 1, Use the function form of "use strict".
public/js/controllers/ListController.js: line 11, col 15, Missing semicolon.
public/js/controllers/ListController.js: line 15, col 59, Missing semicolon.
public/js/controllers/ListController.js: line 3, col 1, 'angular' is not defined.
public/js/controllers/ListController.js: line 10, col 17, 'console' is not defined.

5 errors
```

6.7. – Ejecución de pruebas con JSHint sobre ListController.js.

Aunque el resultado nos devuelva 5 errores, vemos que algunos nos son significativos, porque JSHint evalúa la sintaxis del código y no la orientación a objetos del lenguaje, y por tanto, no sabe de la existencia de angular ni de la instrucción *console.log()*, por ese motivo nos da el error de *'angular' y 'console' is not defined.*

Tras corregir los errores de sintaxis, volvemos a lanzar el test y vemos que ya no aparecen.



6.8. – Resultado de la ejecución del test con JSHint tras la corrección de los errores de sintaxis.

VII. - Conclusiones

1. - Resumen

En este proyecto se ha llevado a cabo el desarrollo de una aplicación web para el seguimiento de lecturas de los usuarios.

El proyecto ha comprendido varias etapas, como la especificación de requisitos, diseño, análisis, implementación, pruebas y conclusiones del trabajo realizado.

Durante la fase inicial del proyecto ha sido necesario consultar una gran cantidad de documentación, por hacer uso de tecnologías y frameworks relativamente nuevos.

2. - Valoración

Mi valoración global sobre el proyecto es muy positiva, ya que he obtenido una base de conocimientos nuevos y soltura sobre estas tecnologías que cada día se utilizan más.

La experiencia de haber utilizado tecnologías como MongoDB y Node.js, que antes de empezar con este proyecto desconocía, ha sido muy satisfactoria. Esto es debido a que supone un cambio a la hora de pensar, sirva de ejemplo el uso de bases de datos no relacionales. Además, la idea de utilizar JavaScript del lado del servidor supone trabajar de una forma distinta a lo que había experimentado en otros proyectos en los cuales se debía trabajar con lenguajes compilados.

3. - Trabajo futuro

A lo largo del desarrollo de la aplicación, han ido surgiendo una gran cantidad de ideas que podrían mejorar la aplicación.

Ya que a lo largo del desarrollo, la aplicación ha evolucionado y ha pasado a ser, más que solamente una aplicación para el seguimiento de lecturas, una red social, me gustaría seguir implementando funcionalidades típicas de las redes sociales.

- Votar comentarios de otros usuarios.
- Recomendaciones de libros según los gustos del usuario.
- Tener una serie de listas por defecto.
- Mostrar la actividad reciente de tus amigos en la aplicación.
- Poder hacer anotaciones personales sobre un libro.
- Múltiples lenguajes en la aplicación

En cuanto a posibles mejoras a nivel de estructura de la aplicación, podemos establecer las siguientes:

- Generar distintos ficheros de enrutamiento para desacoplar unos módulos de otros y de esta forma, favorecer la reutilización de código.
- Implementar una capa de persistencia que funcione bajo el paradigma de un modelo de datos relacional, para así favorecer la utilización de la aplicación en entornos que no soporten los modelos no relacionales.

Aunque la web es *responsive* y podría utilizarse perfectamente desde dispositivos móviles, podrían desarrollarse también en un futuro aplicaciones móviles que hicieran uso de la API REST que se ha implementado.

VIII. - Anexos

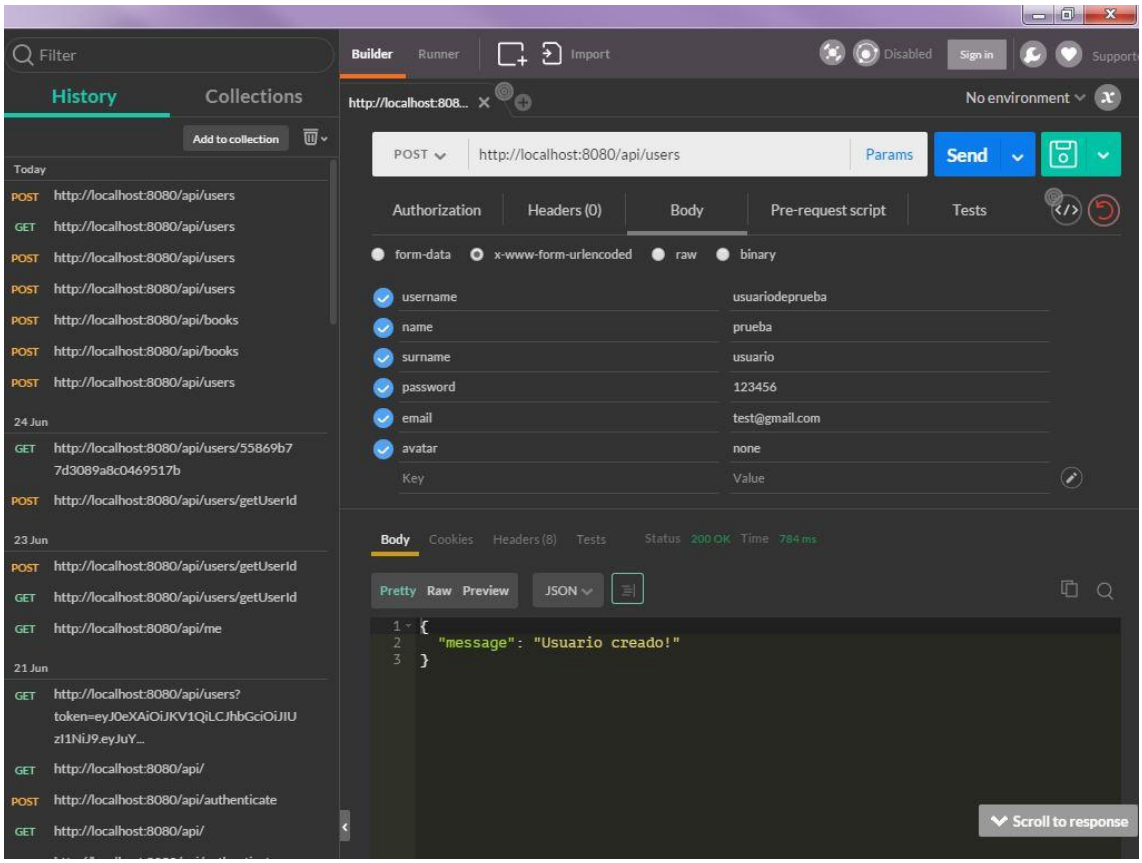
1. - Postman

Debido a la evolución de los protocolos de comunicación via web, se hace necesario el uso de aplicaciones que nos ayuden a los desarrolladores a trabajar más eficientemente bajo estos paradigmas. Una de las herramientas más adecuadas para ello es Postman, la cual podemos encontrar tanto como extensión para navegadores como aplicación.

Postman nos permite simular peticiones web de manera muy sencilla. Podemos configurar dichas peticiones con multitud de opciones: cabeceras a enviar, definir qué tipo de petición se hará (GET, POST, PUT, etc), establecer los parámetros que irán en el cuerpo de nuestra petición (así como el formato en el cual se enviarán éstos) o incluso simular entornos con distintos tipos de autenticación como OAuth, OAuth2 o Digest Auth.

Postman es compatible con cualquier servidor web, lo cual nos ayuda a capturar posibles errores en las llamadas que hacemos a nuestra aplicación o a URLs externas.

Un ejemplo del uso de Postman lo podemos ver en la siguiente imagen:



8.1. – POST a /users con Postman

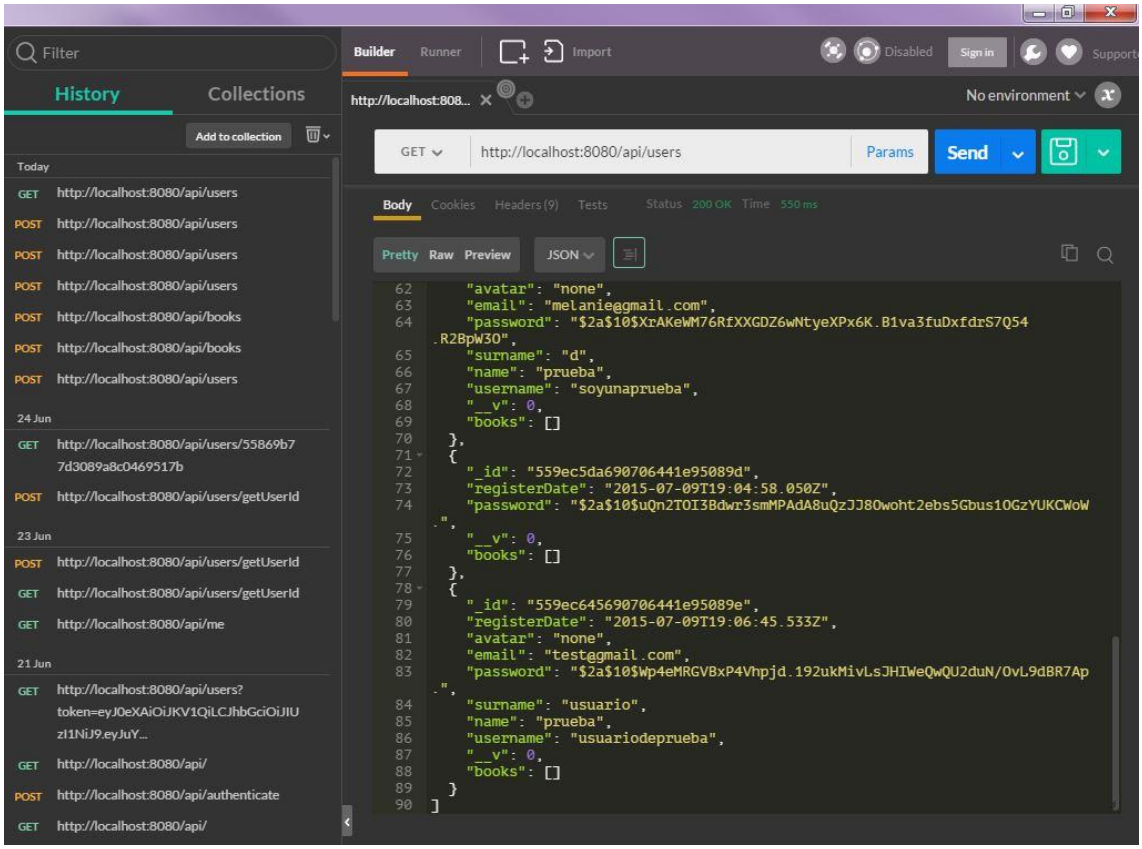
Hemos creado un usuario de prueba para nuestra aplicación.

En el punto 1 podemos ver la URL junto con el método HTTP seleccionado, en nuestro caso POST.

En el punto 2 especificamos los parámetros del cuerpo de la llamada.

Una vez construida nuestra petición, hacemos clic en SEND y el resultado de la llamada aparecerá en el punto 3.

Si ahora realizamos una llamada para obtener todos los usuarios, veremos que aparece el usuario que acabamos de crear en el paso anterior.



8.2. – GET a /users con Postman

2. – Obtención de una API key de Google Books

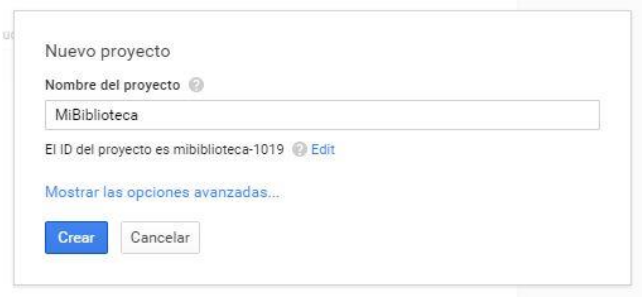
Las peticiones a la API de Google Books deben ir acompañadas de un identificador, que puede ser una API key o un token de acceso.

En nuestro caso, hemos utilizado una API key.

Una API key no es más que un identificador al cual se le asignan todas las operaciones que se hacen por el uso de los servicios de Google Books API.

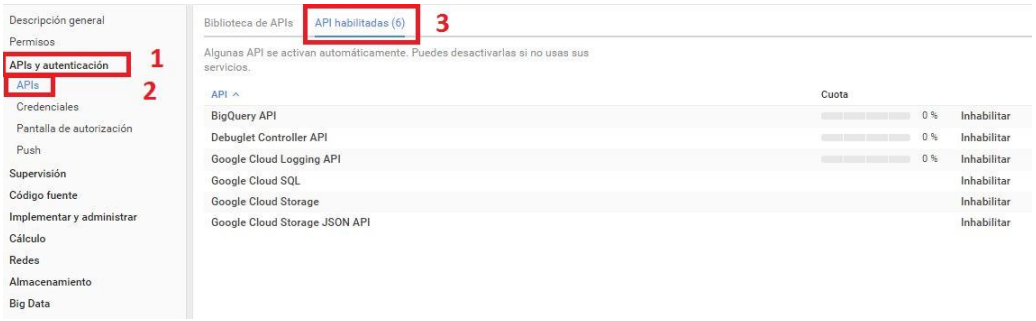
Para obtenerla, hay que seguir una serie de pasos:

- Ir a Google Developers Console e iniciar sesión con nuestra cuenta de Google.
- Seleccionar un proyecto, o en nuestro caso, crear uno nuevo llamado MiBiblioteca.



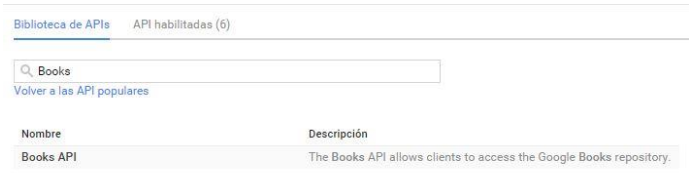
8.3. – Creación de nuevo proyecto en Google Developer Console.

- Hacemos clic sobre el nombre del nuevo proyecto que acabamos de crear y en la barra de la izquierda expandimos APIs y autenticación. Seleccionamos APIs en el menú que se despliega y en la ventana central, seleccionamos la pestaña API habilitadas.



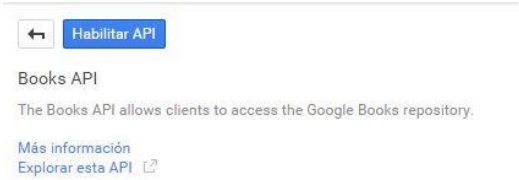
8.4. – Visualización de las APIs habilitadas

- Como podemos ver, no tenemos la API de Google Books activada. Para activarla, seleccionamos la pestaña Biblioteca de APIs y utilizamos el buscador para encontrarla.



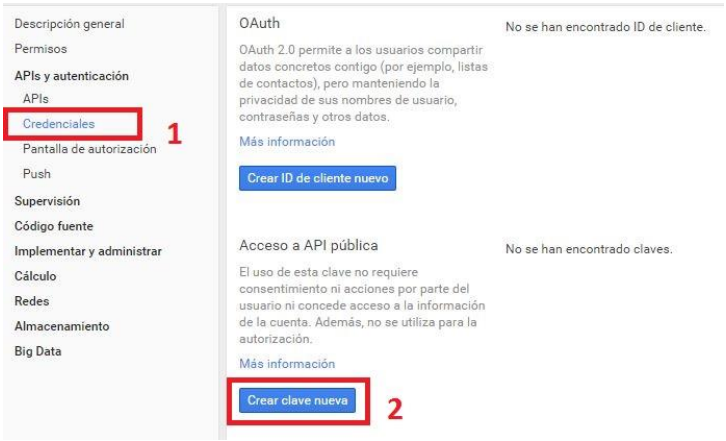
8.5. – Búsqueda de Google Books API en la Biblioteca de APIs de Google.

- Ahora hacemos clic sobre el nombre de la API y navegaremos a una nueva sección donde encontraremos un botón para habilitar la API.



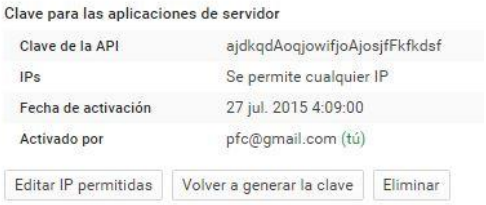
8.6. – Habilitar Google Books API.

- En el último paso para obtener la clave, haremos clic sobre la opción Credenciales del menú izquierdo y luego, clic sobre Crear clave nueva en Acceso a API Pública. Nos preguntará qué tipo de clave queremos crear y seleccionaremos Clave de servidor.



8.7. – Creación de nueva clave

En la siguiente ventana modal que aparecerá, haremos clic en Crear y obtendremos así la clave que necesitaremos para realizar las peticiones a la API de Google Books en nuestra aplicación.



8.8. – Visualización de la nueva API key.

3. - Sublime Text

Sublime Text es un editor de textos muy ligero y multiplataforma que aporta muchas características útiles a la hora de programar o editar código.

El sistema de resaltado de sintaxis de Sublime soporta distintos leguajes de programación o de marcado como ActionScript, ASP, C, C#, C++, CSS, Go, Haskell, HTML, Java, JavaScript, Lua, Matlab, Objective-C, Perl, PHP, Python, XML, etc.

Una característica muy útil de sublime es que permite tener abiertos varios documentos y organizados en pantalla de la forma que tú prefieras.

Con Sublime, podemos seleccionar una variable con el ratón, y pulsando CTRL + D, podemos ir seleccionando las siguientes coincidencias de la variable y modificarlas todas a la vez.

Una de las opciones más útiles que posee Sublime son los Snippets. Un snippet es un fragmento de código reutilizable. Con Sublime, podemos guardar en una plantilla un fragmento de código, de manera que escribiendo el nombre del snippet, seguido de la tecla TAB, se sustituirá ese texto por el de la plantilla. Además de crear snippets propios, Sublime permite importar snippets de otros usuarios.

Sublime permite la instalación de plugins que complementan las funcionalidades que ya tiene por defecto el editor.

Para poder incorporar plugins, debemos tener activado el Package Control, que es un gestor de paquetes que nos permite incluir más funcionalidades, como por ejemplo, resaltado de sintaxis para lenguajes no soportados por defecto (como CoffeeScript) o validadores de código, tomando como ejemplo JSHint el cual evalúa la correcta utilización del lenguaje JavaScript.

IX.- Bibliografía y enlaces consultados

[1] Félix Buendía García. Una guía para la realización y supervisión de proyectos final de carrera (PFC) en el ámbito de la web. 2008-247. Editorial Universidad Politécnica de Valencia, 2008.

[2] Pawel Kozlowski and Peter Bacon Darwin. Mastering Web Application Development with AngularJS. Packt Publishing, 2013.

[3] Chris Sevilleja and Holly Lloyd. MEAN Machine A beginner's practical guide to the JavaScript stack. Leanpub, 2015.

[4] Douglas Crockford. JavaScript: The Good Parts. Working with the Shallow Grain of JavaScript. O'REILLY, 2008.

[5] IEEE Recommended Practice for Software Requirements Specifications. IEEE Std. 830-1998, 1998.

[6] NodeJS.

https://nodejs.org/

[7] Mongoose.

http://mongoosejs.com/

[8] MongoDB.

https://www.mongodb.org

[9] ExpressJS.

http://expressjs.com/es/

[10] Stack MEAN.

http://meanjs.org/

[11] Stack MEAN.

http://mean.io/#!/

[12] AngularJS.

https://angularjs.org/

[13] Bower.

http://bower.io/

[14] JsonWebToken.

http://jwt.io/

[15] Wikipedia.

https://es.wikipedia.org/

[16] Javascript Full Stack. MEAN.

https://carlosazaustre.es/blog/desarrollo-full-stack-javascript-tambien-conocido-como-mean/

[17] NodeJS.

http://soloelectronicos.com/2014/05/27/que-es-y-para-que-sirve-node-js/

[18] NodeJS.

http://www.ibm.com/developerworks/ssa/opensource/library/os-nodejs/

[19] npm.

http://www.nodehispano.com/2012/04/una-introduccion-a-npm-nodejs/

[20] NodeJS y npm.

http://www.genbetadev.com/frameworks/node-js-y-npm

[21] Bower.

http://codehero.co/como-utilizar-bower-parte-i/

[22] NodeJS y ExpressJS

http://codehero.co/nodejs-y-express-instalacion-e-iniciacion/

[23] Gestión de dependencias con Bower.

https://openwebinars.net/gestionando-dependencias-en-front-end-con-bower/

[24] Yeoman.

https://openwebinars.net/yeoman-agiliza-la-creacion-de-tus-proyectos/

[25] Yeoman.

http://yeoman.io/codelab.html

[26] AngularJS y Yeoman.

https://www.airpair.com/js/using-angularjs-yeoman

[27] MVC.

http://www.oscarblancarteblog.com/2014/07/21/patron-de-diseno-modelo-vista-controlador-mvc/

[28] Sublime Text. Plugins.

http://unadocenade.com/una-docena-de-plugins-indispensables-para-sublime-text/

[29] Sublime Text. Plugins.

http://www.desarrolloweb.com/articulos/plugins-configuraciones-esenciales-editor-sublime-text.html

[30] Sublime Text. Package Control.

http://www.davidfraj.com/2013/11/instalar-sublime-text-2-package-control.html

[31] Git. Herramienta de control de versiones. Página oficial.

https://git-scm.com/

[32] GitHub.

https://github.com/

[33] Protractor

https://angular.github.io/protractor/#/

[34] Jasmine

http://jasmine.github.io/

[35] Selenium

http://www.seleniumhq.org/

[36] JsHint

http://jshint.com/

[37] MongoLab.

https://mongolab.com