

CBDMEAN Project

Miembros

Pérez Llorente, Ángel - angperllo@alum.us.es

Valero García, Luis - luivalgar@alum.us.es

Contenido

Introducción y dominio	3
Tecnologías y justificación	4
Instalación	5
Creación y estructura de proyecto plantilla	7
NoSQL vs SQL	8
Planificación y costes.....	11
Planificación.....	11
Costes:.....	12
Implementación.....	13
Base de datos.....	13
API.....	14
Aplicación web	15
Conclusiones.....	19
Bibliografía y referencias	20

Introducción y dominio

En esta memoria, se analizará el paquete de desarrollo MEAN, haciendo hincapié en la base de datos y la conexión de la aplicación con ella. Además, se dará una comparativa de los gestores de base de datos relacionales conocidos en la actualidad y el no relacional utilizado para este proyecto.

Para ello, se realizará un pequeño proyecto de microservicios característico del paquete de tecnologías seleccionado, más concretamente, el proyecto se basará en una API cuyo modelo (NoSQL) consistirá en un usuario que podrá iniciar sesión en el sistema y poseer una librería que contenga su música, película o videojuegos favoritos.

Dichas librerías serán públicas y los demás usuarios podrán filtrarlas por sus distintos atributos como nombre, autor, tipo o calificación, posteriormente tendrán la posibilidad de copiar cualquiera de los elementos buscados a su propia librería. La API será consumida por Angular, quien dará soporte al *frontend* de nuestra aplicación.

Tecnologías y justificación

En esta sección, se enumerarán las tecnologías seleccionadas junto a una breve descripción o justificación de uso. Posteriormente, se explicará detalladamente como se pueden instalar estas herramientas y crear un nuevo proyecto para utilizarlas.

1. **Ubuntu 18.04:** Sistema operativo sobre el que se ha desarrollado el proyecto. Se ha seleccionado debido a su facilidad para instalar herramientas de desarrollo y para virtualizar el desarrollo, reduciendo los problemas de configuración por la instalación en diferentes clientes.
2. **VirtualBox:** Aplicación que da soporte para utilizar la máquina virtual y lanzar el sistema operativo anterior.
3. **Node.js:** Es el encargado de la ejecución de código JavaScript en la parte del servidor o API. Cabe destacar, que utiliza ejecución asíncrona y no bloqueante, pudiendo paralizar la ejecución del código hasta que se recibe respuestas, a diferencia de otras tecnologías como PHP.
4. **Express:** Módulo de Node.js para facilitar el uso de la creación de una API en Node.js, utilizando el protocolo HTTP.
5. **Angular 7:** Framework de desarrollo *frontend*, es decir, de la parte de cliente en la aplicación web. A diferencia de los dos anteriores, utiliza TypeScript en lugar de JavaScript. La principal diferencia entre ambos lenguajes es que se puede añadir tipado explícito a los objetos. La característica por la que destaca más el uso de Angular se trata de, una vez compilado, la web se desarrolla en una sola página (SPA o *single-page application*), modificándola a través de código JavaScript.
6. **MongoDB:** Gestor de base de datos no relacional y *open source*, basados en documentos en formato JSON, los cuales, pueden tener diferentes atributos.
7. **Mongoose:** Módulo de Node.js utilizado como conector de MongoDB, elegido por su similitud con las *queries* nativas y su facilidad de integración con la tecnología utilizada. Incluyendo así, un middleware para la validación de objetos antes de la escritura en base de datos.
8. **MongoDB Compass:** Aplicación de escritorio de MongoDB para poder ver los datos o para facilitar la indexación de la base de datos.
9. **Git:** Utilizado como control de versiones junto a Github. El desarrollo completo del proyecto se encuentra en: <https://github.com/luisval11/cbd-project>
10. **Webstorm:** Entorno de desarrollo para JavaScript de JetBrains. Seleccionado por su cómoda instalación en Ubuntu, herramienta de *debug* e integración con repositorios de Git.
11. **Advanced Rest Client:** Aplicación para hacer peticiones a la API, se puede instalar como un *plugin* de Google Chrome, lo que nos facilitó su uso e hizo que nos

decantáramos por ella, en lugar de otras herramientas más conocidas como *Postman*.

En cuanto la justificación de la pila de trabajo MEAN (MongoDB, Express, Angular y Node.js), se han elegido dichas tecnologías por su buena comunicación entre ellas, utilizando la mayoría o desarrolladas con JavaScript. Todas usan el formato JSON para la transferencia de objetos.

Además, dada la asincronía de las herramientas y la programación orientada a eventos se pueden realizar muchas acciones en tiempo real sin necesidad de recargar la página o provocar prolongadas esperas a los usuarios de la aplicación.

Instalación

El proyecto ha sido desarrollado íntegramente en Ubuntu, por ello, el tutorial de instalación de las herramientas necesarias para la ejecución será sobre dicho sistema operativo.

Primero, abriremos el terminal y nos aseguraremos de que los paquetes previamente instalados están actualizados con el siguiente comando:

```
sudo apt-get update
```

Recordamos que “sudo” se utiliza para lanzar los comandos como administrador del sistema, si queremos evitar ponerlo en cada ejecución podemos ejecutar el comando siguiente:

```
sudo su
```

A continuación, procedemos a instalar las herramientas que serán utilizadas, comenzando por Node.js, que requiere la ejecución de los comandos siguientes:

```
sudo apt install nodejs  
sudo apt install npm
```

Npm nos servirá para instalar las librerías o dependencias del proyecto, incluyendo la instalación de Angular. Para este último, se necesitará ejecutar:

```
npm install -g @angular/cli
```

Por último, será necesaria la instalación de MongoDB, para ello se utilizará el siguiente comando:

```
sudo apt install -y mongodb
```

Una vez instalado, se aconseja utilizar MongoDB Compass, una aplicación de escritorio para poder consultar los datos o introducir directamente sin necesidad de hacer *queries* por consola. Puede ser descargado desde el siguiente enlace: <https://www.mongodb.com/products/compass>

Recordamos que para iniciar el servicio de MongoDB no basta sólo con instalarlo, si no que habrá que ejecutarlo manualmente con:

```
sudo mongod
```

Una vez instaladas todas las herramientas, pasaremos a explicar cómo instalar nuestro proyecto en concreto. Para hacerlo rápidamente lo instalaremos utilizando Git. En el siguiente cuadro de texto dejaremos los comandos para instalar Git y posteriormente nuestro proyecto:

```
sudo apt install git  
git clone https://github.com/luisval11/cbd-project
```

Por último, deberemos instalar las dependencias de Node.js en nuestro proyecto, poblar nuestra base de datos, compilar Angular y lanzar el servidor de Node.js. Para ello se ejecutarán los siguientes comandos, en el directorio del proyecto, siguiendo este orden:

```
npm install  
npm update  
mongo populate.js  
ng build  
node server
```

Tras esto ya podremos acceder a <http://localhost:3000> y utilizar nuestra aplicación web.

Creación y estructura de proyecto plantilla

En este apartado se explicará cómo crear un proyecto y la estructura que hemos seguido durante su desarrollo. Dicho esto, un proyecto en MEAN no tiene una estructura predefinida y deja a elección del desarrollador su estructuración o condicionarlo al desarrollo del proyecto.

Un proyecto en Angular se divide en componentes, los cuales pueden contener más subcomponentes dentro, pudiendo ser independientes entre ellos. Estos componentes contendrán siempre su representación en HTML y CSS además de un archivo *TypeScript* que contendrá la lógica de dicho componente. El componente principal será “app”; para generarlo se utilizará el siguiente comando:

```
ng new [nombre-proyecto]
```

Con esto ya tendríamos la estructura básica de Angular creada, nosotros dentro de app añadimos un directorio “*models*” donde se encontraría los modelos utilizados para el *frontend* de la aplicación. Por otro lado, en Angular se trata de minimizar el tamaño de los componentes, llegando incluso a tener un componente que sea un simple botón de HTML para modularizar el código lo máximo posible y poder mantener o encontrar errores de una manera óptima, además de evitar caer en la duplicación de código. Otra ventaja con la que cuenta Angular es su CLI (*Angular Command Line Interface*), puesto que podemos generar prácticamente todo a través del terminal. Para crear un componente usaremos el comando:

```
ng g c [nombre componente]
```

Respecto a Angular, lo único que nos queda explicar es el uso de servicios. Serán los encargados de pedir datos a otro servidor mediante HTTP, es decir, los que se comunicarán con la API de Node.js. Para crearlo también se puede usar un comando, concretamente:

```
ng g s [nombre servicio]
```

Una vez sabemos como funciona Angular, pasaremos a crear la estructura de Node.js. Para ello, nos situaremos en el primer directorio generado cuando creamos la estructura base de Angular, es decir, en el directorio que se llame como el nombre del proyecto que elegimos. Una vez ahí, ejecutaremos el siguiente comando para inicializar el proyecto en Node.js:

```
npm init --yes
```

Esto nos creará un paquete llamado “package.json”, donde se indicarán las dependencias del proyecto. Tras esto, crearemos, manualmente, los siguientes archivos/directorios:

- **Server.js:** Archivo donde se encontrarán las conexiones con la base de datos, configuración de la API, ruta al proyecto de Angular compilado para usar las vistas, puerto y otros detalles necesarios para lanzar el servidor de manera satisfactoria.
- **Populate.js:** Archivo js para ejecutar con MongoDB, (como vimos en el apartado anterior) y poblar nuestra base de datos.
- **Models:** Directorio donde se encontrarán los esquemas de *Mongoose*, es decir, el modelo del *backend*. Además, se encontrarán las validaciones y restricciones de los distintos modelos. Cada archivo se corresponderá con una colección de MongoDB.
- **Routes:** Directorio donde se encontrará el archivo “api.js”; con los distintos enlaces y métodos a ejecutar. Además de las restricciones como, por ejemplo, si un método no es accesible por un usuario que no haya iniciado sesión en nuestro sistema web.
- **Controllers:** Directorio que contendrá todos los métodos llamados por la API, con la lógica de negocio correspondiente y encargado de mandar las respuestas al cliente.

NoSQL vs SQL

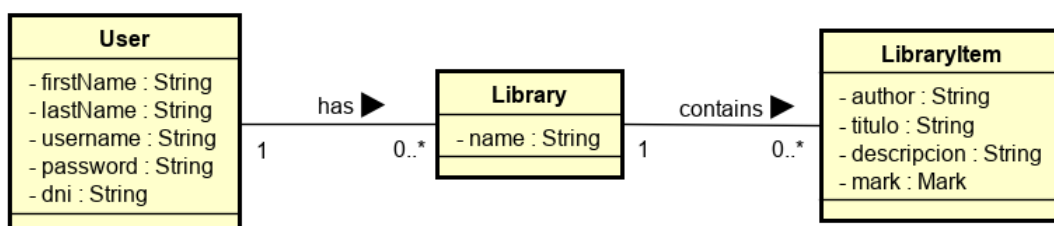
En la actualidad, contamos con 2 opciones principales para elegir el tipo de base de datos que utilizaremos para el desarrollo del proyecto. Dichas opciones son: las bases de datos relacionales (SQL) y las no relacionales (NoSQL). Cada una de ellas se ajusta a un tipo de proyecto, por ello es conveniente tener conocimientos en ambas para ser versátiles y poder adaptarnos a cualquiera de ellas.

- **Escalabilidad:** las bases de datos no relacionales tienden a escalar de manera horizontal, es decir, son capaces de manejar un alto volumen de datos. Utiliza múltiples anfitriones para soportar dicha carga de datos, habitualmente no permitiendo el uso de operaciones del tipo *JOIN*, ya que dificulta la escalabilidad del sistema. Por el contrario, las bases de datos relacionales utilizan *JOIN* para acceder a las informaciones de otras tablas existentes almacenadas en la base de datos, haciendo más complejo su uso. Debido a que las relacionales garantizan ACID (atomicidad, consistencia, aislamiento y durabilidad), por ello, necesitarían sincronizar todas las transacciones y operaciones, paralelamente evitando el interbloqueo de dichas operaciones, mientras que las bases de datos no relacionales siguen el teorema CAP (Consistencia, disponibilidad y

Tolerancia a fallos), pero habitualmente solo podremos disponer de 2 de los 3 conceptos. En los casos de sistemas web, se prioriza la disponibilidad y la tolerancia a fallos, ya que es necesaria una mayor agilidad en las consultas.

- **Lenguaje:** las bases de datos no relaciones no utilizan un esquema definido uniforme para los documentos, es decir, es mucho más flexible a la hora de definir los documentos que almacenará la base de datos ya que no es necesario definir su estructura previamente, así como añadir más atributos. Por ello, cada documento puede tener su estructura de manera única, perteneciendo a una misma colección con otros documentos distintos a él. Por el contrario, las bases de datos relaciones son más restrictivas en el aspecto de la sintaxis, ya que requiere del uso de esquemas predefinidos que establezcan la estructura de los datos que serán almacenados en ella. Deben de seguir todos la misma estructura y los cambios en alguna de las estructuras puede provocar un cambio significativo en el sistema. Además, las bases de datos relacionales al utilizar un único lenguaje para la definición y control de los datos (SQL).
- **Estructura:** Las bases de datos no relacionales constan de seguir una estructura, como hemos mencionado anteriormente, basada en documentos, como es la utilizada en este proyecto (MongoDB), pares de clave-valor (Dynamo, Voldemort), bases de datos orientadas a grafos (Neo4j) o orientadas a columnas (Cassandra). Nos centraremos en analizar a continuación en MongoDB, la cual hemos utilizado en el proyecto. MongoDB se caracteriza por el uso del clave-valor, pero el campo valor es un documento que identifica la base de datos. Almacena la información habitualmente en un formato como JSON o XML, utilizando una clave única para cada documento. MongoDB nos permite realizar búsquedas dentro del documento de manera más eficientes y versátiles y establecer relaciones entre ellos sin la necesidad de usar *joins*. En el caso de las relacionales, requieren que sigan los datos una estructura basada en tablas.

Anteriormente, hemos comentado las características de las bases de datos de manera genérica, ahora pasaremos a analizar en profundidad la desarrollada en este proyecto. Para ello una buena metodología sería empezar comparando modelos. En el caso de la no relacional, lo podemos encontrar en el apartado [implementación de la base de datos](#), en cambio, el modelo relacional lo veremos en la siguiente imagen:



Como podemos observar, si necesitáramos acceder a las librerías de un usuario en concreto, habría que pasar por 2 *joins* tal y como se comentó anteriormente esto aumenta considerablemente la complejidad y reduce el rendimiento al tener que consultar tantas tablas. Sin embargo, tiene una ventaja frente a nuestro modelo, y es a la hora de mostrar todos los ítems de una librería. En el caso no relacional, no queda otra posibilidad que recorrer todos los usuarios e ir añadiendo los objetos pertenecientes a esa librería, en el relacional se podría utilizar esa única tabla sin tener en cuenta las demás.

En cambio, cualquier otra operación será más costosa, ya que tienen muchas restricciones debido a los ids y multiplicidades. En la no relacional, algo significativo es que la clase *library* podría desaparecer como tal, ya que simplemente con tener esos atributos en los documentos no haría falta gestionar sus ids.

Si se deseara añadir una librería nueva, por ejemplo, teatros; en el no relacional conforme un usuario quisiera añadir un elemento a esta librería se le podría crear ese atributo nuevo, pero la relacional requeriría de una operación previa en la que el administrador creara todas estas librerías para todos los *users*, lo cual sería muy costoso computacionalmente. Esto se debe a que en ningún caso permitimos a los usuarios crear sus propias librerías, si no que vienen dadas por defecto por nuestra aplicación web.

Tras leer esto puede surgirnos una duda, ¿cuál sería la operación equivalente al *join* para las no relacionales? o, dicho de otra manera, ¿cómo accedemos a subdocumentos dentro de *arrays*? La manera más efectiva de hacer esto en MongoDB es con el operador `$[<identificador>]` y usando *arrayFilters*. El primero se encarga de otorgar un nombre a una variable, que corresponderá a cada elemento de un *array*, por ejemplo, *user.music.\$[musicItem]*; *musicItem* será la variable donde se guarden los valores de cada elemento como si de un bucle *for-each* se tratase. Para establecer condiciones de filtrado como, por ejemplo: buscar el *musicItem* con nombre X, *ArrayFilters* es el campo que soporta este tipo de condición especificado anteriormente.

Otro tipo de operación a destacar que posee MongoDB, mientras que las bases de datos relaciones no lo hacen, es la agregación. Este tipo de operación puede recorrerse diferentes documentos según: el filtrado que se requiera, añadir o modificar atributos sin persistencia durante el proceso o agruparlos según un atributo concreto. Nosotros usamos este tipo de operación para conseguir todas las librerías de cada usuario, agrupándolas por el tipo de ítem que es, entendiendo como tipo la librería a la que pertenece. No obstante, podríamos aumentar las prestaciones de esta operación, usándola como *map reduce*; varios documentos iguales donde interesa agruparlos en uno solo cambiando un atributo, por ejemplo, sería interesante que en las comandas de un bar, añadir todas las comidas en una única sumando el precio de cada comida. Este tipo de operaciones dan un amplio abanico de posibilidades que no tienen las bases de datos relacionales.

Planificación y costes

Planificación

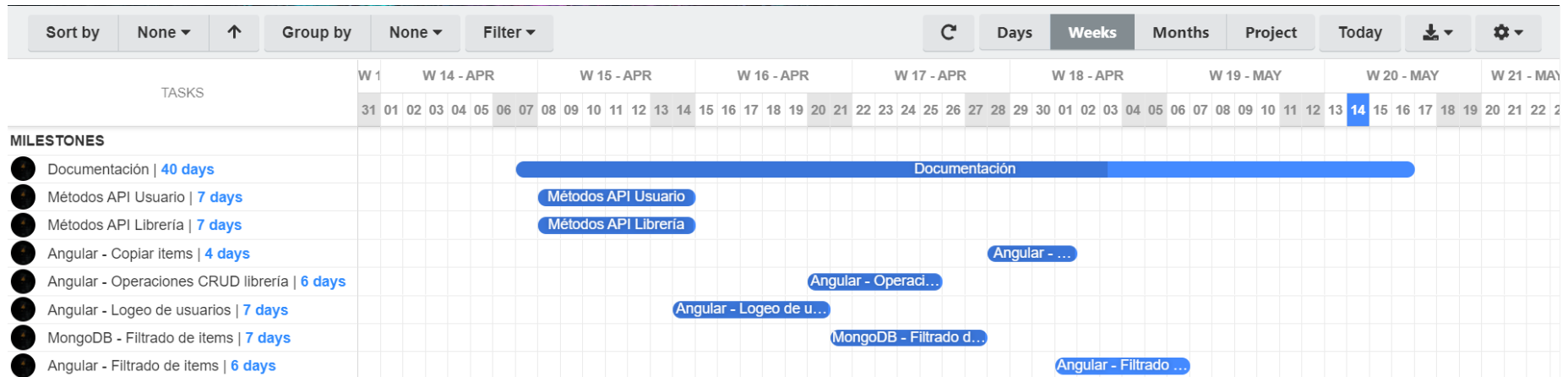


Ilustración 1 Planificación para el desarrollo del proyecto

El plazo para el desarrollo del proyecto está comprendido desde la confirmación de poder realizar este proyecto, en nuestro caso el 11/03, con una fecha límite de entrega establecida en la asignatura para el día 19/05/2019.

Hasta el comienzo de la implementación, período comprendido en el gráfico anterior, consistió en la formación e investigación en las tecnologías a utilizar durante el proyecto. A continuación, presentamos el desglose de tiempo establecido para el desarrollo. Hemos estimado inicialmente un total de 40 horas en total para el desarrollo completo del proyecto:

- **Documentación**: se desarrollaría de manera paralela desde el comienzo de la implementación, para llevarla al día con los avances del proyecto.

- **Métodos de comunicación con la API de Node.js:** Son las operaciones esenciales para el funcionamiento de la aplicación, por tanto, inicialmente estimamos desarrollar los modelos de *frontend* y *backend* y su implementación durante la primera semana de desarrollo.
- **Operaciones de *frontend* con Angular:** Finalizada la API y la conexión entre MongoDB y Node.js, haciendo uso de Express, desarrollaríamos las acciones que estimamos inicialmente para el desarrollo de nuestro proyecto:
 - **Inicio de sesión** de los usuarios en nuestro sistema.
 - **Operaciones CRUD de los ítems** (crear, consultar, actualizar y eliminar).
 - **Copiar los ítems** de otros usuarios.
 - **Filtrado de los ítems** según parámetros introducidos por el usuario como por autor, palabra en la descripción o nota.

Durante el desarrollo del proyecto, no han sido necesarias replanificaciones ocasionadas por retrasos del desarrollo o cambios realizados en lo inicialmente planificado.

Costes:

Para el cálculo de desarrollo del proyecto no se considerará ningún tipo de coste directo monetario para los desarrolladores, ya que se trata de un proyecto únicamente académico y el desarrollo de este proyecto se ha realizado en las instalaciones de la Escuela Técnica Superior de Ingeniería Informática (ETSII), por lo que tampoco se considerarán costes indirectos monetarios.

Por lo citado anteriormente, para contabilizar el coste de proyecto vamos a incluir el coste en tiempo invertido estimado para el desarrollo frente al coste de tiempo real, que ha sido monitorizado con el uso de la aplicación **Clockify**, para el registro de ese coste.

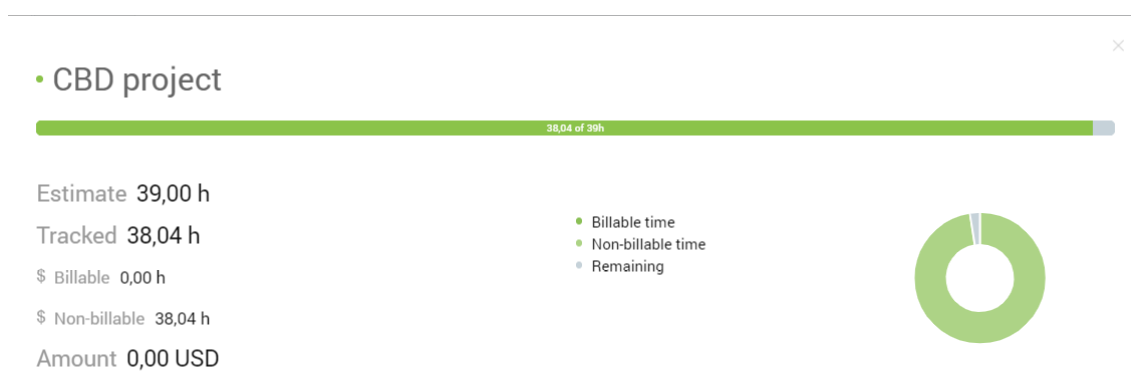


Ilustración 2 Estimación del coste temporal y real del proyecto

Como podemos apreciar, hemos cumplido con las estimaciones realizadas para el desarrollo de la aplicación en el plazo establecido para realizarlo.

Implementación

En esta sección se hablará de cómo se han diseñado e implementado las diferentes partes del proyecto, profundizando aún más en las tecnologías y su funcionamiento.

Base de datos

En cuanto a la base de datos, recordamos que MongoDB funciona a través de colecciones, que almacenan documentos en formato JSON, cuyos atributos no tienen por qué coincidir, a diferencia de las tablas en las bases de datos relacionales.

En nuestro caso, tanto la creación como la validación se lleva a cabo con Mongoose y hemos decidido tener una única colección para este proyecto, aunque se barajaron otras opciones como tener todos los objetos de las librerías en una colección diferente a la de usuarios, pero descartada finalmente debido a que tendría un cierto sentido relacional que queríamos evitar al usar MongoDB.

A continuación, se analizará el modelo de datos propuestos para los documentos, comentando el significado de cada atributo y sus restricciones a la hora de validarlos, si las hubiera.

- **firstName:** Será el nombre del usuario del sistema, como única restricción tiene el ser obligatorio.
- **lastName:** Apellido del usuario, también obligatorio.
- **username:** Alias del usuario, para el inicio de sesión. Además de ser obligatorio, este atributo será único en todos los documentos y poseerá una longitud mínima de 3 caracteres.
- **password:** La contraseña que será cifrada usando el algoritmo de *hashing* SHA256, también es obligatorio y tiene una longitud mínima de 3.
- **dni:** El DNI del usuario, como única restricción tendrá la validación habitual de un DNI. Este campo fue incluido para comprobar que se pueden realizar validaciones personalizadas usando Mongoose.
- **music, films, videogames:** Serán 3 *arrays* que contendrán los mismos atributos para definir los objetos de cada librería. Más concretamente, los atributos de estos subdocumentos serán:
 - **author:** Autor de la obra, atributo obligatorio.
 - **title:** Título de la obra, atributo obligatorio.
 - **description:** Descripción de la obra.
 - **mark:** Nota de la obra dada por el usuario. Como restricción es un enumerado pudiendo tomar únicamente los valores: *masterpiece, great, good, bad, horrible* o *pending*.

API

Ahora analizaremos como se accede a la base de datos a través del servidor de Node.js. Para ello, hemos decidido utilizar un formato de tabla para describir como funciona nuestra API Rest.

Método	URI	Descripción
GET	/api/login	Se le pasa en el <i>header</i> una <i>Authorization basic</i> y devuelve un token que se guardará en las <i>cookies</i> del navegador
GET	/api/logout	Destruye la <i>cookie</i> anterior.
GET	/api/getPrincipal	Devuelve el usuario que ha iniciado sesión en el sistema.
GET	/api/user	Devuelve todos los usuarios del sistema
POST	/api/user	Recibe en el <i>body</i> un usuario con los atributos vistos anteriormente y devolverá usuario creado (200) o los errores de validación por los que no ha podido ser creado (400)
PUT	/api/user	Similar al anterior, salvo que el <i>put</i> requerirá la id también en el <i>body</i> puesto que está actualizando el usuario.
DELETE	/api/user	Borra el usuario
GET	/api/user/library	Devuelve un JSON con un array de tus librerías de música, videojuegos y películas. Se podría filtrar a una única librería añadiendo el parámetro <i>?filter=music</i> ; si se añade más de una se podrá filtrar por esas 2.
GET	/api/library	Devuelve todas las librerías disponibles en el sistema, con el mismo formato del JSON anterior. Aquí se puede usar los siguientes parámetros para filtrar en la url: <ul style="list-style-type: none"> - library: obligatorio si se quiere filtrar, indicará sobre que librería se realizará el filtro - search: un string que buscará alguna coincidencia en el autor, título o descripción de la obra. - mark: nota del objeto guardado en la librería.

GET	/api/library/:id	Devuelve el objeto de la librería cuyo id sea :id o un 400 si no existe. En este caso también se indicará el tipo de librería en el que estaba, music, films o videogame, en el objeto que se devuelve.
POST	/api/user/library/:type	Recibe en el <i>body</i> un objeto JSON y lo añade al tipo de librería que se indique en el :type.
PUT	/api/user/library/:type	Similar al anterior, pero al ser una actualización requerirá en el <i>body</i> el id del objeto a editar
DELETE	/api/user/library/:id	Elimina el objeto con id :id de tus librerías.

Tabla 1 Documentación de API REST

Aplicación web

Retomaremos los pasos que especificamos en la instalación de nuestro proyecto. Por ello, seguiremos desde que introducimos en la URL <http://localhost:3000>.

Seremos redirigidos automáticamente a la vista de filtrado de ítems de la web. Se verá una vista similar a la que se muestra a continuación pudiendo consultar todos los ítems disponibles en la base de datos y ver los atributos que hayan incluido los usuarios.

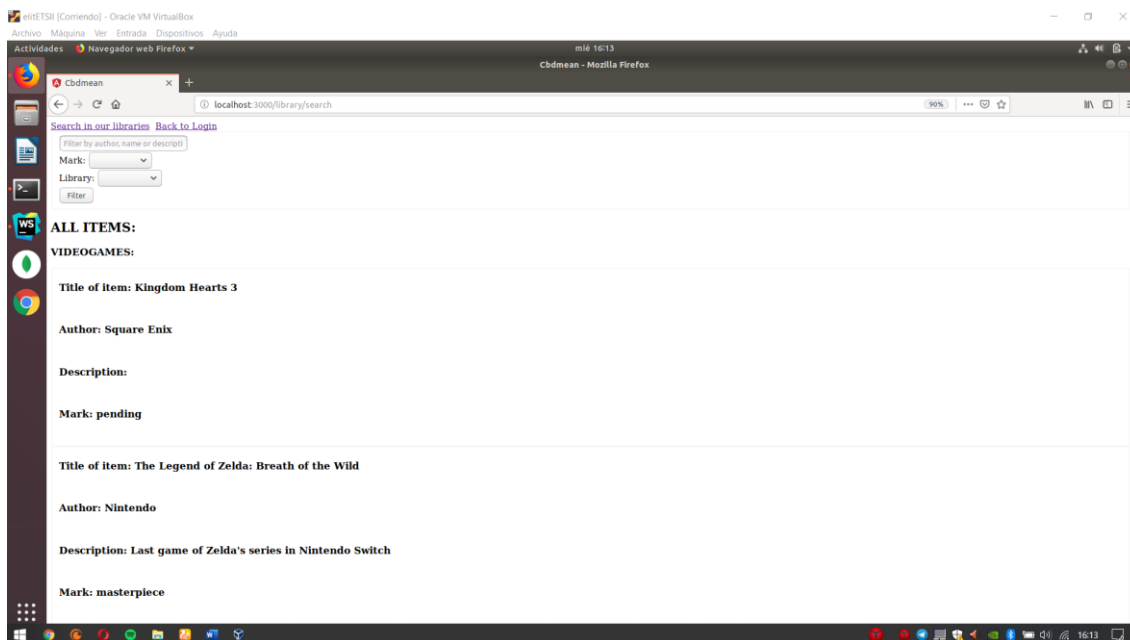


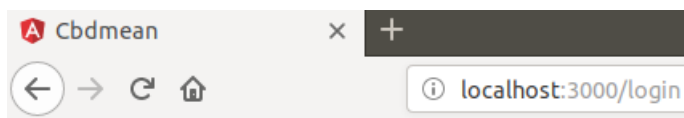
Ilustración 3 Vista de filtrado de la aplicación

En esta vista, está a disposición del usuario las siguientes acciones:

- **Iniciar sesión o My library:** Link hacia la vista de inicio de sesión o, en el caso de haber iniciado sesión, un enlace a *"My library"*, es decir, el conjunto de librerías propias del usuario.

- **Filtrar ítems:** Utilizar el formulario de búsqueda introduciendo: una palabra, una "mark" y especificando a que librería se realizará la consulta, o si se realiza sobre las 3 a la vez, seleccionando en este ultimo caso la opción "All".

En el caso de que el usuario decidiese iniciar sesión, hará clic en el link de "Go to Login", lo que nos llevará a la siguiente vista:



[Search in our libraries](#)

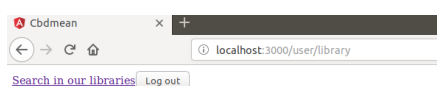
Log In

Username

Password

Ilustración 4 Vista de inicio de sesión

En dicho formulario introduciremos las credenciales para iniciar sesión en la web. Hemos preparado 2 usuarios distintos en el la base de datos poblada previamente. Dichos usuarios son: "admin/admin" y "admin1/admin1". Nosotros utilizaremos el usuario "admin/admin". Una vez introducidos en los correspondientes campos, se nos redirigirá a la vista similar a la siguiente, donde se encuentran todos los ítems de nuestras librerías



My Library:

[Create an item](#)

Video games:

<p>Title of item: Kingdom Hearts 3</p> <p>Author: Square Enix</p> <p>Description:</p> <p>Mark: horrible</p> <p>Edit this item Delete this item</p>
<p>Title of item: The Legend of Zelda: Breath of the Wild</p> <p>Author: Nintendo</p> <p>Description: Last game of Zelda's series in Nintendo Switch</p> <p>Mark: masterpiece</p> <p>Edit this item Delete this item</p>

Music:

<p>Title of item: Cry me out</p> <p>Author: Pixie Lott</p> <p>Description:</p> <p>Mark: masterpiece</p>

Ilustración 5 Vista de "My library"

Como podemos observar en la imagen disponemos de distintas acciones para el usuario:

- **Crear un nuevo ítem:** Hará clic en el link *"Create an item"* y pasará a una vista (Ilustración 6) donde introducir los valores previamente citados que puede tomar un ítem. Una vez completados dichos campos, haremos clic en *"Save"* y seremos redireccionados a la vista *"My library"* donde podremos comprobar que el nuevo ítem ha sido creado satisfactoriamente.
- **Editar un ítem ya existente:** Haciendo clic en el link *"Edit this item"*, nos llevará a una vista de edición (Ilustración 7) para modificar los campos que desee el usuario. Una vez finalizados los cambios, hará clic en *"Save"* para guardar esos cambios en base de datos y volveremos a ver los cambios reflejados en la vista de *"My Library"*.

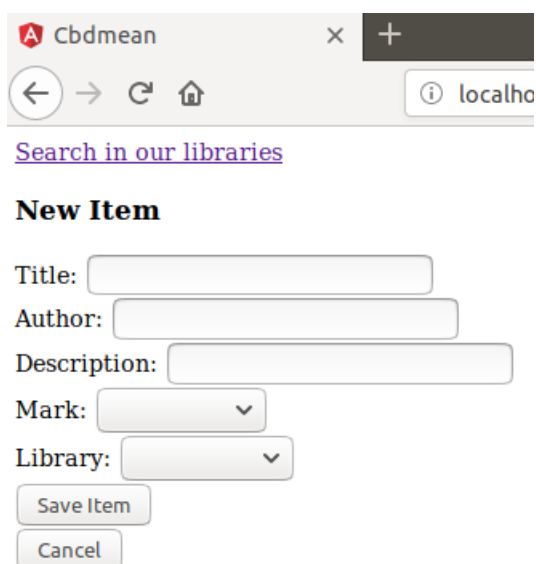


Ilustración 6 Crear un nuevo ítem

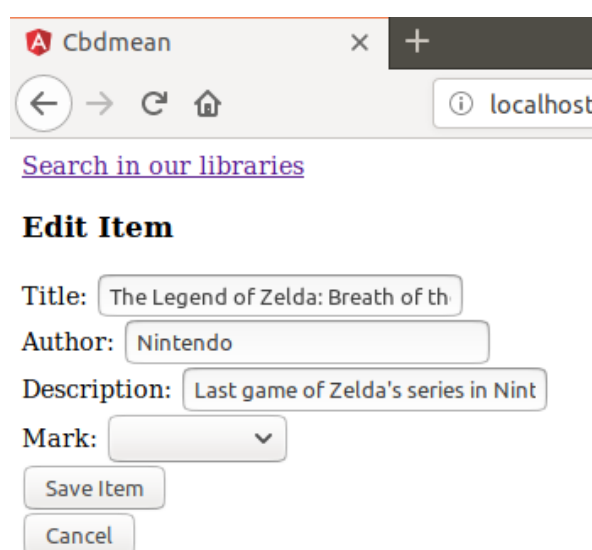


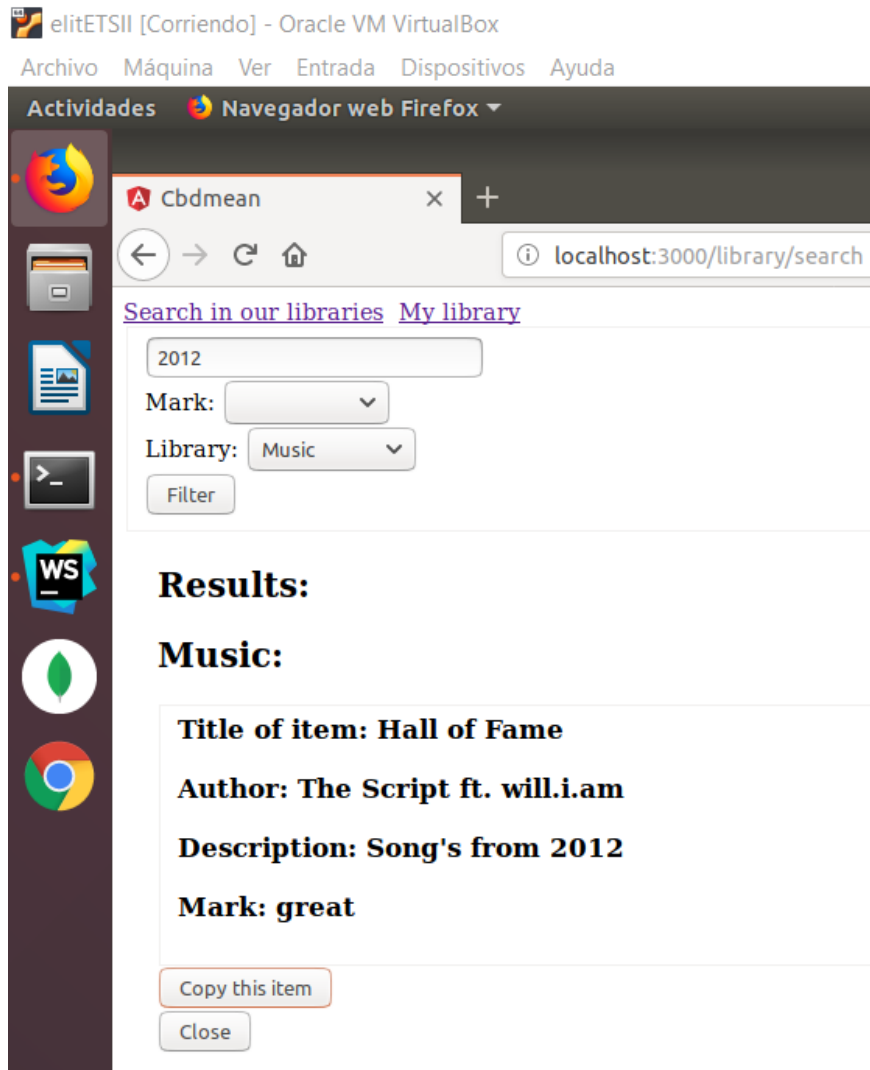
Ilustración 7 Editar un ítem existente

- **Borrar un ítem de una librería:** En caso de que el usuario desee eliminar un ítem de su librería, tan solo necesita hacer clic en el botón *"Delete this item"* que se encuentra debajo de dicho ítem y automáticamente será borrado de la base de datos del sistema y actualizándose, sin necesidad de recarga, la librería actualizada de nuestro usuario.

Para finalizar, si el usuario decide buscar ítems en la aplicación hará clic en *"Search in our libraries"*, introduciría los datos en los campos correspondientes para obtener en función de ellos, unos resultados que coincidan con todos los campos especificados. Por ver algunos de los ejemplos que devuelvan resultados:

- Ítems por descripción: *"All"* y *"from"*
- Ítems por nota: *"All"* y *"Masterpiece"*
- Ítems por librería y autor: *"Music"* y *"The script"*
- Ítems por librería y nombre de ítem: *"Films"* y *"Padrino"*

- Ítems por librería y descripción: “*Music*” y “2012”



Este sería el ejemplo de una búsqueda y sus resultados. En el caso que hayamos iniciado sesión, tendremos un botón disponible, en cada uno de los resultados que aparezcan, para realizar una copia de dicho ítem, incluyéndolo en nuestra biblioteca con los mismos valores a excepción del campo “*mark*” que se establece por defecto a “*pending*”. En el caso de copiamos dicho ítem, seremos redirigidos a la vista de nuestra librería y podremos comprobar que dicho ítem ha sido copiado satisfactoriamente.

Conclusiones

Durante el desarrollo de este proyecto, hemos extraído algunas resoluciones sobre las tecnologías con las que hemos trabajado:

- **NoSQL vs SQL:** De cara al futuro, el uso de NoSQL seguirá extendiéndose gracias a algunas de las características que nos han mostrado las bases de datos no relacionales. En este caso, MongoDB gracias a su almacenamiento de datos en un formato legible para el desarrollador, así como su escalabilidad para aumentar el soporte para usuarios. Pero no debemos olvidar que tiene otras desventajas en las que destaca SQL, por tanto, es necesario saber identificar, según las necesidades o requisitos del sistema, cuál es el tipo de base de datos que vamos a utilizar, ya que podremos dar un mejor producto para nuestro cliente.
- **Coordinación entre las tecnologías utilizadas:** elegimos el uso de MEAN STACK debido a que se trata de un bloque de tecnologías actualmente con mucha demanda en el mercado laboral, lo cual podría ser diferenciador si tuviéramos conocimientos sobre él. No obstante, nos ha sorprendido la buena comunicación existente entre ellas para trabajar y desarrollar una aplicación web utilizando la API REST diseñada para las comunicaciones y seguir siempre el formato JSON.

Bibliografía y referencias

1. <https://www.w3schools.com/nodejs/>
2. <https://nodejs.org/en/>
3. <https://expressjs.com/>
4. <https://angular.io/features>
5. <https://mongoosejs.com/>
6. <https://www.mongodb.com/products/compass>
7. <https://www.jetbrains.com/webstorm/>
8. https://www.researchgate.net/publication/327834151_SQL_vs_NoSQL
Gafaar, Alshafie. (2017). SQL vs NoSQL.
9. Transparencias de la asignatura de Complementos de Bases de Datos