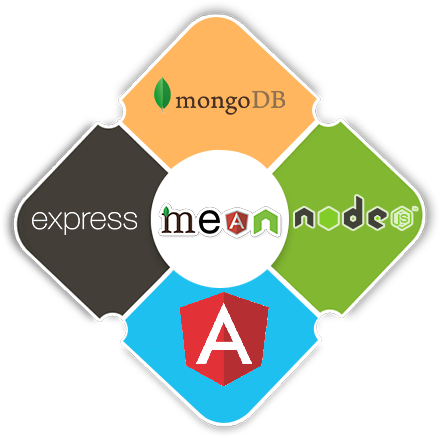
****

**[Por decidir]**

Miembros

**Pérez Llorente, Ángel** -[angperllo](mailto:serortagu@alum.us.es)@alum.us.es

**Valero García, Luis** -l[uivalgar@alum.us.es](mailto:luivalgar@alum.us.es)

Contenido

[Introducción y dominio 3](#_Toc7092482)

[Tecnologías y justificación 4](#_Toc7092483)

[Instalación 5](#_Toc7092484)

[Creación y estructura de proyecto plantilla 5](#_Toc7092485)

[NoSQL vs SQL 5](#_Toc7092486)

[Planificación 6](#_Toc7092487)

[Implementación 7](#_Toc7092488)

[Conclusiones 8](#_Toc7092489)

[Bibliografía y referencias 9](#_Toc7092490)

# Introducción y dominio

En esta memoria se analizarán el paquete de desarrollo MEAN, haciendo especial 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 hasta 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 añadir 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, posteriormente, se explicará detalladamente como se pueden instalar estas herramientas y comenzar un proyecto para ello.

1. **Ubuntu 18.04:** Sistema operativo sobre el que se ha desarrollado el proyecto, se ha elegido 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:** Máquina virtual utilizada para 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 no bloquear el código hasta que se recibe respuestas como otras tecnologías típicas tales como PHP.
4. **Express:** Módulo de Node para facilitar el uso de la creación de una API en Node, utilizando el protocolo HTTP.
5. **Angular 7:** Framework de desarrollo *frontend*, es decir, de la parte de cliente de la aplicación web, a diferencia de los dos anteriores utiliza TypeScript en lugar de JavaScript, cuya diferencia principal es que se puede añadir tipado explícito a los objetos. Lo más característico de angular es que, una vez compilado, desarrolla la web 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 utilizado como conector de MongoDB, elegido por su similitud con las *queries* nativas y su facilidad de integración con la tecnología utilizada. Aporta, además, un gran 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 indexar la base de datos más fácilmente.
9. **Git:** Utilizado como control de versiones junto a Github, en concreto: <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. **Adavanced Rest Client:** Aplicación para hacer peticiones a la API, se puede instalar como un plugin de Google Chrome lo que nos facilitó el uso e hizo que nos decantáramos por ella, en lugar de otras más conocidas como Postman.

En cuanto la justificación de la pila de trabajo MEAN (Mongo, Express, Angular y Node), se han elegido dichas tecnologías por su gran comunicación entre ellas, usando la mayoría javascript o habiendo sido desarrolladas con javascript. Todas usan el formato json para el traspaso 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 dar largas esperas al usuario y que pueda llegar a frustrarse.

## Instalación

Puesto que el proyecto ha sido desarrollado en Ubuntu, el tutorial de instalación de las herramientas necesarias para la ejecución será sobre dicho sistema operativo.

Primero, abriremos el terminal y nos aseguraremos 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 utilizar el siguiente:

sudo su

Ahora, podemos comenzar a instalar las herramientas que serán utilizadas, empezando por Node, que requiere la ejecución de los comandos:

sudo apt install nodejs

sudo apt install npm

Npm nos servirá para instalar las librerías o dependencias del proyecto e incluso para la instalación de Angular, para el que se necesitará ejecutar:

npm install -g @angular/cli

La última de las herramientas que debemos instalar será 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 en 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 mediante 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 en nuestro proyecto, poblar nuestra base de datos, compilar angular y lanzar el servidor de node. Para ello se ejecutaran los siguientes comandos respectivamente, en el directorio del proyecto:

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 nosotros durante el proyecto. Aún así, un proyecto en MEAN no tiene una estructura predefinida y cada desarrollador suele elegir una distinta según sus preferencias y el proyecto que esté realizando.

Un proyecto en angular se divide en componentes, los cuales pueden contener más componentes dentro. Estos componentes contendrán siempre su representación en html y css además de un archivo Typescript donde añadir la lógica. 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 busca tener muchos componentes de tamaño pequeño, incluso pudiendo llegar 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. Otra ventaja de angular es su CLI, puesto que podemos crear 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. Estos servicios se encontrarán dentro del componente que lo utilice, 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. Para ello, nos situaremos en justo en el primer directorio creado cuando creamos la estrudcutra base de angular, es decir, en el directorio que se llame cómo el nombre del proyecto que elegimos. Una vez ahí, ejecutaremos el siguiente comando para inicializar el proyecto en node:

npm init --yes

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

* **Server.js**: Archivo donde se encontrará las conexiones con la base de datos, configuración de la api, ruta al proyecto de angular compilado para usar las vistas, puerto… Es decir, todo lo necesario para lanzar el servidor.
* **Populate.js:** Archivo js para ejecutar con mongo, (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 no logeado.
* **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

# Planificación y costes

# Implementación

En esta sección se hablará de cómo se han diseñado e implementado las diferentes partes del proyecto, adentrándonos 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 Moongose 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 fue rápidamente descartada pues 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**: Nombre de 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 típica de un dni, para demostrar que se pueden hacer 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, para ello, hemos decidido el usar un formato de tabla para describir como funciona nuestra API Rest.

|  |  |  |
| --- | --- | --- |
| **Método** | **URI** | **Descripción** |
| GET | /api/login | Se le pasa en el header una Authorization basic y devuelve un token que se guardará en las  cookies del navegador |
| GET | /api/logout | Destruye la cookie anterior. |
| GET | /api/getPrincipal | Devuelve el usuario logeado. |
| GET | /api/user | Devuelve todos los usuarios del sistema |
| POST | /api/user | Recibe en el body 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 put requerirá la id también en el body puesto que está actualizando el usuario. |
| DELETE | /api/user | Borra el usuario |
| GET | /api/user/library | Devuelve un json con un array de tus librerias de música, videojuegos y películas. Se podría filtrar a una única librería añadiendo el parámetro ?filter=music; 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:   * 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 body 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 body el id del objeto a editar |
| DELETE | /api/user/library/:id | Elimina el objeto con id :id de tus librerias. |

## Aplicación web

# Conclusiones

# 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/>