Descripcion del backend

Tal como se describe en el github del ejercicio el back-end tiene que estar hecho con:

1. Nodejs
2. Expressjs

> Middlewares? En principio creo que no habra que usar muchos ya que esta rest api no requiere de autenticacion, habra que añadir los de siempre, el body-parser, cors? y poco mas

3. MongoDB
4. Typescript?

> *Entregar una solución que se pueda escalar o añadir funcionalidad con facilidad*
> Como pone esto en el github yo creo que usar typescript esta bien ya que no cuesta mucho meterlo y da mas seguridad, por lo menos a mi, para tener menos bugs/fallos

5. Typeorm?

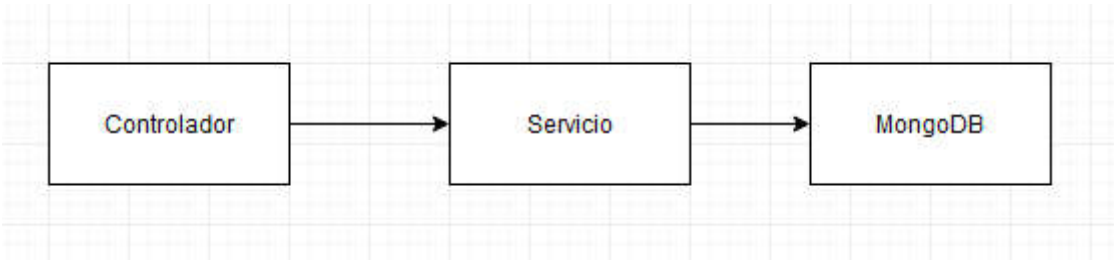
> Al ser tan pequeña la base de datos yo creo que no hace falta, pero a la hora de ampliar siempre esta bien usar un ORM

6. Añadir swagger?

> Para los pocos endpoints que hay no creo que haga falta pero para un proyecto “gordo” si que lo añadiria.

7. Crear y adjuntar el las peticiones del postman.

Arquitectura



Sin mas, un controlador por donde entra todo.
Un servicio que se conecta a mongoDB y se encarga de hacer las querys.
Las respuestas en forma de json siempre.

Endpoints que se tienen que crear

Descripcion	Endpoint	Tipo
Listado de noticias	noticias	get
Detalle de noticia	noticias/:id	get
Eliminación de noticia	noticias/:id	delete
Edicion de noticia	noticias/:id	put
Creacion de noticia	noticias	post

Modelos

titulo	tipo
title	string
body	string
image	?

titulo	tipo
url	string
publisher	enum?

La imagen puedo hacerlo de vairas formas
Subir la imagen al back.
O subir el link a la imagen (pero esto no pinta bien porque como cambien la iamgen de sitio la perderia)

Notas

Como funciona el sistema de rutas del back end:
Cuando se crea enpoints (en este caso *noticias*) la forma de hacerlo es la siguiente:

1. Crear la carpeta que ca a contener estas nuevas rutas
2. Crear un controller y un service
3. importar esto en el controller

```
import express from 'express';
const routerNoticias = express.Router();
```

4. Crear la nueva ruta en el noticias.controller.ts

```
export default routerNoticias.get('/', async (req, res) => {
    res.send('hola');
});
```

5. Importarlo en el index.ts y añadirlo

```
import routerNoticias from './noticias/noticias.controller'
app.use(`/${versionApi}/noticias`, routerNoticias);
```

Añadido validaciones de body
La libreria [express-validator] permite validar el contenido que viene en el body.

Lo que hay que crear es un esquema con `checkSchema` y despues implementar una logica que siempre sea la misma (para devolver mensajes de error).

> Esto solo se aplica a peticiones con body?
> Que pasa con peticiones con queryParams?

```
app.put('/user/:id/password', checkSchema({
  id: {
    // The location of the field, can be one or more of body, cookies, h
    // If omitted, all request locations will be checked
    in: ['params', 'query'],
    errorMessage: 'ID is wrong',
    isInt: true,
    // Sanitizers can go here as well
    toInt: true
  },
```

> Ya esta todo claro, en la parte de in, le decimos que es lo que tiene que validar y donde esta.
> Revisar la documentacion ya que aqui explica como hacer validaciones *custom*

La documentacion de la libreria es bastante pobre, pero he encontrado rebuscando por el codigo el listado de validaciones que tiene la libreria aqui

- Validacion de que no este vacio

Actualmente estoy haciendolo asi

```
        isLength: {
                errorMessage: '... no puede estar vacio',
                options: {min: 1}
        }
```

En el codigo fuente hay un metodo que se llama `notEmpty` y `isEmpty`

> Probar a ver si es mejor?
> Se puede añadir un mensaje personalizado tal como lo hace actualmente?

> Sabiendo que mongoose tiene una forma de validar, habria sido mejor usar esto que la libreria express-validator?

La forma de devolver datos

Por ejemplo estoy haciendo el metodo de guardar datos en la bbdd (save).

```
const resp: NoticiaInterface = await nuevaNoticia.save().then(
        (resp) =>{
                return resp.toObject({getters: true})
        },
        error => {
                ...
        })
```

Mongoose devuelve el objeto insertado en formato `Document`, y lo que hay que hacer es transformar este objeto a uno que tenga la siguiente interfaz: `NoticiaInterface`.

Mongoose tiene una forma de transformarlo gracias a `toObject`.

> Docs oficiales.
> Post de stackoverflow.

Excepciones

La forma de controlar las excepciones es la siguiente:

Desde el controlador hacer un `try/catch`.

En el servicio lanzar un `throw` si ocurre algun error o lanzamos nosotros manualmente.

La interfaz del objeto que se lanza con el throw es el siguiente:

```
{
    msg: string,
    error: object,
    codigo: number
}
```

Codigos de error por si alguna vez hay que revisarlo (wiki)