UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Diseño e implementación de una arquitectura full stack con software gratuito

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Toni Muñoz Ferrer

Tutor: José Salvador Oliver Gil

2018/2019

Resumen

En este trabajo se va a diseñar e implementar una arquitectura *full stack* utilizando tecnologías gratuitas basadas en JavaScript. Se usará NodeJS y Express para crear un *back-end* rápido, robusto y seguro; Vue será el *framework* que dé vida al sitio web, proporcionando facilidades para el diseño de una página web dinámica; y, por último, se utilizará MongoDB, una base de datos no relacional.

El back-end proporcionará *endpoints* para el registro y el acceso de un usuario, así como una API CRUD para el manejo de notas (crear, recuperar, actualizar y eliminar); y será el *front-end* el que haga uso de estos puntos de acceso para darle al usuario un servicio de notas *online*.

Tanto en el servidor, como en el sitio web, nos centraremos en la seguridad. Entre otras cosas: el servidor creará y verificará *tokens* de identificación (*JSONWebToken*) para cada usuario, y el sitio web los usará para saber quien está identificado en cada momento; todas las contraseñas de los usuarios se guardarán en formato hash usando la función *bcrypt*; y, por último, mencionar que todos los datos que reciba el servidor se validarán para evitar usos maliciosos.

Palabras clave: MEVN, Mongo, Express, Vue, NodeJS, API, CRUD, Docker

Abstract

In this proyect we will design and implement a full stack architecture using free JavaScript-based technologies. NodeJS and Express will be used to create a fast, robust and secure backend; Vue will be the framework that gives life to the website, providing facilities for the design of a dynamic web page; and, finally, MongoDB, a non-relational database, will be used.

The back-end will provide endpoints for the signup and signin of a user, as well as a CRUD API for managing notes (create, retrieve, update and delete); and it will be the front-end that makes use of these endpoints to give the user an online notes service.

Both on the server, and on the website, we will focus on security. Among other things: the server will create and verify identification tokens (*JSONWebToken*) for each user, and the website will use them to know who is identified at each moment; all user passwords will be saved in hash form using the bcrypt function; and finally, mention that all the data received by the server will be validated to avoid malicious uses.

Keywords : MEVN, Mongo, Express, Vue, NodeJS, API, CRUD, Docker

Índice de contenidos

1. Introducción

Este proyecto consiste en el diseño de toda una infraestructura web. Se utilizarán tecnologías basadas en JavaScript durante todo el desarrollo. Ahí entra en juego el conjunto de tecnologías MEVN, que hace uso de MongoDB, Express y NodeJS para el lado del servidor; y Vue para el lado del cliente.

Aunque es un proyecto que abarca mucho contenido, intentaremos centrarlo en la seguridad. Se crearán dos puntos de acceso que compondrán la autenticación del trabajo, uno para registrar usuarios y otro para el acceso de estos. Aparte de estos dos endpoints, la API ofrecerá diferentes rutas para proporcionar un servicio CRUD de notas online. Cada usuario registrado podrá crear, recuperar, actualizar y eliminar sus propias notas; y aquí es donde entra la autorización, a cada usuario que haya accedido al sitio web, se lo proporcionará un token con su identificación, y cada petición que realice este usuario al servidor deberá incluir dicho token; de esta forma, cada usuario tiene sus propias notas y solo puede leer y modificar las suyas. Aparte de todo esto, se creará un usuario administrador con la capacidad de acceder a todos los usuarios de la base de datos.

Utilizaremos JSON Web Tokens para crear y validar los tokens de identificación. Y como estamos guardando información de usuarios, usaremos una función de *hashing* de contraseñas llamada bcrypt, basada en el cifrado Blowfish.

1.1 Motivación

Las tecnologías web han ido sustituyendo poco a poco a todas las aplicaciones de escritorio. Todo usuario con un navegador y acceso a internet tiene a su disposición cualquier servicio que desee. Además, el usuario no se tiene que preocupar de si su sistema es compatible, o si tiene que actualizarlo.

El interés en un trabajo de este tipo es conocer más en profundidad las tecnologías que se están usando actualmente; pero, sobre todo, conocer los problemas de seguridad más habituales en este tipo de desarrollo. Estudiar y entender los ataques más comunes a servicios de este tipo y buscarles una buena solución.

Los motivos por los que se ha escogido las tecnologías MEVN son los siguientes:

- Capacidad para separar cliente y servidor, permitiendo un desarrollo totalmente independiente, y con la capacidad de actualizar solo uno de los dos componentes.
- Gran escalabilidad.
- Mismo lenguaje de programación (JavaScript) en el servidor y en el cliente.

1.2 Objetivos

El objetivo del trabajo es crear un servicio capaz de ofrecer autenticación y autorización, y una API CRUD[1] para guardar de forma persistentes notas online.

Aunque el objetivo principal es prácticamente didáctico; conocer, estudiar y comprender las tecnologías que se usan actualmente para crear este tipo de servicios, y detectar cuales pueden ser los puntos vulnerables en su seguridad.

1.3 Definiciones, siglas y abreviaciones

API: Interfaz de programación de aplicaciones (Application Programming Interface). Conjunto de rutas que proporciona acceso a determinadas funciones de un sistema.

CRUD: Crear, leer, actualizar y eliminar (*Create, Read, Update and Delete*). Hace referencia a las acciones básicas de la capa de persistencia.

MEVN: Es un conjunto de tecnologías para desarrollar aplicaciones full stack. Incluye: MongoDB, Express, VueJS y NodeJS.

NoSQL: Es una alternativa a los sistemas de gestión de bases de datos tradicionales. No usan el lenguaje de SQL para realizar consultas y los datos almacenados no requieren de una estructura fija.

JSON: *JavaScript Object Notation*. Formato de texto sencillo para el intercambio de datos.

DB: Base de datos (*Data Base*).

Endpoint/punto de acceso: Nodo de una comunicación en red. En este caso los puntos de acceso de este trabajo son las diferentes rutas ofrecidas por el servidor.

Token: Es una cadena de caracteres que contiene información, cifrada por el servidor, de un usuario en concreto.

Secreto: Es la cadena de caracteres usada por el servidor para cifrar y validar los tokens.

Front-end/sitio web/página web: También denominada “capa de presentación”, es la parte del proyecto creada con Vue y la que se le ofrece al usuario.

Back-end/servidor/API: Parte del proyecto desarrollada con Node y Express que se encarga de procesar las peticiones del front-end e interactúa con la base de datos, de ahí que también reciba el nombre de “capa de acceso a datos”.

Barra de navegación/navbar: Menú en la parte superior del sitio web accesible desde todas las páginas.

Pie de página/footer: Información al final del sitio web, en la parte inferior, que, al igual que la barra de navegación, se mantiene en todas las páginas.

[1] Servicio que proporciona las funciones básicas (crear, leer, actualizar y borrar) de una base de datos a través de varias rutas.

1.4 Estructura del trabajo

El trabajo se ha dividido en tres grandes bloques:

Bloque 1: Diseño

Empezaremos hablando del diseño del proyecto. Veremos las partes que lo componen, junto con las tecnologías usadas y como funcionan todas juntas.

Bloque 2: Implementación

En este bloque veremos más en profundidad las herramientas que se han usado. Se describirán detalladamente todas las decisiones que se han ido tomando en el desarrollo, tanto del front-end, como del back-end.

Bloque 3: Despliegue

Por último, se mostrará como se han desplegado todos los componentes del proyecto para su funcionamiento en producción.

Además de estos tres bloques, se finalizará con las conclusiones. Realizaremos un breve análisis de lo que se ha conseguido en el proyecto y hablaremos de posibles trabajos futuros.

2. Diseño

En este bloque vamos a ver las decisiones de diseño que se han tomado para el trabajo. Empezaremos viendo las tecnologías que hemos usado, y acabaremos describiendo en detalle toda la arquitectura del proyecto con sus diferentes componentes y como se relacionan.

2.1 Tecnologías

Para el desarrollo del proyecto se ha escogido el stack MEVN, el cual se describirá brevemente a continuación.

2.1.1 MEVN

MongoDB

Es una base de datos NoSQL de carácter general, basada en documentos para el desarrollo de aplicaciones modernas. Utiliza estructuras de datos BSON (una especificación similar a JSON) con un esquema dinámico, lo que facilita su acceso desde las aplicaciones.

Hay gran cantidad de alternativas a MongoDB, como puede ser: PostgreSQL, MariaDB, MySQL (si miramos alternativas basadas en SQL). Se ha decidido usar Mongo porque suponía un reto mayor. A lo largo de la carrera estudiamos SQL y bases de datos relacionales, viendo solo de forma teórica otros tipos de bases de datos.

MongoDB junto con un paquete de NPM llamado Mongoose, nos proporcionan un acceso fácil y sencillo para guardar, obtener, actualizar y eliminar datos.

Express

Express es el estándar en todas las aplicaciones servidor creadas en Node. Es un framework que gestiona las peticiones de entrada y las encamina. Aunque últimamente han empezado a utilizarse otros frameworks (Koa, Hapi, FeatherJS, etc.), Express es el más utilizado actualmente y el que forma parte del conjunto de aplicaciones MEVN, igual que Mongo.

VueJS

VueJS es un framework basado en JavaScript para desarrollar sitios web dinámicos. A pesar de contar con otras opciones como Angular o React, al final se ha elegido Vue por su simplicidad y fácil aprendizaje, además de ser puramente JavaScript.

NodeJS

NodeJS es un entorno de desarrollo del lado del servidor basado en JavaScript que usa el motor V8 de Google, asíncrono y gran uso de I/O de datos basado en eventos; creado para ser útil en programas altamente escalables.

Se ha escogido NodeJS como entorno en el lado del servidor principalmente porque nos permite usar JavaScript para el desarrollo del back-end. Por otro lado, cuenta con un gestor de paquetes (NPM) con una enorme comunidad de desarrolladores que le da soporte, esto nos permite encontrar fácilmente paquetes que resuelven problemas sencillos.

2.1.2 Otras tecnologías usadas

NPM

Como hemos mencionado previamente, NPM es un gestor de paquete muy integrado en el entorno de Node, con una gran cantidad de desarrolladores que lo soportan. Se dudó en usar una alternativa, Yarn; pero por la facilidad, tanto a la hora de la instalación, como el uso, se decidió usar NPM.

Docker

Docker es una herramienta de código abierto que automatiza el despliegue de aplicaciones en contenedores virtuales. Gracias a Docker se puede ejecutar y utilizar software sin tener que instalarlo directamente en nuestra máquina.

En este proyecto se ha usado únicamente en la fase de desarrollo, facilitando una base de datos Mongo local a la máquina donde se ha trabajo.

Git

Git es un software de versión de controles, y junto con GitHub, es lo que se ha usado a lo largo del desarrollo del proyecto para mantener una organización adecuada.

Now

Now es una plataforma de despliegue de aplicaciones gratuita. A través de NPM se puede instalar la herramienta Now, que nos permite de forma muy sencilla desplegar por separado el back-end y el front-end.

2.2 Arquitectura

Respecto a la arquitectura, se ha decidido separar el back-end (la parte del servidor) del front-end (el sitio web). Manteniéndolos de forma independiente ganamos comodidad a la hora de desarrollar cada componente y flexibilidad si se desea cambiar alguna tecnología.

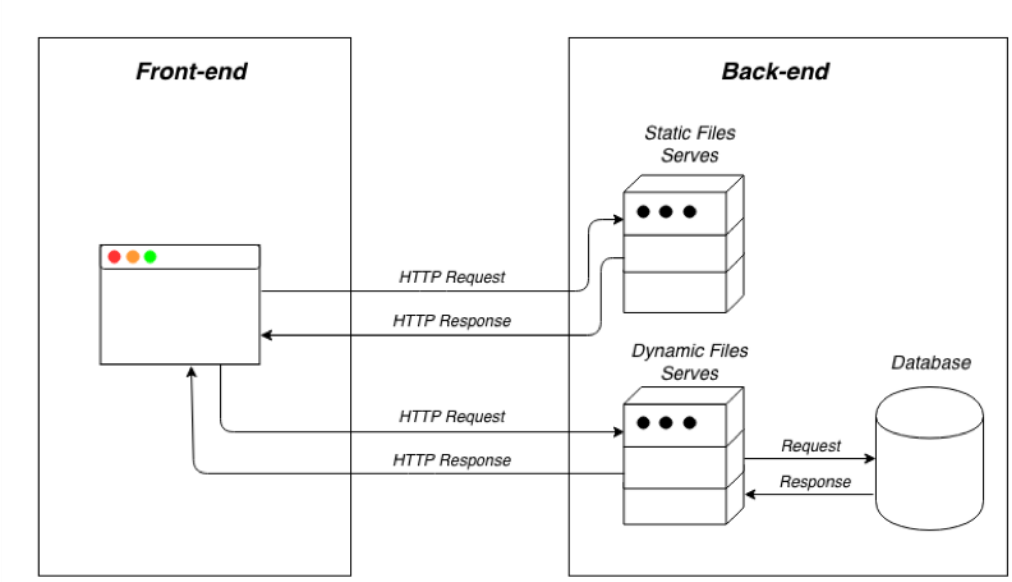

Figura 2.1: Flujo peticiones/respuestas.

Back-end

La API se ha desarrollado en el entorno de ejecución de Node; usando Express como framework para manejar las peticiones (procesamiento de las peticiones, aplicar los middlewares necesarios, etc.) a los diferentes endpoints, y el paquete de NPM: Mongoose, para crear la conexión con la base de datos y disponer de una interfaz para tratar con esta.

Todas las rutas siguen el mismo patrón de diseño; tienen un enrutador que define la ruta a la que responde (teniendo también en cuenta el método HTTP que se usa), un middleware de validación para aquellas rutas que envíen datos (registro, acceso, creación de nota...) y un controlador que procesa la petición.

Front-end

Toda la parte del cliente se realiza con Vue, y hay que diferenciar entre vistas y componentes; las vistas son las diferentes ventanas de la página web (página principal, la sección de notas, la lista de usuarios...), en cambio, los componentes son pequeños trozos de código que se repiten en las diferentes vistas (barra de navegación, *footer*...).

Cada vista tiene asociada un *template* y un *script*. En el template se añade el código en HTML que le da forma a esa vista, pudiendo añadir sintaxis de Vue para las interacciones con el controlador. El script es el controlador, es quien maneja las distintas acciones (pulsación de botones, peticiones a la API...).

Por último, mencionar la gestión de las rutas con Vue. Se ha utilizado una herramienta que proporciona Vue llamada Vue Router. Con esta herramienta, podemos definir diferentes rutas y asignarles vistas diferentes.

3. Implementación

3.1 Back-end

El back-end está creado con Node y Express, por lo tanto, una vez lanzada la aplicación, esta escuchará a todas las peticiones que le lleguen, y si hay alguna ruta establecida la resolverá, y sino devolverá un error.

Express funciona mediante el uso de middlewares, toda petición que llega va pasando por una serie de capas hasta que se resuelve; todas estas capas modifican la petición o imprimen en la consola (si estamos en la fase de desarrollo), etc. En Express, el orden de los middlewares sí importa, es decir, todas las peticiones se procesan de arriba a abajo (pasando por los middlewares que están más arriba primero).

Lo primero que se hizo en el back-end, fue añadir algunos middlewares básicos que nos permite recibir los datos del cliente de forma "limpia". Algunos de estos middlewares, vienen incluidos en Express y son muy sencillos de usar. El más importante es el que *parsea* todos los JSON de la petición (*express.json*) para que nos llegue de forma legible y no tengamos que extraer y ensamblar de forma manual todos los datos de la petición.

Por otra parte, también quitamos la cabecera *X-Powered-By*, la cual indica que tecnología se ha usado para implementar el servidor (Express en nuestro caso), y añadimos la cabecera *X-Content-Type-Options: nosniff*, la cual evita que el navegador intente adivinar el tipo de los ficheros que le llegan. Aunque pueden parecer cambios muy pequeños, todo ayuda a mejorar la seguridad del sistema, evitando que el posible atacante obtenga de forma sencilla información que le puede ser útil.

3.1.1 MongoDB

Conexión

La conexión con MongoDB se hará usando el paquete *Mongoose*. Creamos una función que realice la conexión y la exportamos, para que pueda usarse desde nuestra aplicación:

```
const mongoose = require('mongoose')

const connect = () => {
  mongoose.connect(global.config.db.mongo_url, { useNewUrlParser: true, useFindAndModify: false })
    .then(
      () => console.log('\x1b[33mConnected to database\x1b[0m\n'),
      (err) => console.log(`\x1b[33m${err}\x1b[0m`)
    )
}

module.exports = { connect }
```

La variable global.config.db.mongo_url se encuentra en un fichero de configuración y cambia en función de si nos encontramos en la fase de desarrollo, donde tendría el valor de mongodb://localhost:27017/tfg-db, o en la fase de producción, donde el string de conexión nos lo proporciona MongoDB Atlas.

Una vez exportada la función, solo tenemos que importar el archivo donde se encuentra y usarla:

```
const db = require('./db')
...
db.connect()
```

Modelos de datos

En nuestra aplicación vamos a necesitar dos tipos de datos: los usuarios, y las notas. Para poder manipular este tipo de estructuras en nuestra base datos utilizando Mongoose, debemos definir cada uno de estos modelos, que, al usarlos, se convierten en colecciones de ese tipo de documento en Mongo. A continuación, vemos como se ha definido cada modelo:

```
module.exports = mongoose.model('User', new mongoose.schema({
        username: { type: String, required: true },
        email: { type: String, required: true },
        password: { type: String, required: true },
        admin: { type: Boolean, default: false },
        create_at: { type: Date, default: Date.now }
}))
```

```
module.exports = mongoose.model('Note, new mongoose.schema({
        title: { type: String, required: true },
        body: { type: String, required: true },
        user_id: { type: String, required: true },
        create_at: { type: Date, default: Date.now }
}))
```

3.1.2 Autenticación

Para la autentificación de la aplicación se han creado dos rutas en la API, una para el registro de usuarios, y otra para el acceso en la aplicación.

Ruta	Límite de conexiones
/auth/signup	No
/auth/signin	Sí

```
const router = require('express').Router()
const bouncer = require('express-bouncer')(1500, 60000)

const controllers = require('./auth.controllers')
const validations = require('./auth.validations')

bouncer.blocked = (req, res, next, remaining) => {
        res.status(429)
        next(new Error(`Too many request have been made, wait ${remaining / 1000} seconds`))
}
```

```
router.post('/auth/signup', validations.signup, controllers.signup)
router.post('/auth/signin', bouncer.block, validations.signin, controllers.signin)
```

Ambas rutas usan el método POST, por lo que reciben datos del usuario. Para controlar que no lleguen datos maliciosos, se validan en ambas rutas los datos proporcionados por el usuario antes de pasar al middleware del controlador.

La validación se realiza con un paquete llamado *express-validator*. Este paquete nos permite definir una estructura que valida los campos del cuerpo de una petición, para luego usarla a modo de middleware, antes de que se procese dicha petición.

En este caso solo tenemos que validar tres campos: nombre de usuario, email y contraseña. Los tres campos deben existir, no pueden estar vacíos, y deben ser de tipo String. Además, el nombre de usuario solo puede contener caracteres alfanuméricos y de tener entre 4 y 30 caracteres; el email tiene que tener el formato de un email convencional (pepe@gmail.com); y, por último, la contraseña tiene que tener entre 10 y 128 caracteres, y contener al menos un dígito, una letra minúscula, una mayúscula y no tener espacios en blanco.

En ambas rutas se puede producir un fallo en la validación, lo que provocaría que se devolviese al usuario dicho fallo; en cambio, si la validación es correcta, se procede a resolver cada petición.

En la autentificación, se hace uso de una función hash para tratar con las contraseñas de los usuarios. En el registro se usa para guardar las contraseñas de los nuevos usuarios en forma de hash y no en texto plano, lo que llevaría a una filtración de cuentas de usuarios, si la base de datos es hackeada. Por otro lado, a la hora de acceder, se usará para comprobar si el hash guardado en nuestra base de datos coincide con la contraseña en texto plano proporcionado por el usuario.

Para este proyecto hemos decidido usar la función hash *bcrypt*, como se ha mencionado antes, está basada en la función Blowfish. La usaremos a través del paquete de NPM llamado *bcrypt*. Este paquete nos ofrece dos funciones muy útiles: *bcrypt.hash*, que nos crea el hash de un String; y *bcrypt.compare*, que compara un String con un hash y comprueba si ese hash procede del String.

Por último, hay que mencionar que los tokens se crean usando JSON Web Token. Esta tecnología nos permite "enmascarar" los datos de un usuario a plena vista. Proporcionando un *payload* (en formato JSON) y un secreto, se crea un token, del que más adelante se puede obtener el payload de nuevo utilizando el mismo secreto. En nuestra aplicación lo utilizaremos gracias al paquete *jsonwebtoken* de NPM.

Registro

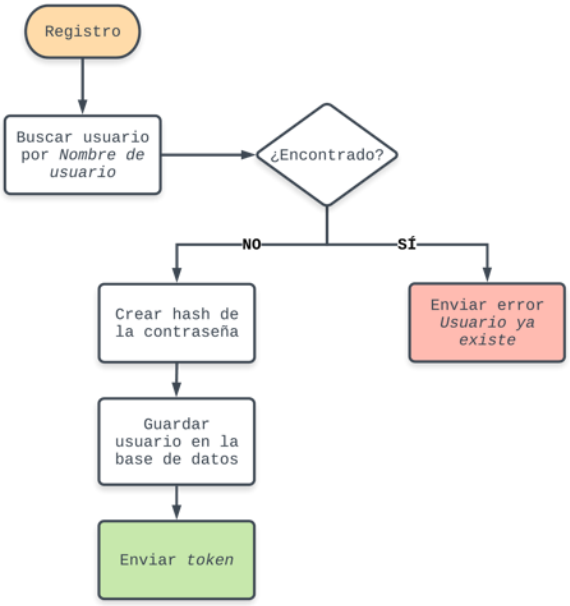

Figura 3.1: Flujo de registro.

Para el registro, del usuario se recibirá: el nombre de usuario, el email y la contraseña. Una vez la validación es correcta, se procede a buscar al usuario en la base de datos, si el usuario ya existe se envía un error, en cambio, si no existe, se procede a crear el hash de la contraseña, guardar dicho usuario en nuestra base de datos y crear y enviar el token de identificación con la información del usuario.

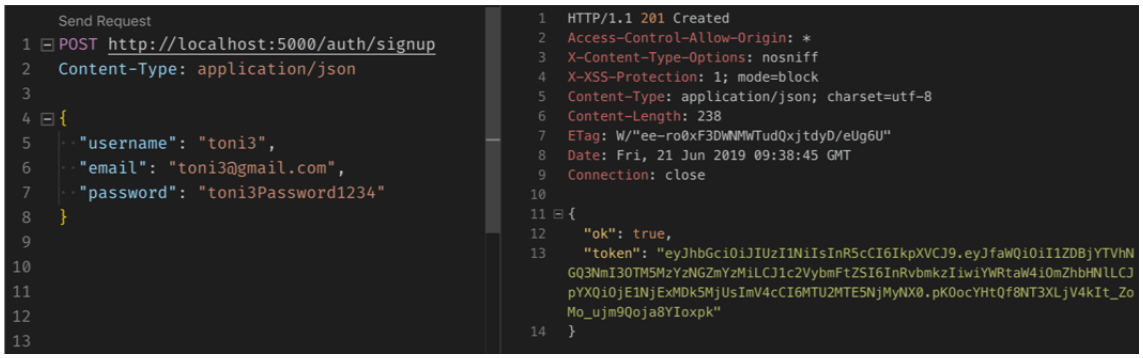

Figura 3.2: Ejemplo petición/respuesta de registro.

Acceso

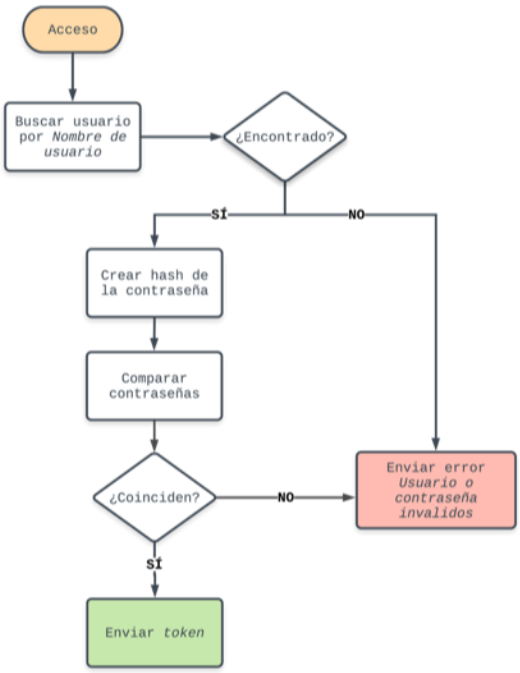

Figura 3.3: Flujo de acceso.

En el acceso, igual que en el registro, lo primero es buscar al usuario en la base de datos. Si el usuario se encuentra en la base de datos, se procede a comparar las dos contraseñas, para ello se hace el hash de la contraseña enviada por el usuario y se comparan los hashes de las dos contraseñas, si coinciden se crean el token de identificación y se le envía al usuario.

```
Send Request
POST http://localhost:5000/auth/signin
Content-Type: application/json

{
  "username": "toni3",
  "password": "toni3Password1234"
}
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Content-Type: application/json; charset=utf-8
Content-Length: 238
ETag: W/"ee-GUdC7FiC9wJ17FR8mgaPvJ+FAEo"
Date: Tue, 25 Jun 2019 10:51:36 GMT
Connection: close

{
  "ok": true,
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI1ZDExZmNiNDM0MWUzMTQ2NzBjODNkYjciLCJ1c2VybmFtZSI6InRvbmkzIiwiYWRtaW4iOmZhbHNlLCJpYXQiOjE1NjE0NTk4OTYsImV4cCI6MTU2MTU0NjI5Nn0.8iU1fz82uG2Kj6JpMGxFhdtsm9o_Kx8ZrZsXlRqlDbE"
}
```

Figura 3.4: Ejemplo petición/respuesta de acceso.

En la ruta de acceso, también se ha implementado un middleware para limitar el número de conexiones; con esto evitamos que se puedan averiguar contraseñas de usuarios por fuerza bruta. Para esto se ha usado un paquete llamado *express-bouncer*, y lo único que necesita es una función para cuando se ha superado el limite de conexiones.

3.1.3 CRUD API

La aplicación, cuenta con dos tipos de rutas, las de los usuarios, y las de las notas. Los usuarios disponen de dos rutas: una para recuperar al usuario identificado, y otra para recuperar a todos los usuarios de la base de datos (solo accesible por el administrador), y las notas disponen de cuatro rutas básicas que forman la API CRUD:

Ruta	Método HTTP	Autorización
/api/user	GET	Usuario
/api/user/all	GET	Administrador
/api/note	GET	Usuario
/api/note	POST	Usuario
/api/note/:id	DELETE	Usuario
/api/note/:id	PUT	Usuario

Todas las rutas siguen el mismo patrón de desarrollo que las dos de autentificación. Las que reciben datos del usuario se validan antes de ser procesadas y cuentan con un controlador cada una que responde con los datos adecuados.

La primera ruta de los usuarios (*/api/user*) devuelve información del usuario identificado por el token, y la segunda devuelve una lista con todos los usuarios guardados en al base de datos; esta última ruta está protegida con el middleware que verifica que el usuario autentificado es el administrador.

Las rutas de las notas son las cuatro rutas básicas que definen un API CRUD: crear, recuperar, actualizar y eliminar. Cada una de estas rutas se diferencia por el método HTTP utilizado, aunque todas podrían usar el mismo método y diferenciarlas por el nombre de la ruta, se ha decidido seguir el estándar HTTP.

Todas estas rutas están protegidas por el middleware que comprueba que hay un usuario autentificado, por lo tanto, cada usuario solo tiene acceso a su información y sus notas.

3.1.4 Autorización

Como hemos mencionado antes, algunas rutas necesitan autorización para poder acceder a ellas. Esta autorización se realiza mediante el token de identificación que se le envía al usuario cada vez que se registra o accede. Cuando un usuario quiere realizar alguna acción, como crear una nota nueva, en esa petición debe enviar el token.

En la parte del servidor, para comprobar si una petición lleva autorización se ha creado un middleware. Este middleware busca en la petición la cabecera *Authorization*, si la encuentra, extrae el token, lo verifica, y si todo es correcto, añade a la petición el usuario extraído del token.

```
const verifyToken = (req, res, next) => {
        const authHeader = req.get('authorization')
        if (authHeader) {
                const token = authHeader.split(' ')[1]
                if (token) {
                        jwt.verify(token, config.auth.jwt_secret, (err, user) => {
```

```
                                        if (!err) req.user = user
                                })
                        }
                }
}
```

Una vez que ya sabemos si la petición de un usuario está autentificada o no, solo tenemos que comprobarlo antes de cada ruta que lo necesite. Para esto se han creado dos middlewares muy sencillos que comprueban si un usuario está autentificado (*isLogged*), y si un usuario es administrador (*isAdmin*), ya que hay rutas que solo son accesibles por el administrador. Si no están autentificados o intentan acceder a una ruta a la que no tienen permiso, se envía al usuario el error correspondiente:

isLogged

En este caso debemos comprobar si el middleware *verifyToken* ha añadida el objeto *user* a la petición, si es así, simplemente pasamos al siguiente middleware, sino pasamos un error de *'Unauthorized'*.

```
const isLogged = (req, res, next) => {
        if (req.user) next()
        else {
                res.status(401)
                next(new Error('Unauthorized'))
        }
}
```

isAdmin

Igual que en el middleware *isLogged*, comprobamos el objeto *user* en la petición, pero en concreto la variable *admin* y comprobamos si es *true*, si es así, se sigue al siguiente middleware, si no se envía un error, pero en este caso de *'Forbidden'*.

```
const isAdmin = (req, res, next) => {
        if (req.user.admin) next()
        else {
                res.status(403)
                next(new Error('Forbidden'))
        }
}
```

Figura 3.5: Ejemplo de petición sin un token válido.

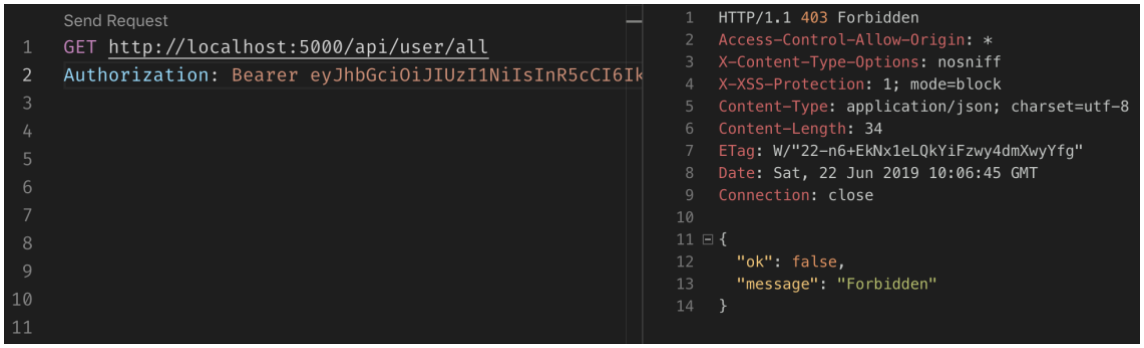

Figure 3.6: Ejemplo de petición sin autorización.

3.1.5 Manejo de errores

Para el manejo de errores, vamos a crear dos middlewares que situaremos al final de la aplicación:

```
const middleware = require('./middleware')
...
app.use(middleware.error.notFound)
app.use(middleware.error.errorHandler)
```

El primero es el error que se da cuando no se encuentra la ruta solicitada, y el segundo es el manejador general de errores. Este último se encargará de recoger cualquier error producido por la aplicación y devolver una respuesta con el error que se ha producido:

notFound

Este middleware lo único que hace es recoger todas las peticiones que han llegado al final sin ser resueltas, lo que indica que la ruta no existe en nuestra aplicación; por lo tanto, envía un error de *'Not Found'*:

```
const notFound = (req, res, next) => {
        res.status(404)
        next(new Error('Not Found'))
}
```

errorHandler

En este caso, se trata del último middleware de la aplicación, que recoge todos los errores producidos y los resuelve. Si el error que recibe tiene un código de error diferente a 200, quiere decir que es un error que hemos creado nosotros, y envía al usuario el mensaje establecido en dicho error, en cambio, si el código de error es 200, es porque se ha producido un error interno en el servidor (fallo al usar un módulo, fallo al conectarse a la base de datos...), por lo tanto cambiamos el código de error a 500 y el mensaje a: *'Internal Server Error'*:

```
const errorHandler = (req, res, next) => {
        res.statusCode !== 200

                ? res.status(res.statusCode).json({ ok: false, message: err.message })

                : res.status(500).json({ ok: false, message: 'Internal Server Error' })
```

```
}
```

3.1.6 Desarrollo vs producción

En este apartado vamos a ver las pequeñas diferencias de nuestra aplicación entre el periodo de desarrollo y el de producción; a la hora de desarrollar el back-end, nos hemos centrado en usar tecnologías sencillas.

Una de las principales diferencias es que en el desarrollo la base de datos de Mongo está en nuestra máquina local, a diferencia de en producción, que está desplegada en la nube. Para tener acceso a Mongo de forma local, hemos usado Docker, una herramienta de virtualización por contenedores que nos permite lanzar aplicaciones dentro de estos contenedores de forma muy sencilla y eficaz a nivel de recursos del PC.

Por otro lado, tenemos los *logs* del sistema del back-end. En el desarrollo todas las conexiones que se hacen a este se muestran en la terminal donde hemos lanzado en proceso:

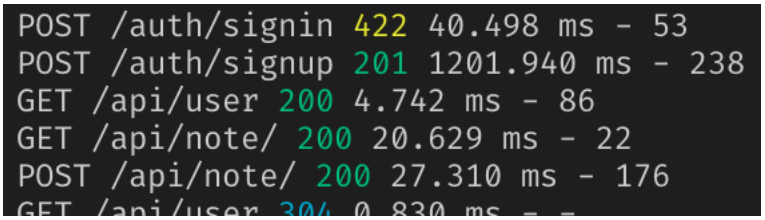

Figura 3.7: Ejemplo de registro de conexiones al back-end en la fase de desarrollo.

En cambio, en producción, el registro de todas las conexiones que se realizan a la API se guarda en unos ficheros *access.log*. Estos ficheros guardan todas las conexiones en orden y se crea un nuevo fichero cada día o cuando supera cierto límite de tamaño. Las conexiones se guardan con el siguiente formato: ***dirección - usuario* [*fecha*] "*método url* HTTP/*version-http*" *código-estado tamaño-respuesta***. Ejemplo:

```
127.0.0.1 - - [08/Jun/2019:14:29:13] "POST /auth/signin HTTP/1.1" 201 237
```

Por último, hay que mencionar el fichero de configuración. Este fichero contiene variables que varían en función de si nos encontramos en la fase de desarrollo, o hemos pasado ya a la de producción. Algunas de estas variables son: el secreto usado en la creación y verificación de los tokens, el string de conexión a la base de datos, etc. Este fichero guarda en variables el valor de esa misma variable de entorno o su alternativa en el desarrollo, por ejemplo:

```
mongo_url = process.env.MONGO_URL || 'mongodb://localhost:27017/tfg-dev'
```

Nuestra aplicación usará la variable *mongo_url* al conectarse a la base de datos, y esta tendrá el valor de *process.env.MONGO_URL* (si existe) o el string entre comillas. En la fase de desarrollo las variables de entorno no existen, pero al desplegar el back-end se crean, cambiando ciertos componentes de la aplicación, como la base de datos a la que conectarse.

3.2 Front-end

El sitio web se ha creado con Vue, cuyos archivos usan la extensión *.vue* . Cada archivo *.vue* que se crea tiene dos secciones principales: *template* y *script*. El template recoge el código HTML y el script la lógica de ese HTML.

Como hemos mencionado anteriormente, en este proyecto se ha usado Vue Router, que nos permite definir diferentes rutas y asignarle a cada una, una vista diferente. Estas vistas, se renderizan todas en el mismo lugar, dejando siempre el *navbar* y el *footer* sin actualizar.

Para conseguir que solo se renderize la parte dinámica de la página, Vue Router utiliza una etiqueta llamada *router-view*, que es la encargada de actualizarse cada vez que se cambia de ruta, manteniendo inalterables los demás elementos. Esta etiqueta se utiliza en el esqueleto de la aplicación, en el archivo *App.vue*, el cual será el que se incruste en el documento HTML.

```
<template>
        <Navbar/>
        <router-view class=”router-container” />
        <Footer/>
</template>
```

3.2.1 Rutas

La aplicación únicamente cuenta con dos rutas: la principal (‘/’) y una ruta a la que solo puede acceder el administrador (‘/users’). La ruta principal tiene dos vistas posibles dependiendo de si el usuario esta identificado en la aplicación o no.

Ruta	Vista	Autorización
/	Home.vue	-
/	Dashboard.vue	Usuario
/users	Users.vue	Administrador

Se ha decidido usar una misma ruta, pero con dos vistas diferentes, para simplificar la navegación por la página. Para diferenciar ambas vistas, se comprueba si el token de identificación esta guardado en el almacenamiento local; si se encuentra, entonces se renderiza el Dashboard, en caso contrario, se renderiza la página principal de acceso.

3.2.2 Vistas

Como se ha mencionado antes, la aplicación cuenta con tres vistas: la ventana principal, donde el usuario puede registrarse o acceder a la aplicación; la ventana donde un usuario identificado puede ver, crear, editar y eliminar sus notas (y si es administrador, también puede navegar a la ventana de usuarios); y, por último, la ventana de usuarios, donde, si el usuario identificado es administrador, se mostrarán todos los usuarios guardados en la base de datos.

Home

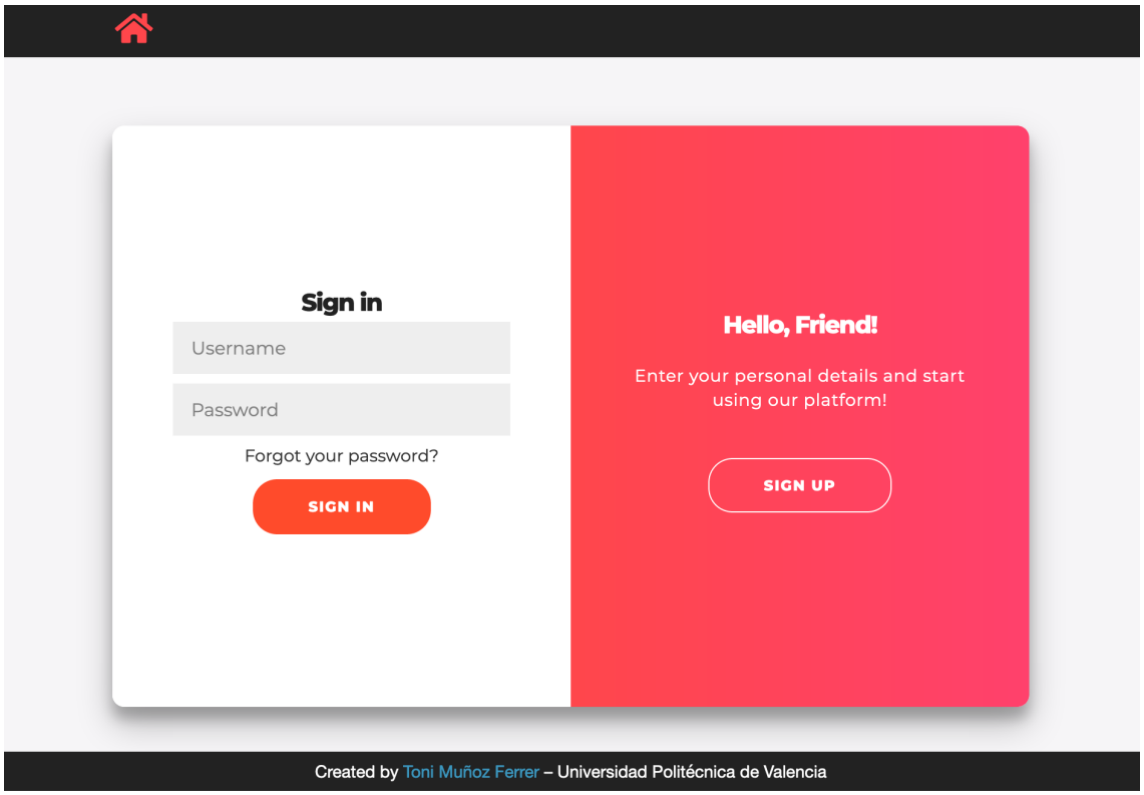

Figura 3.8: Página de inicio.

La vista home es la ventana principal, y la que se muestra cuando un usuario accede por primera vez a la página web. En ella se puede tanto acceder, si ya se posee una cuenta de usuario, como registrarse. Para ello se han creado dos formularios diferentes, los cuales se envían a las dos rutas que ofrece nuestra API para el acceso y el registro de usuarios.

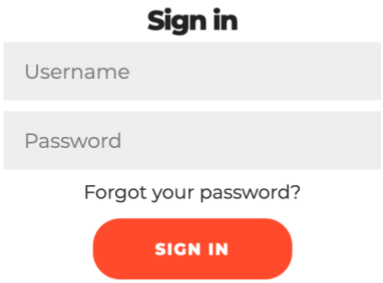

Figura 3.9: Formulario de acceso.

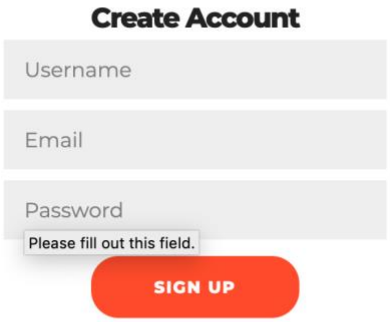

Figura 3.10: Formulario de registro.

Si la respuesta del servidor es un error, este se mostrará en la parte de arriba de la pagina.

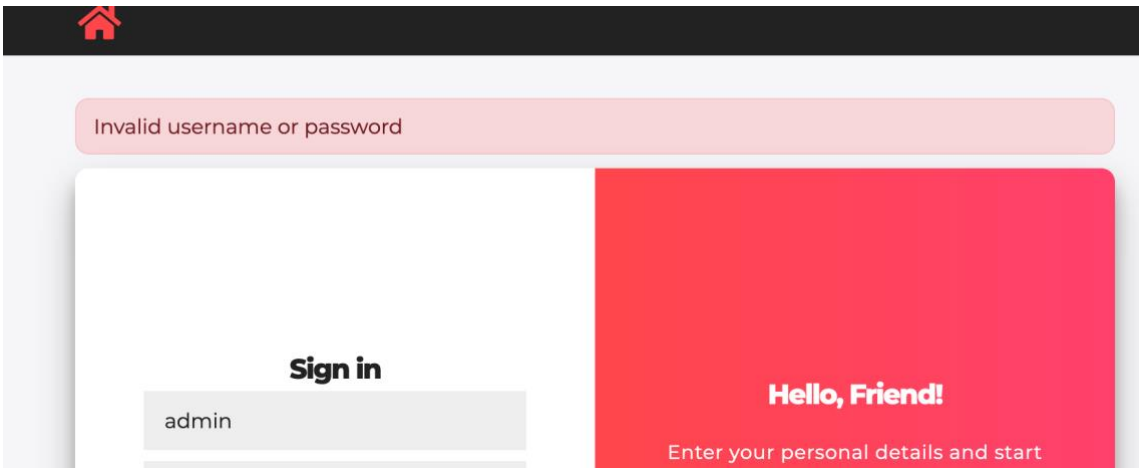

Figura 3.11: Ejemplo de respuesta de error del servidor.

En cambio, si el servidor nos responde satisfactoriamente, eso indica que hemos podido registrarnos o acceder y por lo tanto tenemos el token de identificación, el cual guardaremos en el almacén local del navegador para poder acceder a él desde cualquier parte de la aplicación.

Dashboard

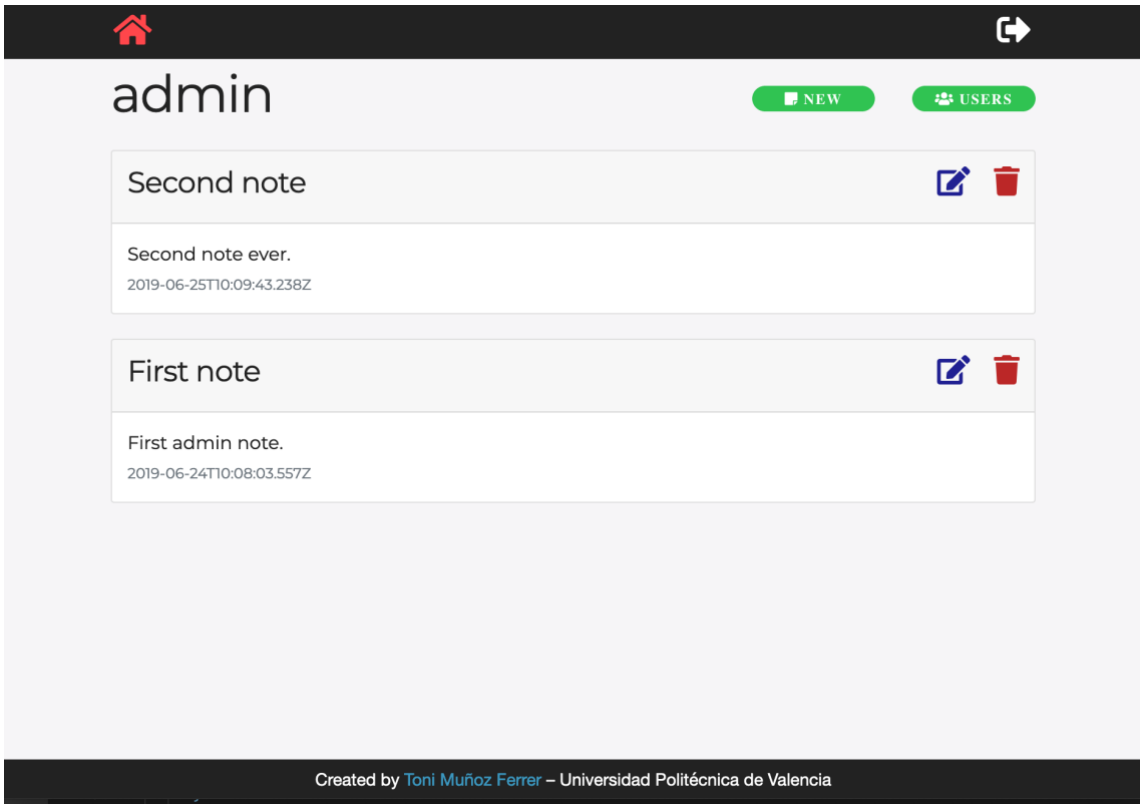

Figura 3.12: Página de inicio con usuario autentificado.

Al acceder al Dashboard, se realizan dos peticiones al servidor: una para recibir los datos del usuario autentificado y otra para obtener todas sus notas (ambas usando el token recibido al acceder al sitio web).

En el Dashboard, el usuario puede ver todas sus notas, con la opción de editarlas y eliminarlas, así como crear nuevas. Cada uno de estos botones (editar, eliminar y crear) realizan una petición a la ruta correspondiente del servidor.

Al pulsar el botón de eliminar, se envía una petición a la ruta */api/note/:id*, donde *:id* corresponde al id de la nota que queremos eliminar. El servidor recibe la petición, si ese id existe, elimina la nota de la base de datos y devuelve una respuesta de OK, y en ese momento, se actualiza la lista de notas del usuario.

En el caso de editar y crear, se abre un formulario en la parte superior de la página, para rellenar el título y el cuerpo de la nota. Si al enviar la petición todo ha ido bien, igual que al eliminar una nota, se actualiza la lista de notas.

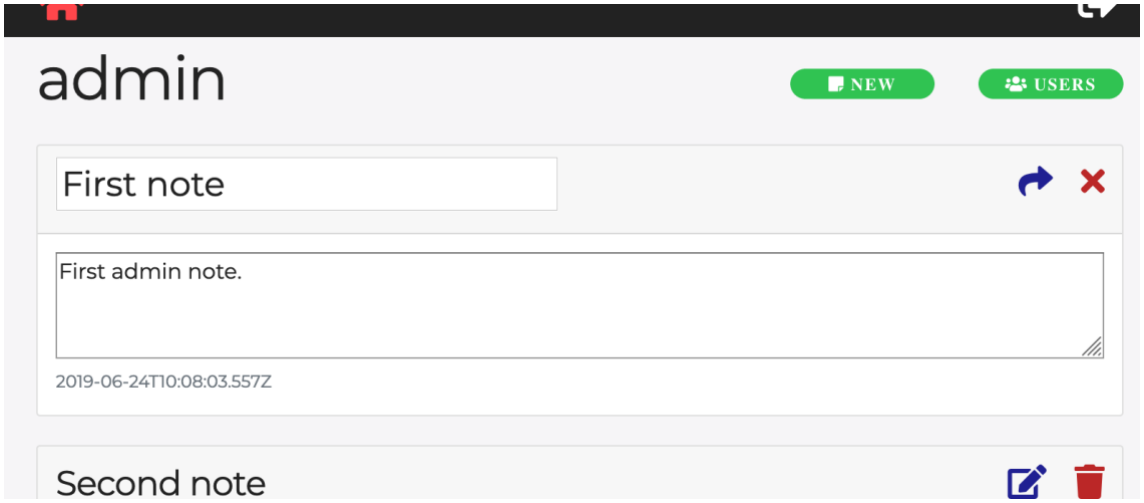

Figura 3.13: Formulario de creación y edición de una nota.

Si en cualquier momento el servidor devuelve un fallo, este se mostrará en la parte superior, igual que en la página de inicio.

Users

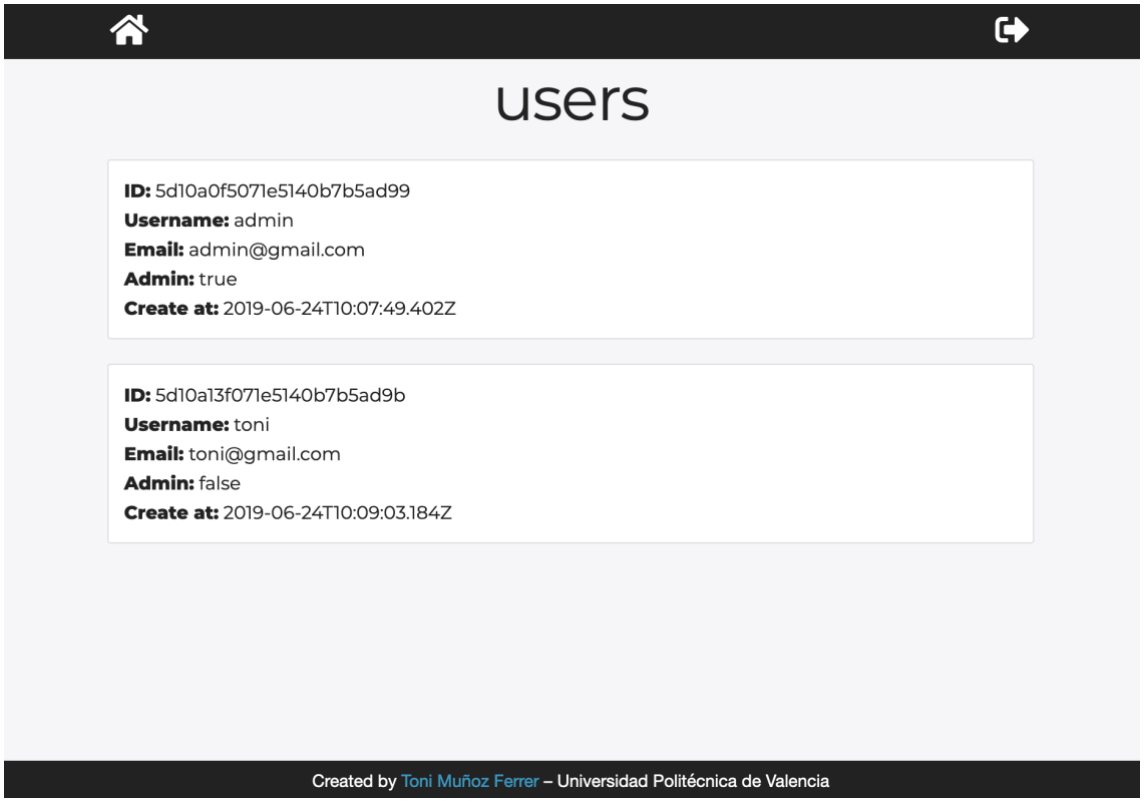

Figura 3.14: Página de usuarios registrados.

En el caso de que el usuario sea administrador, en la vista de Dashboard, se lo mostrará un botón de usuarios, que redirige a la ruta */users* donde se muestran todos los usuarios, con toda su información, guardados en la base de datos.

3.2.3 Componentes

Como hemos mencionado previamente tenemos dos componentes que se mantienen inalterables en las diferentes rutas: la barra de navegación y el pie de página. Estos dos componentes se crean por separado y se importan en el archivo *App.vue*.

```
<template>
        <Navbar/>
        <router-view class=”router-container”/>
        <Footer/>
</template>
<script>
        import Navbar from ‘./components/Navbar’
        import Footer from ‘./components/Footer’
…
</script>
```

La barra de navegación tiene dos botones (en forma de iconos): el primero es un enlace a la página principal (la ruta “/”), y el segundo es un botón para desconectarse de la página, que borra el token de almacén local y redirige a la página de acceso y registro.

3.2.4 Autenticación

La autenticación en el lado del servidor se basa completamente en el token proporcionado por el servidor cuando se accede a la aplicación web. Como hemos mencionado anteriormente, el token se guarda en el almacenamiento local del navegador, pudiendo acceder a este en cualquier momento.

Una vez el token se ha guardado, como hemos mencionado previamente, la vista del Dashboard accede a las dos rutas del servidor que le proporciona la información que necesita: los datos del usuario (id, nombre y si es administrador), y sus notas.

```
fetch(‘http://localhost:5000/api/user’, {
        method: ‘GET’,
        headers: { ‘Authorization’: `Bearer ${localstorage.token}` }
})
…
fetch(‘http://localhost:5000/api/note, {
        method: ‘GET’,
        headers: { ‘Authorization’: `Bearer ${localstorage.token}` }
})
```

Al utilizar el token como identificación, cada usuario solo tiene acceso a su información y sus notas. Siendo, la única manera posible de acceder a las de otro usuario, obteniendo su token.

3.2.5 Autorización

Como hemos mencionado antes, en el almacenamiento local tenemos una variable que nos indica si el usuario actual es administrador o no. Como la única ruta protegida es la de */users*, solo debemos realizar una comprobación antes de entrar a dicha ruta (en el

archivo *router.js*). Si el usuario es administrador se le permite continuar, si no se le redirige a la ruta base:

```
beforeEnter: (to, from, next) => {
        if (localStorage.getItem('admin') === 'true') next()
        else next('/')
}
```

3.2.6 CSS

Aunque no ha sido el foco de este proyecto, sí se ha trabajo un poco el CSS. Se han usado dos ficheros de CSS en todo el trabajo: uno de Fontawesome, para poder usar algunos de sus iconos gratuitos, y uno creado por nosotros.

El CSS creado por nosotros contiene partes importadas de Bootstrap y de Skeleton, dos grandes entidades que ofrecen estilos de CSS gratuitos. Además de estos dos se han implementado varias estructuras que nos permiten, por ejemplo, crear el panel desplazable de la página de inicio.

También cabe mencionar que haciendo uso de CSS y un *csv*, se ha creado el movimiento de carga cuando se espera respuesta del servidor.

4. Despliegue

4.1 MongoDB

Para el despliegue de la base de datos de Mongo, hemos usado MongoDB Atlas. Es una herramienta creada por los desarrolladores de MongoDB, que nos permite crear una base de datos en la nube de forma sencilla y gratuita.

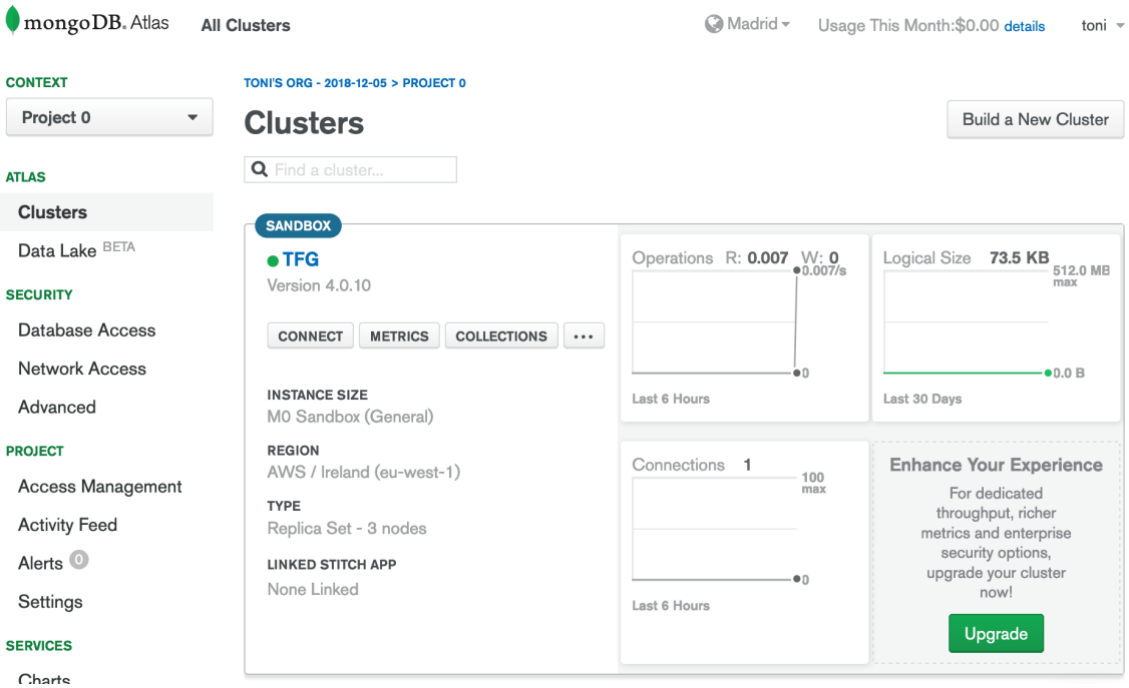

Figura 4.1: Interfaz de MongoDB Atlas.

Lo único que se necesita es una cuenta registrada y crear la BD, proporcionándole un usuario que pueda manipularla y las direcciones desde donde se puede acceder (en nuestro caso todas: 0.0.0.0). Una vez creada, tenemos acceso al string de conexión que podemos usar en nuestra aplicación.

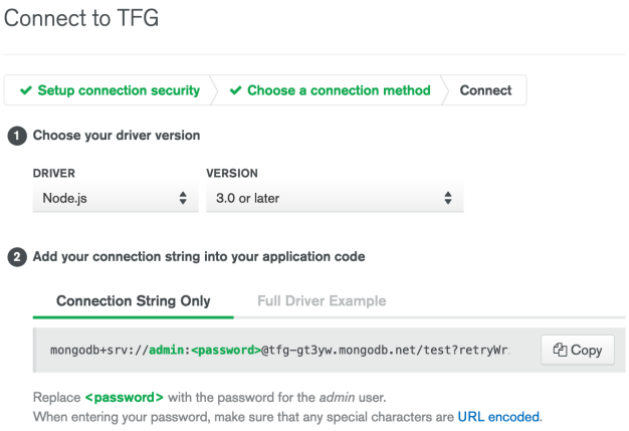

Figura 4.2: Acceso al string de conexión de nuestra DB.

4.2 Back-end

El back-end se despliega con una herramienta llamada Now, desarrollada por ZEIT. Para desplegar un proyecto en Node con esta herramienta, lo mas sencillo es descargarse su módulo a través de NPM.

Una vez descargado lo único que hay que hacer es acceder con nuestra cuenta, y crear el fichero *now.json* en la carpeta raíz del proyecto. Este fichero contendrá la configuración del despliegue como: el nombre del proyecto y las variables de entorno que se necesitan:

```
{
        “name”: “tfg-toni”,
        “version”: 1,
        “env”: {
                “NODE_ENV”: “production”,
                “MONGO_URL”: “mongodb+srv://admin:Admin1234567890@tfg-
gt3yw.mongodb.net/tfg-db?retryWrites=true&w=majority”
                “JWT_SECRET”: “nfeinfekjwf93021ruy89hfi9jqcmijovqdwqdjfe”
        }
}
```

Una vez que hemos accedido con nuestra cuenta y configurado el fichero, en la carpeta raíz del proyecto, lanzamos el comando now. Al finalizar, se nos proporciona una URL que es el acceso a nuestro back-end desplegado.

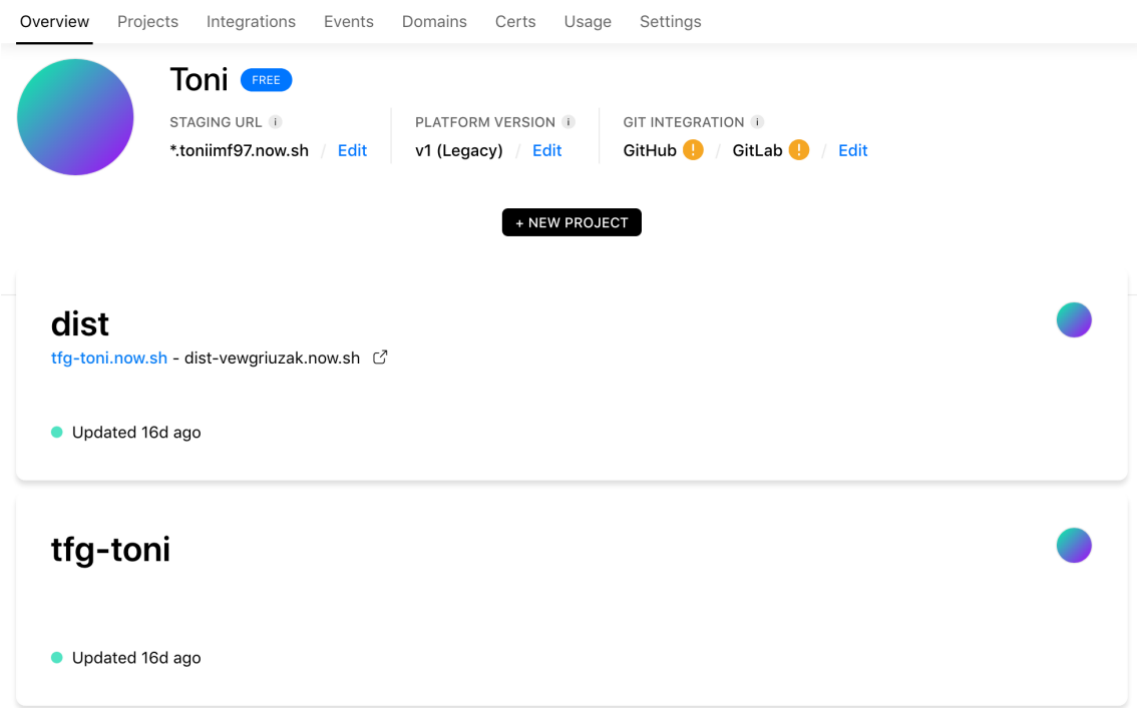

Figura 4.3: Interfaz online de NOW (lista de los proyectos desplegados).

4.3 Front-end

Igual que en el front-end, para el despliegue del sitio web, hemos utilizado Now. En este caso, al ser un proyecto web no se ha necesitado el fichero de configuración. Lo único que se ha tenido que cambiar son los rutas a nuestro servidor, en lugar de ir a la ruta local, hay que cambiarlas por la nueva ruta que se le ha proporcionado a nuestra API al ser desplegada:

```
const url = window.location.hostname === 'localhost' ? 'http://localhost:5000' : 'https://tfg-toni-rfgeltcxmo.now.sh'
```

Añadido este cambio, se construye el proyecto usando el comando `npm run build`, proporcionado por Vue, que crea la carpeta *dist* con todo el contenido de la aplicación web. Igual que en el servidor, con el comando `now` en la carpeta *dist* se despliega el front-end.

A diferencia que el servidor, nos interesa que la url de la página web sea más sencilla. Para añadirle un alias a nuestra dirección web, debemos acceder a la página web de Now y desde nuestro panel de control asignarle un alias al proyecto desplegado. En nuestro caso: tfg-toni.now.sh

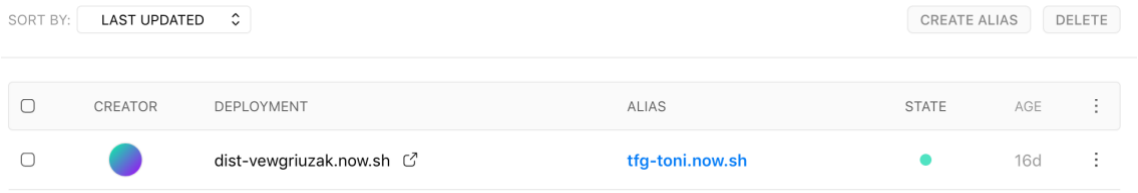

Figura 4.4: Información del proyecto web desplegado.

5. Conclusiones

Al finalizar el proyecto se han conseguido los objetivos planteados al principio de esta memoria. Se ha creado un servicio que ofrece autentificación y autorización, que distingue entre dos tipos de usuarios, además de una API capaz de crear, recuperar, actualizar y eliminar notas online.

Por otro lado, al no haber visto esta tecnología a lo largo de la carrera, se ha tenido que hacer un estudio muy exhaustivo de estas, llegando a conocerlas bastante bien. Aunque sí hemos visto un poco de JS, HTML y CSS en alguna asignatura de la carrera, no ha sido suficiente para desarrollar todo este proyecto.

Al centrar el proyecto en la seguridad, se han visto problemas típicos como el XSS o el CSRF (aunque algunos de estos no afectasen a nuestra aplicación) y que soluciones se les pueden dar. Como el servidor y la página web se encuentran separados, se ha tenido que lidiar con el problema de CORS (*Cross Origin Resource Sharing*). Se estudiaron varias formas de crear la autentificación, decidiéndonos al final por la más convencional (usuario y contraseña), lo que proporciona que cada usuario solo tenga acceso a sus datos y notas.

Siguiendo con el tema de la seguridad, se han visto varias cabeceras que añaden un poco de protección contra posibles ataques: *X-Powered-By* y *X-Content-Type-Options: nosniff*. También se han implementado varios middlewares para validar los datos que provienen del cliente en todas aquellas rutas que lo necesiten. Por último, cabe mencionar que se ha usado una función hash para no guardar las contraseñas de los usuarios en texto plano; evitando así que una filtración de la base de datos provoque la perdida de todos los usuarios registrados.

En este proyecto, no solo hemos aprendido tecnologías de desarrollo, sino también muchas formas de desplegar un proyecto. Aunque nos hemos decidido por Now, ya que nos permite desplegar el servidor y la página web de forma muy sencilla, existen muchas otras opciones igual de viables y gratuitas como Heroku.

Respecto al despliegue de la base de datos de Mongo, en un principio se iba a realizar usando mLab, una plataforma de despliegue online de bases de datos; pero al iniciar el proyecto esta se fusionó con MongoDB Atlas, de ahí que se decidiera utilizar la segunda.

5.1 Relación con los estudios cursados

Respecto a la relación se este proyecto con los estudios cursados, como se ha mencionado antes, se vio por encima JS en la parte de servidor. También estudiamos HTML y CSS, que junto con lo aprendido en la asignatura “Desarrollo Centrado en el Usuario”, se ha creado un sitio web limpio y profesional.

En relación con la seguridad, se han seguido las recomendaciones ofrecidas por las dos asignaturas de seguridad cursadas: “Seguridad Web” y “Hacking Ético”. Se han intentado solucionar todos los problemas típicos que se suelen dar en estos servicios.

Como se ha menciona en el trabajo, en la carrera se estudiaron bases de datos relacionales, en cambio en este trabajo se decidió usar una no relacional porque suponía un reto mayor. Nos hemos encontrado con que trabajar con este tipo de bases de datos es muy sencillo. A diferencia de las bases de datos relacionales, es mucho más flexible, ya que no requiere un modelo de datos concreto. Aunque el uso de SQL es bastante cómodo, no ha resultado tan complicado manipular los datos como suponia en un principio. A pesar de que las bases de datos relacionales son muy usadas actualmente, para proyectos pequeños recomendaríamos usar no relacionales por su comodidad y simplicidad.

5.2 Trabajo futuro

Al usar software gratuito, el despliegue no ofrece un sitio web muy rápido, por lo que se podría migrar el proyecto a una plataforma que ofreciese más velocidad, así como adquirir un dominio propio. Aunque la base de datos también se ha desplegado usando tecnologías gratuitas, el único problema que puede llegar a plantear es la capacidad de almacenamiento, la cual no supone un problema a corto plazo.

Respecto a la autentificación, podría ampliarse añadiendo métodos alternativos para registrarse y acceder al sitio web. Una posible ampliación es el uso de OAuth2.0. Es una tecnología que permite una autentificación segura y sencilla haciendo uso de cuentas de Gmail, Facebook, Microsoft, GitHub, etc. ya existentes.

Otra posible ampliación es aumentar la seguridad de los tokens. Actualmente cualquier persona que se haga con un token ajeno, podría hacerse pasar por dicho usuario. Se podría implemente un mecanismo para que los tokens solo sean validos desde la IP donde se obtuvieron. Además, también se podría disminuir el tiempo de vida de estos tokens, hacerlos de un solo uso, o que sean inválidos una vez se cierra el navegador.

Por último, un cambio sencillo y útil que se puede implementar es ampliar la funcionalidad del rol de administrador, permitiéndole cambiar información de los usuarios, como datos personales.

6. Bibliografía

[1] Sitio web oficial de MongoDB. Consultado en https://www.mongodb.com/es. Abril 2019.

[2] Sitio web oficial de ExpressJS. Consultado en https://expressjs.com/es/. Abril 2019.

[3] Sitio web oficial de NodeJS. Consultado en https://nodejs.org/es/. Abril 2019.

[4] Sitio web oficial de Vue. Consultado en https://vuejs.org/. Abril 2019.

[5] Express 4.x API. Consultado en http://expressjs.com/en/4x/api.html. Abril 2019.

[6] Mongoose API Reference . Consultado en https://mongoosejs.com/docs/guide.html. Abril 2019.

[7] Vue API. Consultado en https://vuejs.org/v2/api/. Mayo 2019

[8] Vue Router API References. Consultado en https://router.vuejs.org/api/. Mayo 2019.

[9] Node: Up and Running. Tom Hughes-Croucher. Editorial O'Reilly. 1º Edición.