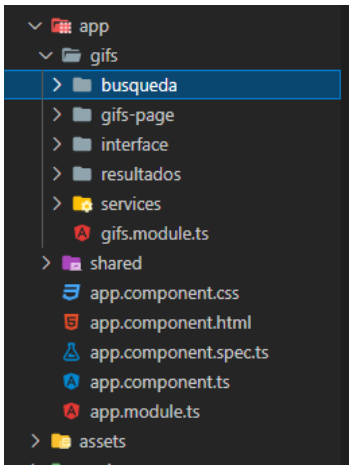
Ana Quesada García
3 DAW
IES Roque Amagro
Curso 2020/2021

El proyecto se divide entres partes importantes a la hora de crearlo:

- Frontend
- Backend
- Aplicación buscadora de gifs

Y luego se une la frontend con la aplicación con la aplicación de gifs para luego unirla al backend.

1. Aplicación buscadora de gifs



Esta es la parte con la que principalmente interactuará el usuario ya que tenemos el buscador de gifs. Está estructurado de la siguiente manera:

1. **Búsqueda:** en la cual tenemos el componente que tiene la barra buscadora de gifs.
2. **Gifs-page:** unimos los componentes para disponerlos como se mostrarán al usuario.
3. **Resultados:** aquí tenemos la parte donde se muestran los resultados de la búsqueda que ha realizado el usuario.
4. **Services:** aquí definimos ciertas funciones que hace que los componentes tengan funcionalidad.

1.1.Componente búsqueda

HTML

```
<h5>Buscar:</h5>
<input
    type="text"
    class="form-control"
    placeholder="Buscar gifs..."
    (keyup.enter)="buscar()"
    #txtBuscar
>
```

Solo tenemos un input con el evento ***keyup.entrer*** al cual le asignamos la función ***buscar*** la cual desarrollamos en el typescript

Typecript

```
export class BusquedaComponent {

  @ViewChild('txtBuscar') txtBuscar!:ElementRef<HTMLInputElement>;

  constructor( private gifsService: GifsService ) {}

  buscar() {

    const valor = this.txtBuscar.nativeElement.value;

    if ( valor.trim().length === 0 ) {
      return;
    }

    this.gifsService.buscarGifs( valor );

    this.txtBuscar.nativeElement.value = '';
  }

}
```

Aquí tenemos la función buscar la cual nos lleva al servicio buscarGifs

Service

```
buscarGifs( query: string = '' ) {

  query = query.trim().toLocaleLowerCase();

  if( !this._historial.includes( query ) ) {
    this._historial.unshift( query );
    this._historial = this._historial.splice(0,10);

    localStorage.setItem('historial', JSON.stringify( this._historial ) );
  }

  const params = new HttpParams()
        .set('api_key', this.apiKey)
        .set('limit', '75')
        .set('q', query );

  this.http.get<SearchGifsResponse>(`${ this.servicioUrl }/search`, { params } )
    .subscribe( ( resp ) => {
      this.resultados = resp.data;
      localStorage.setItem('resultados', JSON.stringify( this.resultados ) );
    });

}
```

En el servicio almacenamos la búsqueda del usuario y la preparamos para que se conecte con el buscador al que conectamos nuestra app para que nos de las imágenes de resultado.

1.2.Componente Resultado

HTML

```
<style>
    .gif{
        max-height: 200px;
    }
</style>

<div class="row">
    <div *ngFor="let gif of resultados"
         class="col-md-4 col-sm-6 animate__animated animate__fadeIn">

            <img [src]="gif.images.downsized_medium.url"
                 [alt]="gif.title"
                 class="card-img-top mb-1 gif">


    </div>
</div>
```

Creamos un bucle for que nos muestre los resultados hasta el número total que le hemos dado en el servicio

Typescript

```
export class ResultadosComponent {

  get resultados() {
    return this.gifsService.resultados;
  }

  constructor( private gifsService: GifsService ) { }


}
```

Llamamos al servicio resultados para que nos responda a la búsqueda del usuario

Service

Aquí recibimos la respuesta de la búsqueda realizada por el usuario y que recibimos desde la api exterior que nos surte los gifs.

```
this.http.get<SearchGifsResponse>(`${ this.servicioUrl }/search`, { params } )
  .subscribe( ( resp ) => {
    this.resultados = resp.data;
    localStorage.setItem('resultados', JSON.stringify( this.resultados ) );
  });
```

1.3.Gifs-Page

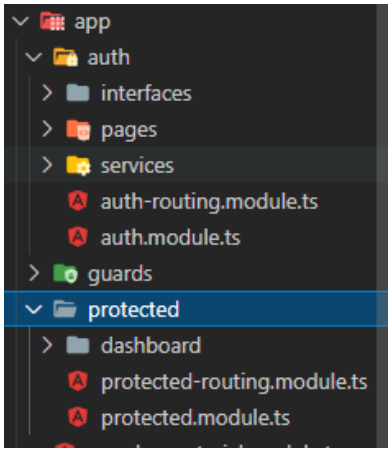
```
<div class="row p-3">
    <div class="col">
        <app-busqueda></app-busqueda>
    </div>
</div>

<div class="row">
    <div class="col">
        <app-resultados></app-resultados>
    </div>
</div>
```

En este componente solamente cargamos los dos anteriores para la vista que recibe el usuario de nuestra aplicación y con la cual interactúa, lo hacemos mediante las clases de los componentes:

- App-búsqueda
- App-resultados

2.Frontend

El frontend está estructurado de la siguiente manera:



1. **Interfaces**: definimos como tiene que ser la estructura de los datos que le mandamos al backend.
2. **Pages:** dentro de pages tenemos las tres páginas que verá el usuario.
   2.1. **Login:** será para que el usuario inicie sesión en la aplicación
   2.2. **Registro:** que servirá para registrarse en la aplicación
   2.3. **Main:** nos sirve para redireccionar y ponerle el fondo al login.
3. **Guards:** La utilizamos para que se validen los jason web token
4. **Services:** aquí definimos ciertas funciones que hace que los componentes tengan funcionalidad.
5. **Protected:** a esta carpeta solo pueden acceder los usuarios que están registrados.
   5.1. **Dashboard:** aquí es donde añadiremos la aplicación de los gifs una vez esté completa.

2.1. Login

HTML

En el html tenemos al típico formulario para iniciar sesión y destacamos la función con la cual verificamos que los datos introducidos son los deseados.

```
<form class="login100-form"
    autocomplete="off"
    [formGroup]="miFormulario"
    (ngSubmit)="login()">
```

En esta parte tenemos el botón de login desactivado hasta que no se introduzca un email y una contraseña que reúna los parámetros.

```
<div class="container-login100-form-btn">
    <div class="wrap-login100-form-btn">
        <div class="login100-form-bgbtn"></div>
        <button class="login100-form-btn"
                type="submit"
                [disabled]="miFormulario.invalid">
            Login
        </button>
    </div>
</div>
```

Typescript

Validamos que el email tiene el formato texto@texto.texto y que la contraseña tiene como mínimo 6 caracteres.

```
export class LoginComponent {

  miFormulario: FormGroup = this.fb.group({
    email:    ['pepe@mail.com', [ Validators.required, Validators.email ]],
    password: ['123456', [ Validators.required, Validators.minLength(6) ]],
  });

  constructor( private fb: FormBuilder,
               private router: Router,
               private authService: AuthService) { }
```

Verificamos que el email y la contraseña son correctos con la base de datos. Y que si es correcto nos redirigirá al componente dashboard donde hemos guardado la aplicación de gifs.

```
login() {

  const { email, password } = this.miFormulario.value;

  this.authService.login( email, password )
    .subscribe( ok => {

      if ( ok === true ) {
        this.router.navigateByUrl('/dashboard');
      } else {
        Swal.fire('Error', ok, 'error');
      }
    });
}
```

2.2. Register

HTML

Aquí tenemos el típico formulario de registro a cual le hacemos una verificación de los datos en el typescript.

Typescript

En estas líneas verificamos que el usuario rellena los tres campos del formulario, los cuales son obligatorios, y que reúnen unos ciertos requisitos.

```
export class RegisterComponent {

  miFormulario: FormGroup = this.fb.group({
    name:     ['María', [ Validators.required ]],
    email:    ['maria@mail.com', [ Validators.required, Validators.email ]],
    password: ['123456', [ Validators.required, Validators.minLength(6) ]],
  });
```

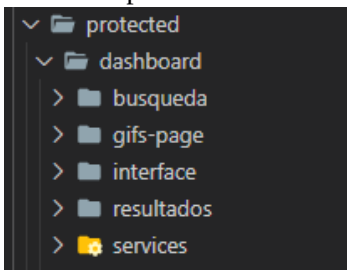
Con la palabra ***required*** hacemos que en campo sea obligatorio rellenarlo

Aquí verificamos que los datos introducidos no están ya registrados en nuestra base de datos, y sino los inserta en esta.

```
registro() {
  const { name, email, password } = this.miFormulario.value;

  this.authService.registro( name, email, password )
    .subscribe( ok => {

      if ( ok === true ) {
        this.router.navigateByUrl('/dashboard');
      } else {
        Swal.fire('Error', ok, 'error');
      }
    });

}
```

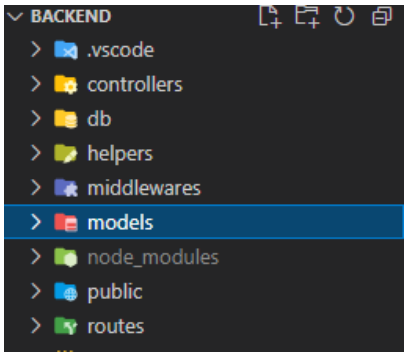
2.3. Dashboard

Aquí podemos ver como en este componente tenemos la aplicación de gifs con su misma estructura para que el usuario pueda hacer uso de la misma cuando inicia sesión.



3. Backend

El backend nos une nuestra aplicación con nuestra base de datos y en la cual añadimos nuestro frontend, una vez está completo y con la aplicación de gifs añadida. Está estructurado de la siguiente manera:



- **controllers:** controlamos las conexiones con la db.
- **db:** que nos comunica con la base de datos.
- **Helpers:** aquí tenemos las funciones que se repiten en distintos componentes para no escribirlas varias veces.
- **Middlewares:** tenemos las validaciones tanto de los formularios como del token.
- **Public:** aquí es donde hemos de guardar el frontend para que se despliegue desde el backend.

En el archivo .env declaramos nuestras variables de entorno , en este caso son las siguientes:

```
PORT=4000
BD_CNN=mongodb+srv://Ana:proyecto2021@micluster.vk7ui.mongodb.net/miBaseDatos?authSource=admin&replicaSet=atlas-aawycr-shard-0&readPreference=p
SECRET_JWT_SEED=EstOdeb3DeSERCompLic4d02080
```

- PORT: en la cual declaramos el puerto por el que salimos
- BD_CNN: nuestra línea de conexión a nuestra base de datos de MongoDB.
- SECRET_JWT_SEED: esta frase es la que os permite “codificar” el jason web token.

3.1. DB

En este directorio tenemos el archivo en el cual se realiza la conexión con nuestra base de datos.

config.js M

db > config.js > ...

```
1  const mongoose = require('mongoose');
2
3  const dbConnection = async() => {
4
5      try {
6
7          await mongoose.connect( process.env.BD_CNN, {
8              useNewUrlParser: true,
9              useUnifiedTopology: true,
10             useCreateIndex: true
11         });
12         console.log('DB Online');
13     } catch (error) {
14         console.log(error);
15         throw new Error('Error a la hora de inicializad DB');
16     }
17 }
18
```

3.2. Controllers

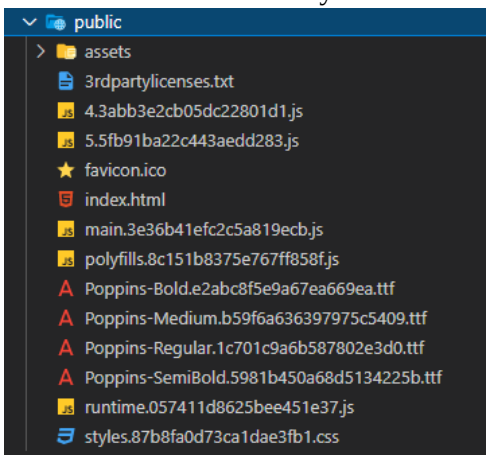
Aquí vemos una parte de los controladores, en este caso para validar el json web token. Con los datos del usuario verificamos que este existe en la base de datos

```
const revalidarToken = async(req, res = response ) => {

    const { uid } = req;

    // Leer la base de datos
    const dbUser = await Usuario.findById(uid);

    // Generar el JWT
    const token = await generarJWT( uid, dbUser.name );

    return res.json({
        ok: true,
        uid,
        name: dbUser.name,
        email: dbUser.email,
        token
    });

}
```

Aquí creamos el json web token el cual nos devolverá y que cada vez que actualizamos el navegador este se actuliza

3.3. Public

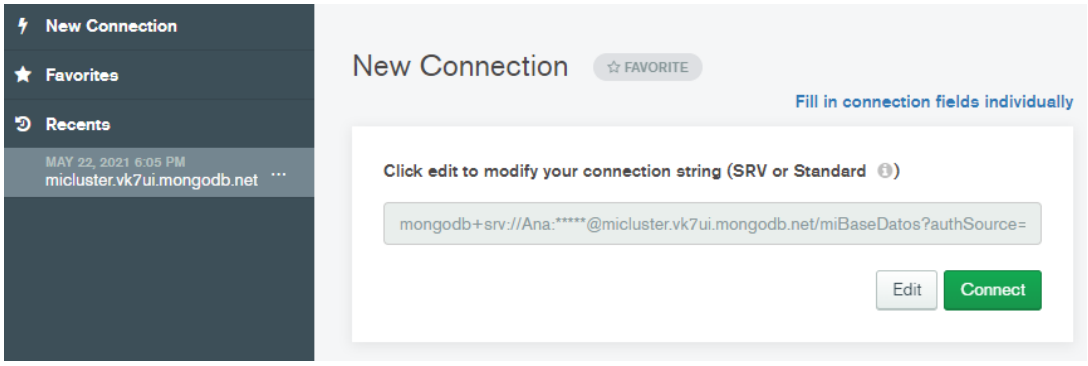
En esta carpeta tenemos la versión de despliegue del frontend, ea cual se realiza con la terminal de comandos en el directorio del mismo y lanzamos el comando ***ng build –prod***.



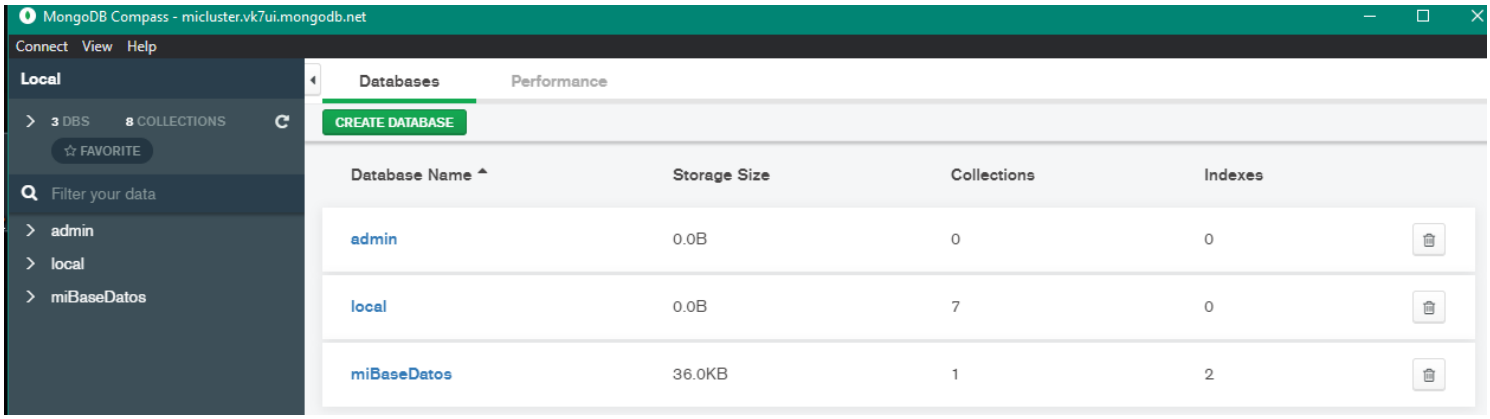
4. Base de datos

La base de datos usada en este proyecto es MongoDB, la cual es una base no sql la cual almacenan los datos guarda los datos en documentos. Estos documentos son almacenados en BSON, que es una representación binaria de JSON.

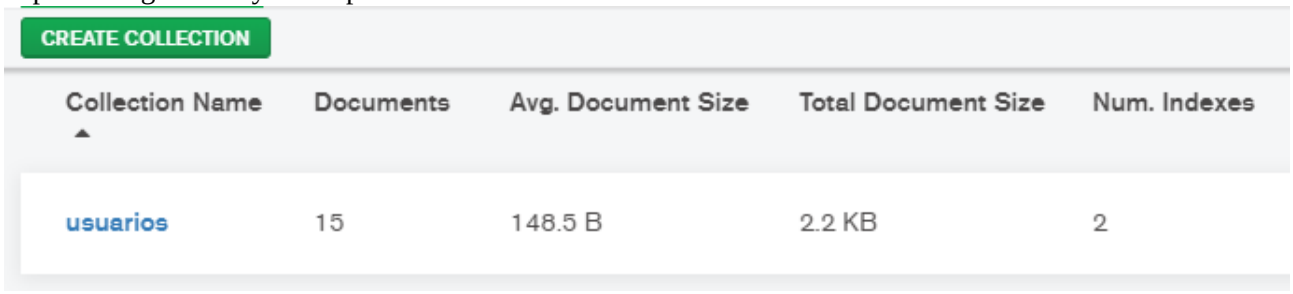
Iniciamos Mongo Compass he introducimos la cadena que usamos para conectarnos con la bd que hemos usado en el archivo de nuestras variables de entorno.



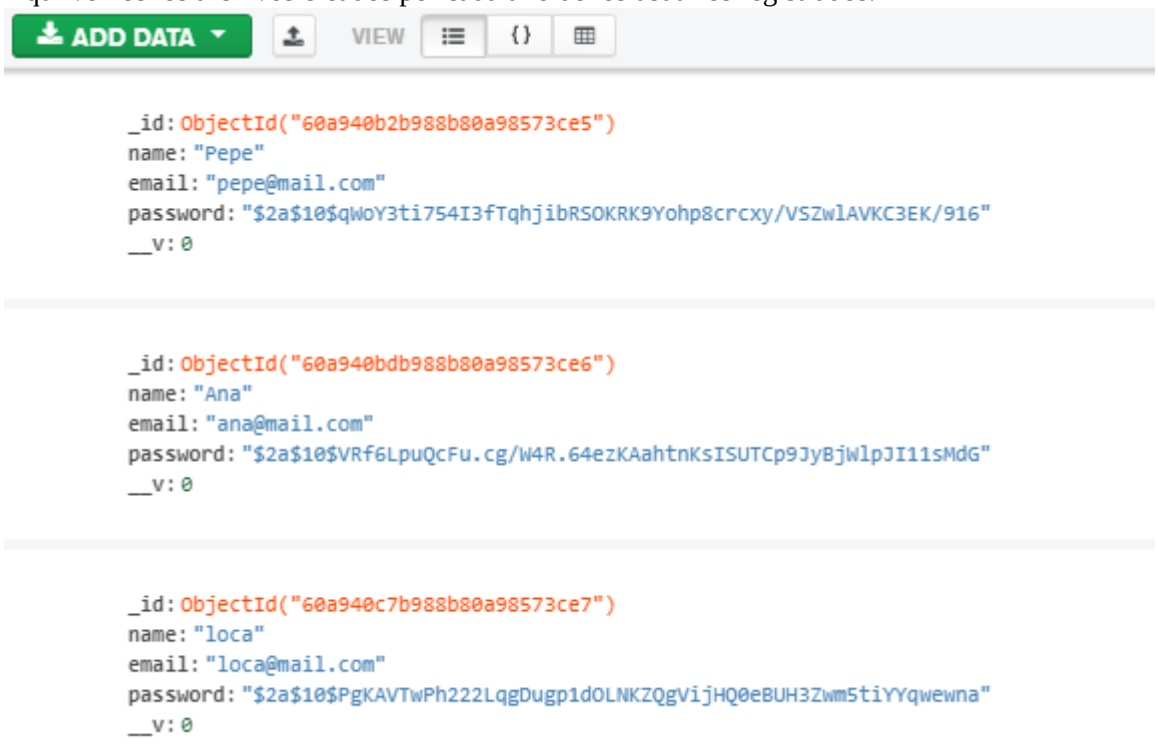
Tras conectarnos se nos muestra la siguiente pantalla donde podemos ver nuestra base de datos, dado que tenemos usuarios registrados. Si no hubiera usuarios la base de datos se borraría sola y al crear un usuario se volvería a crear.
Por eso la creamos en las líneas que mandamos en la conexión entre nuestra base de datos y nuestra aplicación de node.



Aquí podemos ver nuestra única tabla usada en este proyecto. Que también se crea automáticamente al crear el primer registro tal y como pasa con la base de datos.



Aquí vemos los archivos creados por cada uno de los usuarios registrados.



5. Interfaz del usuario

La interfaz de usuario es muy sencilla tenemos un login para iniciar sesión tras la cual nos lleva a la aplicación de buscar gifs, y un formulario de registro.

Para verla en funcionamiento sigue el link.