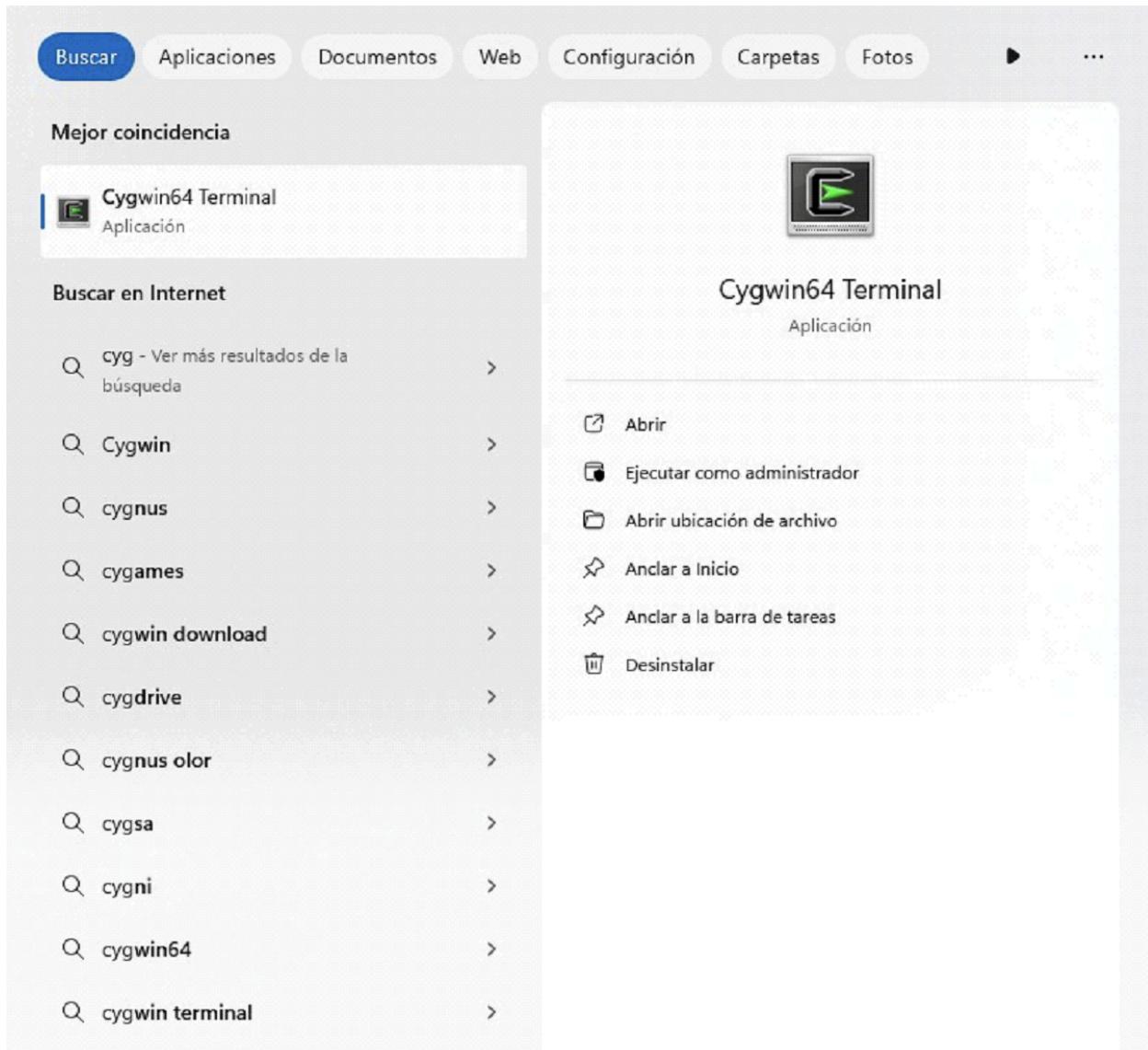


Comenzaremos explicando un poco el desarrollo de la aplicación web, como hemos dicho de desarrollaremos una red social con JavaScript, Angular y NodeJS.

Para ello trabajaremos por secciones, la cual nos complementará en unas 20 secciones para tener la aplicación terminada.

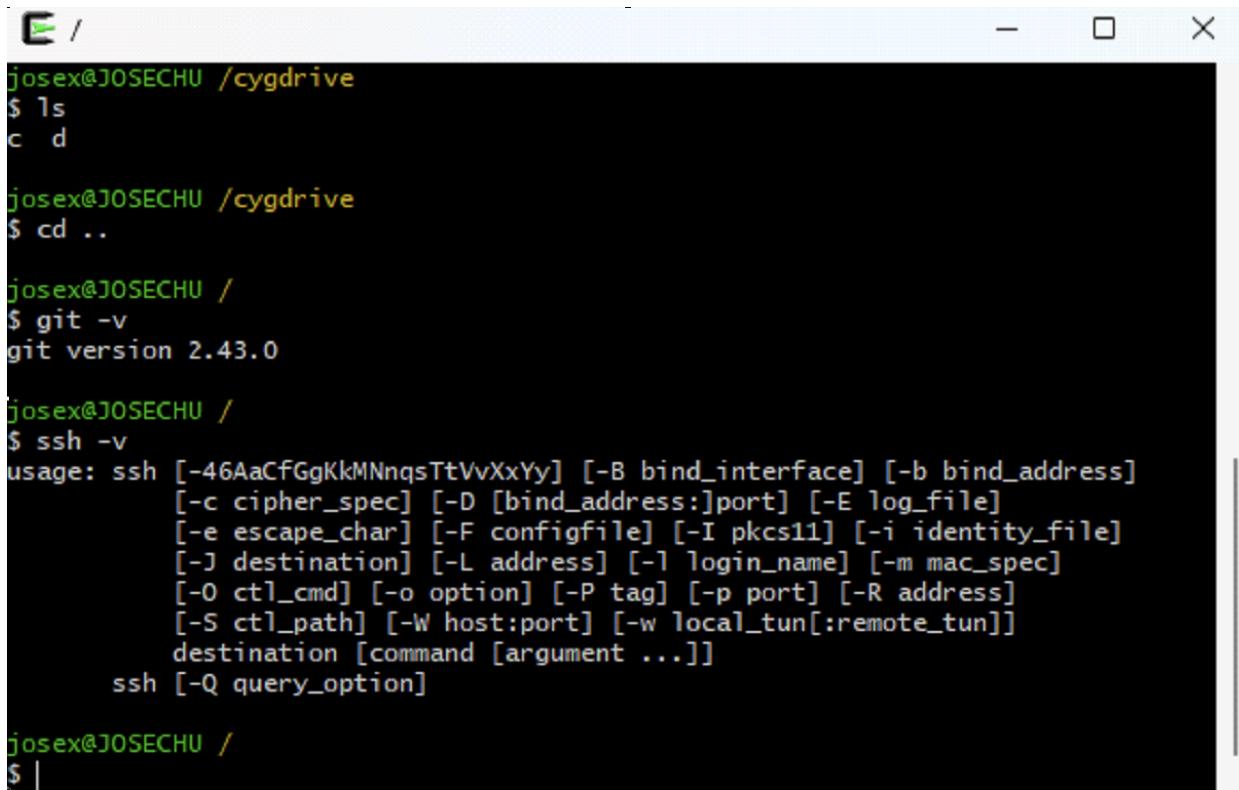
1. Preparación del entorno de desarrollo



En esta sección prepararemos nuestro entorno de trabajo y para ello instalaremos la aplicación de Cygwin: Este es un emulador de la consola de Linux para Windows, este pues nos va a permitir tener una terminal como si fuera la de linux, esta nos va a facilitar el trabajo con la consola.

Esta aplicación, nos permitirá tener el git instalado, el ssh, aplicaciones de consola que están típicamente en linux.

Esta sería la terminal:



```
josex@JOSECHU /cygdrive
$ ls
c d

josex@JOSECHU /cygdrive
$ cd ..

josex@JOSECHU /
$ git -v
git version 2.43.0

josex@JOSECHU /
$ ssh -v
usage: ssh [-46AaCfGgKkMNnqsTtVvXxYy] [-B bind_interface] [-b bind_address]
           [-c cipher_spec] [-D [bind_address:]port] [-E log_file]
           [-e escape_char] [-F configfile] [-I pkcs11] [-i identity_file]
           [-J destination] [-L address] [-l login_name] [-m mac_spec]
           [-O ctl_cmd] [-o option] [-P tag] [-p port] [-R address]
           [-S ctl_path] [-W host:port] [-w local_tun[:remote_tun]]
           destination [command [argument ...]]
           ssh [-Q query_option]

josex@JOSECHU /
$
```

Ahora para seguir preparando el entorno de trabajo, instalaremos MongoDB.

MongoDB es una base de datos no relacional. Es un sistema gestor de base de datos no relacional, que significa esto, que no tenemos ni tablas, ni registros, ni join, ni tablas relacionadas, ni nada de nada, claves primarias, ni nada del estilo. Simplemente se trabaja con objetos JSON o documentos que no se almacenan en JSON como tal si no que se almacenan en BSON, la cual es una representación binaria de JSON.

Una gran diferencia presente en la base de datos relacional es que no tenemos que seguir ningún tipo de esquema, podemos tener documentos relacionados entre sí o documentos dentro de otros.

También vamos a necesitar instalar RoboMongo esta nos permite trabajar con MongoDB de manera visual, con las bases de datos de Mongo y con las colecciones de datos de Mongo. Esta herramienta se llama ROBO3T:

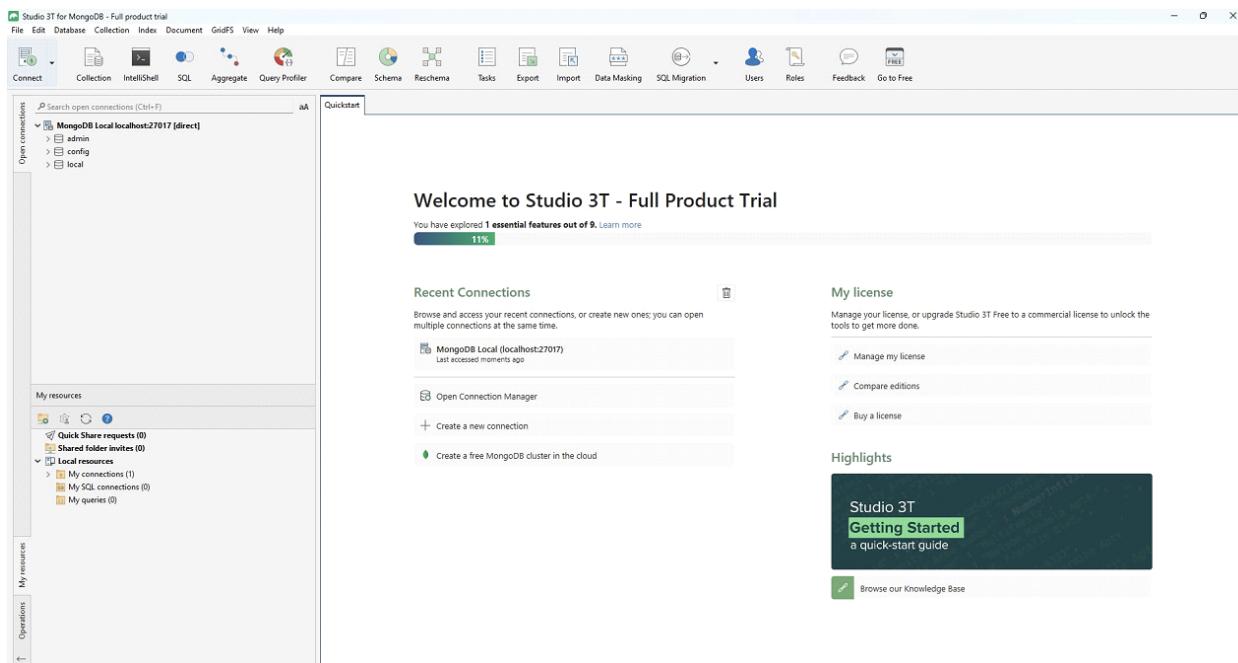
Robo 3T is now **Studio 3T**

Studio 3T Free edition extends and replaces Robo 3T with:

- Easy MongoDB Connections Manager
- IntelliShell with fast auto-completion
- Dark theme, with multiple customizations
- ... and much, much more

[Download Studio 3T today](#)

Esto sería un ejemplo del entorno de trabajo:



Por último en esta sección vamos a instalar NodeJS es un entorno de ejecución de JavaScript multiplataforma, de Código abierto y gratuito que permite a los desarrolladores crear servidores, aplicaciones web, herramientas de línea de comandos y scripts.

Descargar Node.js®

Descarga Node.js como quieras.

Instalador prediseñado Binarios prediseñados Gerente de empaquetación Código fuente

Quiero el v20.12.2 (LTS) versión de Node.js para ventanas correr x64

[⬇ Descargar Node.js v20.12.2](#)

Node.js incluye npm (10.5.0) ↗ .

Lea el registro de cambios para Esta versión ↗

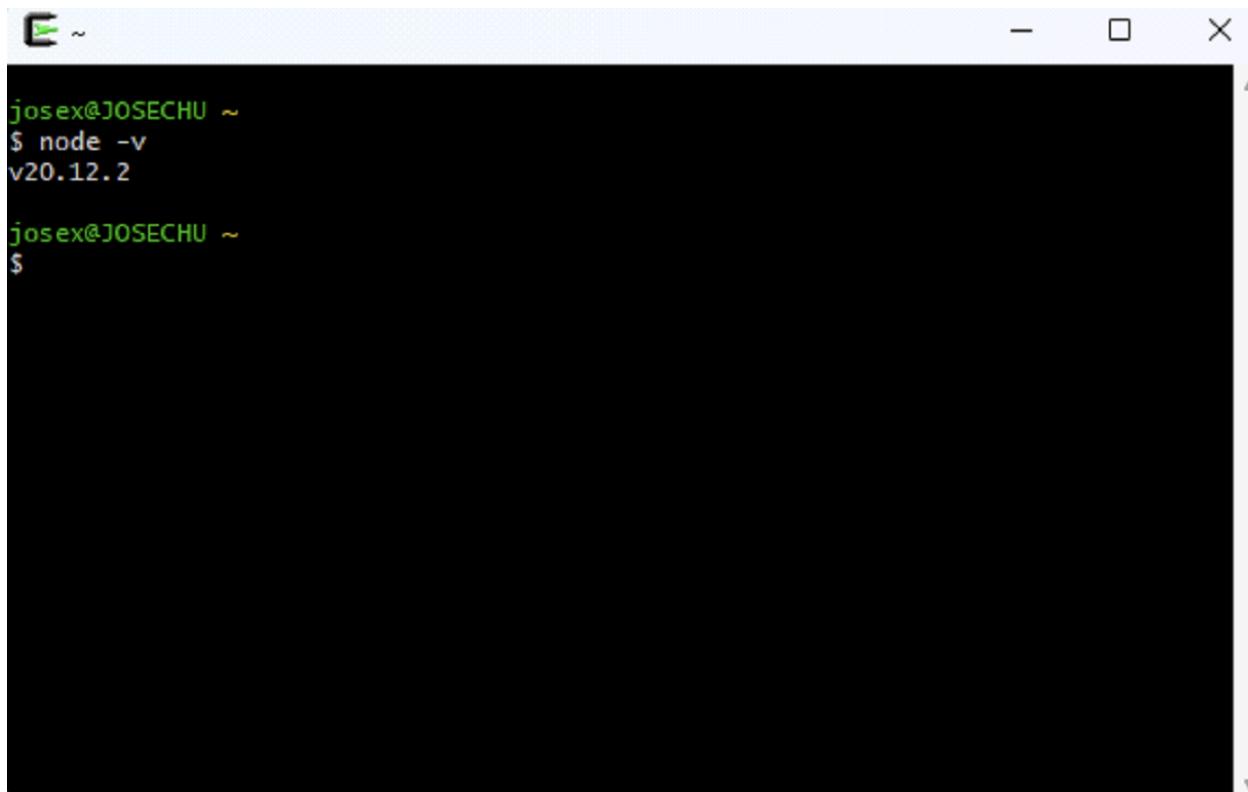
Lea la publicación del blog para esta versión. ↗

Aprenda a verificar SHASUMS firmados ↗

Consulte todas las opciones de descarga de Node.js disponibles ↗

Hemos descargado la versión última de NodeJS:

Aquí podemos observar en la terminal como está ejecutándose:



```
josex@JOSECHU ~
$ node -v
v20.12.2

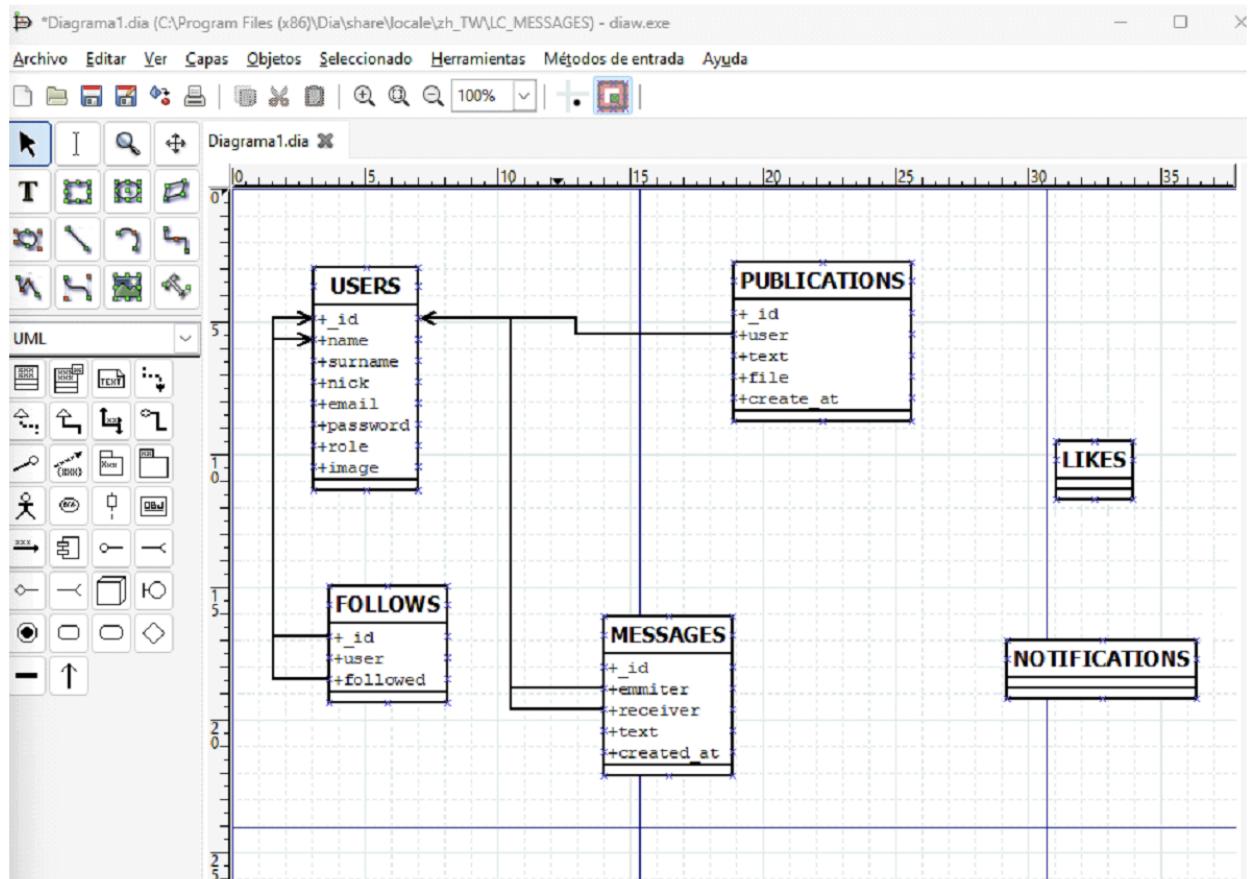
josex@JOSECHU ~
$
```

Resumiendo vamos a utilizar NodeJS, SublimeTest, MongoDB y la consola de Cygwin,

seguidamente vamos a comenzar con el diseño de nuestra base de datos, vamos a hacer un pequeño esquema de que es lo que va a tener nuestra base de datos y posteriormente ya vamos a comenzar a programar la parte de nuestro backend de nuestra red social.

2. La Base de Datos

Vamos hacer el diseño de nuestra base de datos, como bien comentamos anteriormente en MongoDB no existen las tablas como tal, sino que existen las colecciones de documentos y los documentos dentro de ellas. Por lo tanto, vamos a hacer el esquema básico del cómo va a ser cada uno de los documentos que conformen las colecciones:



Esto va a constar nuestra esquema básico, aunque es súper escalable, pero comenzaremos con los usuarios: En cual haremos el registro, el login, después la parte de edición de usuarios, subida de imágenes, el rol, el nick, entre otras cosas y hacer que funcione correctamente.

Después de todo esto tener un sistema de seguimiento entre los usuarios los cuales puedan seguirse unos a otros, tener un listado de los usuarios que se siguen por cada usuario y los usuarios seguidos de cada uno.

Seguidamente nos centraremos en la parte de publicaciones en la cual cada usuario pueda hacer sus publicaciones con sus ficheros adjuntos y por último centrarnos en la parte de mensajes en la cual nosotros podremos enviar y recibir mensajes y listar los mensajes recibidos y enviados.

Sobre todo nos centraremos en hacer el conjunto de todas las funcionalidades principales además hacer trabajar estas 4 entidades comentadas y que funcionen entre sí. Y por último podemos plantearnos el hecho de hacer la parte de LIKES y Notifications.

3. Empezando a Desarrollar el API RESTful – Backend

3.1. Empezar un proyecto de Node.js

Lo primero de todo, será generar un proyecto Node.js, utilizando la carpeta C:/Proyecto, y se creará una carpeta dentro llamada 'proyecto-final-master' con el comando mkdir, y acto seguido, se ejecutará el comando 'npm init' para crear el documento package.json, el cual nos lanzará un pequeño asistente preguntando por diferentes parámetros, como se muestra a continuación:

```

/cydrive/c/Proyecto/proyecto-final-master/api
javierlopezgarcia@Javier ~
$ node -v
v16.20.0
javierlopezgarcia@Javier ~
$ cd c:/Proyecto
javierlopezgarcia@Javier /cydrive/c/Proyecto
$ mkdir proyecto-final-master
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.
See `npm help init` for definitive documentation on these fields
and exactly what they do.

use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press AC at any time to quit.
package name: (proyecto-final-master)
version: 1.0.0
description: Backend para el proyecto final de Máster Full Stack III de desarrollo de una red social con Angular y Node
entry point: (index.js)
test command:
git repository: github.com/Javieleh41/Proyecto-Final-M-ster
keywords:
author: Javier Y José Luis
license: (ISC) MIT
About to write to c:/Proyecto/proyecto-final-master/package.json:

{
  "name": "proyecto-final-master",
  "version": "1.0.0",
  "description": "Backend para el proyecto final de Máster Full Stack III de desarrollo de una red social con Angular y Node",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "github.com/Javieleh41/Proyecto-Final-M-ster"
  },
  "author": "Javier Y José Luis",
  "license": "MIT"
}

Is this OK? (yes)
npm notice
npm notice New major version of npm available! 8.29.4 => 10.5.2
npm notice ChangeLog: https://github.com/npm/cli/releases/tag/v10.5.2
npm notice Run `npm install -g npm@10.5.2` to update!
npm notice

```

Por tanto, al abrir el proyecto en Visual Studio, quedará el siguiente archivo package.json:

```

{} package.json ×
api > {} package.json > {} devDependencies > nodemon
1  {
2    "name": "proyecto-final-master",
3    "version": "1.0.0",
4    "description": "Backend para el proyecto final de Máster Full Stack III de desarrollo de una red social c",
5    "main": "index.js",
6    ▷ Debug
7    "scripts": {
8      "test": "echo \\\"Error: no test specified\\\" && exit 1"
9    },
10   "repository": {
11     "type": "git",
12     "url": "github.com/Javieleh41/Proyecto-Final-M-ster"
13   },
14   "author": "Javier Y Jose Luis",
15   "license": "MIT",

```

3.2. Instalar librerías y paquetes

Se instalarán las siguientes librerías y dependencias:

- "bcrypt-nodejs": "^0.0.3" --> sirve para utilizar el algoritmo de encriptación de crypt, que es mucho más optimizado para generar contraseñas
- "body-parser": "^1.20.2" --> se utilizar para convertir el contenido que nos llega en la petición en contenido usable por JavaScript

- "connect-multiparty": "^2.2.0" --> sirve para realizar subidas de archivos en el middleware
- "express": "^4.19.2" --> es el framework HTTP, y el que nos va a generar la ruta
- "jwt-simple": "^0.5.6" --> va a servir para gestionar los tokens y la autenticación
- "moment": "^2.30.1" --> se utiliza para generar fechas
- "mongoose": "^8.3.2" --> para trabajar con MongoDB dentro de Node.js
- "nodemon": "^3.1.0" --> su función es refrescar el servidor cada vez que se realizar algún tipo de cambio en nuestro código fuente en el backend con Node.js

A continuación, se representa en la terminal la ejecución de las dependencias anteriores:

```
/cygdrive/c/Proyecto/proyecto-final-master/api
javierlopezgarcia@javier /cygdrive/c/Proyecto/proyecto-final-master/api
$ npm install express
added 64 packages, and audited 65 packages in 9s
12 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities

javierlopezgarcia@javier /cygdrive/c/Proyecto/proyecto-final-master/api
$ npm install bcrypt-nodejs --save
npm WARN deprecated bcrypt-nodejs@0.3: bcrypt-nodejs is no longer actively maintained. Please use bcrypt or bcryptjs. See https://github.com/kelektiv/node.bcrypt.js/wiki/bcrypt-vs-brypt.js to learn more about these two options

added 1 package, and audited 66 packages in 3s
12 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities

javierlopezgarcia@javier /cygdrive/c/Proyecto/proyecto-final-master/api
$ npm install body-parser --save
up to date, audited 66 packages in 2s
12 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities

javierlopezgarcia@javier /cygdrive/c/Proyecto/proyecto-final-master/api
$ npm install connect-multiparty --save
added 14 packages, and audited 80 packages in 5s
12 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities

javierlopezgarcia@javier /cygdrive/c/Proyecto/proyecto-final-master/api
$ npm install mongoose --save
npm WARN EBADENGINE Unsupported engine {
  npm WARN EBADENGINE   package: 'mongoose@8.3.2',
  npm WARN EBADENGINE   required: { node: '>=16.20.1' },
  npm WARN EBADENGINE   current: { node: 'v16.20.0', npm: '8.19.4' }
  npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
  npm WARN EBADENGINE   package: 'bson@6.6.0',
  npm WARN EBADENGINE   required: { node: '>=16.20.1' },
  npm WARN EBADENGINE   current: { node: 'v16.20.0', npm: '8.19.4' }
  npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
  npm WARN EBADENGINE   package: 'mongodb@6.5.0',
  npm WARN EBADENGINE   required: { node: '>=16.20.1' },
  npm WARN EBADENGINE   current: { node: 'v16.20.0', npm: '8.19.4' }
  npm WARN EBADENGINE }

added 20 packages, and audited 100 packages in 11s
```

```

[javierlopezgarcia@javier /cygdrive/c/Proyecto/proyecto-final-master/api]
$ npm install --save-dev
added 20 packages, and audited 100 packages in 11s
13 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities

[javierlopezgarcia@javier /cygdrive/c/Proyecto/proyecto-final-master/api]
$ npm install moment
npm WARN EBADENGINE Unsupported engine {
  npm WARN EBADENGINE   package: 'bson@6.6.0',
  npm WARN EBADENGINE   required: { node: '^>16.20.1' },
  npm WARN EBADENGINE   current: { node: 'v16.20.0', npm: '8.19.4' }
}
npm WARN EBADENGINE Unsupported engine {
  npm WARN EBADENGINE   package: 'mongodb@6.5.0',
  npm WARN EBADENGINE   required: { node: '^>16.20.1' },
  npm WARN EBADENGINE   current: { node: 'v16.20.0', npm: '8.19.4' }
}
npm WARN EBADENGINE Unsupported engine {
  npm WARN EBADENGINE   package: 'mongoose@8.3.2',
  npm WARN EBADENGINE   required: { node: '^>16.20.1' },
  npm WARN EBADENGINE   current: { node: 'v16.20.0', npm: '8.19.4' }
}
npm WARN EBADENGINE Unsupported engine {
  npm WARN EBADENGINE   package: 'node@16.20.0',
  npm WARN EBADENGINE   required: { node: '^>16.20.1' },
  npm WARN EBADENGINE   current: { node: 'v16.20.0', npm: '8.19.4' }
}

added 1 package, and audited 101 packages in 3s
13 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities

[javierlopezgarcia@javier /cygdrive/c/Proyecto/proyecto-final-master/api]
$ npm install moment --save
npm WARN EBADENGINE Unsupported engine {
  npm WARN EBADENGINE   package: 'bson@6.6.0',
  npm WARN EBADENGINE   required: { node: '^>16.20.1' },
  npm WARN EBADENGINE   current: { node: 'v16.20.0', npm: '8.19.4' }
}
npm WARN EBADENGINE Unsupported engine {
  npm WARN EBADENGINE   package: 'mongodb@6.5.0',
  npm WARN EBADENGINE   required: { node: '^>16.20.1' },
  npm WARN EBADENGINE   current: { node: 'v16.20.0', npm: '8.19.4' }
}
npm WARN EBADENGINE Unsupported engine {
  npm WARN EBADENGINE   package: 'mongoose@8.3.2',
  npm WARN EBADENGINE   required: { node: '^>16.20.1' },
  npm WARN EBADENGINE   current: { node: 'v16.20.0', npm: '8.19.4' }
}
npm WARN EBADENGINE Unsupported engine {
  npm WARN EBADENGINE   package: 'node@16.20.0',
  npm WARN EBADENGINE   required: { node: '^>16.20.1' },
  npm WARN EBADENGINE   current: { node: 'v16.20.0', npm: '8.19.4' }
}

added 1 package, and audited 102 packages in 5s
13 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities

[javierlopezgarcia@javier /cygdrive/c/Proyecto/proyecto-final-master/api]
$ npm install nodemon --save-dev
npm WARN EBADENGINE Unsupported engine {
  npm WARN EBADENGINE   package: 'bson@6.6.0',
  npm WARN EBADENGINE   required: { node: '^>16.20.1' },
  npm WARN EBADENGINE   current: { node: 'v16.20.0', npm: '8.19.4' }
}

```

```

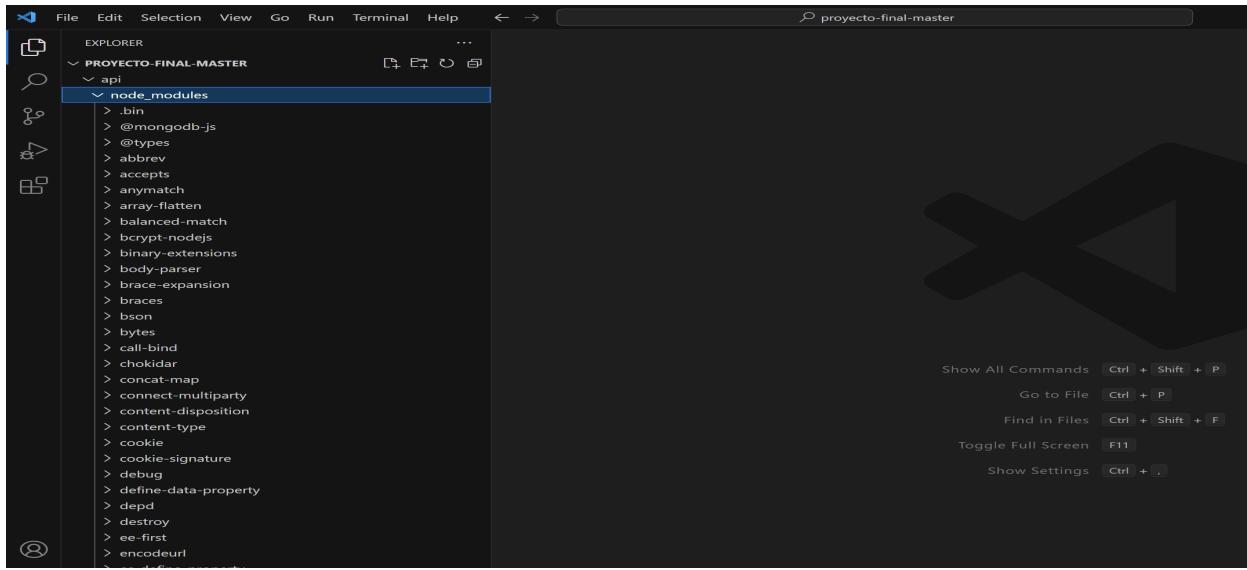
[javierlopezgarcia@Javier /cygdrive/c/Proyecto/proyecto-final-master/api]
$ npm install nodemon --save-dev
npm WARN EBADENGINE Unsupported engine {
  npm WARN EBADENGINE   package: 'bson@6.6.0',
  npm WARN EBADENGINE   required: { node: '^>16.20.1' },
  npm WARN EBADENGINE   current: { node: 'v16.20.0', npm: '8.19.4' }
}
npm WARN EBADENGINE Unsupported engine {
  npm WARN EBADENGINE   package: 'mongodb@6.5.0',
  npm WARN EBADENGINE   required: { node: '^>16.20.1' },
  npm WARN EBADENGINE   current: { node: 'v16.20.0', npm: '8.19.4' }
}
npm WARN EBADENGINE Unsupported engine {
  npm WARN EBADENGINE   package: 'mongoose@8.3.2',
  npm WARN EBADENGINE   required: { node: '^>16.20.1' },
  npm WARN EBADENGINE   current: { node: 'v16.20.0', npm: '8.19.4' }
}
npm WARN EBADENGINE Unsupported engine {
  npm WARN EBADENGINE   package: 'node@16.20.0',
  npm WARN EBADENGINE   required: { node: '^>16.20.1' },
  npm WARN EBADENGINE   current: { node: 'v16.20.0', npm: '8.19.4' }
}

added 33 packages, and audited 135 packages in 7s
17 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities

[javierlopezgarcia@Javier /cygdrive/c/Proyecto/proyecto-final-master/api]
$ 

```

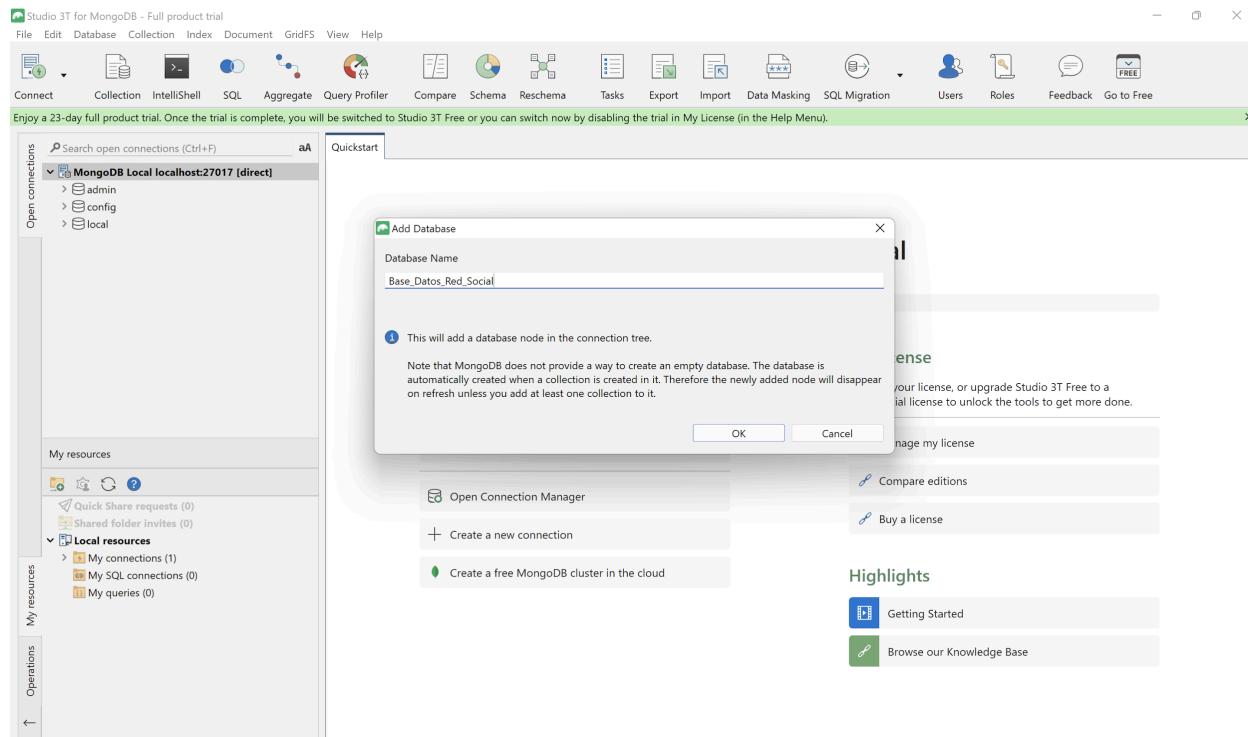
Con todo lo anterior, la carpeta ‘node_modules’, y los archivos ‘package-lock.json’ y ‘package.json’, quedaría configurado de la siguiente manera:



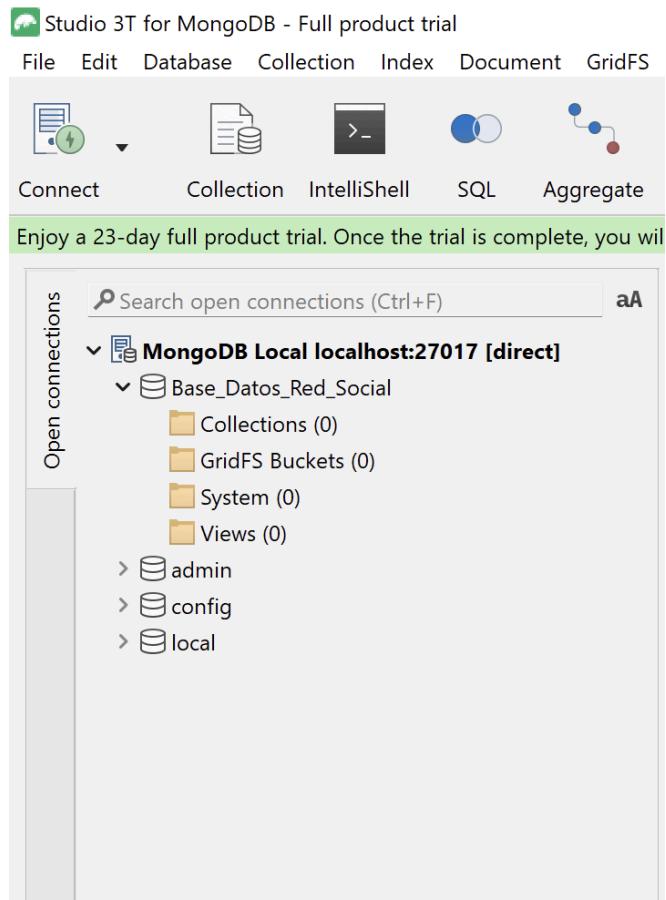
```
1  {
2   "name": "proyecto-final-master",
3   "version": "1.0.0",
4   "lockfileVersion": 2,
5   "requires": true,
6   "packages": {
7     ".": {
8       "name": "proyecto-final-master",
9       "version": "1.0.0",
10      "license": "MIT",
11      "dependencies": {
12        "bcrypt-nodejs": "^0.0.3",
13        "body-parser": "1.20.2",
14        "connect-multiparty": "^2.2.0",
15        "express": "4.19.2",
16        "jwt-simple": "^0.5.6",
17        "moment": "2.30.1",
18        "mongoose": "^8.3.2"
19      },
20      "devDependencies": {
21        "nodemon": "^3.1.0"
22      }
23    },
24    "node_modules/@mongodb-js/saslprep": {
25      "version": "1.1.5",
26      "resolved": "https://registry.npmjs.org/@mongodb-js/saslprep/-/saslprep-1.1.5.tgz",
27      "integrity": "sha512-XLNOMH66KhJzUJNwT/q1MnS4WsND05ASdySH3EtK+f4r/CFGa3jT4GNi4m0itGwXtdLgQjkQjxs",
28      "dependencies": {
29        "sparse-bitfield": "3.0.3"
30      }
31    },
32    "node_modules/@types/webidl-conversions": {
33      "version": "7.0.3",
34      "resolved": "https://registry.npmjs.org/@types/webidl-conversions/-/webidl-conversions-7.0.3.tgz",
35      "integrity": "sha512-CijJvcRtIgzadHCYkw7dqEnMNrhGZ1Yk95Mj90yktqV8uVT8fD2BF0B751uwBE3kjZz+4UyPmFw/Ix",
36    },
37    "node_modules/@types/whatwg-url": {
38      "version": "11.0.4",
39      "resolved": "https://registry.npmjs.org/@types/whatwg-url/-/whatwg-url-11.0.4.tgz",
40    }
41  }
42
```

3.3. Crear la base de datos

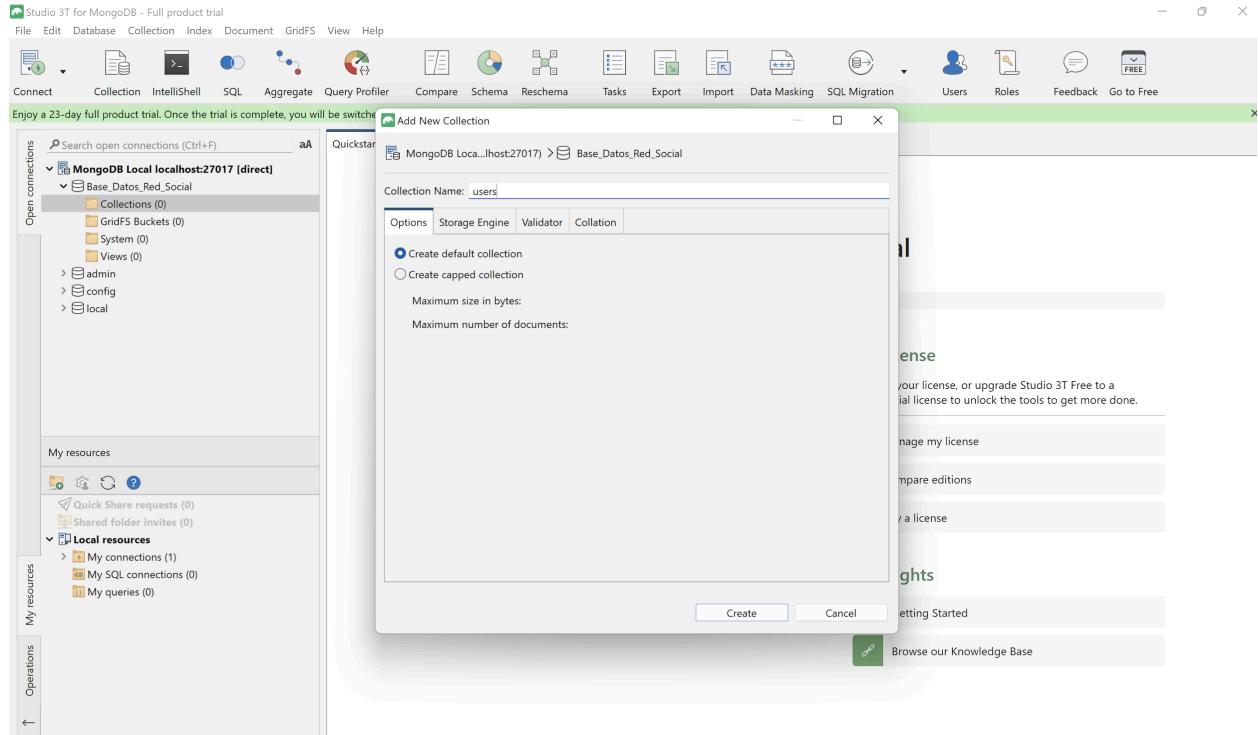
Para ello vamos a crear la base de datos con el programa anteriormente instalado, llamado 'Studio 3T'



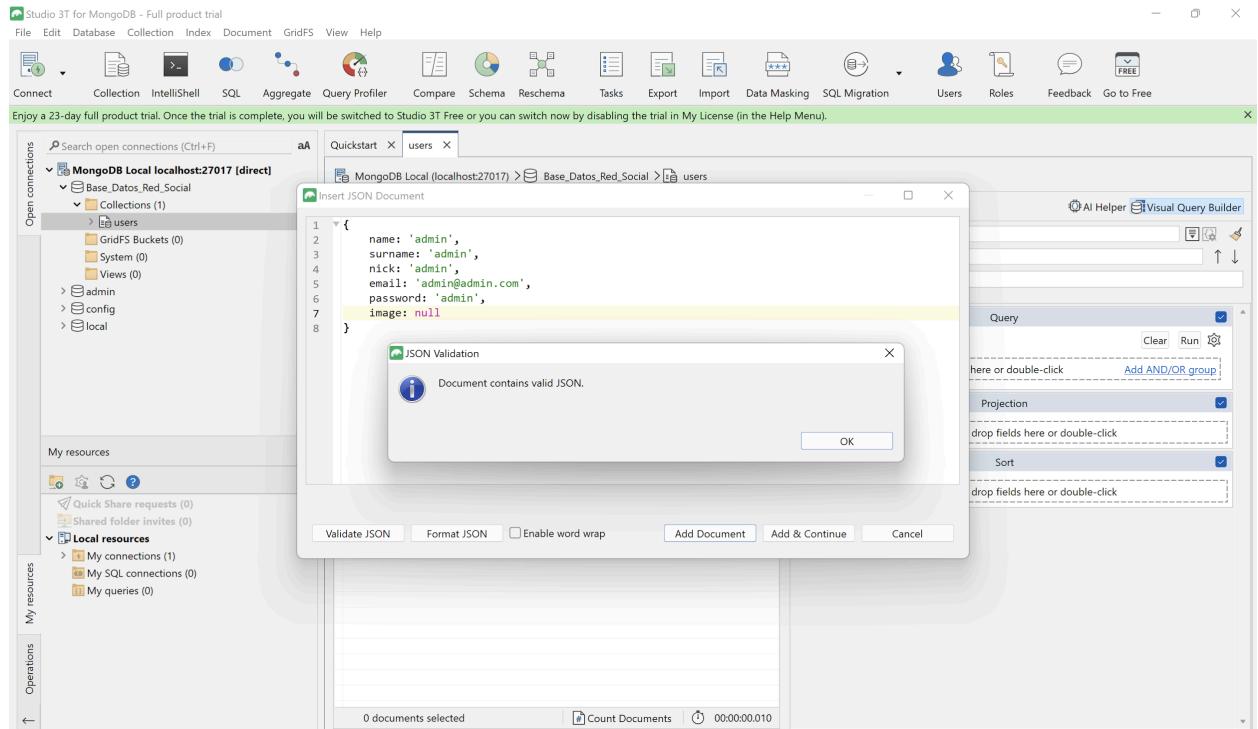
Al darle en ‘ok’, se creará dicha base de datos y aparecerá en el panel izquierdo



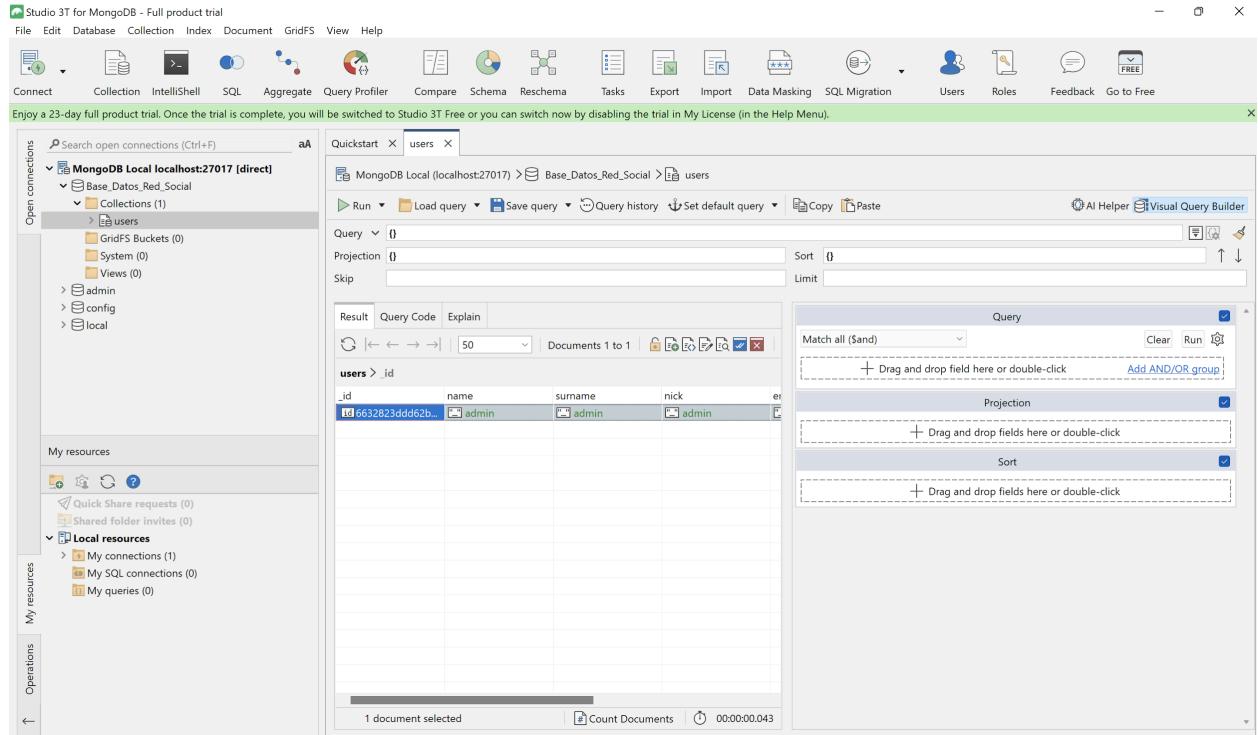
A continuación, se va a crear una nueva colección llamada ‘users’



Y posteriormente, se va a llevar a cabo la creación de un documento json con diferentes campos, y su validación dentro de dicha colección:



Consecuentemente, al darle al play, aparecerá el usuario creado con los campos:



3.4. Conexión a la base de datos

Ahora, se creará el archivo ‘index.js’ dentro de la carpeta ‘API’, que será el encargado de lanzar nuestro backend, y aquí también se realizará la conexión a Mongodb y la conexión de un servidor web con node, y generalmente a partir de este fichero se harán otras peticiones y carga de diferentes procesos. Se implementará el siguiente código ‘JavaScript’:

```

    api > JS index.js > ...
1  'use strict';
2
3  var mongoose = require ('mongoose');
4
5  mongoose.Promise = global.Promise;
6  mongoose.connect('mongodb://localhost:27017/Base_Datos_Red_Social', {useMongoClient: true})
7      .then(() => {
8          console.log("La conexión a la base de datos Base_Red_Social se ha realizado correctamente")
9      })
10     .catch(err => console.log(err));

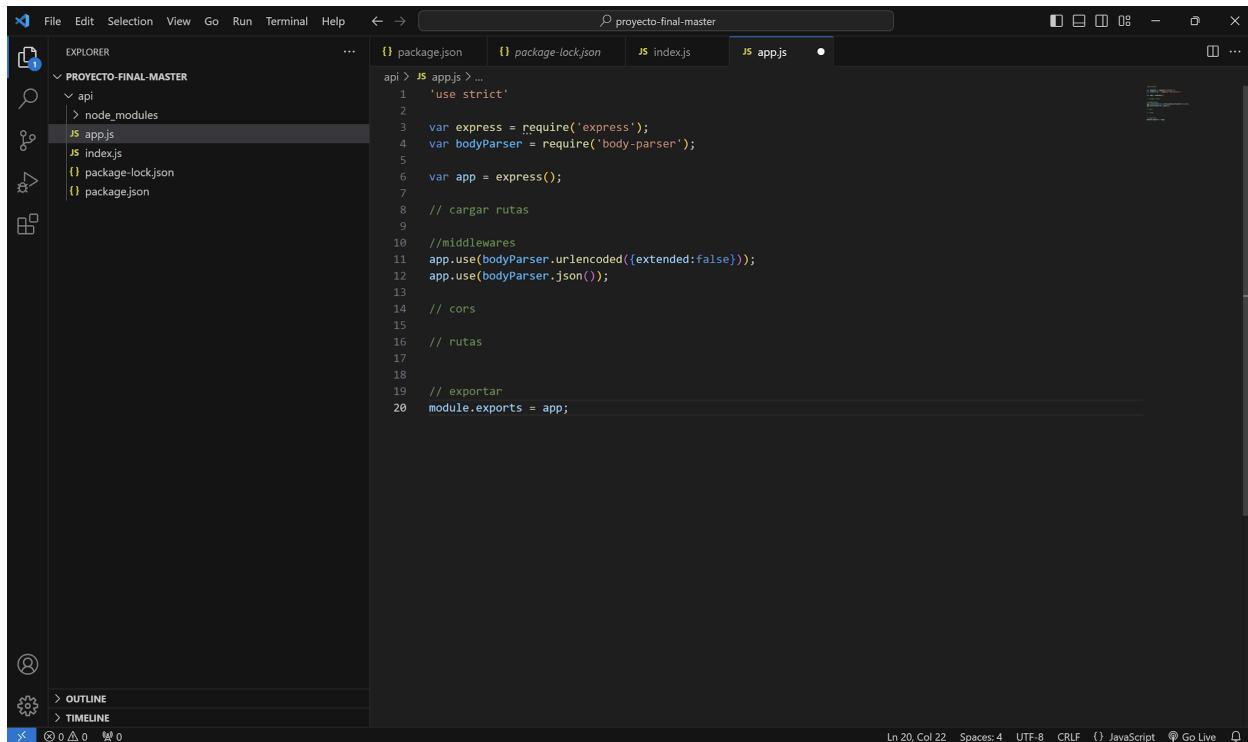
```

Posteriormente, el siguiente paso es arrancar el proyecto, lanzando el fichero anterior ‘index.js’ con la terminal con el comando ‘node index.js’:

```
javierlopezgarcia@Javier /cygdrive/c/Proyecto/proyecto-final-master/api
$ node index.js
(node:2580) [MONGODB DRIVER] warning: useNewUrlParser is a deprecated option; useNewUrlParser has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
(Use node --trace-warnings ... to show where the warning was created)
(node:2580) [MONGODB DRIVER] warning: useUnifiedTopology is a deprecated option; useUnifiedTopology has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
La conexión a la base de datos Base_Red_Social se ha realizado correctamente
```

3.5. Crear el servidor web

Lo primero que se va a hacer es crear un archivo llamado ‘app.js’, y se va a configurar dentro de este archivo todo lo relacionado con ‘express’, bodyParser (convertir los body que nos lleguen en las peticiones en objetos JavaScript), se cargarán también rutas y finalmente se implementará en el archivo ‘index.js’. Este será el código implementado, con los middlewares y el archivo a exportar:



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under 'PROYECTO-FINAL-MASTER'. It includes 'api', 'node_modules', 'app.js', 'index.js', 'package-lock.json', and 'package.json'.
- Code Editor:** The 'app.js' file is open and visible. The code is as follows:

```
api > JS app.js > ...
1  'use strict';
2
3  var express = require('express');
4  var bodyParser = require('body-parser');
5
6  var app = express();
7
8  // cargar rutas
9
10 //middlewares
11 app.use(bodyParser.urlencoded({extended:false}));
12 app.use(bodyParser.json());
13
14 // cors
15
16 // rutas
17
18
19 // exportar
20 module.exports = app;
```

At the bottom of the editor, status bar text includes: Ln 20, Col 22, Spaces: 4, UTF-8, CRLF, {} JavaScript, Go Live.

Entonces, en el archivo ‘index.js’ se cargará el archivo anterior recientemente creado ‘app.js’ con el comando ‘var app = require('./app’); ‘var puerto = 3800’ y el listen dentro del then, quedando todo el código de esta manera:

```

    api > JS index.js > ...
1   'use strict';
2
3   const mongoose = require('mongoose');
4   var app = require('../app');
5   var port = 3800;
6
7   // Conexión Database
8   mongoose.Promise = global.Promise;
9   mongoose.connect('mongodb://localhost:27017/Base_Datos_Red_Social', { useNewUrlParser: true, useUnifiedTopology: true })
10  .then(() => {
11    console.log("La conexión a la base de datos Base_Red_Social se ha realizado correctamente!");
12
13    // crear servidor
14    app.listen(port, () => {
15      console.log(`Servidor corriendo en http://localhost:3800`);
16    });
17
18  })
19  .catch(err => console.log(err));

```

Automáticamente al guardar el archivo, en la consola, al tener el script 'start' configurado, nos lanzará que la conexión a la base de datos se ha realizado correctamente y el servidor está corriendo en localhost:3800:

```

La conexión a la base de datos Base_Red_Social se ha realizado correctamente!!
[nodemod] restarting due to changes...
[nodemod] starting...
(node:11028) [MONGODB DRIVER] warning: useNewUrlParser is a deprecated option: useNewUrlParser has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
(Use `node --trace-warnings ...` to show where the warning was created)
(node:11028) [MONGODB DRIVER] warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js driver version 4.0.0 and will be removed in the next major version
La conexión a la base de datos Base_Red_Social se ha realizado correctamente!!
Servidor corriendo en http://localhost:3800

```

3.6. Configuración Cliente RESTful con Postman

Es necesario descargar e instalar Postman desde el sitio oficial, y se podrán hacer peticiones GET y POST:

You are using the Lightweight API Client, sign in or create an account to work with collections, environments and unlock all free features in Postman.

History New Import GET http://localhost:3800/pruebas + ***

HTTP http://localhost:3800/pruebas

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Key	Value
Key	Value

Body Cookies Headers (7) Test Results Status: 200 OK Time: 103 ms Size: 292 B Save Response

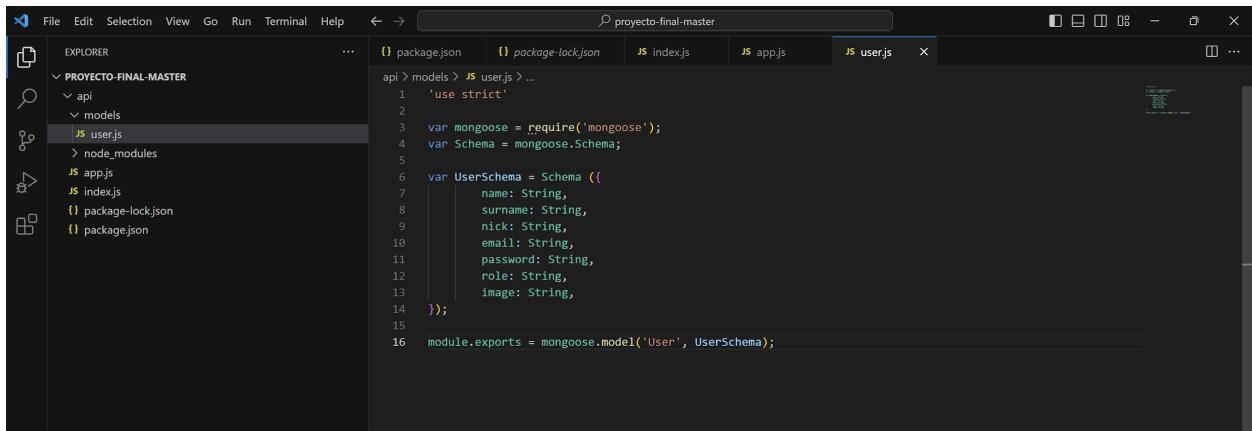
Pretty	Raw	Preview	Visualize	JSON
1	2	3	"message": "Acción de pruebas en el servidor de NodeJS"	

Console Not connected to a Postman account

3.7. Crear de todos los modelos

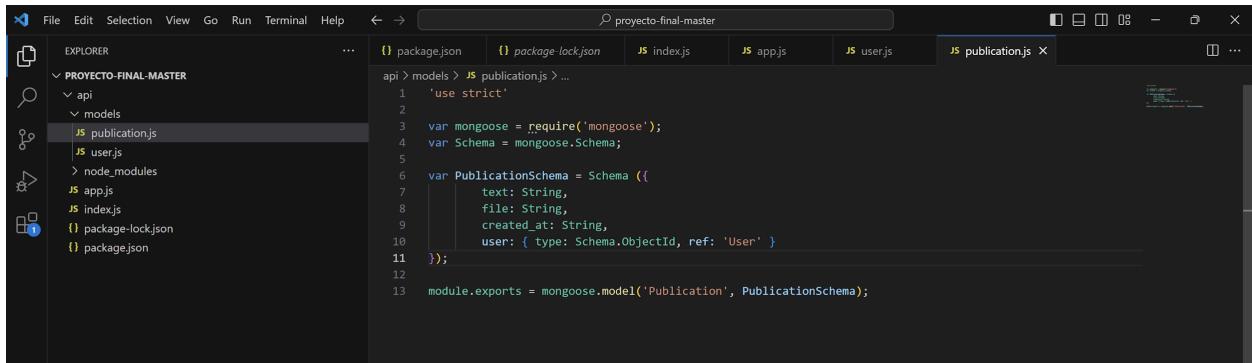
Se crearán cada uno de los modelos para construir la aplicación: Users, Follows, Publications, Likes. Y por tanto, para crear los modelos, se necesitará crear una carpeta llamada ‘Models’ en el directorio de la API dentro de nuestro proyecto y un archivo de js para cada uno de los modelos dentro de la carpeta. Adicionalmente, en cada modelo se van a introducir los campos que se definieron en el esquema dibujado de la base de datos:

User.js



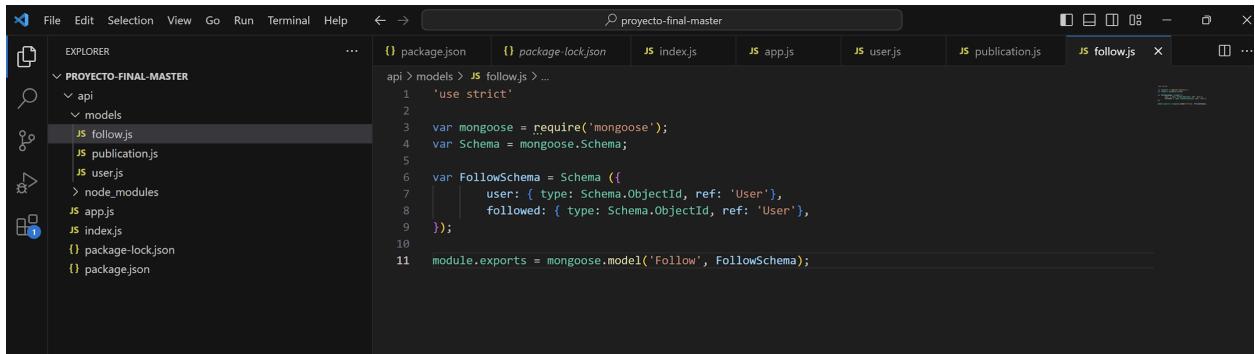
```
api > models > JS user.js > ...
1 'use strict'
2
3 var mongoose = require('mongoose');
4 var Schema = mongoose.Schema;
5
6 var UserSchema = Schema ({
7   name: String,
8   surname: String,
9   nick: String,
10  email: String,
11  password: String,
12  role: String,
13  image: String,
14 });
15
16 module.exports = mongoose.model('User', UserSchema);
```

Publication.js



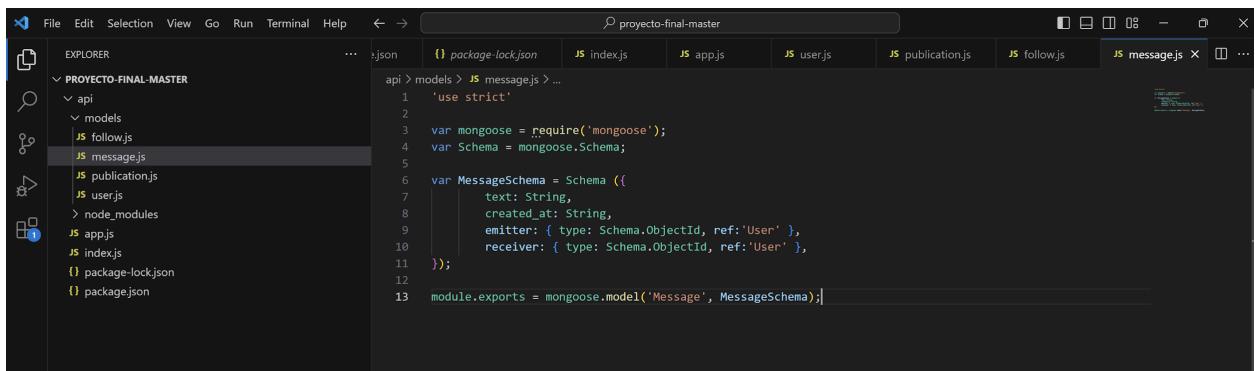
```
api > models > JS publication.js > ...
1 'use strict'
2
3 var mongoose = require('mongoose');
4 var Schema = mongoose.Schema;
5
6 var PublicationSchema = Schema ({
7   text: String,
8   file: String,
9   created_at: String,
10  user: { type: Schema.ObjectId, ref: 'User' }
11 });
12
13 module.exports = mongoose.model('Publication', PublicationSchema);
```

Follow.js



```
api > models > follow.js > ...
1  'use strict'
2
3  var mongoose = require('mongoose');
4  var Schema = mongoose.Schema;
5
6  var FollowSchema = Schema ({
7      user: { type: Schema.ObjectId, ref: 'User' },
8      followed: { type: Schema.ObjectId, ref: 'User' },
9  });
10
11 module.exports = mongoose.model('Follow', FollowSchema);
```

Message.js

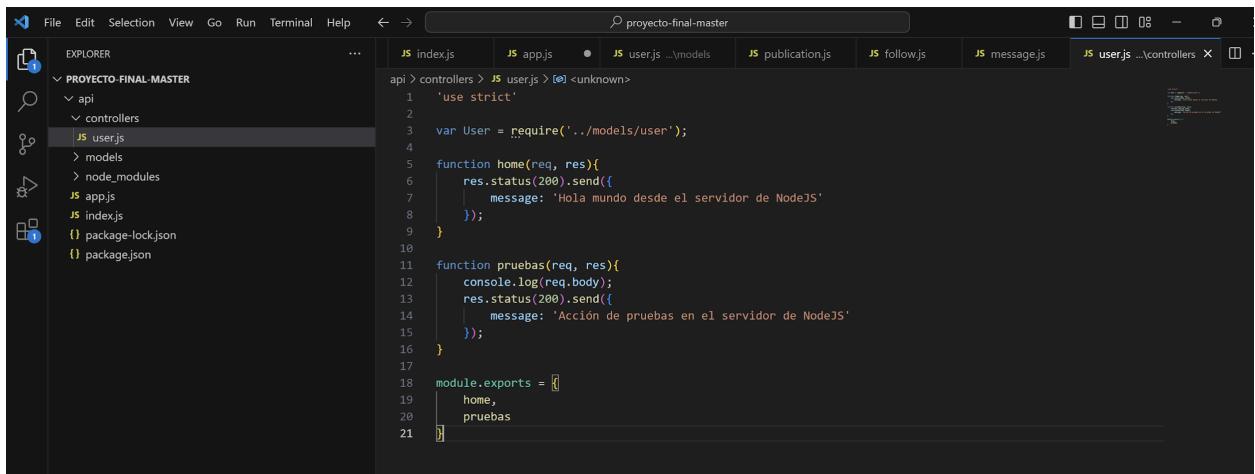


```
api > models > message.js > ...
1  'use strict'
2
3  var mongoose = require('mongoose');
4  var Schema = mongoose.Schema;
5
6  var MessageSchema = Schema ({
7      text: String,
8      created_at: String,
9      emitter: { type: Schema.ObjectId, ref:'User' },
10     receiver: { type: Schema.ObjectId, ref:'User' },
11 });
12
13 module.exports = mongoose.model('Message', MessageSchema);
```

4. Usuarios, login y registro

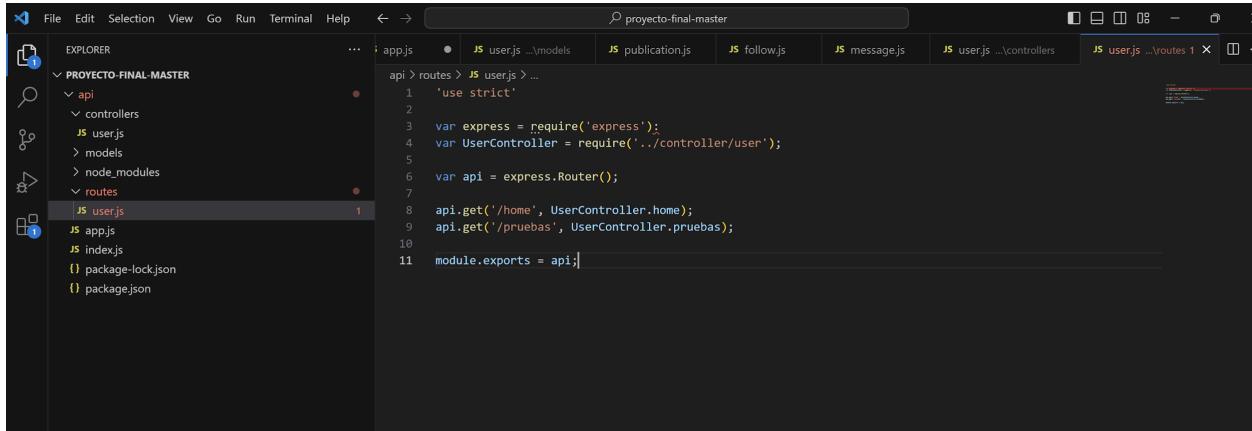
4.1. Controlador de usuarios

Lo primero de todo sería crear una nueva carpeta llamada ‘controllers’ y dentro de ella un archivo llamado ‘user.js’, con el siguiente código:



```
api > controllers > user.js > [o] <unknown>
1  'use strict'
2
3  var User = require('../models/user');
4
5  function home(req, res){
6      res.status(200).send({
7          message: 'Hola mundo desde el servidor de NodeJS'
8      });
9  }
10
11 function pruebas(req, res){
12     console.log(req.body);
13     res.status(200).send({
14         message: 'Acción de pruebas en el servidor de NodeJS'
15     });
16 }
17
18 module.exports = [
19     home,
20     pruebas
21 ];
```

El paso siguiente sería crear una carpeta llamada ‘routes’ para exportar las rutas de home y pruebas creadas en el archivo anterior, y va a tener dentro el archivo ‘user.js’ (el controlador de rutas del archivo ‘user.js’)

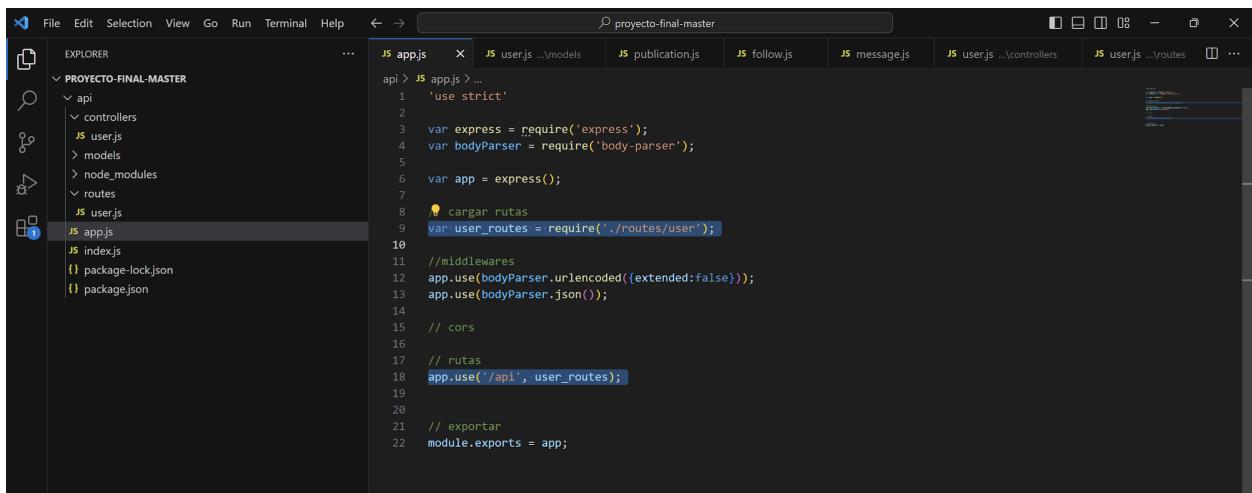


The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure: PROYECTO-FINAL-MASTER > api > controllers > models > node_modules > routes > user.js.
- Code Editor:** The active file is user.js, which contains the following code:

```
api > routes > JS user.js > ...
1  'use strict'
2
3  var express = require('express');
4  var UserController = require('../controller/user');
5
6  var api = express.Router();
7
8  api.get('/home', UserController.home);
9  api.get('/pruebas', UserController.pruebas);
10
11 module.exports = api;
```
- Tab Bar:** Other tabs visible include app.js, publication.js, follow.js, message.js, and user.js ...controllers.

Adicionalmente, en el archivo ‘app.js’ se implementará el código para cargar rutas y el de rutas para el usuario (que nos permite hacer middleware, ya que en cada petición que se haga, el middleware siempre se va a ejecutar antes de llegar al controlador)



The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure: PROYECTO-FINAL-MASTER > api > controllers > models > node_modules > routes > user.js > app.js.
- Code Editor:** The active file is app.js, which contains the following code:

```
api > JS app.js > ...
1  'use strict'
2
3  var express = require('express');
4  var bodyParser = require('body-parser');
5
6  var app = express();
7
8  // cargar rutas
9  var user_routes = require('../routes/user');
10
11 // middlewares
12 app.use(bodyParser.urlencoded({extended:false}));
13 app.use(bodyParser.json());
14
15 // cors
16
17 // rutas
18 app.use('/api', user_routes);
19
20
21 // exportar
22 module.exports = app;
```
- Tab Bar:** Other tabs visible include user.js ...models, publication.js, follow.js, message.js, and user.js ...controllers.

Potencialmente, se hará una prueba desde Postman con el método GET, para saber que el controlador y la ruta están funcionando correctamente:

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:3800/api/pruebas`. The response body is:

```
1 "message": "Acción de pruebas en el servidor de NodeJS"
```

Al irnos a la ruta de home, igual tendremos el body desde Postman al hacer el GET:

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:3800/api/home`. The response body is:

```
1 "message": "Hola mundo desde el servidor de NodeJS"
```

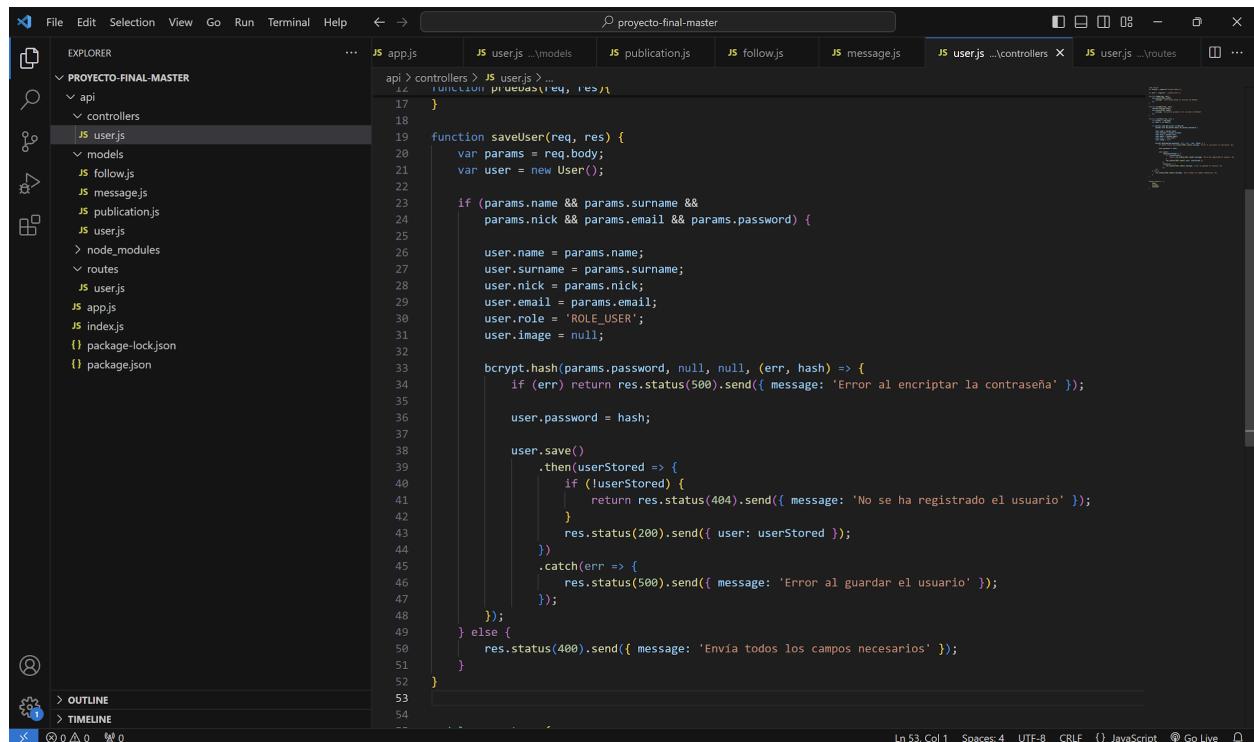
4.2. Registro de usuarios en el backend

En controlador de Usuario, se va a crear un método nuevo que se va a llamar saveUser. Este método llamado saveUser se encarga de procesar la solicitud de registro de un nuevo usuario. Aquí está el detalle de lo que hace:

- Obtención de parámetros: Recibe la solicitud (req) y extrae los parámetros del cuerpo de la solicitud (req.body). Estos parámetros incluyen el nombre, apellido, apodo, correo electrónico y contraseña del nuevo usuario.
- Validación de parámetros: Verifica que todos los campos necesarios (nombre, apellido, apodo, correo electrónico y contraseña) estén presentes en los parámetros recibidos. Si falta alguno, devuelve un mensaje de error indicando que se deben enviar todos los campos necesarios.
- Creación de un nuevo usuario: Si todos los campos están presentes, crea un nuevo objeto de usuario utilizando el modelo User.
- Encriptación de la contraseña: Utiliza la función bcrypt.hash() para encriptar la contraseña proporcionada por el usuario. Si hay un error durante la encriptación, devuelve un mensaje de error indicando que hubo un problema al encriptar la contraseña.
- Guardado del usuario en la base de datos: Una vez que la contraseña está encriptada, llama al método save() del objeto de usuario para guardarlo en la base de datos. Utiliza una promesa para manejar el resultado de la operación de guardado.
- Manejo de la respuesta del guardado: Si el usuario se guarda correctamente, devuelve una respuesta con el código de estado 200 y el usuario guardado en el cuerpo de la respuesta. Si no se puede guardar el usuario, devuelve un mensaje de error con el código de estado 500.
- Manejo de errores: Si hay algún error durante el proceso de guardado del usuario, se maneja utilizando el método catch() de la promesa. En caso de error, devuelve un mensaje de error con el código de estado 500.

- Manejo de falta de usuario: Si por alguna razón el usuario no se guarda correctamente, devuelve un mensaje de error indicando que no se ha registrado el usuario con el código de estado 404.

El código implementado, quedaría de la siguiente manera:

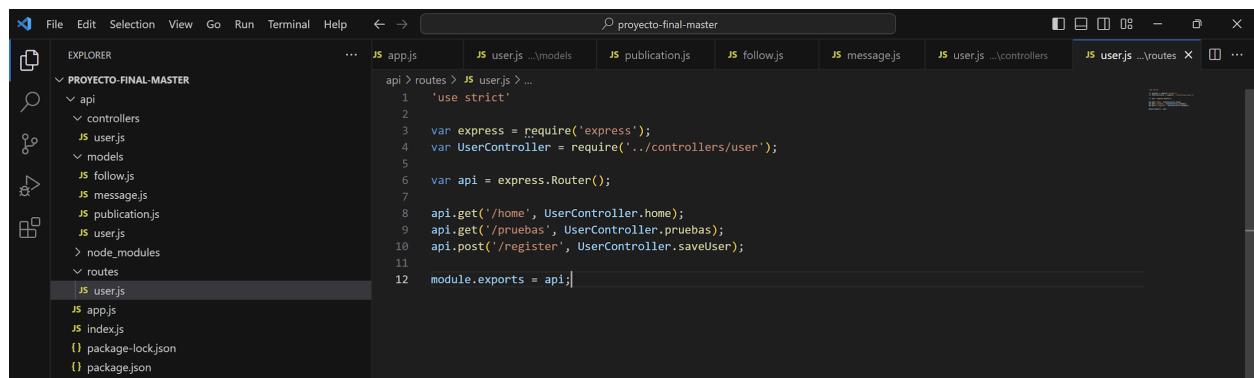


```

File Edit Selection View Go Run Terminal Help < > proyector-final-master
EXPLORER PROYECTO-FINAL-MASTER
api controllers
  JS user.js
models
  JS follow.js
  JS message.js
  JS publication.js
  JS user.js
routes
  JS user.js
JS app.js
JS index.js
package-lock.json
package.json
JS app.js JS user.js ...models JS publication.js JS follow.js JS message.js JS user.js ...controllers JS user.js ...routes ...
17 }
18
19 function saveUser(req, res) {
20   var params = req.body;
21   var user = new User();
22
23   if (params.name && params.surname &&
24     params.nick && params.email && params.password) {
25
26     user.name = params.name;
27     user.surname = params.surname;
28     user.nick = params.nick;
29     user.email = params.email;
30     user.role = 'ROLE_USER';
31     user.image = null;
32
33     bcrypt.hash(params.password, null, null, (err, hash) => {
34       if (err) return res.status(500).send({ message: 'Error al encriptar la contraseña' });
35
36       user.password = hash;
37
38       user.save()
39         .then(userStored => {
40           if (!userStored) {
41             return res.status(404).send({ message: 'No se ha registrado el usuario' });
42           }
43           res.status(200).send({ user: userStored });
44         })
45         .catch(err => {
46           res.status(500).send({ message: 'Error al guardar el usuario' });
47         });
48     });
49   } else {
50     res.status(400).send({ message: 'Envía todos los campos necesarios' });
51   }
52 }
53
54
Ln 53, Col 1 Spaces: 4 UTF-8 CRLF {} JavaScript ⚡ Go Live ⚡

```

Paralelamente, en la ruta de 'user.js' se deberá especificar la ruta de la que se llamará desde Postman



```

File Edit Selection View Go Run Terminal Help < > proyector-final-master
EXPLORER PROYECTO-FINAL-MASTER
api routes
  JS user.js
models
  JS follow.js
  JS message.js
  JS publication.js
routes
  JS user.js
JS app.js
JS index.js
package-lock.json
package.json
JS app.js JS user.js ...models JS publication.js JS follow.js JS message.js JS user.js ...controllers JS user.js ...routes ...
1 'use strict'
2
3 var express = require('express');
4 var UserController = require('../controllers/user');
5
6 var api = express.Router();
7
8 api.get('/home', UserController.home);
9 api.get('/pruebas', UserController.pruebas);
10 api.post('/register', UserController.saveUser);
11
12 module.exports = api;
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
305
306
307
307
308
309
309
310
311
311
312
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
137
```

HTTP <http://localhost:3800/api/register> Save

POST http://localhost:3800/api/register Send

Params Authorization Headers (8) Body **Body** Pre-request Script Tests Settings Cookies

None form-data x-www-form-urlencoded raw binary

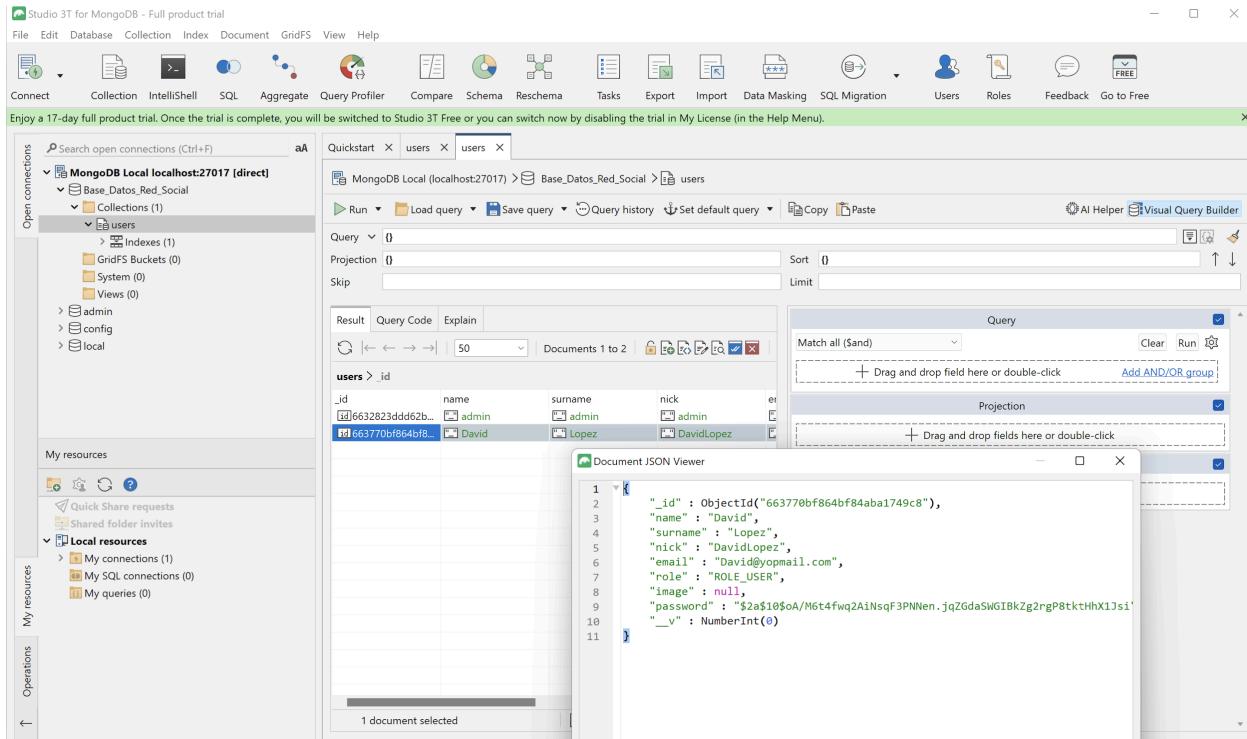
Key	Value
name	David
surname	Lopez
nick	DavidLopez
email	David@yopmail.com
password	david

Body Cookies Headers (7) Test Results Status: 200 OK Time: 1085 ms Size: 474 B Save Response

Pretty Raw Preview Visualize JSON

```
1 "user": {  
2     "_id": "663770bf864bf84aba1749c8",  
3     "name": "David",  
4     "surname": "Lopez",  
5     "nick": "DavidLopez",  
6     "email": "David@yopmail.com",  
7     "role": "ROLE_USER",  
8     "image": null,  
9     "password": "$2a$10$oA/M6t4fwq2AiNsqF3PNNen.jqZGdaSWGIBkZg2rgP8tktHhX1Jsi",  
10    "__v": 0  
11 }  
12 }
```

Entonces, en la base de datos, ese usuario se habrá registrado correctamente:

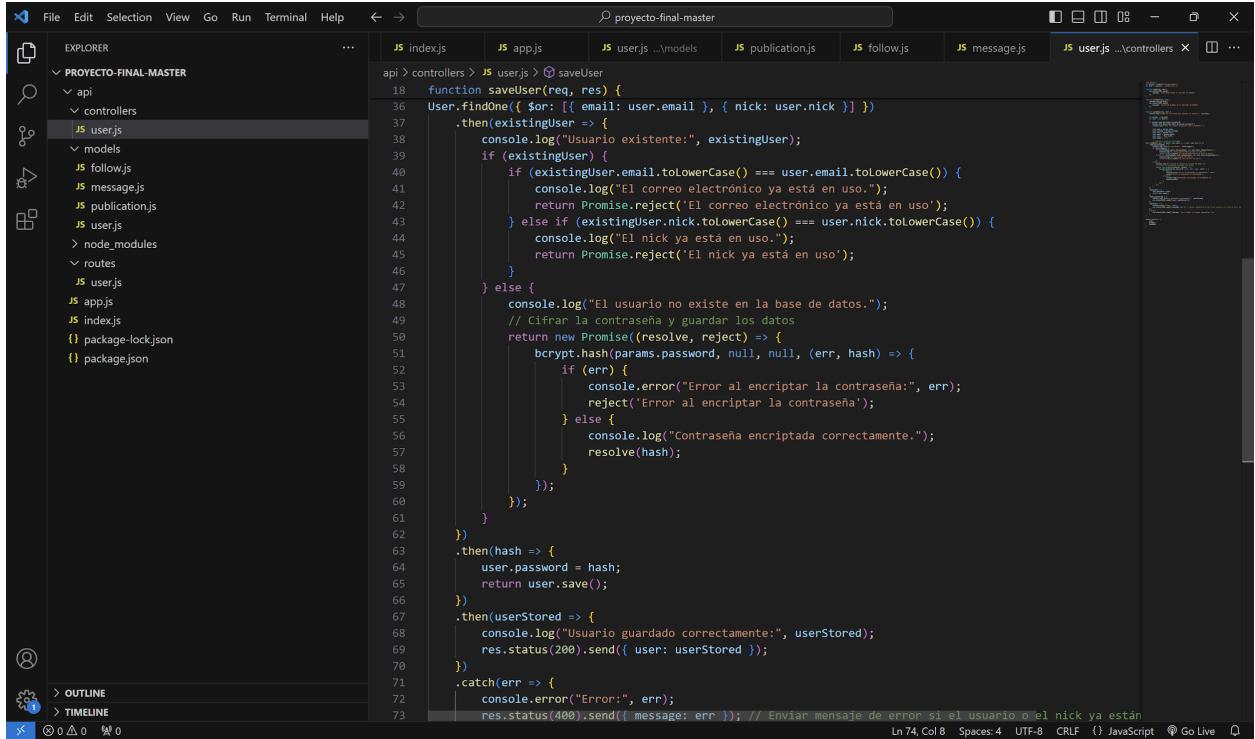


4.3. Control de usuarios duplicados

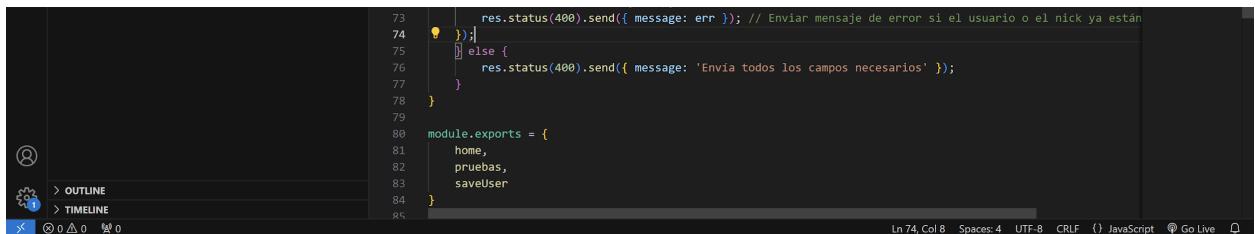
Se va a realizar la validación que cuando un email o un nick ya existan en la base de datos, no se permita introducir un usuario duplicado con el mismo valor de los campos anteriores.

Introduciremos nuevo código en el archivo ‘user.js’ para controlar usuarios duplicados: Antes de guardar un nuevo usuario, busca en la base de datos si ya existe algún usuario con el mismo correo electrónico o el mismo apodo (nick). Si encuentra usuarios que coinciden con estos criterios, devuelve un mensaje indicando que el usuario ya existe. Esto ayuda a evitar registros duplicados en la base de datos y garantiza la unicidad de los usuarios.

Cifrar la contraseña y guardar los datos del usuario: Si no se encuentran usuarios duplicados y se proporcionan todos los campos necesarios, la contraseña del usuario se cifra utilizando el algoritmo bcrypt para mayor seguridad. Luego, los datos del usuario (incluida la contraseña cifrada) se guardan en la base de datos. Si el usuario se guarda correctamente, se devuelve un mensaje de éxito junto con los datos del usuario almacenado. Si ocurre algún error durante el proceso de guardado, se devuelve un mensaje de error correspondiente.



```
api > controllers > JS user.js > saveUser
18  function saveUser(req, res) {
36    User.findOne({ $or: [{ email: user.email }, { nick: user.nick }] })
37      .then(existingUser => {
38        console.log("Usuario existente:", existingUser);
39        if (existingUser) {
40          if (existingUser.email.toLowerCase() === user.email.toLowerCase()) {
41            console.log("El correo electrónico ya está en uso.");
42            return Promise.reject('El correo electrónico ya está en uso');
43          } else if (existingUser.nick.toLowerCase() === user.nick.toLowerCase()) {
44            console.log("El nick ya está en uso.");
45            return Promise.reject('El nick ya está en uso');
46          }
47        } else {
48          console.log("El usuario no existe en la base de datos.");
49          // Cifrar la contraseña y guardar los datos
50          return new Promise((resolve, reject) => {
51            bcrypt.hash(params.password, null, null, (err, hash) => {
52              if (err) {
53                console.error("Error al encriptar la contraseña:", err);
54                reject('Error al encriptar la contraseña');
55              } else {
56                console.log("Contraseña encriptada correctamente.");
57                resolve(hash);
58              }
59            });
60          });
61        }
62      })
63      .then(hash => {
64        user.password = hash;
65        return user.save();
66      })
67      .then(userStored => {
68        console.log("Usuario guardado correctamente:", userStored);
69        res.status(200).send({ user: userStored });
70      })
71      .catch(err => {
72        console.error("Error:", err);
73        res.status(400).send({ message: err }); // Enviar mensaje de error si el usuario o el nick ya están
74      })
75    } else {
76      res.status(400).send({ message: 'Envía todos los campos necesarios' });
77    }
78  }
79
80  module.exports = {
81    home,
82    pruebas,
83    saveUser
84 }
```



```
73    res.status(400).send({ message: err }); // Enviar mensaje de error si el usuario o el nick ya están
74  } else {
75    res.status(400).send({ message: 'Envía todos los campos necesarios' });
76  }
77
78 }
79
80 module.exports = {
81   home,
82   pruebas,
83   saveUser
84 }
```

Hecho esto, al hacer una llamada POST desde Postman para registrar un nuevo usuario introduciendo un nick o email existentes en la base de datos, se reciben los errores de validación:

POST http://localhost:3800/api/register

HTTP POST http://localhost:3800/api/register

Body

Key	Value
name	David
surname	Lopez
nick	DavidLopez
email	David@yopmail.com
password	david

Status: 400 Bad Request Time: 95 ms Size: 281 B Save Response

```

1 "message": "El nick ya está en uso"
2
3

```

POST http://localhost:3800/api/register

HTTP POST http://localhost:3800/api/register

Body

Key	Value
name	David
surname	Lopez
nick	DavidLopes
email	David@yopmail.com
password	david

Status: 400 Bad Request Time: 307 ms Size: 296 B Save Response

```

1 "message": "El correo electrónico ya está en uso"
2
3

```

4.4. Método de login

Para loguear los usuarios en nuestro backend, es necesario crear un método nuevo.

Este método `loginUser` se encarga de autenticar a un usuario en el sistema. Aquí está un resumen de lo que hace:

- Obtiene los parámetros de la solicitud del cuerpo (`req.body`), que deberían incluir el correo electrónico (`email`) y la contraseña (`password`) proporcionados por el usuario durante el intento de inicio de sesión.
- Utiliza el correo electrónico proporcionado para buscar un usuario en la base de datos utilizando `User.findOne({ email: email })`. Si no se encuentra ningún usuario con el correo electrónico proporcionado, devuelve un mensaje de error indicando que el usuario no se ha podido identificar.
- Si se encuentra un usuario con el correo electrónico proporcionado, compara la contraseña proporcionada con la contraseña almacenada en la base de datos utilizando `bcrypt.compare(password, user.password)`. Si las contraseñas no coinciden, devuelve un mensaje de error indicando que el usuario no se ha podido identificar.
- Si las contraseñas coinciden, lo que significa que el usuario ha proporcionado credenciales válidas, devuelve los datos del usuario al cliente en la respuesta (`res.status(200).send({ user })`).
- Si ocurre algún error durante el proceso de autenticación, como un error de base de datos o un error al comparar contraseñas, maneja el error devolviendo un mensaje de error genérico indicando que hubo un error en la petición (`res.status(500).send({ message: 'Error en la petición' })`).

A continuación, se expone lo que se ha implementado en el archivo ‘`user.js`’

The screenshot shows the VS Code interface with the file `user.js` open. The code is a Node.js module for a login function. It includes validation for required fields, checks if the user exists, compares the password using bcrypt, and returns a 404 error if no user is found. The code is well-structured with comments explaining each step.

```

api > controllers > JS user.js > loginUser
18   function saveUser(req, res) {
19     // ...
20     res.status(400).send({ message: 'Envía todos los campos necesarios' });
21   }
22
23   function loginUser(req, res) {
24     var params = req.body;
25
26     var email = params.email;
27     var password = params.password;
28
29     User.findOne({ email: email })
30       .then(user => {
31         if (!user) {
32           return res.status(404).send({ message: 'El usuario no se ha podido identificar!' });
33         }
34
35         return bcrypt.compare(password, user.password);
36       })
37       .then(check => {
38         if (!check) {
39           return res.status(404).send({ message: 'El usuario no se ha podido identificar' });
40         }
41
42         // Devolver datos de usuario
43         res.status(200).send({ user });
44       })
45       .catch(err => {
46         console.error("Error:", err);
47         res.status(500).send({ message: 'Error en la petición' });
48       });
49
50   }
51
52   module.exports = {
53     home,
54     pruebas,
55     saveUser,
56     loginUser
57   };
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112

```

Entonces, al intentar loguear un usuario que no existe desde Postman, recibiremos la siguiente validación:

The screenshot shows a Postman request to a login endpoint. The body contains a JSON object with `email` and `password` fields. Both fields are checked. The response status is 404 Not Found, and the response body is a JSON object with a `message` field containing the string "El usuario no se ha podido identificar".

Key	Value	Description
email	david4@david.com	
password	david	

body 404 Not Found 8 ms 294 B Save as example

Pretty Raw Preview Visualize JSON

```

1  [
2    "message": "El usuario no se ha podido identificar"
3 ]

```

Pero por lo contrario si colamos el email correcto y las password, si nos logea el usuario:

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** http://localhost:3800/api/login
- Body:** x-www-form-urlencoded
- Body Content:**

Key	Value	Description
email	david@david.com	
password	david	
- Response Status:** 200 OK
- Response Time:** 170 ms
- Response Size:** 398 B
- Response Preview:**

```
1 {  
2   "user": {  
3     "_id": "6638f8129de7c02e2271a6b8",  
4     "name": "David",  
5     "surname": "Lopez",  
6     "nick": "davidlopez",  
7     "email": "david@david.com",  
8     "role": "ROLE_USER",  
9     "image": null,  
10    "__v": 0  
11  }  
12 }
```

4.5. Servicio de login y JWT

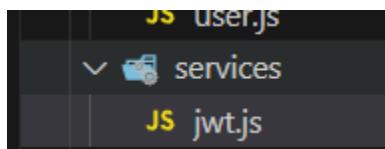
Ahora vamos a empezar a trabajar con JWT y a la hora de devolver los datos de usuario vamos a poder tener la opción, poder devolver o bien los datos del usuario o bien un token dentro del cual, van a ir todos los datos del usuario pero codificados o encriptados dentro de un hash.

Entonces después de donde hemos colocado lo de devolver datos de usuarios, vamos hacer un if, para ver si los parámetros recibidos los recibimos por un POST para poder devolver el token o no.

En el caso que yo necesite devolver el token, haremos el primer if con los params.gettoken para ver si llega un true y se devuelve y si no, pues devolveremos los datos del usuario.

Por tanto, en el caso de que el usuario nos pida el token, por eso, vamos hacer un servicio que no es más que una clase que consta de una serie de métodos dentro de ella, el cual nos permite generar un token para poder utilizarlo dentro del controlador.

Creamos dentro la carpeta raíz Api, una carpeta nueva llamada services y a su vez dentro de ella un archivo llamado jwt.js.



Ahora vamos a crear el método, que nos va a generar el token.

Lo primero de todo creamos 2 variables una jwt para el token y otra de moment para poder generar fechas.

Ahora vamos a utilizar el exports y el nombre de la función, donde recibiremos un objeto de usuario, en este caso le vamos a pasar varios datos. como el name, surname,email,role, etc. Además haremos uso del parámetro iat y exp.

iat: lo que nos permite es poner la fecha de creación del token, utilizando moment y unix.

exp: será la fecha de expedición del token, pero le daremos 30 días.

Haremos un return para poder cifrar el token con un clave secreta que solo sabremos nosotros el desarrollador.

Por tanto, quedaría de la siguiente manera:

```
api > services > JS jwt.js > ...
💡 Click here to ask Blackbox to help you code faster
1  'use strict';
2
3  let jwt = require('jwt-simple');  2.1k (gzipped: 1k)
4  let moment = require('moment');  61.5k (gzipped: 19.8k)
5  let secret = 'clave_secreta_curso_desarrollar_red_social_angular';
6
7  exports.createToken = function(user) {
8      let payload = {
9          sub: user._id,
10         name: user.name,
11         surname: user.surname,
12         nick: user.nick,
13         email: user.email,
14         role: user.role,
15         image: user.image,
16         iat: moment().unix(),
17         exp: moment().add(30, 'days').unix()
18     };
19
20     // Encode the payload into a JWT using jwt-simple and return it
21     return jwt.encode(payload, secret);
22 };
23
```

Por eso, una vez que nosotros le pasemos la clave, vamos a poder generar el token.

Es decir, le vamos a pasar los datos el objeto y la clave secreta en base a eso, el método que vemos de encode lo codifica todo y genera un hash, genera un token enorme que es el que vamos a utilizar.

Ahora vamos a utilizar el servicio JWT dentro de nuestro controlador para devolver y generar el token.

Vamos a llamar al servicio arriba del todo primero dentro del controlador de usuario:

```
const jwt = require('../services/jwt');
```

Y después haremos uso del servicio JWT más abajo:

```
// Devolver datos de usuario
if(params.gettoken){
    // generar y devolver token
    return res.status(200).send({
        token: jwt.createToken(user)
    });
}
```

Haremos una pequeña prueba con postman para que nos devuelva realmente el token, por tanto, si le pasamos los datos correctos, vemos que sigue funcionando:

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** http://localhost:3800/api/login
- Body (x-www-form-urlencoded):**

<input type="checkbox"/>	surname	Lopez
<input type="text"/>	nick	davidlopez
<input checked="" type="checkbox"/>	email	david@david.com
<input checked="" type="checkbox"/>	password	david
- Response Headers:** 200 OK, 147 ms, 398 B
- Response Body (Pretty):**

```

1 {
2     "user": {
3         "_id": "6638f8129de7c02e2271a6b8",
4         "name": "David",
5         "surname": "Lopez",
6         "nick": "davidlopez",
7         "email": "david@david.com",
8         "role": "ROLE_USER",
9         "image": null,
10        "__v": 0
11    }
12 }
```

Si ahora le enviamos nuestro servicio de gettoken y le pasamos a true, veremos como nos envía el token encriptado, dentro del cual está clave secreta, los datos de usuario, la fecha en cual se ha creado el token, etc. Todos estos datos están encriptados

dentro de este token y después podríamos hacer un decode y conseguir esa información también.

The screenshot shows a Postman interface. At the top, it says "POST" and "http://localhost:3800/api/login". Below that, there are tabs for "Params", "Auth", "Headers (9)", "Body", "Pre-req.", "Tests", and "Settings". The "Body" tab is selected and has a dropdown "x-www-form-urlencoded". Under "Body", there are two rows: "password" with value "david" and "gettoken" with value "true". Below the table, it says "Key" and "Value". At the bottom, it shows "Body" with "Pretty", "Raw", "Preview", "Visualize", "JSON", and other options. The JSON response is:

```
1 {  
2   "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.  
eyJzdWIiOiI2NjM4ZjgxMjlkZTdjMDJlMjI3MWE2YjgiLCJuYW1I  
joiRGF2aWQiLCJzdXJuYW1IjoiTG9wZXoiLCJuaWNrIjoiZGF2aW  
Rsb3BleiIsImVtYWlsIjoiZGF2aWRAZGF2aWQuY29tIiwicm9sZSI  
6I1JPTEVfVVNFUiIsImItYWdlIjpudWxsLCJpYXQiOjE3MTUwMTkx  
MDYsImV4cCI6MTcxNzYxMTEwNn0.  
_Q7MY8ej2K5rfaOfWWhwMcurds_0HUD0-qoK3urNQPs"  
3 }
```

4.6. Middleware de autenticación

En este apartado vamos a crear un middleware que compruebe el token que nosotros le estamos pasando en cada una de las peticiones. Porque cuando un cliente se conecta a nuestra api en la mayoría de métodos va a tener que enviar un token, el cual es el token de autenticación el cual es este:

```
return res.status(200).send({  
  token: jwt.createToken(user)
```

Por lo tanto, lo que va a hacer el cliente será enviar ese token y entonces nosotros vamos a tener que detectar automáticamente si ese token es correcto o no, para dejarle pasar el método o no dejarle pasar al método de la api que está solicitando.

Por ello vamos hacer un middleware este es método que se va a ejecutar antes de ejecutar la acción del controlador que nosotros vayamos a ejecutar, este hará una pequeña lógica, es decir, va a comprobar algo, en este caso si el token es correcto o no. Si es correcto nos dejará entrar al método de la api que estamos solicitando y en el caso de que no fuera correcto lo que va hacer es negarlo y nos denegará el acceso, este enviará una respuesta diciendo que el token no es válido y que no estamos autenticados correctamente.

Vamos a crear como siempre una carpeta nueva dentro de nuestra carpeta raíz(api), llamada middlewares y un archivo nuevo dentro de ella llamada authenticated.js:



Ahora cogeremos y crearemos las 3 variables anteriores la de jwt,moment y la clave secreta.

Después cogeremos y haremos un exports. y llamaremos como método de middleware así(ensureAuth) y dentro de ella vamos a definir una función anónima dentro de este método, que esta recoge varias cosas (request, response y un next)

- request: Estos serán los datos que recibimos en la petición.
- response: Esta será la respuesta que nosotros estaremos escupiendo.
- next: Es la funcionalidad que nos permite saltar a otra cosa. Es decir, cuando nosotros entremos en middleware, hasta que nosotros no lancemos el método next el programa nodeJS no va a salir del middleware. hasta que no ejecutamos este next, para que pueda salir e ir a ruta que estamos ejecutando.

Lo primero de todo es que el token nos va a llegar en una cabecera por tanto, vamos a hacer una comprobación con un if, además, le vamos aadir los mensajes:

- (mesage(401): para decirle que el token ha expirado
- mesage(404): para decirle que el token es válido

Por tanto, dejaríamos el archivo así:

```

1  Click here to ask Blackbox to help you code faster
2
3  'use strict';
4
5  let jwt = require('jwt-simple');  2.1k (gzipped: 1k)
6  let moment = require('moment');  61.5k (gzipped: 19.8k)
7  let secret = 'clave_secreta_curso_desarrollar_red_social.angular';
8
9  exports.ensureAuth = function(req, res, next) {
10    if (!req.headers.authorization) {
11      return res.status(403).send({ message: 'La petición no tiene la cabecera de autenticación' });
12    }
13
14    let token = req.headers.authorization.replace(/["]+/g, '');
15
16    try {
17      let payload = jwt.decode(token, secret);
18      if (payload.exp <= moment().unix()) {
19        return res.status(401).send({ message: 'El token ha expirado' });
20      }
21
22      req.user = payload;
23      next();
24    } catch(ex) {
25      return res.status(404).send({ message: 'El token no es válido' });
26    }
}

```

Ahora deberíamos irnos a nuestro archivo de rutas y crear este método:

```

let md_auth = require('../middlewares/authenticated');

```

Y abajo vamos a decirle la ruta donde se sitúa este método, y cogeremos la ruta de pruebas y le diremos que recoja también esta variable md_auth.ensureAuth, de esta manera el segundo parámetro es el middleware y esto ya la coge de manera dinámica, por lo tanto, va a pasar por el middleware va a comprobar que el token es correcto, pasará al siguiente método correlativo.

```

7  let api = express.Router();
8  let md_auth = require('../middlewares/authenticated');
9
10 api.get('/home',UserController.home);
11 api.get('/pruebas',md_auth.ensureAuth,UserController.pruebas);
12 api.post('/register',UserController.saveUser);
13 api.post('/login',UserController.loginUser);
14

```

Ahora haremos una comprobación con Postman para ver si funciona o no. De tal manera, que le pasaremos si en token:

GET http://localhost:3800/api/pruebas

Params Auth Headers (10) Body ● Pre-req. Tests Settings

Headers 9 hidden

	Key	Value	...	Bulk Edit	Presets
<input type="checkbox"/>	Authorization	eyJ0eXAiOiJKV1QiLCJ...			
	Key	Value	Description		

Body 403 Forbidden 17 ms 307 B

Pretty Raw Preview Visualize

```
1 {  
2   "message": "La petición no tiene la cabecera de  
3     autenticación"  
4 }
```

Nos situamos en la petición en el headers y le colamos el authorization y le colocamos un token malo y nos arroja esto:

GET http://localhost:3800/api/pruebas

Params Auth Headers (8) Body Scripts Tests Settings

Headers 7 hidden

	Key	Value	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Authorization	dsafasdf			
	Key	Value	Description		

Ahora cogeremos el token que nos enviamos en la petición anterior y se lo pondremos para ver cómo se actúa y nos devuelve el método que buscamos:

The screenshot shows the Postman interface. At the top, there is a search bar and a dropdown menu. Below that, a 'Send' button is highlighted in blue. Underneath the search bar, there are tabs for 'Params', 'Auth', 'Headers (10)', 'Body', 'Pre-req.', 'Tests', 'Settings', and '...'. The 'Headers (10)' tab is selected. A table below shows one header entry: 'Authorization' with value 'eyJ0eXAiOiJKV1QiLCJ...'. There are also columns for 'Key', 'Value', and 'Description'. At the bottom of the interface, there is a status bar showing 'ody' (status), '404 Not Found' (status code), '5 ms' (time taken), '278 B' (body size), and a 'Save as example' button.

```
1 {  
2   "message": "El token no es válido"  
3 }
```

4.7. Ruta para devolver los datos del usuario

Vamos a crear una ruta que nos va a permitir actualizar los datos del usuario y también vamos a crear otra ruta que nos permita sacar los datos de un usuario, por ejemplo:

solicitar la información o los datos del perfil del usuario con id,15. Por tanto, vamos hacer estas 2 URL.

Vamos a crear un método el cual nos permite listar y devolvernos los datos de usuario. Vamos a tener posteriormente ese método el de getUser la URL que nos permita listar o devolver los datos y después tendremos un método para que nos devuelva todos los usuarios que hay guardados en la base de datos, pero este listado ya irá paginado, irá también con la información del seguimiento de esos usuarios, es decir, que nosotros mediante el token que le pasemos y la petición al método de la api, será capaz de

listar todos los usuarios y detectar si nosotros como usuario identificado seguimos o no seguimos a ese usuario con lo cual será muy interesante.

Pero de momento nos vamos a conformar con hacer un método que nos devuelva los datos de un usuario en concreto, por ello ,vamos a crear el método para conseguir los datos de un usuario.

Creamos la un función llamada getUser y le vamos a pasar un request y una response como mencionamos anteriormente para que iba destinada cada una. Además tenemos que recoger en ella el ID del usuario, este ID nos va llegar por la URL, para ello creamos una variable llamada userId y le pasaremos el parámetro que nos llegara por la URL.

Importante:

- Cuando nos llegan datos por la URL se utiliza el **params**.
- Cuando llegan datos por POST o por PUT utilizamos **body**.

Después cogeremos el método User y utilizaremos el método findById para buscar un documento por el ID. Entonces, le pasamos como primer parámetro el userID y como segundo parámetro tendremos un callback en cual vamos a tener 2 parámetros, un posible error y el usuario que nos va a devolver la consulta a la base de datos.

Para ello haremos una comprobación si tenemos un error o no. Dando uso de las respuestas con los mensajes 500 y 404.

- Err (500) nos listará un mensaje de error en la petición.
- Err(404) el usuario no existe.

Y por último utilizaremos el status 200 devolviendo el usuario.

Por lo tanto, quedaría de la siguiente manera:

```
//Conseguir datos de un usuario

function getUser(req,res){
    let userId = req.params.id;

    User.findById(userId,(err, user) => {
        if(err) return res.status(500).send({message: 'Error en la petición'});

        if(!user) return res.status(404).send({message: 'El usuario no existe'});

        return res.status(200).send({user});
    });
}
```

Finalmente tenemos que devolverlo en el module.exports para poder crear una ruta y poder utilizar este método fuera de este fichero.

Abajo del todo en module.exports lo metemos:

```
// Exportar Métodos
module.exports = {
    home,
    pruebas,
    saveUser,
    loginUser,
    getUser
};
```

Ahora nos vamos a nuestro fichero de rutas y dentro de él nos creamos un nuevo método, además irá protegido por autenticación. Será un método por GET y el pat, es decir, la URL será /user/:id(Aquí le estamos pasando un id por la URL). Así quedaría:

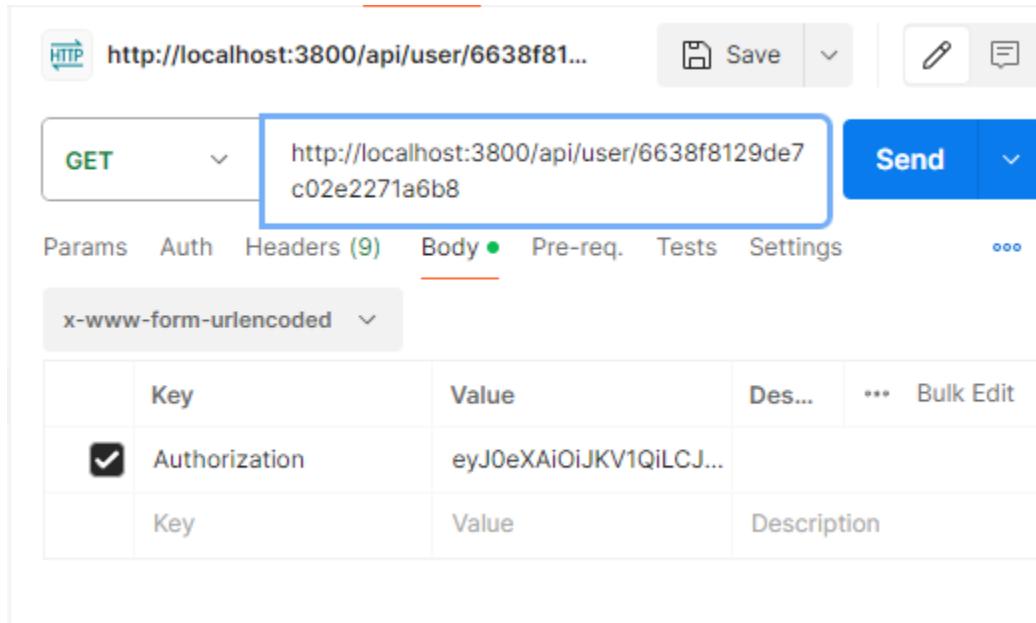
```
api.get('/user/:id',md_auth.ensureAuth, UserController.getUser);
```

Para comprobarlo, utilizamos postman, para ello tendremos que pasarle como parámetro un id registrado en la base de datos, por tanto, nos iremos a robomongo y cogeremos un id de los que hemos creado en este caso sería este:



```
1      "_id" : ObjectId("6638f8129de7c02e2271a6b8"),
2      "name" : "David",
3      "surname" : "Lopez",
4      "nick" : "davidlopez",
5      "email" : "david@david.com",
6      "role" : "ROLE_USER",
7      "image" : null,
8      "password" : "$2a$10$ZsQzw40lEV0GOwX/1.62.bqMbHgRucak1hiPMCdmaD1AE4TwZdP.",
9      "__v" : NumberInt(0)
10
11 }
```

Se enviará en la URL como parámetro y te devuelve los datos del usuario:



The screenshot shows a Postman interface with the following details:

- Method:** GET
- URL:** `http://localhost:3800/api/user/6638f8129de7c02e2271a6b8`
- Headers:** `x-www-form-urlencoded`
- Body:** `Authorization: eyJ0eXAiOiJKV1QiLCJ...`

4.8. Usuarios paginados

Ahora vamos a hacer un método para nuestra API que nos permite listar todos los usuarios que tenemos en nuestra plataforma, este listado irá de forma paginada así que utilizaremos un paquete mongoose pagination.

Comenzamos creando una función llamada `getUsers`, esta recibirá por la URL un número de página. entonces, va a paginar el número de usuarios que están en nuestra plataforma. Para ello crearemos una variable llamada `identity_user_id` en esta variable vamos a recoger el id del usuario que está logueado en ese momento, para

conseguirlo si nos acordamos anteriormente hemos brindado una propiedad user a la request con el middleware, entonces, hay tenemos un objeto completo del usuario el cual nos esta mandado el token. El objeto del usuario que se ha decodificado del token que nos envía la petición.

Para ello cogemos el res.user.sub(este sub recoge el id):

```
exports.createToken = function(user) {
  let payload = {
    sub: user.id,
```

Y quedaría así:

```
function getUsers(req,res){
  let identity_user_id = req.user.sub;
```

Ahora tenemos que comprobar que nos llega por la URL la página para ello utilizaremos un response. Aquí tenemos el id del usuario que esta logueado y la página:

```
function getUsers(req,res){
  let identity_user_id = req.user.sub;

  let page = 1;
  if(req.params.page){
    page = req.params.page;
  }
```

Ahora vamos a crear otra variable, la cual recogerá el número de elementos que se mostrará por página.

```
}
```

```
let itemsPerPage = 5;
```

Nos quedaría hacer un find, para listar todos los usuarios de la base de datos, para ello utilizamos el modelo User y el método find, también vamos a ordenador por id

utilizando el sort. Seguido utilizamos el método paginate para paginar esos resultados, vamos a pasarle el número de la página donde estamos actualmente con page, el itemsPerPage, es decir, la cantidad de registros que hay por página y por último una función de callback.

```
User.find().sort('_id').paginate(page, itemsPerPage, (err, users, total) => {
  if(err) return res.status(500).send({message: 'Error en la petición'});
  if(!users) return res.status(404).send({message: 'No hay usuarios disponibles'});

  return res.status(200).send({
    users,
    total,
    pages: Math.ceil(total/itemsPerPage)
  });
});
```

Por último exportamos abajo y nos vamos a nuestra hoja de rutas y creamos la ruta para utilizar este método.

```
api.get('/users/:page?', md_auth.ensureAuth, UserController.getUsers);
```

4.9. Actualizar los datos del usuario.

Ahora vamos a crear un método que nos permita actualizar los datos de un usuario pasándole nueva información.

Lo primero de todo que vamos hacer es crear un función llamada updateUser que recogerá un request y una response. Por tanto, lo primero que tenemos que hacer es recoger por la URL el ID del usuario el cual vamos a actualizar.

Creamos una variable llamada userId, está lo que hará será recoger el ID de la URL un parámetro que vamos a tener por la URL. Después vamos a conseguir todos los datos, actualizar el objeto que nos llega en la request la cual incluye todos los campos con los datos nuevos que vamos actualizar como por ejemplo: el nombre, el nick, el apellido, etc. Para ello tenemos que crear una variable nueva llamada update. Esta recogerá el body de la request, esto será lo que le pasaremos luego al update.

Una cosa importante llegados a este punto, tenemos que eliminar la propiedad password del objeto update porque ese tipo de datos es recomendable tenerlo en un método separado el cual se encargue solamente de la actualización de la contraseña.

Después tendremos que comprobar si el userId que estamos recibiendo por la URL es diferente al userId que tengo en el request, porque este método de actualización del usuario solamente lo vamos a utilizar cuando el propio usuario que se ha creado su cuenta quiera modificarse sus propios datos.

Tenemos que tener en cuenta que userId que recibimos por la URL tiene que ser igual que el id del objeto del usuario identificado.

Por eso haremos varias comprobaciones buscando por Id actualizado y ver si tenemos todo correcto, por si entra o no el usuario con los datos nuevos.

También cuando nosotros hacemos un update dentro de mongoDB y Mongoose este nos devuelve el objeto en el userUpdate. Nos devuelve el objeto anterior que ha actualizado, es decir, me devuelve el objeto original, no me devuelve el nuevo objeto con los datos actualizados. Entonces, para que nos devuelva los nuevos datos podríamos pasarle un tercer parámetro,

Por lo tanto quedaría así:

```
// Edición de datos de usuario
async function updateUser(req,res){
  const userId = req.params.id;
  const update = req.body;

  // Borrar la propiedad password
  delete update.password;

  if(userId != req.user.sub){
    return res.status(500).send({message: 'No tienes permiso para actualizar los datos del usuario'});
  }
  User.findByIdAndUpdate(userId, update, {new:true}, (err,userUpdate)=>{
    if(err) return res.status(500).send({message: 'Error en la petición'});
    if(!userUpdate) return res.status(404).send({message: 'No se ha podido actualizar el usuario'});

    return res.status(200).send({user: userUpdate});
  });
}
```

Ahora exportamos nuestro nuevo método abajo:

```
// Exportar Métodos
module.exports = {
  home,
  pruebas,
  saveUser,
  loginUser,
  getUser,
  getUsers,
  updateUser
};
```

Y nos situamos en nuestra carpeta de rutas, importante, para actualizar un objeto hay que utilizar en este caso por método de PUT, ni GET ni POST. Entonces, quedaría así:

```
api.put('update-user/:id', md_auth.ensureAuth,UserController.updateUser);
```

4.9.1. Subir avatar de usuario

Una vez terminado el método de actualizar los datos del usuario. Ahora vamos a crear un método que nos permita subir una imagen de usuario y luego poder actualizar el objeto con esta nueva imagen y actualizar el campo de imagen que tiene cada uno de los usuarios.

Vamos a crear primero de todo una función llamada uploadImage y le pasamos una request y una response como parámetro. Ahora necesitamos crear otra variable, porque necesitamos el id del usuario para actualizarlo, cabe destacar que solamente puede subir archivos de imagen el propio usuario dueño de su cuenta. Por lo tanto, si solo el userID es exactamente igual al userId que hay en la request podrá estar logueado y poder cambiar su imagen. Pero si no está logueado y el mismo userId no le vamos a permitir actualizar su imagen y datos de usuario.

También destacar que dentro de la request cuando enviamos archivos hay una propiedad files (una array de archivos que estamos intentando subir).

Por eso haremos referencia si estamos enviado algún fichero pues podremos subir el fichero en sí y así poder guardar el fichero en la base de datos.

Por tanto quedaría de la siguiente manera:

```

// Subir archivos de imagen/avatar de usuario
async function uploadImage(req, res) {
  const userId = req.params.id;

  if (req.files) {
    const filePath = req.files.image.path;
    console.log(filePath);

    const fileSplit = filePath.split('\\');
    console.log(fileSplit);

    const fileName = fileSplit[2];
    console.log(fileName);

    const extSplit = fileName.split('.');
    const fileExt = extSplit[1];

    if (userId != req.user.sub) {
      removeFilesOfUploads(res, filePath, 'No tienes permiso para actualizar los datos del usuario');
    }

    if (
      fileExt == 'png' ||
      fileExt == 'jpg' ||
      fileExt == 'jpeg' ||
      fileExt == 'gif'
    ) {
      // Actualizar documento de usuario logueado
      User.findByIdAndUpdate(userId, { image: fileName }, { new: true }, (err, userUpdate) => {
        if (err) return res.status(500).send({ message: 'Error en la petición' });

        if (!userUpdate) return res.status(404).send({ message: 'No se ha podido actualizar' });

        return res.status(200).send({ user: userUpdate });
      });
    } else {
      removeFilesOfUploads(res, filePath, 'Extensión no válida');
    }
  }
}

async function removeFilesOfUploads(res, filePath, message) {
  fs.unlink(filePath, (err) => {
    if (err) {
      return res.status(500).send({ message: 'Error al eliminar el archivo' });
    }
    return res.status(200).send({ message: message });
  });
}

```

The screenshot shows the continuation of the code from the previous snippet. The uploadImage function ends with a closing brace for the else block. Below it, the removeFilesOfUploads function is defined. It uses the fs module's unlink method to delete a file at the specified filePath. If an error occurs during deletion, it returns a 500 status with an error message. Otherwise, it returns a 200 status with the provided message.

```

} else {
  return res.status(200).send({ message: 'No se ha podido subir la imagen' });
}

async function removeFilesOfUploads(res, filePath, message) {
  fs.unlink(filePath, (err) => {
    if (err) {
      return res.status(500).send({ message: 'Error al eliminar el archivo' });
    }
    return res.status(200).send({ message: message });
  });
}

```

Por último igual que todo lo demás, cogemos y exportamos abajo la función:

```
module.exports = {
  home,
  pruebas,
  saveUser,
  loginUser,
  getUser,
  getUsers,
  updateUser,
  uploadImage
};
```

Y creamos en el archivo de rutas a través de un método POST, su ruta:

```
api.post('upload-image-user/:id', [md_auth.ensureAuth, md_upload], UserController.uploadImage);
```

4.9.2. Devolver imagen de usuario

Ahora vamos a crear un método para devolver la imagen de un usuario. Va a ser una URL que irá protegida por autenticación y esa URL nos escupirá la imagen del usuario que hemos solicitado. Además solamente será accesible por usuarios logueados.

Esta nueva función la vamos a llamar getImageFile que recogerá una request y una response. Además dentro de ella creamos una nueva variable (imageFile) y tendrá como valor una resquest y como parámetros imageFile este parámetro lo va a recoger por la URL , entonces ese nombre de fichero que le vamos a pasar por la URL es el que va a recibir el método y va a sacar esa imagen del sistema de ficheros y nos la sacará.

Una vez tengamos las variables que recogen los ficheros de imagen, comprobaremos si este fichero existe.

Quedaría de la siguiente manera, utilizando y comprobando si funciona o no:

```
async function getImageFile(req, res) {
  const imageFile = req.params.imageFile;
  const pathFile = `./uploads/users/${imageFile}`;

  fs.exists(pathFile, (exists) => {
    if (exists) {
      res.sendFile(path.resolve(pathFile));
    } else {
      res.status(200).send({ message: 'No existe la imagen' });
    }
  });
}
```

Lo debemos de exportar abajo en el module.exports:

```
module.exports = {
  home,
  pruebas,
  saveUser,
  loginUser,
  getUser,
  getUsers,
  updateUser,
  uploadImage,
  getImageFile
};
```

Y por último tenemos que crearle la ruta para ello en el archivo de rutes la creamos:

```
api.get('get-image-user/:imageFile', UserController.getImageFile);
```

5. Sistema de seguimiento / follow

5.1. Controlador y rutas de seguimiento

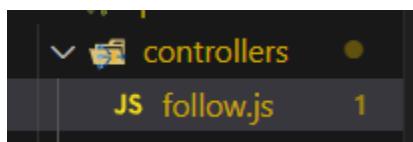
Ya tenemos a grandes rasgos el controlador de usuarios anteriormente explicado.

Ahora vamos a crear el controlador que se va a encargar del sistema de seguimiento.

Este sistema de seguimiento se va a encargar de darle a los usuarios la posibilidad de seguirse y dejar de seguirse entre ellos.

Además nos va a permitir mostrar un listado de los usuarios que nos siguen y los usuarios seguidos y muy parecido a lo que tiene twitter, vamos a construir a nivel de backend este sistema de seguimiento.

Vamos a comenzar creando un nuevo fichero dentro de nuestra carpeta de controller, llamado follow.js.



Dentro de este vamos a crear las variables de los controladores que vamos a utilizar y de los sistemas como mongoosePagination entre otros, además al final del todo lo exportamos.

```
* Click here to ask BlackBox to help you code faster.
1  'use strict'
2
3  let path = require('path');
4  let fs = require('fs');
5  let mongoosePaginate = require('mongoose-pagination', 'mc');
6
7  let User = require('../models/user');
8  let follow = require('../models/follow');
9
10 function prueba(req,res){
11     res.status(200).send({message: 'Hola mundo desde el controlador'});
12 }
13
14 module.exports = {
15     prueba
16 }
17
```

Una vez tengamos este controlador, vamos a crear su ruta, para ello, nos vamos a la carpeta de rutes y creamos un nuevo archivo llamado follow.js, en este vamos a situar la hoja de ruta que vamos a utilizar, en ella estará el controlador, express, la api y el middleware.

Además para utilizarla le vamos a poner un método get y le vamos a exportar:

```
* Click here to ask BlackBox to help you code faster.
'use strict'

const express = require('express');
const FollowController = require('../controllers/follow');
const api = express.Router();
const md_auth = require('../middlewares/authenticated');

api.get('/pruebas-follow', md_auth.ensureAuth, FollowController.prueba);

module.exports = api;
```

Ahora tenemos que exportarla a la api, entonces, nos vamos al archivo app.js este fichero es el que lleva toda la configuración de express.

Recordar que hicimos el fichero index.js, este lo utilizamos para hacer las conexiones y la creación del servidor y el archivo app.js es para llevar cargas de fichero, configuración,etc.

Por lo tanto, en el archivo app.js en el apartado de cargar rutas, cargamos nuestra dirección, sería el segundo:

```
// Cargar rutas

let user_routes = require('./routes/user');
let follow_routes = require('./routes/follow');
```

Y ahora en el apartado de rutas, hay que crear una ruta, utilizando el middleware sobre escribimos la ruta que ya hay por defecto, es decir, a partir de la /api vamos a cargar todas las rutas que tengan que ver con el follow.

```
// rutas
app.use('/api', user_routes);
app.use('/api', follow_routes);
```

Ahora para probarlo, nos vamos a nuestro postman y lo probamos:

The screenshot shows a Postman interface with a GET request to `http://localhost:3800/api/pruebas-follow`. In the Headers section, there is a checked checkbox next to the `Authorization` header, which contains a long JWT token. In the Body section, the response is displayed as JSON, showing a single key-value pair where the key is `message` and the value is `"Hola mundo desde el controlador de follow"`.

5.2. Seguir a un usuario

Vamos a crear un método para la nuestra api que le permita a un usuario seguir a otro. Para ello nos situamos en nuestro controlador y añadiremos este nuevo método.

Vamos a crear una función llamada `saveFollow` en la cual recogeremos lo mencionado anteriormente. En ella vamos a utilizar el modelo de `follow` el cual hemos creado antes. Para ello hacemos referencia creando una variable nuevo para recoger los datos `new Follow();`

```
function saveFollow(req,res){
```

```
let follow = new follow();
```

Ahora debemos setear la información, pero primero vamos a recoger los parámetros que nos llegan por la petición con el `params.body`, para que este recoja el resultado del `body`.

```
let params = req.body;
```

Después vamos setear el valor a cada una de las propiedades de follow:

```
follow.user = req.user.sub;
```

Lo hacemos así, porque en la propiedad user del objeto req he adjuntado a la hora de hacer la autenticación he adjunto un objeto con todo el usuario que está logueado. Entonces el user(follow.user) que es el usuario que sigue, es decir, nosotros, guardo el id del usuario identificado.

Ahora el usuario seguido (follow.followed) pues será el usuario que yo le paso por la petición. Esto tengo que pasarlo por la petición POST:

```
follow.followed = params.followed;|
```

Una vez que tenemos el objeto lleno, vamos hacer un follow.save(); vamos a guardar con esto el objeto, además vamos hacer unas comprobaciones por si nos llega bien o no:

```
follow.save((err, followStored) => {
  if(err) return res.status(500).send({message: 'Error al guardar el seguimiento'});

  if(!followStored) return res.status(404).send({message: 'El seguimiento no se ha guardado'});

  return res.status(200).send({follow:followStored});
});
```

Por último, lo exportamos abajo:

```
module.exports = {
  saveFollow
}
```

Y ahora nos vamos a nuestra hoja de ruta y creamos su ruta:



```
api.post('/follow', md_auth.ensureAuth, FollowController.saveFollow);
```

Ahora si nos vamos a postman y probamos, deberemos coger y en el apartado de body poner el id del usuario que queremos seguir, este caso vamos a utilizar el de admin:

The screenshot shows the Postman interface with a successful API call. The top bar indicates a POST request to `http://localhost:3800/api/follow`. The 'Body' tab is selected, showing a table with three rows: 'followed' (checked), 'inma' (unchecked), and 'true' (unchecked). The 'Body' section at the bottom displays a JSON response with the follow object containing user and followed user details.

	Key	Value	Description
<input checked="" type="checkbox"/>	followed	663537b39dc0f8256f...	
<input type="checkbox"/>		inma	
<input type="checkbox"/>		true	
	Key	Value	Description

Body Pretty Raw Preview Visualize JSON CSV XML Copy Save as example ...

```
1 {  
2   "follow": {  
3     "_id": "663caf8c3b02b8cfad528f11",  
4     "user": "663b8ab864a7fdcd87d25149",  
5     "followed": "663537b39dc0f8256fa92283",  
6     "__v": 0  
7   }  
8 }
```

Y efectivamente nos devuelve el follow es decir, el usuario que yo sigo. El user soy yo identificado y el followed el usuario que estamos siguiendo que en este caso el admin:

Aquí podemos ver que el user admin termina en 83 su id en la base de datos:

```
{  
    "_id" : ObjectId("663537b39dc0f8256fa92283"),  
    "name" : "admin",  
    "surname" : "admin",  
    "nick" : "admin",  
    "email" : "admin@admin.com",  
    "password" : "admin",  
    "image" : null  
}
```

Y el nuestro en 49:

```
{  
    "_id" : ObjectId("663b8ab864a7fdcd87d25149"),  
    "name" : "inma",  
    "surname" : "corcoles",  
    "nick" : "inmacorcoles",  
    "email" : "inma@inma.com",  
    "password" : "$2a$10$q86neJ70zs306qNkJ6Avi.04bD3dc.mRJDC2jUCAKybqQo8h3fFmm",  
    "role" : "ROLE_USER",  
    "image" : null,  
    "__v" : NumberInt(0)  
}
```

Para ver que se está haciendo correctamente, nos situamos en nuestra base de datos y vemos que ahora en colecciones tenemos un nuevo apartado llamado follow y dentro de él, observamos que tenemos este nuevo seguimiento:

The screenshot shows the Robo 3T MongoDB interface. The top navigation bar has tabs for 'Quickstart', 'users', and 'follows'. Below the navigation is a toolbar with 'Run', 'Load query', 'Save query', 'Query history', 'Set default query', and 'Copy'. The main area has sections for 'Query' (empty), 'Projection' (empty), and 'Skip' (empty). The 'Result' section shows a single document in JSON format:

```
1 {  
2   "_id" : ObjectId("663caf8c3b02b8cfad528f11"),  
3   "user" : ObjectId("663b8ab864a7fdcd87d25149"),  
4   "followed" : ObjectId("663537b39dc0f8256fa92283"),  
5   "__v" : NumberInt(0)  
6 }
```

Below the result, there is a table with a single row labeled '_id' containing the value '663caf8c3b02b8cfad528f11'.

De igual forma podemos hacer con el resto de usuarios, si nos vamos a nuestra base de datos y cogemos el otro usuario, cogemos su id:

The screenshot shows the Robo 3T MongoDB interface with the 'users' collection selected. The document shown is:

```
1 {  
2   "_id" : ObjectId("663b66724ad7013c390267e5"),  
3   "name" : "jose",  
4   "surname" : "lopez",  
5   "nick" : "joselopez",  
6   "email" : "jose@jose.com",  
7   "role" : "ROLE_USER",  
8   "image" : null,  
9   "password" : "$2a$10$QxWijkE8j9N3cY11Cl9wHeZP3KKDKm4aT8Pl.9kQ03WV.8e5PZCgG",  
10  "__v" : NumberInt(0)  
11 }
```

Y en postman se lo pegamos y hacemos otra petición:

POST Send

Params Auth Headers (10) Body Scripts Tests Settings

x-www-form-urlencoded

	Key	Value	Description
<input checked="" type="checkbox"/>	followed	663b66724ad7013c3...	
<input type="checkbox"/>		inma	
<input type="checkbox"/>		true	
	Key	Value	Description

Body 200 OK 7 ms 361 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {  
2   "follow": {  
3     "_id": "663cb4e63b02b8cfad528f13",  
4     "user": "663b8ab864a7fdcd87d25149",  
5     "followed": "663b66724ad7013c390267e5",  
6     "__v": 0  
7   }  
8 }
```

Ahora en nuestra base de datos en la colección de follow tendremos otra:

The screenshot shows the Robo 3T MongoDB interface. The top navigation bar has tabs for 'Quickstart', 'users', and 'follows'. Below the navigation is a toolbar with 'Run', 'Load query', 'Save query', 'Query history', 'Set default query', 'Copy', and other options. The main area has sections for 'Query' (with a dropdown set to 0), 'Projection' (set to 0), and 'Skip' (set to 0). A 'Result' tab is selected, showing a table with one row. The table has columns for '_id', 'follows > _id', and '_id'. The first row shows '_id' values '663caf8c3b0...' and '663cb4e63b0...'. To the right is a 'Document JSON Viewer' window. The JSON content is:

```
1 {  
2     "_id" : ObjectId("663cb4e63b02b8cfad528f13"),  
3     "user" : ObjectId("663b8ab864a7fdcd87d25149"),  
4     "followed" : ObjectId("663b66724ad7013c390267e5"),  
5     "__v" : NumberInt(0)  
6 }
```

At the bottom of the JSON viewer are buttons for 'JSON: Legacy Mongo Shell Format' and 'Enable word wrap', and a 'Close' button.

Con este nuevo método podremos ver cómo los usuarios se siguen entre sí. Y esa colección, en esos documentos se va a guardar esa relación entre un usuario y otro.

5.3. Dejar de seguir a un usuario

Como bien hemos hecho anteriormente con el método de poder seguir a un usuario, ahora vamos a proceder hacer el de poder dejar de seguir a un usuario. Poder borrar un follow.

Para ello vamos a nuestro controlador deleteFollow a este lo mismo de siempre le vamos a pasar una request y una response, en esta tendremos que recoger userId y el followId.

Por eso vamos a crear una variable llamada userID, es decir, el usuario que está siguiendo ahora mismo, el que está logueado. Y después el usuario que vamos a dejar de seguir followID y le pasaremos los parámetro que pasaran por la URL.

```
function deleteFollow(req,res){  
    let userId = req.user.sub;  
    let followId = req.params.id;  
}
```

Ahora lo que vamos a hacer es buscar por ID tanto el usuario logueado que sigue, como al usuario que estamos siguiendo, para poder borrar, utilizando find y remove:

```
Follow.find({ 'user':userId, 'followed':followId}).remove(err =>{  
    if(err) return res.status(500).send({message: 'Error al dejar de seguir'});  
  
    return res.status(200).send({message: 'El follow se ha eliminado'});  
});
```

Entonces quedaría así:

```
function deleteFollow(req,res){  
    let userId = req.user.sub;  
    let followId = req.params.id;  
  
    Follow.find({ 'user':userId, 'followed':followId}).remove(err =>{  
        if(err) return res.status(500).send({message: 'Error al dejar de seguir'});  
  
        return res.status(200).send({message: 'El follow se ha eliminado'});  
    });  
}
```

Ahora tendremos que exportarlo abajo:

```
// Export the function  
module.exports = [  
    saveFollow,  
    deleteFollow  
];
```

Nos tendremos que situar en el archivo de ruta y crear su ruta, para poder utilizarlo, teniendo en cuenta que la petición se hará por DELETE.

```
api.delete('/follow/:id',md_auth.ensureAuth,FollowController.deleteFollow);
```

Ahora haremos una comprobación desde el Postman indicando el id arriba en la URL:

The screenshot shows the Postman interface. At the top, it says "DELETE" and "http://localhost:3800/api/follow/663b66724...". Below that, there are tabs for "Params", "Auth", "Headers (10)", "Body", "Scripts", "Tests", and "Settings". The "Body" tab is selected and has a dropdown set to "x-www-form-urlencoded". The body table contains three rows:

Key	Value	Description
followed	663b66724ad7013c3...	
name	inma	
active	true	

At the bottom, it shows "Body" with "Pretty", "Raw", "Preview", "Visualize", and "JSON" tabs. The JSON tab is selected and displays the following response:

```
1 {  
2   "message": "El follow se ha eliminado"  
3 }
```

Y si nos situamos en nuestra base de datos observamos que antes teníamos 2 seguidores y ahora solo nos queda uno:

_id	user	followed	__v
663caf8c3b02b8...	663b8ab864a7f...	663537b39dc0f...	0

5.3. Listado de usuarios que sigo

Ahora vamos a crear una serie de métodos, para poder recoger los usuarios que seguimos, los usuarios que nos siguen .

Lo primero que vamos hacer es el poder listar de forma paginada a los usuarios que estamos siguiendo. Para ello vamos a crear un método nuevo que se va a llamar getFollowingUsers, dentro de ella lo primero que tenemos que hacer es recoger el usuario logueado creando un variable user y donde recogeremos el id. Después tendremos que comprar si nos llega por un parámetro de la URL el usuario, porque en el caso de que nos llegue un ID de usuario este será prioritario para utilizarlo y si no nos llega nada, utilizamos el ID del usuario identificado.

```
function getFollowingUsers(req,res){
  let userId = req.user.sub;
}
```

```
    if (req.params.id){
      userId = req.params.id;
    }
```

Ahora también tendremos que comprobar si llega la página.

```
let page = 1;
if (req.params.page){
  page = req.params.page;
}
```

Crearemos una variable para mostrar por página 4 usuarios, en la cual observaremos si entra mostrando una serie de mensajes:

```
let itemsPerPage = 4;

follow.find({user:userId}).populate({path: 'followed'}).paginate(page, itemsPerPage, (err,follows,total) => [
  if (err) return res.status(500).send({message: 'Error en el servidor'});

  if (!follows) return res.status(404).send({message: 'No estás siguiendo a ningun usuario'});

  return res.status(200).send({
    total: total,
    pages: Math.ceil(total/itemsPerPage),
    follows
  });
]);
```

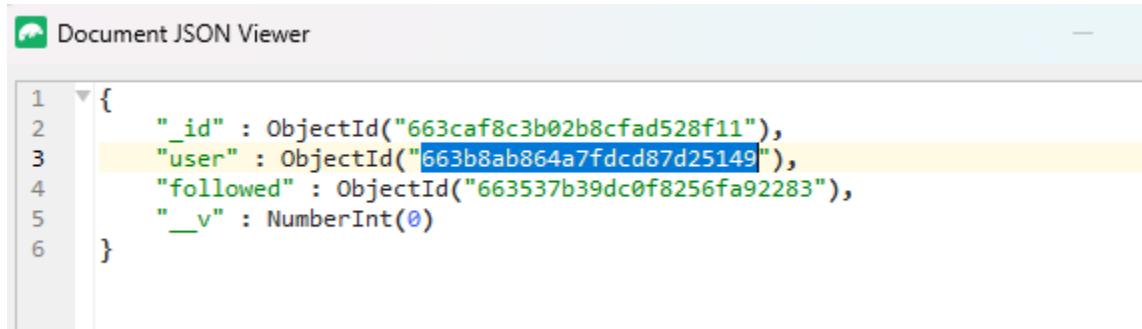
Ahora vamos a exportar abajo del todo, este nuevo método:

```
// Export the function
module.exports = [
  saveFollow,
  deleteFollow,
  getFollowingUsers
];
```

Y por último creamos la ruta, donde le vamos a pasar un id y una pagina:

```
api.delete('/follow/:id',md_auth.ensureAuth,FollowController.deleteFollow),
api.get('/following/:id/:page',md_auth.ensureAuth,FollowController.getFollowingUsers);
```

Ahora vamos hacer una comprobación a través de postman, para ello cogeremos el id del usuario que queremos ver desde la base de datos:



```
1 {  
2   "_id" : ObjectId("663caf8c3b02b8cfad528f11"),  
3   "user" : ObjectId("663b8ab864a7fdcd87d25149"),  
4   "followed" : ObjectId("663537b39dc0f8256fa92283"),  
5   "__v" : NumberInt(0)  
6 }
```

Después en la petición de postman se lo pasaremos por la URL, además, también le pasaremos la página en este caso uno y enviamos la petición por el método GET y abajo nos muestra el número total de elementos, la página y en follows nos muestra un arraïd de documentos con todos los datos, con mi ID de usuario, pero además me muestra populada la información y lo que me ha hecho cambiando el id por el objeto completo el cual yo estoy siguiendo. Esto vendrá genial para hacer un listado de los usuarios que yo estoy siguiendo.

GET Send

Params Auth Headers (8) Body Scripts Tests Settings ...

Headers 7 hidden

	Key	Value	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Authorization	eyJ0eXAiOiJKV1QiLCJ...			
	Key	Value	Description		

Body 200 OK 21 ms 500 B Save as example ...

Pretty Raw Preview Visualize JSON ≡ □ Q

```
1 {  
2   "total": 1,  
3   "pages": 1,  
4   "follows": [  
5     {  
6       "_id": "663caf8c3b02b8cfad528f11",  
7       "user": "663b8ab864a7fdcd87d25149",  
8       "followed": {  
9         "_id": "663537b39dc0f8256fa92283",  
10        "name": "admin",  
11        "surname": "admin",  
12        "nick": "admin",  
13        "email": "admin@admin.com",  
14        "password": "admin",  
15        "image": null  
16      },  
17      "__v": 0  
18    }  
19  }  
20 }
```

Ahora hemos cogido y seguido a algún usuario más y al probar nuestro método, observamos que nos lista y muestra el total de objetos que tenemos, en este caso 6 en total y tenemos 2 páginas.

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:3800/api/following/663b8at...`. The response body is a JSON object:

```
1 {  
2   "total": 6,  
3   "pages": 2,  
4   "follows": [  
5     {  
6       "_id": "663caf8c3b02b8cfad528f11",  
7       "user": "663b8ab864a7fdcd87d25149",  
8       "followed": {  
9         "_id": "663537b39dc0f8256fa92283",  
10        "name": "admin",  
11        "surname": "admin",  
12        "nick": "admin",  
13        "email": "admin@admin.com",  
14        "password": "admin",  
15        "image": null  
16      },  
17      "__v": 0  
18    },  
19    {  
20      "_id": "663cee0f4176cddd8f7cd5ed",  
21    }  
]
```

5.4. Listado de seguidores

Ahora vamos a crear un método que nos va a sacar los usuarios que nos están siguiendo.

Para ello vamos a crear un función llamada `getFollowedUsers`, está recogerá un `request` y una `response`. Como anteriormente creamos una variable donde recoja el id del usuario identificado y haremos una serie de comprobaciones para ver si está logueado el usuario y nos sigue. Es muy parecido al método anterior:

```
85 function getFollowedUsers(req, res) {
86   // Retrieve the user ID from the request parameters or the authenticated user's ID
87   const userId = req.params.id || req.user.sub;
88
89   // Validate the user ID
90   if (!userId) {
91     return res.status(500).send({ message: 'Error en el servidor' });
92   }
93
94   // Set the page number and number of items per page
95   let page = parseInt(req.params.page) || 1;
96   let itemsPerPage = 4;
97
98   // Retrieve a list of followers for the user
99   Follow.find({ followed: userId })
100  .then((follows) => [
101    // Validate the list of followers
102    if (!follows) {
103      return res.status(404).send({ message: 'No te esta siguiendo ningun usuario' });
104    }
105
106    // Populate the 'followed' field of each follower object
107    follows = follows.map((follow) => {
108      return follow.populate('user');
109    });
110
111    // Wait for all of the follower objects to be populated
112    return Promise.all(follows);
113  ])
114  .then((follows) => {
115    // Send a JSON response containing the total number of followers, the number of pages of followers, and the array of follower objects
116    return res.status(200).send({
117      total: follows.length,
118      pages: Math.ceil(follows.length / itemsPerPage),
119      follows
120    });
121  })
122  .catch((err) => {
123    // Handle any errors that occur during the execution of the function
124    console.log(err);
125    return res.status(500).send({ message: 'Error en el servidor' });
126  });
127}
```

Finalmente exportamos este nuevo método:

```
// Export the function
module.exports = {
  saveFollow,
  deleteFollow,
  getFollowingUsers,
  getFollowedUsers
};
```

Y creamos en el archivo de ruta, su ruta propia:

```
api.get('/followed/:id/:page',md_auth.ensureAuth,FollowController.getFollowedUsers);
```

Ahora hacemos la comprobación de este método a través del postman: Aquí vemos como nos sale que nos sigue un usuario.

GET Send

Params Auth Headers (8) Body Scripts Settings ...

Query Params

	Key	Value	Description	Bulk Edit
	Key	Value	Description	

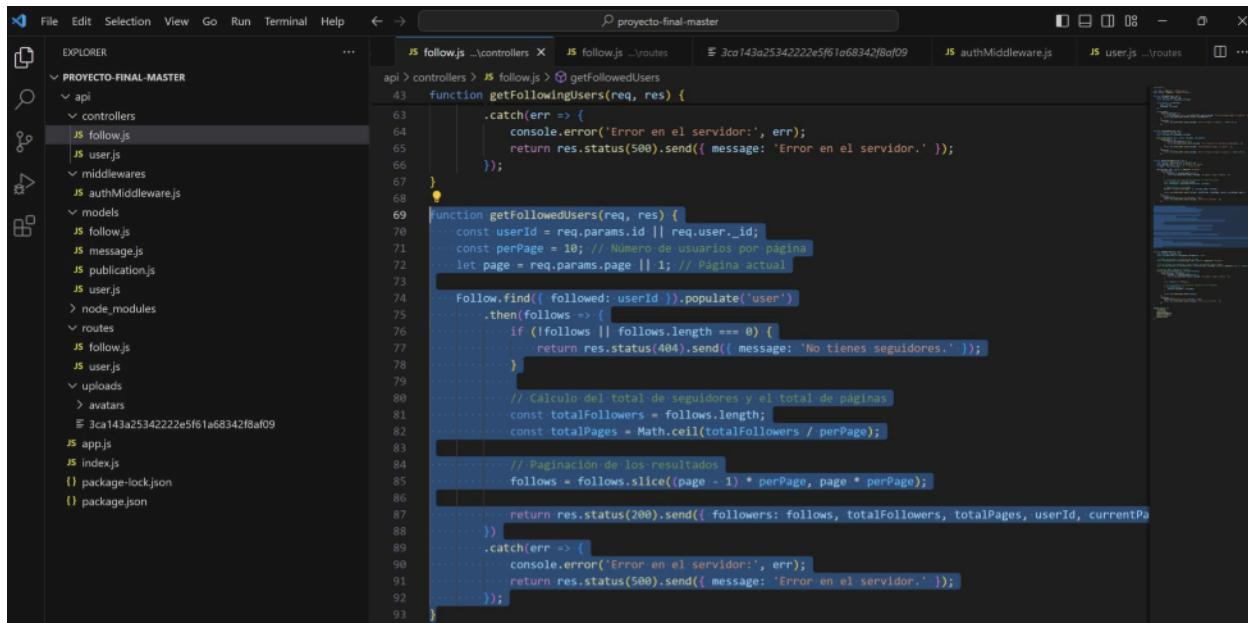
Body ... 200 OK 22 ms 588 B Save as example ...

Pretty Raw Preview Visualize JSON

```
1 {  
2   "total": 1,  
3   "pages": 1,  
4   "follows": [  
5     {  
6       "_id": "663caf8c3b02b8cfad528f11",  
7       "user": {  
8         "_id": "663b8ab864a7fdcd87d25149",  
9         "name": "inma",  
10        "surname": "corcoles",  
11        "nick": "inmacorcoles",  
12        "email": "inma@inma.com",  
13        "password": "$2a$10$q86neJ70zs306qNkJ6Avi.  
14          04bD3dc.mRJDC2jUCAKybqQo8h3fFmm",  
15        "role": "ROLE_USER",  
16        "image": null,  
17        "__v": 0  
18     },  
19     "followed": "663537b39dc0f8256fa92283",  
20     "__v": 0
```

5.5. Listado de usuarios sin paginar

Creamos dos funciones diseñadas para obtener la lista de usuarios que sigue un usuario específico y la lista de usuarios que siguen a un usuario específico, respectivamente. `getFollowingUsers(req, res)` Esta función recibe una solicitud `req` y devuelve una respuesta `res`. Primero, verifica si se proporciona un ID de usuario en los parámetros de la solicitud. Si no se proporciona, utiliza el ID de usuario del usuario autenticado (`req.user._id`). Luego, realiza una búsqueda en la colección `Follow` para encontrar todos los documentos donde el campo `user` coincide con el ID de usuario proporcionado. Utiliza `.populate('followed')` para poblar los documentos con la información completa de los usuarios seguidos. Después de obtener los resultados, realiza el cálculo del total de usuarios seguidos y el total de páginas necesarias para paginar los resultados. Finalmente, devuelve los resultados paginados junto con la información sobre el número total de usuarios seguidos y la página actual.



```
File Edit Selection View Go Run Terminal Help < > proyector-final-master
EXPLORER PROYECTO-FINAL-MASTER
api controllers
  follow.js
  user.js
  authMiddleware.js
models
  follow.js
  message.js
  publication.js
  user.js
routes
  follow.js
  user.js
uploads
  avatars
  E:\ca143a2534222e5f61a68342f8af09
app.js
index.js
package-lock.json
package.json

JS follow.js ...controllers JS follow.js ...routes JS 3ca143a2534222e5f61a68342f8af09 JS authMiddleware.js JS user.js ...routes ...
```

```
function getFollowingUsers(req, res) {
  ...
}

function getFollowedUsers(req, res) {
  const userId = req.params.id || req.user._id;
  const perPage = 10; // Número de usuarios por página
  let page = req.params.page || 1; // Página actual

  Follow.find({ followed: userId }).populate('user')
    .then(follows => {
      if (!follows || follows.length === 0) {
        return res.status(404).send({ message: 'No tienes seguidores.' });
      }

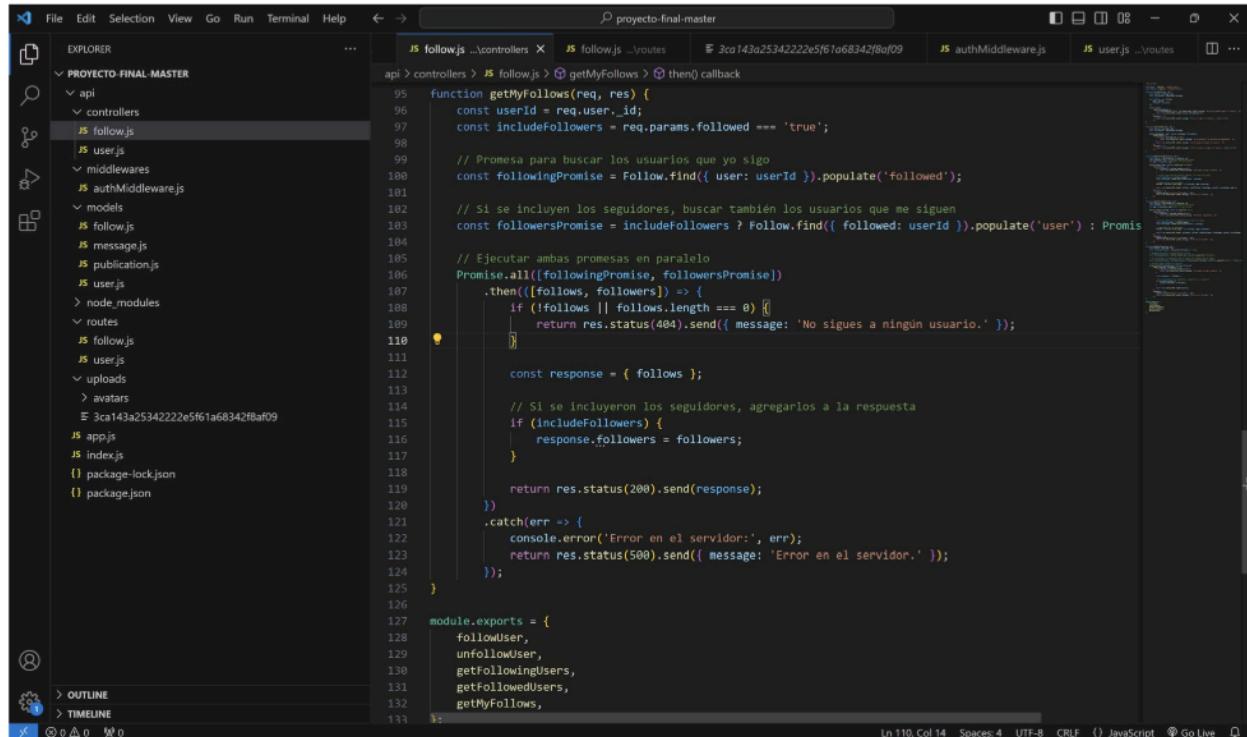
      // Cálculo del total de seguidores y el total de páginas
      const totalFollowers = follows.length;
      const totalPages = Math.ceil(totalFollowers / perPage);

      // Paginación de los resultados
      follows = follows.slice((page - 1) * perPage, page * perPage);

      return res.status(200).send({ followers: follows, totalFollowers, totalPages, userId, currentPage: page });
    })
    .catch(err => {
      console.error('Error en el servidor:', err);
      return res.status(500).send({ message: 'Error en el servidor.' });
    });
}
```

`getFollowedUsers(req, res)` Similar a `getFollowingUsers`, esta función también recibe una solicitud `req` y devuelve una respuesta `res`. Al igual que en la función anterior, verifica si se proporciona un ID de usuario en los parámetros de la solicitud. Si no se proporciona, utiliza el ID de usuario del usuario autenticado (`req.user._id`). Luego, realiza una búsqueda en la colección `Follow` para encontrar todos los documentos donde el campo `followed` coincide con el ID de usuario proporcionado. Utiliza `.populate('user')` para poblar los documentos con la información completa de los usuarios que siguen al usuario específico. Después de obtener los resultados, realiza el cálculo del total de seguidores y el total de páginas necesarias para paginar los

resultados. Finalmente, devuelve los resultados paginados junto con la información sobre el número total de seguidores y la página actual.



The screenshot shows the Visual Studio Code interface with the file `follow.js` open in the center editor pane. The code is a Node.js module for handling user follows. It includes imports from `Follow`, `User`, and `mongoose`. The `getMyFollows` function takes a `req` and `res` object. It retrieves the user ID from the request, sets a flag for including followers, and then runs two parallel promises: one to find users followed by the user and another to find users who follow the user. It then merges the results, handles pagination, and returns the response. A `module.exports` object is defined at the bottom with methods for following and unfollowing users and getting follow counts.

```
function getMyFollows(req, res) {
  const userId = req.user._id;
  const includeFollowers = req.params.followed === 'true';

  // Promesa para buscar los usuarios que yo sigo
  const followingPromise = Follow.find({ user: userId }).populate('followed');

  // Si se incluyen los seguidores, buscar tambien los usuarios que me siguen
  const followersPromise = includeFollowers ? Follow.find({ followed: userId }).populate('user') : Promise.resolve();

  // Ejecutar ambas promesas en paralelo
  Promise.all([followingPromise, followersPromise]).then(([follows, followers]) => {
    if (!follows || follows.length === 0) {
      return res.status(404).send({ message: 'No sigues a ningún usuario.' });
    }

    const response = { follows };

    // Si se incluyeron los seguidores, agregarlos a la respuesta
    if (includeFollowers) {
      response.followers = followers;
    }

    return res.status(200).send(response);
  })
  .catch(err => {
    console.error('Error en el servidor:', err);
    return res.status(500).send({ message: 'Error en el servidor.' });
  });
}

module.exports = {
  followUser,
  unfollowUser,
  getFollowingUsers,
  getFollowedUsers,
  getMyFollows,
}
```

Aquí podemos ver una comprobación desde postman:

GET http://localhost:3800/api/get-my-follows Send

Params Auth Headers (8) Body Scripts Settings ...

Query Params

	Key	Value	Des...	...	Bulk Edit
	Key	Value	Description		

Body 200 OK 29 ms 3.2 KB Save as example ...

Pretty Raw Preview Visualize JSON ≡ □ Q

```
4   "_id": "663caf8c3b02b8cfad528f11",
5   "user": {
6     "_id": "663b8ab864a7fdcd87d25149",
7     "name": "inma",
8     "surname": "corcoles",
9     "nick": "inmacorcoles",
10    "email": "inma@inma.com",
11    "password": "$2a$10$q86neJ70zs306qNkJ6Avi.
12      04bD3dc.mRJDC2jUCAKybqQo8h3fFmm",
13    "role": "ROLE_USER",
14    "image": null,
15    "__v": 0
16  },
17  "followed": {
18    "_id": "663537b39dc0f8256fa92283",
19    "name": "admin",
20    "surname": "admin",
21    "nick": "admin",
22    "email": "admin@admin.com",
23    "password": "admin",
24    "image": null
25  },
26  "__v": 0
```

Si le pasamos como parámetro arriba en al URL un true, no saca el mensaje de que nadie no está siguiendo:

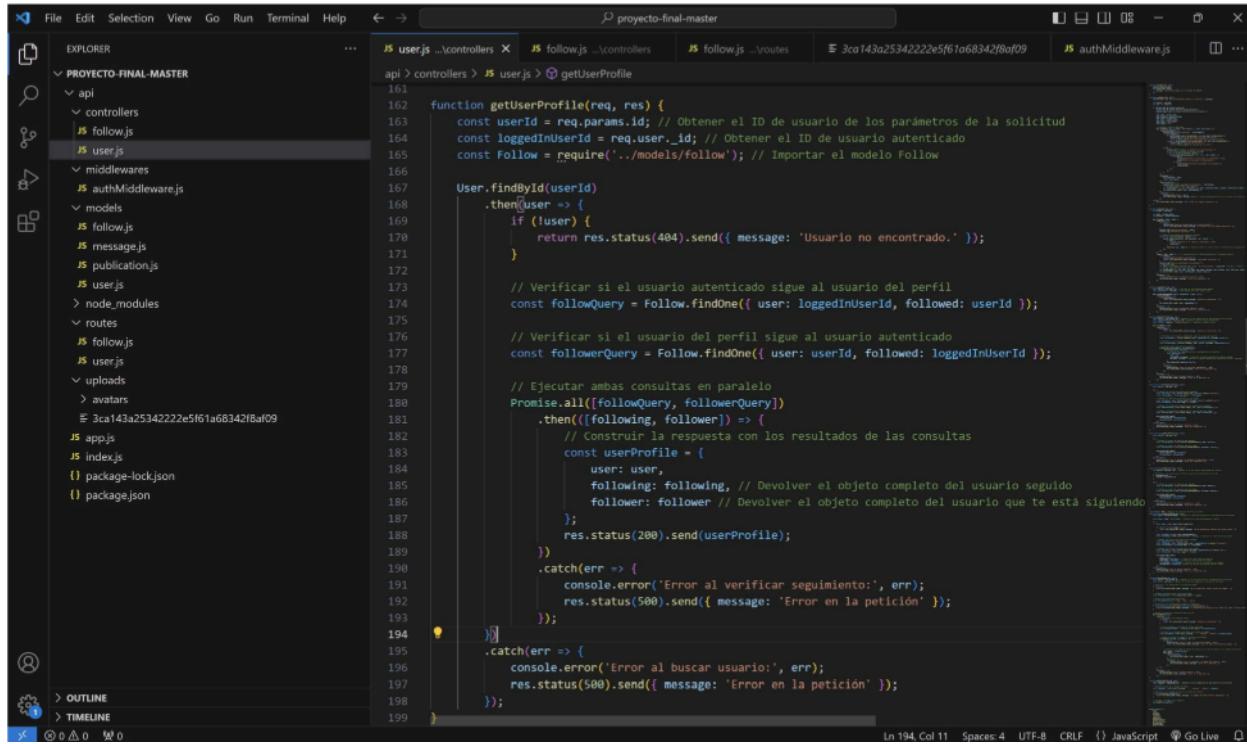
The screenshot shows the Postman interface. At the top, there is a header bar with 'GET' selected, a URL input field containing 'http://localhost:3800/api/get-my-follows/true', and a large blue 'Send' button. Below the header, there are tabs for 'Params', 'Auth', 'Headers (8)', 'Body', 'Scripts', and 'Settings'. The 'Params' tab is currently active. Under 'Query Params', there is a table with columns: Key, Value, Description, Des..., and Bulk Edit. There is one row in the table with 'Key' and 'Value' both empty. At the bottom of the interface, there is a 'Body' section with a dropdown menu set to 'Pretty'. The main content area displays the response: '404 Not Found' with a clock icon, '7 ms', '283 B', and a 'Save as example' button. The JSON response is shown as follows:

```
1 {  
2   "message": "No sigues a ningún usuario"  
3 }
```

5.6. ¿Sigo a este usuario?

getUserProfile(req, res) Esta función se encarga de obtener el perfil de un usuario específico, incluida la información sobre si el usuario autenticado sigue al usuario del perfil y si el usuario del perfil sigue al usuario autenticado. Primero, obtiene el ID del usuario del perfil desde los parámetros de la solicitud (req.params.id). Luego, obtiene el ID del usuario autenticado (req.user._id). Utiliza User.findById para buscar el usuario del perfil en la base de datos. Después, utiliza dos consultas en paralelo utilizando

Promise.all. La primera consulta verifica si el usuario autenticado sigue al usuario del perfil y la segunda consulta verifica si el usuario del perfil sigue al usuario autenticado. Una vez que se completan ambas consultas, construye la respuesta con los resultados de las consultas y envía el perfil del usuario junto con la información sobre el seguimiento en la respuesta. Si ocurre algún error durante el proceso, devuelve un mensaje de error 500.



```

File Edit Selection View Go Run Terminal Help < > proyector-final-master
EXPLORER JS user.js ...controllers JS follow.js ...controllers JS follow.js ...routes JS 3ca143a2534222e5f61a68342f8af09 JS authMiddleware.js ...
PROYECTO-FINAL-MASTER
api > controllers > JS user.js > getUserProfile
161
162 function getUserProfile(req, res) {
163   const userId = req.params.id; // Obtener el ID de usuario de los parámetros de la solicitud
164   const loggedInUserId = req.user._id; // Obtener el ID de usuario autenticado
165   const Follow = require('../models/follow'); // Importar el modelo Follow
166
167   User.findById(userId)
168     .then(user => {
169       if (!user) {
170         return res.status(404).send({ message: 'Usuario no encontrado.' });
171     }
172
173     // Verificar si el usuario autenticado sigue al usuario del perfil
174     const followQuery = Follow.findOne({ user: loggedInUserId, followed: userId });
175
176     // Verificar si el usuario del perfil sigue al usuario autenticado
177     const followerQuery = Follow.findOne({ user: userId, followed: loggedInUserId });
178
179     // Ejecutar ambas consultas en paralelo
180     Promise.all([followQuery, followerQuery])
181       .then(([following, follower]) => {
182         // Construir la respuesta con los resultados de las consultas
183         const userProfile = {
184           user: user,
185           following: following, // Devolver el objeto completo del usuario seguido
186           follower: follower // Devolver el objeto completo del usuario que te está siguiendo
187         };
188         res.status(200).send(userProfile);
189       })
190       .catch(err => {
191         console.error('Error al verificar seguimiento:', err);
192         res.status(500).send({ message: 'Error en la petición' });
193       });
194     })
195     .catch(err => {
196       console.error('Error al buscar usuario:', err);
197       res.status(500).send({ message: 'Error en la petición' });
198     });
199   }

```

Después cogeríamos y haríamos una prueba en postman sobre ello, para ello cogeríamos y nos registramos con el usuario que más seguidores tienen, cogiendo su token y arriba en la URL le pasamos el usuario que nosotros queremos, para comprobar si lo seguimos o no, en este caso si se siguen:

Vemos por una parte que nos saca la parte del user el cual hemos pasado por URL y abajo nos pone el follow y nos identifica el usuario que seguimos y que nos sigue:

GET <http://localhost:3800/api/user/663ceecf6032d06f24afc7b9> Send

Params Auth Headers (8) Body Scripts Settings [...](#)

Headers [7 hidden](#)

	Key	Value	...	Bulk Edit	Presets	⋮
<input checked="" type="checkbox"/>	Authorization	eyJ0eXAiOiJKV1QiLCJ...				⋮
	Key	Value	Description			⋮

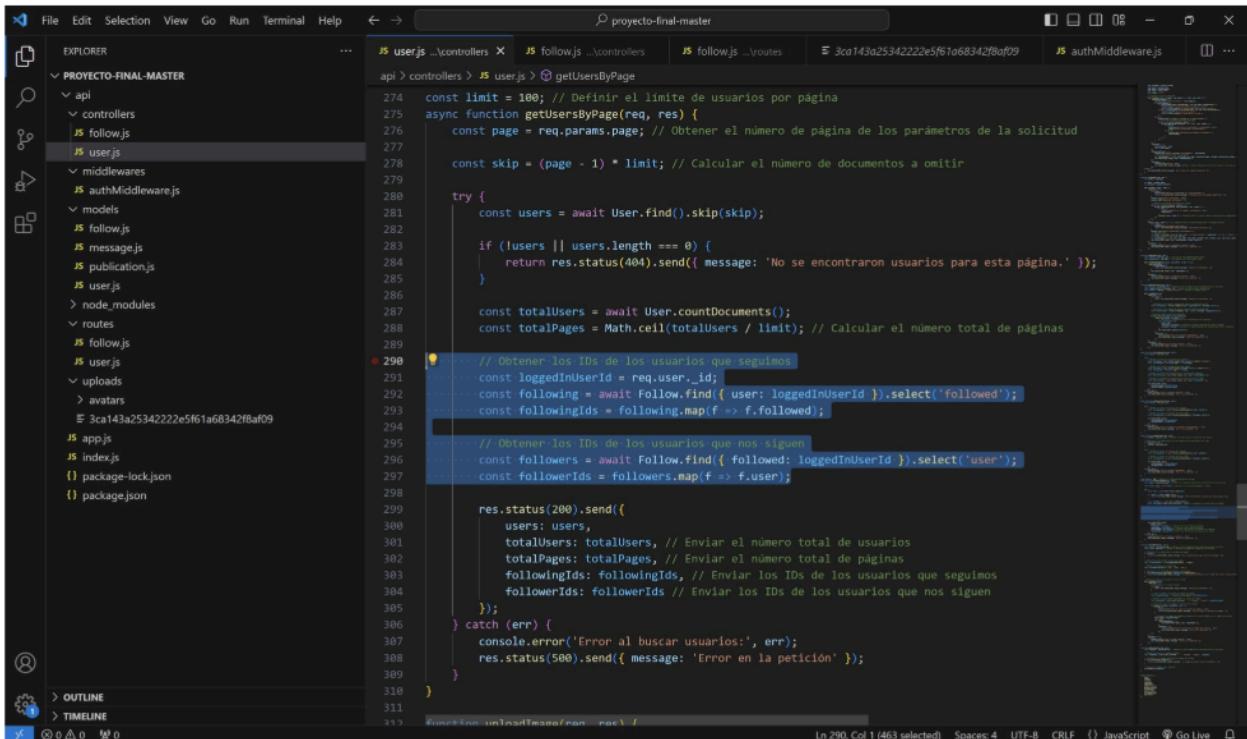
Body [Pretty](#) [Raw](#) [Preview](#) [Visualize](#) [JSON](#) [CSV](#) [Copy](#) [Save as example](#) [...](#)

```
1 {
2   "user": {
3     "_id": "663ceecf6032d06f24afc7b9",
4     "name": "jesus",
5     "surname": "Lopez",
6     "nick": "jesuslopez",
7     "email": "jesus@jesus.com",
8     "role": "ROLE_USER",
9     "image": null,
10    "password": "$2a$10$ZsQzw40ljEV0GOWx/1.62.
11      bqMbHgRucak1hiPMCdmaD1AE4TwZdP.",
12    "__v": 0
13  },
14  "follow": {
15    "_id": "663cef864176cddd8f7cd5f1",
16    "user": "663b8ab864a7fdcd87d25149",
17    "followed": "663ceecf6032d06f24afc7b9",
18    "__v": 0
19 }
```

5.7. Async y Await

Ahora, vamos a implementar las funcionalidades de `async` y `await`. La función `getUsersByPage` se encarga de obtener una lista de usuarios paginada y proporcionar información sobre los usuarios que seguimos y los que nos siguen. Veamos qué hace el `async` y `await` en esta función:

1. `async`: El uso de `async` en la declaración de la función indica que esta función es asincrónica, lo que significa que puede contener operaciones asincrónicas, como consultas a la base de datos, que pueden tardar un tiempo en completarse. Esto permite el uso de `await` dentro de la función para esperar que estas operaciones asincrónicas se completen antes de continuar con la ejecución del código.
2. `await`: El `await` se utiliza dentro de la función para esperar a que las operaciones asincrónicas se completen y devuelvan un resultado. En este caso, se utiliza `await` con operaciones como `User.find()`, `User.countDocuments()`, y las consultas a la colección `Follow`. Al utilizar `await`, la ejecución del código se detiene en ese punto hasta que la operación asincrónica haya completado su ejecución y devuelva un resultado. Esto permite que el código sea más fácil de leer y escribir, ya que se parece más a un estilo de programación sincrónica. En resumen, el uso de `async` y `await` en esta función facilita la gestión de operaciones asincrónicas al esperar a que se completen antes de continuar con la ejecución del código, lo que mejora la legibilidad y el manejo de errores.



```
File Edit Selection View Go Run Terminal Help < > proyecto-final-master
EXPLORER PROYECTO-FINAL-MASTER
api controllers
  user.js
  follow.js
  user.js
  authMiddleware.js
models
follow.js
message.js
publications.js
user.js
node_modules
routes
follow.js
user.js
uploads
  avatars
  sc143a2534222e5f61a68342f8af09
app.js
index.js
package-lock.json
package.json

JS user.js ...controllers JS follow.js ...controllers JS follow.js ...routes JS sc143a2534222e5f61a68342f8af09 JS authMiddleware.js

274 const limit = 100; // Definir el límite de usuarios por página
275 async function getUsersByPage(req, res) {
276   const page = req.params.page; // Obtener el número de página de los parámetros de la solicitud
277
278   const skip = (page - 1) * limit; // Calcular el número de documentos a omitir
279
280   try {
281     const users = await User.find().skip(skip);
282
283     if (!users || users.length === 0) {
284       return res.status(404).send({ message: 'No se encontraron usuarios para esta página.' });
285     }
286
287     const totalUsers = await User.countDocuments();
288     const totalPages = Math.ceil(totalUsers / limit); // Calcular el número total de páginas
289
290     // Obtener los IDs de los usuarios que seguimos
291     const loggedInUserId = req.user._id;
292     const following = await Follow.find({ user: loggedInUserId }).select('followed');
293     const followingIds = following.map(f => f.followed);
294
295     // Obtener los IDs de los usuarios que nos siguen
296     const followers = await Follow.find({ followed: loggedInUserId }).select('user');
297     const followerIds = followers.map(f => f.user);
298
299     res.status(200).send({
300       users,
301       totalUsers: totalUsers, // Enviar el número total de usuarios
302       totalPages: totalPages, // Enviar el número total de páginas
303       followingIds: followingIds, // Enviar los IDs de los usuarios que seguimos
304       followerIds: followerIds // Enviar los IDs de los usuarios que nos siguen
305     });
306   } catch (err) {
307     console.error('Error al buscar usuarios:', err);
308     res.status(500).send({ message: 'Error en la petición' });
309   }
310 }
311
312
313
314
315
316
317
318
319
```

Una vez creada esta función asíncrona, veremos haciendo una comprobación postman, que nos remite el valor que nosotros le estamos pasando:

GET ▼ http://localhost:3800/api/user/663cef15603... Send ▼

Params Auth Headers (8) Body Scripts Settings ...

Headers 👁 7 hidden

	Key	Value	...	Bulk Edit	Presets	▼
<input checked="" type="checkbox"/>	Authorization	eyJ0eXAiOiJKV1QiLCJ...		trash		
	Key	Value	Description			

Body ▼ 🌐 200 OK 21 ms 626 B 💾 Save as example ...

Pretty Raw Preview Visualize JSON ▼ ≡ ☒ 🔍

```
1 {  
2   "user": {  
3     "_id": "663cef156032d06f24afc7bc",  
4     "name": "enrique",  
5     "surname": "garcia",  
6     "nick": "enriquegarcia",  
7     "email": "enrique@enrique.com",  
8     "role": "ROLE_USER",  
9     "image": null,  
10    "password": "$2a$10$ZsQzw40ljEV0GOWx/1.62.  
11      bqMbHgRucak1hiPMCdmaD1AE4TwZdP.",  
12    "__v": 0  
13  },  
14  "following": {  
15    "_id": "663cefa54176cd8f7cd5f3",  
16    "user": "663b8ab864a7fdcd87d25149",  
17    "followed": "663cef156032d06f24afc7bc",  
18    "__v": 0  
19  },  
20  "followed": null
```

5.8. Ids de usuarios, comprobar el seguimiento

Implementamos en este apartado la siguiente función, FollowUsersIds, que está diseñada para obtener los usuarios que seguimos y los usuarios que nos siguen, junto con su información completa. Aquí está la explicación paso a paso de su implementación:

- Obtención del ID del usuario autenticado: Se obtiene el ID del usuario autenticado desde req.user._id. Esto asegura que solo obtendremos la información de seguimiento relacionada con el usuario que está realizando la solicitud.
- Consulta de los usuarios que seguimos: Utilizando await Follow.find({ user: userId }).select('followed'), buscamos en la colección Follow todos los documentos donde el campo user coincide con el ID del usuario autenticado. La función select('followed') se utiliza para especificar que solo queremos recuperar el campo followed de los documentos coincidentes, que contiene los IDs de los usuarios que seguimos. El resultado de esta consulta se almacena en la variable following.
- Extracción de los IDs de los usuarios que seguimos: Utilizando map sobre el array following, extraemos los IDs de los usuarios que seguimos y los almacenamos en followingIds.
- Consulta de los usuarios que nos siguen: Similar a la consulta anterior, utilizamos await Follow.find({ followed: userId }).select('user') para encontrar todos los documentos en la colección Follow donde el campo followed coincide con el ID del usuario autenticado. También seleccionamos solo el campo user, que contiene los IDs de los usuarios que nos siguen. El resultado de esta consulta se almacena en la variable followers.
- Extracción de los IDs de los usuarios que nos siguen: Usando map sobre el array followers, obtenemos los IDs de los usuarios que nos siguen y los almacenamos en followerIds.
- Obtención de la información completa de los usuarios: Utilizando await User.find({ _id: { \$in: followingIds } }) y await User.find({ _id: { \$in: followerIds } }), buscamos en la colección User los usuarios cuyos IDs están presentes en los arrays followingIds y followerIds, respectivamente. Esto nos da la información completa de los usuarios que seguimos y los usuarios que nos siguen. Los resultados se almacenan en las variables usersFollowing y usersFollowers, respectivamente.
- Envío de la respuesta: Finalmente, enviamos una respuesta HTTP con estado 200 que contiene los usuarios que seguimos y los usuarios que nos siguen, junto con su información completa. Si en algún momento ocurre un error durante el proceso, como una excepción en alguna de las consultas a la base de datos, capturamos ese error en el bloque catch y enviamos una respuesta de error con estado 500. Esto garantiza que la aplicación maneje adecuadamente los errores y proporcione una respuesta adecuada al cliente.

```

PROJECT-FINAL-MASTER
  api
    controllers
      follow.js
      user.js
    middlewares
      authMiddleware.js
    models
      follow.js
      message.js
      publication.js
    routes
      follow.js
      user.js
    uploads
      avatars
    app.js
    index.js
    package-lock.json
    package.json

JS user.js ...controllers JS follow.js ...controllers JS follow.js ...routes JS 3ca143a2534222e5f61a68342f8af09 JS authMiddleware.js

api > controllers > JS user.js > FollowUserIds
162   function getUserProfile(req, res) {
195     .catch(err => {
198   });
199 }
200
201 async function FollowUserIds(req, res) {
202   const userId = req.user._id;
203
204   try {
205     // Obtener los IDs de los usuarios que seguimos
206     const following = await Follow.find({ user: userId }).select('followed');
207     const followingIds = following.map(f => f.followed);
208
209     // Obtener los IDs de los usuarios que nos siguen
210     const followers = await Follow.find({ followed: userId }).select('user');
211     const followerIds = followers.map(f => f.user);
212
213     // Obtener la información completa de los usuarios que seguimos
214     const usersFollowing = await User.find({ _id: { $in: followingIds } });
215
216     // Obtener la información completa de los usuarios que nos siguen
217     const usersFollowers = await User.find({ _id: { $in: followerIds } });
218
219     res.status(200).send({
220       usersFollowing: usersFollowing,
221       usersFollowers: usersFollowers
222     });
223   } catch (err) {
224     console.error('Error al obtener usuarios:', err);
225     res.status(500).send({ message: 'Error en la petición' });
226   }
227
228
229 async function getCountFollow(req, res) {
230   const userId = req.user._id;
231
232   try {
233     // Obtener el número de usuarios que seguimos
234     const followingCount = await Follow.countDocuments({ user: userId });

```

The screenshot shows a code editor with several tabs open. The active tab is 'user.js' located at 'PROJECT-FINAL-MASTER/controllers'. The code implements two asynchronous functions: 'getUserProfile' and 'FollowUserIds'. The 'FollowUserIds' function retrieves the IDs of users followed by the authenticated user and the IDs of users who follow the authenticated user. It then fetches the full profiles of these users from the 'User' collection. The 'getCountFollow' function counts the number of users followed by the authenticated user. The interface includes a sidebar with project files like 'app.js', 'index.js', and 'package.json', and a bottom status bar showing file statistics and encoding.

5.9. Devolver contadores y estadísticas

Para conseguir esto, implementamos dos funciones, diseñadas para obtener el recuento de usuarios que seguimos y el recuento de usuarios que nos siguen, respectivamente. Aquí está una explicación detallada de cada una:

- Función getCountFollow(req, res):** Esta función se encarga de obtener el número de usuarios que seguimos y el número de usuarios que nos siguen para el usuario autenticado. Primero, obtenemos el ID del usuario autenticado desde `req.user._id`. Luego, dentro de un bloque `try`, realizamos dos consultas a la colección `Follow` utilizando el método `countDocuments` de Mongoose. La primera consulta cuenta los documentos donde el campo `user` coincide con el ID del usuario autenticado, lo que nos da el número de usuarios que seguimos. La segunda consulta cuenta los documentos donde el campo `followed` coincide con el ID del usuario autenticado, lo que nos da el número de usuarios que nos siguen. Después de obtener ambos recuentos, enviamos una respuesta HTTP con estado 200 que contiene los recuentos de usuarios que seguimos y usuarios que nos siguen.
- Función get_counters(req, res):** Esta función es similar a `getCountFollow(req, res)`, pero tiene la capacidad adicional de aceptar un ID de usuario opcional a través de los parámetros de la URL. Primero, establecemos el `userId` en el ID del usuario autenticado por defecto. Luego, verificamos si se proporcionó un ID de usuario en los parámetros de la URL. Si se proporciona, actualizamos el `userId` con ese valor. Despu  s, dentro de un bloque `try`, realizamos las mismas consultas a la colecci  n

Follow utilizando el método `countDocuments` de Mongoose, pero esta vez usando el `userId` actualizado. Finalmente, enviamos una respuesta HTTP con estado 200 que contiene los recuentos de usuarios que seguimos y usuarios que nos siguen. Ambas funciones utilizan el manejo de errores con bloques `try-catch` para capturar cualquier error que pueda ocurrir durante la ejecución de las consultas a la base de datos. Si se produce un error, se registra en la consola y se envía una respuesta HTTP con estado 500 indicando un error en la petición.

The screenshot shows a code editor interface with a dark theme. The top navigation bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar containing 'proyecto-final-master'. The left sidebar, titled 'EXPLORER', lists project files and folders: 'PROYECTO-FINAL-MASTER' (api, controllers, models, node_modules, routes, uploads), 'JS follow.js', 'JS user.js' (selected), 'JS authMiddleware.js', 'JS message.js', 'JS publication.js', 'JS user.js', 'node_modules', 'routes', 'JS follow.js', 'JS user.js', 'uploads', 'avatars', and database documents like '_id: 3ca143a2534222e5f61a68342f8af09'. Below the sidebar are buttons for 'OUTLINE' and 'TIMELINE'. The main editor area displays 'user.js' with the following code:

```
JS user.js ...controllers JS follow.js ...controllers JS follow.js ...routes JS 3ca143a2534222e5f61a68342f8af09 JS authMiddleware.js

api > controllers > JS user.js > getCountFollow
229     async function getCountFollow(req, res) {
230         const userId = req.user._id;
231
232         try {
233             // Obtener el número de usuarios que seguimos
234             const followingCount = await Follow.countDocuments({ user: userId });
235
236             // Obtener el número de usuarios que nos siguen
237             const followerCount = await Follow.countDocuments({ followed: userId });
238
239             res.status(200).send({
240                 followingCount,
241                 followerCount
242             });
243         } catch (err) {
244             console.error('Error al obtener el contador de seguimiento:', err);
245             res.status(500).send({ message: 'Error en la petición' });
246         }
247     }
248
249     async function get_counters(req, res) {
250         let userId = req.user._id; // Obtenir el ID del usuario autenticado por defecto
251
252         // Verificar si se proporcionó un ID de usuario en los parámetros de la URL
253         if (req.params.id) {
254             userId = req.params.id;
255         }
256
257         try {
258             // Obtener el número de usuarios que seguimos
259             const followingCount = await Follow.countDocuments({ user: userId });
260
261             // Obtener el número de usuarios que nos siguen
262             const followerCount = await Follow.countDocuments({ followed: userId });
263
264             res.status(200).send({
265                 followingCount,
266                 followerCount
267             });
268         } catch (err) {
269             console.error('Error al obtener contadores:', err);
270             res.status(500).send({ message: 'Error en la petición' });
271         }
272     }

```

The status bar at the bottom indicates 'Ln 243, Col 11' and 'JavaScript'. On the right side of the editor, there is a vertical panel showing a preview of the code or a related document.

Ahora si nosotros hacemos la comprobación en nuestro postman, veremos como nos saca lo que le estamos pidiendo en este caso:

The screenshot shows the Postman interface for a GET request to `http://localhost:3800/api/counters/664214af...`. The 'Headers' tab is selected, displaying one header: `Authorization` with value `eyJ0eXAiOiJKV1QiLCJ...`. The response section shows a status of `200 OK`, `17 ms` duration, and `263 B` size. The response body is a JSON object:

```
1 {  
2   "following": 0,  
3   "followed": 0  
4 }
```

6. Las Publicaciones

6.1. Controlador y rutas de publicaciones

Primeramente, se creará el archivo 'publication.js' y hará las siguientes funcionalidades:

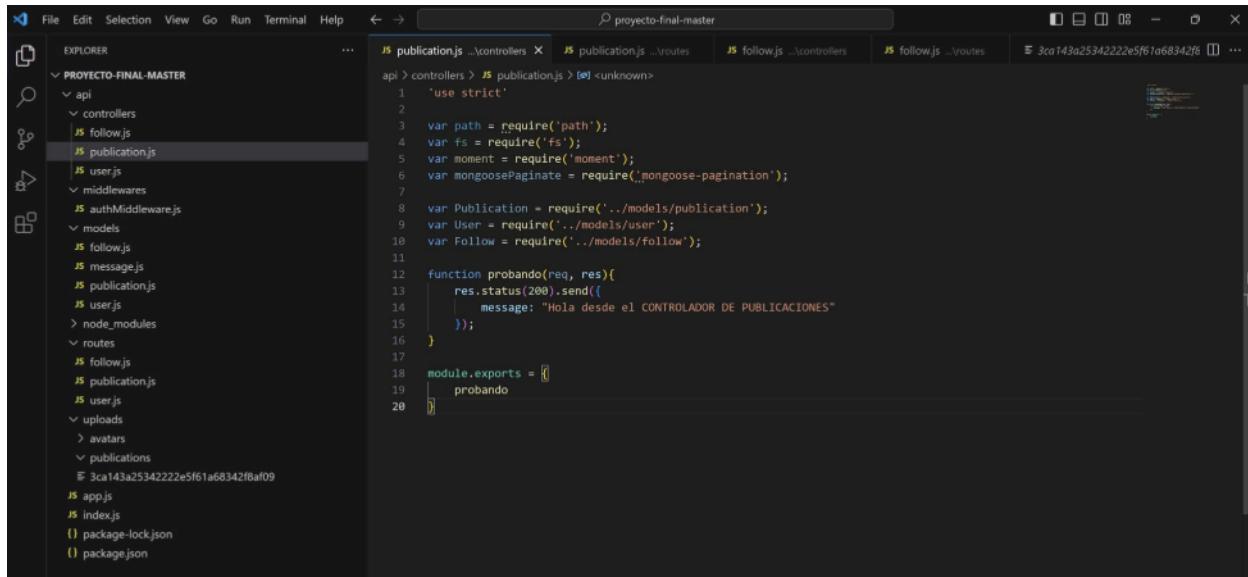
1. Require de módulos: Las primeras líneas del código importan módulos de Node.js y otros módulos instalados mediante npm. path, fs, moment y mongoose-pagination son

módulos de Node.js que proporcionan funcionalidades para manejar rutas de archivos, operaciones de sistema de archivos, manipulación de fechas y paginación en MongoDB usando Mongoose, respectivamente. Estos módulos son utilizados para operaciones de entrada y salida, manipulación de fechas y paginación en la aplicación.

2. Require de modelos: Luego, el código importa los modelos definidos en archivos externos. Los modelos Publication, User y Follow se refieren a los modelos de datos definidos en los archivos '../models/publication', '../models/user' y '../models/follow', respectivamente. Estos modelos probablemente definen la estructura y el comportamiento de los datos en la base de datos, y son utilizados en las operaciones de controlador.

3. Función de controlador: La función probando es un controlador de ruta que se utilizará para manejar solicitudes HTTP. En este caso, cuando se recibe una solicitud, la función envía una respuesta con un objeto JSON que contiene un mensaje de saludo.

4. Exportación del controlador: Finalmente, el controlador probando se exporta como parte de un objeto que se asigna a module.exports. Esto permite que otros archivos de la aplicación importen y utilicen esta función de controlador.



```
File Edit Selection View Go Run Terminal Help ⏎ → ⌂ projecto-final-master
EXPLORER PROYECTO-FINAL-MASTER
api controllers publication.js follow.js
publication publication.js user.js
user.js
middlewares authMiddleware.js
follow.js message.js publication.js user.js
node_modules
routes follow.js publication.js user.js
uploads avatars publications
3ca143a2534222e5f61a68342f8af09
app.js index.js package-lock.json package.json

publication.js ..\controllers < unknown>
api > controllers > publication.js > publication.js ..\routes < unknown>
publication publication.js
follow.js ..\controllers follow.js ..\routes < unknown>
follow.js ..\routes < unknown>
3ca143a2534222e5f61a68342f8af09

1 'use strict'
2
3 var path = require('path');
4 var fs = require('fs');
5 var moment = require('moment');
6 var mongoosePaginate = require('mongoose-pagination');
7
8 var Publication = require('../models/publication');
9 var User = require('../models/user');
10 var Follow = require('../models/follow');
11
12 function probando(req, res){
13   res.status(200).send({
14     message: "Hola desde el CONTROLADOR DE PUBLICACIONES"
15   });
16 }
17
18 module.exports = [
19   probando
20 ]
```

Seguidamente, se creará el archivo de rutas 'publication.js'

The screenshot shows the VS Code interface with the project 'PROYECTO-FINAL-MASTER' open. The Explorer sidebar shows files like 'api', 'controllers', 'middlewares', 'models', 'routes', and 'uploads'. The 'publication.js' file in the 'models' folder is selected and its content is displayed in the main editor:

```
api > routes > publication.js <unknown>
1  'use strict';
2
3  const express = require('express');
4  const authMiddleware = require('../middlewares/authMiddleware');
5  const PublicationController = require('../controllers/publication');
6
7  const router = express.Router();
8
9  // Utiliza el middleware 'md_auth.ensureAuth' en la ruta '/probando-pub'
10 router.get('/probando-pub', authMiddleware, PublicationController.probando);
11
12 module.exports = router;
```

También, se creará

The screenshot shows the VS Code interface with the project 'PROYECTO-FINAL-MASTER' open. The Explorer sidebar shows files like 'api', 'controllers', 'middlewares', 'models', 'routes', and 'uploads'. The 'publication.js' file in the 'models' folder is selected and its content is displayed in the main editor:

```
api > models > publication.js > ...
1  'use strict';
2
3  var mongoose = require('mongoose');
4  var Schema = mongoose.Schema;
5
6  var PublicationSchema = Schema ({
7    text: String,
8    file: String,
9    created_at: String,
10   user: { type: Schema.ObjectId, ref: 'User' }
11 });
12
13 module.exports = mongoose.model('Publication', PublicationSchema);
```

el archivo 'publication.js' dentro de la carpeta models:

Y por último, es necesario crear la carpeta 'publications' dentro de la carpeta 'uploads'

Por tanto, a la hora de probar el endpoint `http://localhost:3800/api/probando-pub` desde 'Postman', vemos correctamente el mensaje, significando que ya tenemos listo el método de publicaciones para añadir diferentes funcionalidades a la API:

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:3800/api/probando-pub`. The response body is JSON, containing the message: "message": "Hola desde el CONTROLADOR DE PUBLICACIONES".

Key	Value
Authorization	eyJhbGciOiJIUzI1NilsInR5cCl6IkpxVCJ9eyJ1c2VyX2lkjoINjYzMl5YjE2M...
Key	Value

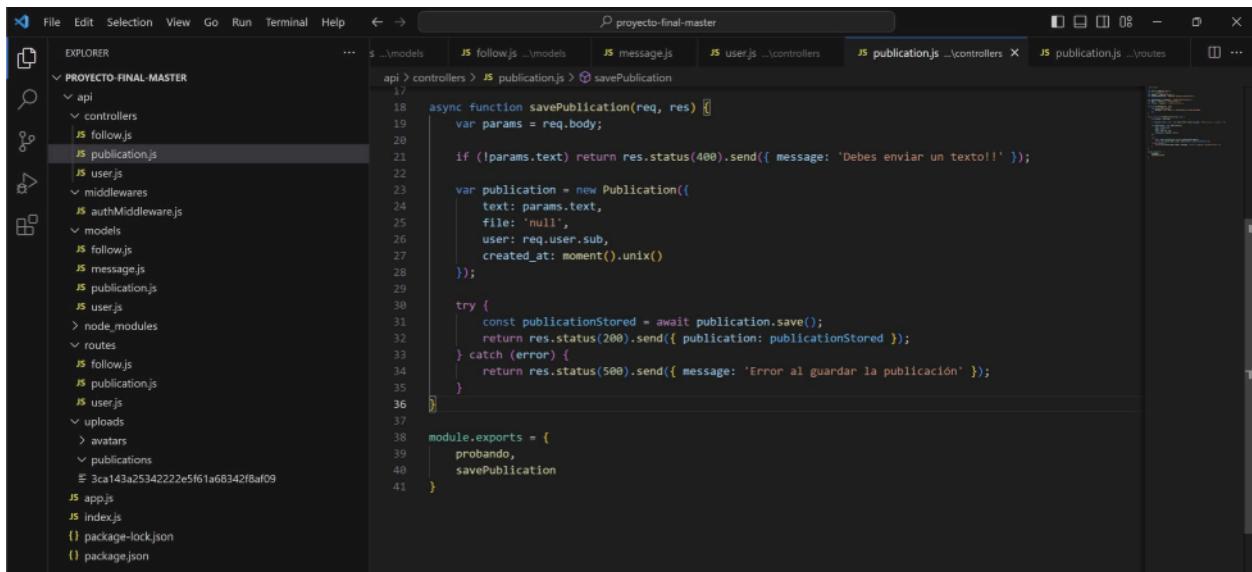
6.2. Guardar nuevas publicaciones

Es momento ahora de crear un nuevo método en el archivo ‘publications.js’ dentro de la carpeta ‘controllers’ que se va a encargar de dar de alta nuevas publicaciones. Esta función `savePublication` es un controlador de ruta en una aplicación Node.js que utiliza Express y Mongoose para manejar solicitudes HTTP destinadas a guardar una nueva publicación en una base de datos MongoDB. Aquí está el desglose de lo que hace la función:

```

Obtener parámetros de la solicitud: var params = req.body;:: Extrae los
parámetros enviados en el cuerpo de la solicitud HTTP utilizando el middleware
body-parser de Express. Estos parámetros pueden contener información sobre la
nueva publicación, como el texto de la publicación. Validar los parámetros: if
(!params.text) return res.status(400).send({ message: 'Debes enviar un texto!' });
Verifica si se proporciona un texto para la publicación. Si no se proporciona, devuelve
una respuesta con un código de estado 400 Bad Request indicando que es necesario
enviar un texto. Es un error del cliente. Crear y guardar la publicación: var publication =
new Publication({ ... });:: Crea una nueva instancia del modelo Publication. Los datos
necesarios para la publicación se asignan a sus propiedades: text, file, user, y
created_at. try {...} catch {...}: Utiliza un bloque try-catch para manejar errores de forma
adecuada. Dentro del bloque try, intenta guardar la publicación llamando a
publication.save(). const publicationStored = await publication.save(); Utiliza await para
esperar la resolución de la promesa devuelta por la función save(). La función save()
    
```

devuelve la publicación guardada en la base de datos. return res.status(200).send({ publication: publicationStored })); Si la publicación se guarda correctamente, devuelve una respuesta con un código de estado 200 OK y un objeto JSON que contiene la publicación guardada. Manejar errores: return res.status(500).send({ message: 'Error al guardar la publicación' })); Si hay un error al guardar la publicación, devuelve una respuesta con un código de estado 500 Internal Server Error y un mensaje de error. En resumen, esta función maneja el proceso de guardar una nueva publicación en la base de datos MongoDB, realizando validaciones, creando una instancia de la publicación, guardando la publicación y respondiendo con el resultado de la operación. Utiliza funciones asíncronas (async/await) para manejar operaciones asíncronas de manera más limpia y legible.



The screenshot shows a code editor window with the title "proyecto-final-master". The left sidebar displays a file tree for a Node.js project named "PROYECTO-FINAL-MASTER". The "api/controllers" folder contains several files: follow.js, publication.js, user.js, authMiddleware.js, models/follow.js, models/message.js, models/publication.js, models/user.js, routes/follow.js, routes/publication.js, routes/user.js, uploads/avatars, and publications. The right pane shows the content of the publication.js file. The code defines an asynchronous function savePublication that takes req and res as parameters. It first checks if params.text is provided; if not, it returns a 400 status with an error message. Then it creates a new Publication object with text, file ('null'), user (req.user.sub), and created_at (moment().unix()). It tries to save the publication and if successful, returns a 200 status with the saved publication. If there's an error during saving, it returns a 500 status with an error message. The module exports an object with probando and savePublication methods.

```
File Edit Selection View Go Run Terminal Help < > proyecto-final-master
EXPLORER
PROYECTO-FINAL-MASTER
api > controllers > publication.js > savePublication
    17
    18     async function savePublication(req, res) {
    19         var params = req.body;
    20
    21         if (!params.text) return res.status(400).send({ message: 'Debes enviar un texto!' });
    22
    23         var publication = new Publication({
    24             text: params.text,
    25             file: 'null',
    26             user: req.user.sub,
    27             created_at: moment().unix()
    28         });
    29
    30         try {
    31             const publicationStored = await publication.save();
    32             return res.status(200).send({ publication: publicationStored });
    33         } catch (error) {
    34             return res.status(500).send({ message: 'Error al guardar la publicación' });
    35         }
    36     }
    37
    38     module.exports = {
    39         probando,
    40         savePublication
    41     }

```

Finalmente, al hacer la petición POST desde Postman, y mandar un texto, nos devuelve el siguiente objeto:

The screenshot shows the Postman application interface. At the top, there is a header bar with tabs for 'POST http://localhost:3800/api/publication' and '[CONFLICT] GET http://localhost:3800/api/publication'. Below the header, the main window has a title 'HTTP http://localhost:3800/api/publication'. The 'Body' tab is selected, showing a table with fields: 'surname' (lopez), 'nick' (Juanele), 'email' (Juan@yopmail.com), 'password' (juan), 'followed' (6632823ddd62b6142cb747ca), and 'text' (Hola que tal estas). The 'text' field has a checked checkbox. The 'Body' section also includes a 'Key' column and a 'Value' column. Below the table, there are tabs for 'Body', 'Cookies', 'Headers (7)', and 'Test Results'. On the right side, status information is displayed: 'Status: 200 OK', 'Time: 256 ms', 'Size: 362 B', and a 'Save Response' button. At the bottom, there are buttons for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'JSON' (which is currently selected).

Key	Value
surname	lopez
nick	Juanele
email	Juan@yopmail.com
password	juan
followed	6632823ddd62b6142cb747ca
text	Hola que tal estas

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 "publication": {  
2     "text": "Hola que tal estas",  
3     "file": "null",  
4     "created_at": "1715535684",  
5     "_id": "6640ff44316797a506965eb7",  
6     "__v": 0  
7 }  
8  
9
```

la publicación se crea en la base de datos:

POST http://localhost:3800/api/publication [CONFLICT] GET http://localhost:3800/api/publication POST http://localhost:3800/api/publication + ***

<http://localhost:3800/api/login>

POST http://localhost:3800/api/publication

Send

Params Authorization Headers (9) Body Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary

Key	Value	Bulk Edit
<input type="checkbox"/> name	juanma	
<input type="checkbox"/> surname	garola	
<input type="checkbox"/> nick	juanmele	
<input type="checkbox"/> email	Juanma@yopmail.com	
<input type="checkbox"/> password	juanma	
<input checked="" type="checkbox"/> text	Hola que tal estas	

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 407 ms Size: 396 B Save Response

Pretty Raw Preview Visualize JSON

```

1
2 "publication": {
3     "text": "Hola que tal estas",
4     "file": "null",
5     "created_at": "1715545998",
6     "user": "664120b2de7ac1c4aeccaf2b",
7     "_id": "6641278e0084413f568609c8",
8     "__v": 0
9 }
10

```

Studio 3T for MongoDB - Full product trial

File Edit Database Collection Index Document GridFS View Help

Connect Collection Intellishell SQL Aggregate Query Profiler Compare Schema Reschema Tasks Export Import Data Masking SQL Migration Users Roles Feedback Go to Free

Enjoy a 10-day full product trial. Once the trial is complete, you will be switched to Studio 3T Free or you can switch now by disabling the trial in My License (in the Help menu).

Open connections: Search open connections (Ctrl+F)

- MongoDB Local localhost:27017 [direct]
 - Base_Datos_Red_Social
 - Collections (3)
 - > follows
 - > publications
 - > users

Quickstart follows publications users

MongoDB Local (localhost:27017) > Base_Datos_Red_Social > publications

Run Load query Save query Query history Set default query Copy Paste AI Helper Visual Query Builder

Query: |||

Projection: |||

Skip: 0 Sort: 0 Limit: 0

Result Query Code Explain

documents 1 to 1

_id	text	file	created_at
6641278e0084413f568609c8	Hola que tal estas	null	1715545998

Match all (Sand) Clear Run

publications > _id

Document JSON Viewer

```

1
2     "_id": ObjectId("6641278e0084413f568609c8"),
3     "text": "Hola que tal estas",
4     "file": "null",
5     "created_at": "1715545998",
6     "user": ObjectId("664120b2de7ac1c4aeccaf2b"),
7     "__v": NumberInt(0)
8

```

1 document selected

6.3. Métodos publicaciones timeline

Básicamente, se implementará la función `getPublications`, que se encargará de recuperar las publicaciones almacenadas en la base de datos y devolverlas como respuesta, junto con información sobre la paginación.

Aquí está el funcionamiento detallado de la función:

- Inicialización de variables: Se establece la página predeterminada en 1 y el número de publicaciones por página en 10 (a través de `itemsPerPage`).
- Verificación de parámetros de página: Se verifica si se proporciona el parámetro de página en la solicitud. Si se proporciona, se convierte a un número entero y se asigna a la variable `page`.
- Recuperación del total de publicaciones: Se utiliza `Publication.countDocuments()` para contar el total de documentos en la colección de publicaciones.
- Cálculo del total de páginas: Se calcula el total de páginas dividiendo el total de publicaciones entre el número de publicaciones por página y redondeando hacia arriba utilizando `Math.ceil()`.
- Verificación de la validez de la página: Se verifica si la página solicitada está dentro del rango válido. Si la página es menor que 1 o mayor que el total de páginas, se devuelve un mensaje de error.
- Recuperación de las publicaciones: Se utiliza `Publication.find()` para obtener las publicaciones de la base de datos. Se utiliza el método `populate()` para poblar el campo `user` con los detalles del usuario asociado a cada publicación. Las publicaciones se ordenan por fecha de creación (`created_at`) en orden descendente y se limitan y omiten según la página actual y el número de publicaciones por página.
- Respuesta: Se devuelve una respuesta con un estado HTTP 200 que contiene el número total de publicaciones, el total de páginas, la página actual y las publicaciones recuperadas.

En resumen, esta función proporciona una forma de paginar las publicaciones, permitiendo a los usuarios obtener un subconjunto de publicaciones por página y navegar entre ellas utilizando los parámetros de página en la URL.

```

    api > controllers > publication.js > getPublications
    49  async function getPublications(req, res) {
    50      let page = 1; // Página por defecto
    51      const itemsPerPage = 10; // Número de publicaciones por página
    52
    53      // Verificar si se proporciona el parámetro de página en la solicitud
    54      if (req.params.page) {
    55          page = parseInt(req.params.page); // Convertir el número de página a entero
    56      }
    57
    58      try {
    59          const count = await Publication.countDocuments(); // Contar el total de publicaciones
    60
    61          // Calcular el total de páginas
    62          const totalPages = Math.ceil(count / itemsPerPage);
    63
    64          // Verificar si la página solicitada está dentro del rango válido
    65          if (page < 1 || page > totalPages) {
    66              return res.status(404).send({ message: 'Página no encontrada' });
    67          }
    68
    69          // Obtener las publicaciones para la página actual
    70          const publications = await Publication.find()
    71              .populate('user')
    72              .sort('-created_at')
    73              .skip((page - 1) * itemsPerPage)
    74              .limit(itemsPerPage)
    75              .exec();
    76
    77
    78          return res.status(200).send({
    79              total_items: count,
    80              pages: totalPages,
    81              page: page,
    82              publications: publications
    83          });
    84      } catch (error) {
    85          return res.status(500).send({ message: 'Error al obtener las publicaciones' });
    86      }
    87  }

```

Adicionalmente, se implementará la ruta para realizar el endpoint de GET

```

    api > routes > publication.js > ...
    1  'use strict';
    2
    3  const express = require('express');
    4  const authMiddleware = require('../middlewares/authMiddleware');
    5  const PublicationController = require('../controllers/publication');
    6
    7  const router = express.Router();
    8
    9  // Utilizar el middleware 'md_auth.ensureAuth' en la ruta '/probando-pub'
    10 router.get('/probando-pub', authMiddleware, PublicationController.probando);
    11 router.post('/publication', authMiddleware, PublicationController.savePublication);
    12 router.get('/publications/:page?', authMiddleware, PublicationController.getPublications);
    13
    14 module.exports = router;
    15

```

Entonces, al lanzar la llamada GET desde Postman, recibiremos las publicaciones que tiene el usuario que está logueado:

The screenshot shows a Postman interface with a GET request to `http://localhost:3800/api/publications/1`. The request body is set to `x-www-form-urlencoded` with the following fields:

Key	Value
surname	garcia
nick	juanmele
email	Juanma@yopmail.com
password	juanma
<input checked="" type="checkbox"/> text	Hola que tal estas

The response status is 200 OK, and the JSON payload is:

```

2   "total_items": 7,
3   "pages": 1,
4   "page": 1,
5   "publications": [
6     {
7       "_id": "66412bbb23b59bed6c0b3849",
8       "text": "Hola que tal estas",
9       "file": "null",
10      "created_at": "1715547067",
11      "user": {
12        "_id": "664120b2de7ac1c4aeccaf2b",
13        "name": "Juanma",
14        "surname": "García",
15        "nick": "juanmele",
16        "email": "Juanma@yopmail.com",
17        "password": "juanma"
18      }
19    }
20  ]
  
```

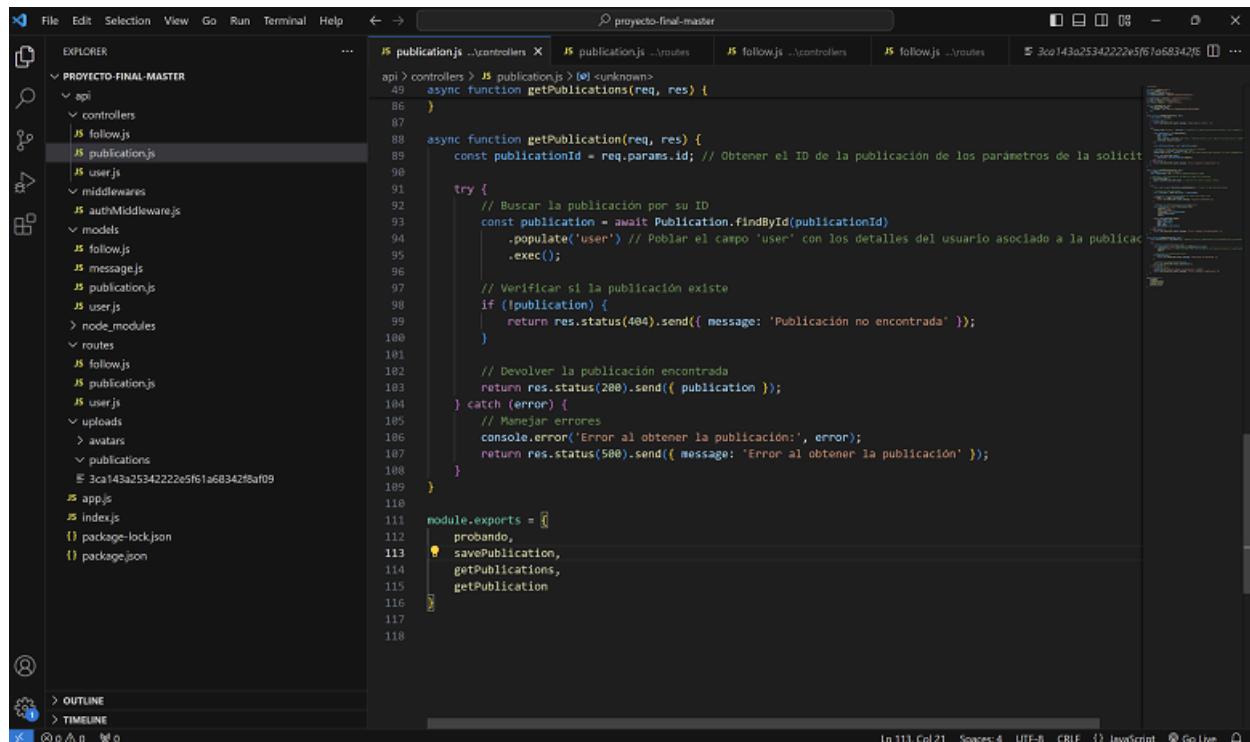
6.4. Devolver una publicación

Se implementará esta función `getPublication`, que busca una publicación en la base de datos en función del ID de la publicación proporcionado en los parámetros de la solicitud (`req.params.id`). Aquí está lo que hace paso a paso:

- Obtener el ID de la publicación:** Extrae el ID de la publicación de los parámetros de la solicitud.
- Buscar la publicación por su ID:** Utiliza el método `Publication.findById()` proporcionado por Mongoose para buscar la publicación en la base de datos utilizando el ID obtenido en el paso anterior.
- Poblar el campo 'user':** Utiliza el método `populate('user')` para incluir los detalles del usuario asociado a la publicación en la consulta. Esto significa que además de obtener los detalles de la publicación, también se recuperarán los detalles del usuario que creó la publicación.

- Verificar si la publicación existe:** Si no se encuentra ninguna publicación con el ID proporcionado, se devuelve un mensaje de error indicando que la publicación no se encontró.
- Devolver la publicación encontrada:** Si se encuentra la publicación, se devuelve un objeto de respuesta con el estado HTTP 200 y la publicación encontrada.
- Manejar errores:** Cualquier error que ocurra durante el proceso, como un error de base de datos o una excepción en el código, se manejará y se devolverá un mensaje de error con el estado HTTP 500. Los detalles específicos del error se registrarán en la consola para facilitar la depuración.

En resumen, esta función se encarga de buscar una publicación por su ID y devolverla junto con los detalles del usuario asociado, si existe.



```

File Edit Selection View Go Run Terminal Help < > proyecto-final-master
EXPLORER
PROYECTO-FINAL-MASTER
api
  controllers
    follow.js
    publications.js
    user.js
  middlewares
    authMiddleware.js
  models
    follow.js
    message.js
    publication.js
    user.js
  node_modules
  routes
    follow.js
    publications.js
    user.js
  uploads
    avatars
    publications
      3ca143a2534222e5f61a68342f8af09
  app.js
  index.js
  package-lock.json
  package.json

JS publication.js ..\controllers x JS publication.js ..\routes JS follow.js ..\controllers JS follow.js ..\routes 3ca143a2534222e5f61a68342f8af09

api > controllers > JS publication.js > [x] <unknown>
49  async function getPublications(req, res) {
86  }
87
88  async function getPublication(req, res) {
89    const publicationId = req.params.id; // Obtener el ID de la publicación de los parámetros de la solicitud
90
91    try {
92      // Buscar la publicación por su ID
93      const publication = await Publication.findById(publicationId)
94        .populate('user') // Poblar el campo 'user' con los detalles del usuario asociado a la publicación
95        .exec();
96
97      // Verificar si la publicación existe
98      if (!publication) {
99        return res.status(404).send({ message: 'Publicación no encontrada' });
100
101      // Devolver la publicación encontrada
102      return res.status(200).send({ publication });
103    } catch (error) {
104      // Manejar errores
105      console.error('Error al obtener la publicación:', error);
106      return res.status(500).send({ message: 'Error al obtener la publicación' });
107    }
108  }
109
110  module.exports = [
111    probando,
112    savePublication,
113    getPublications,
114    getPublication
115  ];
116
117
118

```

Paralelamente, se incluye la ruta en el archivo 'publications.js' de la carpeta 'routes'

The screenshot shows a code editor interface with the title bar "proyecto-final-master". The left sidebar displays the project structure:

```

PROJECT-FINAL-MASTER
  - api
    - controllers
      - follow.js
      - publication.js
      - user.js
    - middlewares
      - authMiddleware.js
    - models
      - follow.js
      - message.js
      - publication.js
      - user.js
    - routes
      - node_modules
      - routes
        - follow.js
        - publications.js
        - user.js
      - uploads
        - avatars
        - publications
          - 3ca143a2534222e5f61a683428af09
    - app.js
    - index.js
  - package-lock.json
  - package.json

```

The main editor area shows the content of the `publication.js` file:

```

1  'use strict';
2
3  const express = require('express');
4  const authMiddleware = require('../middlewares/authMiddleware');
5  const PublicationController = require('../controllers/publication');
6
7  const router = express.Router();
8
9  // Utiliza el middleware 'md_auth.ensureAuth' en la ruta '/probando-pub'
10 router.get('/probando-pub', authMiddleware, PublicationController.probando);
11 router.post('/publication', authMiddleware, PublicationController.savePublication);
12 router.get('/publications/:page', authMiddleware, PublicationController.getPublications);
13 router.get('/publication/:id', authMiddleware, PublicationController.getPublication); // Ruta para obtener una publicación
14
15 module.exports = router;
16

```

Entonces, al hacer el GET y poner un id de la publicación deseada, obtenemos lo siguiente, desde Postman:

The screenshot shows a Postman collection with the following items:

- POST http://localhost:3800/api/login**: [CONFLICT] (Status: 409)
- GET http://localhost:3800/api/publication/66412bb723b59bed6c0b3846**: (Status: 200 OK)

The "GET" request details are as follows:

- Method**: GET
- URL**: `http://localhost:3800/api/publication/66412bb723b59bed6c0b3846`
- Headers** (9):

Key	Value
Authorization	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfc2VrX2lkIjoiNjY0MTIwYjJkZ... (A long JWT token)
Key	Value
- Body**: (Pretty, Raw, Preview, Visualize, JSON)
- Test Results**: Status: 200 OK, Time: 124 ms, Size: 601 B

The response body is displayed in "Pretty" format:

```

1
2   "publication": {
3     "_id": "66412bb723b59bed6c0b3846",
4     "text": "Hola que tal estas",
5     "file": "null",
6     "created_at": "1715547063",
7     "user": {
8       "_id": "664120b2de7ac1c4aeccaf2b",
9       "name": "juanma",
10      "surname": "garcia",
11      "nick": "juanmele",
12      "email": "juanma@correo.ugr.es"
13    }
14  }

```

6.5. Eliminar publicaciones

En este caso, se implementará la función ‘deletePublication’, que va a realizar los siguientes pasos:

- **Obtención del ID de la publicación:** Extrae el ID de la publicación de los parámetros de la solicitud.
 - **Búsqueda y eliminación de la publicación:** Utiliza el método `Publication.findByIdAndDelete()` proporcionado por Mongoose para buscar la publicación en la base de datos utilizando el ID obtenido y eliminarla.
 - **Verificación de la existencia de la publicación:** Si no se encuentra ninguna publicación con el ID proporcionado, se devuelve un mensaje de error indicando que la publicación no se encontró.
 - **Mensaje de éxito:** Si se elimina correctamente la publicación, se devuelve un objeto de respuesta con el estado HTTP 200 y un mensaje indicando que la publicación se eliminó correctamente.

Esta función está lista para ser exportada junto con las demás funciones del controlador de

De igual manera, se tiene que incluir la ruta ligada a la función anterior:

```

api > routes > publication.js > ...
1  'use strict';
2
3  const express = require('express');
4  const authMiddleware = require('../middlewares/authMiddleware');
5  const PublicationController = require('../controllers/publication');
6
7  const router = express.Router();
8
9  // Utiliza el middleware 'md_auth.ensureAuth' en la ruta '/probando-pub'
10 router.get('/probando-pub', authMiddleware, PublicationController.probando);
11 router.post('/publication', authMiddleware, PublicationController.savePublication);
12 router.get('/publications/:page?', authMiddleware, PublicationController.getPublications);
13 router.get('/publication/:id', authMiddleware, PublicationController.getPublication); // Ruta para obtener
14 router.delete('/publication/:id', PublicationController.deletePublication); // Ruta para eliminar una publi
15
16 module.exports = router;
17

```

Y para cerrar este método, se hacen las pruebas en Postman, intentando eliminar una publicación particular con el ID:

Key	Value
Authorization	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoiNjY0MTIwYjkZ...
Key	Value

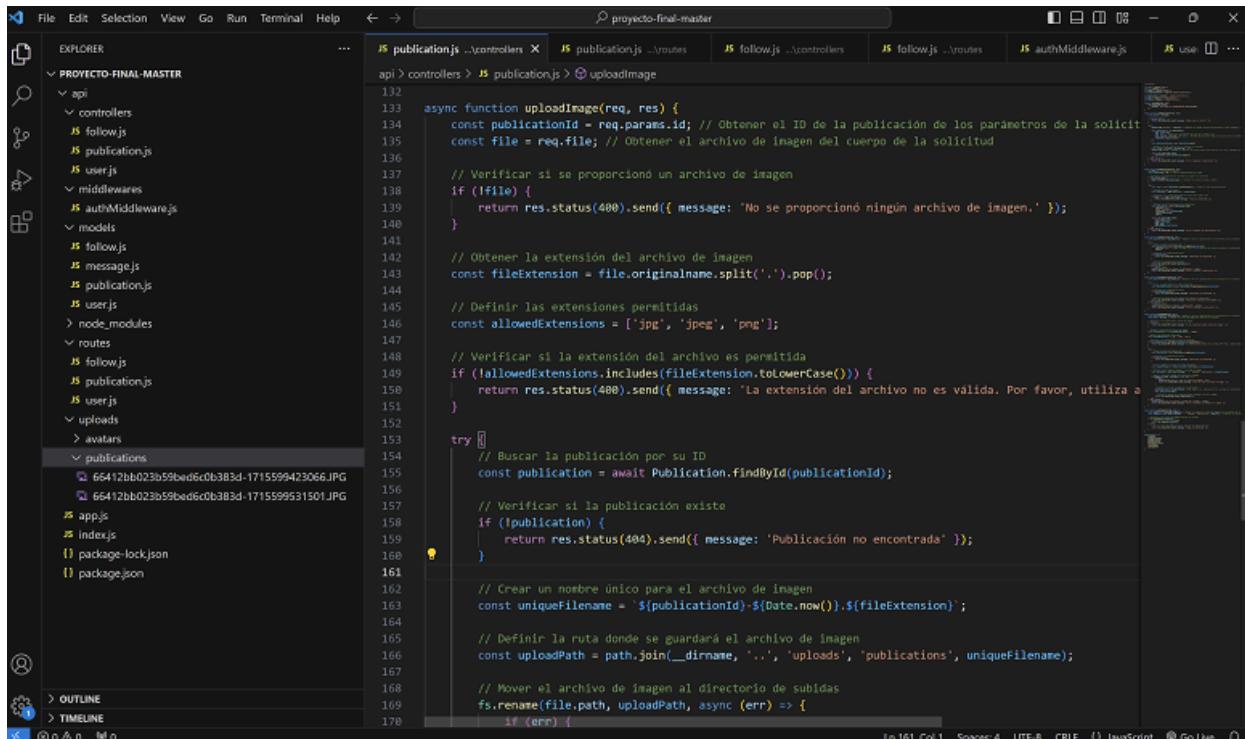
Body: {"message": "Publicación eliminada correctamente"}

6.6. Subir ficheros en las publicaciones

Se implementará la función uploadImage, que maneja la subida de archivos de imagen asociados a una publicación. Aquí está el flujo de la función:

- **Obtención de parámetros:** Se obtiene el ID de la publicación desde los parámetros de la solicitud (req.params.id) y el archivo de imagen desde el cuerpo de la solicitud (req.file).
- **Validación del archivo de imagen:** Se verifica si se proporcionó un archivo de imagen. Si no se proporciona, se devuelve un mensaje de error.
- **Validación de la extensión del archivo:** Se obtiene la extensión del archivo de imagen y se verifica si es una extensión permitida (JPG, JPEG o PNG). Si no es una extensión válida, se devuelve un mensaje de error.
- **Búsqueda de la publicación:** Se busca la publicación correspondiente al ID proporcionado en la base de datos.
- **Creación de un nombre único para el archivo:** Se crea un nombre único para el archivo de imagen, que incluye el ID de la publicación y la marca de tiempo actual.
- **Definición de la ruta de almacenamiento:** Se define la ruta donde se guardará el archivo de imagen en el servidor.
- **Movimiento del archivo de imagen al directorio de subidas:** Se mueve el archivo de imagen desde su ubicación temporal (donde se almacena temporalmente por multer) a la ubicación definitiva en el servidor.
- **Actualización de la publicación con la ruta de la imagen:** Se actualiza la propiedad file de la publicación con el nombre único del archivo de imagen.
- **Respuesta con el objeto de la publicación actualizado:** Se devuelve un objeto JSON con el detalle de la publicación actualizada, que incluye la URL del archivo de imagen.

En caso de que ocurra algún error durante este proceso, se maneja adecuadamente devolviendo un mensaje de error y un código de estado HTTP 500 para indicar un error interno del servidor



```
132
133     async function uploadImage(req, res) {
134         const publicationId = req.params.id; // Obtener el ID de la publicación de los parámetros de la solicitud
135         const file = req.file; // Obtener el archivo de imagen del cuerpo de la solicitud
136
137         // Verificar si se proporcionó un archivo de imagen
138         if (!file) {
139             return res.status(400).send({ message: 'No se proporcionó ningún archivo de imagen.' });
140         }
141
142         // Obtener la extensión del archivo de imagen
143         const fileExtension = file.originalname.split('.').pop();
144
145         // Definir las extensiones permitidas
146         const allowedExtensions = ['jpg', 'jpeg', 'png'];
147
148         // Verificar si la extensión del archivo es permitida
149         if (!allowedExtensions.includes(fileExtension.toLowerCase())) {
150             return res.status(400).send({ message: 'La extensión del archivo no es válida. Por favor, utiliza una de las extensiones permitidas.' });
151         }
152
153     try {
154         // Buscar la publicación por su ID
155         const publication = await Publication.findById(publicationId);
156
157         // Verificar si la publicación existe
158         if (!publication) {
159             return res.status(404).send({ message: 'Publicación no encontrada' });
160         }
161
162         // Crear un nombre único para el archivo de imagen
163         const uniqueFilename = `${publicationId}-${Date.now()}.${fileExtension}`;
164
165         // Definir la ruta donde se guardará el archivo de imagen
166         const uploadPath = path.join(__dirname, '..', 'uploads', 'publications', uniqueFilename);
167
168         // Mover el archivo de imagen al directorio de subidas
169         fs.rename(file.path, uploadPath, async (err) => {
170             if (err) {
```

Consecuentemente, al realizar el POST desde Postman para subir una imagen en una publicación determinada, la imagen se lanza correctamente a la base de datos y VSCode:

```

async function getCounters(req, res) {
    let userId = req.user._id; // Obtener el ID del usuario autenticado por defecto

    // Verificar si se proporcionó un ID de usuario en los parámetros de la URL
    if (req.params.id) {
        userId = req.params.id;
    }

    try {
        // Obtener el número de usuarios que seguimos
        const followingCount = await Follow.countDocuments({ user: userId });

        // Obtener el número de usuarios que nos siguen
        const followerCount = await Follow.countDocuments({ followed: userId });

        // Obtener el número de publicaciones del usuario
        const publicationCount = await Publication.countDocuments({ user: userId });

        res.status(200).send({
            followingCount,
            followerCount,
            publicationCount
        });
    } catch (err) {
        console.error('Error al obtener contadores:', err);
        res.status(500).send({ message: 'Error en la petición' });
    }
}

```

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows the project structure under "PROYECTO-FINAL-MASTER". The "message.js" file is selected in the list.
- Editor Area:** Displays the code for "message.js" which defines a Mongoose schema for a "Message" document.
- Code Content:**

```

api > models > message.js > MessageSchema
1  'use strict';
2
3  var mongoose = require('mongoose');
4  var Schema = mongoose.Schema;
5
6  var MessageSchema = Schema ({
7      text: String,
8      viewed: String,
9      created_at: String,
10     emitter: { type: Schema.ObjectId, ref:'User' },
11     receiver: { type: Schema.ObjectId, ref:'User' },
12 });
13
14 module.exports = mongoose.model('Message', MessageSchema);

```

The screenshot shows a code editor interface with the following details:

- File Explorer:** On the left, it shows a tree view of the project structure under "PROYECTO-FINAL-MASTER". The "message.js" file is selected and highlighted.
- Code Editor:** The main area displays the content of the "message.js" file, which is part of the "api" routes. The code uses Express.js and Multer middleware to handle file uploads.

```

    1  'use strict';
    2
    3  const express = require('express');
    4  const multer = require('multer');
    5  const authMiddleware = require('../middlewares/authMiddleware');
    6  const MessagesController = require('../controllers/message');
    7  const path = require('path');
    8
    9  const router = express.Router();
   10
   11  router.get('/probando-md', authMiddleware, MessagesController.probando);
   12
   13  module.exports = router;
  
```

The screenshot shows the Postman application interface with the following details:

- Request URL:** `http://localhost:3800/api/login`
- Method:** POST
- Headers:** A table showing the "Authorization" header is checked and contains the value: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJtc2VyX2lkIjoiNjY0MTIwYjkZ...`
- Body:** The body tab is visible but empty.
- Tests:** The tests tab is visible.
- Settings:** The settings tab is visible.

The screenshot shows the Postman application interface with the following details:

- Response Status:** 200 OK
- Response Headers:** Status: 200 OK, Time: 306 ms, Size: 434 B
- Response Body:** A JSON object representing a publication document.

```

1  {
2     "publication": {
3         "_id": "66412bb023b59bed6c0b383d",
4         "text": "Hola que tal estas",
5         "file": "66412bb023b59bed6c0b383d-1715599531501.JPG",
6         "created_at": "1715547056",
7         "user": "66412bb2de7ac1c4aeccaf2b",
8         "__v": 0
9     }
10
  
```

The screenshot shows the Studio 3T interface for MongoDB. On the left, the 'Open connections' sidebar lists 'MongoDB Local localhost:27017 [direct]' with its database structure: 'Base_Datos_Red_Social' containing 'Collections (3)' (follows, publications, users), 'GridFS Buckets (0)', 'System (0)', and 'Views (0)'. Below this are sections for 'My resources' (Quick Share requests, Shared folder invites, Local resources: My connections (1), MySQL connections (0), My queries (0)) and 'Operations'.

The main workspace has tabs for 'Quickstart', 'follows', 'publications', and 'users'. The 'publications' tab is active, showing a query results grid. The query is:

```
Match all ($and)
  + Drag and drop field here or double-click
```

The results show 5 documents:

	text	file	crea
[6641278e0084413156609c8]	Hola que tal estas	null	
[66412bb023b59bed6c0b383a]	Hola que tal estas	null	
[66412bb023b59bed6c0b383d]	Hola que tal estas	66412bb023b59...	
[66412bb223b59bed6c0b3840]	Hola que tal estas	null	
[66412bb223b59bed6c0b3843]	Hola que tal estas	null	
[66412bb223b59bed6c0b3849]	Hola que tal estas	null	

Below the grid, it says '1 document selected' and 'Count Documents'.

The screenshot shows a Microsoft Edge browser window with the title 'proyecto-final-master'. The page displays a psychology test with multiple-choice questions and handwritten answers. The test includes sections like 'PARTES TEÓRICA' and 'PREGUNTAS'. Handwritten notes include '7/3 (1)' at the bottom right and 'MOCICO' near the top center. The browser interface shows tabs for 'publication.js ...controllers', 'publication.js ...routes', 'follow.js ...controllers', and 'follow.js ...routes'. The left sidebar shows the project structure: 'PROYECTO-FINAL-MASTER' with 'controllers', 'middlewares', 'models', 'routes', and 'uploads' subfolders, along with files like 'app.js', 'index.js', 'package-lock.json', and 'package.json'.

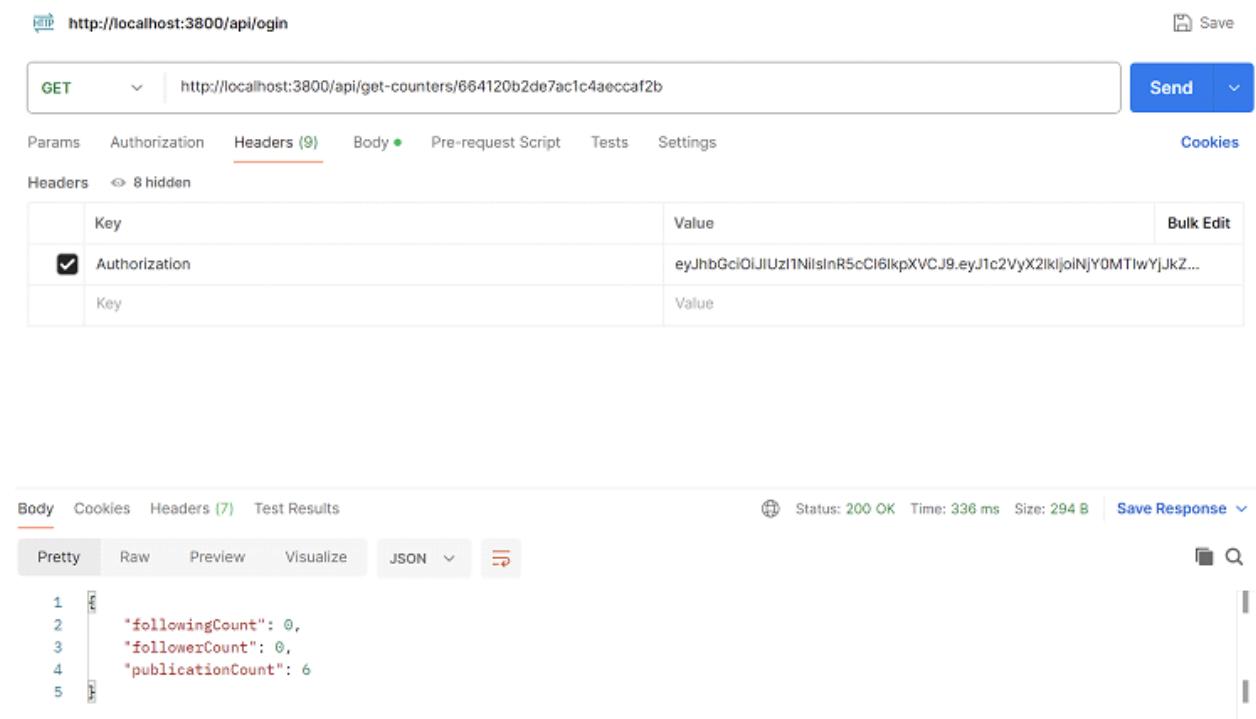
6.6. Número total de publicaciones

Utilizando el modelo Publication, se cuenta el número de documentos que tienen como campo user el ID del usuario en cuestión. Esto proporciona el número total de

publicaciones que ha realizado ese usuario. Entonces, se aplica esto a la función `getCounters`:

De igual manera, al hacer el GET desde Postman, nos dará el número de publicaciones que un usuario contiene en la base de datos:

7. Mensajería privada



The screenshot shows a Postman request configuration and its resulting response.

Request URL: `http://localhost:3800/api/get-counters/664120b2de7ac1c4aeccaf2b`

Headers (9):

Key	Value
Authorization	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoiNjY0MTIwYjJkZ... [REDACTED]
Key	[REDACTED]

Response Body (Pretty):

```
1
2   "followingCount": 0,
3   "followerCount": 0,
4   "publicationCount": 6
5
```

7.1. Mejoras en el modelo Message

Aquí, básicamente se va a crear un nuevo campo llamado ‘viewed’

7.2. Crear controlador, acciones y rutas

Se crea el siguiente controlador llamado ‘message.js’:

Además, también es necesario la ruta: ‘message.js’

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure under "PROYECTO-FINAL-MASTER".
- Editor:** Displays the content of `message.js` in the "api/controllers" folder.
- Code:** The code in `message.js` is as follows:

```
api > controllers > JS message.js <(0)> unknown>
1 'use strict'
2
3 var moment = require('moment');
4 var mongoosePaginate = require('mongoose-pagination');
5
6 var User = require('../models/user');
7 var Follow = require('../models/follow');
8 var Message = require('../models/message');
9
10 function probando(req, res){
11   res.status(200).send({message: 'Hola que tal'});
12 }
13
14 module.exports = {
15   probando
16 }
```

Y por último, se hará la prueba desde Postman que está funcionando de manera correcta:

The screenshot shows the Postman interface with the following details:

- Request URL:** `http://localhost:3800/api/probando-md`
- Method:** GET
- Headers:** Authorization (with value eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ9.eyJcI2Vx2lkjoiNjY0MTIwYJKZ...)
- Body:** JSON response containing `{"message": "Hola que tal"}`
- Status:** 200 OK
- Time:** 88 ms
- Size:** 261 B

7.3. Enviar mensajes

Vamos a crear el primer método de esta sección el primero de todos se va llamar saveMessage, será el encargado de poder enviar los mensajes entre los usuarios.

Para ello creamos esta nueva función de tal manera que le pasaremos los parámetro correspondientes:

```
function saveMessage(re,res){
    let params = req.body;

    if(!params.text || !params.receiver) res.status(200).send({message:'Envia los datos necesarios'});

    let message = new MessageChannel();
    message.emmite = req.user.sub;
    message.receiver = params.receiver;
    message.text = params.text;
    message.create_at = moment().unix();

    message.save((err,messageStored) =>{
        if(err) return res.status(500).send({message:'Error en la peticon'});
        if(!messageStored) return res.status(500).send({message: 'Error al enviar el mensaje'});

        return res.status(200).send({message: messageStored});
    });
}
```

Por último la exportamos abajo:

```
module.exports = [
    probando,
    saveMessage
]
```

Y ahora creamos la hoja de ruta:

```

api > routes > JS message.js > ...
    ⚡ Click here to ask Blackbox to help you code faster
1  'use strict'
2
3  const express = require('express');
4  const MessageController = require('../controllers/message');
5  const md_auth = require('../middlewares/authenticated');
6
7  const api = express.Router();
8
9  api.get('/probando-md', md_auth.ensureAuth, MessageController.probando);
10 api.post('/message', md_auth.ensureAuth, MessageController.saveMessage);
11
12

```

Después vamos a hacer una prueba desde postman, vamos a crear un token nuevo, y nos iremos a nuestra base de datos y cogeremos el id del usuario que queremos envíe el mensajes y colocaremos el token del usuario al que queremos que se lo envie, tendremos que pasarle en el headers el token y en el body el texto que nosotros deseamos y en el receiver el id del usuario que queremos que envíe el mensaje:

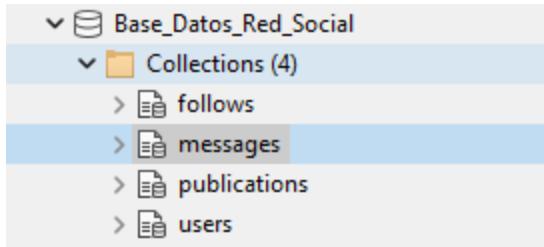
Key	Value	Description
text	Hola que tal estas Inma	
receiver	663ceecf6032d06f24...	

```

1 {
2   "message": {
3     "_id": "6644815ec6328f32abc09191",
4     "emitter": "663b8ab864a7fdc087d25149",
5     "receiver": "663ceecf6032d06f24afc7b9",
6     "text": "Hola que tal estas Inma",
7     "__v": 0
8   }
9 }

```

Ahora si nos vamos a nuestra base de datos, tendremos otra nueva colección donde aparece la nueva sección creada de mensajes:



Y dentro de ella tendremos los mensajes enviados:

A screenshot of a 'Document JSON Viewer' window. The title bar says 'Document JSON Viewer'. The main area shows a single JSON document with the following content:

```
1 {  
2   "_id" : ObjectId("6644815ec5328f32abc09191"),  
3   "emitter" : ObjectId("663b8ab864a7fdcd87d25149"),  
4   "receiver" : ObjectId("663ceecf6032d06f24afc7b9"),  
5   "text" : "Hola que tal estas Inma",  
6   "__v" : NumberInt(0)  
7 }
```

The code editor has syntax highlighting for JSON, with colors for different types like ObjectId and NumberInt.

7.4. Mensajes enviados

Este método que vamos a crear para nuestra api rest este nos permitirá listar los mensajes recibidos.

Para ello creamos una nueva función llamada `getReceiverMessage()`. Este se encargará de otorgar un listado paginado de todos los mensajes recibidos en nuestra cuenta de usuario.

```

function getReceivedMessage(req, res){
    let userId = req.user.sub;

    let page= 1;
    if(req.params.page){
        page = req.params.page;
    }

    let itemsPerPage = 4;

    Message.find({receiver: userId}).populate('emitter').paginate(page, itemsPerPage)
    .then((messages) => {
        if(!messages) return res.status(404).send({message: 'No hay mensajes'});
        return res.status(200).send({
            total: messages.length,
            pages: Math.ceil(messages.length/itemsPerPage),
            messages: messages
        });
    })
    .catch((err) => {
        return res.status(500).send({message: 'Error en la peticion'});
    });
}

```

Ahora exportamos abajo del todo el modelo:

```

module.exports = {
    probando,
    saveMessage,
    getReceivedMessage
}

```

Y nos vamos a nuestra hoja de ruta y la creamos a través del método GET:

```

api.get('/my-messages', md_auth.ensureAuth, MessageController.getReceivedMessage);

```

Por último, nos logueamos con el usuario al cual anteriormente le hemos enviados los mensajes y listamos los mensajes recibidos, para ello haremos una comprobación con nuestro Postman:

http://localhost:3800/api/my-messages

Save Share </>

GET http://localhost:3800/api/my-messages

Params Authorization Headers (8) Body Scripts Settings Cookies

Headers (7 hidden)

Key	Value	Description	... Bulk Edit Presets
<input checked="" type="checkbox"/> Authorization	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI2NjM4ZjgxMjlkZTdjMDJl...		
Key	Value	Description	

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```

1 {
2   "total": 4,
3   "pages": 1,
4   "messages": [
5     {
6       "_id": "6644864a10f4023f036ef487",
7       "emitter": {
8         "_id": "663b8ab864a7fdcd87d25149",
9         "name": "inma",
10        "surname": "corcoles",
11        "nick": "immacorcoles",
12        "email": "inma@inma.com",
13        "password": "$2a$10$q86neJ70zs306qNkJ6Avi.04bD3dc.mRJDC2jUCAKybqQo8h3fFmm",
14        "role": "ROLE_USER",
15        "image": null,
16        "__v": 0
17      },
18      "receiver": "6638f8129de7c02e2271a6b8",
19      "text": "hola admin soy inma123456",
20      "__v": 0
21    },
22    {
23      "_id": "6644864a10f4023f036ef489",
24      "emitter": {
25        "_id": "663b8ab864a7fdcd87d25149",
26        "name": "inma",
27        "surname": "corcoles",
28        "nick": "immacorcoles",
29        "email": "inma@inma.com",
30        "password": "$2a$10$q86neJ70zs306qNkJ6Avi.04bD3dc.mRJDC2jUCAKybqQo8h3fFmm",
31        "role": "ROLE_USER",
32        "image": null,
33        "__v": 0
34      },
35      "receiver": "6638f8129de7c02e2271a6b8",
36      "text": "hola admin soy inma12345",
37      "__v": 0
38    },
39    {
40      "_id": "6644865110f4023f036ef48b",
41      "emitter": {
42        "_id": "663b8ab864a7fdcd87d25149",
43        "name": "inma",

```

Status: 200 OK Time: 10 ms Size: 1.65 KB Save as example

Postbot Runner Start Proxy Cookies Vault Trash

```

40      "_id": "6644865110f4023f036ef48b",
41      "emitter": {
42          "_id": "663b8ab864a7fdcd87d25149",
43          "name": "inma",
44          "surname": "corcoles",
45          "nick": "inmacorcoles",
46          "email": "inma@inma.com",
47          "password": "$2a$10$q86neJ70zs306qNkJ6Avi.04bD3dc.mRJDC2jUCAKybqQo8h3fFmm",
48          "role": "ROLE_USER",
49          "image": null,
50          "__v": 0
51      },
52      "receiver": "6638f8129de7c02e2271a6b8",
53      "text": "hola admin soy inma1234",
54      "__v": 0
55  ],
56  {
57      "_id": "6644865310f4023f036ef48d",
58      "emitter": {
59          "_id": "663b8ab864a7fdcd87d25149",
60          "name": "inma",
61          "surname": "corcoles",
62          "nick": "inmacorcoles",
63          "-----"

```

7.5. Devolver campos específicos con Mongoose

Es importante tener en cuenta que cuando nosotros enviamos y listamos los mensajes, muchos datos de ellos, quizás sería bueno emitirlos, como es el caso de la password, el email, el role.

Para ello en el populate, podemos describir los campos que nosotros queremos que se emitan, sería de esta manera:

```
Message.find({receiver: userId}).populate('emitter','name surname _id').paginate(page, itemsPerPage)
```

Y entonces solo nos muestra los descritos, para ello lo comprobamos desde postman:

The screenshot shows a JSON response in a browser's developer tools. The response is a single object with the following structure:

```

1  {
2      "total": 3,
3      "pages": 1,
4      "messages": [
5          {
6              "_id": "6644865510f4023f036ef48f",
7              "emitter": {
8                  "_id": "663b8ab864a7fdcd87d25149",
9                  "name": "inma",
10                 "surname": "corcoles"
11             },
12             "receiver": "6638f8129de7c02e2271a6b8",
13             "text": "hola admin soy inma12",
14             "__v": 0
15         }

```

7.6. Listado de mensajes enviados

Ya tenemos un método que nos lista los mensajes que hemos recibido, por lo tanto, vamos hacer un método que nos liste los mensajes que nosotros hemos enviado.

Es muy parecido al método anterior:

```

function getEmmitMessage(req, res){
    let userId = req.user.sub;

    let page= 1;
    if(req.params.page){
        page = req.params.page;
    }

    let itemsPerPage = 4;

    Message.find({emitter: userId}).populate('emitter receiver','name surname _id nick image').paginate(page, itemsPerPage)
    .then((messages) => {
        if(!messages) return res.status(404).send({message: 'No hay mensajes' });
        return res.status(200).send({
            total: messages.length,
            pages: Math.ceil(messages.length/itemsPerPage),
            messages: messages
        });
    })
    .catch((err) => {
        return res.status(500).send({message: 'Error en la peticion' });
    });
}

```

Exportamos abajo:

```
module.exports = [
  probando,
  saveMessage,
  getReceivedMessage,
  getEmmitMessage
]
```

Y creamos la ruta en el archivo routes:

```
api.get('/messages/:page?', md_auth.ensureAuth, MessageController.getEmmitMessage);
```

Y ahora nos situamos en nuestro postman y nos logueamos con el usuario que ha enviado mensajes:

The screenshot shows a Postman interface with a GET request to `http://localhost:3800/api/messages`. The Headers tab is selected, showing an `Authorization` header with a value of a JWT token. The Body tab shows a JSON response with the following structure:

```
{
  "total": 4,
  "pages": 1,
  "messages": [
    {
      "emitter": {
        "_id": "6644816ec5328f32abc09191",
        "name": "Inma",
        "surname": "corcoles",
        "nick": "inmacorcoles",
        "image": null
      },
      "receiver": {
        "_id": "663ceecf6032d06f24afc7b9",
        "name": "jesus",
        "surname": "Lopez",
        "nick": "jesuslopez",
        "image": null
      },
      "text": "Hola que tal estas Inma",
      "_v": 0
    }
  ]
}
```

The status bar at the bottom indicates a 200 OK response with a size of 1.46 KB.

7.7. Contar los mensajes sin leer

Vamos a crear un método para leer los mensajes no leídos con el viewed, que es la propiedad que tienen los mensajes:

```
✓ function getUnViewedMessages(req, res){  
  let userId = req.user.sub;  
  
  Message.countDocuments({receiver: userId, viewed:'false'})  
  .then((count) => {  
    if(err) return res.status(500).send({message: 'Error en al peticion'});  
    return res.status(200).send({  
      'unviewed': count  
    });  
  })  
  .catch((err) => {  
    return res.status(500).send({message: 'Error en al peticion'});  
  });  
}
```

Lo exportamos abajo:

```
module.exports = {  
  probando,  
  saveMessage,  
  getReceivedMessage,  
  getEmmitMessage,  
  getUnViewedMessages  
}
```

Creamos la ruta:

```
api.get('/unviewed-messages', md_auth.ensureAuth, MessageController.getUnViewedMessages);
```

7.8. Marcar mensajes como leídos

Vamos a hacer un método que se encargue de actualizar todos los documentos de la colección de mensajes que nosotros hayamos creado cuyo userId sea el nuestro y actualizar el flap viewer a true.

Sería de la siguiente manera, creando una función nueva, pasandole el received a true:

```
function setViewedMessages(req, res) {
  let userId = req.user.sub;

  Message.updateMany({ receiver: userId, viewed: 'false' }, { viewed: 'true' })
    .then((messageUpdate) => {
      res.status(200).send({ message: messageUpdate });
    })
    .catch((err) => {
      res.status(500).send({ message: 'Error en la petición' });
    });
}

endCode
```

Exportamos abajo este nuevo método:

```
module.exports = {
  probando,
  saveMessage,
  getReceivedMessage,
  getEmmitMessage,
  getUnViewedMessages,
  setViewedMessages
}
```

Y creamos la ruta:

```
api.get('/set-viewed-messages', md_auth.ensureAuth, MessageController.setViewedMessages);
```

Comprobamos por postman para ver si funciona:

The screenshot shows a POSTMAN interface with the following details:

- Method:** GET
- URL:** http://localhost:3800/api/set-viewed-message
- Headers:** Authorization (with value eyJ0eXAiOiJKV1QiLCJ...)
- Body:** JSON response (Pretty) showing the following message object:

```
1 {  
2   "message": {  
3     "acknowledged": true,  
4     "modifiedCount": 0,  
5     "upsertedId": null,  
6     "upsertedCount": 0,  
7     "matchedCount": 0  
8   }  
9 }
```
- Status:** 200 OK (15 ms, 340 B)

7.9. Configurar cabeceras HTTP y acceso CORS

Para terminar el backend de momento vamos a configurar las cabeceras para permitir el acceso CORS y así evitarnos los típicos problemas a la hora de hacer peticiones AJAX, de JavaScript o de cualquier fronted.

Para ello abrimos el archivo app.js y en el apartado cors, vamos a configurar todas esas cabeceras.

```
// cors
app.use((req, res, next) => {
  res.header('Access-Control-Allow-Origin', '*');
  res.header('Access-Control-Allow-Headers', 'Authorization, X-API-KEY, Origin, X-Requested-With, Content-Type, Accept, Access-Control-Allow-Request-Method');
  res.header('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, DELETE');
  res.header('Allow', 'GET, POST, OPTIONS, PUT, DELETE');

  next();
});
```

Y así no tendremos problemas de cara a futuro en ningún frontend.

8.0. Empezando el frontend con Angular

Ahora vamos a empezar a desarrollar la parte del frontend, de momento a nivel de backend tenemos una api Rest o servicio Rest Full que nos permite registrar nuevos usuarios, loguearnos, editar sus datos tenemos un sistema en el cual podemos seguir a un usuario, dejarlo de seguir y cualquier usuario se puede seguir entre sí.

Tenemos también una parte en la que podemos guardar nuevas publicaciones, listar todas esas publicaciones de manera paginada para construir posteriormente una timeline, un controlador de mensajes privados que nos permite enviar mensajes privados, listar esos mensajes, etc, es decir, un servicio Rest de backend completamente funcional para hacer una red social y ahora vamos hacer la parte de frontend.

Entonces la parte de frontend la vamos hacer con angular y ya vamos hacer que visualmente tengamos una web y vayamos haciendo peticiones al servicio Rest a la api, al backend, este se va a encargar de guardar las cosas en la base de datos y hacer todas las operaciones necesarias de backend y con la parte del frontend solo tenemos que interpretar los datos que nos envía el backend y pues enviarle información a la api.

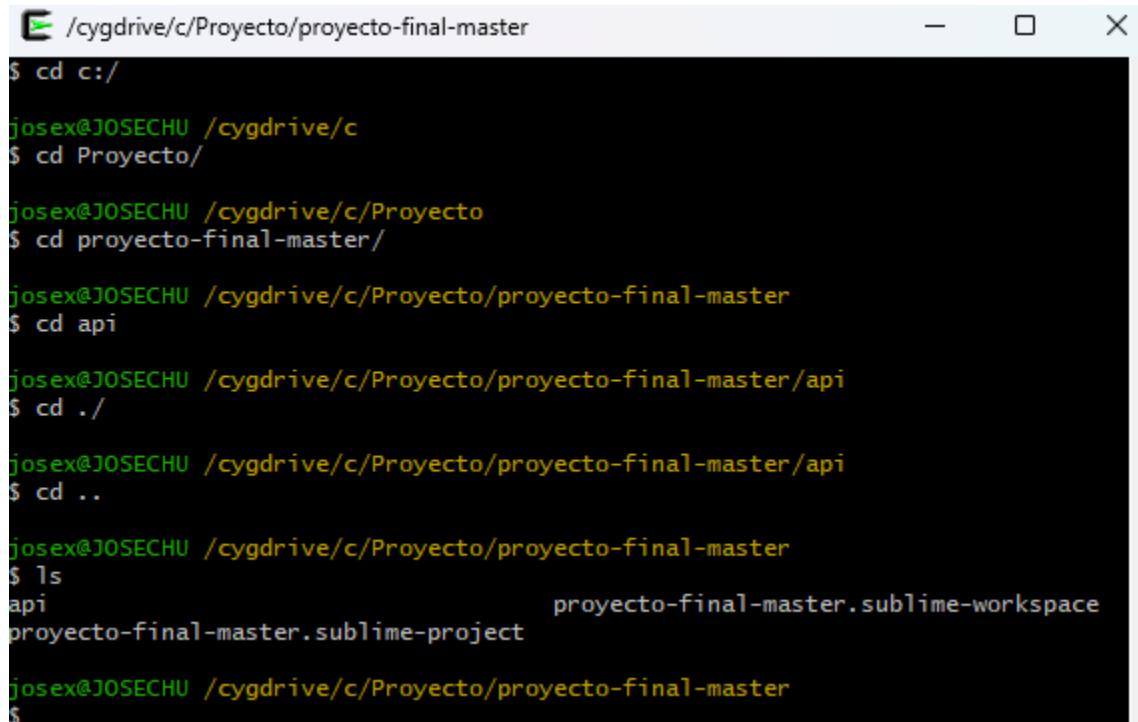
Por lo tanto, al final tendremos una web spa con angular, es decir, una página web que no va a recargar la página en ningún momento, que va a ser completamente dinámica, asíncrona y reactiva.

Con asíncrona nos referimos a que va estar haciendo peticiones AJAX al servidor continuamente con lo cual el usuario no se va a enterar de que por hay un backend, simplemente le va a dar clic a una sección y eso va a cargar de manera muy rápida.

Y con reactiva nos referimos a que nosotros cuando hacemos cualquier tipo de cambio ya sea a nivel de URL, saltar de una página a otra, como a nivel de llenar un formulario los datos se cambian de manera instantánea y reactiva, es decir, que si yo modiflico un valor de una propiedad de un componente en una vista eso también va a

cambiar el modelo de dato y todo es super dinámico y super reactivo, cuando yo hago un cambio se cambia el valor de todo donde se sitúe esa propiedad.

Para comenzar, lo que tenemos que hacer es crear un nuevo proyecto de angular dentro de nuestro carpeta donde tenemos la api, para ello, abrimos una consola y nos situamos donde se encuentra la carpeta del proyecto:



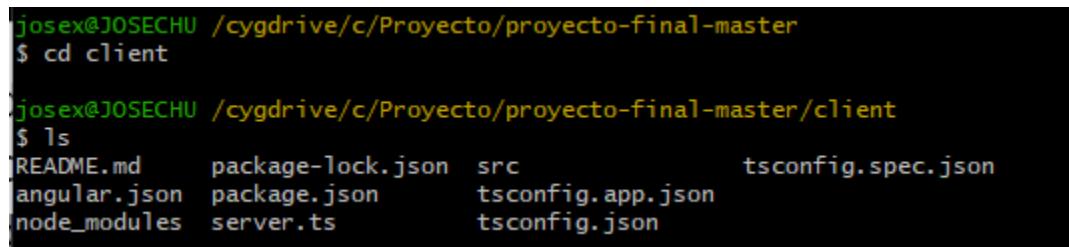
```
cygdrive/c/Proyecto/proyecto-final-master
$ cd c:/
josex@JOSECHU /cygdrive/c
$ cd Proyecto/
josex@JOSECHU /cygdrive/c/Proyecto
$ cd proyecto-final-master/
josex@JOSECHU /cygdrive/c/Proyecto/proyecto-final-master
$ cd api
josex@JOSECHU /cygdrive/c/Proyecto/proyecto-final-master/api
$ cd ..
josex@JOSECHU /cygdrive/c/Proyecto/proyecto-final-master
$ ls
api                               proyecto-final-master.sublime-workspace
proyecto-final-master.sublime-project
josex@JOSECHU /cygdrive/c/Proyecto/proyecto-final-master
$
```

Si nos damos cuenta, tenemos la carpeta de la api y los documentos de sublime text. Pues aquí es donde vamos a instalar el nuevo proyecto. Despues lo creamos con este comando:



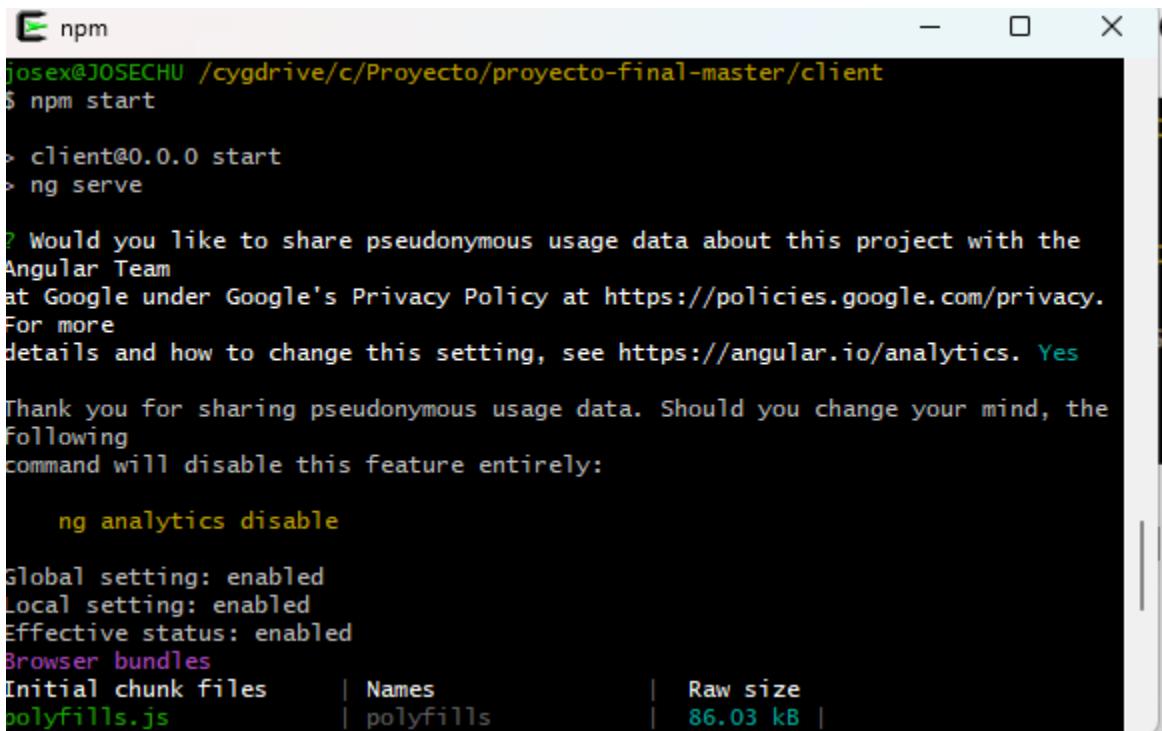
```
josex@JOSECHU /cygdrive/c/Proyecto/proyecto-final-master
$ ng new client
```

Ahora si nos movemos a la carpeta client y hacemos un ls para ver que tenemos observamos que tenemos todas las carpetas del proyecto de angular:



```
josex@JOSECHU /cygdrive/c/Proyecto/proyecto-final-master
$ cd client
josex@JOSECHU /cygdrive/c/Proyecto/proyecto-final-master/client
$ ls
README.md      package-lock.json   src          tsconfig.spec.json
angular.json    package.json       tsconfig.app.json
node_modules   server.ts         tsconfig.json
```

Y si hacemos npm start, nos arranca un servidor local de angular:



```
npm
josex@JOSECHU /cygdrive/c/Proyecto/proyecto-final-master/client
$ npm start

> client@0.0.0 start
> ng serve

? Would you like to share pseudonymous usage data about this project with the
Angular Team
at Google under Google's Privacy Policy at https://policies.google.com/privacy.
For more
details and how to change this setting, see https://angular.io/analytics. Yes

Thank you for sharing pseudonymous usage data. Should you change your mind, the
following
command will disable this feature entirely:

  ng analytics disable

Global setting: enabled
Local setting: enabled
Effective status: enabled
Browser bundles
Initial chunk files      | Names          | Raw size
polyfills.js             | polyfills     | 86.03 kB |
```

8.1. Instalar librerías externas

Ahora vamos a instalar las librerías que vamos a utilizar en angular. Para ello vamos abrir en Vcode la carpeta package.json dentro de la carpeta client y vamos a instalar las dependencias y librerías que nos hacen falta.

Hemos instalado las 2 últimas, bootstrap y jquery:

```
JS app.js          package.json M X
client > package.json > {} dependencies > jquery
  4   "scripts": {
 10     "serve:ssr:client": "node dist/client/server/server.js",
 11   },
 12   "private": true,
 13   "dependencies": [
 14     "@angular/animations": "^17.3.0",
 15     "@angular/common": "^17.3.0",
 16     "@angular/compiler": "^17.3.0",
 17     "@angular/core": "^17.3.0",
 18     "@angular/forms": "^17.3.0",
 19     "@angular/platform-browser": "^17.3.0",
 20     "@angular/platform-browser-dynamic": "^17.3.0",
 21     "@angular/platform-server": "^17.3.0",
 22     "@angular/router": "^17.3.0",
 23     "@angular/ssr": "^17.3.7",
 24     "express": "^4.18.2",
 25     "rxjs": "~7.8.0",
 26     "tslib": "^2.3.0",
 27     "zone.js": "~0.14.3",
 28     "bootstrap": "3.3.6",
 29     "jquery": "1.12"
 30   ]

```

Ahora en la consola hacemos un npm update para actualizar el fichero y arrancamos el proyecto con npm start:

```
josex@JOSECHU /cygdrive/c/Proyecto/proyecto-final-master/client
$ npm update

added 1 package, changed 2 packages, and audited 927 packages in 6s

120 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

josex@JOSECHU /cygdrive/c/Proyecto/proyecto-final-master/client
$ npm start

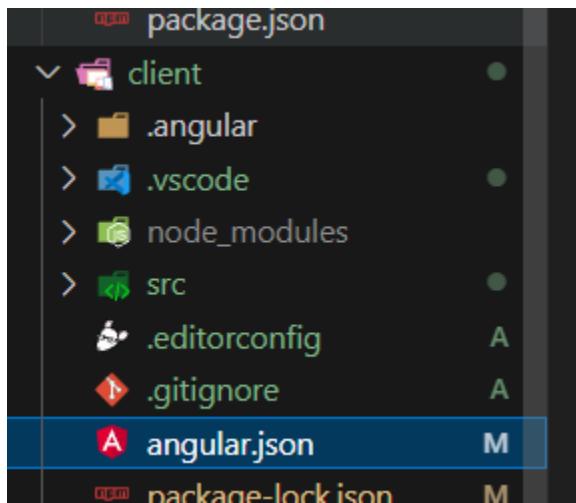
> client@0.0.0 start
> ng serve

Browser bundles
Initial chunk files      | Names           | Raw size
polyfills.js              | polyfills       | 86.03 kb |
```

8.2. Configurar librerías en Angular

Ya tenemos nuestras dos librerías externas instaladas en angular, vamos a configurarlos para poder cargarlos dentro del proyecto que además se incluyen en la compilación y unificación del proyecto.

Para ello vamos abrir el archivo de cli.json de angular:



Las corregimos y las ponemos en scripts:

```
"styles": [
    "src/assets/styles.css",
    "node_modules/bootstrap/dist/css/bootstrap.min.css"
],
"scripts": [
    "node_modules/jquery/dist/jquery.min.js",
    "node_modules/bootstrap/dist/js/bootstrap.min.js"
]
```

8.3. Barra de navegación

El siguiente código se aplicará en el archivo ‘app.component.html’ y hará lo siguiente: crea una barra de navegación con la clase navigation que ocupa toda la anchura del contenedor (col-lg-12). Aquí se desglosa lo que hace cada parte del código:

Contenedor Principal

html

[Copy code](#)

```
class "navigation col-lg-12"
```

navigation: Clase utilizada para aplicar estilos específicos definidos en el archivo CSS.

col-lg-12: Clase de Bootstrap que indica que este elemento ocupará toda la anchura disponible en dispositivos grandes (de escritorio y superiores).

Barra de Navegación

html

[Copy code](#)

```
class "navbar navbar-default"
```

navbar: Clase de Bootstrap que aplica estilos básicos de barra de navegación.

navbar-default: Clase que aplica estilos predeterminados a la barra de navegación.

Contenedor Fluido

html

[Copy code](#)

```
class "container-fluid"
```

container-fluid: Clase de Bootstrap que permite que el contenedor ocupe el 100% del ancho de la pantalla, permitiendo una disposición de contenido fluida y responsiva.

Cabecera de la Barra de Navegación

html

Copy code

```
<div class="navbar-header"> <a href="#" class="navbar-brand">{{title}}</a> </div>
```

navbar-header: Clase de Bootstrap que agrupa los elementos dentro de la cabecera de la barra de navegación.

navbar-brand: Clase de Bootstrap que aplica estilos al nombre de la marca o título del sitio web.

{{title}}: Enlace con una expresión que se usa generalmente en frameworks como Angular para insertar el título dinámicamente.

Lista de Enlaces de Navegación

html

```
<ul class="nav navbar-nav"> <li> <a href="#"> <span class="glyphicon glyphicon-home"></span> Inicio </a> </li> <li> <a href="#"> <span class="glyphicon glyphicon-th-list"></span> Timeline </a> </li> <li> <a href="#"> <span class="glyphicon glyphicon-user"></span> Gente </a> </li> </ul>
```

nav navbar-nav: Clase de Bootstrap que aplica estilos a la lista de enlaces de navegación, alineándolos horizontalmente.

li: Cada elemento de lista representa un enlace de navegación.

a: Enlace de navegación. Cada uno contiene un ícono de Bootstrap Glyphicons y texto descriptivo:

Ícono de casa (glyphicon glyphicon-home): Representa el enlace de inicio.

Ícono de lista (glyphicon glyphicon-th-list): Representa el enlace del timeline.

Ícono de usuario (glyphicon glyphicon-user): Representa el enlace de la página "Gente".

Comportamiento General

La barra de navegación ocupa toda la anchura disponible del contenedor y tiene un fondo de color rojo (#e53c37) según el CSS definido.

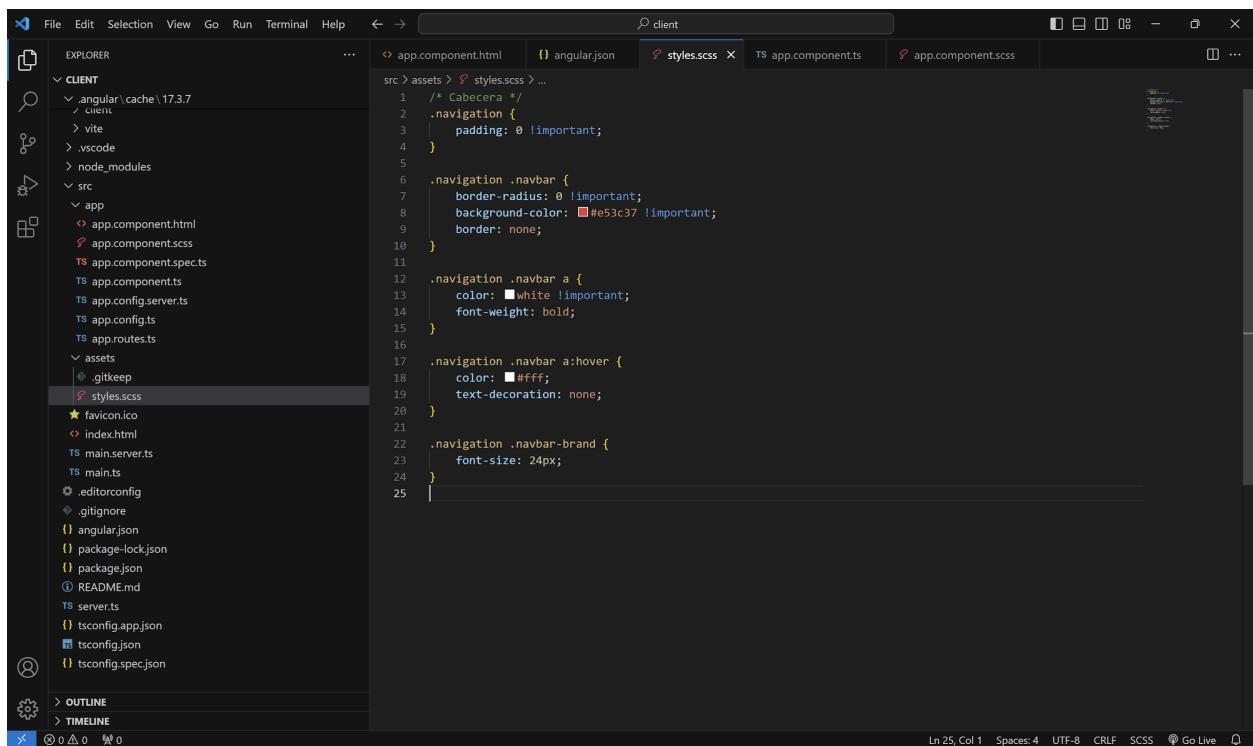
El título o nombre de la marca se centra verticalmente en la barra de navegación.

Los enlaces de navegación (Inicio, Timeline, Gente) están alineados horizontalmente y cada uno tiene un ícono representativo.

Los íconos y el texto de los enlaces de navegación se mostrarán en blanco, según los estilos CSS definidos.

Este código establece una estructura de navegación limpia y profesional, adecuada para sitios web responsivos que utilizan Bootstrap y Angular para la inserción dinámica del título.

Además, se aplicarán los siguientes estilos CSS:

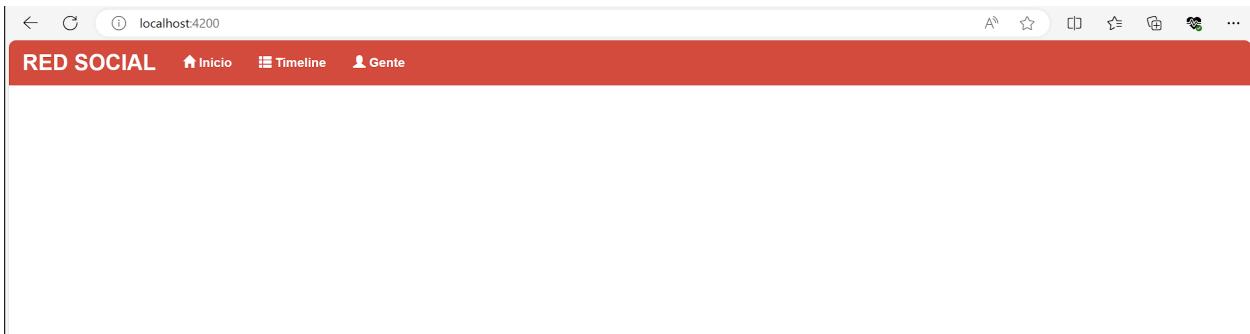


The screenshot shows a code editor interface with the following details:

- File Explorer (Left):** Shows the project structure under "CLIENT". It includes files like "angular.cache.17.3.7", ".client", "vite", ".vscode", "node_modules", "src/app/app.component.html", "app.component.scss", "app.component.spec.ts", "app.component.ts", "app.config.server.ts", "app.config.ts", "app.routes.ts", "assets/.gitkeep", "styles.scss", "favicon.ico", "index.html", "main.server.ts", "main.ts", ".editorconfig", ".gitignore", "angular.json", "package-lock.json", "package.json", "README.md", "server.ts", "tsconfig.app.json", "tsconfig.json", and "tsconfig.spec.json".
- Code Editor (Right):** The active file is "styles.scss". The code defines a navigation bar with a red background (#e53c37), white text, and a central title. It also includes styles for navigation links and their hover states.
- Status Bar (Bottom):** Shows "Ln 25, Col 1" and other file-related information.

```
src > assets > styles.scss > ...
1  /* Cabecera */
2  .navigation {
3      padding: 0 !important;
4  }
5
6  .navigation .navbar {
7      border-radius: 0 !important;
8      background-color: #e53c37 !important;
9      border: none;
10 }
11
12 .navigation .navbar a {
13     color: white !important;
14     font-weight: bold;
15 }
16
17 .navigation .navbar a:hover {
18     color: #ffff;
19     text-decoration: none;
20 }
21
22 .navigation .navbar-brand {
23     font-size: 24px;
24 }
```

Al abrir la url <http://localhost:4200/> se reflejará lo siguiente:



8.4. Creación de 2 componentes

Para ello, se creará dentro de la carpeta 'apps' una llamada 'components', y dentro de esta dos carpetas más, llamadas: 'login' y 'register', y dentro de estas los siguientes archivos:

```
// login.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'login',
  standalone: true,
  templateUrl: './login.component.html',
})

export class LoginComponent {
  public title: string = 'Identificate';

  ngOnInit() {
    console.log('Componente de login cargado...');
  }
}
```

```
// register.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'register',
  standalone: true,
  templateUrl: './register.component.html',
})

export class RegisterComponent {
  public title: string = 'Registrate';

  ngOnInit() {
    console.log('Componente de register cargado...');
  }
}
```

The screenshot shows the VS Code interface with the 'client' workspace selected. The Explorer sidebar shows a project structure under 'CLIENT'. In the main editor area, the file 'login.component.html' is open, displaying the following code:

```
<h1>{{ title }}</h1>
```

The screenshot shows the VS Code interface with the 'client' workspace selected. The Explorer sidebar shows a project structure under 'CLIENT'. In the main editor area, the file 'register.component.html' is open, displaying the following code:

```
<h1>{{ title }}</h1>
```

Y la carpeta 'app.module.ts' se implementará de la siguiente manera:

The screenshot shows the VS Code interface with the 'client' workspace selected. The Explorer sidebar shows a project structure under 'CLIENT'. In the main editor area, the file 'app.module.ts' is open, displaying the following code:

```
1 // app.module.ts
2 import { NgModule } from '@angular/core';
3 import { BrowserModule } from '@angular/platform-browser';
4 import { RouterModule } from '@angular/router';
5
6 // Componentes standalone
7 import { LoginComponent } from './components/login/login.component';
8 import { RegisterComponent } from './components/register/register.component';
9
10 // Rutas
11 import { routes } from './app.routes';
12
13 @NgModule({
14   declarations: [], // No declara componentes standalone aquí
15   imports: [
16     BrowserModule,
17     RouterModule.forRoot(routes),
18     LoginComponent, // Importa componentes standalone aquí
19     RegisterComponent // Importa componentes standalone aquí
20   ],
21   providers: [],
22   bootstrap: [] // No hay bootstrap en AppModule porque AppComponent es standalone
23 })
24 export class AppModule { }
```

Entonces, al cargar el localhost, aparecerán los siguientes componentes

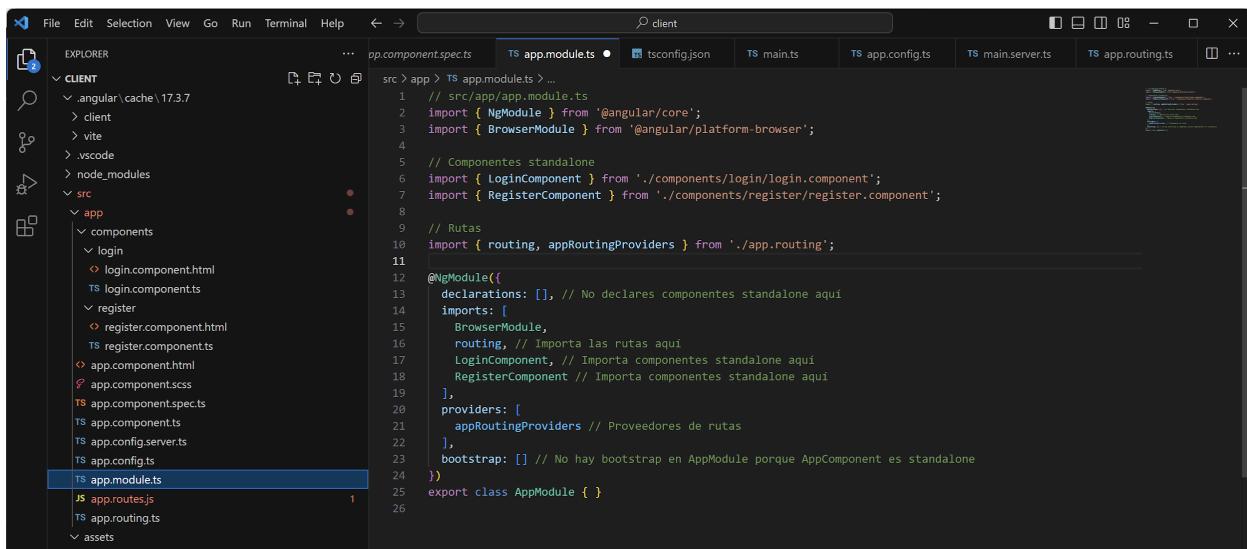
The screenshot shows a web browser window with the URL 'localhost:4200/#'. The page has a red header bar with the text 'RED SOCIAL' and navigation links for 'Inicio', 'Timeline', and 'Gente'. Below the header, there are two buttons: 'Identificate' and 'Registerate'.

```
src > app > TS app.routing.ts > ...
1 // src/app/app.routing.ts
2 import { ModuleWithProviders } from '@angular/core';
3 import { Routes, RouterModule } from '@angular/router';
4 import { LoginComponent } from './components/login/login.component';
5 import { RegisterComponent } from './components/register/register.component';
6
7 export const appRoutes: Routes = [ // Asegurate de exportar appRoutes
8   { path: '', component: LoginComponent },
9   { path: 'login', component: LoginComponent },
10  { path: 'register', component: RegisterComponent }
11 ];
12
13 export const appRoutingProviders: any[] = [];
14 export const routing: ModuleWithProviders<any> = RouterModule.forRoot(appRoutes);
15
```

8.5. Configurar el routing

Se comenzará creando el archivo ‘app.routing.ts’ en la carpeta ‘app’, como se muestra a continuación:

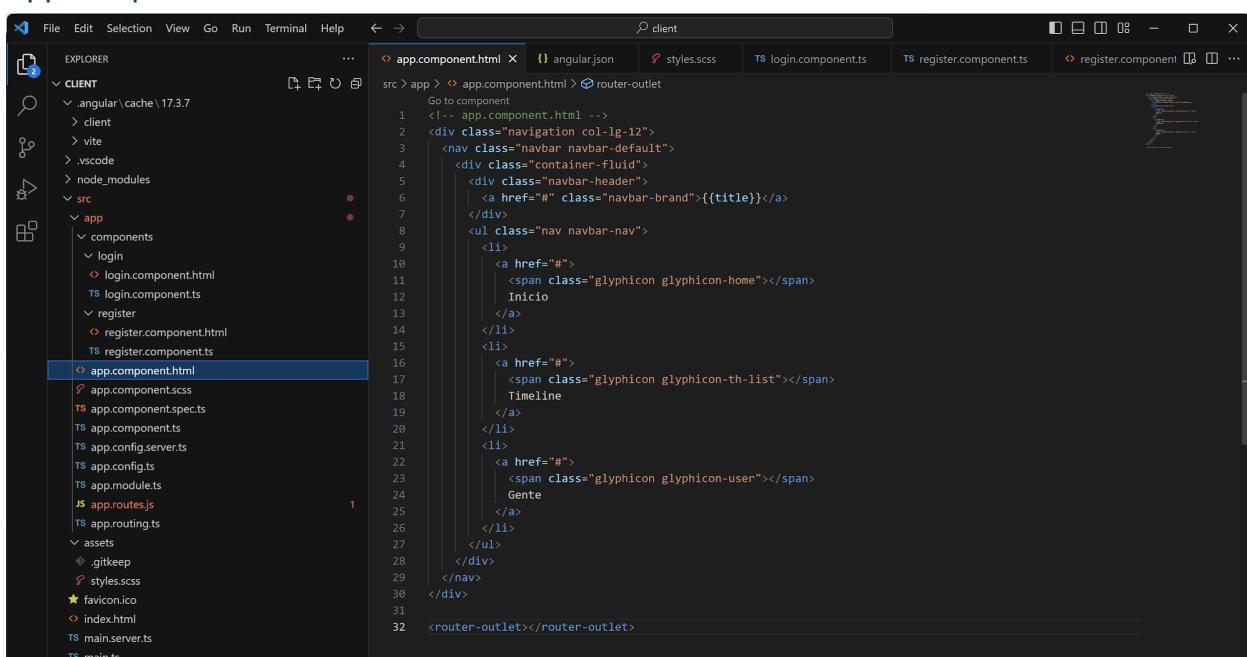
Adicionalmente, es necesario modificar el archivo 'app.module.ts'



The screenshot shows the VS Code interface with the 'client' workspace selected. The Explorer sidebar on the left shows the project structure under the 'CLIENT' folder. The 'app.module.ts' file is open in the main editor area. The code defines an AppModule with imports for BrowserModule, NgModule, and RouterModule, declarations for LoginComponent and RegisterComponent, providers for AppRoutingModule, and a bootstrap array.

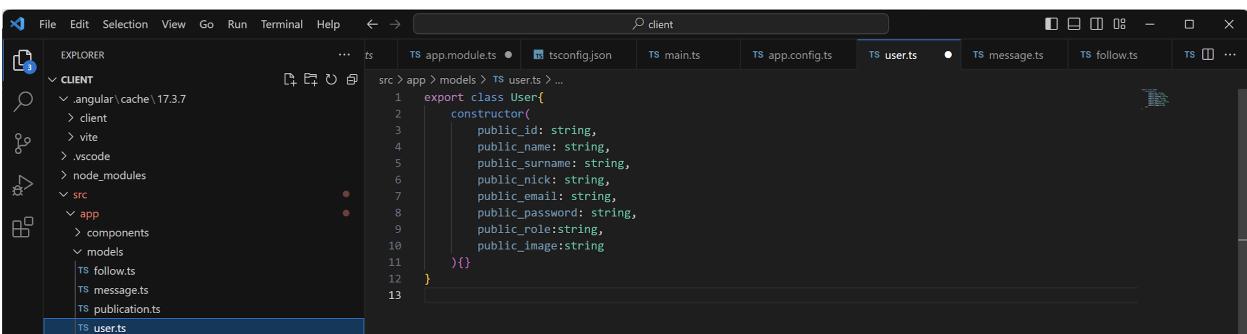
```
src > app > TS app.module.ts ...
1 // src/app/app.module.ts
2 import { NgModule } from '@angular/core';
3 import { BrowserModule } from '@angular/platform-browser';
4
5 // Componentes standalone
6 import { LoginComponent } from './components/login/login.component';
7 import { RegisterComponent } from './components/register/register.component';
8
9 // Rutas
10 import { routing, appRoutingProviders } from './app.routing';
11
12 @NgModule({
13   declarations: [], // No declares componentes standalone aqui
14   imports: [
15     BrowserModule,
16     routing, // Importa las rutas aqui
17     LoginComponent, // Importa componentes standalone aqui
18     RegisterComponent // Importa componentes standalone aqui
19   ],
20   providers: [
21     appRoutingProviders // Proveedores de rutas
22   ],
23   bootstrap: [] // No hay bootstrap en AppModule porque AppComponent es standalone
24 })
25 export class AppModule { }
```

También, ahora se puede utilizar la directiva router-outlet en el archivo 'app.component.html'



The screenshot shows the VS Code interface with the 'client' workspace selected. The Explorer sidebar on the left shows the project structure under the 'CLIENT' folder. The 'app.component.html' file is open in the main editor area. It contains an ng-component element with a router-outlet directive. The router-outlet is used to display content from the LoginComponent and RegisterComponent components.

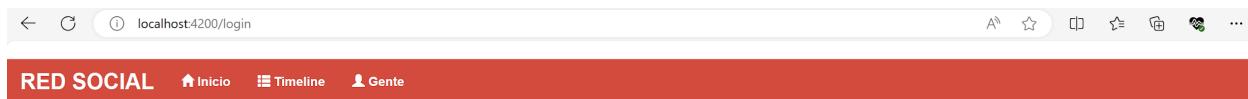
```
src > app > <!-- app.component.html --> <router-outlet>
1 <!-- app.component.html -->
2 <div class="navigation col-lg-12">
3   <nav class="navbar navbar-default">
4     <div class="container-fluid">
5       <div class="navbar-header">
6         <a href="#" class="navbar-brand">{{title}}</a>
7       </div>
8       <ul class="nav navbar-nav">
9         <li>
10           <a href="#">
11             <span class="glyphicon glyphicon-home"></span>
12             Inicio
13           </a>
14         </li>
15         <li>
16           <a href="#">
17             <span class="glyphicon glyphicon-th-list"></span>
18             Timeline
19           </a>
20         </li>
21         <li>
22           <a href="#">
23             <span class="glyphicon glyphicon-user"></span>
24             Gente
25           </a>
26         </li>
27       </ul>
28     </div>
29   </nav>
30 </div>
31 <router-outlet></router-outlet>
```



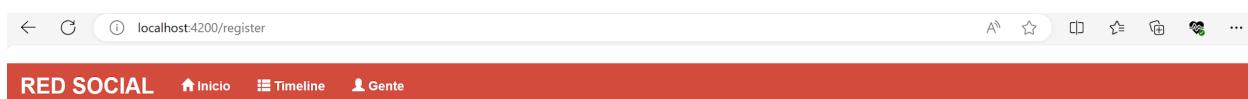
The screenshot shows the VS Code interface with the 'client' workspace selected. The Explorer sidebar on the left shows the project structure under the 'CLIENT' folder. The 'user.ts' file is open in the main editor area. It defines a User class with properties for public_id, public_name, public_surname, public_nick, public_email, public_password, public_role, and public_image.

```
src > app > models > TS user.ts ...
1 export class User {
2   constructor(
3     public_public_id: string,
4     public_public_name: string,
5     public_public_surname: string,
6     public_public_nick: string,
7     public_public_email: string,
8     public_public_password: string,
9     public_public_role: string,
10    public_public_image: string
11  ){}
12 }
```

Consecuentemente, ahora al poner en la url login o register, se redirige correctamente



Identificate



8.6. Los modelos

Se creará una nueva carpeta 'models' dentro de 'app' y todos los modelos necesarios que se enlazarán con el backend:

```
File Edit Selection View Go Run Terminal Help < > ⌘ client
EXPLORER
CLIENT
src > app > models > TS message.ts > Message > constructor
1 export class Message{
2   constructor(
3     public_id: string,
4     public_text: string,
5     public_viewed: string,
6     public_created_at: string,
7     public_emitter: string,
8     public_receiver: string
9  ){}
10 }
```

```
File Edit Selection View Go Run Terminal Help < > ⌘ client
EXPLORER
CLIENT
src > app > models > TS follow.ts > Follow > constructor
1 export class Follow{
2   constructor(
3     public_id: string,
4     public_user: string,
5     public_followed: string
6  ){}
7 }
```

```
File Edit Selection View Go Run Terminal Help < > ⌘ client
EXPLORER
CLIENT
src > app > models > TS publication.ts > Publication > constructor
1 export class Publication{
2   constructor(
3     public_id: string,
4     public_text: string,
5     public_file: string,
6     public_created_at: string,
7     public_user: string
8  ){}
9 }
```

9. Registro, Login y Mis datos

9.1 Página y formulario de registro

Se implementará un código en el archivo ‘register.component.html’ que define un formulario de registro en un componente Angular. A continuación, desgloso cada parte del código y explico su funcionalidad:

Estructura General del Código

```
<div class="col-lg-6"> <h1>{{ title }}</h1> <form #registerForm="ngForm"
(ngSubmit)="onSubmit(registerForm)" class="col-lg-10"> ... </form> </div>
```

<div class="col-lg-6">: Un contenedor con una clase de Bootstrap para la disposición en una cuadrícula de 12 columnas.

<h1>{{ title }}</h1>: Un encabezado que muestra el título del formulario, que se enlaza a la propiedad title del componente Angular.

<form #registerForm="ngForm" (ngSubmit)="onSubmit(registerForm)"
class="col-lg-10">: El elemento del formulario con:

#registerForm="ngForm": Una referencia local que vincula el formulario a ngForm para habilitar las funcionalidades de formularios de Angular.

(ngSubmit)="onSubmit(registerForm)": Un evento que llama al método onSubmit del componente cuando se envía el formulario, pasando el formulario como argumento.

class="col-lg-10": Una clase de Bootstrap para el diseño.

Campos del Formulario

Nombre

```
<p> <label>Nombre</label> <input type="text" name="name" #name="ngModel"
[(ngModel)]="user.name" class="form-control" required /> <span *ngIf="!name.valid &&
name.touched"> El nombre es obligatorio!! </span> </p>
```

<label>Nombre</label>: Etiqueta del campo de nombre.

<input type="text" name="name" #name="ngModel" [(ngModel)]="user.name"
class="form-control" required />: Campo de entrada de texto:

name="name": Nombre del campo.

#name="ngModel": Referencia local vinculada a ngModel para validar y acceder a las propiedades del modelo.

[(ngModel)]="user.name": Enlaza el valor del campo a la propiedad name del objeto user en el componente.

class="form-control": Clase de Bootstrap para estilo.

required: Atributo HTML para hacer el campo obligatorio.

: Mensaje de error que se muestra si el campo no es válido (!name.valid) y ha sido tocado (name.touched).

Apellidos

```
<p> <label>Apellidos</label> <input type="text" name="surname" #surname="ngModel"
[(ngModel)]="user.surname" class="form-control" required /> <span
*ngIf="!surname.valid && surname.touched"> Los apellidos son obligatorios!! </span>
</p>
```

Similar al campo de nombre, pero para surname.

Apodo

```
<p> <label>Apodo</label> <input type="text" name="nick" #nick="ngModel"
[(ngModel)]= "user.nick" class="form-control" required /> <span *ngIf="!nick.valid &&
nick.touched"> El apodo es obligatorio!! </span> </p>
```

Similar al campo de nombre, pero para nick.

Correo Electrónico

```
<p> <label>Correo electrónico</label> <input type="email" name="email"
#email="ngModel" [(ngModel)]= "user.email" class="form-control" required
pattern="[a-zA-Z0-9.!#$%&'^+=?^`{}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*" /> <span
*ngIf="!email.valid && email.touched"> El correo es obligatorio y debe tener un formato
válido!! </span> </p>
```

<input type="email">: Campo de entrada específico para correos electrónicos.

pattern="...": Atributo HTML para validar el formato del correo electrónico.

Mensaje de error que se muestra si el campo no es válido y ha sido tocado.

Contraseña

```
<p> <label>Contraseña</label> <input type="password" name="password"
#password="ngModel" [(ngModel)]= "user.password" class="form-control" required />
<span *ngIf="!password.valid && password.touched"> La contraseña es obligatoria!!
</span> </p>
```

<input type="password">: Campo de entrada específico para contraseñas.

Similar a los campos anteriores, con validación y mensaje de error.

Botón de Envío

```
<input type="submit" [value]="title" class="btn btn-success"
[disabled]= "!registerForm.form.valid" />
```

<input type="submit">: Botón para enviar el formulario.

[value]="title": El valor del botón se enlaza a la propiedad title del componente.

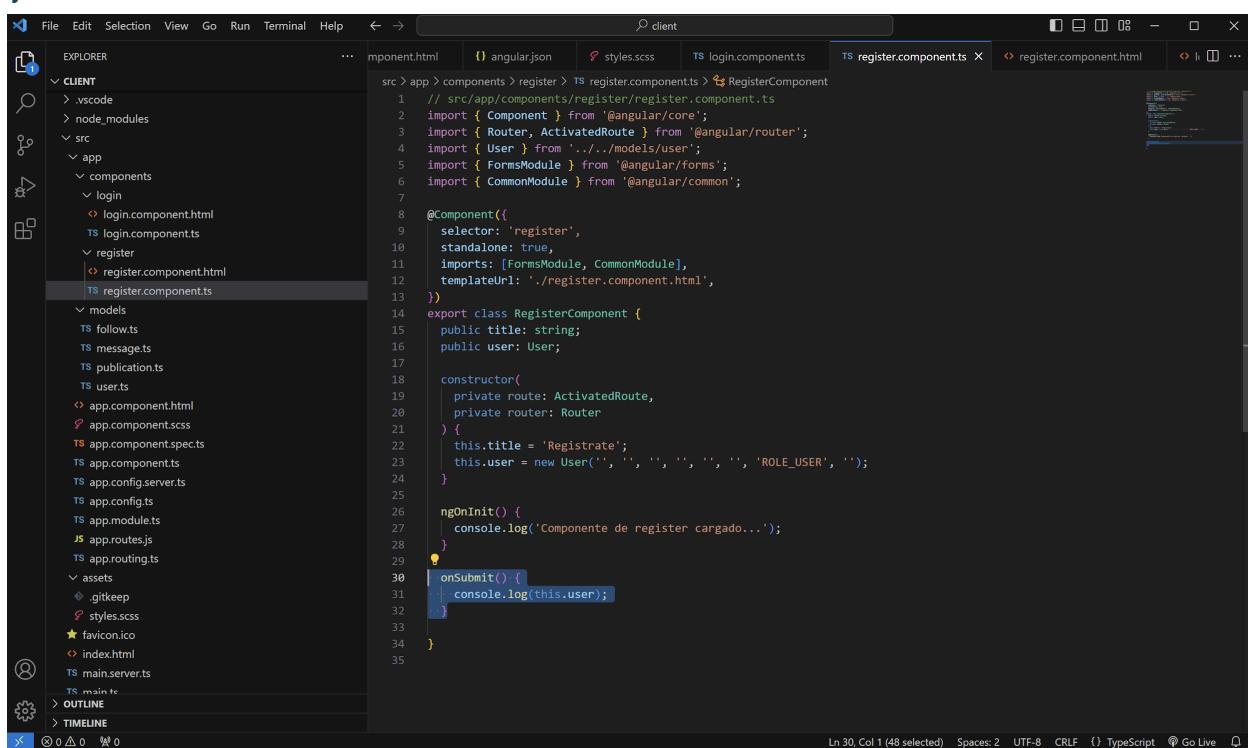
class="btn btn-success": Clase de Bootstrap para estilo.

[disabled]!="registerForm.form.valid": El botón se deshabilita si el formulario no es válido.

9.2. Recibir datos del formulario

Para recibir los datos del formulario implementamos el siguiente código en el archivo 'register.component.ts':

```
onSubmit() {  
  console.log(this.user);  
}
```



```
onSubmit() {  
  console.log(this.user);  
}
```

Entonces, para probar que los datos llegan correctamente, y tras implementar la línea <form #registerForm="ngForm" (ngSubmit)="onSubmit()" class="col-lg-12"> en el archivo 'register.component.html', se rellenan todos los datos y se envían, y se refleja

en la consola de la web:

The screenshot shows a web browser window with the URL `localhost:4200/register`. The page title is "RED SOCIAL". The main content is a "Register" form with fields for Nombre, Apellidos, Apodo, Correo electrónico, and Contraseña. The "Nombre" field contains "juan". The "Correo electrónico" field contains "juan@gmail.com". The "Contraseña" field contains "....". A green "Registrate" button is at the bottom. To the right of the browser window is an open developer console. The console output shows:

```
Angular is running in development mode. core.mjs:29869
Componente de register cargado... register.component.ts:27
register_component_ts:31
User (id: '', name: 'juan', surname: 'juan', nick: 'juan', email: 'juan@mail.com', ...)
```

9.3. Crear servicio de usuarios

En este sentido, es necesario crear una nueva carpeta llamada 'services', y dentro de ella dos nuevos archivos: 'global.ts' y 'user.service.ts'.

La clase UserService es un servicio en Angular que utiliza HttpClient para realizar solicitudes HTTP. Aquí tienes una descripción de lo que hace cada parte del código:

@angular/core: Proporciona el decorador `@Injectable()`, que marca la clase como un servicio inyectable.

@angular/common/http: Proporciona HttpClient y HttpHeaders para hacer solicitudes HTTP.

rxjs: Biblioteca para manejar operaciones asíncronas, aunque en este fragmento no se utiliza directamente.

./global: Archivo que probablemente contiene constantes globales, como la URL base de tu API.

./models/user: Modelo de datos User, que probablemente define la estructura de un objeto de usuario.

Decorador Injectable Indica que esta clase se puede injectar en otras partes de la aplicación Angular mediante el sistema de inyección de dependencias.

Propiedades

url: Cadena de texto que contiene la URL base de tu API. Se inicializa con el valor de GLOBAL.url.

Constructor

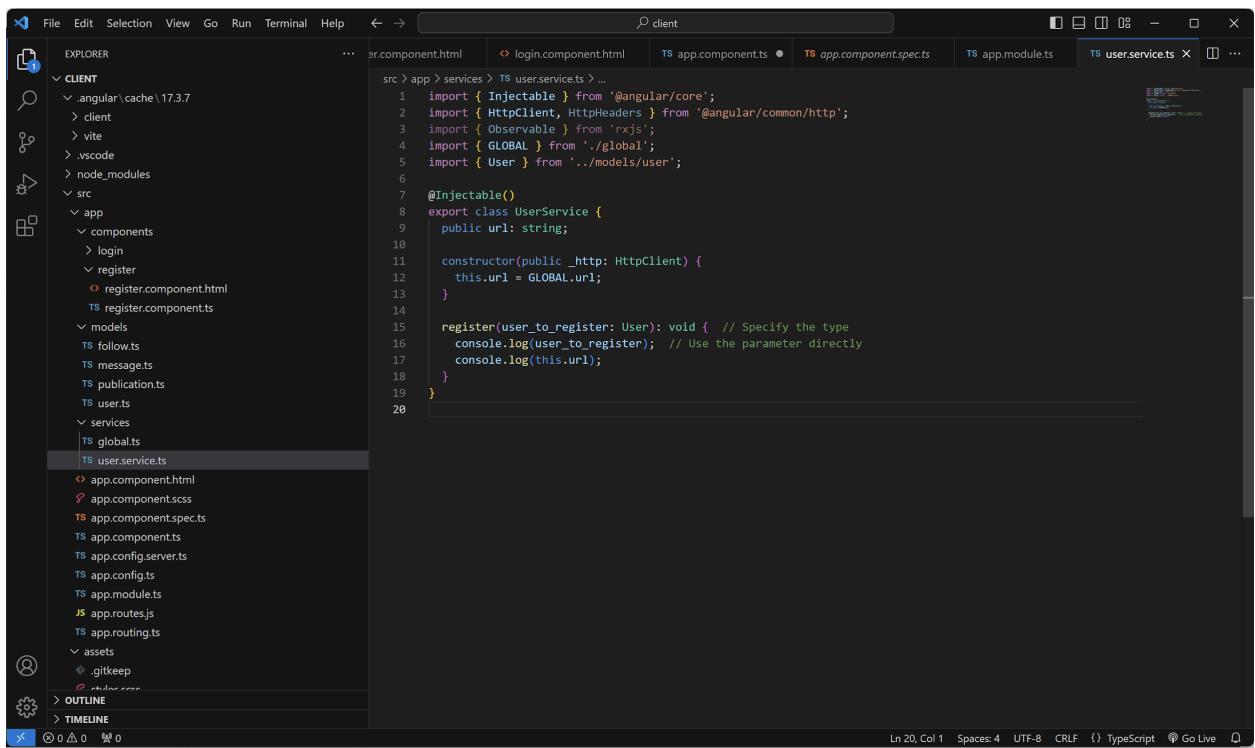
constructor(public _http: HttpClient):

HttpClient es un servicio proporcionado por Angular para realizar solicitudes HTTP.

Al declararlo como public _http, Angular lo inyecta automáticamente cuando se crea una instancia de UserService.

Este servicio (UserService) está configurado para manejar solicitudes HTTP relacionadas con usuarios en tu aplicación Angular. Actualmente, el método register simplemente imprime el usuario a registrar y la URL base en la consola. En una implementación real, probablemente este método realizaría una solicitud HTTP para

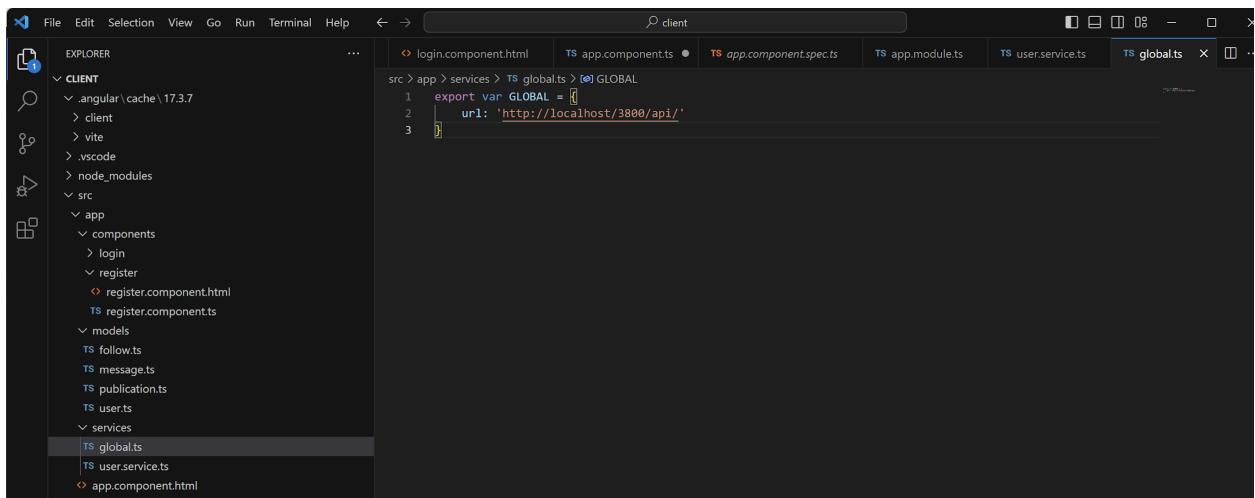
registrar el usuario en un servidor.



The screenshot shows the VS Code interface with the 'client' workspace selected. The Explorer sidebar on the left shows the project structure under the 'CLIENT' folder. The 'user.service.ts' file is open in the main editor area. The code defines a User service class with a register method that logs the user to register and the URL.

```
src > app > services > user.service.ts ...
1 import { Injectable } from '@angular/core';
2 import { HttpClient, HttpHeaders } from '@angular/common/http';
3 import { Observable } from 'rxjs';
4 import { GLOBAL } from './global';
5 import { User } from '../models/user';
6
7 @Injectable()
8 export class UserService {
9   public url: string;
10
11   constructor(public _http: HttpClient) {
12     this.url = GLOBAL.url;
13   }
14
15   register(user_to_register: User): void { // Specify the type
16     console.log(user_to_register); // Use the parameter directly
17     console.log(this.url);
18   }
19 }
20
```

Adicionalmente, este será el archivo 'global.ts':



The screenshot shows the VS Code interface with the 'client' workspace selected. The Explorer sidebar on the left shows the project structure under the 'CLIENT' folder. The 'global.ts' file is open in the main editor area. It exports a variable 'GLOBAL' containing a URL.

```
src > app > services > global.ts > GLOBAL
1 export var GLOBAL = [
2   |   url: 'http://localhost:3800/api/'
3 ]
```

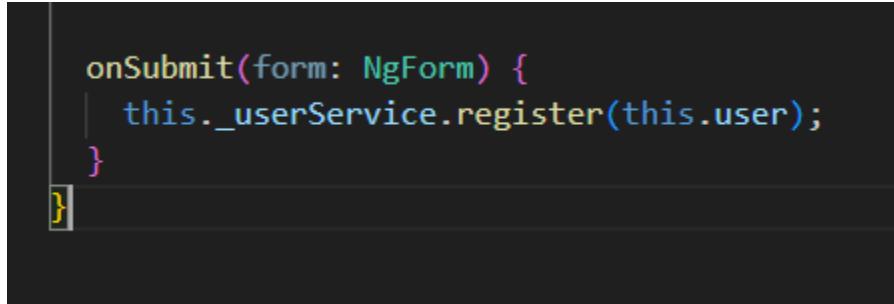
Y finalmente, al registrar un usuario, nos aparece el objeto y la url de la API:

The screenshot shows a browser window with a red header bar containing 'RED SOCIAL' and navigation links. Below it is a 'Registerate' form with fields for Nombre, Apellidos, Apodo, Correo electrónico, and Contraseña, each with placeholder text like 'asdfs', 'sdfs', and '.....'. A green 'Registrate' button is at the bottom. To the right is the browser's developer tools 'Console' tab, which displays the following log output:

```
Angular is running in development mode.
Componente de register cargado...
register.component.ts:30
user.service.ts:16
▶ User {id: '', name: 'asdfs', surname: 'sdfs', nick: 'sdfs', email: 'sdfs@hotmail.com', ...}
http://localhost:3800/api/
user.service.ts:17
```

9.4. Registrar Usuarios

Ahora vamos hacer la petición ajax a la api, que guarde el usuario que le estamos pasando a este método, es decir, cuando nosotros le damos al botón de enviar y esto lanza el método Onsubmit:



Esto llama al servicio y aquí yo tengo que guardar el usuario:

Lo primero que tenemos que hacer es convertir lo que nos llega como parámetros en un objeto json, después tenemos que pasarle las cabeceras, y por último pasarle la respuesta que nosotros queremos que nos devuelva:

```
register(user: User): Observable<any> {
  let json = JSON.stringify(user);
  let params = json;
  let headers = new HttpHeaders().set('Content-Type', 'application/json');

  return this._http.post(this.url + 'register', params, { headers: headers }).pipe(
    catchError((error: any) => {
      console.error('Error occurred:', error);
      return throwError(error || 'Server error');
    })
  );
}
```

Ahora lo que devuelve es un observable y ahora yo me puedo suscribir a ese observable y utilizar sus callbacks:

```
onSubmit(form: NgForm) {
  this._userService.register(this.user).subscribe(
    response => {
      if (response.user && response.user.id) {
        console.log(response.user);
      }
    },
    error => {
      console.log(<any>error);
    }
  );
}
```

También creamos una propiedad llamada status y se la colocamos abajo:

```

export class RegisterComponent implements OnInit {
  public title: string;
  public user: User;
  public status: string;

  constructor(
    private route: ActivatedRoute,
    private router: Router,
    private _userService: UserService
  ) {
    this.title = 'Registrate';
    this.user = new User('', '', '', '', '', '', 'ROLE_USER');
  }

  ngOnInit() {
    console.log('Componente de register cargado...');
  }

  onSubmit(form: NgForm) {
    this._userService.register(this.user).subscribe(
      response => {
        if (response.user && response.user._id) {
          console.log(response.user);

          this.status = 'success';
        }
      }
    );
  }
}

```

Después de esto iremos al html y colocamos este cambio, para poder registrar al usuario haciendo uso del ngIf:

```

<div class="col-lg-5">
  <h1 class="h1-strong">{{ title }}</h1>

  <div class="alert alert-success" *ngIf="status == 'success'">
    Registro Completo correctamente, <a routerLink="/login">identificate aquí</a>
  </div>

  <div class="alert alert-danger" *ngIf="status == 'error'">
    El registro no ha podido completarse, quizás tu email ya esté en uso, intentalo de nuevo con otros datos
  </div>

```

Ahora si registramos el usuario nos saltara esto:

The screenshot shows a web browser window with three tabs open: "Course: Desarrollar una red soc...", "Client", and "ChatGPT". The main content area displays a registration form titled "RED SOCIAL". The form fields include "Nombre" (asdsa), "Apellidos" (dasdsad), "Apodo" (dasdas), and "Correo electrónico" (asdasd@asdas.com). Below the form is a green success message: "Registro Completo correctamente, identificate aquí". At the bottom of the page, the developer tools' "Console" tab is active, showing the following output:

```
Angular is running in development mode.
Componente de register cargado...
Object {
  email: "asdasd@asdas.com"
  image: null
  name: "asdsa"
  nick: "dasdas"
  password: "$2a$10$8uMU0Plq/nnxqIjWdwq0cORz21z0UHQ75KpTwQ/4MvYymOgn519H6"
  role: "ROLE_USER"
  surname: "dasdsad"
  __v: 0
  _id: "664e36cbd5ad87b9e3d1ae0d"
} [[Prototype]]: Object
```

Y podemos clicar en Identificate aquí y nos llevará a la página de identificación.

Aquí podemos ver cómo hemos ido registrando usuario en nuestra base de datos:

Open connections

Quickstart X users X

MongoDB Local (localhost:27017) > Base_Datos_Red_Social > users

Run Load query Save query Query history Set default query Copy

Query: {}

Projection: {}

Skip: 0

Sort: {}

Limit: 0

Result | Query Code | Explain

← → | 50 | Documents 1 to 15 | 🔒 +

users > name

_id	name	surname	nick
663537b39dc0f8...	admin	admin	adm
6638f8129de7c0...	David	Lopez	david
663b66724ad701...	jose	lopez	jose
663b8ab864a7fd...	inma	corcoles	inma
663ceecf6032d0...	jesus	Lopez	jesus
663cef156032d0...	enrique	garcia	enriq
663cef4b6032d0...	raul	perez	raulp
664e3258d5dd8...	dasdas	dasd	dasd
664e36cbd5dd8...	asdsa	dasdsad	dasda
664e3874d5dd8...	José Luis	López Camins	jose
664e389ed5dd8...	fafafsFED	pepeDSFAF	joseF
664e38d3d5dd8...	rodrigo	perez	rodrig
664e39c7d5dd8...	hytgkn	hntnmvn	hrnfv
664e3acd5dd8...	hysrhyjn	sghnfn	fsgng
664e3accd5dd8...	afwefwe	teraghg	aerth

Query

Match all (\$and)

+ Drag and drop field here or double-click

Projection

+ Drag and drop fields here

Sort

+ Drag and drop fields here

9.5. Formulario login

Ya tenemos la página de registro en la cual podemos registrarnos correctamente utilizando el formulario y ahora vamos hacer la página de login. En este tendremos un formulario de login y vamos a poder identificarnos en la plataforma.

Ahora cogemos y colocamos igual que en el de registro el encabezado:

The screenshot shows a code editor with two tabs open: 'login.component.html' and 'login.component.ts'. The 'login.component.html' tab contains the following code:

```
<div class="col-lg-5">
  <h1 class="h1-strong">{{ title }}</h1>
</div>
```

Ahora tendremos que crear una propiedad user, la cual va a estar modificando las cosas.

The screenshot shows the 'login.component.ts' file with the following code:

```
import { Component } from '@angular/core';
import { FormsModule } from '@angular/forms'; // Importa FormsModule
import { Router, ActivatedRoute, Params } from '@angular/router';
import { User } from '../../../../../models/user';
import { UserService } from '../../../../../services/user.service';

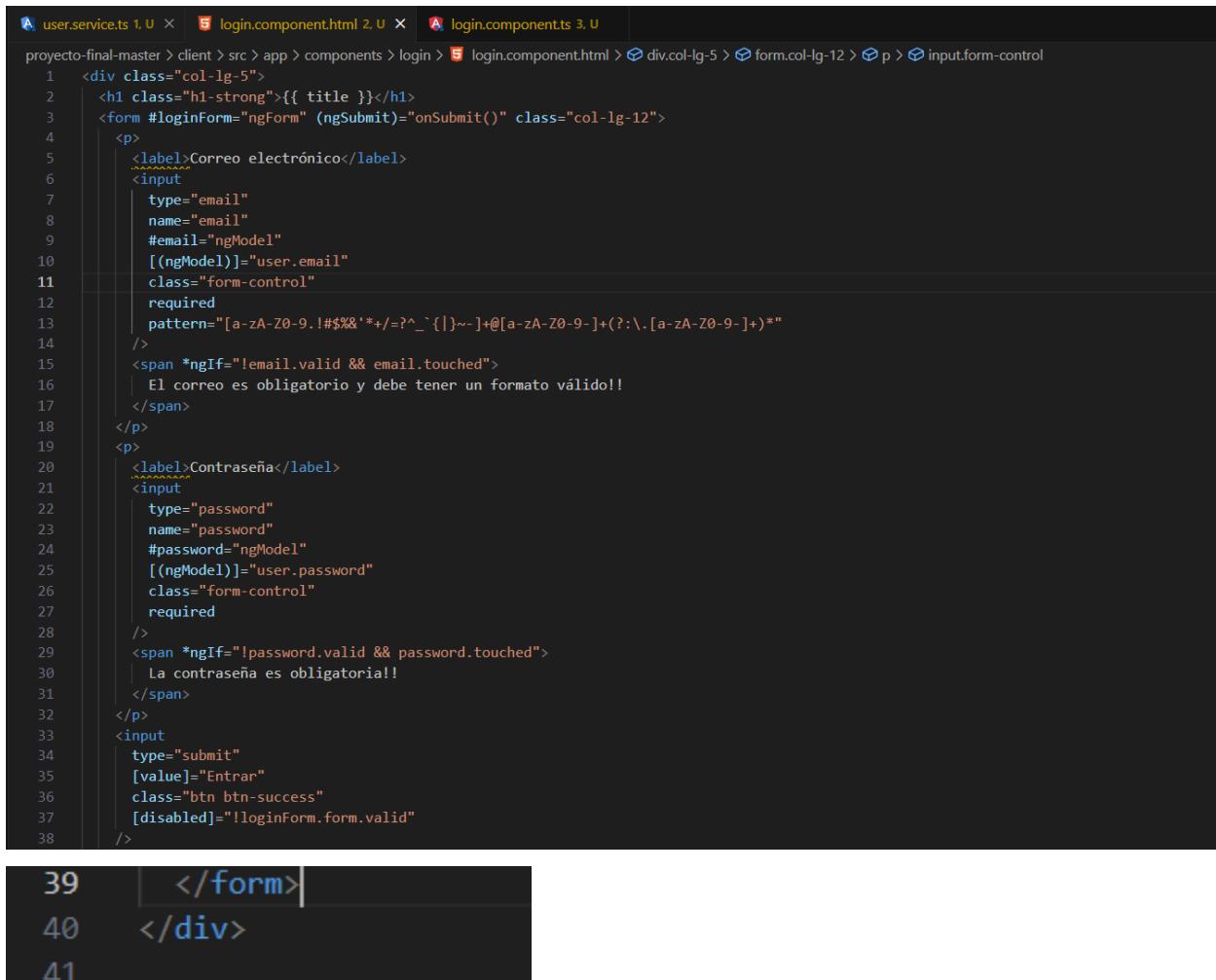
@Component({
  selector: 'login',
  standalone: true,
  imports: [FormsModule], // Importa FormsModule aquí
  templateUrl: './login.component.html',
  providers: [UserService]
})
export class LoginComponent {
  public title: string;
  public user: User;

  constructor(
    private _route: ActivatedRoute,
    private _router: Router,
    private _userService: UserService
  ) {
    this.title = 'Identificate';
    this.user = new User('', '', '', '', '', '', 'ROLE_USER', '');
  }

  ngOnInit() {
    console.log('Componente de login cargado...');
  }

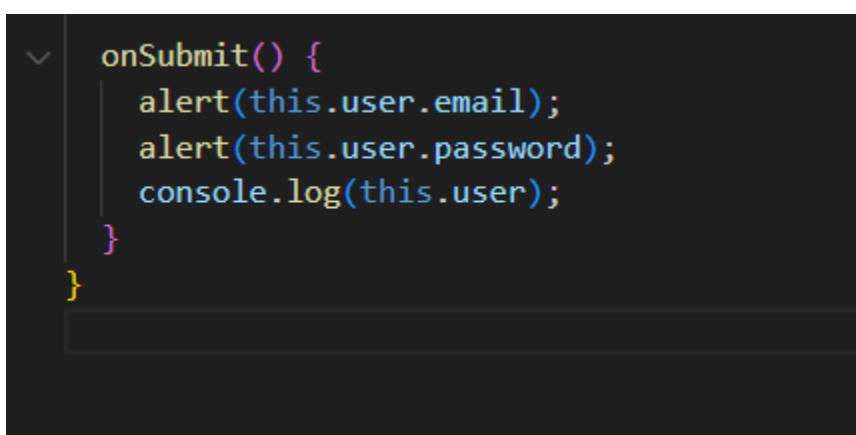
  onSubmit(loginForm: any) {
    console.log(loginForm.value);
  }
}
```

Ahora vamos a crear nuestro formulario:



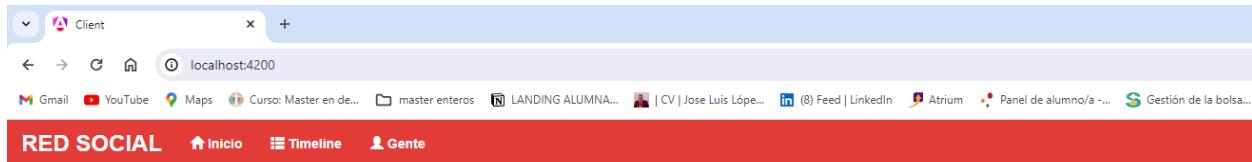
```
❶ user.service.ts 1.U ✘ login.component.html 2.U ✘ login.component.ts 3.U
proyecto-final-master > client > src > app > components > login > login.component.html > div.col-lg-5 > form.col-lg-12 > p > input.form-control
  1  <div class="col-lg-5">
  2    <h1 class="h1-strong">{{ title }}</h1>
  3    <form #loginForm="ngForm" (ngSubmit)="onSubmit()" class="col-lg-12">
  4      <p>
  5        <label>Correo electrónico</label>
  6        <input
  7          type="email"
  8          name="email"
  9          #email="ngModel"
 10          [(ngModel)]="user.email"
 11          class="form-control"
 12          required
 13          pattern="^[_a-zA-Z0-9.]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$"
 14        />
 15        <span *ngIf="!email.valid && email.touched">
 16          | El correo es obligatorio y debe tener un formato válido!
 17        </span>
 18      </p>
 19      <p>
 20        <label>Contraseña</label>
 21        <input
 22          type="password"
 23          name="password"
 24          #password="ngModel"
 25          [(ngModel)]="user.password"
 26          class="form-control"
 27          required
 28        />
 29        <span *ngIf="!password.valid && password.touched">
 30          | La contraseña es obligatoria!!
 31        </span>
 32      </p>
 33      <input
 34        type="submit"
 35        [value]="Entrar"
 36        class="btn btn-success"
 37        [disabled]="!loginForm.form.valid"
 38      />
 39      </form>
 40    </div>
 41
```

Además le vamos a cambiar el Onsubmit en nuestro componente para que muestre los datos:



```
onSubmit() {
  alert(this.user.email);
  alert(this.user.password);
  console.log(this.user);
}
```

Si registramos el usuario en la web nos aparece esto:



Identificate

Correo electrónico

jhon@jhon.com

Contraseña

....

Entrar



Identificate

Correo electrónico

jhon@jhon.com

Contraseña

....

Entrar



Identificate

Correo electrónico

jhon@jhon.com

Contraseña

....

Entrar

9.6. Identificar Usuarios

Ahora vamos hacer que el formulario de login funcione y cuando yo envíe mis datos que la api Rest nos devuelva un resultado que nos diga que estamos identificados o no lo estamos., además, nos devuelva los datos si el usuario se ha registrado con las credenciales correctas.

Lo primero de todo es modificar nuestro servicio y añadir un nuevo método, vamos a añadir un método llamado signUp el cual es el método de login del servicio.

En nuestro user.service.ts lo creamos. A este le vamos a pasar lo siguiente, un usuario de tipo User. Este va a devolver un observable de tipo any y también le vamos a pasar un segundo parámetro que será gettoken que irá a null.

Esto quiere decir que cuando no le pasemos nada pues los datos que nos devolverá el api los datos identificado y si le pasamos el segundo nos pasara el token del usuario identificado y este token será el que identificará al usuario contra el api frente a cada una de las peticiones que hagamos desde el frontend.

```
register(json: string): Observable<any> {
  let json = JSON.stringify(user);
  let params = json;
  let headers = new HttpHeaders().set('Content-Type', 'application/json');

  return this._http.post(this.url + 'register', params, { headers: headers })
    .catchError((error: any) => {
      console.error('Error occurred:', error);
      return throwError(error || 'Server error');
    })
  );
}

signup(user: User, gettoken = null): Observable<any> {
  if (gettoken != null) {
    user.gettoken = gettoken;
  }
}
```

Ahora creamos otra variable para poder enviar los parametros desde el formulario:

```
let json = JSON.stringify(user);
let params = json;
let headers = new HttpHeaders().set('Content-Type', 'application/json');

return this._http.post(this.url + 'login', params, { headers: headers }).pipe(
  catchError((error: any) => {
    console.error('Error occurred:', error);
    return throwError(error || 'Server error');
  })
);
```

Ahora nos situamos en nuestro componente y creamos una variable status y además vamos a loguear y conseguir los datos del usuario:

```

21  public status: string = ''; // Definir la propiedad Entrar
22
23  constructor(
24    private _route: ActivatedRoute,
25    private _router: Router,
26    private _userService: UserService
27 ){
28    this.title = 'Identificate';
29    this.user = new User('', '', '', '', '', '', 'ROLE_USER', '');
30    this.Entrar = 'Entrar'; // Asignar un valor a Entrar
31  }
32
33  ngOnInit() {
34    console.log('Componente de login cargado...');
35  }
36
37  onSubmit() {
38    // loguear al usuario y conseguir sus datos
39    this._userService.signup(this.user).subscribe(
40      response => {
41        console.log(response.user);
42
43
44      },
45      error =>{
46        let errorMessage = <any>error;
47        console.log(errorMessage);
48
49        if(errorMessage != null){
50          this.status = 'error';
51        }
52      }
53    );

```

Modificamos la variable status y le daremos un succés además le pondremos en HTML unos mensajes:

```

onSubmit() {
    // loguear al usuario y conseguir sus datos
    this._userService.signup(this.user).subscribe(
        response => {
            console.log(response.user);
            this.status = 'success';

```

Y en HTML:

```

<div class="alert alert-success" *ngIf="status == 'success'>
    Te has identificado correctamente, bienvenido!!
</div>

<div class="alert alert-danger" *ngIf="status == 'error'">
    No te has podido identificar correctamente
</div>

```

Ahora si nos identificamos en la web, esta nos devuelve correctamente los datos:

The screenshot shows a web browser window with a red header bar labeled "RED SOCIAL". Below it is a login form titled "Identificate" with fields for "Correo electrónico" (jhon@jhon.com) and "Contraseña" (redacted). A green "Entrar" button is at the bottom. At the bottom of the page, the Angular DevTools console is open, showing the following output:

```

Angular is running in development mode.
Componente de login cargado...
User {id: '', name: '', surname: '', nick: '', email: 'jhon@jhon.com', ...}
  id: ''
  name: ''
  surname: ''
  nick: ''
  password: 'dasdasd'
  role: 'ROLE_USER'
  email: 'jhon@jhon.com'
  ...
  [[Prototype]]: Object

```

9.7. Persistir la sesión del usuario

Seguimos con el proyecto y lo que vamos hacer es persistir la información que nos devuelve la api, es decir, la información del usuario identificado y el token del usuario, vamos a persistir en el localStorage.

El localStorage es una pequeña memoria que nos da el navegador web para almacenar información, es decir, el almacenamiento local de información del navegador, cada URL tiene su propia localStorage. Una vez que esta almacenada dicha información vamos poder acceder desde cualquier URL de nuestra api.

Lo primero que haremos sera persistir los datos del usuario:

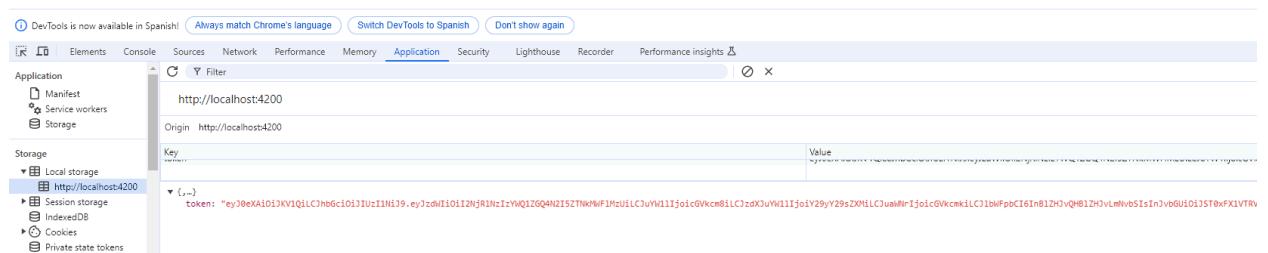
```
// PERSISTIR DATOS DEL USUARIO
localStorage.setItem('identity', JSON.stringify(this.identity));
```

Y ahora nos centramos en el token:

```
console.log('TOKEN OBTENIDO', this.token);

// PERSISTIR TOKEN DEL USUARIO
localStorage.setItem('token', this.token);
```

Ahora si nos vamos a la web y abajo en application observamos veremos como se nos guarda en el localStorage:



Ahora vamos a crear un par de método que consultan el localStorage y nos saquen la identidad del usuario y el identity, con todos los datos del usuario y el otro será sacar el token de una manera rápida utilizando el service:

Primero creamos las 2 variables que vamos a utilizar en este caso identity y token:

```
@Injectable()
export class UserService {
  public url: string;
  public identity: any;
  public token: string | null = null;
```

Y después creamos los métodos:

```
getIdentity() {
  let identity = JSON.parse(localStorage.getItem('identity') || 'null');

  if (identity && identity !== "undefined") {
    this.identity = identity;
  } else {
    this.identity = null;
  }
  return this.identity;
}

getToken() {
  let token = localStorage.getItem('token');
  if (token && token !== "undefined") {
    this.token = token;
  } else {
    this.token = null;
  }
  return this.token;
}
```

Ya tenemos nuestros 2 servicios y podemos utilizarlos donde queramos, para ello nos vamos a ir a app.component.ts y cargamos nuestro servicio:

```
import { UserService } from './services/user.service';
```

Ahora lo metemos dentro de nuestro providers:

```
providers: [UserService]
```

Y ahora creamos el constructor:

```
constructor(private _userService: UserService) {  
  this.title = 'RED SOCIAL';  
}  
  
ngOnInit() {  
  this.identity = this._userService.getIdentity();  
  console.log(this.identity);  
}  
}
```

Ahora cambiamos el HTML para mostrar estas funciones:

```
<ul class="nav navbar-nav navbar-right" *ngIf="!identity">  
  <li>  
    <a [routerLink]=["/login"] [routerLinkActive]=["active"]>  
      <span class="glyphicon glyphicon-log-in"></span>  
      Login  
    </a>  
  </li>  
  <li>  
    <a [routerLink]=["/register"] [routerLinkActive]=["active"]>  
      <span class="glyphicon glyphicon-user"></span>  
      Registro  
    </a>  
  </li>  
</ul>  
</div>  
</nav>  
</div>
```

Aplicaremos ahora un servicio llamado DoCheck y lo importaremos:

```
ngDoCheck(){  
  this.identity = this._userService.getIdentity();  
}  
}
```

Tendremos que importarlo:

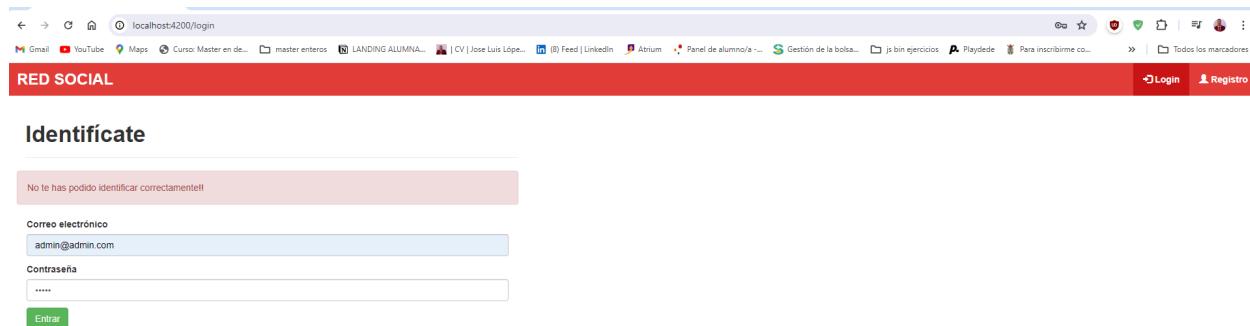
```
import { Component, OnInit, DoCheck } from '@angular/core';
```

Y después lo exportamos:

```
export class AppComponent implements OnInit, DoCheck {
```

Por lo tanto, al actualizar la página todo funciona dinámicamente, entonces, el componente ha visto que ha sucedido un cambio que a nivel de la aplicación ha cambiado algo y como se ha producido ese cambio pues entonces ha vuelto a ejecutar lo que hay dentro de DoCheck.

Que ha sucedido en la aplicación, pues qué hemos cambiado un valor en el localStorage, hemos hecho algún evento dentro del componente y cada vez que se ha lanzado ese evento, pues este DoCheck se vuelve a ejecutar con cual de esta manera dinámica transparente nos actualiza lo que hay en la barra de navegación. Con lo cual ya tenemos el login hecho, también tenemos la parte de persistencia de los datos de usuario logueado.



The screenshot shows a browser window with multiple tabs open. The active tab is a login page for a social media application. The URL is localhost:4200. The page has a red header with the text 'RED SOCIAL' and navigation links like 'Inicio', 'Timeline', 'Gente'. Below the header is a form with two input fields: 'Correo electrónico' containing 'pedro@pedro.com' and 'Contraseña' which is empty. There is also a green 'Identificate' button.

The screenshot shows the Chrome DevTools Console tab. The log output shows several entries of 'localStorage identity: null'. Then there is one entry for a login attempt: '> (email: "pedro@pedro.com", password: "pedro")'. This is followed by another 'localStorage identity: null' entry. To the right of the console, there is a sidebar titled 'Issues' which lists 3 issues related to 'user.service.ts:37'.

9.8. Página home

Ahora lo que vamos hacer es un nuevo componente para la página de inicio. Para ello en la carpeta de componentes, creamos una nueva carpeta llamada home y dentro de ella un archivo llamado home.component.ts y en él crearemos los básicos que tiene un componente.

```
proyecto-final-master > client > src > app > components > home >  home.component.ts >  HomeComponent
  1 import { Component, OnInit } from '@angular/core';
  2
  3 @Component({
  4   selector: 'home',
  5   templateUrl: './home.component.html',
  6 })
  7 export class HomeComponent implements OnInit {
  8   public title: string;
  9
 10   constructor(){
 11     this.title = 'Bienvenido a Red Social';
 12   }
 13
 14   ngOnInit() {
 15     console.log('home.component cargado !!!');
 16   }
 17 }
 18
```

Después vamos a crear su vista, creando un nuevo archivo llamado `home.component.html`:

Tendrá lo siguiente:

```
>  home.component.html >  div.col-lg-8.col-lg-offset-2 >  h1.h1-strong
  1 <div class="col-lg-8 col-lg-offset-2">
  2   <h1 class="h1-strong">{{title}}</h1>
  3
  4 </div>
  5
```

Ahora tendremos que utilizar el componente, tendremos que darlo de alta en el `app.module` y utilizarlo.

Lo importamos y después lo declaramos abajo:

```

import { HomeComponent } from './components/home/home.component';

// Rutas
import { routing, appRoutingProviders } from './app.routing';

const routes: Routes = [
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent },
  // other routes
];

@NgModule({
  declarations: [
    LoginComponent,
    RegisterComponent,
    HomeComponent,
    AppComponent
  ]
})

```

Nos queda declarar la ruta para este componente:

```

import { RegisterComponent } from './components/register/register.component';
import { HomeComponent } from './components/home/home.component';
const appRoutes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'home', component: HomeComponent },
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent }
];

```

Y por último en el html le ponemos la ruta en inicio:

```

<li>
  <a [[routerLink]="'/home'" routerLinkActive="['active']">
    <span class="glyphicon glyphicon-home"></span>
    Inicio
  </a>
</li>

```

Colocaremos un menú dentro de la página home:

```
proyecto-final-master > client > src > app > components > home > 5 home.component.html > ...
1  <div class="col-lg-8 col-lg-offset-2">
2
3
4  <div class="jumbotron">
5    <h1 class="h1-strong">{{title}}</h1>
6    <p>Disfruta de las publicaciones de tus amigos y comparte tu vida!!</p>
7    <p>
8      <a href="#" class="btn btn-success btn-lg" role="button">
9        Mi perfil
10     </a>
11     <a href="#" class="btn btn-primary btn-lg" role="button">
12       Timeline
13     </a>
14     <a href="#" class="btn btn-default btn-lg" role="button">
15       Gente
16     </a>
17   </p>
18 </div>
19
20 </div>
21
```

The screenshot shows a web browser window with the URL `localhost:4200` in the address bar. The page has a red header bar with the text "RED SOCIAL" and navigation links for "Inicio", "Timeline", and "Gente". On the right side of the header are "Login" and "Registro" buttons. Below the header, the main content area features a large "Bienvenido a Red Social" heading and a subtext "Disfruta de las publicaciones de tus amigos y comparte tu vida". At the bottom of this section are three buttons: "Mi perfil" (green), "Timeline" (blue), and "Gente" (white).

9.9. Menú de usuario

Ahora vamos a modificar el app.component para mostrar un pequeño menú en la parte izquierda, donde situaremos el nombre del usuario y la posibilidad de cerrar sesión.

```
42 |     <ul class="nav navbar navbar-right" **ngIf="identity">
43 |       <li class="avatar">
44 |         | <!--cargar imagen de usuario-->
45 |       </li>
46 |       <li class="dropdown">
47 |         <a class="dropdown-toggle" data-toggle="dropdown" href="#">
48 |           {{identity.name}} <span class="caret"></span>
49 |         </a>
50 |         <ul class="dropdown-menu">
51 |           <li>
52 |             <a href="#">
53 |               <span class="glyphicon glyphicon-user"></span>
54 |               Mi perfil
55 |             </a>
56 |           </li>
57 |           <li>
58 |             <a href="#">
59 |               <span class="glyphicon glyphicon-cog"></span>
60 |               Mis datos
61 |             </a>
62 |           </li>
63 |           <li>
64 |             <a href="#">
65 |               <span class="glyphicon glyphicon-log-out"></span>
66 |               Cerrar sesión
67 |             </a>
68 |           </li>
69 |         </ul>
70 |       </li>
71 |     </ul>
72 |   </div>
```

Cogeremos y le modificaremos los estilos:

```
▽ .navigation .navbar .dropdown-menu{
  color: □black !important;
}

▽ .navigation-default .navbar-nav>.open>a, .navigation-default .navbar-nav>.open>a:focus{
  background-color: ▢#cc1717 !important;
}

▽ .navigation .navbar .dropdown-menu a:hover{
  background: □#eee !important;
}
```

9.9.1. Cerrar sesión

Aquí haremos la funcionalidad de cerrar sesión, nos situaremos en nuestro componente app.component.html y comenzaremos.

Para ello nos situamos en nuestro botón de cerrar sesión y le colocamos un evento el cual será logout, que lo crearemos en nuestro archivo app.component.ts:

```
<li>
  <a class="pointer" (click)="logout()">
    <span class="glyphicon glyphicon-log-out"></spa
    Cerrar sesión
  </a>
</li>
```

Y ahora creamos el evento:

```
38   logout(){
39     localStorage.clear();
40     this.identity = null;
41     this._router.navigate(['/']);
42   }
43
44
```

9.9.2. Conseguir estadísticas

Cuando hacemos login en el método gettoken habíamos puesto que deberíamos conseguir los contadores o estadísticas del usuario y eso es interesante porque vamos a necesitar mostrar los datos del usuario en algún momento, por ejemplo: cuantos seguidores tiene a cuantos usuarios sigue y cuantas publicaciones tiene.

Por lo tanto, vamos a crear un método en nuestro servicio que nos permita llamar a los contadores y sacar esos datos de la base de datos.

```

75  getStats() {
76    let stats = null;
77    try {
78      const statsItem = localStorage.getItem('stats');
79      if (statsItem) {
80        stats = JSON.parse(statsItem);
81      }
82    } catch (e) {
83      console.error("Error parsing stats from localStorage", e);
84    }
85
86    this.stats = stats !== null ? stats : null;
87    return this.stats;
88  }
89
90  getCounter(userId: string | null = null): Observable<any> {
91    let token = this.getToken();
92    let headers = new HttpHeaders().set('Content-Type', 'application/json');
93    if (token) {
94      headers = headers.set('Authorization', token);
95    }
96
97    if (userId !== null) {
98      return this._http.get(this.url + 'counter/' + userId, { headers: headers });
99    } else {
100      return this._http.get(this.url + 'counter', { headers: headers });
101    }
102  }
103}
104

```

Después la declaramos en el login:

```

93
94  getCounters() {
95    this._userService.getCounter().subscribe(
96      response => {
97        localStorage.setItem('stats', JSON.stringify(response));
98        this.status = 'success';
99        this._router.navigate(['/']);
100      },
101      error => {
102        console.log(<any>error);
103      }
104    );
105  }
106}
107

```

9.9.3. Componente y ruta Mis datos

Creamos esta carpeta con estos 2 archivos:



Después creamos el código necesario para ello:

```
 proyecto-final-master > client > src > app > components > user-edit > user-edit.component.html
 1 <div class="col-lg-5">
 2   <h1 class="h1-strong">{{ title }}</h1>
 3
 4 </div>
 5
```

Después el del componente:

```
proyecto-final-master > client > src > app > components > user-edit > A user-edit.component.ts > ↗ UserEditComponent > ↗ pu
  1 import{ Component, OnInit} from '@angular/core';
  2 import {Router, ActivatedRoute, Params } from '@angular/router';
  3 import{ User } from '../../models/user';
  4 import { CommonModule } from '@angular/common';
  5 import { UserService } from '../../services/user.service';
  6
  7 @Component({
  8   selector: 'user-edit',
  9   templateUrl: './user-edit.component.html',
10   imports:[CommonModule],
11   providers:[UserService]
12 })
13 export class UserEditComponent implements OnInit{
14   public title: string;
15   public user: User;
16   public identity;
17   public token;
18   public status = '';
19
20   constructor(
21     private _route: ActivatedRoute,
22     private _router: Router,
23     private _userService: UserService
24   ){
25     this.title = 'Editar usuario';
26     this.user = this._userService.getIdentity();
27     this.identity = this.user;
28     this.token = this._userService.getToken();
29   }
30   ngOnInit(): void {
31     console.log(this.user);
32     console.log('user-edit.component.ts cargado');
33   }
34 }
35
36
```

9.9.4. Formulario para actualizar mis datos de usuario

Para ello creamos el formulario completo:

```

4   <form #userEditForm="ngForm" (ngSubmit)="OnSubmit" class="col-lg-12">
5     <p>
6       <label for="name">Name</label>
7       <input type="text" name="name" #name="ngModel" [(ngModel)]="user.name"
8         required class="form-control"/>
9       <span *ngIf="!name.value && name.touched">
10      | El nombre es obligatorio
11      </span>
12    </p>
13    <p>
14      <label for="surname">Apellidos</label>
15      <input type="text" name="surname" #surname="ngModel" [(ngModel)]="user.surname"
16        required class="form-control"/>
17      <span *ngIf="!surname.value && surname.touched">
18      | Los apellidos son obligatorios
19      </span>
20    </p>
21    <p>
22      <label for="nick">Nick</label>
23      <input type="text" name="nick" #nick="ngModel" [(ngModel)]="user.nick"
24        required class="form-control"/>
25      <span *ngIf="!nick.value && nick.touched">
26      | El nick es obligatorio
27      </span>
28    </p>
29    <p>
30      <label for="email">Correo Electronico</label>
31      <input type="text" name="email" #email="ngModel" [(ngModel)]="user.email"
32        required class="form-control"/>
33      <span *ngIf="!email.value && email.touched">
34      | El email es obligatorio
35      </span>
36    </p>
37    <input type="submit" value="{{title}}" class="btn btn-warning"
38      [disabled]="!userEditForm.form.valid"/>
39  </form>
40 </div>
41

```

9.9.5. Modificar el usuario

Vamos a crear un método en nuestro servicio de angular para actualizar los datos y hacer una petición ajax al servidor para que nos cambie, modifique y actualice los datos del usuario.

Creamos el método aquí en user.service.ts:

```

104 //Nuevo método para actualizar los datos del servicio:
105
106 updateUser(user: User): Observable<any> {
107   let params = JSON.stringify(user);
108   let headers = new HttpHeaders()
109     .set('Content-Type', 'application/json') // Corrección aquí
110     .set('Authorization', this.getToken());
111   return this._http.put(this.user + 'update-user/' + user._id, params, { headers: headers });
112 }
113 }
114

```

Ahora nos vamos a nuestro componente y llamamos a este método:

```

36  onSubmit(){
37    console.log(this.user);
38    this._userService.updateUser(this.user).subscribe(
39      response => {
40        if(!response.user){
41          this.status = 'error';
42        }else{
43          this.status = 'success';
44          localStorage.setItem('identity', JSON.stringify(this.user));
45          this.identity = this.user;
46
47          //SUBIDA DE IMAGEN DE USUARIO
48        }
49      },
50      error =>{
51        let errorMessage = <any>error;
52        console.log(errorMessage);
53
54        if(errorMessage != null){
55          this.status = 'error';
56        }
57      }
58    )
59  }
60 }
61

```

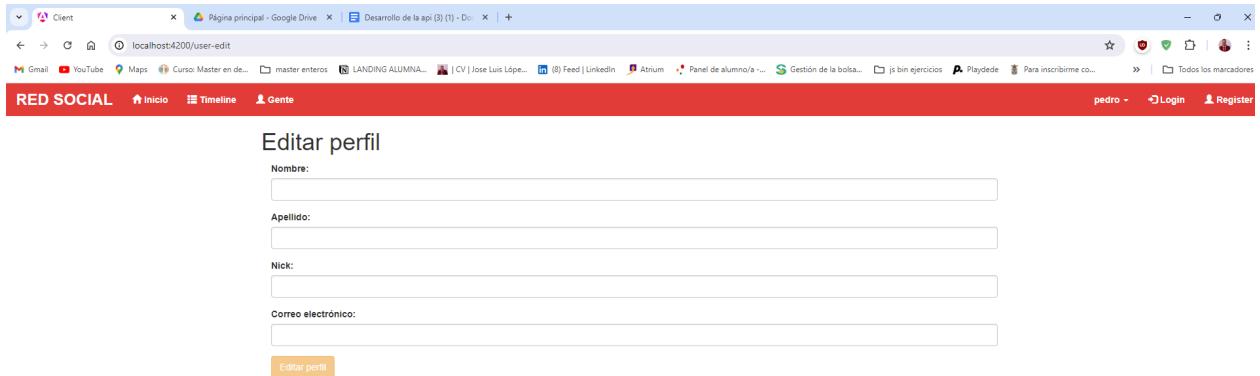
Y ahora lo que haremos será pintar los mensajes de error en la vista:

```

4  <div class="alert alert-success" **ngIf="status == 'success'">
5    | Se han actualizado tus datos correctamente!!
6  </div>
7  <div class="alert alert-danger" **ngIf="status == 'error'">
8    | No se han podido actualizar tus datos!!
9  </div>

```

Aquí tendremos la página una vez toquemos para actualizar:



9.9.6. Mejoras en el Backend

Vamos hacer unas mejoras en la api y evitemos poner un email y un nick duplicado en la actualización de los datos.

Para ello modificamos la función de updateUser:

```
async function updateUser(req, res) {
  try {
    const userId = req.params.id;
    const update = req.body;

    // Eliminar la contraseña del cuerpo de la solicitud si está presente
    delete update.password;

    // Verificar si el usuario tiene permiso para actualizar sus datos
    if (userId !== req.user.sub) {
      return res.status(403).send({ message: 'No tienes permiso para actualizar los datos del usuario' });
    }

    // Verificar si el email o nick ya están en uso por otro usuario
    const existingUser = await User.findOne({ $or: [{ email: update.email.toLowerCase() }, { nick: update.nick.toLowerCase() }] });
    if (existingUser && existingUser._id != userId) {
      return res.status(400).send({ message: 'El correo electrónico o el apodo ya están en uso' });
    }

    // Actualizar el usuario
    const userUpdate = await User.findByIdAndUpdate(userId, update, { new: true });
    if (!userUpdate) {
      return res.status(404).send({ message: 'No se ha podido actualizar el usuario' });
    }

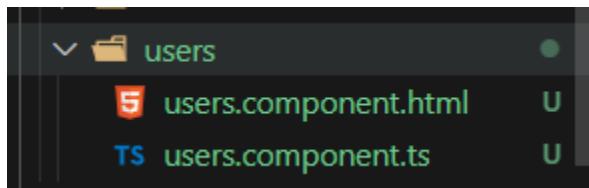
    return res.status(200).send({ user: userUpdate });
  } catch (error) {
    console.error('Error en la función updateUser:', error);
    return res.status(500).send({ message: 'Error en la petición' });
  }
}
```

Así de esta manera evitaremos copiar correos y nick duplicados en nuestra api.

10. Sección de gente

Lo primero de todo, será crear un componente nuevo, para poder crear esta sección, creamos una carpeta nueva y dos archivos.

El componente nuevo se llamará users y dentro de él, tendremos 2 archivos, uno será el componente.ts y el otro será el component.html:



Ahora dentro de nuestro archivo users.component.ts, creamos lo necesario para poder utilizarlo, es decir, el importamos las rutas que necesitamos, creamos el constructor y el ngOnInit:

```
client > src > app > components > users > TS users.component.ts > ...
1  import { Component, OnInit } from '@angular/core';
2  import { Router, ActivatedRoute, Params } from '@angular/router'; // Importar desde @angular/router
3  import { User } from '@app/models/user';
4  import { UserService } from '@app/services/user.service';
5  import { GLOBAL } from '@app/services/global';
6
7  @Component({
8    selector: 'users',
9    templateUrl: './users.component.html',
10   providers: [UserService]
11 })
12 export class UsersComponent implements OnInit {
13   public title: string;
14   public identity;
15   public token;
16
17   constructor(
18     private _route: ActivatedRoute,
19     private _router: Router,
20     private _userService: UserService
21   ) {
22     // Inicializar propiedades en el constructor
23     this.title = 'Gente';
24     this.identity = this._userService.getIdentity();
25     this.token = this._userService.getToken();
26   }
27
28   ngOnInit() {
29     console.log('users.component.ts cargado');
30   }
31 }
32 |
```

Después creamos nuestro users.component.html:

```
1      Go to component
2      <!DOCTYPE html>
3      <html>
4          <head>
5              <title>Gente</title>
6          </head>
7          <style>
8              .header {
9                  width: 100%;
10                 display: flex;
11                 justify-content: flex-end; /* Alinea todo el contenido de la clase header a la derecha */
12                 background-color: #f5f5f5;
13                 padding: 10px 20px;
14             }
15
16             .menu {
17                 display: flex;
18                 align-items: center;
19             }
20
21             .logo {
22                 margin-right: 20px; /* Espacio entre el logo y la barra de búsqueda */
23             }
24
25             .search-bar {
26                 margin-right: 20px; /* Espacio entre la barra de búsqueda y el menú de navegación */
27             }
28
29             .search-bar input {
30                 padding: 5px;
31                 font-size: 14px;
32             }
33
34             nav ul {
35                 display: flex;
36                 list-style: none;
37                 padding: 0;
38                 margin: 0;
39             }
```

```
40  nav ul li {  
41    margin-left: 20px; /* Espacio entre los elementos del menú */  
42  }  
43  
44  nav ul li a {  
45    text-decoration: none;  
46    color: #333;  
47    font-size: 14px;  
48  }  
49  
50  nav ul li a:hover {  
51    color: #0073b1; /* Color al pasar el cursor */  
52  }  
53  
54  </style>  
55  <body>  
56  <div class="container">  
57  <header>  
58  <div class="logo">  
59  <a href="#">  
60    
64  </a>  
65  </div>  
66  <div class="search-bar">  
67  <input type="text" placeholder="Buscar" />  
68  </div>
```

```

69      <nav>
70          <ul>
71              <li><a href="#">Inicio</a></li>
72              <li><a href="#">Mi red</a></li>
73              <li><a href="#">Empleos</a></li>
74              <li><a href="#">Mensajes</a></li>
75              <li><a href="#">Notificaciones</a></li>
76              <li><a href="#">Yo</a></li>
77              <li><a href="#">Para negocios</a></li>
78          </ul>
79      </nav>
80  </header>
81  <main>
82      <div class="jumbotron">
83          <h2 class="h1-strong">Gestionar mi red</h2>
84          <button class="btn btn-success btn-lg" role="button">Amplía tu red</button>
85          <button class="btn btn-success btn-lg" role="button">Ponte al día</button>
86          <p>No hay invitaciones pendientes</p>
87          <button class="btn btn-success btn-lg" role="button">Gestionar</button>
88      </div>
89
90
91
92      <div class="jumbotron">
93          <h2 class="h1-strong">Personas cercanas que podrías conocer</h2>
94          <button class="btn btn-success btn-lg" role="button">Amplía tu red</button>
95          <button class="btn btn-success btn-lg" role="button">Ponte al día</button>
96          <p>No hay invitaciones pendientes</p>
97          <button class="btn btn-success btn-lg" role="button">Gestionar</button>
98      </div>
99
100
101      <div class="people-list">
102          <div class="jumbotron">
103              <h3 class="h1-strong">Personas de tu entorno laboral</h3>
104              <p>Cisco Networking Academy</p>
105              <button class="btn btn-success btn-lg" role="button">Amplía tu red</button>
106              <button class="btn btn-success btn-lg" role="button">Ponte al día</button>
107              <div class="invitations">
108                  <p>No hay invitaciones pendientes</p>
109                  <button class="btn btn-success btn-lg" role="button">Gestionar</button>
110                  <p></p>
111              </div>
112          </div>
113      </div>
114  </main>
115
116  </div>
117 </body>
118 </html>
119

```

Para terminar, nos situamos en nuestro archivo de rutas y creamos esta nueva ruta para este componente:

```
8 import { UsersComponent } from './components/users/users.component';
9
10
11
12 export const routes: Routes = [
13   { path: '', component: HomeComponent },
14   { path: 'home', component: HomeComponent },
15   { path: 'login', component: LoginComponent },
16   { path: 'register', component: RegisterComponent },
17   { path: 'user-edit', component: UserEditComponent },
18   { path: 'users', component: UsersComponent},
19   { path: '**', component: HomeComponent }
20 ];
21
```

Y por último en nuestro archivo principal de html, le damos ese valor a ruta, para poder cargar la pestaña de gente:

```
<li>
  <a href="/users">
    <span class="glyphicon glyphicon-user"></span>
    Gente
  </a>
</li>
</ul>
```

Ahora si pinchamos en nuestra pestaña, nos crea la página:

localhost:4200/users

Gmail YouTube Maps Curso Master en de... master enteros LANDING ALUMNA... CV Jose Luis López Feed | LinkedIn Atrium Panel de alumno/a... Gestión de la bolsa... js bin ejercicios Playdele Para inscribirme co... Todos los marcadores

pedro - Login Register

RED SOCIAL Inicio Timeline Gente

 Buscar

Inicio Mi red Empleos Mensajes Notificaciones Yo Para negocios

Gestionar mi red

Amplía tu red Ponte al día

No hay invitaciones pendientes

Gestionar

Personas cercanas que podrías conocer

Amplía tu red Ponte al día

No hay invitaciones pendientes

Gestionar

