

PROYECTO DE CICLO DE
Desarrollo de aplicaciones web

SOCIAL MEDIA APP



PABLO CRUZ GARRIDO
CURSO 2022/23

ÍNDICE

1. RESUMEN.....	4
1.1. PROBLEMA ESTUDIADO.....	4
1.2. OBJETIVOS.....	4
1.3. METODOLOGÍA.....	5
1.4. RESULTADOS ESPERADOS.....	6
2. INTRODUCCIÓN.....	7
3. OBJETIVOS.....	8
3.1. OBJETIVOS INICIALES.....	8
3.2. OBJETIVOS SECUNDARIOS.....	10
4. MATERIAL.....	11
4.1. HARDWARE.....	11
4.1.1. Ordenador.....	11
4.2. SOFTWARE.....	11
4.2.1. Angular 6.....	11
4.2.2. Node.....	11
4.2.3. NPM.....	12
4.2.4. Express.....	12
4.2.5. Javascript.....	12
4.2.6. TypeScript.....	13
4.2.8. JWT.....	13
4.2.9. Multer.....	13
4.2.10. Git.....	14
4.2.11. Bootstrap.....	14
4.2.12. CSS.....	14
4.2.13. HTML.....	14
4.2.14. MariaDB.....	14
4.2.15. Visual Studio Code.....	15

4.2.16. Bash.....	15
5. MÉTODO.....	16
5.1. INSTALACIÓN.....	16
5.2. ESTRUCTURA APP.....	19
5.2.1. Estructura server.....	19
5.2.2. Estructura cliente.....	21
5.3. INSTALACIÓN DE LAS DEPENDENCIAS.....	22
5.4. DESPLIEGUE DE LA APLICACIÓN.....	22
6. TIEMPO DE EJECUCIÓN.....	23
7. RESULTADOS.....	25
8. CONCLUSIONES.....	32
9. ANEXOS.....	33
9.1. ARCHIVOS DE CONFIGURACIÓN.....	33
9.1.1. Archivo de configuración .env del server.....	33
9.1.2. Archivo de configuración del usuario de la base de datos.....	33
9.1.3. Archivo de configuración constantes Angular.....	34
9.2. UML DE LA BASE DE DATOS.....	34
9.3. EXPLICACIÓN FUNCIONAMIENTO DE LA APP.....	36
9.3.1. Funcionamiento Servidor: Node.....	38
9.3.2. Funcionamiento Cliente: Angular.....	40
10. BIBLIOGRAFÍA Y WEBGRAFÍA.....	44

1. RESUMEN

1.1. PROBLEMA ESTUDIADO

En este proyecto se ha buscado crear una app web que sea una red social ,separando claramente el lado del cliente y del servidor, donde poder registrarte o loguearte, subir fotos, ver fotos de otros usuarios, comentar en las fotos, tener followers y followings, recibir notificaciones cada vez que recibes un mensaje, un comentario en una foto o una petición de amistad, enseñar perfil de usuarios con sus fotos controlando si es amigo o no, bloquear usuarios, deshabilitar cuentas, buscar usuarios por nickname y bloquear usuarios. Todo esto garantizando que el usuario siempre es quien dice ser mediante su JWT y teniendo en cuenta aspectos como si esta bloqueado o no por el otro usuario.

1.2. OBJETIVOS

Los objetivos principales del proyecto son:

- Aprender como trabajan stack modernos como el de MEAN (mysql, express, angular, node) donde el lado del server y cliente estan muy diferenciados.
- Como securizar una app mediante JWT asegurándonos que en todo momento el usuario es quien dice ser.

-
- Aprender a planificar un proyecto antes de empezar a programar para ser lo mas eficientes posibles
 - Uso de git para copias de seguridad.
 - Correcta creacion de base de datos en mysql
 - Creacion de triggers en la base de datos.
 - Profundizar en JS, angular, node, bootstrap, HTML, Css, mysql y git.

1.3.METODOLOGÍA

La metodología usada para este desarrollo ha sido:

- Fase Análisis:
 - Descripción funcionalidades: aquí describíamos como sera nuestra app y lo que queremos que haga.
- Fase Diseño:
 - Diagrama E-R: creamos como sera la base de datos, es decir, sus tablas, campos y relaciones.
 - Análisis de clases: donde identificamos los modelos, atributos y métodos. (UML)
 - Diseño interfaz grafica (prototipo): empezamos a pensar como lucirá nuestra app mediante bocetos.
- Fase Desarrollo:
 - Crear sprints: nos marcamos una serie de metas y su fecha deseada de consecución.
 - Codificación y pruebas.

1.4. RESULTADOS ESPERADOS

Los resultados deseados con este proyecto han sido varios:

- Empezando con un mayor entendimiento por parte del desarrollador de como funcionan y se conectan entre si apps con lenguaje servidor (Java, node, ruby...) y con lenguaje cliente (react, Angular, Vue..).
- Hacer una app web funcional compleja como una red social donde muchas variables deben ser tenidas en cuenta.
- Mejora de habilidad de planificación de un proyecto para evitar el mayor numero posible de problemas durante la codificación de este .
- Tener en cuenta que las verificaciones se deben hacer tanto del lado cliente como del lado servidor.

2. INTRODUCCIÓN

Este proyecto nacio con el deseo de crear un proyecto dificil donde poder expandir mis conocimientos en el campo de programación y aunque al principio pense en Ruby on Rails, finalmente se elegio el stack MEAN (Mysql, Express, Angular, Node) con animo de aprender el stack de tecnologias usado en mi empresa de FCT y descubrir como funcionan las app de este tipo.

Se escogió una red social ya que me pareció un proyecto interesante e iterable, es decir, que en cada iteración me centraba en una característica en concreto, por lo que aun no consiguiendo todos mis objetivos sigue habiendo bastantes funcionalidades y si en el futuro quiero añadir mas cosas, se puede fácilmente ya que se ha intentado hacer lo mas modular posible.

También otro punto es la seguridad, ya que tenemos que estar logueados en todo momento y hacer peticiones al server con nuestro token tanto para rutas como para archivos.

3. OBJETIVOS

3.1.OBJETIVOS INICIALES

Este proyecto al ser una red social, ha tenido muchos objetivos. Casi todos se lograron pero otros se desecharon por falta de tiempo pero pueden encontrarse a la hora de hacer queries, en la documentacion del analisis y en el E-R de la base de datos para que si se itera en el futuro la actual versión y se quiere añadir esas funcionalidades sea lo mas rápido posible.

Los objetivos iniciales conseguidos son:

- Creación de un registro que se valida tanto en el lado cliente (mediante los validators de Angular) como en el lado servidor (mediante el método asignado a esa ruta). Cifrado de contraseña a la hora de crear el usuario y comprobación de que el nickname y correo sean únicos y no estén ya relacionados con otro usuario. En caso contrario, se muestra mensaje de error.
- Login: comprueba si el usuario y contraseña introducidos son correctos. Si todo esta bien, server devuelve un JWT que Angular guardara para las siguientes requests.
- Perfil: mostrara info del usuario como su nickname, nombre, followers... y fotos.
- Ajustes: donde el usuario podrá cambiar ciertos datos sobre el.
- Búsqueda de usuarios mediante su nickname: donde introducimos un texto y nos devuelve todos los usuarios con ese texto en su nickname.

-
- Subida de fotos: usuario puede subir fotos que se mostraran a los demás usuarios si perfil esta activo, no están bloqueado el otro usuario, perfil es publico y en caso de no serlo, usuario es follower del que ha subido la foto.
 - Comentarios: podrás comentar las fotos de los usuarios que puedas ver
 - Conversaciones: te aparecen las distintas conversaciones que has tenido donde enviás y/o recibes mensajes de algún usuario.
 - Notificaciones: cada vez que el usuario reciba un mensaje, petición o comentario en una foto le aparecerá una nueva notificación en su navbar.
 - Peticiones: podrás enviar peticiones a usuarios que no están entre tus following para añadirlos a tus following.
 - Bloqueos: un usuario puede bloquear a otro y ambos user no podrán ver el perfil del otro.
 - Implementación de un método de autenticación que nos asegura en todo momento que un usuario es quien dice ser.
 - Deshabilitar cuenta: si un usuario se deshabilita la cuenta, su perfil no podrá ser visualizado

Ahora, los no logrados:

- Perfiles públicos y/o privados: si un usuario es privado y no lo tienes en tu lista de following no se te mostrara.
- Suspensiones: este es la característica eliminada mas grande ya que en etapas conceptuales, se pensó en poder reportar a otros usuarios, que admin viesen el report y decidir si actuar o no. Se deshecho debido a que su implementación requeria incluso diferenciar entre usuarios normales y admin, numero de penalizaciones de un usuario, según estas cuanto tiempo... Aun así, referencias a esto puede ser encontrado en la base

de datos donde si en una futura iteración se desea hacer, la base de datos no sera un problema ya que tiene las relaciones y campos ya creadas.

3.2.OBJETIVOS SECUNDARIOS

Durante el desarrollo del proyecto se desecharon algunos objetivos anteriores y se crearon otras nuevos para mejorar lo creado como:

- Mostrar followers y following de un user
- Borrar un usuario de tus followers o followings.
- Foto de perfil del usuario: incluyendo mostrarla y cambiarla.
- Añadir titulo cuando se sube una foto
- Diferenciación de un usuario de otro en una conversación.
- Flash messages: que son cuadros de texto que nos aparecieran tras hacer una accion indicando error, exito o info sobre algun proceso.
- Diferenciación de las notificaciones nuevas de las antiguas.
- Cancelacion de una request no respondida.

4. MATERIAL

4.1. HARDWARE

4.1.1. Ordenador

Se ha usado un ordenador para hacer todos los pasos de este proyecto.

El ordenador funciona con un Ryzen 5, 8GB de ram y ssd.

4.2. SOFTWARE

4.2.1. Angular 6

Angular es uno de los framework de desarrollo JavaScript más potentes y populares en la actualidad.

Se trata de un framework MVC (Modelo Vista Controlador), desarrollado por Google que facilita la creación y programación del frontend de aplicaciones web

4.2.2. Node

Node.js fue creado por los desarrolladores originales de JavaScript. Lo transformaron de algo que solo podía ejecutarse en el navegador en algo que se podría ejecutar en los ordenadores como si de aplicaciones independientes se tratara.

4.2.3. **NPM**

npm es el Node Package Manager que viene incluido y ayuda a cada desarrollo asociado a Node. Durante años, Node ha sido ampliamente utilizado por los desarrolladores de JavaScript para compartir herramientas, instalar varios módulos y administrar sus dependencias.

4.2.4. **Express**

El framework proporciona un conjunto de herramientas para aplicaciones web, peticiones y respuestas HTTP, enrutamiento y middleware para construir y desplegar aplicaciones a gran escala y preparadas para la empresa.

También proporciona una herramienta de interfaz de línea de comandos (CLI) llamada Node Package Manager (NPM), donde los desarrolladores pueden obtener paquetes desarrollados. También obliga a los desarrolladores a seguir el principio de No te repitas (DRY).

El principio DRY pretende reducir la repetición de patrones de software, sustituyéndolos por abstracciones, o utilizando normalizaciones de datos para evitar la redundancia.

4.2.5. **Javascript**

En el contexto actual, JavaScript se utiliza para todo, gracias a la introducción de Node.js. Esta tecnología crea software robusto para empresas en todo el mundo. Por si fuera poco, organizaciones como LinkedIn y Medium lo implementan al construir plataformas para que los usuarios tengan acceso a sus servicios.

Lo que se puede hacer con JavaScript abarca diferentes tipos de software, como juegos, programas de computadora, aplicaciones web y hasta tecnologías de blockchain.

4.2.6. TypeScript

TypeScript (TS) es un lenguaje de programación construido a un nivel superior de JavaScript (JS). Esto quiere decir que TypeScript dota al lenguaje de varias características adicionales que hacen que podamos escribir código con menos errores, más sencillo, coherente y fácil de probar, en definitiva, más limpio y sólido.

4.2.7. NVM

nvm es un software que podemos instalar para contar con varias versiones distintas de NodeJS en un mismo ordenador. Gracias a nvm podemos instalar cualquier versión de NodeJS que necesitemos en un proyecto, ya sea una versión actual o una versión antigua. Además permite intercambiar la versión de NodeJS que tenemos activa en un momento dado, entre todas las que se hayan instalado en el sistema.

4.2.8. JWT

Un JSON Web Token es un token de acceso estandarizado en el RFC 7519 que permite el intercambio seguro de datos entre dos partes. Contiene toda la información importante sobre una entidad, lo que implica que no hace falta consultar una base de datos ni que la sesión tenga que guardarse en el servidor (sesión sin estado).

Por este motivo, los JWT son especialmente populares en los procesos de autenticación.

4.2.9. Multer

Cuando un cliente web sube un archivo a un servidor, generalmente lo envía a través de un formulario y se codifica como multipart/form-data. Multer es un middleware para Express y Node.js que hace que sea fácil manipular este multipart/form-data cuando tus usuarios suben archivos.

4.2.10. Git

Git es un sistema de control de versiones, un software que sirve básicamente para gestionar las versiones por las que va pasando el código de los proyectos.

Git es el más popular de los sistemas de control de versiones en la actualidad y una de las herramientas más indispensables para el desarrollo de proyectos.

4.2.11. Bootstrap

Bootstrap es un framework de desarrollo web gratuito y de código abierto. Está diseñado para facilitar el proceso de desarrollo de los sitios web responsivos y orientados a los dispositivos móviles, proporcionando una colección de sintaxis para diseños de plantillas.

4.2.12. CSS

CSS son las siglas en inglés para «hojas de estilo en cascada» (Cascading Style Sheets). Básicamente, es un lenguaje que maneja el diseño y presentación de las páginas web, es decir, cómo lucen cuando un usuario las visita. Funciona junto con el lenguaje HTML que se encarga del contenido básico de las páginas.

4.2.13. HTML

HTML es el lenguaje con el que se define el contenido de las páginas web. Básicamente se trata de un conjunto de etiquetas que sirven para definir el texto y otros elementos que compondrán una página web, como imágenes, listas, vídeos, etc.

4.2.14. MariaDB

MariaDB es un sistema de gestión de bases de datos que está muy relacionado con MySQL, ya que fue desarrollado por uno de los desarrolladores, Michael “Monty” Widenius. El objetivo de

su desarrollo fue el de mantener el software de gestión de base de datos en un modelo de software libre.

4.2.15. Visual Studio Code

Visual Studio Code (VS Code) es un editor de código fuente desarrollado por Microsoft. Es software libre y multiplataforma, está disponible para Windows, GNU/Linux y macOS. VS Code tiene una buena integración con Git, cuenta con soporte para depuración de código, y dispone de un sinnúmero de extensiones, que básicamente te da la posibilidad de escribir y ejecutar código en cualquier lenguaje de programación.

4.2.16. Bash

Bash (acrónimo de Bourne-Again Shell) es un intérprete de comandos y lenguaje de programación integrado que corre bajo el macroprocesador Shell de Unix.

Este programa ejecuta una a una las órdenes que el usuario pone en una ventana de texto o las que se encuentran contenidas en un script o bash script (archivo con todas las instrucciones), para luego devolver los resultados.

5. MÉTODO

En esta sección del documento procederé a hablar sobre la aplicación en si, explicando como se instala y como funciona:

5.1. INSTALACIÓN

Esta aplicación necesita: node v19 y v16, angular V6, nvm, xamp y un equipo con linux, recomiendo que sea Ubuntu ya que su desarrollo se hizo en el y la instalación se explicara para este, pero no es muy difícil hacerlo correr en otros S.O o distros, simplemente instalar lo siguiente para tu opcion:

Instalación Node:

- En la terminal: `sudo apt-get install -y nodejs`

Instalación angular V6:

Aquí decir que al ser una versión antigua, hay que hacer una ligera modificación.

En caso de tener una version de angular ya instalada, primero nos vamos a la consola y:

- `npm uninstall -g @angular/cli`
- `npm cache verify`

Ahora la instalación:

- `npm install -g @angular/cli@6 *`

Como vemos, esta se hace mediante NPM (Node project manager).

Si intentamos ejecutar ahora no funcionara y es debido a que las versiones mas recientes de Node son incompatibles con Angular 6. La solución a esto es NVM.

Instalación nvm: necesario para gestionar las distintas versiones de Node. *

- `curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash`
- Automaticamente esto debe añadir la configuracion del NVM a nuestro usuario, pero en caso de que no, al estar usando bash ponemos : `~/.bash_profile` y buscamos si tiene algo

```
como esto y si no se lo añadimos nosotros: export NVM_DIR="$([ -z "${XDG_CONFIG_HOME-}" ] && printf %s "${HOME}/.nvm" || printf %s "${XDG_CONFIG_HOME}/nvm")"
```

```
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
```

- Ahora recargamos la configuración de la shell: `source ~/.bashrc`
- Confirmamos que existe: `nvm -v`

Si nos muestra algo, ya hemos instalado nvm. En caso de fallo, se ruega buscar un manual online ya que puede ser por una ligera variación de la distribución o versión linux usada.

- Procedemos a instalar la versión de node 16 con nvm: `nvm install 16`

Con esto ya habríamos instalado la versión 16 de node junto con la mas reciente.

Instalación XAMPP:

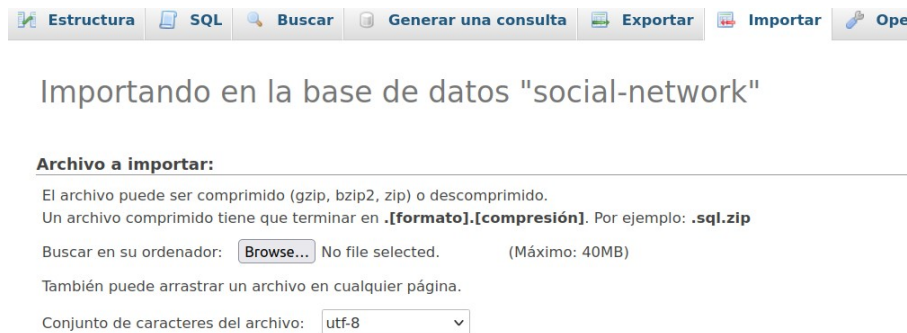
- Nos bajamos el instalador de:
<https://www.apachefriends.org/download.html#download-linux>
- Damos permisos de ejecución: `chmod 755 xampp-linux-*-installer.run`
- Instalamos: `sudo ./xampp-linux-*-installer.run`
- Con esto se abrirá el asistente y configuramos lo que necesitamos
- Tras instalarlo, arrancamos XAMPP visualmente o mediante el comando : `sudo /opt/lampp/lampp start`
- Con esto, si accedemos a la dirección (por defecto) de <https://localhost/phpmyadmin/>, ya podremos acceder a nuestra base de datos.

Muy bien, tras esto ya tendríamos todo el software necesario para empezar a trabajar en nuestra app.

Ahora procedemos a configurar la base de datos importándola y dando permisos a un usuario para usarla con un archivo. Esto ya queda a elección del usuario si lo quiere hacer así o de otra manera.

Configuración base de datos: importándola y dando permisos a un usuario para usarla con un archivo. La configuración del usuario sera puesta a modo de ejemplo y queda a elección del usuario si quiere usarla o no.

Para importarla nos vamos a phpmyadmin, creamos una nueva base de datos, accedemos a ella, en el menú de navegación le damos a importar y seleccionamos el archivo social-network-db.sql de la carpeta raíz del proyecto.



Le damos a continuar y ya tendremos nuestra base de datos con todas las tablas y triggers necesarios.

Crear usuario con todos los permisos solo para esta tabla: Este paso es opcional, pero recomendable por temas de seguridad. Para ello recomiendo ejecutar la sentencia sql que adjunto en el archivo "user_grants.txt" cambiando lo que sea necesario. Así hacemos que un usuario sea admin de esa tabla pero no pueda hacer nada mas en las otras.

Configuración .env de nuestro server: En la carpeta /server creamos un archivo .env con las siguientes variables :

- USER_DB = el usuario de nuestra base de datos
- PASSWORD_DB = la contraseña de ese usuario
- HOST_DB = si estais en local "localhost"
- PORT_DB = puerto donde esta corriendo vuestra DB
- DB_NAME = nombre de la base de datos
- PORT = puerto donde queréis que corra la aplicación de express

- SECRETORPRIVATEKEY = clave para firmar nuestros JWT
- FOLDER_IMAGES = "./static"
- FOLDER_IMAGES_USERS = "./static/files"

Las dos ultimas hacen referencia a carpetas de imágenes y es importante que se queden como se muestra.

Plantilla de esto en el apartado anexos.

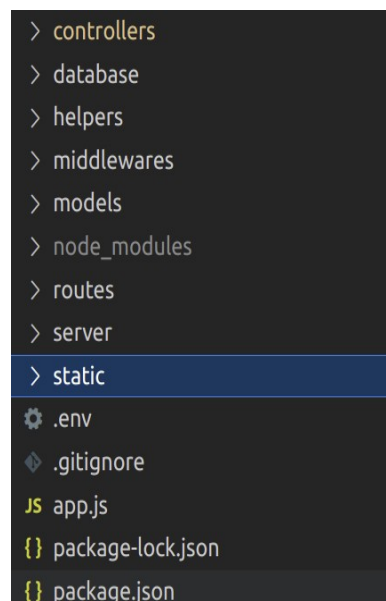
5.2. ESTRUCTURA APP

La carpeta de nuestra app esta diferenciada en dos partes /server y /social-media-app.

El funcionamiento de la misma se explica mediante un ejemplo en el anexo.

5.2.1. Estructura server

La carpeta /server, como su nombre indica, almacena todo lo relacionado con el server y vamos a explicar un poco su estructura



- **Controllers:** aquí están los métodos que harán operaciones sobre los modelos. Estos métodos se asocian a una ruta y nos aseguramos de que no sean ejecutados por cualquiera como veremos mas adelante
- **database:** contiene los archivos de conexión a nuestra base de datos.
- **helpers:** son operaciones que repetiremos varias veces (como crear notificación) y que llamaremos desde un método de los controllers.
- **middlewares:** esta es bastante importante, ya que comprueba que antes de ejecutar un método de los controllers que se haya en una ruta que todo este correctamente e incluso podemos añadir información a la request para el controller. Ejemplo: usuario hace una petición con un JWT, comprobamos que este sea correcto antes de acceder al controller. Si todo esta bien, añadimos el usuario a la request de forma que ya es accesible desde el controller y si el JWT es incorrecto o esta caducado devolvemos un mensaje de error con su status al usuario.
- **models:** representan los modelos de las tablas de la base de datos. Necesarios ya que estamos usando el ORM sequelize.
- **node_modules:** es la carpeta donde se bajaran las dependencias de node
- **routes:** es donde asociamos las rutas a los controladores.
- **server:** es el archivo de configuración de nuestro server y es donde hacemos diversas tareas como asignarle a una ruta un archivo de routes
- **static:** hace referencia a archivos como imágenes.
- **.env:** donde guardamos algunas constantes necesarias
- **.gitignore:** que archivos no queremos que se suban a git como por ejemplo los de node_modules
- **app.js:** es donde se crea el server y el archivo que usamos para arrancar la app.
- **package.json:** contiene las dependencias del proyecto.

5.2.2. Estructura cliente

Dentro de esta estructura, voy a destacar solo un par ya que la mayoría son carpetas de componentes o son similares a lo visto en el server (como package.json) y quedaría muy extenso.

Dentro de la carpeta src/app nos vamos a encontrar los componentes, services, guards, helpers, el app.component, app.module y app.routing.module.

Los services tendrán varias funciones pero sobre todo harán las peticiones HTTP con el HttpClient de angular ya que se considera mala practica que las haga el componente directamente.

Guards: los guards son muy parecidos a los middlewares del server, en este caso tenemos un guard que verifica que haya un token en el sessionstorage correspondiente al usuario y si no lo hay lo redirige al home. Este se usara en todas las rutas en las que debemos de tener una sesion abierta.

Helpers: donde crearemos nuestros interceptors que interceptan cada peticion HTTP que hagamos y uno se encarga de añadirle el JWT de nuestro session storage a la request y el otro, que si recibimos un codigo HTTP 401 nos devuelva a la pantalla home. Este código indica que el usuario no esta logueado y puede ser debido a que debe de refrescar el token iniciando sesión de nuevo.

Services: donde casi todos se encargan de hacer peticiones HTTP a la api mediante el HttpClient, pero tenemos otros como puede ser el flas-messages que se encarga de crear mensajes flash de error, info o exito para ciertos procesos como registrarse correctamente. El token-storage que se encarga de guardar y/o leer el usuario y token almacenados en nuestro session storage...

Assets: que es donde guardaremos las imagenes

Environments: finalidad parecida a la de .env del server

5.3. INSTALACIÓN DE LAS DEPENDENCIAS

Antes del primer deploy, tenemos que instalar sus dependencias

Nos situamos con la consola en las carpetas raíces de nuestros proyectos e insertamos "npm install". Es posible que para instalar el de Angular, al ser una versión antigua, sea necesario un downgrade de nuestra versión de node mediante "nvm use 16" y volver a ejecutar "npm install".

5.4. DESPLIEGUE DE LA APLICACIÓN

Para que la aplicación funcione necesitamos dos consolas, una para lanzar el cliente y otra para lanzar el server

En la carpeta server debemos de introducir:

"node app" o "nodemon app"

```
a@a:~/daw/proyecto-final/server$ node app
Servidor corriendo en puerto 8000
Executing (default): SELECT 1+1 AS result
```

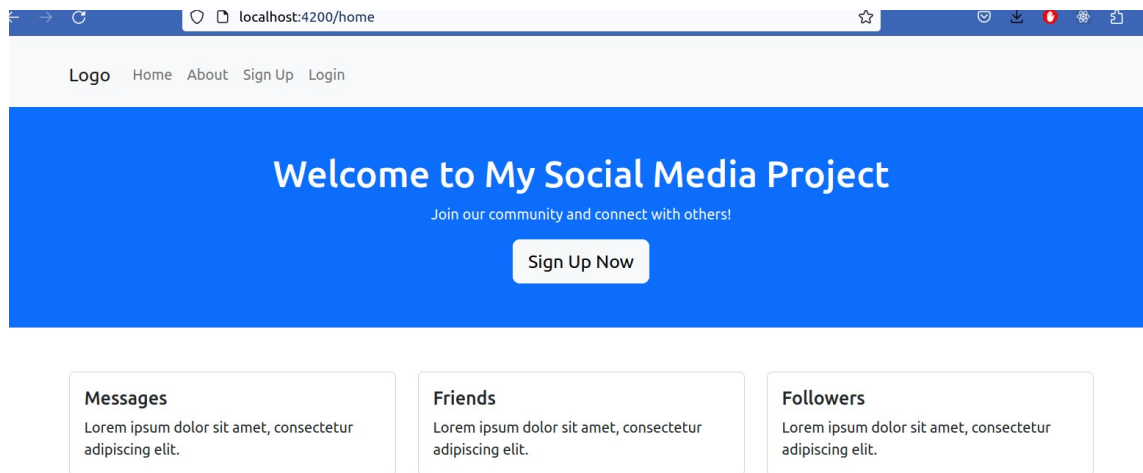
En la consola cliente:

"nvm use 16": para usar la version 16 de node

"ng serve": para arrancar el servidor de angular.

```
a@a:~/daw/proyecto-final/social-media-app$ nvm use 16
Now using node v16.20.0 (npm v8.19.4)
a@a:~/daw/proyecto-final/social-media-app$ ng serve
```

Nos vamos a nuestro navegador y accedemos a <http://localhost:4200/home>



What Our Users Say

6. TIEMPO DE EJECUCIÓN

Voy a explicar mediante rangos de fechas cuanto he tardado mas o menos en hacer cada cosa ya que se me hace imposible calcular el numero de horas al ser tantas.

Meteré la formación ya que aunque ha sido continua, especialmente en el caso de Angular, antes de empezar el proyecto tuve que ponerme de manera intensiva con ella al nunca haber usado Node o Angular:

-Formación en Node y Express : del 20 de Marzo al 05 de Abril

-Formación en Angular: del 05 al 25 de Abril.

-Análisis y documentación previa del proyecto: del 02 de Abril al 03 de Mayo. Aquí cree como iban a ser las clases, la base de datos, definir que quería hacer...Tarde tanto debido a que lo quería todo lo mas definido posible y definí cosas que finalmente no dieron tiempo a implementar pero su implementación no llevaría muchos problemas.

-Creación de la base de datos y creación del proyecto: del 03 al 05 de Mayo. Fueron días donde tuve que gastar muchas horas ya que hacer la base de datos y sus modelos en node usando sequelize (el ORM) fue muy costoso y aunque encontré una librería que automatizaba el proceso de crear los modelos a partir de la base de datos, tuve que aprender a usarlo y corregir sus fallos.

-Codificación: del 05 de Mayo al 13 de Junio.

7. **RESULTADOS**

Aquí pondré el resultado de mi red social:

-/home

Logo

Home

About

Sign Up

Login

Welcome to My Social Media Project

Join our community and connect with others!

Sign Up Now

Messages

Lorem ipsum dolor sit amet, consectetur adipiscing elit.


Friends

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Followers


Lorem ipsum dolor sit amet, consectetur adipiscing elit.

What Our Users Say




User 1

"This platform has changed the way I connect with others. Highly recommended!"



User 2

"I've met so many amazing people through this platform. It's been an incredible experience!"



User 3

"Finally, a social media platform that puts users first. I'm loving it!"

Terms of Service

Privacy Policy

Cookie Policy

Contact Us

About Us

FAQ

Formulario de registro:

Name:

Last Name:

Nickname:

Telephone:

Telephone is required.
Not a valid phone number.

Email:

nsadna@gma.com

Password:

Password must be at least 8 characters long.


Sex:

Create User


En este controllo que haya una longitud minima, que el telefono sea valido mediante una expresion regular, que sea un email... y si en el server el email o nickname ha sido ya tomado se le muestra al usuario el error.

Accedemos al usuario y vemos su perfil:

Logo Profile Settings Chats Upload photo Search Search Notifications



Juan Carrillo Gonzalez
@ejemplo
Followers: 0
Following: 0



Disfrutando en ordesa

Ajustes:

Settings

First Name

Last Name

Password

Visibility

Select Visibility

Save Settings

Change profile picture

Image

Browse...

No file selected.

Change Image

Drop account

Delete User

Conversaciones:

io

Profile Settings Chats Upload photo

Search

Conversations

@benito

Hola

Jun 15, 2023

Mostrar conversación.

Chat

Benito Jun 15, 2023

Hola

Juan Jun 15, 2023

Hola

Juan Jun 15, 2023

Cuanto tiempo

Búsqueda de usuarios mediante nickname:

User Search

Profile

Benito García Jiménez

@benito

Send Request

Send Message

Profile

Veronica Jimenez Lago

@ver_96

Send Request

Send Message

Ver perfil de otro usuario:

Veronica

Send Request

Send Message

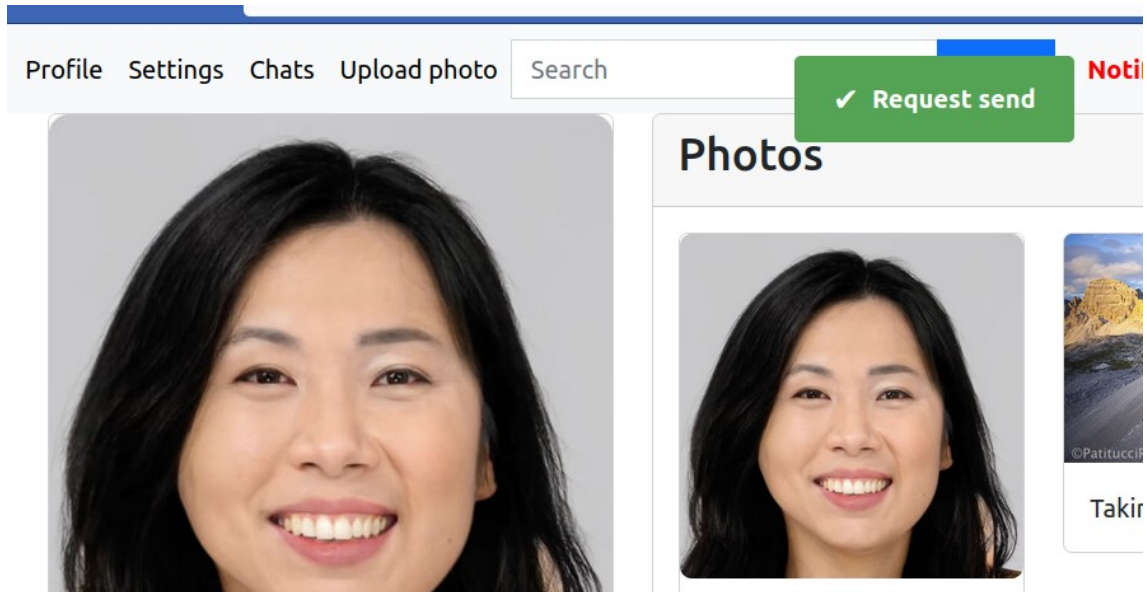
Report User

Block User

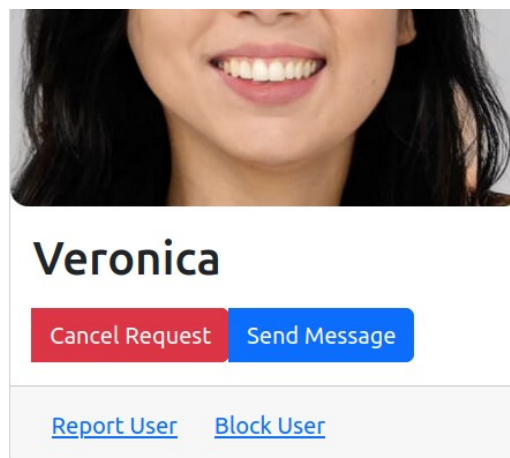
Photos

Taking a relax

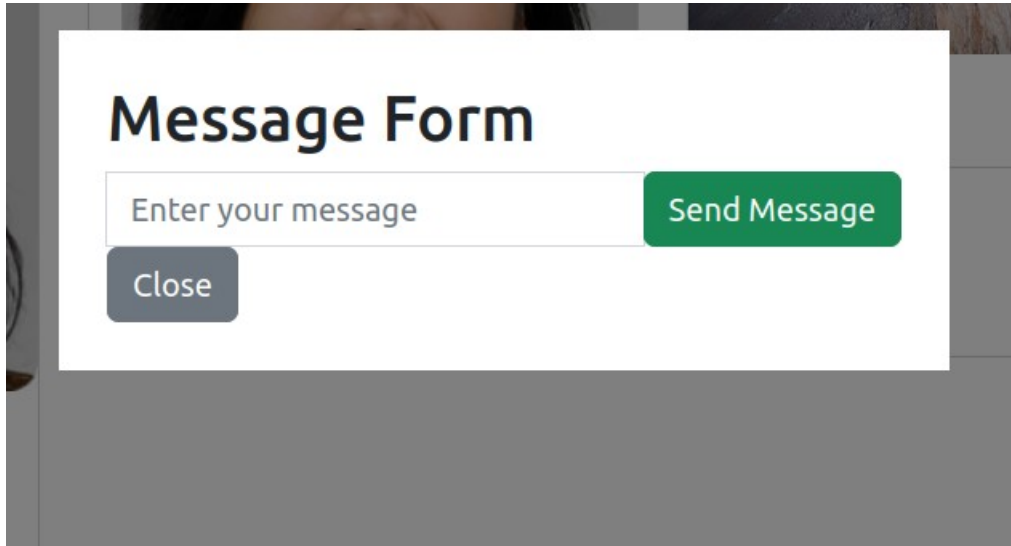
Mensajes flash, en este caso de envío de petición de amistad:



Cambio botones cuando tienes la petición pendiente:

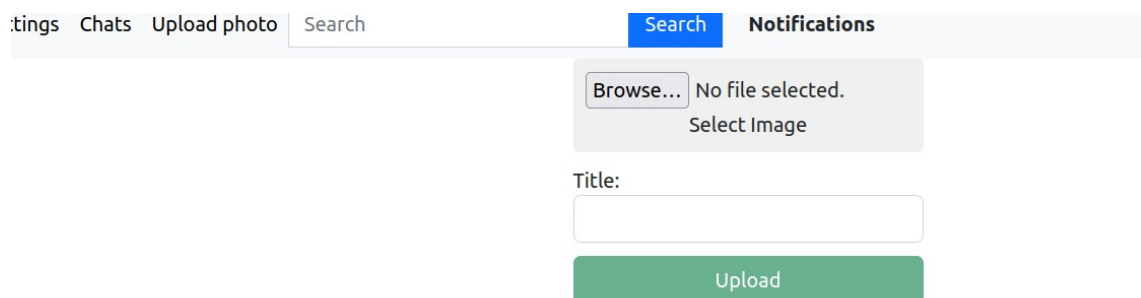


Envío de mensajes desde el perfil:



A screenshot of a 'Message Form' modal dialog. The dialog has a white background and is centered on a blurred background. It features a title 'Message Form' in a large, bold, dark font. Below the title is a text input field with the placeholder text 'Enter your message'. To the right of the input field is a green button labeled 'Send Message'. Below the input field is a grey button labeled 'Close'.

Subida de foto:




A screenshot of a photo upload form. At the top, there is a navigation bar with links: 'Settings', 'Chats', 'Upload photo', 'Search', 'Search', and 'Notifications'. Below the navigation bar, there is a 'Browse...' button. To the right of the button, there is a message: 'No file selected. Select Image'. Below this, there is a 'Title:' label and a text input field. At the bottom, there is a green button labeled 'Upload'.

Followers Requests:

Followers requests

Profile



Veronica Jimenez Lago


@ver_96

Accept Request

Reject Request

Followers:

followers



Veronica

@ver_96

Delete

Notificaciones:

Notifications

benito

@benito [sends you a request](#)

Jun 15, 2023

benito

@benito [sends you a message](#)

Jun 15, 2023

Comentarios:

Title: Disfrutando en ordesa

Uploaded by: [ejemplo](#)

Upload date: 2023-06-15



Comments

[ver_96](#) - 2023-06-16

Nice views!

Add a comment

Post

8. CONCLUSIONES

El proyecto para mi ha sido mucho mas difícil de lo esperado. La comunicación entre Angular y Node ha tenido una profundidad y dificultad que me ha sorprendido y hasta que aprendí como manejar esa comunicación, securizar con JWT, manejar las request en el server y las response en angular...ha sido algo mucho mas complejo de lo esperado, pero aun así me hallo satisfecho con el resultado ya que aunque esperaba mas características para la gran cantidad de horas invertidas, lo que hay es solido.

La verdad que acostumbrado a hacer proyectos de programación con frameworks como Ruby on Rails, Symphony o Spring boot, este proyecto me ha supuesto un cambio de paradigma en la programación mayor del esperado pero a lo largo de este proyecto he aprendido a manejar con cierta soltura y comprendido como este stack funciona lo que me servirá de gran ayuda en el mundo laboral.

Pablo Cruz Garrido

Curso 2022/23

9. ANEXOS

9.1. ARCHIVOS DE CONFIGURACIÓN

9.1.1. Archivo de configuración .env del server

```
USER_DB=""  
PASSWORD_DB=""  
HOST_DB=""  
PORT_DB=  
DB_NAME=""  
PORT=8000  
SECRETORPRIVATEKEY=""  
FOLDER_IMAGES="./static"  
FOLDER_IMAGES_USERS="./static/files"
```

9.1.2. Archivo de configuración del usuario de la base de datos

Hacer esto como una querie tras crear la base de datos cambiando los parámetros necesarios:

```
GRANT ALL PRIVILEGES ON *.* TO `proyect`@`localhost` IDENTIFIED BY PASSWORD  
'password' WITH GRANT OPTION;
```

```
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, REFERENCES, INDEX,  
ALTER, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, CREATE VIEW,  
SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, EVENT, TRIGGER ON `social-  
network`.* TO `proyect`@`localhost`;
```

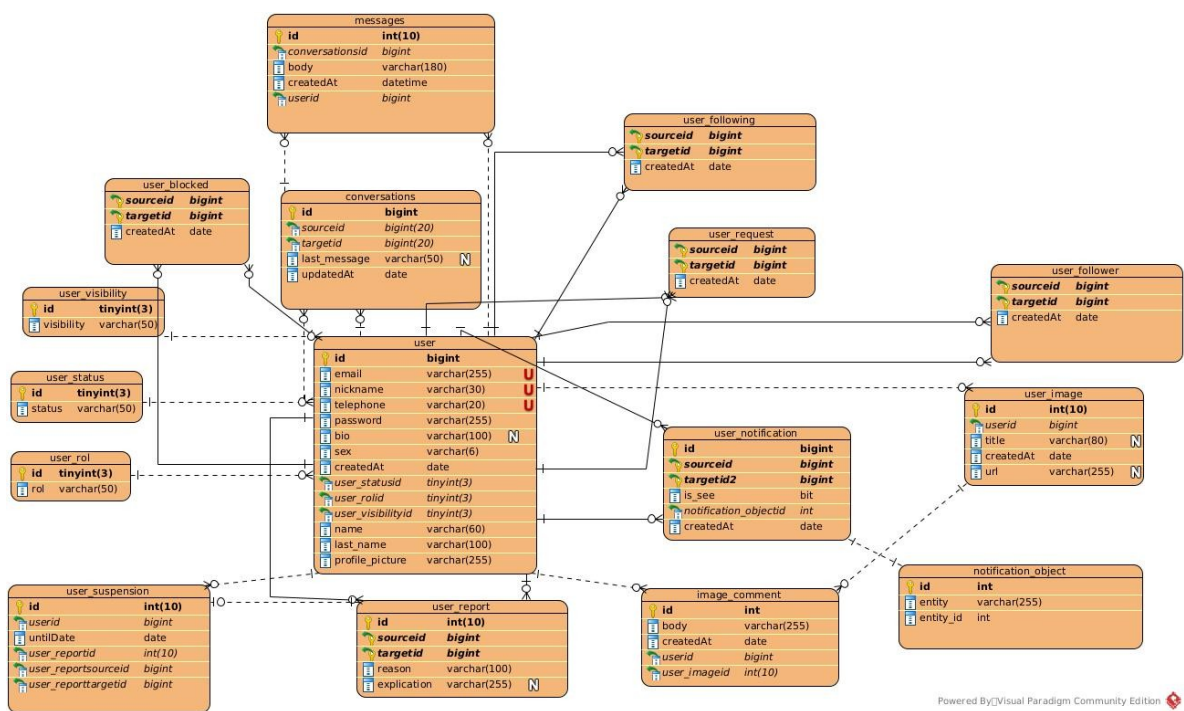
9.1.3. Archivo de configuración constantes Angular

En el archivo src/environments/environment.ts se encuentran la constante de conexión a la api, si cambiamos el puerto o host será necesario cambiarlo en este

```
export const environment = {  
  production: false,  
  apiUrl: 'http://localhost:8000/api',  
  imageUrl: '/images'  
};
```

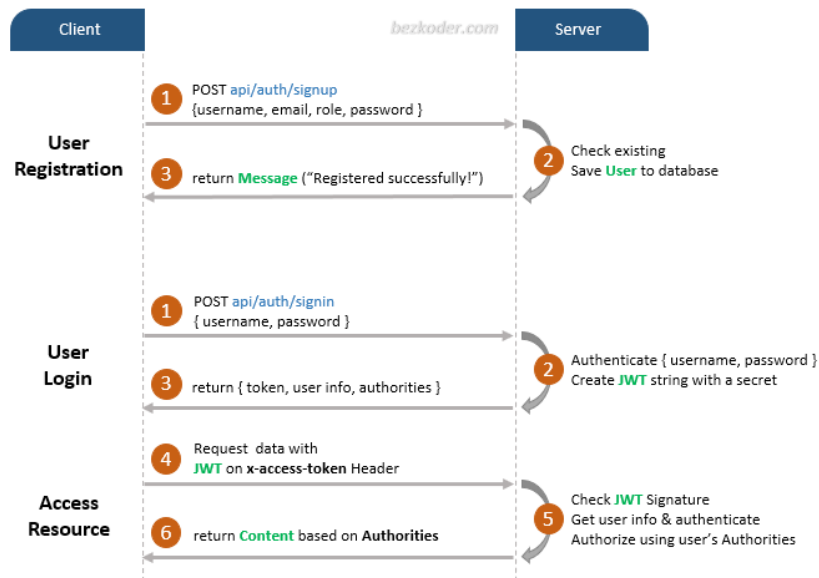
9.2. UML DE LA BASE DE DATOS

Creación del mismo usando Visual Paradigm Studio:

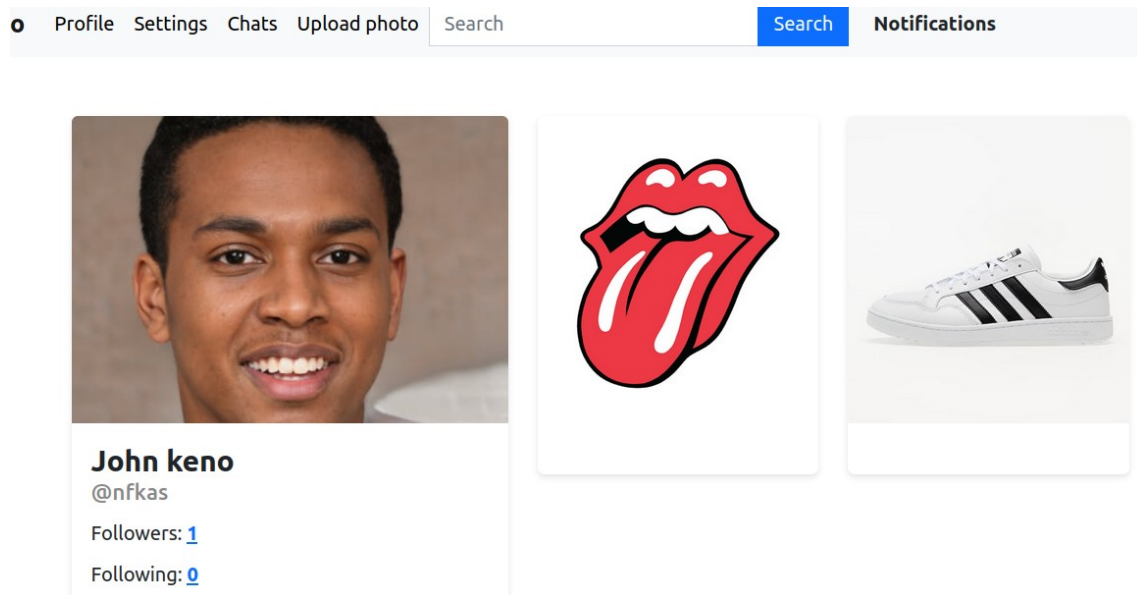


9.3. EXPLICACIÓN FUNCIONAMIENTO DE LA APP

El funcionamiento de la misma tiene dos partes fuertemente diferenciadas entre el cliente y servidor. El flujo de nuestra app seria como este pero sin las authorities:

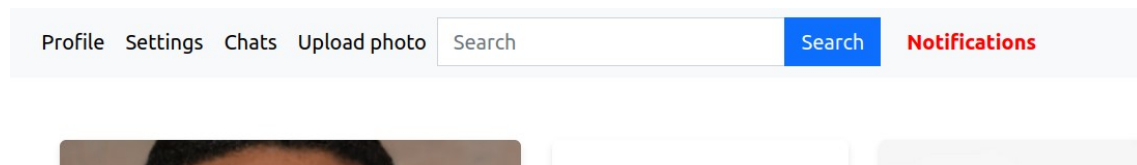


Voy a explicar este flujo usando las notificaciones de ejemplo.

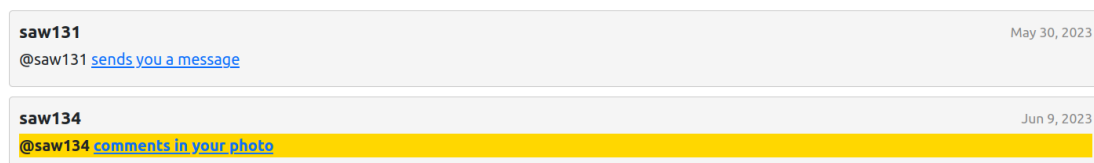


El usuario recibirá notificaciones cada vez que reciba un mensaje, comentario o petición.

Como vemos, en el navbar observamos como tenemos un apartado notifications que se pondra en rojo si tenemos notificaciones nuevas.



Ahora vamos a meternos y vemos lo siguiente:



Vemos como entre las notificaciones destaca la que no ha sido vista.

9.3.1. Funcionamiento Servidor: Node

En el archivo de configuración del server tenemos un método que asocia cada ruta a un archivo routes que llama a los middlewares y controladores.

```
routes() {  
  this.app.use( this.authPath, require('../routes/auth'));  
  this.app.use( this.usuariosPath, require('../routes/users'));  
  this.app.use( this.actionsPath, require('../routes/users-actions'));  
  this.app.use( this.messagesPath, require('../routes/messages'));  
  this.app.use( this.followsPath, require('../routes/friendship'));  
  this.app.use( this.imagesPath, require('../routes/images'));  
  this.app.use( this.registerPath, require('../routes/register'));  
}
```

Nos vamos al user-actions donde esta la ruta de las notificaciones.

```
router.get('/notifications',[validateJWT], getNotifications );
```

En este router estamos definiendo el metodo de la petición, la url, el array de middlewares (en este caso 1) y el método del controller que se llamara si todo esta correcto.

Como veis pasamos por dos routers, dando resultado a que la url a la que hay que hacer la petición al server sea: **localhost:8000/api/actions/notifications**

Ahora vamos a explicar el método del controller:

```
rollers > JS notificationController.js > [?] getNotifications
const getNotifications = async(req, res = response) => {
  //Catch the current user thanks to the JWT
  const targetid = req.user.id;

  try {
    //Catch the notification for that user
    const notifications = await UserNotification.findAll({
      where: {
        targetid: targetid
      },
      include: [
        {
          model: User,
          as: 'source',
          attributes: ['nickname', 'profile_picture']
        },
        {
          model: NotificationObject,
          as: 'notification_object',
          attributes: ['description']
        }
      ]
    });
  }
};
```

Tenemos dos variables en el método, la request y la response, que son objetos de express.

Vemos como al haber pasado por el middleware de validar el JWT, ese usuario esta dentro de la request y como tenemos la variable UserNotification que es del Sequelize y nos permite obtener todas las notificaciones, info del usuario que la emite y que tipo de notificación es.

Notar que tenemos que hacer un await ya que se trata de un método asíncrono y esta entre try y catch.

Tenemos que usar include a la hora de hacer la query ya que queremos traer los resultados de otros modelos que tenemos relacionado mediante clave ajena en la base de datos.

```
    },
    if(!notifications) return res.status(200)
    //Put is_see to true
    await UserNotification.update({ is_see: true},{
      where: {
        is_see: false,
        targetid: targetid
      }
    });

    return res.status(200).json({"ok":true, notifications});
```

si no hay notificaciones devolvemos una respuesta con status 200 (todo correcto) y si hay, cambiamos las notificaciones del usuario a vistas en la base de datos y devolvemos las notificaciones.

Ahora toca recoger esa info en Angular mediante los componentes y servicios para mostrarla.

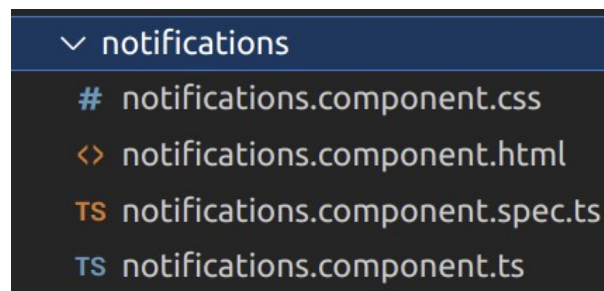
9.3.2. Funcionamiento Cliente: Angular

Empezamos explicando como funcionan los componentes que es lo que se mostrara el usuario.

Cada vez que creamos un componente, lo haremos en la consola con "**ng g c component-name --module app.module**".

Este, creara una carpeta en el src/app del proyecto con el nombre del componente, 3 archivos y la importación de ese modulo al app.module.ts para que este disponible en la app.

La estructura del componente creado sera:



Lo que hace cada archivo es:

-
- *.css : crear el estilo del html
 - *.html: define como sera visualmente el componente, pero no es un html normal ya que lee variables de la clase del componente (el archivo .ts) y tiene expresiones de Angular como if, pipes...
 - *.spec.ts: sirve para hacer testing en el componente
 - .ts: aqui definiremos como va a ser la clase del componente.

Como esto puede ser lioso, explicare como funcionan los componentes de angular mediante un ejemplo. Por ejemplo, en este caso el componente notifications hace una consulta a la API (en express) para pedirle info sobre las notificaciones del usuario actual y que le devuelva todos los registros de estas.

Lo primero, es nuestra clase (el .ts) el cual por defecto tendrá un constructor y el método ngOnInit (que hace referencia a uno de los ciclos de vida de un componente Angular).

```
import { Component, OnInit } from '@angular/core';
import { ActionsService } from '../services/actions.service';
import { FlashMessagesService } from '../services/flash-messages.service';

You, 7 seconds ago | 1 author (You)
@Component({
  selector: 'app-notifications',
  templateUrl: './notifications.component.html',
  styleUrls: ['./notifications.component.css']
})
export class NotificationsComponent implements OnInit {
  notifications: any;

  constructor(private actionsService: ActionsService,
    private flashMessagesService: FlashMessagesService) {}

  ngOnInit() {
    this.fetchNotifications();
  }
}
```

Vemos como tenemos los imports, el decorator `@Component` y la clase del componente.

Los decorators de Angular sirven para indicar metadatos de este componente. En este caso:

- **selector:** indicamos como se hará referencia a este componente desde un html de la app y en este ejemplo sería con la etiqueta `<app-notifications>`
- **templateUrl:** indica cual es el archivo html del componente
- **styleUrls:** el archivo css del componente

En la clase vemos como declaramos la variable `notifications`, donde tras llamar a la api guardaremos las notificaciones en ella y en html las enseñaremos.

En el constructor estamos inyectando los servicios que hemos creado.

Destacar que en Angular, cualquier cosa que sea una petición HTTP debe ser siempre mediante un service y no en el componente directamente.

Método ngOnInit: donde tras cargarse el componente, llamara a un método que nos sirve para cargar las notificaciones.

Puede que alguno este pensando ¿Porque no llamar directamente a ese método en el constructor? ¿No es algo que debe estar desde el momento que se muestra el componente? Y es que en Angular, el constructor debe ser solo para declarar los servicios y se considera una mala practica que haya alguna lógica en el constructor por cuestiones relacionadas al rendimiento de Angular. Se aconseja que sea siempre en el ngOnInit ya que este se ejecuta al haber cargado el componente reduciendo los problemas.

```
}  
  
fetchNotifications() {  
  this.actionsService.getNotifications().subscribe(  
    (response: any) => {  
      this.notifications = response.notifications;  
    },  
    (error) => {  
      this.flashMessagesService.showError('Error fetching notifications: ' + error);  
    }  
  );  
}
```

Este método es el encargado de mediante un método del actionsService hacer una request a la API para traer las notificaciones.

Como vemos no usamos un .then como en las promises si no un .subscribe y es que el paradigma de programación de Angular no es el clásico, si no que tiene una fuerte orientación al paradigma del programación reactiva.

Pero como vemos, una vez se ha tenido éxito, almacenamos en la variable las notificaciones y si no, mediante el flashMessagesService mostramos un dialogo de error ya que ha habido algún error.

Este es el HTML:

```
<div class="notification" *ngFor="let notification of notifications">
  <div class="notification-header">
    <h4 class="notification-title">{{ notification.source.nickname }}</h4>
    <span class="notification-date">{{ notification.createdAt | date: 'mediumDate' }}</span>
  </div>
  <!--If is a message -->
  <p class="notification-message" [ngClass]='{"new-notification": !notification.isMessage}'>
    @{{ notification.source.nickname }} <a [href]='"/chats/" + notification.entityId'>Ver mensaje</a>
  </p>
</div>
```

Como vemos esta usando cosas de angular hacemos un bucle for que nos mostrara cada notificación y sus datos.

9.4. CREDENCIALES DE PRUEBA

Si se quiere experimentar con la app, tras importar la base de datos en el login puedes introducir:

email: ejemplo@gmail.com

contraseña: hola1234

9.5. LINK AL REPOSITORIO

Link en github: <https://github.com/pcg98/social-media-node>

10. BIBLIOGRAFÍA Y WEBGRAFÍA

- <https://www.knowledgehut.com/blog/web-development/install-nodejs-on-ubuntu>
- <https://stackoverflow.com/questions/43344600/how-to-install-a-specific-version-of-angular-with-angular-cli>
- <https://www.freecodecamp.org/news/node-version-manager-nvm-install-guide/>
- <https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.bezkoder.com%2Fnode-js-angular-11-jwt-authentication%2F&psig=AOvVaw0okbxkCDv5LKRe0OYCxtC8&ust=1686929651659000&source=images&cd=vfe&ved=0CA4QjRxqFwoTCPCPsOfMxf8CFQAAAAAdAAAAABAD>
- <https://www.hostinger.es/tutoriales/que-es-npm>

-
- <https://www.tokioschool.com/noticias/que-es-angular/>
 - <https://kinsta.com/es/base-de-conocimiento/que-es-express/#qu-es-expressjs>
 - <https://profile.es/blog/que-es-typescript-vs-javascript/>
 - <https://desarrolloweb.com/home/nvm>
 - <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/json-web-token-jwt/>
 - <https://code.tutsplus.com/es/tutorials/file-upload-with-multer-in-node--cms-32088>
 - <https://desarrolloweb.com/home/git>
 - <https://www.hostinger.es/tutoriales/que-es-bootstrap>
 - <https://blog.hubspot.es/website/que-es-css#que-es>
 - <https://desarrolloweb.com/articulos/que-es-html.html>
 - <https://www.hostingplus.com.es/blog/que-es-mariadb-y-cuales-son-sus-caracteristicas/>
 - <https://openwebinars.net/blog/que-es-visual-studio-code-y-que-ventajas-ofrece/>