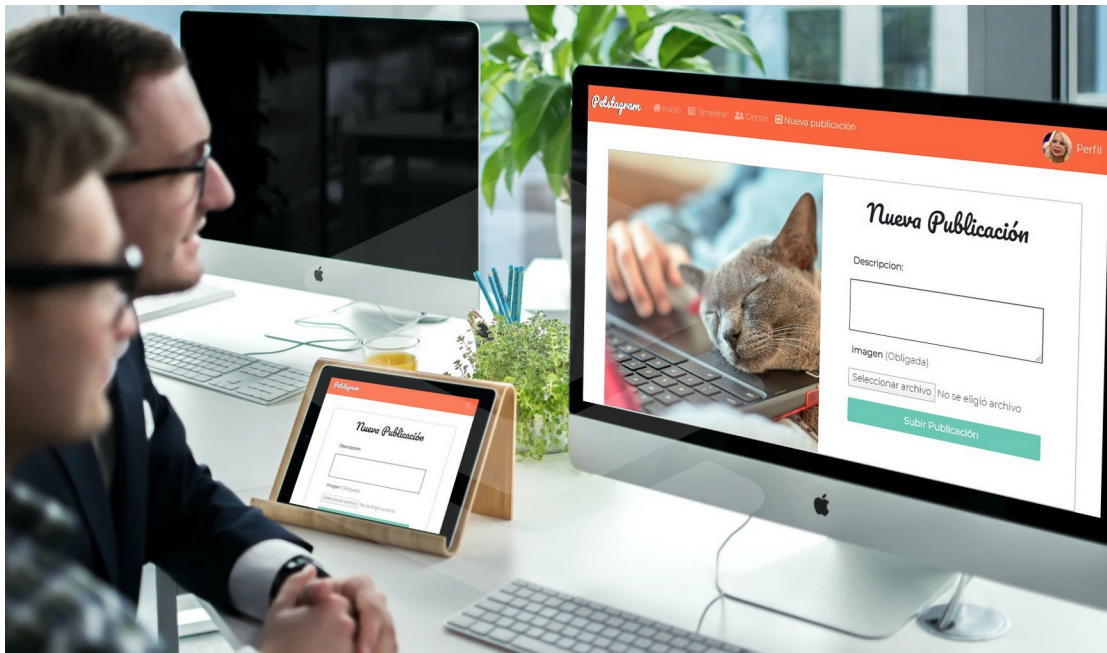


Petstagram



Proyecto final de curso de:
Eric Cano Nebot
Ciclo Superior D.A.W. (2º curso)
Promoción: 2019-2020

Índice

1. Resumen.....	3
2. Abstract.....	4
3. Backend.....	5
3.1 Preparación del entorno de desarrollo.....	5
3.2 Entorno de node.js.....	6
3.3 Creación de la base de datos.....	8
3.4 Creación de los modelos a usar en el backend.....	9
3.5 Creación de los controladores a usar en el backend.....	10
3.6 Creación de las rutas del en el backend.....	11
3.7 Login de usuarios.....	12
3.8 Servicio y tokens JWT.....	13
3.9. Middlewares de autenticación.....	15
3.10. Devolver los datos del usuario.....	17
3.11 Paginar Usuarios.....	18
3.12 Actualizar datos de usuario.....	19
3.13 Recibir el avatar de un usuario.....	22
3.14. Controlador de seguimiento.....	23
3.15 Publicaciones.....	28
3.16 Modulo de mensajes.....	31
4. Frontend.....	33
4.1 Creación del proyecto e instalación de librerías.....	33
4.2 Creación de usuarios.....	36
4.3 Login de Usuarios.....	37
4.4 Sección de gente de la plataforma.....	38
4.5 Mis datos.....	39
4.6 Cerrar Sesión.....	40
4.7 Perfil.....	41
4.8 Crear Publicaciones.....	42
4.9 Mensajería entre usuarios.....	43
5. Vision Comercial.....	45
5.1 Matriz DAFO.....	45
5.2 Marketing Mix.....	46
6 Bibliografía.....	46

1. Resumen.

Lo que se va a desarrollar en este proyecto va a ser una red social enfocada a la temática animal. En esta aplicación web se podrán realizar acciones muy comunes de otras redes sociales como Instagram o twitter. Para mi proyecto en concreto me fijo como modelo a Instagram para hacerla lo mas parecida posible pero a mi manera y con unas tecnologías diferentes. Aquí podrás hacer en la medida de lo probable lo mismo que se puede hacer en la red social nombrada anteriormente. Acciones como seguir a usuarios, ver sus publicaciones, enviar mensajes de texto asi como crear nuestras propias publicaciones y compartirlas con nuestros seguidores.

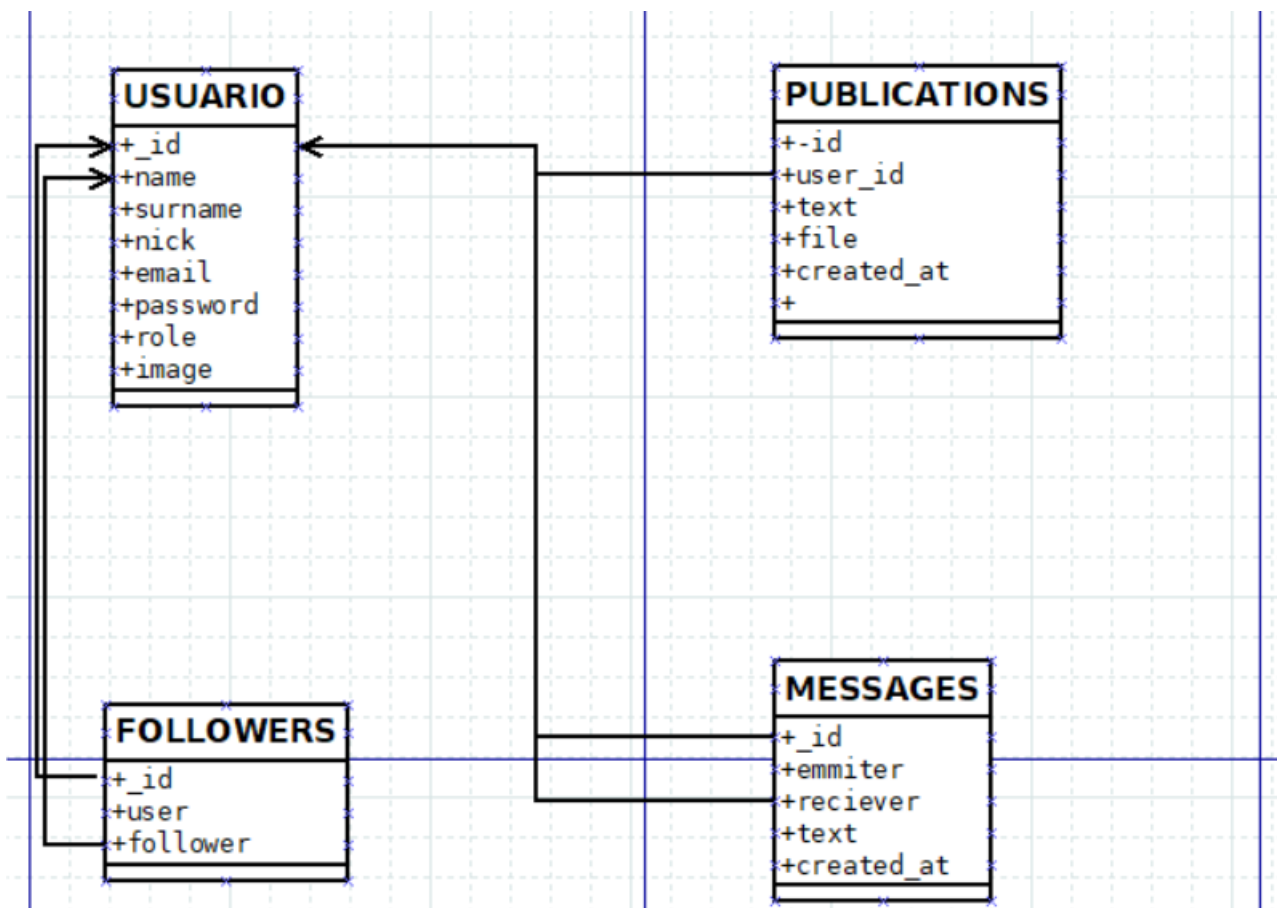
2. Abstract.

What is going to be developed in this project is going to be a social network focused on the animal theme. In this web application you can carry out very common actions from other social networks such as Instagram or twitter. For my project in particular I look at Instagram as a model to make it as similar as possible but in my own way and with different technologies. Here you can do as much as possible the same thing that can be done in the social network named above. Actions such as following users, viewing their posts, sending text messages as well as creating our own posts and sharing them with our followers.

3. Backend

3.1 Preparación del entorno de desarrollo

- Descargar e instalar Cygwin (Terminal en windows que funciona con comandos de linux).
- Descargar e instalar MongoDB la que será nuestra base de datos en el proyecto.
- Descargar e instalar Robo 3T, un administrador visual de MongoDB.
- Instalar Node.js
- Instalar DIA para hacer un diagrama del diseño de nuestra base de datos
- Instalar Postman para hacer peticiones.



3.2 Entorno de node.js

Crear un proyecto de node js con el comando.

```
npm -init
```

Creamos una carpeta llamada api en la que meteremos el archivo package.json que se nos ha generado.

Dentro de la carpeta api empezaremos a instalar dependencias y librerías que necesitamos en el proyecto

Instalamos Bcrypt para la encriptación de contraseñas de una manera sencilla

```
npm install bcrypt --save
```

Instalamos body-parser para convertir nuestras peticiones a objetos utilizables por javascript

```
npm install body-parser --save
```

Instalamos connect-multiparty para la subida de ficheros

```
npm install connect-multiparty --save
```

Instalamos Express que es el framework http que nos generara las rutas

```
npm install express --save
```

Instalamos jwt-simple nos va a servir para gestionar tokens e identificación

```
npm install jwt-simple --save
```

Instalamos moment para la generar de fechas y timestamp

```
npm install moment --save
```

Instalamos mongoose que es el ORM de mongoDB para trabajar con Node.js

```
npm install mongoose --save
```

Instalamos nodemon para refrescar el servidor cada vez que hagamos algún cambio en el código fuente.

```
npm install nodemon --save-dev (para usar esta dependencia solo en el desarrollo o local)
```

Instalamos mongoose-pagination para poder paginar los usuarios de nuestra base de datos

```
npm install --save mongoose-pagination
```

Instalamos moment pero en ANGULAR (!IMPORTANT) para poder gestionar las fechas

```
npm install moment --save
```

3.3 Creación de la base de datos.

Primero arrancamos el mongod.exe para la ejecución de la base de datos en segundo plano, luego usando el programa Robo 3T crearemos una nueva base de datos en nuestra conexión local con el nombre de curso_mean_social.

Creamos un index.js en la carpeta de API con las siguientes lineas.

```
'use strict'

var mongoose = require('mongoose');

mongoose.Promise = global.Promise;
mongoose.connect('mongodb://localhost:27017/curso_mean_social',{ useMongoClient: true })
  .then(() => {
    console.log("La conexion a la base de datos curso_mean_social correcta");
  })
  .catch(err => console.log(err));
```

Y con la instrucción siguiente lo arrancamos

- *node index.js*

Si todo va bien nos saldría el mensaje que que hemos puesto en el console log de lo contrario nos pintaría un error en la consola.

3.4 Creación de los modelos a usar en el backend.

Creamos una carpeta de models dentro de la carpeta api ,dentro vamos creando los diferentes modelos con las siguientes características:

(Modelo de publicaciones)

```
'use strict'

var mongoose = require('mongoose');
var Schema = mongoose.Schema;

// Creamos el modelo de publications
var PublicationSchema = Schema({
  text: String,
  file: String,
  created_at: String,
  user: {type: Schema.ObjectId, ref: 'User'}
});

/*Exportamos el modelo es primer parametro seria el nombre de la entidad y el segundo
los campos que va a tener el objeto*/

module.exports = mongoose.model('Publication', PublicationSchema);
```

3.5 Creación de los controladores a usar en el backend.

Creamos una carpeta de controllers dentro de la carpeta api .

Dentro vamos creando los diferentes controladores con las siguientes características:
(Controlador de usuarios).

```
'use strict'

var User = require('../models/user');

function home(req, res) {

  console.log(req.body)
  res.status(200).send({
    message: 'Hola soy home desde node.js'
  })
}

function pruebas(req, res) {

  console.log(req.body)
  res.status(200).send({
    message: 'Hola soy pruebas'
  })
}

module.exports = {
  home,
  pruebas
}
```

3.6 Creación de las rutas del en el backend.

```
'use strict'

var express = require('express');
var UserController = require('../controllers/user');
var api = express.Router();

api.get('/home',UserController.home);
api.get('/pruebas',UserController.pruebas);

module.exports =api;
```

Primero de todo tenemos que cargar en una variable express para poder usar el router luego cargamos el controlador y a partir de ahí inicializamos la en una variable el router de express.

Al final debemos siempre exportar los el router de express pa poder utilizarlo en app.js.

Registro de usuarios en Node.js y MongoDB

```
function saveUser(req, res) {
  var params = req.body;
  var user = new User();

  if (params.name && params.surname && params.email && params.nick && params.password) {
    user.name = params.name;
    user.surname = params.surname;
    user.email = params.email;
    user.nick = params.nick;
    user.role = 'ROLE_USER';
    user.image = null;
  }
}
```

Después de esto tenemos que hacer exportar el método y crear una ruta en `*routes/user.js*` para poder probarlo en Postman y ver que funciona bien.

```
api.post('/register',UserController.saveUser);
```

3.7 Login de usuarios.

Vamos a hacer ahora el login de usuarios, para ello creamos un método llamado `loginUser` que tendrá a su vez dos parámetros como siempre la petición(`req`) y la respuesta (`res`).

```
function loginUser(req, res) {  
  var params = req.body;  
  
  var email = params.email;  
  var password = params.password;  
  
  User.findOne({ email: email }, (err, user) => {  
    if (err) return res.status(500).send({ message: 'Error en la petición' });  
  
    if (user) {  
      bcrypt.compare(password, user.password, (err, check) => {  
        if (check) {  
          //devolver datos de usuario//  
          return res.status(200).send({ user });  
        } else {  
          return res.status(404).send({ message: 'No se ha podido identificar al usuario' });  
        }  
      })  
    }  
    else {  
      return res.status(404).send({ message: 'No se ha podido identificar al usuario!!' });  
    }  
  })  
}
```

Hecho esto hacemos como en anteriores métodos y exportamos la función y creamos una ruta para ello.

```
api.post('/login',UserController.loginUser);
```

Igualmente lo podemos probar en el Postman.

3.8 Servicio y tokens JWT.

Esto nos servirá para guardar los datos codificados en un token.

Primero de todo crearemos una carpeta llamada servicios y dentro un archivo llamado *jwt.js* con los siguientes datos.

```
var jwt = require('jwt-simple');
var moment = require('moment');
var secret = 'clave_secreta_curso_desarrollar_red_social_angular';
exports.createToken = function(user)
{
  var payload = {
    sub: user._id,
    name : user.name,
    surname : user.surname,
    nick : user.nick,
    email : user.email,
    role : user.role,
    image: user.image,
    iat: moment().unix(),
    exp : moment().add(30,'days').unix
  };

  return jwt.encode(payload,secret);
};
```

Explicación

Tenemos que cargar dos variables para poder usar jwt(creacion de tokens) y moment (para control de fechas)

Luego creamos la función a exportar con los datos del modelo de usuarios ademas de las siguientes propiedades

- La propiedad *sub* seria el identificador del token en jwt.
- La propiedad *iat* seria la encargada de decirnos cuando se ha creado el token
- La propiedad *exp* seria la encargada de decirnos el tiempo de expiación del token.

Y también le pasamos la variable secret que es una password que solo conocemos nosotros como programadores del backend.

Una vez creado el servicio tenemos que importarlo en el controlador para poder usarlo.

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

  if (user) {
    bcrypt.compare(password, user.password, (err, check) => {
      if (check) {
        //devolver datos de usuario//
        if (params.gettoken) {
          //devolver token y generar el token

          return res.status(200).send({
            token: jwt.createToken(user)
          })
        }
        else {
          //devolver datos de usuario//
        }
        user.password = undefined; /*Hacemos esto para no devolver la contraseña*/

        return res.status(200).send({ user });
      } else {
        return res.status(404).send({ message: 'No se ha podido identificar al usuario' });
      }
    })
  }
  else {
    return res.status(404).send({ message: 'No se ha podido identificar al usuario!!' });
  }
});
```

Podríamos probarlo en el postman para ver si funciona, para ello tenemos que pasarle una propiedad que sería getToken con el valor true y la petición nos devolvería un hash con toda la información del objeto.

3.9. Middlewares de autenticación.

Es un método que se va ejecutar antes de la función del controlador que nuestra api. Lo que va hacer es una comprobación para ver si el token es valido y en caso de que sea dejar acceder o hacer cualquier petición que pida el usuario en caso contrario nos dirá que ha habido algún error en la comprobación del token.

Nos creamos una carpeta llamada middlewares y en ella creamos un archivo con los siguientes datos:

```
'use strict'

var jwt = require('jwt-simple');
var moment = require('moment');
var secret = 'clave_secreta_curso_desarrollar_red_social_angular';

exports.ensureAuth = function(req,res,next){
{
  if(!req.headers.authorization)
  {
    return res.status(403).send({
      message: 'La cabecera no tiene autorizacion'
    });
  }

  var token = req.headers.authorization.replace(/[""]+/g,'');

  try{
    var payload = jwt.decode(token, secret);

    if(payload.exp <= moment().unix())
    {
      return res.stat(401).send({
        message : 'El token ha expirado'
      });
    }
  }catch(ex){
    return res.status(404).send({
      message : 'El token no es valido'
    });
  }

  req.user = payload;

  next();
}
```

Explicación

Primero tenemos que comprobar que la cabecera lleva la autenticación necesaria y si es así ya guardamos el token en una variable.

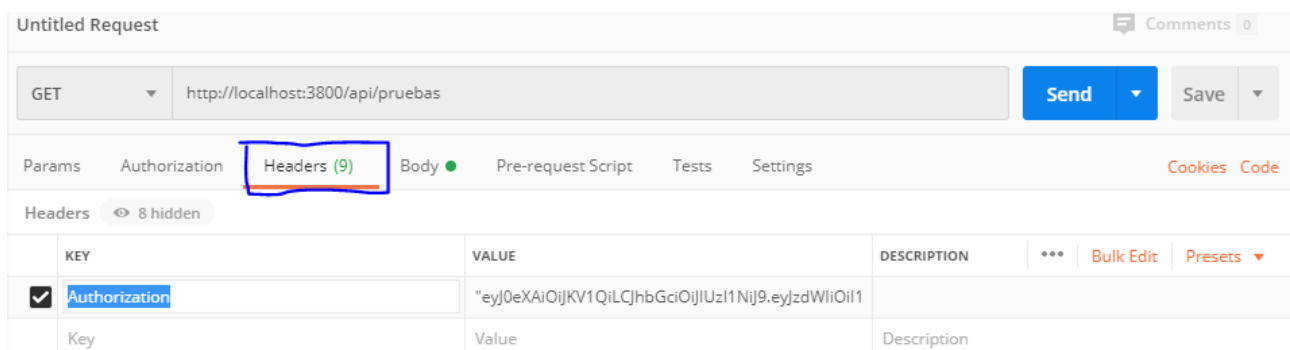
Una vez tengamos esto debes decodificar el token con el método decode que nos pedirá el mismo token y la clave secreta que hemos utilizado anteriormente para codificar.

Es recomendable utilizar el payload dentro de un try catch debido a que es sensible a errores.

Una vez hecho esto debemos volver al archivo de las rutas para cargar en middleware en una variable y poder utilizarla en alguna de nuestras rutas

```
var md_auth = require('../middlewares/authenticated');  
  
api.get('/pruebas',md_auth.ensureAuth, UserController.pruebas);
```

En el postman en la parte de *Headers* tenemos que pasarle un parámetro llamado Authorization y el valor sería el de nuestro token.



3.10. Devolver los datos del usuario

Para conseguir los datos del usuario tenemos que hacer una función muy sencilla

Para probarlo creamos la función en las rutas

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

En el postman le pasamos el ide por la URL y podemos comprobar que funciona.

```
function getUser(req,res)
{
    var userId = req.params.id;
    User.findById(userId,(err,user) => {
        if(err)
            res.status(500).send({
                message:'Error en la petición'
            });

        if(!user)
            res.status(404).send({
                message:'No existe el usuario'
            });

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

3.11 Paginar Usuarios

Vamos a hacer ahora un método para paginar todos los usuarios de nuestra red social.

```
//Devolver un listado de usuarios paginados//  
  
function getUsers(req,res)  
{  
  var identity_user_id = req.user.sub;  
  var page = 1;  
  if(req.params.page)  
  {  
    page = req.params.page;  
  }  
  
  var itemPerPage = 5;  
  
  User.find().sort('_id').paginate(page,itemPerPage, (err,users,total) => {  
    if(err)  
    {  
      return res.status(500).send({ message: 'Error en la petición' });  
    }  
  
    if(!users)  
    {  
      if (err) return res.status(404).send({ message: 'No hay usuarios en la plataforma disponibles' });  
    }  
  
    return res.status(200).send({  
      users,  
      total,  
      pages: Math.ceil(total/itemPerPage)  
    })  
  })  
}
```

Explicación

Tenemos el id del usuario guardado en la propiedad sub así que podemos ir sacándolos. Luego tenemos que devolver una función callback del User.find.sort

3.12 Actualizar datos de usuario

Vamos a crear la función para poder actualizar los datos de los usuarios. Lo que tenemos que tener en cuenta aquí es no actualizar la contraseña ya que es el dato mas delicado. Con `*delete*` conseguimos esto.

```
// Editar datos de usuarios

function updateUser(req, res) {
  var userId = req.params.id;
  var update = req.body;

  // Tenemos que quitar el parametro de la contraseña ya que es un dato muy delicado //

  delete update.password;

  if (userId !== req.user.sub) {
    return res.status(500).send({ message: 'No tienes permisos para actualizar' });
  }

  User.findByIdAndUpdate(userId, update, {new:true}, (err, userUpdated) => {
    if(err)
    {
      return res.status(500).send({message:'Error en la petición'});
    }
    if(!userUpdated)
    {
      return res.status(404).send({message:'No se ha podido actualizar el usuario'});
    }

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

Luego tenemos que crear la ruta para poder actualizar que en este caso tiene que ser put:

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

Si quisiéramos probarlo en el Postman primero deberíamos hacer una petición de login para conseguir el token y ya luego pasar los datos como tocan

Pasamos ahora a subir un avatar en nuestro perfil de usuario. Lo primero que tenemos que hacer es vigilar que el usuario que esta logueado es quien dice ser. Luego de esta pequeña comprobación tenemos que guardar el path del archivo a subir en variables.

```
function uploadImage(req, res) {  
    var userId = req.params.id;  
  
    if (req.files) {  
        var file_path = req.files.image.path;  
        var file_split = file_path.split('\\');  
  
        var file_name = file_split[2];  
        var ext_split = file_name.split('\\.');  
  
        var file_ext = ext_split[1];  
    }  
}
```

Explicación

- La primera variable nos recoge toda la ruta del archivo donde se va a guardar, en el controlador veremos como guardar utilizando la librería multipart.
- En la segunda variable cortaremos el nombre del archivo subido (uploads/users/foto.png).
- En la tercera nos quedaríamos solamente con el nombre del archivo que es lo que realmente queremos.
- Y en la ultima separaríamos la extensión en una variable aparte para hacer las siguientes comprobaciones para saber que se trata de una imagen.

```

if (file_ext == 'png' || file_ext == 'PNG' || file_ext == 'jpg' || file_ext == 'jpeg' || file_ext == 'gif') {
  //Actualizar avatar del usuario
  console.log('Archivo valido');
  User.findByIdAndUpdate(userId, {image:file_name}, { new: true }, (err, userUpdated) => {
    if (err) {
      return res.status(500).send({ message: 'Error en la petición' });
    }
    if (!userUpdated) {
      return res.status(404).send({ message: 'No se ha podido actualizar el usuario' });
    }
    return res.status(200).send({
      user: userUpdated
    });
  });
}
else {
  console.log('Archivo no valido');
  return removeFilesOfUploads(res, file_path, 'Extension no valida');
}
}

```

Una vez hecho esto importamos multipart y creamos un middleware para poder subir archivos en node.js y luego creamos la ruta para poder probarlo:

```
var multipart = require('connect-multiparty');
```

```
var md_upload = multipart({ uploadDir: './uploads/users' });
```

Ruta

```

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

```

Como vemos en la imagen superior para poder pasar mas de un middleware tenemos que pasarlo como una colección.

3.13 Recibir el avatar de un usuario

Para recibir el avatar de un usuario vamos a hacer lo siguiente

```
//Devolver imagen del usuario//  
  
function getImageFile(req, res) {  
  var image_file = req.params.imageFile;  
  
  var path_file = 'uploads/users/' + image_file;  
  console.log(path_file)  
  fs.exists(path_file, (exists) => {  
  
    if (exists) {  
      res.sendFile(path.resolve(path_file));  
    }  
    else {  
      res.status(200).send({ message: 'No existe la foto' });  
    }  
  
  });  
}
```

Explicación

Recibiremos los datos por la URL y la concatenaremos con nuestro el nombre de la imagen para comprobar que existe. Luego solo tenemos que hacer una comprobación y lo tendríamos, creamos la ruta por GET para probarlo en POSTMAN

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

3.14. Controlador de seguimiento

Este controlador de follows nos va a permitir seguir a usuarios seguirse y dejar de seguirse así como ver listados de quien nos sigue y a quien seguimos.

Creamos el controlador de follows y las rutas de follows igual que hemos hecho con usuarios.js

****follow routes****

```
'use strict'

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

api.post('/follow', md_auth.ensureAuth, FollowController.saveFollow);
api.delete('/follow/:id', md_auth.ensureAuth, FollowController.deleteFollow);
api.get('/following/:id?:page?', md_auth.ensureAuth, FollowController.getFollowingUsers);
module.exports = api;
```

En app.js tenemos que cargar el controlador para poder usar la rutas

```
var user_routes = require('./routes/user');
var follow_router = require('./routes/follow');

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

Luego podemos ir creando los diferentes métodos que nos harán falta en el controlador.

Seguir a un usuario

Vamos a crear un método para que un usuario permita seguir a otro:

```
'use strict'

//var path = require('path');
//var fs = require('fs');
var mongoosePaginate = require('mongoose-pagination');

var User = require('../models/user');
var Follow = require('../models/follow');

//Seguir usuarios//
function saveFollow(req, res) {

  var params = req.body;

  var follow = new Follow();

  //Es el usuario autenticado en el middleware (nosotros) //
  follow.user = req.user.sub;
  //Es el usuario al cual queremos seguir que nos llegara por la petición//
  follow.followed = params.followed;

  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 });
  })
}
```

Luego lo que tenemos que hacer es crear la ruta para probarlo

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


Le pasaríamos el middleware de autenticación para saber quienes somos y por parámetro a quien queremos seguir.

En la base de datos se nos quedaría así guardada la colección:

```
/* 1 */
{
  "_id" : ObjectId("5e8c9fe09007a33e24ff2d2f"),
  "user" : ObjectId("5e8111d0668bf1129c51cb30"),
  "followed" : ObjectId("5e8111db668bf1129c51cb33"),
  "__v" : 0
}

/* 2 */
{
  "_id" : ObjectId("5e8ccae7ca57f44194bd29e5"),
  "user" : ObjectId("5e8111d0668bf1129c51cb30"),
  "followed" : ObjectId("5e810dac4525ac1ab4865a3c"),
  "__v" : 0
}
```

- En ***azul*** podemos ver que es el mismo usuario el que sigue a los ***rojos*** que serian dos usuarios distintos

05-04-2020

Dejar de seguir

Vamos a crear el método para dejar de seguir a usuarios.

```
//Dejar de seguir usuarios//

function deleteFollow(req, res) {
  //el usuario logeado//
  var userId = req.user.sub;
  //el usuario que vamos a dejar de seguir//
  var follwId = req.params.id;

  Follow.find({ 'user': userId, 'followed': follwId }).deleteOne(err => {
    if (err)
      return res.status(500).send({ message: 'Error al dejar de seguir' });
    return res.status(200).send({ message: 'El follow a sido eliminado correctamente' });
  })
}
```

Explicación

Lo único que hacemos es ver el usuario que somos nosotros y ver si existe el follow para dejar de seguirlo (en este caso borrar el seguimiento de la base de datos).

Y su respectiva ruta que en este caso es un DELETE:

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

Listado de usuarios que sigo

Necesitamos un listado paginado de los usuarios que nos siguen para eso creamos el siguiente método:

```
//Paginacion de usuarios que seguimos//
function getFollowingUsers(req, res) {
  var userId = req.user.sub;
  if (req.params.id && req.params.page) {
    |   userId = req.params.id;
  }

  var page = 1;
  if (req.params.page) {
    |   page = req.params.page;
  } else {
    |   page = req.params.id;
  }

  var itemsPerPage = 2;

  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 < 1) return res.status(404).send({ message: 'No sigues a ningun usuario' });
    |   return res.status(200).send({
    |     |   total: total,
    |     |   pages: Math.ceil(total / itemsPerPage),
    |     |   follows
    |   });
  });
}
```

Solo necesitamos el usuario del cual queramos sacar a los seguidores y una vez lo tengamos hacemos el populate (que seria como un join en bases de datos sql) y luego lo paginamos.

Luego creamos la ruta.

```
api.get('/following/:id?/:page?',md_auth.ensureAuth,FolllowController.getFollowingUsers);
```

El resultado seria este en el postman.

```
1 {
2   "total": 2,
3   "pages": 1,
4   "follows": [
5     {
6       "_id": "5e8c9fe09007a33e24ff2d2f",
7       "user": "5e8111d0668bf1129c51cb30",
8       "followed": {
9         "_id": "5e8111db668bf1129c51cb33",
10        "name": "Manuel",
11        "surname": "Cano",
12        "email": "manuel@gmail.com",
13        "nick": "manuelito",
14        "role": "ROLE_USER",
15        "password": "$2a$10$mT20BV9oo5EgB9FueU2q8.ojPxtov.0oA4UiTnse2iUgdH3FeVoBS",
16        "__v": 0
17      },
18      "__v": 0
19    },
20    {
21      "_id": "5e8ccae7ca57f44194bd29e5",
22      "user": "5e8111d0668bf1129c51cb30",
23      "followed": {
24        "_id": "5e810dac4525ac1ab4865a3c",
25        "name": "Eric",
26        "surname": "Cano",
27        "email": "eric@gmail.com",
28        "nick": "Xatito",
29        "role": "ROLE_USER",
30        "password": "$2a$10$twB0EcsMc26VH35.f0C200KrIcr5j/3LQ2VotLICv6gnJrhXwuHAa",
31        "__v": 0,
32        "image": "yea3SpUF5PYkna4cgnWPJMKV.PNG"
33      },
34      "__v": 0
35    }
36  ]
37 }
```

Podemos ver que este usuario sigue a dos usuarios y lo ha paginado todo en una misma pagina.

3.15 Publicaciones

Ahora vamos a implementar un método que nos permita crear publicaciones.

```
function savePublication(req, res) {
  var params = req.body;

  if (!params.text) {
    return res.status(200).send({ message: "No puedes enviar una publicaion vacia" });
  }

  var publication = new Publication();

  publication.text = params.text;
  publication.file = 'null';
  publication.user = req.user.sub;
  publication.created_at = moment().unix();

  publication.save((err, publicationStored) => {
    if (err) {
      return res.status(500).send({ message: "Error al guardar la publicacion" });
    }

    if (!publicationStored) {
      return res.status(404).send({ message: "No se ha podido guardar la publicacion" });
    }

    return res.status(200).send({ publication: publicationStored });
  })
}
```

Con esto hecho solo nos faltaría exportar el método y utilizarlo en la en archivo de rutas para poder probarlo con el postman

Subir imágenes a la publicaciones

Para subir una imagen a nuestro objeto de publicaciones tenemos que tener varias cosas en cuenta como partir el archivo que nos llega en el nombre, el formato y donde se va a guardar para tenerlo todo lo controlado posible.

```
function uploadImage(req, res) {  
  var publicationId = req.params.id;  
  
  if (req.files) {  
    var file_path = req.files.image.path;  
    var file_split = file_path.split('\\');  
  
    var file_name = file_split[2];  
    var ext_split = file_name.split('.');  
  
    var file_ext = ext_split[1];  
  
    if (file_ext == 'png' || file_ext == 'PNG' || file_ext == 'jpg' || file_ext == 'jpeg' || file_ext == 'gif') {  
      Publication.findOne({ 'user': req.user.sub, '_id': publicationId }).exec((err, publication) => {  
        console.log(req.user.sub);  
        console.log(publicationId);  
        console.log(publication);  
  
        if (publication) {  
          //Actualizar el documento de la publicacion del usuario//  
          console.log('Archivo valido');  
          Publication.findByIdAndUpdate(publicationId, { file: file_name }, { new: true }, (err, publicationUpdated) => {  
            if (err) {  
              return res.status(500).send({ message: 'Error en la peticion' });  
            }  
            if (!publicationUpdated) {  
              return res.status(404).send({ message: 'No se ha podido actualizar el usuario' });  
            }  
            return res.status(200).send({  
              publication: publicationUpdated  
            });  
          });  
        }  
        else {  
          return removeFilesOfUploads(res, file_path, 'No tienes permisos para actualizar');  
        }  
      });  
    }  
    else {  
      console.log('Archivo no valido');  
  
      return removeFilesOfUploads(res, file_path, 'Extension no valida');  
    }  
  }  
  else {  
    return res.status(200).send({ message: 'No se ha podido subir la imagen' });  
  }  
}
```

En la carpeta uploads/publication podemos ver las imágenes que hemos subido.

GetPublications y deletePublications

Con estos dos métodos podemos coger una publicación de la base de datos para por ejemplo poder pintarla en la vista o para borrarla de la misma base de datos.

```
function getPublication(req, res) {
  var publicationId = req.params.id

  Publication.findById(publicationId, (err, publication) => {
    if (err) {
      return res.status(500).send({ message: "Error al devolver publicaciones" });
    }

    if (!publication) {
      return res.status(404).send({ message: "No hay publicaciones" });
    }
    return res.status(200).send({ publication })
  })
}

function deletePublication(req, res) {
  var publicationId = req.params.id

  Publication.find({ 'user': req.user.sub, '_id': publicationId }).deleteOne((err, publicationRemoved) => {
    if (err) {
      return res.status(500).send({ message: "Error al devolver publicaciones" });
    }

    console.log(publicationId);
    return res.status(200).send({ publication: publicationRemoved });
  });
}
```

3.16 Modulo de mensajes

Vamos a hacer todo lo necesario para poder crear mensajes de texto que se comuniquen entre los usuarios de la plataforma.

Lo priermo que haré será crerar un funcion para guardar un mensaje con las propiedades del emisor del mensaje y el receptor para saber a quien enviamos el mensaje y quien lo recibe

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

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

  var message = new Message();

  message.emmitter = req.user.sub;
  message.receiver = params.receiver;
  message.text = params.text;
  message.created_at = moment().unix();
  message.viewed = 'false';

  message.save((err,messageStored) =>
  {
    if(err)
    {
      return res.status(500).send({message:'Error en la peticion'});
    }

    if(!messageStored)
    {
      return res.status(500).send({message:'Error al enviar el mensaje'});
    }
    return res.status(200).send({messageStored});
  });
}
```

Otra de los métodos que necesitaremos para añadir al frontend sería saber que mensajes hemos recibidos nosotros y que mensajes hemos enviado. Esto lo conseguimos con dos funciones muy parecidas:

Mensajes recibidos:

```
function getReceivedMessages(req,res)
{
  var userId = req.user.sub;
  var page = 1;

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

  var itemsPerPage = 4;
  Message.find({receiver:userId}).populate('emmitter','name surname image nick _id').sort('-created_at').paginate(page,itemsPerPage, (err,messages,total) =>{
    if(err)
    {
      return res.status(500).send({message:'Error en la petición'});
    }

    if(!messages)
    {
      return res.status(404).send({message:'No hay mensajes'});
    }

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

Mensajes enviados

```
function getEmmitMessages(req,res)
{
  var userId = req.user.sub;
  var page = 1;

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

  var itemsPerPage = 3;
  Message.find({emmitter:userId}).populate('emmitter receiver','name surname image nick _id').sort('-created_at').paginate(page,itemsPerPage, (err,messages,total)
  {
    if(err)
    {
      return res.status(500).send({message:'Error en la petición'});
    }

    if(!messages)
    {
      return res.status(404).send({message:'No hay mensajes'});
    }

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


4. Frontend

4.1 Creación del proyecto e instalación de librerías

Tenemos que arrancar un proyecto de angular desde una terminal

- *ng new clientmene*

Luego de esto tenemos que instalar algunas librerías que necesitaremos en nuestro proyecto de angular.

-*Bootstrap*

-*Jquery*

-*Material*

Lo primero que vamos a crear será la barra de navegación de nuestra aplicación, con el comando.

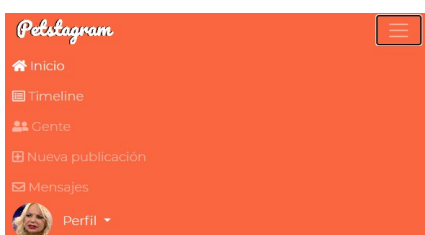
- *ng g c navbar*

Luego la maquetaríamos

a nuestro gusto, aquí dejo como nos quedaría.



Y la versión responsive para distintos tamaños como el móvil.



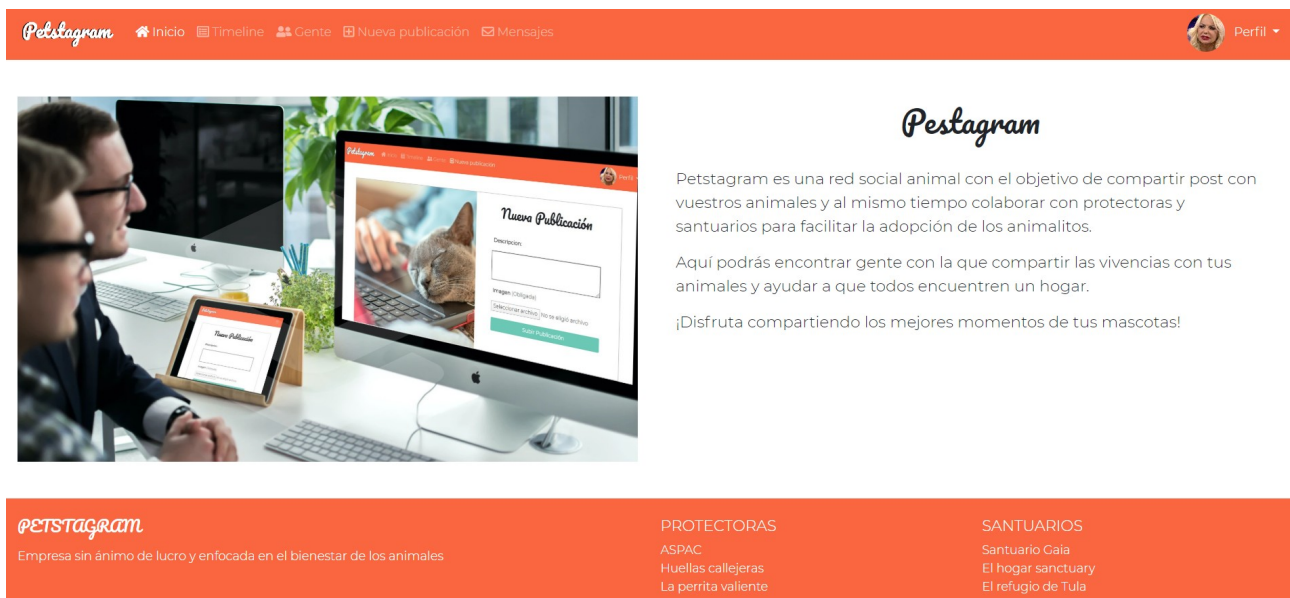
Así como hemos creado nuestra navbar vamos ahora a crear diferentes partes de nuestra web empezando con la pantalla de inicio o home que nos explica un poco de que va esta web.

Creamos el componente home así como el componente de footer.

-ng g c home


-ng g c footer

Y los maquetamos para que muestre un poco de que va nuestra web.



Una cosa importante que hay que tener en cuenta a la hora de crear este tipo de web es que queremos que el navbar y el footer se vean en todos los componentes así que tenemos que indicarse-lo a angular de la siguiente manera.

Vamos al archivo que esta en de app.component que es el archivo que llama a todos los componentes de la web y hacemos que todos los componentes a ser llamados estén encapsulados entre estos dos componentes

```
src > app >  app.component.html > ...  
1 <app-navbar></app-navbar>  
2  
3 <router-outlet></router-outlet>  
4  
5 <app-footer class="mt-auto"></app-footer>  
6 |
```

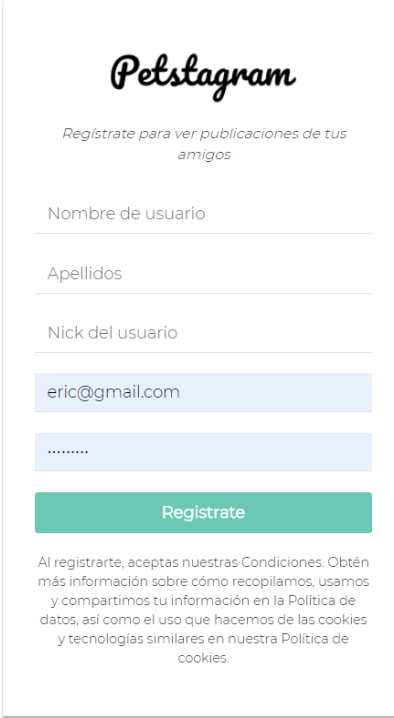
El componente de footer le damos la clase “mt-auto” para que se quede al final de la pagina pegado.

4.2 Creación de usuarios

Para la creación de usuarios vamos a crear un componente llamado register

-ng g c register

Dentro de este componente vamos a crear un formulario que no permita elegir nuestros datos personales



The image shows a registration form for 'Petstagram'. At the top is the logo 'Petstagram' in a script font. Below it is a subtitle 'Regístrate para ver publicaciones de tus amigos'. The form contains five input fields: 'Nombre de usuario', 'Apellidos', 'Nick del usuario', an email field with 'eric@gmail.com' pre-filled, and a password field with '.....'. A green 'Regístrate' button is below the password field. A disclaimer text is below the button, and a link to log in is at the bottom.

Petstagram

Regístrate para ver publicaciones de tus amigos

Nombre de usuario

Apellidos

Nick del usuario

eric@gmail.com

.....

Regístrate

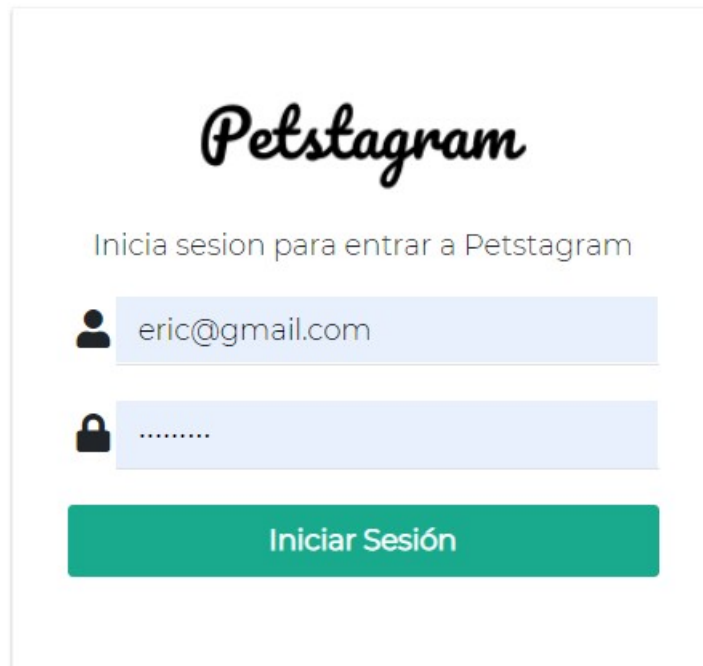
Al registrarte, aceptas nuestras Condiciones. Obtén más información sobre cómo recopilamos, usamos y compartimos tu información en la Política de datos, así como el uso que hacemos de las cookies y tecnologías similares en nuestra Política de cookies.

¿Ya tienes una cuenta creada?[Iniciar Sesión.](#)

Luego de esto nos saldrá un mensaje de verificación si todo va bien o si va mal uno de error y podremos volver a intentarlo

4.3 Login de Usuarios

Crearemos un formulario también para el login en el que usaremos el correo electrónico y la contraseña para acceder a la web.



The image shows a login form for 'Petstagram'. At the top is the 'Petstagram' logo in a cursive font. Below it is the text 'Inicia sesion para entrar a Petstagram'. There are two input fields: the first is for the email address, with a user icon on the left and the text 'eric@gmail.com' inside; the second is for the password, with a lock icon on the left and a series of dots inside. Below these fields is a green button with the text 'Iniciar Sesión'.

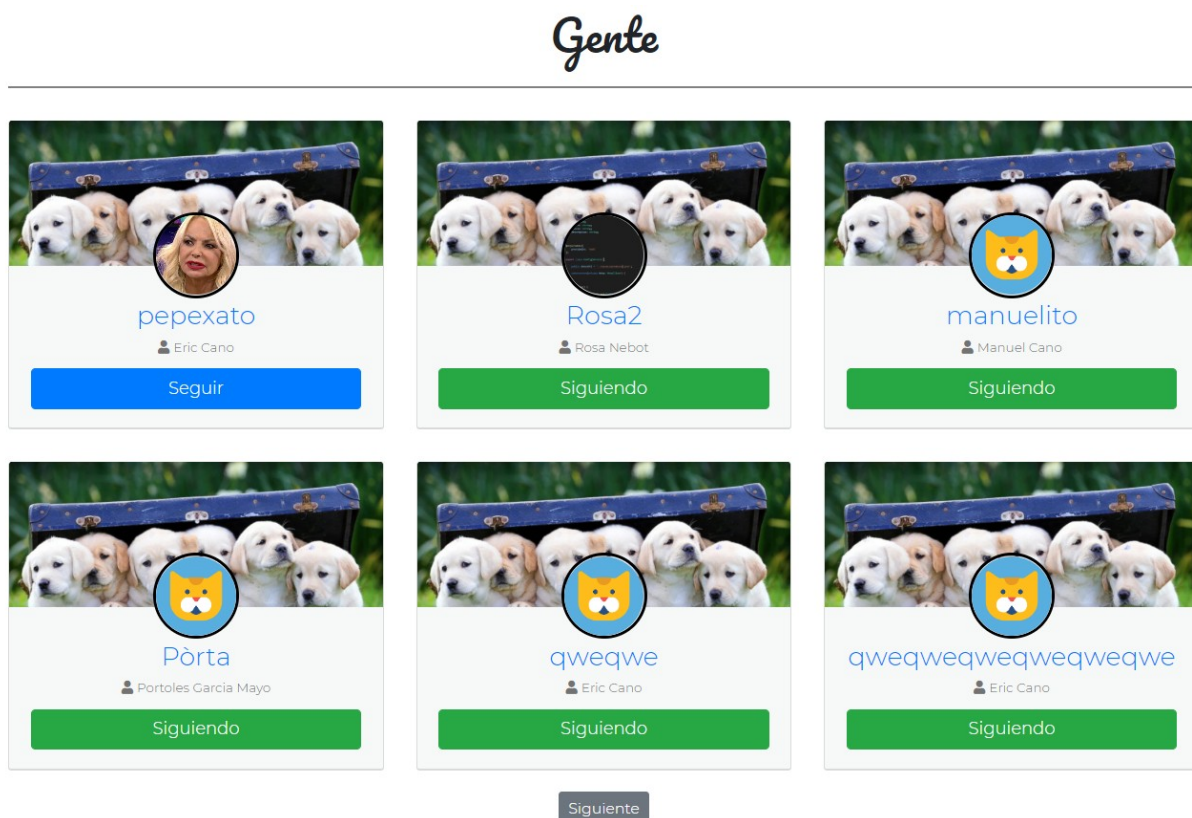
Si el login es correcto nos redirigirá a la pantalla de home y desde ahí ya se nos activaran los demás enlaces de la web para seguir usuarios, ver publicaciones y muchas cosas mas.

4.4 Sección de gente de la plataforma.

Creamos el componente de users

- `ng g c users`

Luego de esto lo maquetamos sabiendo que podemos mostrar 6 usuarios por pagina como le hemos indicado en el backend por lo tanto maquetamos el componente con esto en mente.



Las fotos de perfil de usuario por defecto nada mas creamos un usuario es el gatito de la foto, así como el background de los perritos. La foto de perfil de por ejemplo los dos primeros usuarios se puede cambiar desde el siguiente componente que vamos a crear que sera el de datos personales

Ademas desde aquí podemos ver a que usuarios seguimos así como dejarlos de seguir o seguir a otros.

4.5 Mis datos

Crearemos el componente de edituser.

- *ng g c edituser*

En este componente tenemos un pequeño formulario que nos permitirá editar nuestros datos personales, desde el menú desplegable de perfil podemos hacer clic aquí y editarlo.

Actualizar mis datos



Seleccionar archivo

No se eligió archivo

Actualizar mis datos

4.6 Cerrar Sesión

Para cerrar sesión lo que hacemos es que dentro del componente de navbar tenemos una función que se activa cuando hacemos click en el enlace de cerrar sesión y que nos limpia los datos del localStorage y deja nuestro objeto de identity a null entonces nos redirecciona a la pagina de login

```
logout() {  
  localStorage.clear();  
  this.identity = null;  
}
```

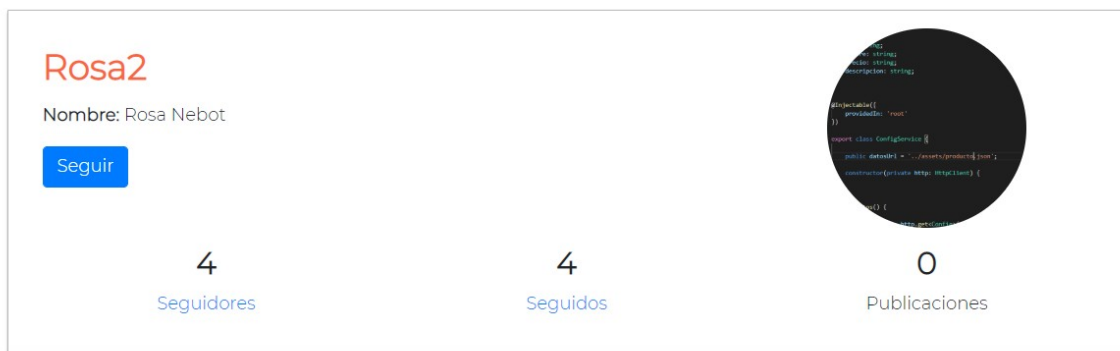

4.7 Perfil

Creamos el componente de perfil.

- `ng g c profile`

Dentro maquetaremos un perfil de usuario en el que se verán datos como las publicaciones que tiene creadas, los usuarios que sigue así como los que te siguen.

Perfil



A la opción de perfil podemos entrar desde ese menú o entrar al de cualquiera ya que tenemos un enlace que nos lleva ahí desde su nickname.

4.8 Crear Publicaciones

Creamos el componente para crear publicaciones.

- *ng g c publications*

Este componente tiene un pequeño formulario en el que podemos poner una descripción y subir una foto (ambos campos obligatorios)



Nueva Publicación

Descripción:

Imagen (Obligada)

No se eligió archivo

En el backend comprobamos que el archivo a subir sea una foto mirando sus extensiones para que nos deje subirla.

4.9 Mensajería entre usuarios

Como el mensajería tendrá unos cuantos componentes relacionados lo que vamos a hacer es una carpeta que se llamará messages y dentro de esta crearemos 4 componentes

El primer componente sera el de main que lo que tendrá es a la izquierda un menú en el que podremos cargar a la derecha el componente de enviar, mensajes recibidos y mensajes enviados

Main

Mensajería Privada

 Enviados

 Recibidos

 Enviar

Enviar

Enviar Mensaje


Para:

Texto:

Enviar Mensaje

Mensajes recibidos

Mensajes Recibidos

 pepexato


" Hombre Rosa que perrito mas mono tienes!!"

hace 32 minutos

Aquí podemos ver los mensajes que hemos recibido por orden cronológico.

Mensajes enviados

Mensajes Enviados

 Rosa2

Enviado a: [pepexato](#)

"Buenas pepexato me gusta una de tus mascostas que has publicado"

hace 3 días

En enviados podemos ver quien es el remitente y entrar desde ahí a su perfil.

5. Vision Comercial

5.1 Matriz DAFO

En mi proyecto el tema comercial creo que se puede resumir en hacer un plan para publicaciones promocionadas por empresas relacionadas al ambito de las mascotas asi como diferentes asociaciones familiares.

Igualmente se podría interpretar de la siguiente forma para poder realizar estas publicaciones promocionadas:

Lo más efectivo para situar un punto de partida es realizar un pequeño análisis DAFO (debilidades, amenazas, fortalezas y oportunidades) que permita dibujar una rápida radiografía del punto de partida en el que se encuentra la compañía. Eso nos servirá para establecer:

-Debilidades: La falta de experiencia, de recursos humanos, de tiempo, de presupuesto para la elaboración de contenidos... pueden ser algunas de las debilidades que presente la compañía y es conveniente tenerlas en cuenta a la hora de planificar la estrategia en redes sociales.

-Amenazas: Por ejemplo, si los principales competidores ya desarrollan una estrategia de contenidos férrea y exitosa en las redes sociales, eso mermará nuestra capacidad para llamar la atención de los usuarios y nos hará más difícil triunfar en las redes sociales.

-Fortalezas: Habría que analizar si se cuenta, por ejemplo, con una comunidad de usuarios afín e identificada con los valores de la marca. Estos grupos de personas, a priori, serán más receptivos a los mensajes que se le puedan transmitir. Esto puede ser una de las fortalezas que presente la compañía en las redes sociales.

-Oportunidades: Si somos conscientes de que los usuarios demandan contenidos específicos sobre un tema que ningún competidor está cubriendo, tenemos una oportunidad importante por delante.

5.2 Marketing Mix.

Al ser una red social de la cual solo podemos poner post como forma de distribución y venta mi idea seria que directamente enlazar a las paginas de comidas de mascotas por ejemplo.

Si la red social creciera mucho también se podria sacar una red de productos propia para la cual podría abrir un departamento que se encargara de estas funciones.

6 Bibliografía

- <https://medium.com/@osxerik/angular-8-bootstrap-4-footer-at-bottom-or-after-content-2596f0e1e1ba>
- <https://nodejs.org/es/docs/guides/>
- <https://expressjs.com/es/guide/routing.html>
- <https://docs.mongodb.com/manual/>
- <https://stackoverflow.com/questions/58559787/cors-angular-8-it-gives-me>
- <https://www.youtube.com/watch?v=1tRLveSyNz8>
- <https://www.npmjs.com/package/ngx-moment>
- <https://www.udemy.com/course/desarrollar-una-red-social-con-javascript-angular-y-nodejs-mongodb/>
- <https://openwebinars.net/academia/portada/mongodb/>
- <https://openwebinars.net/academia/portada/nodejs/>
- <https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html>
- <https://angular.io/docs>