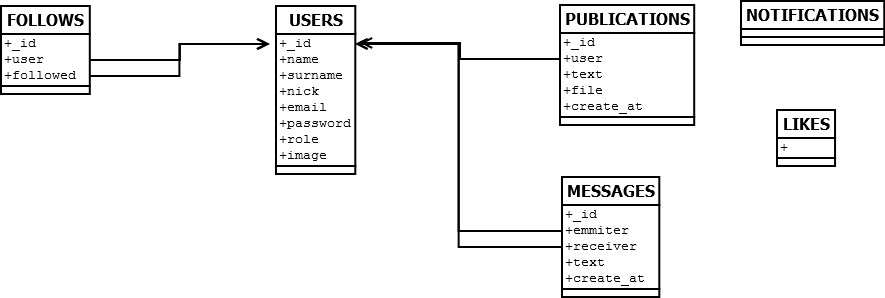
**PROYECTO MEAN 5**

**Schema de base de Datos:**



1-creamos un proyecto en nodejs con **npm init** damos nombre al proyecto y seguimos los pasos normales.

2-creamos un repositorio en git y lo enlazamos con el comando

git remote add origin <https://github.com/jesusdario10/Redsocial.git>

3-empecemos instalando las dependencias que necesitamos para nuestro backend

1. **Npm install express --save** : librería de node que nos permite estableces un servidor que trabaja con protocolos http, rutas, controladores etc.
2. **Npm install bcrypt-nodejs --save** :librería que se encarga del encriptado de contraseñas
3. **Npm install body-parser –save**: Librería que nos permite convertir los json que nos llegan a la api o al backend a un objeto javascript totalmente usable y funcional
4. **Npm install connect-multiparty --save**  : librería que se encarga del proceso de la subida de los ficheros o : archivos es decir fotos, documentos etc.
5. **Npm install jwt-simple --save:**librería que nos permite cifrar y descifrar los token para la autenticación u otros procesos en los que se requieran
6. **Npm install mongoose:** librería que nos permite trabajar de una manera mas simple con mongodb
7. **Npm install moment --save :** librería para trabajar las fechas, horas y todo lo que tenga que ver con tiempo.
8. **Npm install nodemon --save-dev** : librería que refresca el servidor cada que guardamos para no estar parando y arrancando el servidor.

Nuestro package.json debería tener esta forma.

{

"name": "backendred",

"version": "1.0.0",

"description": "backend de una red social para practicar los conocimientos",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "jdmp",

"license": "MIT",

"dependencies": {

"bcrypt-nodejs": "0.0.3",

"body-parser": "^1.18.3",

"connect-multiparty": "^2.2.0",

"express": "^4.16.4",

"jwt-simple": "^0.5.5",

"moment": "^2.22.2",

"mongoose": "^5.3.7"

},

"devDependencies": {

"nodemon": "^1.18.5"

}

}

**Creemos la base de datos:**

Usaremos robo3T

La base se llamara RedS

1-creemos una colleccion llamada users

2-insertemos un documento:

/\* 1 \*/

{

"\_id" : ObjectId("5bd75ca10629c34bdb9a15f9"),

"name" : "admin",

"surname" : "admin",

"nick" : "admin",

"email" : "admin@admin.com",

"password" : "123456",

"image" : null

}

**3-conectemos la api a mongodb:**

1. **Creamos el archivo index.js en la raiz**
2. **Conectar mongodb usando promesas:**

'use strict'

var mongoose = require('mongoose');

mongoose.Promise = global.Promise;

mongoose.connect('mongodb://localhost:27017/RedS', {useMongoClient: true})

.then(()=>{

console.log("conectado a la db RedS");

})

.catch(err=>{

console.log("Error al conectarse a la db");

})

1. **Configurar el comando npm start con nodemon en el package.json:**

"scripts": {

"start":"nodemon index.js",

"test": "echo \"Error: no test specified\" && exit 1"

},

**Creemos el servidor web:**

**1-creamos el fichero app.js:** aq1ui configuraremos todo lo que tiene que ver con express es decir body-parser, rutas etc y luego este fichero lo importaremos en el index.js donde culminaremos la creación final del servidor web.

1. Lo primero que haremos sera cargar express, body parser e instanciar express:

'use strict'

var express = require('express');

var bodyParser = require('body-parser');

var app = express();

1. Cargaremos los middlewares del body parser que sonmetodos que se ejecutan antes de que los controladores se ejecuten. Estos middleares del body parser son necesarios para la ejecución del bodyparser, y luego hacemos que convierta lo que nos llegue en el body a un objeto json, esto ocurrirá cada que hagamos peticiones

*//cargar middlewares*

app.use(bodyParser.urlencoded({extended:false}));

app.use(bodyParse.json());

1. Ahora debemos exportar la configuración: ya que lo podremos importar en otros ficheros en este caso en el index.js

module.exports = app;

1. Carguemos el app.js en el index.js

'use strict'

var mongoose = require('mongoose');

var app = require('./app')

1. Creemos también una variable port donde le indicaremos al puerto donde vamos a trabajar:

var port = 3800;

1. Usemos la variable app dentro del then para arrancar el servidor de modo que el archivo index.js quedaría de la siguiente manera.

'use strict'

var mongoose = require('mongoose');

var app = require('./app');

var port = 3800;

*//Conexion a la base de datos*

mongoose.Promise = global.Promise;

mongoose.connect('mongodb://localhost:27017/RedS', {useMongoClient: true})

.then(()=>{

console.log("conectado a la db RedS");

*//Crear servidor*

app.listen(port, ()=>{

console.log("Servidor NodeJS corriendo in port 3800");

})*//-fin de la creacion del servidor*

})

.catch(err=>{

console.log("Error al conectarse a la db");

})

**Creemos una ruta de prueba en app.js para saber desde el navegador si en realidad nuestro servidor esta corriendo:**

*//cargar rutas*

app.get('/', (req, res)=>{

res.status(200).json({

message:'Accion de pruebas en el servidor de NodeJs'

})

})

Efectivamente en el navegador podemos observar como funciona

<http://localhost:3800/>

{"mensaje": "Acción de pruebas en el servidor de NodeJs"}

**El cliente ResFull**

Postman nos permitirá realizar peticiones http de cualquier tipo y manipularla a nuestro antojo para hacer pruebas ojo los middlewares siempre van primero que las rutas:

El archivo app.js quedaría de esta manera por ahora:

'use strict'

var express = require('express');

var bodyParser = require('body-parser');

var app = express();

*//cargar middlewares*

app.use(bodyParser.urlencoded({ extended: false }))

app.use(bodyParser.json())

*//cargar rutas*

app.post('/', (req, res)=>{

console.log(req.body);

res.status(200).json({

message:'Accion de pruebas en el servidor de NodeJs'

})

})

*//cors y cabeceras*

*//rescribir rutas*

module.exports = app;

**creemos los modelos:**

1-creemos una carpeta llamada model y dentro creemos el modelo de usuario llamado userModel.js.

2-se inicia con ‘use strict’

3-cargamos el modulo de mongoose

var mongoose = require('mongoose');

4-crearemos una variable llamada Schema que nos permitirá definir nuevos Schemas.

var Schema = mongoose.Schema;

5-ahora creemos la variable UserSchema que sera la entidad que reutilizaremos cada que quedamos crear un usuario:

'use strict'

var mongoose = require('mongoose');

var Schema = mongoose.Schema;

var UserSchema = Schema({

name: String,

surname : String,

nick : String,

email: String,

password : String,

role : String,

image : String

})

module.exports = mongoose.model('User', UserSchema);

**6-**creemos el modelo de publication que se llamara publicationModel.js

'use strict'

var mongoose = require('mongoose');

var Schema = mongoose.Schema;

var PublicationSchema = Schema({

text: String,

file: String,

create\_at:String,

user : { type: Schema.ObjectId, ref: 'User'}

})

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

**7-**vamos con el modelo de followers que se llamara followModel.js

'use strict'

var mongoose = require('mongoose');

var Schema = mongoose.Schema;

var FollowSchema = Schema({

user:{type : Schema.ObjectId, ref : 'User'},//es el user que sigue

followed:{type : Schema.ObjectId, ref : 'User'}//es el user seguido

})

module.exports = mongoose.model('Follow', FollowSchema);

8-vamos con el modelo de mensajes que lo llamaremos messageModel.js

'use strict'

var mongoose = require('mongoose');

var Schema = mongoose.Schema;

var MessageSchema = Schema({

emmiter : {type: Schema.ObjectId, ref :'User'}, *//usuario que emite el mensaje*

receiber :{type: Schema.ObjectId, ref :'User'},*//usuario recibe el mensaje*

text:String,

created\_at:String

});

moule.exports = mongoose.model('Message', MessageSchema);

**usemos la arquitectura MVC**

**CREEMOS LOS CONTROLADORES**

1-creemos una nueva carpeta llamada controllers y alli dentro crearemos los controladores.

2-creemos el controlador de usuario al que llamaremos userControllers.js.

3-debemos requerir el modelo de usuarios para usarlo dentro de este.

'use strict'

var UserModel = require('../models/userModel');

4-aqui definiremos los metodos que utilizaran las peticiones y las exportaremos por ejemplo:

'use strict' //ESTE ES EL USER CONTROLLERS

var UserModel = require('../models/userModel');

*//probaremos el home aqui*

function home(req, rs){

console.log(req.body);

res.status(200).send({

message:'Esto es una prueba en el home'

})

}

module.exports = {

home

}

5-ahora para probar estos métodos, debemos crear la carpeta que contendrá las rutas de la aplicación esta carpeta se llamara routes y dentro contendrá las diferentes rutas, de modo que creemos también dentro el fichero que tendrá las rutas del controlador userController.js, dicho fichero se llamara userRoutes.js

1. Aquí deberemos cargar express
2. Cargar el controlador del usuario en este caso
3. Crear una variable en este caso api y cargar el router de express con esto tendre acceso a los metodos get, post y todos los metodos html.
4. Después debemos definir cada una de las rutas
5. Al final solo exportamos el api

'use strict'

var express = require('express');

var userController = require('../controllers/userController');

var api = express.Router();

api.get('/home', userController.home);

module.exports = api;

1. Para que este fichero de rutas funcione tendremos que cargarlo en el app.js

*//cargando ficheros de rutas*

var userRoutes = require('./routes/userRoutes');

1. Y luego creamos el middleware de las rutas para definir una ruta base de modo que si usamos la siguiente ruta podemos obtener la respuesta.

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

{

"message": "Esto es una prueba en el home"

}

**Ahora creemos los métodos de usuario en este caso saveUser que nos permitirá guardar los usuarios en la db.**

Debemos crear una instancia del modelo de usuario para poder guardarlo como se muestra a continuación.

*//Guardar usuario*

function saveUser(req, res){

var user = new UserModel();

}

Ahora para recoger los parámetros de la request también crearemos una variable llamad params que nos ayudara con esto

var params = req.body;

ahora hagamos una condición if que nos permita saber que todos los campos necesarios vienen cargados y luego setiamos el objeto user creado.

if(params.name && params.surname && params.nick && params.email && params.password){

user.name = params.name;

user.surname = params.surname;

user.nick = params.nick;

user.email = params.email;

}

Ahora vamos a encriptar el password, para eso hay que requerir el modulo de bcrypt

var bcrypt = require('bcrypt-nodejs');

luego vayamos nuevamente vamos al método y lo encriptamos usando el método hash de bcryp

el primer parámetro sera la cadena a encriptar que seria el password y existen 3 parametros mas los 2 primeros son para hacer mas encriptación sobre la encriptación y el tercero es una función callback que nos arrojara la encriptación y el error como se muestra a continuación de modo que el user.password lo igualaremos al hash resultante.

if(params.name && params.surname && params.nick && params.email && params.password){

user.name = params.name;

user.surname = params.surname;

user.nick = params.nick;

user.email = params.email;

*//encriptando el password*

bcrypt.hash(params.password, null, null, (err, hash)=>{

user.password = hash;

})

Por ultimo guardaremos el usuario:

bcrypt.hash(params.password, null, null, (err, hash)=>{

user.password = hash;

*//guardando el usuario el la base de datos:*

user.save((err, userGuardado)=>{

if(err) res.status(500).send({message:"error al gaurdar el usuario"});

if(userGuardado){

res.status(200).send({user:userGuardado});

}else{

res.status(404).send({message:"No se ha registrado el usuario"});

}

})

})

Al final el método **saveUser** quedaría de la siguiente manera:

*//Guardar usuario*

function saveUser(req, res){

var user = new UserModel();

var params = req.body;

if(params.name && params.surname && params.nick && params.email && params.password){

user.name = params.name;

user.surname = params.surname;

user.nick = params.nick;

user.email = params.email;

*//encriptando el password*

bcrypt.hash(params.password, null, null, (err, hash)=>{

user.password = hash;

*//guardando el usuario el la base de datos:*

user.save((err, userGuardado)=>{

if(err) res.status(500).send({message:"error al gaurdar el usuario"});

if(userGuardado){

res.status(200).send({user:userGuardado});

}else{

res.status(404).send({message:"No se ha registrado el usuario"});

}

})

})

}else{

res.status(200).send({

message:"Todos los datos son necesrios"

})

}

}

Para probar este metodo deberemos exportarlo de uestro controlador

module.exports = {

home,

saveUser

}

Ahora nos quedaría crear la ruta en las rutas de usuario.

api.post('/register', userController.saveUser);

ahora podemos probar en postman

hagamos un control para no permitir usuarios duplicados

esto lo haremos por el campo email o por el nick: y utilizaremos una clausula de guarda (**termino:es el significado de una texcnica de programacion**) que es llegar a un punto del código y retornar un resultado para que el código llegue y solo llegue hasta allí. de modo que el metodo saveUser quedara de la siguiente forma:

*//==========Guardar usuario=================//*

function saveUser(req, res){

var user = new UserModel();

var params = req.body;

if(params.name && params.surname && params.nick && params.email && params.password){

user.name = params.name;

user.surname = params.surname;

user.nick = params.nick;

user.email = params.email;

user.role ='ROLE\_USER';

user.image = null;

*//Validndo si un existe un usuario antes de guardarlo*

UserModel.find({$or:[*//1.busco con un or es decir si el email es igual o si el nick es igual*

{email: user.email.toLowerCase()}, *//si el email es igual al email que trae la instancia*

{nick: user.nick.toLowerCase()}*// si el nick es igual al nick que trae la instancia*

]}).exec((err, users)=>{

if(err) return res.status(500).send({message:"Error en la peticion de usuarios"});

if(users && users.length >= 1){

*/\*utilizaremos una clausula de guarda que es llegar a un punto del código y retornar un resultado*

*para que el código llegue y solo llegue hasta allí. con esto el bcrypt y el codigo que sigue no*

*se ejecutara, esto se hace para ahorrarnos callback, if anidados y codigo de ejecucion\*/*

return res.status(200).send({message:"el usuario que intenta registrar ya existe"});

}else{

*//encriptando el password(solo si el find anterior en su ultima condicion es decir if(users && users.length >= 1)*

*// es falso llegara a este punto)*

bcrypt.hash(params.password, null, null, (err, hash)=>{

user.password = hash;

*//guardando el usuario el la base de datos:*

user.save((err, userGuardado)=>{

if(err) res.status(500).send({message:"error al gaurdar el usuario"});

if(userGuardado){

res.status(200).send({user:userGuardado});

}else{

res.status(404).send({message:"No se ha registrado el usuario"});

}

});

});

}

});

}else{

res.status(200).send({

message:"Todos los datos son necesrios"

})

}

}

Al probarlo y registrar un usuario en

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

con los datos

"name": "Jesus Dario",

"surname": "Marenco Porto",

"nick": "daro",

"email": "admin@admin.com",

"password": "123",

El resultado sera un usuario guardado en la base de datos de la siguiente forma:

{

"user": {

"\_id": "5bd8832632c4f935880b5604",

"name": "Jesus Dario",

"surname": "Marenco Porto",

"nick": "daro",

"email": "admin@admin.com",

"role": "ROLE\_USER",

"image": null,

"password": "$2a$10$1UlqqXS5.C9.xX6VZqcrd.owK5xfV7l4toQbhuLVzZZBn69ODgIp2",

"\_\_v": 0

}

}

Y al querer guardar otro usuario con el mismo Nick o email este sera el resultado.

{

"message": "el usuario que intenta registrar ya existe"

}

**De modo que actualmente validamos que los email y Nick sean únicos.**

**Creemos el metodo de login**

1-crear la función loginUser

2-recogemos los parámetros que lleguen por post

3-creamos una variable para email y otra para el password

4-hacemos un find para ver si el email que llega coinciden con lo que está en la base de datos

5-si existe la búsqueda hare un bcryp.compare al cual le pasare el password que llega por body y se comparara con el password del usuario que esta encriptado, este proceso lo hace el bcrypt compare de esta forma si esta condición se cumple pasamos e ejecuta el tercer parámetro que es una función callback con el resultado.

6-enviamos el usuario para verificar la función:

7-exportamos la función.

8-creamos la ruta

9-probamos con postman

*//==============================Login de Usuario===============================//*

function loginUser(req, res){

var body = req.body;

var email = body.email;

var password = body.password;

UserModel.findOne({email:email}, (err, user)=>{

if(err) res.status(500).send({message:"error al gaurdar el usuario"});

if(user){

bcrypt.compare(password, user.password, (err, check)=>{

if(check){

*//devolvemos el usuario*

return res.status(200).send({

message:"Usuario Logueado correctamente",

user:user,

check:check

});

}else{

return res.status(404).send({message:"el usuario no pudo ser identificado"});

}

});

}else{

return res.status(504).send({message:"el usuario no pudo ser identificado!!"});

}

});

}

Creando ruta en userRoutes.js

api.post('/login', userController.loginUser);

probando en postman:

1-cuando existe un usuario y su contraseña es correcta

http://localhost:3800/api/login

{

"message": "Usuario Logueado correctamente",

"user": {

"\_id": "5bd8832632c4f935880b5604",

"name": "Jesus Dario",

"surname": "Marenco Porto",

"nick": "daro",

"email": "admin@admin.com",

"role": "ROLE\_USER",

"image": null,

"password": "$2a$10$1UlqqXS5.C9.xX6VZqcrd.owK5xfV7l4toQbhuLVzZZBn69ODgIp2",

"\_\_v": 0

},

"check": true

}

2-cuando la contraseña es incorrecta

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

{

"message": "el usuario no pudo ser identificado"

}

3-cuando el usuario es incorrecto

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

{

"message": "el usuario no pudo ser identificado!!"

}

**Usar jwt(JAON WEB TOKEN PARA EL LOGIN)**

Cuando el metodo de login devuelve los datos es decir el login esta correcto, nos devuelve todo incluido el password y todo el hash(encriptación de bcrypt) de la contraseña, esto es mejor no devolverlo y que se quede en el backend, para eliminar estos datos en el código antes de mandar los datos puedo setiar el password que me trae el user para que envie otra cosa.

bcrypt.compare(password, user.password, (err, check)=>{

if(check){

*//devolvemos el usuario*

user.password = '.i.'

ahora a la hora de devolver los datos del usuario podremos devolver un token en donde iran todos los datos del usuario pero codificados o encriptados dentro de un hash

**para esto deberemos crear un servicio dentro de nodejs**

1-creamos una carpeta en la raiz llamada services, y dentro un fichero llamado jwt.js quesera el servicio de los json webs token

'use strict'

var jwt = require('jwt-simple');

var moment = require('moment');

var secret ='clave\_secreta\_aprendiendo\_mas\_angular'*/\*este es un String secreto que solo nosotros como programadores sabremos, el token es casi que imposible de descifrarlo sin esta clave\*/*

exports.createToken = function(user){

*//variable que contendra los datos que quiero codificar*

var payload = {

sub:user.\_id,

name:user.name,

surname : user.surname,

nick : user.nick,

email : user.email,

rool : user.role,

image : user.image,

iat: moment().unix(), *// fecha de creacion del token*

exp: moment().add(1, 'days').unix*//fecha de expiracion a la fecha actual le añadimos 30 dias*

};

*//el metodo encode lo codifica todo y genera un hash con ayuda de la clave secret*

return jwt.encode(payload, secret);

}

Ahora utilicemos este servicio jwt.js dentro de nuestro controlador de usuarios userController.js

1-var jwt = require('../services/jwt');

2- ahora lo usamos en nuestra function de login

*//==============================Login de Usuario===============================//*

function loginUser(req, res){

var body = req.body;

var email = body.email;

var password = body.password;

UserModel.findOne({email:email}, (err, user)=>{

if(err) res.status(500).send({message:"error al gaurdar el usuario"});

if(user){

bcrypt.compare(password, user.password, (err, check)=>{

if(check){

***//generar y devolver el token que contiene el usuario encriptado con el secret***

**if(body.gettoken){**

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

}else{

*//devolvemos el usuario normal*

user.password = undefined;

return res.status(200).send({

message:"Usuario Logueado correctamente",

user:user,

check:check

});

}

}else{

return res.status(404).send({message:"el usuario no pudo ser identificado"});*//si la contraseña es incorrecta*

}

});

}else{

return res.status(504).send({message:"el usuario no pudo ser identificado!!"});

}

});

}

Entonces por la prueba para generarlo ahora agreguemos en el postman una propiedad mas llamada gettoken con valor true para que se cumpla la condición resaltada en amarillo:

El resultado es el siguiente:

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

{

"token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI1YmQ4ODMyNjMyYzRmOTM1ODgwYjU2MDQiLCJuYW1lIjoiSmVzdXMgRGFyaW8iLCJzdXJuYW1lIjoiTWFyZW5jbyBQb3J0byIsIm5pY2siOiJkYXJvIiwiZW1haWwiOiJhZG1pbkBhZG1pbi5jb20iLCJyb29sIjoiUk9MRV9VU0VSIiwiaW1hZ2UiOm51bGwsImlhdCI6MTU0MDkyOTQ5OX0.bg9hyjA\_TFw9V3Ga-KYO7yTFjHT5hIc0ijVclPOxb7U"

}

**Creemos un middleware de autenticación.**

Este middleware comprobara el token que se envía para ver si es correcto y si es correcto puede entrar a la sesion

1-creamos una carpeta mueva que se llama middlewares y dentro un fichero llamado autenticated.js

'use strict'

var jwt = require('jwt-simple');

var moment = require('moment');

var secret ='clave\_secreta\_aprendiendo\_mas\_angular'*/\*este es un String secreto que solo nosotros como*

*programadores sabremos, el token es casi que imposible de descifrarlo sin esta clave\*/*

exports.ensureAuth = function(req, res, next){

if(!req.headers.authorization){

return res.status(403).send({message:"La peticion no tiene la cabecera de autenticacion"});

}

*//aqui cargamos el token*

var token = req.headers.authorization.replace(/['"]+/g,'');*//remplaza cualquier comilla doble o simple por vacio*

try{

*//decodificamos el token*

var payload = jwt.decode(token, secret);*/\* el payload es sencible a errores y exepciones que*

*causa que la aplicacion pare por eso lo meto en un try cacth\*/*

*//si la fecha de expiracion es menor que la actual diga que expiro*

if(payload.exp <=moment().unix()){

return res.status(401).send({message:"Token expiro"});

}

}catch(ex){

return res.status(404).send({message:"el token no es valido"});

}

*//adjuntamos el payload a la req para tener siempre a la mano dentro de los controladores el objeto del usuario logeado*

req.user = payload;

*//ahora hacemos uso del metodo next para pasar a la siguiente accion*

next();

}

Estos middlewares los usaremos en las rutas de modo que deberemos cargarlos allí vamos a userRoutes.js y:

Creemos una variable md\_auth para cargar el middleware de autenticación:

var md\_auth = require('../middlewares/autenticated');

ahora usémoslo en la ruta del home

api.get('/home', md\_auth.ensureAuth, userController.home);

problemos en postman:

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

{

"message": "La peticion no tiene la cabecera de autenticacion"

}

Esto es porque aun no he añadido el token entonces en postman voy a Headers le pongo un componente llamado authorization que fue el que definimos que vendría en el req del middleware autenticated.js

Recordando:

exports.ensureAuth = function(req, res, next){

if(!req.headers.authorization){

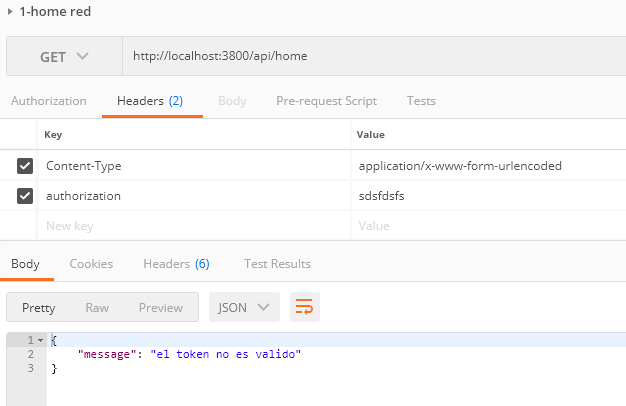
return res.status(403).send({message:"La peticion no tiene la cabecera de autenticacion"});

}

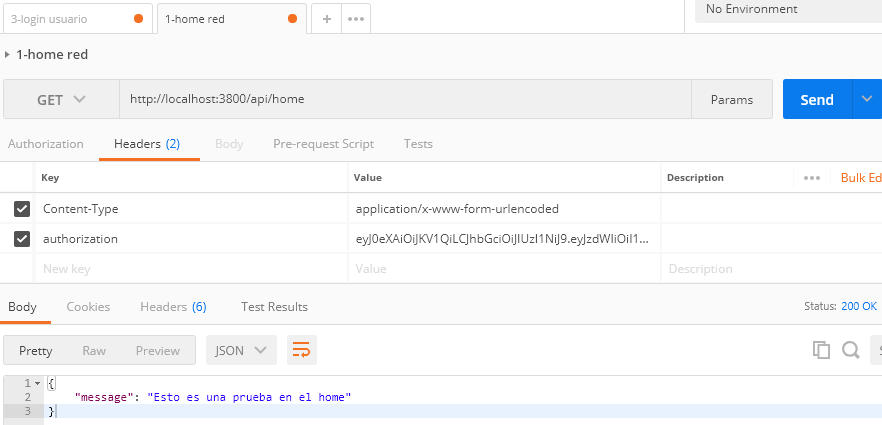
*//aqui cargamos el token*

var token = req.headers.authorization.replace(/['"]+/g,'');*//remplaza cualquier comilla doble o simple por vacio*

y le vamos a pasar un token no valido es decir por ejemplo:

 la respuesta sera token no valido

Pero si le colocamos el token generado y si no ha expirado la respuesta sera:



Bienvenido al home de esta manera estamos validando la ejecución de los controladores por medio de las rutas para que se ejecuten solo aquellos que tienen un token valido.

Creemos otros métodos para los usuarios:

**Listar los datos de un usuario:**

*//==============================LISTAR UN USUARIO=========================================//*

function getUser(req, res){

*//el id del usuario me llegara por la URL*

var userId = req.params.id;

UserModel.findById(userId, (err, user)=>{

if(err) return res.status(500).send({message:"Error en la peticion"});

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

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

})

}

Exportamos el metodo

module.exports = {

home,

saveUser,

loginUser,

getUser,

}

Ahora creemos la ruta usando el middleware de autenticación para comprobar si el usuario que hace la consulta esta logueado

*//OBTENER DATOS DE UN USUARIO*

api.get('/user/:id', md\_auth.ensureAuth, userController.getUser);

probemos tener en cuenta que como usams el middleware hay que enviar el token en el headers del postman

<http://localhost:3800/api/user/5bd8cd8f55244c3a244d5fe5>

{

"user": {

"\_id": "5bd8cd8f55244c3a244d5fe5",

"name": "Cindy ",

"surname": "Sanchez Quintana",

"nick": "nena",

"email": "cindyyu\_28@gmail.com",

"role": "ROLE\_USER",

"image": null,

"password": "$2a$10$ae2Z1QlafXIff6GqCeJD3u448i0gZ4/7ByftwybdNQMmC52TrIfMi",

"\_\_v": 0

}

}

Efectivamente si el usuario existe lo devuelve y si no existe envía lo siguiente:

{

"message": "Error en la peticion"

}

**Ahora creemos un metodo para listar todos los usuarios de manera paginada para paginar instalaremos el pakcage mongoose-pagination:**

Npm install mongoose-pagination

Ya cuando se instala en el package.json

"mongoose-pagination": "^1.0.0"

Ahora creemos el metodo

*//======================LISTAR TODOS LOS USUARIOS PAGINADOS===============================//*

function getUsers(req, res){

*//aqui recogeremos el id del usuario que esta logueado que se ha decodificado del token*

var identity\_user\_id = req.user.sub; *// esta en la propiedad sub porque fue la propiedad que definimos en el jwt*

*//page es el numero de pginas que utilizaremos para mostrar los datos*

var page = 1;

if(req.params.page){

page = req.params.page;

}

*//cantidad de usuarios que se mostraran por pagina*

var itemsPerPage = 5;

UserModel.find()

.sort('\_id')

.paginate(page, itemsPerPage, (err, users, total)=>{

if(err) return res.status(500).send({message:"Error en la peticion"});

if(!users) return res.status(404).send({message:"no existen usuarios"});

return res.status(200).send({

users,

total,

pages : Math.ceil(total/itemsPerPage)

});

});

}

Exportamos:

module.exports = {

home,

saveUser,

loginUser,

getUser,

getUsers

}

Creamos la ruta:

*//OBTENER TODOS LOS USUARIOS*

api.get('/users/:page?', md\_auth.ensureAuth, userController.getUsers);

probando en postman

<http://localhost:3800/api/users/1>

{

"users": [

{

"\_id": "5bd8832632c4f935880b5604",

"name": "Jesus Dario",

"surname": "Marenco Porto",

"nick": "daro",

"email": "admin@admin.com",

"role": "ROLE\_USER",

"image": null,

"password": "$2a$10$1UlqqXS5.C9.xX6VZqcrd.owK5xfV7l4toQbhuLVzZZBn69ODgIp2",

"\_\_v": 0

},

{

"\_id": "5bd8cd8f55244c3a244d5fe5",

"name": "Cindy ",

"surname": "Sanchez Quintana",

"nick": "nena",

"email": "cindyyu\_28@gmail.com",

"role": "ROLE\_USER",

"image": null,

"password": "$2a$10$ae2Z1QlafXIff6GqCeJD3u448i0gZ4/7ByftwybdNQMmC52TrIfMi",

"\_\_v": 0

}

],

"total": 2,

"pages": 1

}

**Actualizar Datos de un usuario**:

1-creemos el metodo.

2-importante eliminar la propiedad password ya que esto se hara en otro metodo es una recomendación.

*//==============Edicion de Datos de usuario=====================//*

function updateUser(req, res){

*//regoge desde la url el id del usuario que vamos a actualizar*

var userId = req.params.id;

var body = req.body;

*//borrar la propiedad password usando el metodo delete*

delete body.password;

*//ahora validemos que solo el mismo usuario pueda actualizar sus datos*

if(userId != req.user.sub){

return res.status(500),send({message:"no tienes permiso para actualiar los dtos del usuario"});

}

*/\*usamos el findByIdAndUpdate el segundo parametro son los datos a actualizar*

*el tercer parametro es opcional para que me devuelva los datos del usuario actualizado\*/*

UserModel.findByIdAndUpdate(userId, body, {new:true}, (err, userUpdate)=>{

if(err) return res.status(500).send({message:"Error en la peticion"});

if(!userUpdate) return res.status(404).send({message:"no se pudo actualizar el usuario"});

return res.status(200).send({

message:"usuario Actualizado",

userUpdate

});

})

}

Exportamos el metodo:

module.exports = {

home,

saveUser,

loginUser,

getUser,

getUsers,

updateUser

}

Creamos la ruta:

api.put('/updateuser/:id', md\_auth.ensureAuth, userController.updateUser );

probamos:

hay que poner el token en la cabecera:

<http://localhost:3800/api/updateuser/5bd8cd8f55244c3a244d5fe5>

intentemos con un usuario que no es el que vamos a actualizar:

{

"message": "no tienes permiso para actualiar los dtos del usuario"

}

Intentemos con el token del usuario que vamos a actualizar

{

"message": "usuario Actualizado",

"userUpdate": {

"\_id": "5bd8cd8f55244c3a244d5fe5",

"name": "cindy",

"surname": "sanchez",

"nick": "nena",

"email": "cindy@gmail.com",

"role": "ROLE\_USER",

"image": null,

"password": "$2a$10$ae2Z1QlafXIff6GqCeJD3u448i0gZ4/7ByftwybdNQMmC52TrIfMi",

"\_\_v": 0

}

}

**Método para subir avatar del usuario:**

Hacemo lo mismo:

*//===============================SUBIR IMAGEN DE AVATAR DE USUARIO===========================//*

function uploadImage(req, res){

*//regoge desde la url el id del usuario que vamos a actualizar*

var userId = req.params.id;

*//ahora validemos que solo el mismo usuario pueda actualizar sus datos*

if(userId != req.user.sub){

return res.status(500).send({message:"no tienes permiso para actualiar los datos del usuario"});

}

*//si existe el componente files dentro del req quiere decir que hay un archivo cargando*

if(req.files.image != undefined){

*//el campo que vamos a pasar por post es el campo image*

var file\_path = req.files.image.path;

console.log(file\_path);

*//en la variable file\_split convertimos tdo el path en un array con el metodo .split*

var file\_split = file\_path.split('\\');

*//ahora tomamos la ultima posicionq ue es donde se guardo el nombre de la imagen que ira a uploads/users*

var file\_name = file\_split[2];

*//ahora para validar que sea una imagen debo validar su extension*

var ext\_split = file\_name.split('\.')

var file\_ext = ext\_split[1];

*//si el archivo contiene una extension de tipo de imagen entonces actualiza la imagen del usuario*

if(file\_ext == "png" || file\_ext == "jpg" || file\_ext == "jpeg" || file\_ext == "gif"){

*//actualicemos la imagen del usuario*

UserModel.findById(userId, (err, userImage)=>{

if(err){

return res.status(400).json({message:"error en la peticion."});

}

var pathViejo = "./uploads/users/"+userImage.image;

*// ====en caso de que ya exista una imagen la borramos ===== //*

if(fs.existsSync(pathViejo)){

fs.unlink(pathViejo)

}

*//guardamos la imagen nueva*

userImage.image = file\_name;

userImage.save((err, user)=>{

if(err) return res.status(500).send({message:"Error en la peticion"});

user.password = undefined;

return res.status(200).json({user});

})

})

}else{*//sino enviar un mensaje que diga no es un tip de archivo valido*

removeFieUploads(res, file\_path);

}

}else{

return res.status(400).json({message:"No files were uploaded."});

}

}

function removeFieUploads(res, file\_path, message){

fs.unlink(file\_path, (err)=>{*//elimina el archivo subido de la ruta*

return res.status(200).json({message:"Tipo de archivo no valido."});

});

}

Creamos el middleware de subida que es connect multiparty

Requemimos el middleware de subida en la ruta creamos la ruta y agregamos este middleware de súbita a la ruta.

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

*//creamos el middleware multiparty*

var md\_upload = multipart({uploadDir : './uploads/users'});

ahora devolvamos la imagen al usuario creando el metodo getImageFile:

*//==================================Devolver Imagen Al usuario============================//*

function getImageFile(req, res){

var image\_file = req.params.imageFile;

var path\_file = './uploads/users/'+image\_file;

fs.exists(path\_file, (exists)=>{

if(exists){

return res.sendFile(path.resolve(path\_file))

}else{

return res.status(200).json({message:"No existe la imagen"});

}

})

}

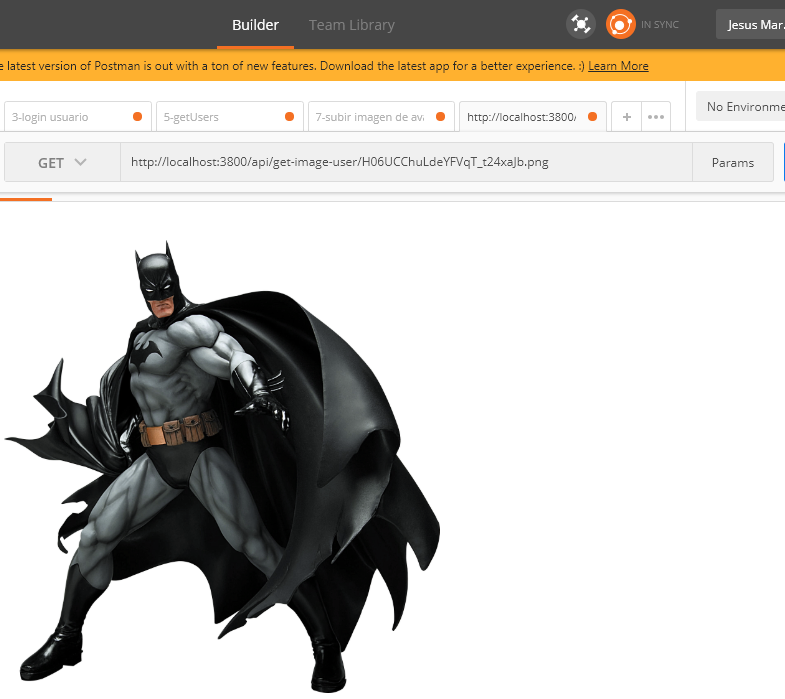
Creo la ruta

*//OBTENER LA IMAGEN DE AVATAR DEL USUARIO*

api.get('/get-image-user/:imageFile', userController.getImageFile)

ejecuto en postman

<http://localhost:3800/api/get-image-user/H06UCChuLdeYFVqT_t24xaJb.png>



Creemos el controlador encargado del sistema de seguimiento: es decir dar la posibilidad de seguirse y dejar de seguirse entre ellos(también mostrar un listado de los usuarios que nos siguen y a los que seguimos):

1-creemos un nuevo controlador llamado followController.js

2-carguemos mongoose-paginate;

3-carguemos el modelo de usuario;

4-cargamos el modelo de Folow;

5-creemos un metodo de prueba

6-exportemoslo

'use strict'

*//var path = require('path');*

*//var fs = require('fs');*

var UserModel = require('../models/userModel');

var FollowModel = require('../models/followModel');

function prueba(req, res){

return res.status(200).send({ok:"hola mundo desde flollows"});

}

module.exports = {

prueba

}

7-creemos un fichero de rutas para el controlador followController.js que se llame followRoutes.js

8- requerimos express, el controlador de followers, el routes de express almacenado en la variable api, el middleware de autenticación y la ruta de prueba para probar.

9-exportamos todas las rutas.

'use strict'

var express = require('express');

var followController = require('../controllers/followController');

var api = express.Router();

var md\_auth = require('..//middlewares/autenticated');

*//===============RUTAS GET=======================================//*

api.get('/pruebas', md\_auth.ensureAuth, followController.prueba);

module.exports = api;

ahora debemos cargar esta ruta e el fichero app.js que es el fichero que lleva toda la configuración de express.

var followRoutes = require('./routes/followRoutes');

app.use('/api', followRoutes);

hagamos la prueba

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

{

"ok": "hola mundo desde flollows"

}

**Creemos un metodo para permitir a un usuario seguir a otro: este metodo se llamara** **saveFollow**

*//========================SEGUIR A UN USUARIO====================================//*

function saveFollow(req, res){

var body = req.body;

var follow = new FollowModel();

follow.user = req.user.sub; *//este es el usuario que esta logueado y que mas adelante seguira a otro*

follow.followed = body.followed;*// este es el usuario que sera seguido*

follow.save((error, followStored)=>{*//Stored el ingles es almacenamiento(user seguido almacenado)*

if(error) return res.status(500).send({message:"error al seguir"});

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

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

})

}

Exportamos el metodo:

module.exports = {

prueba,

saveFollow

}

Creamos la ruta:

*//===============RUTAS POST======================================//*

api.post('/save-follow', md\_auth.ensureAuth, followController.saveFollow);

PROBEMOS

<http://localhost:3800/api/save-follow>

{

"\_id" : ObjectId("5bd9f7c3bb1ec61f680dcf61"),

"user" : ObjectId("5bd8cd8f55244c3a244d5fe5"),

"followed" : ObjectId("5bd8832632c4f935880b5604"),

"\_\_v" : 0

}

**Cremos el metodo dejar para crear de seguir a un usuario que se llamara** **deleteFollow**

*//=========================DEJAR DE SEGUIR A UN USUARIO============================//*

function deleteFollow(req, res){

var userId = req.user.sub;*//yo*

var followId = req.params.id;*//usuario que dejare de seguir*

FollowModel.find({'user':userId, 'followed':followId}).remove(error=>{

if(error) return res.status(500).send({message:"error al dejar de seguir"});

return res.status(200).send({message:'el follow se ha eliminado'});

});

}

module.exports = {

prueba,

saveFollow,

deleteFollow

}

Creemos la ruta

*//===============RUTAS DELETE====================================//*

api.delete('/delete-follow/:id', md\_auth.ensureAuth, followController.deleteFollow);

Probemos

<http://localhost:3800/api/delete-follow/5bd8832632c4f935880b5604>

{

"message": "el follow se ha eliminado"

}

**Cremos el metodo para listar a los usuarios que seguimos llamado** : **getFollowindUser**:

*//===========================LISTAR A LOS USUARIOS QUE SIGO==========================//*

function getFollowindUser(req, res){

var userId = req.user.sub;*//recojo el usuario logueado*

var page = 1;*// numero de paginas por defecto*

if(req.params.page){

page = req.params.page;

}

var itemsPerPage = 4;*//numero de usuarios por pgina*

FollowModel.find({'user':userId})

.populate({path:'followed', select:'name'}).paginate(page, itemsPerPage, (error, follows, total)=>{

if(error) return res.status(500).send({message:"error al traer los datos"});

if(!follows) return res.status(404).send({message:"no sigues a nadie"});

return res.status(200).send({

total,

pages: Math.ceil(total/itemsPerPage),

follows

});

})

}

Exportamos

module.exports = {

prueba,

saveFollow,

deleteFollow,

getFollowindUser

}

Creamos la ruta

*/\*-----------listar usuarios seguidos------\*/*

api.get('/following/:id?/:page?', md\_auth.ensureAuth, followController.getFollowindUser);

probamos

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

{

"total": 2,

"pages": 1,

"follows": [

{

"\_id": "5bda09c680d9dc1cd0876879",

"user": "5bda098280d9dc1cd0876878",

"followed": {

"\_id": "5bd8cd8f55244c3a244d5fe5",

"name": "cindy"

},

"\_\_v": 0

},

{

"\_id": "5bda09d080d9dc1cd087687a",

"user": "5bda098280d9dc1cd0876878",

"followed": {

"\_id": "5bd8832632c4f935880b5604",

"name": "Jesus Dario"

},

"\_\_v": 0

}

]

}

**Ahora creemos un metodo que nos muestre quien nos siguie este metodo se llamara:** **getFollowedUsers**

*//==============================LISTAR LOS USUARIOS QUE NOS SIGUEN======================//*

function getFollowedUsers(req, res){

var userId = req.user.sub;

console.log(userId);

FollowModel.find({"followed": userId})

.populate({path:'user', select:'name'})

.exec((error, followeds)=>{

if(error) return res.status(500).send({message:"error al traer los datos"});

if(!followeds) return res.status(404).send({message:"no te sigue nadie"});

return res.status(200).send({followeds});

});

}

module.exports = {

prueba,

saveFollow,

deleteFollow,

getFollowindUsers,

getFollowedUsers

}

*/\*-----------listar usuarios que me siguen------\*/*

api.get('/followed/:id?/:page?', md\_auth.ensureAuth, followController.getFollowedUsers);

probando

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

{

"followeds": [

{

"\_id": "5bda0fe511910637b4a7a7e3",

"user": {

"\_id": "5bd8832632c4f935880b5604",

"name": "Jesus Dario"

},

"followed": "5bda098280d9dc1cd0876878",

"\_\_v": 0

}

]

}

**Ahora modifiquemos los métodos getUser y getUsers del controlador UsersControllers para comprobar y esos usuarios nos siguen o ya los seguimos y tener la posibilidad de seguirlos desde el frontend:**

**Iniciemos con el getUser:**

1-requerir el modelo de FollowModel

Entonces queremos saber si seguimos a un usuario

*//==============================LISTAR UN USUARIO=========================================//*

function getUser(req, res){

*//el id del usuario me llegara por la URL*

var userId = req.params.id;

UserModel.findById(userId, (err, user)=>{

if(err) return res.status(500).send({message:"Error en la peticion"});

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

FollowModel.findOne({'user':req.user.sub, "followed":userId})

.populate({path:'user', select:'name'})

.populate({path:'followed', select:'name'})

.exec((error, follow)=>{

if(err) return res.status(500).send({message:"Error en la peticion"});

if(!follow) return res.status(404).send({message:"no lo estas siguiendo"});

user.password = undefined;

user.role = undefined;

return res.status(200).send({user, follow})

})

})

}

Probemos:

<http://localhost:3800/api/user/5bd8cd8f55244c3a244d5fe5>

{

"user": {

"\_id": "5bd8cd8f55244c3a244d5fe5",

"name": "cindy",

"surname": "sanchez",

"nick": "nena",

"email": "cindy@gmail.com",

"image": "H06UCChuLdeYFVqT\_t24xaJb.png",

"\_\_v": 0

},

"follow": {

"\_id": "5bda086a7a7f244f98f05584",

"user": {

"\_id": "5bd8832632c4f935880b5604",

"name": "Jesus Dario"

},

"followed": {

"\_id": "5bd8cd8f55244c3a244d5fe5",

"name": "cindy"

},

"\_\_v": 0

}

}

**Usemos Async-Away**

Hagamos en el mismo método el querer saber si ese usuario al que yo sigo me sigue a mi

1-definimos una función asíncrona llamada followThisUser que me devolverá el usuario al que sigo y si el me sigue a mi, estos datos me son devueltos en forma de promesa por ende cuando llame a esta función podre utilizar el metodo then.

*//=============FUNCION ASINCRONA PARA SABER SI EL USUARIO AL QUE SIGO ME SIGUE A MI=======//*

async function followThisUser(identity\_user\_id, user\_id){

*/\*cuando se ejecute algo esperara a que consiga el resultado para continuar*

*es decir se convertira en un proceso sincrono\*/*

*//con following se si estoy siguiendo a alguien*

var following = await FollowModel.findOne({'user':identity\_user\_id, "followed":user\_id}, (error, follow)=>{

if(error) return handleError(error);

return follow;

});

*//con followed se si ese alguien al que estoy siguiendo me sigue*

var followed = await FollowModel.findOne({'user':user\_id, "followed":identity\_user\_id}, (error, follow)=>{

if(error) return handleError(error);

return followed;

});

return {

following,

followed

}

}

2-utilizando la función anterior en el metodo getUser

*//============LISTAR UN USUARIO Y SABER SI ME SIGUE O YO LO SIGO=======//*

function getUser(req, res){

*//el id del usuario me llegara por la URL*

var userId = req.params.id;

UserModel.findById(userId, (err, user)=>{

if(err) return res.status(500).send({message:"Error en la peticion"});

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

user.password = undefined;

user.role = undefined;

*//usando el metodo async-away creado*

followThisUser(req.user.sub, userId).then((value)=>{

return res.status(200).send({user, value})

});

});

}

Probemos: después de una mejora

<http://localhost:3800/api/user/5bd8cd8f55244c3a244d5fe5>

{

"user": {

"\_id": "5bd8cd8f55244c3a244d5fe5",

"name": "cindy",

"surname": "sanchez",

"nick": "nena",

"email": "cindy@gmail.com",

"image": "H06UCChuLdeYFVqT\_t24xaJb.png",

"\_\_v": 0

},

"following": {

"\_id": "5bda086a7a7f244f98f05584",

"user": "5bd8832632c4f935880b5604",

"followed": "5bd8cd8f55244c3a244d5fe5",

"\_\_v": 0

},

"followed": {

"\_id": "5bda07d17a7f244f98f0557a",

"user": "5bd8cd8f55244c3a244d5fe5",

"followed": "5bd8832632c4f935880b5604",

"\_\_v": 0

}

}

**Ahora hagamos lo mismo pero con todos los usuarios es decir con el getUsers:**

**Es decir saber que usuarios nos siguen a cuales seguimos hagamoslo asíncronamente:**

1-creemos una función asíncrona que se llame followUserIds:

*//=====FUNCION ASINCRONA PARA SABER A QUIENES SEGUIMOS Y QUIENES NOS SIGUEN===========//*

async function followUsers(user\_id){*//los campos con 0 no saldra*

try{

var followings = await FollowModel.find({"user": user\_id}).select({":id":0, '\_v':0, 'user':0}).exec()

.then((follows)=>{

var follows\_clean = [];

follows.forEach((follow)=>{

follows\_clean.push(follow.followed);

})

return follows\_clean;

}).catch((err)=>{

return handleerror(err);

})

var followeds = await FollowModel.find({"followed": user\_id}).select({":id":0, '\_v':0, 'followed':0}).exec()

.then((users)=>{

var user\_clean = [];

users.forEach((user)=>{

user\_clean.push(user.user);

})

return user\_clean;

}).catch((err)=>{

return handleerror(err);

})

}catch(e){

console.log(e);

}

return {

followings:followings,

followeds:followeds

}

}

Asi queda la función de getUsers

*//======================LISTAR TODOS LOS USUARIOS PAGINADOS===============================//*

function getUsers(req, res){

*//aqui recogeremos el id del usuario que esta logueado que se ha decodificado del token*

*//var identity\_user\_id = req.user.sub; // esta en la propiedad sub porque fue la propiedad que definimos en el jwt*

*//page es el numero de pginas que utilizaremos para mostrar los datos*

var page = 1;

*//console.log(req.user);*

if(req.params.page){

page = req.params.page;

}

*//cantidad de usuarios que se mostraran por pagina*

var itemsPerPage = 5;

var identity\_user\_id = req.user.sub;

UserModel.find()

.sort('\_id')

.paginate(page, itemsPerPage, (err, users, total)=>{

if(err) return res.status(500).send({message:"Error en la peticion"});

if(!users) return res.status(404).send({message:"no existen usuarios"});

followUsers(identity\_user\_id).then((value)=>{

return res.status(200).send({

users,

user\_followings : value.followings,

user\_follow\_me : value.followeds,

total,

pages : Math.ceil(total/itemsPerPage)

});

});

});

}

Este es el resultado

<http://localhost:3800/api/users/1>

{

"users": [

{

"\_id": "5bd8832632c4f935880b5604",

"name": "Jesus Dario",

"surname": "Marenco Porto",

"nick": "daro",

"email": "admin@admin.com",

"role": "ROLE\_USER",

"image": null,

"password": "$2a$10$1UlqqXS5.C9.xX6VZqcrd.owK5xfV7l4toQbhuLVzZZBn69ODgIp2",

"\_\_v": 0

},

{

"\_id": "5bd8cd8f55244c3a244d5fe5",

"name": "cindy",

"surname": "sanchez",

"nick": "nena",

"email": "cindy@gmail.com",

"role": "ROLE\_USER",

"image": "H06UCChuLdeYFVqT\_t24xaJb.png",

"password": "$2a$10$ae2Z1QlafXIff6GqCeJD3u448i0gZ4/7ByftwybdNQMmC52TrIfMi",

"\_\_v": 0

},

{

"\_id": "5bda098280d9dc1cd0876878",

"name": "Lulu",

"surname": "Lulu",

"nick": "lulu",

"email": "lulu@gmail.com",

"role": "ROLE\_USER",

"image": null,

"password": "$2a$10$oRR.gfTMFNcfoKDBtxJn8OCRFR7dLz0py2i45QfpdJMu..Do8I7a2",

"\_\_v": 0

},

{

"\_id": "5bdb57d255bcf4111845cc52",

"name": " ",

"surname": "tito",

"nick": "tito",

"email": "tito@gmail.com",

"role": "ROLE\_USER",

"image": null,

"password": "$2a$10$ZodNEl4jxm1sZ7SPLshTyuKZlDS5I8QmjrjEwbgmjFWc9.vyPJ4Fi",

"\_\_v": 0

}

],

**"user\_followings": [**

**"5bd8cd8f55244c3a244d5fe5",**

**"5bda098280d9dc1cd0876878"**

**],**

**"user\_follow\_me": [**

**"5bd8cd8f55244c3a244d5fe5",**

**"5bda098280d9dc1cd0876878",**

**"5bdb57d255bcf4111845cc52"**

**],**

"total": 4,

"pages": 1

}

Ahora creemos un metodo en nuestra api que se encargue de mostrar los contadores de los usuarios que seguimos y los que nos siguen se llama **getCounters**;

Primero creemos una función asíncrona que me regrese estos valores:

*//Funcion que me cuenta cuantos seguidores tengo y a cuantos sigo==========*

async function getCountFollow(user\_id){

var following = await FollowModel.count({'user': user\_id}, (error, count)=>{

if(error) return handleError(err);

return count;

})

var followed = await FollowModel.count({"followed":user\_id}, (error, count)=>{

if(error) return handleError(err);

return count;

})

return {

followin: following,

followed : followed

}

}

Función getCouter

*//====================CONTADOR DE USUARIOS QUE NOS SIGUEN Y LOS QUE SEGUIMOS================//*

function getCounters(req, res){

var user\_id = req.user.sub;

if(req.params.id){

var user\_id = req.params.id;

}

getCountFollow(user\_id).then((value)=>{

return res.status(200).send({value});

})

}

Exportemos

module.exports = {

home,

saveUser,

loginUser,

getUser,

getUsers,

updateUser,

uploadImage,

getImageFile,

getCounters

}

Creemos la ruta:

*//OBTENER LOS CONTADORES DE SEGUIDOS Y SEGUIDORES DE UN USUARIO*

api.get('/counters/:id?', md\_auth.ensureAuth, userController.getCounters);

resultado:

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

{

"value": {

"followin": 2,

"followed": 3

}

}

**Creando el controlador de publicaciones:**

2-creemos el controlador publicaciones

1-importemos lo que vamos a necesitar

2-creemos una función de prueba

3-exportemos la función

'use strict'

var path = require('path');

var fs = require('fs');

var moment = require('moment');

var mongoosePaginate = require('mongoose-pagination');

var PublicationModel = require('../models/publicationModel');

var UsuarioModel = require('../models/userModel');

var FollowModeol = require('../models/followModel');

*//========pruba==========//*

function probando(req, res){

res.status(200).send({ok:"hola desde publication"});

}

module.exports = {

probando

}

4-creemos el fichero publicationsRoutes.js para las rutas

5-requiramos express y el controlador de publicaciones

6-Activemos en la variable api el routeo de express con express.Router().

7-requiramos el middleware de autenticación.

8-requiramos el módulo conect-multiparty para la administración de archivos

9- creemos la variable md\_upload donde le pasemos al multiparty la ruta donde guardaremos los archivos.

10-definimos la ruta de prueba

11- exportamos api

'use strict'

var express = require('express');

var PublicationController = require('../controllers/publicationController');

var api = express.Routes();

var md\_auth = require('../middlewares/autenticated');

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

var md\_upload = multipart({uploadDir: './uploads/publications'});

api.get('/probando', md\_auth.ensureAuth, PublicationController.probando);

module.exports = api;

12-ahora vamos al app.js y carguemos el nuevo fichero de rutas

*//cargando ficheros de rutas*

var userRoutes = require('./routes/userRoutes');

var followRoutes = require('./routes/followRoutes');

var publicationRoutes = require('./routes/publicationRoutes');

*//middleware para definir la ruta base*

app.use('/api', userRoutes);

app.use('/api', followRoutes);

app.use('/api', publicationRoutes);

probemos en el postman:

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

{

"ok": "hola desde publication"

}

**Creemos el metodo de guardar nuevas publicaciones:**

*//========================SAVE PUBLICATION==============================//*

function savePublication(req, res){

var body = req.body;

if(!body.text) return res.status(200).send({message:"debes enviar un texto"})

var publicationModel = new PublicationModel;

publicationModel.text = body.text;

publicationModel.file = null;

publicationModel.user = req.user.sub;*// el usuario dueño de la publicacion es decir el usuario logueado*

publicationModel.create\_at = moment().unix();

publicationModel.save((error, publicationStored)=>{

if(error) return res.status(500).send({message:"error al guardar la publicacion"});

if(!publicationStored) return res.status(404).send({message:"la publicacion no fue guardada"});

return res.status(200).send({

message:"Ok",

publication : publicationStored

});

})

}

module.exports = {

probando,

savePublication

}

Creamos la ruta

*//============================AQUI VAN LOS POST========================================//*

*/\*guardar una publicacion\*/*

api.post('/save-publication', md\_auth.ensureAuth, PublicationController.savePublication);

probamos:

<http://localhost:3800/api/save-publication>

{

"message": "Ok",

"publication": {

"\_id": "5bdcb8e0834d061874b22ab7",

"text": "mi tercerapublicacion",

"file": null,

"user": "5bd8832632c4f935880b5604",

"create\_at": "1541191904",

"\_\_v": 0

}

}

Ahora hagamos un metodo que nos liste las publicaciones de la aplicacion:

Básicamente esta función devolverá todas las publicaciones de los usuarios que yo sigo, entonces se recogerá el id del usuario identificado se hace un find de todos los usuarios que estoy siguiendo y devolverme las publicaciones de esos usuarios que estoy siguiendo.

1-lo primero es recoger el parámetro de la pagina que vendrá por la url

2-validamos si viene o no por el parans y le asignamos el valor

3-declaramos la variable itemperpage que contendrá cuantas publicaciones por pagina queremos.

*//=================GETPUBLICATION==================================//*

function getPublications(req, res){

var id\_user = req.user.sub;

var page = 1;

if(req.params.page){

page = req.params.page;

}

var itemsPerPage = 4;

}

5-buscamos todos los usuarios que sigamos y los metemos en un array

6-con la sentencia $in de mongoose buscare todas las publicaciones de todos los usuarios que están dentro del array follows\_clean y con esto obtendremos todas las publicaciones de los usuarios que seguimos de menor fecha de creación a mayor fecha de creación

*//=================GETPUBLICATION================================================//*

function getPublications(req, res){

var id\_user = req.user.sub;

var page = 1;

if(req.params.page){

page = req.params.page;

}

var itemsPerPage = 4;

FollowModel.find({user:id\_user}).populate('followed').exec((error, follows)=>{

if(error) return res.status(500).send({message:"error al ejecutar la consulta"});

if(!follows) return res.status(404).send({message:"no sigues a nadie"});

var follows\_clean = [];

follows.forEach((follow)=>{

follows\_clean.push(follow.followed);*//en este array tendre a todos los users que sigo*

})

*/\*ahora con la sentencia $in en mongoose buscare todas las publicaciones de esos usuarios*

*que estan dentro del array follows\_clean\*/* */\* cor el sort ordenaremos las*

*publicaciones de las nuevas a viejas \*/*

PublicationModel.find({user:{'$in':follows\_clean}}).sort('-created\_at').populate('user')

.paginate(page, itemsPerPage, (error, publications, total)=>{

if(error) return res.status(500).send({message:"error al ejecutar la consulta"});

if(!publications) return res.status(404).send({message:"no sexisten publicaciones"});

return res.status(200).send({

total\_items : total,

publicatios: publications,

pages : Math.ceil(total/itemsPerPage),

page : page

});

})

})

}

exportemos

module.exports = {

probando,

savePublication,

getPublications

}

Probemos

<http://localhost:3800/api/publications/1>

{

"total\_items": 3,

"publicatios": [

{

"\_id": "5bdcb857ee99f31904e91ce5",

"text": "mi primera publicacion",

"file": null,

"user": {

"\_id": "5bd8832632c4f935880b5604",

"name": "Jesus Dario",

"surname": "Marenco Porto",

"nick": "daro",

"email": "admin@admin.com",

"role": "ROLE\_USER",

"image": null,

"password": "$2a$10$1UlqqXS5.C9.xX6VZqcrd.owK5xfV7l4toQbhuLVzZZBn69ODgIp2",

"\_\_v": 0

},

"create\_at": "1541191767",

"\_\_v": 0

},

{

"\_id": "5bdcb8976d5e160598e2b69d",

"text": "mi segunda publicacion",

"file": null,

"user": {

"\_id": "5bd8832632c4f935880b5604",

"name": "Jesus Dario",

"surname": "Marenco Porto",

"nick": "daro",

"email": "admin@admin.com",

"role": "ROLE\_USER",

"image": null,

"password": "$2a$10$1UlqqXS5.C9.xX6VZqcrd.owK5xfV7l4toQbhuLVzZZBn69ODgIp2",

"\_\_v": 0

},

"create\_at": "1541191831",

"\_\_v": 0

},

{

"\_id": "5bdcb8e0834d061874b22ab7",

"text": "mi tercerapublicacion",

"file": null,

"user": {

"\_id": "5bd8832632c4f935880b5604",

"name": "Jesus Dario",

"surname": "Marenco Porto",

"nick": "daro",

"email": "admin@admin.com",

"role": "ROLE\_USER",

"image": null,

"password": "$2a$10$1UlqqXS5.C9.xX6VZqcrd.owK5xfV7l4toQbhuLVzZZBn69ODgIp2",

"\_\_v": 0

},

"create\_at": "1541191904",

"\_\_v": 0

}

],

"pages": 1,

"page": "1"

}

**Creemos un metodo que nos devuelva una sola publicacion es decir conseguir una publicacion por su id:**

*//==== GET PUBLICATION ES DECIR CONSEGUIR UNA PUBLICACION POR SU ID QUE NOS LLEGARA POR URL ============*

function getPublication(req, res){

var id\_publication = req.params.id;

PublicationModel.findById(id\_publication, (error, publication)=>{

if(error) return res.status(500).send({message:"error al ejecutar la consulta"});

if(!publication) return res.status(404).send({message:"no existe la publicacion"});

return res.status(200).send({

publicatio: publication,

});

})

}

Exportamos:

module.exports = {

probando,

savePublication,

getPublications,

getPublication

}

Creamos la ruta:

*/\*getPublicatios -->trae una publicacion por su id \*/*

api.get('/publication/:id', md\_auth.ensureAuth, PublicationController.getPublication);

probemos:

<http://localhost:3800/api/publication/5bdcb857ee99f31904e91ce5>

{

"publicatio": {

"\_id": "5bdcb857ee99f31904e91ce5",

"text": "mi primera publicacion",

"file": null,

"user": "5bd8832632c4f935880b5604",

"create\_at": "1541191767",

"\_\_v": 0

}

}

**Creemos el método para eliminar publicaciones:**

*//======================DELETE PUBLICATION========================//*

function deletePublication(req, res){

var id\_publication = req.params.id;

*//de esta manera puedo ta,bien eliminar documentos, puedes observar que solo se elimina el documento*

*//de la publicacion si es el usuario que lo creo el que hace la solicitud*

PublicationModel.remove({'user':req.user.sub, '\_id':id\_publication}).exec((error, publicationRemove)=>{

if(error) return res.status(500).send({message:"error al borrar la publicacion"});

if(!publicationRemove) return res.status(404).send({message:"no existe la publicacion"});

res.status(200).send({publication:publicationRemove});

});

}

module.exports = {

probando,

savePublication,

getPublications,

getPublication,

deletePublication

}

*/*

*/=========================AQUI VAN LOS DELETE====================================//*

*/\*borrar publicacion\*/*

api.delete('/delete-publication/:id', md\_auth.ensureAuth, PublicationController.deletePublication);

probemos:

si un usuario que no creo la publicacion trata de borrarla no podra hacerlo y recibira la siguiente respuesta:

<http://localhost:3800/api/delete-publication/5be1980dd1674e0aec732b87>

{

"publication": {

"n": 0,

"ok": 1

}

}

Pero si es el usuario dueño de la publicación quien la borra podrá borrarla y recibirá el siguiente mensaje:

{

"publication": {

"n": 1,

"ok": 1

}

}

**Creemos la función para añadir una imagen a la publicación:**

*//===============================SUBIR IMAGEN DE LA PUBLICACION===========================//*

function uploadImage(req, res){

*//regoge desde la url el id del usuario que vamos a actualizar*

var publicationId = req.params.id;

*//si existe el componente files dentro del req quiere decir que hay un archivo cargando*

if(req.files.image != undefined){

*//el campo que vamos a pasar por post es el campo image*

var file\_path = req.files.image.path;

console.log(file\_path);

*//en la variable file\_split convertimos tdo el path en un array con el metodo .split*

var file\_split = file\_path.split('\\');

*//ahora tomamos la ultima posicionq ue es donde se guardo el nombre de la imagen que ira a uploads/users*

var file\_name = file\_split[2];

*//ahora para validar que sea una imagen debo validar su extension*

var ext\_split = file\_name.split('\.')

var file\_ext = ext\_split[1];

*//si el archivo contiene una extension de tipo de imagen entonces actualiza la imagen del usuario*

if(file\_ext == "png" || file\_ext == "jpg" || file\_ext == "jpeg" || file\_ext == "gif"){

*//solo el dueño de la publicacion puede subir imagenes en ella*

PublicationModel.find({'user':req.user.sub, '\_id':publicationId}).exec((error, publication)=>{

if(publication.length >= 1){

*//actualicemos el documento de la publicacion*

PublicationModel.findById(publicationId, (err, publicationImage)=>{

if(err){

return res.status(400).json({message:"error en la peticion."});

}

var pathViejo = "./uploads/publications/"+publicationImage.file;

*// ====en caso de que ya exista una imagen la borramos ===== //*

if(fs.existsSync(pathViejo)){

fs.unlink(pathViejo)

}

*//guardamos la imagen nueva*

publicationImage.file = file\_name;

publicationImage.save((err, publicationImageSave)=>{

if(err) return res.status(500).send({message:"Error en la peticion"});

return res.status(200).json({publicationImageSave});

})

})

}else{

return res.status(404).json({message:"no tienes permiso para subir una imagen a esta publicacion"});

}

})

}else{*//sino enviar un mensaje que diga no es un tip de archivo valido*

removeFieUploads(res, file\_path);

}

}else{

return res.status(400).json({message:"No files were uploaded."});

}

}

function removeFieUploads(res, file\_path, message){

fs.unlink(file\_path, (err)=>{*//elimina el archivo subido de la ruta*

return res.status(200).json({message:"Tipo de archivo no valido."});

});

}

*//==================================Devolver Imagen de la publicacion ============================//*

function getImageFile(req, res){

var image\_file = req.params.imageFile;

var path\_file = './uploads/publications/'+image\_file;

fs.exists(path\_file, (exists)=>{

if(exists){

return res.sendFile(path.resolve(path\_file))

}else{

return res.status(200).json({message:"No existe la imagen"});

}

})

}

module.exports = {

probando,

savePublication,

getPublications,

getPublication,

deletePublication,

uploadImage,

getImageFile

}

Creemos las rutas:

*/\*getimagen de la publication\*/*

api.get('/get-image-pub/:imageFile', md\_auth.ensureAuth, PublicationController.getImageFile );

*//==========================AQUI VIENEN LOS POST==================================//*

*/\*upload de la imagen de la publication\*/*

api.post('/publication-image-pub/:id', [md\_auth.ensureAuth, md\_upload], PublicationController.uploadImage);

probemos

<http://localhost:3800/api/publication-image-pub/5be1a51cb9c806041c662410>

{

"publicationImageSave": {

"\_id": "5be1a51cb9c806041c662410",

"text": "eeeeeeeeeeeeeeeeeeeeeeeeee",

"file": "y1bLCrTRw2UHQ3ijEGtuLD48.jpg",

"user": "5bd8cd8f55244c3a244d5fe5",

"create\_at": "1541514524",

"\_\_v": 0

}

}

**Modifiquemos el metodo getcountFollow de usersControllers para que nos cuente las publicaciones que hemos creado y las publicaciones en las que hemos comentado o dado like o algo.**

De modo que en userController crearemos este metodo:

1-requerimos el modelo de publicaciones:

2-vamos al metodo getCountFollow

*//Funcion que me cuenta cuantos seguidores tengo y a cuantos sigo==========*

async function getCountFollow(user\_id){

var following = await FollowModel.count({'user': user\_id}, (error, count)=>{

if(error) return handleError(err);

return count;

})

var followed = await FollowModel.count({"followed":user\_id}, (error, count)=>{

if(error) return handleError(err);

return count;

})

*//contemos nuestras publications:*

var publications = await PublicationModel.count({"user":user\_id}, (error, count)=>{

if(error) return handleError(err);

return count;

})

return {

followin: following,

followed : followed,

publications :publications

}

}

<http://localhost:3800/api/counters/5bd8cd8f55244c3a244d5fe5>

{

"value": {

"followin": 1,

"followed": 2,

"publications": 7

}

}

**CREEMOS EL CONTROLADOR DE LA MENSAJERIA PRIVADA:**

1-en el modelo de mensaje hay un par de cosas por mejorar, agregaremos el campo viwed que nos indicara si esta visto o no visto:

'use strict'

var mongoose = require('mongoose');

var Schema = mongoose.Schema;

var MessageSchema = Schema({

emmiter : {type: Schema.ObjectId, ref :'User'}, *//usuario que emite el mensaje*

receiber :{type: Schema.ObjectId, ref :'User'},*//usuario recibe el mensaje*

text:String,

created\_at:String,

viewed : String *//este campo guaradra si el mensaje esta visto o no esta visto*

});

moule.exports = mongoose.model('Message', MessageSchema);

2-creemos el Nuevo controlador llamado messageController.js

3-carguemos las librerías necesarias:

3-creemos el método de prueba

4-exportamos

5-creamos fichero messageRoutes

6-cargarmos express e inicializamos el Router

6.0-requiero el controlador de mensajes

6.1-crear la ruta

7-exportamos api

8-requerimos la ruta en app.js

9-inicializamos el middleware.

10-probamos

'use strict';

var moment = require('moment');

var mongoosePaginate = require('mongoose-pagination');

var UserModel = require('../models/userModel');

var FollowModel = require('../models/followModel');

var MessageModel = require('../models/messageModel');

function pruebaMessage(req, res){

res.status(200).send({message:"bien esta ok la prueba"});

}

module.exports ={

pruebaMessage

}

'use strict'

var express = require('express');

var api = express.Router();

var md\_auth = require('../middlewares/autenticated');

var MessageController = require('../controllers/messageController');

*//=========================AQUI VIENEN TODOS LOS GETS================================//*

*/\*Metodo de prueba para los mensajes\*/*

api.get('/pruebamessage', md\_auth.ensureAuth, MessageController.pruebaMessage );

module.exports = api;

var messageRoutes = require('./routes/messageRoutes');

*//middleware para definir la ruta base*

app.use('/api', userRoutes);

app.use('/api', followRoutes);

app.use('/api', publicationRoutes);

app.use('/api', messageRoutes);

prueba

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

{

"message": "bien esta ok la prueba"

}

h

**creemos un metodo para enviar mensajes privados:**

*//==================SAVE MESSAGE*

function saveMessage(req, res){

var body = req. body;

*//si hay texto y me llega un reciber entonces continuo*

if(body.text.length > 1 && body.receiber){

var message = new MessageModel();

message.emitter = req.user.sub;

message.receiber = body.receiber;

message.text = body.text;

message.created\_at = moment().unix();

message.save((error, messageStored)=>{

if(error) return res.status(500).send('erro en la peticion');

if(!messageStored) return res.status(404).send('error al enviar el mensaje');

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

})

}else{

return res.status(200).send({message:'envia los datos necesarios'});

}

}

module.exports ={

pruebaMessage,

saveMessage

}

api.post('/message', md\_auth.ensureAuth, MessageController.saveMessage );

**creemos el metodo para listar los mensajes que recibimos.**

*//=====================GER RECEIVER MESSAGE*

function getReceiverMessage(req, res){

var userId = req.user.sub;

var page = 1;

if(req.params.page >=1){

page = req.params.page;

}

var itemsPerPage = 4;

MessageModel.find({receiver:userId})

.populate('emitter', 'name')

.paginate(page, itemsPerPage, (error, messages, total)=>{

if(error) return res.status(500).send('erro en la peticion');

if(!messages) return res.status(404).send('no hay mensajes que mostrar');

return res.status(200).send({

total:total,

pages: Math.ceil(total/itemsPerPage),

messages

});

});

}

module.exports ={

pruebaMessage,

saveMessage,

getReceiverMessage

}

Creemos la ruta

api.get('/my-messages', md\_auth.ensureAuth, MessageController.getReceiverMessage );

probemos:

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

{

"total": 2,

"pages": 1,

"messages": [

{

"\_id": "5be1c187dc9e2a1c786fc1d1",

"emitter": {

"\_id": "5bd8cd8f55244c3a244d5fe5",

"name": "cindy"

},

"receiver": "5bd8832632c4f935880b5604",

"text": "te encuentras alli?",

"created\_at": "1541521799",

"\_\_v": 0

},

{

"\_id": "5be1c18edc9e2a1c786fc1d2",

"emitter": {

"\_id": "5bd8cd8f55244c3a244d5fe5",

"name": "cindy"

},

"receiver": "5bd8832632c4f935880b5604",

"text": "hola como estas",

"created\_at": "1541521806",

"\_\_v": 0

}

]

}

**Listemos los mensajes que enviamos:**

*//====================GET EMITT MESSAGE lista los mensajes que he enviado*

function getEmmitMessages(req, res){

var userId = req.user.sub;

var page = 1;

if(req.params.page >=1){

page = req.params.page;

}

var itemsPerPage = 4;

console.log("este es el usuario que hace la consulta", req.user.sub);

MessageModel.find({emitter:userId})

.populate('emitter receiver', 'name surname nick image')

.paginate(page, itemsPerPage, (error, messages, total)=>{

if(error) return res.status(500).send('erro en la peticion');

if(!messages) return res.status(404).send('no hay mensajes que mostrar');

return res.status(200).send({

total:total,

pages: Math.ceil(total/itemsPerPage),

messages

});

});

}

module.exports ={

pruebaMessage,

saveMessage,

getReceiverMessage,

getEmmitMessages

}

*/\*Metodo de para obtener mensajes que el usuario ha enviado\*/*

api.get('/messages/:page?', md\_auth.ensureAuth, MessageController.getEmmitMessages );

prueba

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

{

"total": 2,

"pages": 1,

"messages": [

{

"\_id": "5be1c187dc9e2a1c786fc1d1",

"emitter": {

"\_id": "5bd8cd8f55244c3a244d5fe5",

"name": "cindy",

"surname": "sanchez",

"nick": "nena",

"image": "H06UCChuLdeYFVqT\_t24xaJb.png"

},

"receiver": {

"\_id": "5bd8832632c4f935880b5604",

"name": "Jesus Dario",

"surname": "Marenco Porto",

"nick": "daro",

"image": null

},

"text": "te encuentras alli?",

"created\_at": "1541521799",

"\_\_v": 0

},

{

"\_id": "5be1c18edc9e2a1c786fc1d2",

"emitter": {

"\_id": "5bd8cd8f55244c3a244d5fe5",

"name": "cindy",

"surname": "sanchez",

"nick": "nena",

"image": "H06UCChuLdeYFVqT\_t24xaJb.png"

},

"receiver": {

"\_id": "5bd8832632c4f935880b5604",

"name": "Jesus Dario",

"surname": "Marenco Porto",

"nick": "daro",

"image": null

},

"text": "hola como estas",

"created\_at": "1541521806",

"\_\_v": 0

}

]

}

Creemos un metodo que nos cuente los mensajes que tenemos sin leer

*//GETUNVIEWEDMESSAGES este metodo muestra el conteo de los mensajes*

*// que me enviaron pero no he visto es decir los no vistos*

function getUnviewedMessages(req, res){

var userId = req.user.sub;

MessageModel.count({receiver:userId, viewed:'false'}).exec((error, count)=>{

if(error) return res.status(500).send('error en la peticion');

if(!count) return res.status(404).send('no hay mensajes que mostrar');

return res.status(200).send({

unviewed:count

});

})

}

module.exports ={

pruebaMessage,

saveMessage,

getReceiverMessage,

getEmmitMessages,

getUnviewedMessages

}

*/\*Metodo de paracontar los mensajes no leidos\*/*

api.get('/unviewed-messages', md\_auth.ensureAuth, MessageController.getUnviewedMessages );

<http://localhost:3800/api/unviewed-messages>

{

"unviewed": 1

}

**Creemos un metodo para marcar los mensajes como leidos:**

Es decir cuando leamos un mensaje que se marque como leído:

*//SETVIEWEDMESSAGES este metodo cambiara el estado de los mensajes que leemos a leidos*

function setViewedMessagess( req, res){

var userId = req.user.sub;

MessageModel.update({receiver:userId, viewed:'false'}, {viewed:'true'}, {'multi':true}, (error, messageUpdate)=>{

if(error) return res.status(500).send('error en la peticion');

if(!messageUpdate) return res.status(404).send('no hay mensajes que mostrar');

return res.status(200).send({

messageUpdate:messageUpdate

});

})

}

module.exports ={

pruebaMessage,

saveMessage,

getReceiverMessage,

getEmmitMessages,

getUnviewedMessages,

setViewedMessagess

}

Creamos la ruta

*/\*Metodo para que al ver los mensajes los pase a leidos o mas bien cambie su estado a true(leidos)\*/*

api.get('/set-viewed-messages', md\_auth.ensureAuth, MessageController.setViewedMessagess );

<http://localhost:3800/api/set-viewed-messages>

{

"messageUpdate": {

"n": 3,

"nModified": 3,

"ok": 1

}

}

**Acceso CORS**: El Intercambio de Recursos de Origen Cruzado ([CORS](https://developer.mozilla.org/en-US/docs/Glossary/CORS)) es un mecanismo que utiliza encabezados adicionales [HTTP](https://developer.mozilla.org/en-US/docs/Glossary/HTTP) para permitir que un [user agent](https://developer.mozilla.org/en-US/docs/Glossary/user_agent) obtenga permiso para acceder a recursos seleccionados desde un servidor, en un origen distinto (dominio), al que pertenece. Un agente crea una petición HTTP de origen cruzado cuando solicita un recurso desde un dominio distinto, un protocolo o un puerto diferente al del documento que lo generó.

Un ejemplo de solicitud de origen cruzado: El código JavaScript frontend de una aplicación web que se localizada en http://domain-a.com utiliza [XMLHttpRequest](https://developer.mozilla.org/es/docs/Web/API/XMLHttpRequest) para cargar el recurso http://api.domain-b.com/data.json.

Por razones de seguridad, los exploradores restringen las solicitudes HTTP de origen cruzado iniciadas dentro de un script. Por ejemplo, [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) y la API [Fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) siguen la [política de mismo-origen](https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy). Ésto significa que una aplicación que utilice esas APIs [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) sólo puede hacer solicitudes HTTP a su propio dominio a menos que se utilicen encabezados CORS.

**Configurtacion del CORS:**

Esto evitra problemas a la hora de hacer peticiones Ajax desde javascript o desde un frontend cualquiera.

Hoy vamos a ver como configurar cabeceras y el **acceso CORS en NodeJS** para un proyecto que estemos desarrollando. Si lo tenemos en producción es recomendable que el origen de las peticiones este limitado a la ip de tu cliente, pero bueno, esto depende de las necesidades.

Veamos las cabeceras necesarias para que a la hora de hacer peticiones AJAX desde un [frontend con Angular](https://www.udemy.com/curso-de-angular-2-4-5-avanzado-mean-jwt/?couponCode=ANAV-WEB)por ejemplo no haya ningun tipo de problema:

|  |
| --- |
| // configurar cabeceras http |
| 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(); | |

|  |
| --- |
| }); |

**INICIEMOS CON EL FRONTENT CREANDO UN PROYECTO DE ANGULAR CON new frontRed:**

1-instalemos la librerías que vamos a usar dentro de angular:

1. En el package.json del frontend en las dependencies vamos a añadir el bootstrap y jquery

"dependencies": {

"@angular/animations": "^6.0.3",

"@angular/common": "^6.0.3",

"@angular/compiler": "^6.0.3",

"@angular/core": "^6.0.3",

"@angular/forms": "^6.0.3",

"@angular/http": "^6.0.3",

"@angular/platform-browser": "^6.0.3",

"@angular/platform-browser-dynamic": "^6.0.3",

"@angular/router": "^6.0.3",

"core-js": "^2.5.4",

"rxjs": "^6.0.0",

"zone.js": "^0.8.26",

**"bootstrap": "3.3.6",**

**"jquery": "1.12"**

},

1. Ahora vamos a la consola y corremos el siguiente comando

npm update con esto se insyalaran las nuevas dependencias añadidas es decir el boostrap y el jquery.

1. Ahora vamos a configurar estas dos nuevas librerías para poder usarlas, vamos al fichero angular.json, también cambiemos el lugar de los estilos movamos el archivo raiz style.css a los assets, en el angular.json en lugar de que el css este en la raiz agregaremos la nueva dirección

"styles": [

"src/assets/styles.css"

],

1. En los scripts metamos todos los plugisns que hemos instalado nuevios es decir bootstrap y jquery.

"scripts": [

"node\_modules/jquery/dist/jquery.min.js",

"node\_modules/bootstrap/dist/js/bootstrap.min.js"

]

1. Ojo que hay que añadir los estilos del boostrap también en los styles

"styles": [

"src/assets/css/styles.css",

"node\_modules/bootstrap/dist/css/bootstrap.min.css"

],

1. Ahora hagamos que jquery pueda funcionar, para eso hay que declarar las variables de jquery ls componentes donde lo usemos, las declararemos debajo de los import como:

declare var jQuery:any;

declare var $:any;

**Hagamos la barra de navegacion que va a tener el proyecto:**

Tomemos el componente el html del componente app.component.ts y limpiémoslo para quitar la plantilla que trae angular 6 de manera que todo quede en blanco.

Creemos inicialmente una barra de navegación de la siguiente manera:

*<*div class="navigation col-lg-12"*>*

*<*nav class="navbar navbar-default"*>*

*<*div class="container-fluid"*>*

*<*div class="navbar-header"*>*

*<*a href="#" class="navbar-brand"*>*{{title}}*</*a*>*

*</*div*>*

*</*div*>*

*</*nav*>*

*</*div*>*

Este es el componente

import { Component } from '@angular/core';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

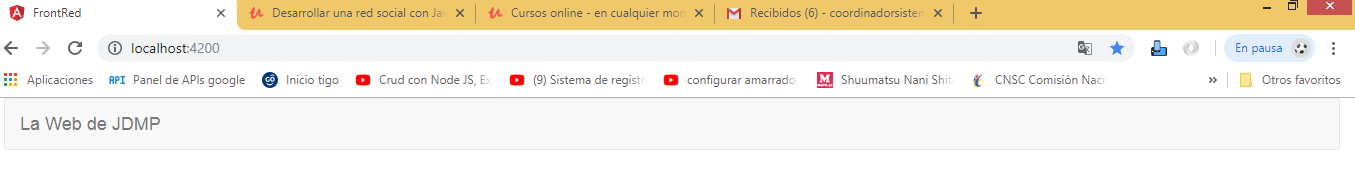
})

export class AppComponent {

title = 'La Web de JDMP';

}

Y asi se ve:



Ahora con css:

*/\*HEADER\*/*

.navigation{

margin:0px !important;

padding: 0px !important;

}

.navigation .navbar{

border-radius: 0px !important;

background: #e53c37;

border:none;

}

*/\*todas las etiquetas que esten dentro de la clase navbar tendran un color blanco\*/*

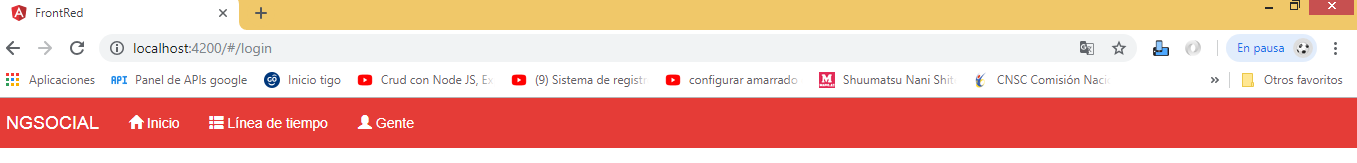
.navigation .navbar \*{

color:white;

}

<https://bootswatch.com> es una pagina de temas para boostrap aquí solo hay que reemplazar el archivo css que nos ofrecen por el original(recomendable al original cambiarle el nombre) para tener acceso a todo esto.

Vamos asi;



Creemos un par de componentes

1-creemos dentro de app una carpeta llamada components donde iran los componentes vamos a crear dos componentes inicialmente el login y el regfistro.

Ng g c components/login --spec=false

2-creemos el segundo componente el register

Ng g c components register --spec=false

3-creemos el routing de la aplicacion para utilizer las rutas y en function de la url se cargue un component u otro. Dentro de app vamos a crear un fichero llamado app.routing.ts que se encargara de esto.

1. Debemos importar los modulos y componentes necesarios del router. Estos dos son obligatorio

import { ModuleWithProviders } from '@angular/core';

import { Routes, RouterModule } from '@angular/router';

1. Importamos los componentes que creamos:

*//importamos los componentes creados*

import { LoginComponent } from './components/login/login.component';

import { RegisterComponent } from './components/register/register.component';

1. Creamos la constante donde almacenaremos las rutas y dentro crearemos las rutas relacionadas a sus componentes.

const appRoutes : Routes = [

{path: '', component: LoginComponent},

{path: 'login', component: LoginComponent},

{path: 'register', component: RegisterComponent},

];

1. Exportemos el provider de las rutas y el modulo de rutas con la nueva configuración para poder probar

export const appRoutingProviders : any[]=[];

export const routing : ModuleWithProviders = RouterModule.forRoot(appRoutes)

para que este archivo de rutas funcione hay que ir al app.module.ts e importar las dos constantes que exportamos en el fichero de routing.ts lo necesario que es:

import { routing, appRoutingProviders } from './app.routing';

y ahora deberemos pasar estas dos configuraciones los imports que se usan para cargar modulos es decir le pasaremos la instancia de un modulo que es el modulo de rutas.

imports: [

BrowserModule,

routing

],

Y en los providers cargamos los servicios de manera globar asi que debemos pasarle el appRoutingProvioders

providers: [

appRoutingProviders

],

Ahora para que todo esto se de hay que cargar en el html del componente principal una etiqueta muy importante llamada **<router-outlet></ router-outlet>**

*<*div class="navigation col-lg-12"*>*

*<*nav class="navbar navbar-default"*>*

*<*div class="container-fluid"*>*

*<*div class="navbar-header"*>*

*<*a href="#" class="navbar-brand"*>*{{title}}*</*a*>*

*</*div*>*

*<*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*>*

*</*div*>*

*</*nav*>*

*</*div*>*

*<*router-outlet*></*router-outlet*>*

Con esto ya podre usar las rutas de la url para cambiar de componentes

**Ahora creemos los modelos:**

1-dentro de app creemos una carpeta llamada models y allí dentro crearemos nuestros ficheros de modelos. El primer modelo que crearemos sera el de usuario, el fichero se llamara userModel.ts

2-exportamos enseguida y cresmos el constructor

3-cuando definimos propiedades como parámetros en el construcxtor me salto de golpe 3 pasos que abria que hacer y esto me evita tener que definir la propiedad, tener que pasarla como parámetro y tener que inicializarla con el this

export class UserModel{

constructor(

public \_id: string,

public name: string,

public surname : string,

public nick : string,

public email: string,

public password : string,

public role : string,

public image : string

){}

}

4-sigamos con el modelo de follow

export class FollowModel{

constructor(

\_id:string,

user : string,

followed:string

){}

}

5-vamos con el modelo de publication

export class PublicationModel{

constructor(

public \_id:string,

public text :string,

public create\_at:string,

public user:string

){}

}

5-vamos con el modelo de message:

export class MessageModel{

constructor(

public \_id: string,

public emitter: string,

public receiver : string,

public text : string,

public created\_at: string,

public viewed :string

){}

}

**Pagina de registro:**

Trabajemos en el componente de register.

Debemos importar el modelo de usuario y los modulos de rutas para hacer las redirecciones.

import { Router, ActivatedRoute, Params } from '@angular/router';

import { UserModel } from '../../models/userModel';

configuramos las propiedades del Router y del modelo para poder usarlas en la clase, esto se hace en el constructor

constructor(

private \_route: ActivatedRoute,

private \_router : Router

) {

this.title = "Registrate"

}

Ahora creamos una propiedad llamada user de tipo UserModel que luego instanciaremos en el constructor y le pasaremos sus propiedades vacias.

title : string;

user : UserModel;

constructor(

private \_route: ActivatedRoute,

private \_router : Router

) {

this.title = "Registrate"

this.user = new UserModel("", "", "", "", "", "", "", "");*//8 campos vacios corresponden a las 8n propiedades del modelo*

}

Para usar los formularios debere importar el modulo de los formularios en el **app.module.ts** que es el modulo principal el modulo a importar es el siguiente:

import { FormsModule } from '@angular/forms';

luego cargarlo en los imports

imports: [

BrowserModule,

routing,

FormsModule

],

Aprovechemos para cargar tambien en el app.module.ts el modulo para los servicios http

import { HttpModule } from '@angular/http';

y carguemoslo en los import

imports: [

BrowserModule,

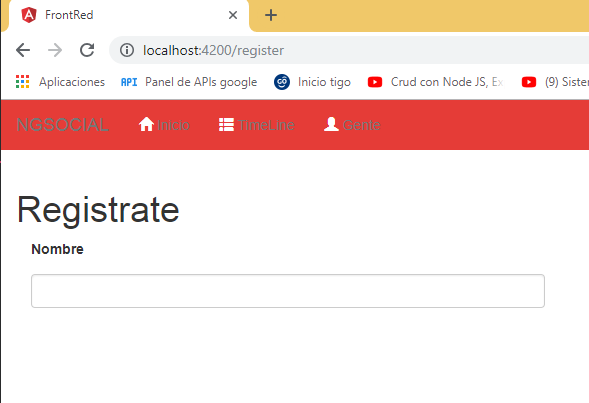
routing,

FormsModule,

HttpModule

],

Despues de esto podemos observer nuestro primer campo del formulario



Entonces este es el html del register.component.html.

*<*div class="col-lg-5"*>*

*<*h1 class="h1-strong"*>*{{title}}*</*h1*>*

*<*form #registerForm="ngForm" (ngSubmit)="onSubmit(registerForm)" class="col-lg-12"*>*

*<*p*><*label*>*Nombre*</*label*>*

*<*input type="text" name="name" #name="ngModel" [(ngModel)]="user.name" class="form-control" required*>*

*<*span \*ngIf="!name.valid && name.touched "*>*

Este campo es obligatiorio

*</*span*>*

*</*p*>*

*<*p*><*label*>*Apellidos*</*label*>*

*<*input type="text" name="surname" #surname="ngModel" [(ngModel)]="user.surname" class="form-control" required*>*

*<*span \*ngIf="!surname.valid && surname.touched "*>*

Este campo es obligatiorio

*</*span*>*

*</*p*>*

*<*p*><*label*>*Apodo*</*label*>*

*<*input type="text" name="nick" #nick="ngModel" [(ngModel)]="user.nick" class="form-control" required*>*

*<*span \*ngIf="!nick.valid && nick.touched "*>*

Este campo es obligatiorio

*</*span*>*

*</*p*>*

*<*p*><*label*>*Email*</*label*>*

*<*input type="email" name="email" #email="ngModel" [(ngModel)]="user.email" class="form-control" required pattern="[a-z0-9.\_%+-]+@[a-z0-9.-]+\.[a-z]{2,4}$"*>*

*<*span \*ngIf="!email.valid && email.touched "*>*

Este campo es obligatiorio o email ingresado es incorrecto

*</*span*>*

*</*p*>*

*<*p*><*label*>*Password*</*label*>*

*<*input type="password" name="password" #password="ngModel" [(ngModel)]="user.password" class="form-control" required*>*

*<*span \*ngIf="!password.valid && password.touched "*>*

Este campo es obligatiorio

*</*span*>*

*</*p*>*

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

*</*form*>*

*</*div*>*

Aeste es el css actual del register es decir lo que esta en el header y los generales.

*/\*ESTILOS GENERALES\*/*

.h1-strong{

font-size: 35px;

font-weight: bold;

margin-left: 13px;

border-bottom: 1px solid #eee;

padding-bottom: 15px;

margin-bottom: 20px;

}

*/\*HEADER\*/*

.navigation{

margin:0px !important;

padding: 0px !important;

}

.navigation .navbar{

border-radius: 0px !important;

background: #e53c37;

border:none;

}

*/\*todas las etiquetas que esten dentro de la clase navbar tendran un color blanco\*/*

.navigation .navbar \*{

color:white !important;

}

Y este es el componente como esta actualmente:

import { Component, OnInit } from '@angular/core';

import { Router, ActivatedRoute, Params } from '@angular/router';

import { UserModel } from '../../models/userModel';

@Component({

selector: 'app-register',

templateUrl: './register.component.html',

styleUrls: ['./register.component.css']

})

export class RegisterComponent implements OnInit {

title : string;

user : UserModel;

constructor(

private \_route: ActivatedRoute,

private \_router : Router

) {

this.title = "Registrate"

this.user = new UserModel("", "", "", "", "", "", "", "");*//8 campos vacios corresponden a las 8n propiedades del modelo*

}

ngOnInit() {

}

}

**Creemos el servicio que se comunique con el api rest y podamos consumir para poder guardar desde el formulario de registro los usuarios.**

Para poder hacer esto debemos importar el httpClient que esta mas optimizado , esto lo importamos en el app.module.ts

import { HttpClientModule } from '@angular/common/http';

y quitamos el menos optimizado:

*//import { HttpModule } from '@angular/http'; este no es tan optimizado como el siguiente*

imports: [

BrowserModule,

routing,

FormsModule,

HttpClientModule

],

Ahora para crear los servicios crearemos una carpeta llamada services dentro tendremos los servicios de cada entidad del modelo

ng g s services/userServices –spec=false

se nos crea un servicio con el modulo Injectable que es el que permite inyectar el servicio en los componentes y otros servicios.

import { Injectable } from '@angular/core';

@Injectable({

providedIn: 'root'

})

export class UserServicesService {

constructor() { }

}

Vamos a usar el modulo HttpClient y el HttpHeaders para poder hacer las peticiones Ajax y enviar las cabeceras. También importaremos el objeto observable para poder recoger las respuesta que devuelve el api, e importamos el modelo de usuario también.

import { Injectable } from '@angular/core';

import { HttpClient, HttpHeaders } from '@angular/common/http';

import { Observable, Subject, pipe } from 'rxjs';

import { UserModel } from '../models/userModel';

@Injectable({

providedIn: 'root'

})

export class UserServicesService {

constructor() { }

}

Creemos un fichero en services llamado global donde tendre una variable exportada con la url.

export var GLOBAL = {

url : 'http://localhost/3800/api/'

}

Importemos esta variable en el servicio de usuario. Inicialicemos el HttpClient en el constructor

Y también la url.

import { Injectable } from '@angular/core';

import { HttpClient, HttpHeaders } from '@angular/common/http';

import { Observable, Subject, pipe } from 'rxjs';

import { UserModel } from '../models/userModel';

import { GLOBAL } from './global';

@Injectable({

providedIn: 'root'

})

export class UserServicesService {

public url : string;

constructor(

public \_http : HttpClient

){

this.url = GLOBAL.url;

}

register(){

console.log(this.url);

}

}

Creemos el primer servicio llamado register y pasamos en un clg la url, para probarlo vamos al componente register, importamos el servicio, lo inicializamos en el constructor y ejecutamos en el oninit debería mostrarnos en la consola la url.

import { Component, OnInit } from '@angular/core';

import { Router, ActivatedRoute, Params } from '@angular/router';

import { UserModel } from '../../models/userModel';

import { UserServices } from 'src/app/services/user-services.service';

@Component({

selector: 'app-register',

templateUrl: './register.component.html',

styleUrls: ['./register.component.css']

})

export class RegisterComponent implements OnInit {

title : string;

user : UserModel;

constructor(

private \_route: ActivatedRoute,

private \_router : Router,

private \_user : UserServices

) {

this.title = "Registrate"

this.user = new UserModel("", "", "", "", "", "", "", "");*//8 campos vacios corresponden a las 8n propiedades del modelo*

}

ngOnInit() {

this.\_user.register();

}

onSubmit(){

console.log(this.user);

}

}

Efectivamente en la consola.

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

ahora si en el servicio el metodo quedara de la siguiente manera hay que iomportar el map

import { map, catchError } from "rxjs/operators";

register(user:UserModel):Observable<any>{

let params = JSON.stringify(user);

let headers = new HttpHeaders().set('Content-Type', 'application/json');*//cuando envie el json a la api lo podra procesar*

return this.\_http.post(this.url+'register', params, {headers:headers}).pipe(

map((resp:any)=>{

return resp;

})

)

}

El componente de esta manera:

onSubmit(){

this.\_user.register(this.user)

.subscribe((datos:any)=>{

console.log(datos);

});

}

**Ahora vamos al login**

Vamos al componente login:

Este es el html

*<*div class="col-lg-5"*>*

*<*h1 class="h1-strong"*>*{{title}}*</*h1*>*

*<*form #loginForm="ngForm" (ngSubmit)="onSubmit(loginForm)"*>*

*<*p*>*

*<*label*>*Correo Electronico*</*label*>*

*<*input type="email" name="email" #email="ngModel" [(ngModel)]="user.email" class="form-control" required pattern="[a-z0-9.\_%+-]+@[a-z0-9.-]+\.[a-z]{2,4}$"*>*

*<*span \*ngIf="!email.valid && email.touched "*>*

Este campo es obligatiorio o email ingresado es incorrecto

*</*span*>*

*</*p*>*

*<*p*>*

*<*label*>*Password*</*label*>*

*<*input type="password" name="password" #password="ngModel" [(ngModel)]="user.password" class="form-control" required *>*

*<*span \*ngIf="!password.valid && password.touched "*>*

Este campo es obligatiorio

*</*span*>*

*</*p*>*

*<*input type="submit" value="{{title}}" class="btn btn-success" [disabled]="!loginForm.form.valid"*>*

*</*form*>*

*</*div*>*

Este es el .ts de login

import { Component, OnInit } from '@angular/core';

import { Router, ActivatedRoute, Params } from '@angular/router';

import { UserModel } from '../../models/userModel';

import { UserServices } from '../../services/user-services.service';

@Component({

selector: 'app-login',

templateUrl: './login.component.html',

styleUrls: ['./login.component.css']

})

export class LoginComponent implements OnInit {

title : string;

user : UserModel;

constructor(

private \_user : UserServices

) {

this.title = "Identificate",

this.user = new UserModel("", "", "", "", "", "", "", "");*//8 campos vacios corresponden a las 8n propiedades del modelo*

}

ngOnInit() {

}

onSubmit(){

console.log(this.user);

}

}

Ahora creemos el servicio de logueo para el api rest