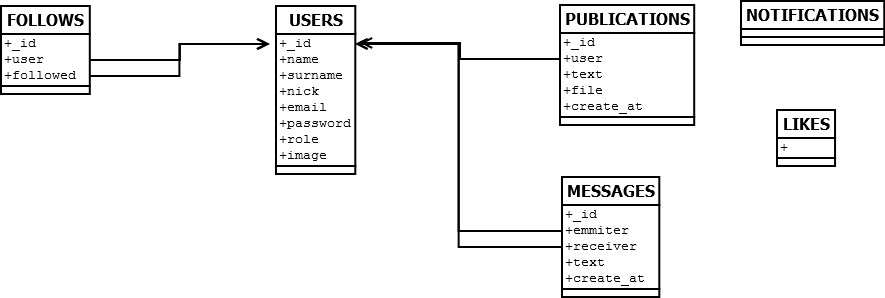
**PROYECTO MEAN 5**

**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 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 setial 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){

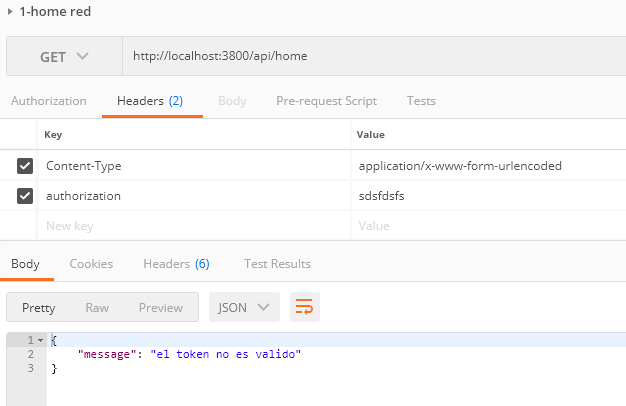
return res.status(403).send({message:"La peticion no tiene la cabecera de autenticacion"});

}

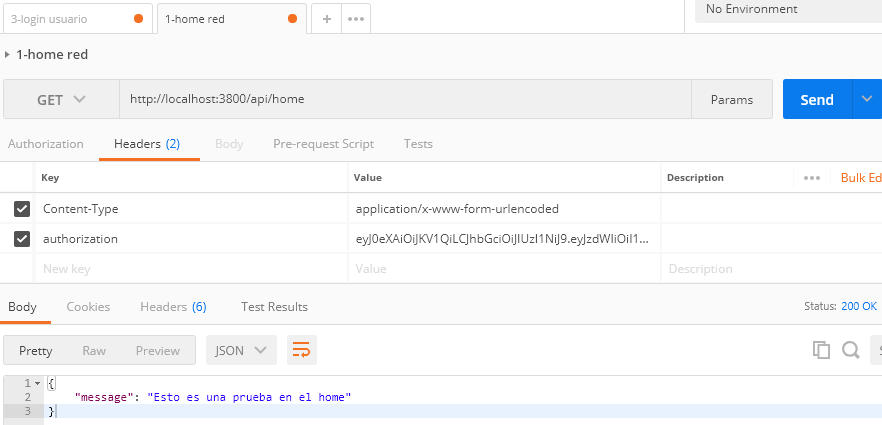
*//aqui cargamos el token*

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.