1. **Crear carpeta y abrimos (207)**

Crear los paquetes

Npm init : información de la api

Git init : control de versiones

1. **Inicio con el index.ts**

Tsc –init : archivo de validación tsconfig.json

Activo lo que necesito:

Outdir dist/

npm install -g typescript

Compilacion auto guardado de tsc a jsc tsc –w esta por defecto se install

Compilación auto npm install –g nodemon o confgurar pm2

1. **dependencias necesarias: (208)**

npm install express: para el servisio de web o rest

body-parser: tranformar información en js ojebto y poder utilizarla

cors: para las peticiones con header http ip etz

mongoose: DB

express-fileupload: para postear las imágenes

jsonwebtoken: para uutenticacion de app y sesión de usuarios

bcrypt: encritar las contraseñas

1. **Instalar las ayudas de tipado en modo dependencias de desarrollo –save-dev**

Npm install @types/express –save-dev

Npm install @types/mongoose –save-dev

Npm install @types/bcrypt –save-dev

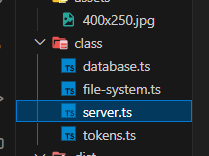
Npm install @types/jsonwebtoken –save-dev

Npm install @types/express-fileupload –save-dev

Npm install @types/uniqid –save-dev

Npm install @types/cors –save-dev

1. **Crear carpeta class y archivo server.ts donde estarán las instancias como express y el port: (209)**

****

import  express  from "express";

// exporto por defecto el servidor de escucha

export default class Server {

    public app: express.Application;

    public port: number= 3000;

    // start the class with this constructor

    constructor() {

        this.app =express();

    }

    //end constructor

    // listen for port

    start(callback: ()=> void ) {

     this.app.listen(this.port, callback);

    }

// and listen

}

server.ts {

* Import express for do express
* Export default classs : definimos la app y el port con sus atributos
* Constructor() : converimos el app como servidor express
* Start() se crea este método start() como callback y usamos app para que escuche por medio del port

Ya estaría lista y la importamos como instancia en el índex.ts

* Importamos
* Extanciamos en una variable : consta serve = new Server;
* Llamamos el método start y usamos una función de flecha para iniciar el servidor

serve.start(callback: ()=>{

console.log(`servidor corre en ${serve.port}`)});

1. **Estructurar las rutas dentro de una carpeta(210)**

* Carpeta Rutas, archivo usuario.ts
* Crear constante usando Router(); y se importa de express
* Creamos la ruta get definiendo la ruta, req, res y su => {res.json({})}
* Se exporta esta constante que contiene esta ruta
* Se importa en el index y la pasamos por el servidor serve.app.use(‘/use’, constante import);

import { Router} from "express";

const UserRoutes = Router();

  // validar usuario con token esto es un ejemplo para tener idea de como va

        UserRoutes.get('/', ( req: any, res: Response)=>{

            const usuario = req.usuario;

            res.json({

                ok: true,

                usuario

            });

        })

export default UserRoutes;

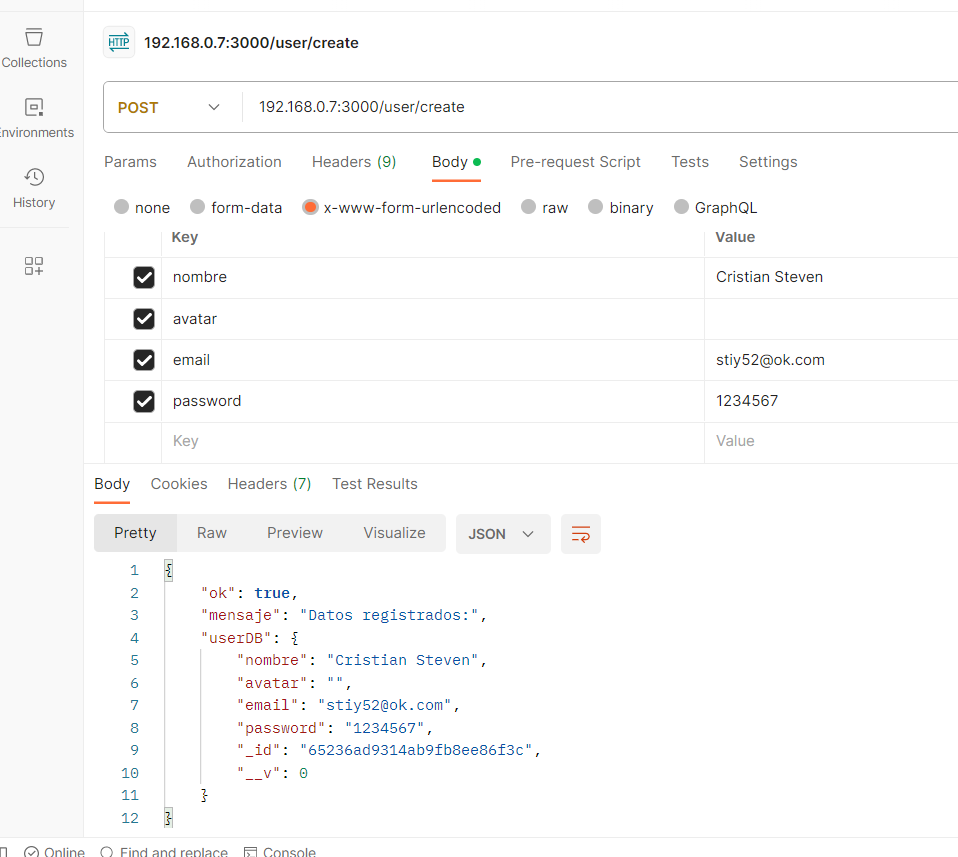
1. **instalar la DB mongodb o SQL o MYSQL y luego levantarla(211)**

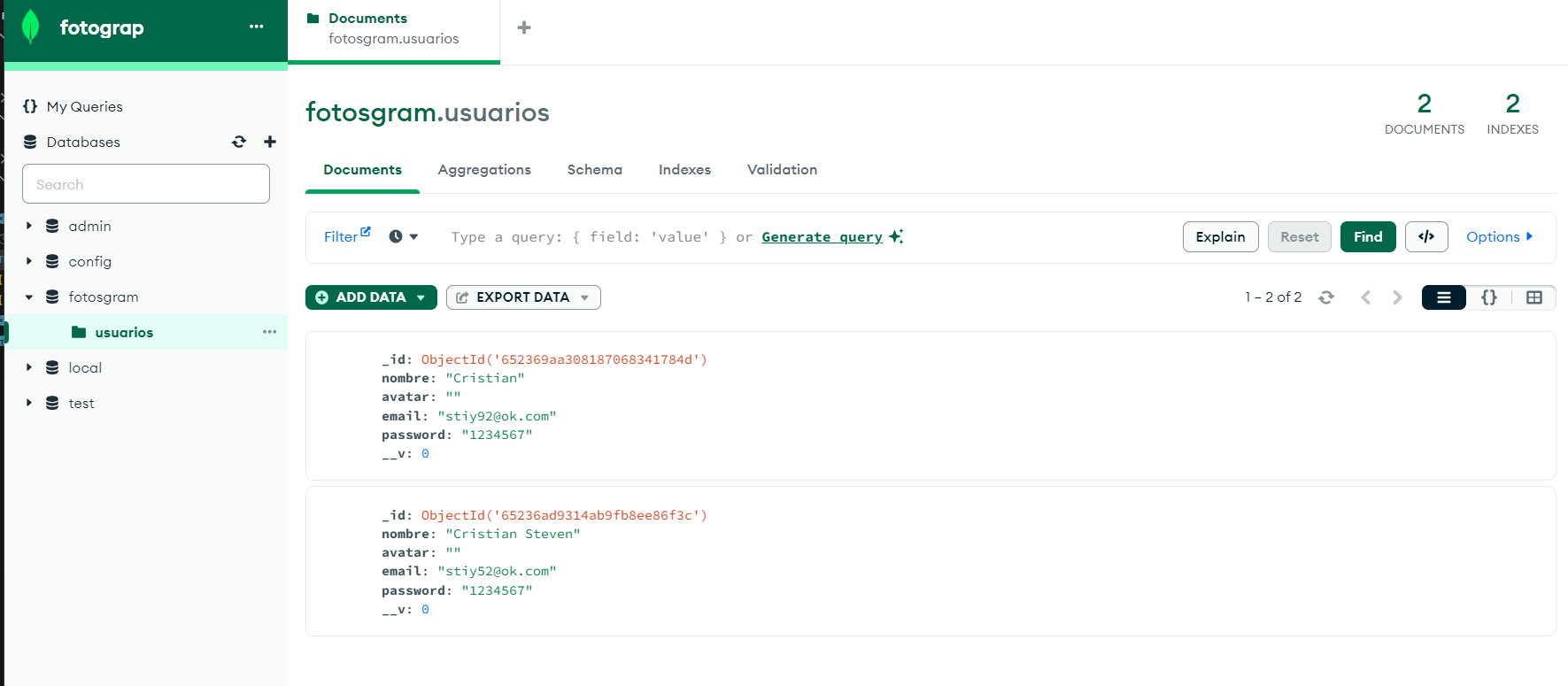
* Abrir la pagina oficial de mongodb y seguir las instrucciones de instalación
* Directorio de la db cd c:\
* Md “\data\db”
* Lanzar la db
* "C:\Program Files\MongoDB\Server\7.0\bin\mongod.exe" --dbpath="c:\data\db"
* Crear una carpetcta de conectiones a db y luego usarla en index o crear la conexión como mongoose y pasar los parámetros que dan la ubicación de la db e imprimir su conexión online.
* importar la constante desde el index y usar el server.app.use para que pase como ruta del usuario y cargue sus datos esto es para el registro desde el fronter

1. **Modelo de datos user(212)**

* Crear la carpeda de models
* Crear el usuario.model.ts
* En este archivo se debe de importar Schemas, model, document from mongoose
* Se crea una variable interface de los datos del usuario con el nombre que quiera jejej Iusuario para eso es el document aunque esto es opcional.
* Interface Isuario extends Document {definir sus campos: string}
* Luego se exporta como una constante nueva uniendo el modelo y el schema usuario
* Export const Usuario = model<Iusuario>(‘usuario’, usuarioSchema);

1. **Crear usuario desde postman**

* Aquí se empieza a utilizar los middleware en el index
* Método para recibir los datos por la ruta
* //middleware with bodyparser
* server.app.use(bodyParser.urlencoded({ extended:true}));
* server.app.use(bodyParser.json);
* then en la ruta usuario.ts recibimos los datos que vengan por medio de la ruta ósea desde cualquier otra app como postman
* const user = {
* nombre: req.body.nombre,
* avatar: req.body.avatar,
* email: req.body.email,
* password: req.body.password,
* }
* Then importo la clase modelo para hacer el proceso de registro a la base de datos
* Usuario.create(user).then(userDB =>{
* res.json({
* ok: true,
* mensaje: 'Datos registrados:',
* userDB
* });
* }).catch(err => {
* res.json({
* ok: false,
* err
* });
* });
* Los middleware deben de ir primero para que cuando llegue a la ruta ya existan los datos guardados como json en la constante user
* En postman podemos hacer el registro de prueba
* 



1. **encriptar contraseñas**

* Se importa bcrypt como dependencia de desarrollo (npm install) en el archivo usuario
* Then pasamos la contraseña con esta dependencia
* password: bcrypt.hashSync(req.body.password, 10)
* 10 vueltas y es necesario la promesa Sync

1. **Servicios de login user**

* Crear ruta para login en la ruta usuario.ts utilizando la estructura de promesas async await try catch
* UserRoutes.post('/login', async (req: Request, res: Response) =>{
* Dentro se crea una constante cargando el cuerpo del post
* try{
* const body = req.body;
* con esto buscamos el usuario
* const userDB = await Usuario.findOne({ email: body.email});
* if (!userDB) {
* return res.json({
* ok: false,
* mensaje: 'usuario/contraseña no son correctos'
* });
* }
* if (userDB.compararPassword(body.password)){
* res.json({
* ok: true,
* token: 'fdgjsklgsdfñgksdfgsdfjg'
* });
* } else {
* return res.json({
* ok:false,
* mensaje: 'usuario/contraseña no son correctos \*\*\*\*'
* });
* }
* } catch (err){
* throw err;
* }
* })
* En el modelo comparamos los datos con un método
* usuarioSchema.method('compararPassword', function(password: string = ''): boolean {
* if (bcrypt.compareSync(password, this.password)){
* return true,
* } else {
* return false;
* }
* });
* Importamos bcrypt
* Luego utilizo este método para validar la contraseña en el usuario.ts

Ejemplo de arriba

* if (userDB.compararPassword(body.password)){
* res.json({
* ok: true,
* token: 'fdgjsklgsdfñgksdfgsdfjg'
* }),
* } else {
* return res.json({
* ok:false,
* mensaje: 'usuario/contraseña no son correctos'
* });
* Es importante definir esta dependencia opcional arriba se puso de otro modo
* "noImplicitThis": false,                           /\* Enable error reporting when 'this' is given the type 'any'. \*/
* en el tsconfig.json para evitar este error
* if (bcrypt.compareSync(password, this.password)

11 creacion de jsonwebtoken 216

* Creamos una clase llamada token en la carpeta clase
* Luego exportamos clase por defecto y definimos dentro unas variables de forma privadas y estáticas donde estará la semilla con algún dato de refencia de token puede ser id o cc, como esta es mi semilla y otra variable con el tiempo de caducida
* import Jwt  from "jsonwebtoken";
* export default class TOKEN{
* private static token: string= 'this-is-my-seed';
* private static caducida: string= '30d';
* constructor(){
* }
* //payload pasan los datos para el token
* static getJwToken( payload: any): string {
* // firma
* return Jwt.sign({
* usuario: payload
* }, this.token, {expiresIn: this.caducida});
* }
* //comparo los datos que vienen con promesa
* static comprobarToken( userToken: string){
* return new Promise((resolve, reject)=>{
* Jwt.verify(userToken, this.token, (err, decoded)=> {
* if (err) {
* reject();
* } else {
* resolve( decoded);
* }
* })
* });
* }}

Al terminar esta clase nos dirigimos a la clase usuario y importamos esta clase para usarlo en una rurta login

Secrea una variable y se estancia Token

Al utilizar el . se obtiene los métodos stacticos los cuales no se necesitan inicializarce

if (userDB.compararPassword(body.password)){

            const tokenUser = TOKEN.getJwToken({

                \_id: userDB.\_id,

                nombre: userDB.nombre,

                email: userDB.email,

                avatar: userDB.avatar

            });

            res.json({

                ok: true,

                token: tokenUser

            });

Este mismo código lo pongo cuando se esta creando un usuario

Usuario.create(user).then(userDB =>{

        const tokenUser = TOKEN.getJwToken({

            \_id: userDB.\_id,

            nombre: userDB.nombre,

            email: userDB.email,

            avatar: userDB.avatar

        });

        res.json({

            ok: true,

            token: tokenUser

        });

}).catch(err => {

    res.json({

        ok: false,

        err

    });

Probamos en postman y nos debe de funionar

217 verificacion de un jsonwebtoken con ruta de actualizar usuario

* Crear carpeta middleware
* Autenticación.ts
* De la siguiente forma se recibe el toquen para autenticar el usuario puede ser por header o body
* import { Response, Request, NextFunction } from "express";
* import TOKEN from "../class/tokens";
* export const verificaToken = ( req: any, res: Response, next: NextFunction)=>{
* const userToken = req.get('x-token') || '';
* TOKEN.comprobarToken( userToken)
* .then( (decoded: any) =>{
* console.log ('Decoded', decoded);
* req.usuario = decoded.usuario;
* next();
* })
* .catch( err =>{
* res.json({
* ok: false,
* mensaje: 'Token falso'
* });
* });
* }

1. Update user from DB 218

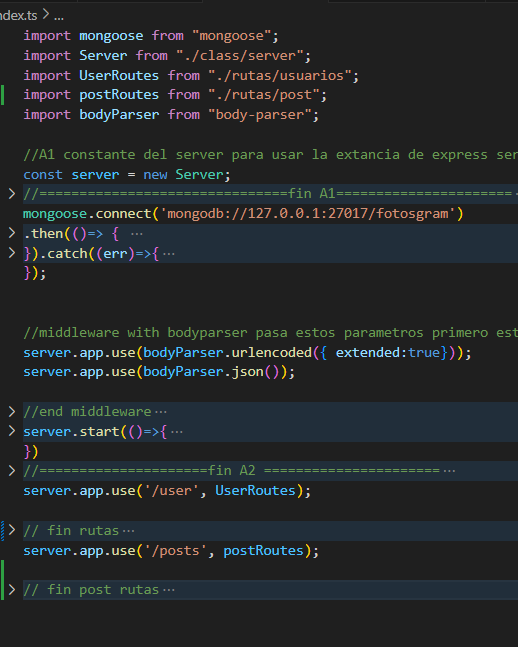
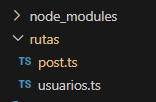
* Se crea la siguiente ruta en usuario.ts
* //ruta update usuario
* UserRoutes.post('/update', verificaToken, (req:any, res: Response)=> {
* const user = {
* nombre: req.body.nombre || req.usuario.nombre,
* email: req.body.email || req.usuario.email,
* avatar: req.body.avatar || req.usuario.avatar,
* };
* Usuario.findOneAndUpdate( req.usuario.\_id, user, {new: true}).then(
* (userDB) => {
* if(!userDB){
* return res.json({
* ok: false,
* mensaje: 'No existe un usuario con ese ID'
* });
* }
* const tokenUser = TOKEN.getJwToken({
* \_id: userDB.\_id,
* nombre: userDB.nombre,
* email: userDB.email,
* avatar: userDB.avatar
* });
* res.json({
* ok: true,
* token: tokenUser
* });
* }).catch( err =>{
* res.json({
* ok: false,
* mensaje: 'Error con la verificacion del token'
* });
* });
* });

224- modelo de base de datos post

* Primero creamos la clase en modelos
* Y agregamos este código
* import { Schema, Document, model } from "mongoose";
* const postSchema = new Schema({
* created: {
* type: Date
* },
* mensaje: {
* type: String
* },
* img: {
* type: String
* },
* coords: {
* type: String  //-13.3333
* },
* usuario:{
* type: Schema.Types.ObjectId,
* ref: 'Usuario',
* required: [true, 'Debe de existir una referencia a un usuario']
* }
* });
* // guardar fecha y se le anexa una interfaz
* postSchema.pre<Ipost>('save', function(next){
* this.created = new Date();
* next();
* });
* interface Ipost extends Document {
* created: Date;
* mensaje: string;
* img: string[];
* coords: string;
* usuario: string;
* }
* export const Post = model<Ipost>('post', postSchema);

225 Servicio para crear un POST

* creamos la clase post y le agregamos su ruta en el index.ts.



Dentro de la clase post ingresamos este código

import { Router, Response } from 'express';

import { Post } from '../Models/post.model';

import { verificaToken } from '../middlewares/autenticacion';

const postRoutes = Router();

//ruta verificar token de usuario

postRoutes.post('/', [verificaToken], (req: any, res:Response)=>{

    const body = req.body;

    body.usuario = req.usuario.\_id;

    Post.create( body ).then( async postDB => {

        //populate es para poblar la informacion del usuario deacuerdo a todos sus posteos

        await postDB.populate('usuario', '-password');

        res.json({

            ok: true,

            post: postDB

        })

    }).catch( err =>{

        res.json(err)

    });

})

export default postRoutes;

modifique el modelo.post

import { Schema, Document, model } from "mongoose";

import mongoose from 'mongoose';

const postSchema = new mongoose.Schema({

    created: {

        type: Date

    },

    mensaje: {

        type: String

    },

    img: [{

        type: String

    }],

    coords: {

        type: String  //-13.3333

    },

    usuario:{

        type: mongoose.Schema.Types.ObjectId,

        ref: 'usuario',

        required: [true, 'Debe de existir una referencia a un usuario']

    }

});

// guardar fecha y se le anexa una interfaz

postSchema.pre<Ipost>('save', function(next){

    this.created = new Date();

    next();

});

interface Ipost extends Document {

    created: Date;

    mensaje: string;

    img: string[];

    coords: string;

    usuario: string;

}

export const Post = mongoose.model<Ipost>('post', postSchema);

227 servicio para subir archivos

En el index

// FileUpload

server.app.use( fileUpload());

// fin FileUpload

Luego creamos la ruta en el post.ts

//ruta servicio para subir archivos

postRoutes.post( '/upload', [verificaToken], (req:any, res:Response)=>{

    if ( !req.files){

        return res.status(400).json({

            ok: false,

            mensaje: 'No se subió ninguna imagen'

        });

    }

    const file: FileUpload = req.files.image;

    if(!file){

        return res.status(400).json({

            ok: false,

            mensaje: 'No se subió ninguna archivo imagen'

        });

    }

    if(!file.mimetype.includes('image')){

        return res.status(400).json({

            ok: false,

            mensaje: 'Lo que subió no es una imagen'

        });

    }

    res.json({

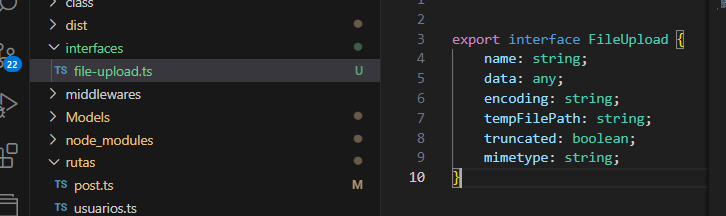
        ok: false,

        file: file.mimetype

    });

})

Creamos una carpeta de interfaces, un archivo ts y agregamos un modelo de archivos especificando los datos que se reciben al cargar un archivo



export interface FileUpload {

    name: string;

    data: any;

    encoding: string;

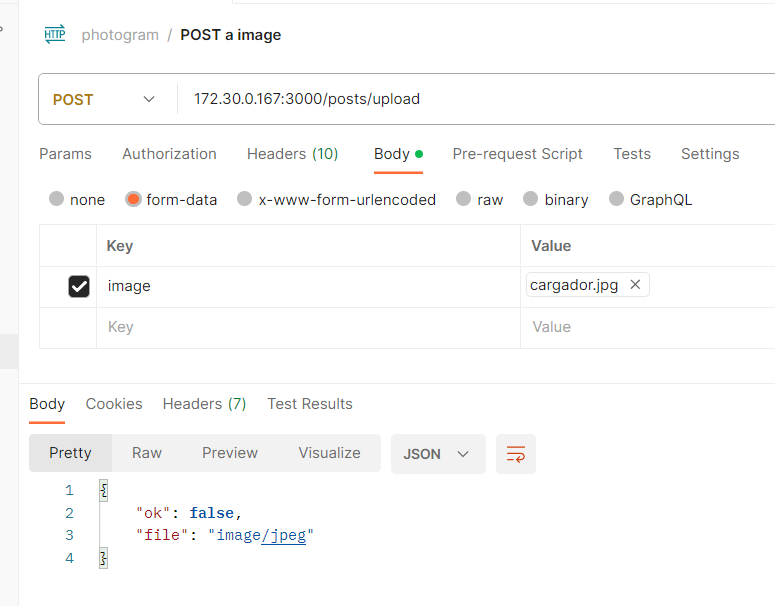
    tempFilePath: string;

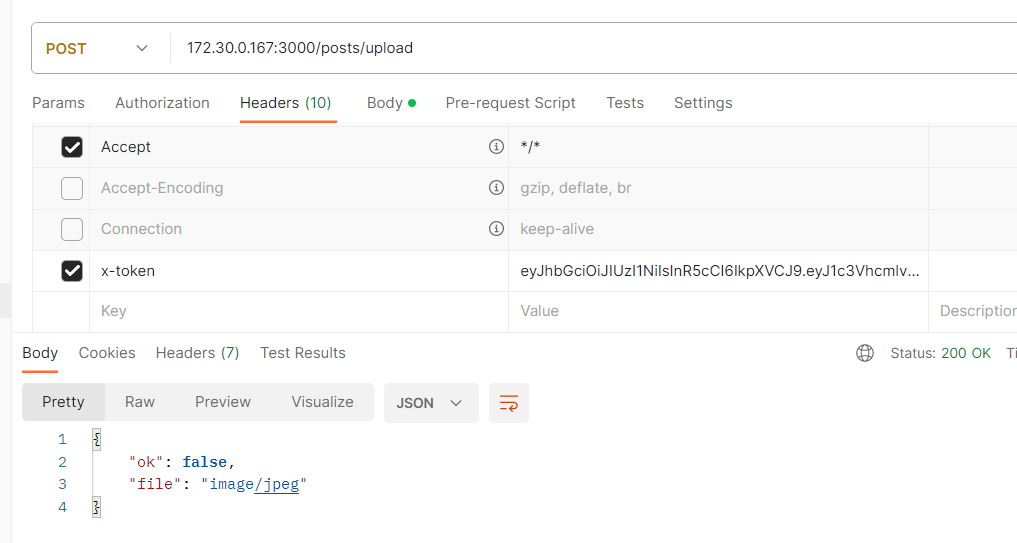
    truncated: boolean;

    mimetype: string;

}

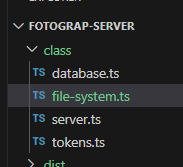
Luego probamos en postman con las rutas





229 clase para el manejo de filesystem

Creamos una nueva clase llamada filesystem.ts



import { FileUpload } from '../interfaces/file-upload';

//propio de node para las rutas de los archivos

import path from 'path';

//verificar si la carpeta< existe

import fs from 'fs';

export default class FileSystem {

    constructor(){};

    guardarImagenTemporal( file: FileUpload, userId: string){

        const path = this.crearCarpetaUsuario( userId);

    }

    private crearCarpetaUsuario( userId: string){

        const pathUser = path.resolve( \_\_dirname, '../uploads/', userId);

        // cargar imagen temporal para luego hacer el push

        const pathUserTemp = pathUser + '/temp';

        console.log(pathUser);

        //verificar si existe

        const existe= fs.existsSync( pathUser);

        //si no existe crea el directorio

        if(!existe){

            fs.mkdirSync( pathUser);

            fs.mkdirSync( pathUserTemp);

        }

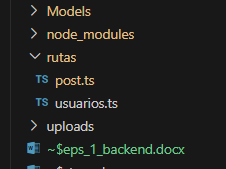
//retornar las imagenes temporales

        return pathUserTemp

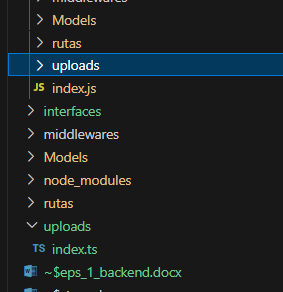
    }

    }

Creamos una carpeta temporal

 uploads donde quedaran los archivos

Debo de crear un archivo ts para que se cree en el dist automáticamente



Esto es para cuando envie en el post o app detecte esta carpeta y cree las temporales y el archivo

En la clase post

Importamos esa clase de guardar imagen temporal

En este punto ya el post iría así

import { Router, Response } from 'express';

import { Post } from '../Models/post.model';

import { verificaToken } from '../middlewares/autenticacion';

import { FileUpload } from '../interfaces/file-upload';

import FileSystem from '../class/file-system';

const postRoutes = Router();

   //crear instancia de guardar imagen temporal

   const fileSystem = new FileSystem();

//ruta  agregar un post con verificar token de usuario y almacenar sus posteos==========================

postRoutes.post('/', [verificaToken], (req: any, res:Response)=>{

    const body = req.body;

    body.usuario = req.usuario.\_id;

    Post.create( body ).then( async postDB => {

        //populate es para poblar la informacion del usuario deacuerdo a todos sus posteos

        await postDB.populate('usuario', '-password');

        res.json({

            ok: true,

            post: postDB

        })

    }).catch( err =>{

        res.json(err)

    });

})

//=======================================================================================================

// ruta buscar POST Paginados

postRoutes.get('/', async (req: any, res: Response)=> {

let pagina = Number(req.query.pagina) || 1;

let skip = pagina -1;

skip = skip \* 10;

const posts = await Post.find()

.sort({ \_id: -1})

.skip( skip)

.limit(10)

.populate('usuario', '-password')

.exec();

    res.json({

        pagina,

        ok: true,

        posts

    });

})

//ruta servicio para subir archivos

postRoutes.post( '/upload', [verificaToken], (req:any, res:Response)=>{

    if ( !req.files){

        return res.status(400).json({

            ok: false,

            mensaje: 'No se subió ninguna imagen'

        });

    }

    const file: FileUpload = req.files.image;

    if(!file){

        return res.status(400).json({

            ok: false,

            mensaje: 'No se subió ninguna archivo imagen'

        });

    }

    if(!file.mimetype.includes('image')){

        return res.status(400).json({

            ok: false,

            mensaje: 'Lo que subió no es una imagen'

        });

    }

 // llamar metodo de la instancia filesystem

 fileSystem.guardarImagenTemporal( file,  req.usuario.\_id);

    res.json({

        ok: true,

        file: file.mimetype

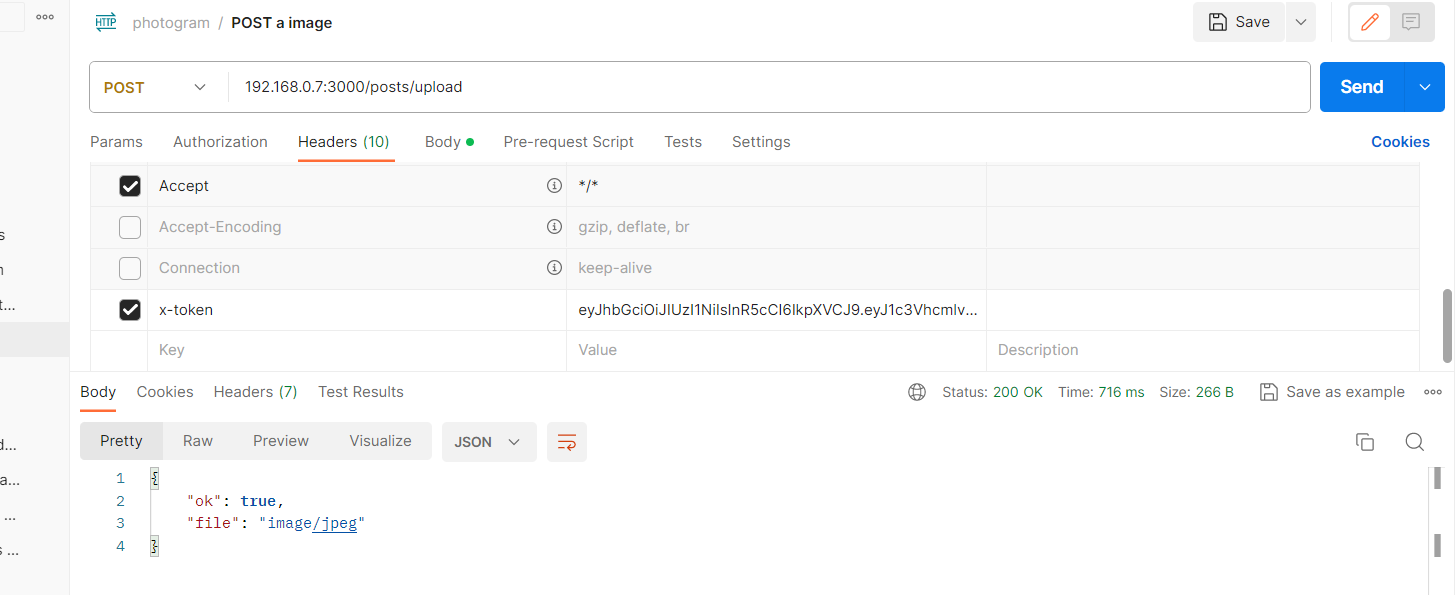
    });

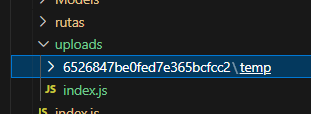
})

export default postRoutes;

al probar en postman enviando una imagen

me crea un diretorio llamado temporal y guarda el archivo





230 generar un nombre único a la imagen

En la clase file.system.ts creamos el método

export default class FileSystem {

    constructor(){};

    guardarImagenTemporal( file: FileUpload, userId: string){

        const path = this.crearCarpetaUsuario( userId);

    }

    //metodo para generar nombre

    private generarNombreUnico( nombreOriginal: string){

        // splinte ejje array bueno aqui separo el nombre y estencion del image

        const nombreArr = nombreOriginal.split('.');

        const extension = nombreArr[ nombreArr.length -1 ];

//generar id unico constante

        const idUnico= uniqid();

    }

Luego instalamos esta dependencia para generar id únicos

Npm install uniqid

Lugo lo importamos para utilizarlo, va necesitar el tipado

Por el mometo queda así y lo probamos

import { FileUpload } from '../interfaces/file-upload';

import uniqid from 'uniqid';

//propio de node para las rutas de los archivos

import path from 'path';

//verificar si la carpeta< existe

import fs from 'fs';

export default class FileSystem {

    constructor(){};

    guardarImagenTemporal( file: FileUpload, userId: string){

        // crear carpetas

        const path = this.crearCarpetaUsuario( userId);

        // nombre de la carpeta

        const nombreArchivo = this.generarNombreUnico( file.name);

        console.log( file.name );

        console.log( nombreArchivo );

    }

    //metodo para generar nombre

    private generarNombreUnico( nombreOriginal: string){

        // splinte ejje array bueno aqui separo el nombre y estencion del image

        const nombreArr = nombreOriginal.split('.');

        const extension = nombreArr[ nombreArr.length -1 ];

        //generar id unico constante

        const idUnico= uniqid();

        return `${idUnico}.${ extension}`

    }

    private crearCarpetaUsuario( userId: string){

        const pathUser = path.resolve( \_\_dirname, '../uploads/', userId);

        // cargar imagen temporal para luego hacer el push

        const pathUserTemp = pathUser + '/temp';

        console.log(pathUser);

        //verificar si existe

        const existe= fs.existsSync( pathUser);

        //si no existe crea el directorio

        if(!existe){

            fs.mkdirSync( pathUser);

            fs.mkdirSync( pathUserTemp);

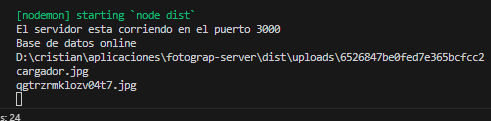
        }

//retornar las imagenes temporales

        return pathUserTemp

    }

    }



231 mover el archivo físico a la carpeta que hemos creados

Interfaces file add this

export interface FileUpload {

    name: string;

    data: any;

    encoding: string;

    tempFilePath: string;

    truncated: boolean;

    mimetype: string;

    mv: Function;

}

Then en en file-system.ts creamos una proimesa

export default class FileSystem {

    constructor(){};

    guardarImagenTemporal( file: FileUpload, userId: string){

        // ejecutar este metodo con promesa

        return new Promise( (resolve:any, reject:any)=>{

         // crear carpetas

         const path = this.crearCarpetaUsuario( userId);

         // nombre de la carpeta

         const nombreArchivo = this.generarNombreUnico( file.name);

         //mover el archivo del temp a nuestra carpeta

         file.mv( `${path}/${nombreArchivo}`, (err:any)=>{

            if(err){

                reject(err);

            } else {

                resolve();

            }

         });

         // console.log( file.name );

       //  console.log( nombreArchivo );

        })

    }

Y en post.ts pasamos un awit y async para que se ejecute estos datos antes de confirmar registro

//ruta servicio para subir archivos

postRoutes.post( '/upload', [verificaToken], async (req:any, res:Response)=>{

    if ( !req.files){

        return res.status(400).json({

            ok: false,

            mensaje: 'No se subió ninguna imagen'

        });

    }

    const file: FileUpload = req.files.image;

    if(!file){

        return res.status(400).json({

            ok: false,

            mensaje: 'No se subió ninguna archivo imagen'

        });

    }

    if(!file.mimetype.includes('image')){

        return res.status(400).json({

            ok: false,

            mensaje: 'Lo que subió no es una imagen'

        });

    }

 // llamar metodo de la instancia filesystem

 await fileSystem.guardarImagenTemporal( file,  req.usuario.\_id);

    res.json({

        ok: true,

        file: file.mimetype

    });

})

export default postRoutes;

si en caso tal no sube la data de la imagen agregar en el index.ts fileupload this

// FileUpload

server.app.use( fileUpload({useTempFiles:true}));

// fin FileUpload

232 Mover imágenes del Temp personalizado a la carpeta Post

file-system-.ts

agrego el nuevo método y hasta ahora estaría así el código

import { FileUpload } from '../interfaces/file-upload';

import uniqid from 'uniqid';

//propio de node para las rutas de los archivos

import path from 'path';

//verificar si la carpeta< existe

import fs from 'fs';

export default class FileSystem {

    constructor(){};

    guardarImagenTemporal( file: FileUpload, userId: string){

        // ejecutar este metodo con promesa

        return new Promise( (resolve:any, reject:any)=>{

         // crear carpetas

         const path = this.crearCarpetaUsuario( userId);

         // nombre de la carpeta

         const nombreArchivo = this.generarNombreUnico( file.name);

         //mover el archivo del temp a nuestra carpeta

         file.mv( `${path}/${nombreArchivo}`, (err:any)=>{

            if(err){

                reject(err);

            } else {

                resolve();

            }

         });

         // console.log( file.name );

       //  console.log( nombreArchivo );

        })

    }

    //metodo para generar nombre

    private generarNombreUnico( nombreOriginal: string){

        // splinte ejje array bueno aqui separo el nombre y estencion del image

        const nombreArr = nombreOriginal.split('.');

        const extension = nombreArr[ nombreArr.length -1 ];

        //generar id unico constante

        const idUnico= uniqid();

        return `${idUnico}.${ extension}`

    }

    private crearCarpetaUsuario( userId: string){

        const pathUser = path.resolve( \_\_dirname, '../uploads/', userId);

        // cargar imagen temporal para luego hacer el push

        const pathUserTemp = pathUser + '/temp';

        console.log(pathUser);

        //verificar si existe

        const existe= fs.existsSync( pathUser);

        //si no existe crea el directorio

        if(!existe){

            fs.mkdirSync( pathUser);

            fs.mkdirSync( pathUserTemp);

        }

//retornar las imagenes temporales

        return pathUserTemp

    }

    //pasar imagenes de temporal a carpeta post

    imagenesDeTempHaciaPost( userId: string){

        const pathTemp = path.resolve( \_\_dirname, '../uploads/', userId, 'temp');

        const pathPost = path.resolve( \_\_dirname, '../uploads/', userId, 'posts');

        if ( !fs.existsSync(pathTemp)){

            return [];

        }

        if ( !fs.existsSync(pathPost)){

            fs.mkdirSync(pathPost);

        }

        const imagenesTemp = this.obtenerImagenesEnTemp( userId);

        imagenesTemp.forEach( imagen => {

            fs.renameSync( `${pathTemp}/${imagen}`,`${pathPost}/${imagen}`)

        });

        return imagenesTemp;

    }

    private obtenerImagenesEnTemp( userId: string){

        const pathTemp = path.resolve( \_\_dirname, '../uploads/', userId, 'temp');

        return fs.readdirSync( pathTemp) || [];

    }

    }

En el post llamo ese nuevo método y iría el código de esta forma

import { Router, Response } from 'express';

import { Post } from '../Models/post.model';

import { verificaToken } from '../middlewares/autenticacion';

import { FileUpload } from '../interfaces/file-upload';

import FileSystem from '../class/file-system';

const postRoutes = Router();

   //crear instancia de guardar imagen temporal

   const fileSystem = new FileSystem();

//ruta  agregar un post con verificar token de usuario y almacenar sus posteos==========================

postRoutes.post('/', [verificaToken], (req: any, res:Response)=>{

    const body = req.body;

    body.usuario = req.usuario.\_id;

    // constante con metodo que envia las imagenes temporales a la carpeta de post

    const imagenes= fileSystem.imagenesDeTempHaciaPost( req.usuario.\_id);

    //se agregan al body del modelo agregar s al modelo

    body.imgs = imagenes;

    Post.create( body ).then( async postDB => {

        //populate es para poblar la informacion del usuario deacuerdo a todos sus posteos

        await postDB.populate('usuario', '-password');

        res.json({

            ok: true,

            post: postDB

        })

    }).catch( err =>{

        res.json(err)

    });

})

//=======================================================================================================

// ruta buscar POST Paginados

postRoutes.get('/', async (req: any, res: Response)=> {

let pagina = Number(req.query.pagina) || 1;

let skip = pagina -1;

skip = skip \* 10;

const posts = await Post.find()

.sort({ \_id: -1})

.skip( skip)

.limit(10)

.populate('usuario', '-password')

.exec();

    res.json({

        pagina,

        ok: true,

        posts

    });

})

//ruta servicio para subir archivos

postRoutes.post( '/upload', [verificaToken], async (req:any, res:Response)=>{

    if ( !req.files){

        return res.status(400).json({

            ok: false,

            mensaje: 'No se subió ninguna imagen'

        });

    }

    const file: FileUpload = req.files.image;

    if(!file){

        return res.status(400).json({

            ok: false,

            mensaje: 'No se subió ninguna archivo imagen'

        });

    }

    if(!file.mimetype.includes('image')){

        return res.status(400).json({

            ok: false,

            mensaje: 'Lo que subió no es una imagen'

        });

    }

 // llamar metodo de la instancia filesystem

 await fileSystem.guardarImagenTemporal( file,  req.usuario.\_id);

    res.json({

        ok: true,

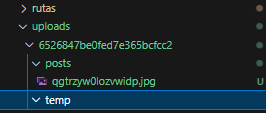
        file: file.mimetype

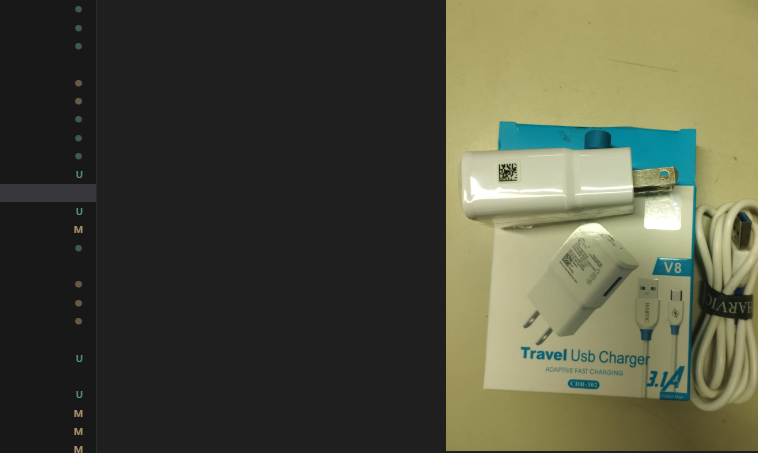
    });

})

export default postRoutes;

al probar en postman las imágenes temporales se iran a la carpeta post y se guardan en la base de datos

listo



233 Servicio para mostrar una imagen por url

Crear método ruta en file.system.ts

   private obtenerImagenesEnTemp( userId: string){

        const pathTemp = path.resolve( \_\_dirname, '../uploads/', userId, 'temp');

        return fs.readdirSync( pathTemp) || [];

    }

           //mostrar foto en la app

    getFotoUrl( userId: string, img : string){

        // path Posts

        const pathFoto = path.resolve( \_\_dirname, '../uploads/', userId, 'posts', img);

        //si la imagen no existe

    const existe = fs.existsSync( pathFoto)

        if ( !existe){

            return path.resolve( \_\_dirname, '../assets/400x250.jpg');

        }

        return pathFoto;

    }

Luego en post.ts agregamos una ruta utilizando ese método de file\_system

 // llamar metodo de la instancia filesystem

 await fileSystem.guardarImagenTemporal( file,  req.usuario.\_id);

    res.json({

        ok: true,

        file: file.mimetype

    });

});

// ruta para buscar imagen desde DB

postRoutes.get('/imagen/:userid:img', (req:any, res:Response) =>{

    const userId = req.params.userid;

    const img = req.params.img;

    const pathFoto = fileSystem.getFotoUrl( userId, img);

    res.sendFile( pathFoto);

})

Debo de revisar esto por que la imagen no me carga pero la de error si

234 Retomar información del usuario por Token

En usuarios.ts rutas agrego esta ruta

        // validar usuario con token

        UserRoutes.get('/', [verificaToken], ( req: any, res: Response)=>{

            const usuario = req.usuario;

            res.json({

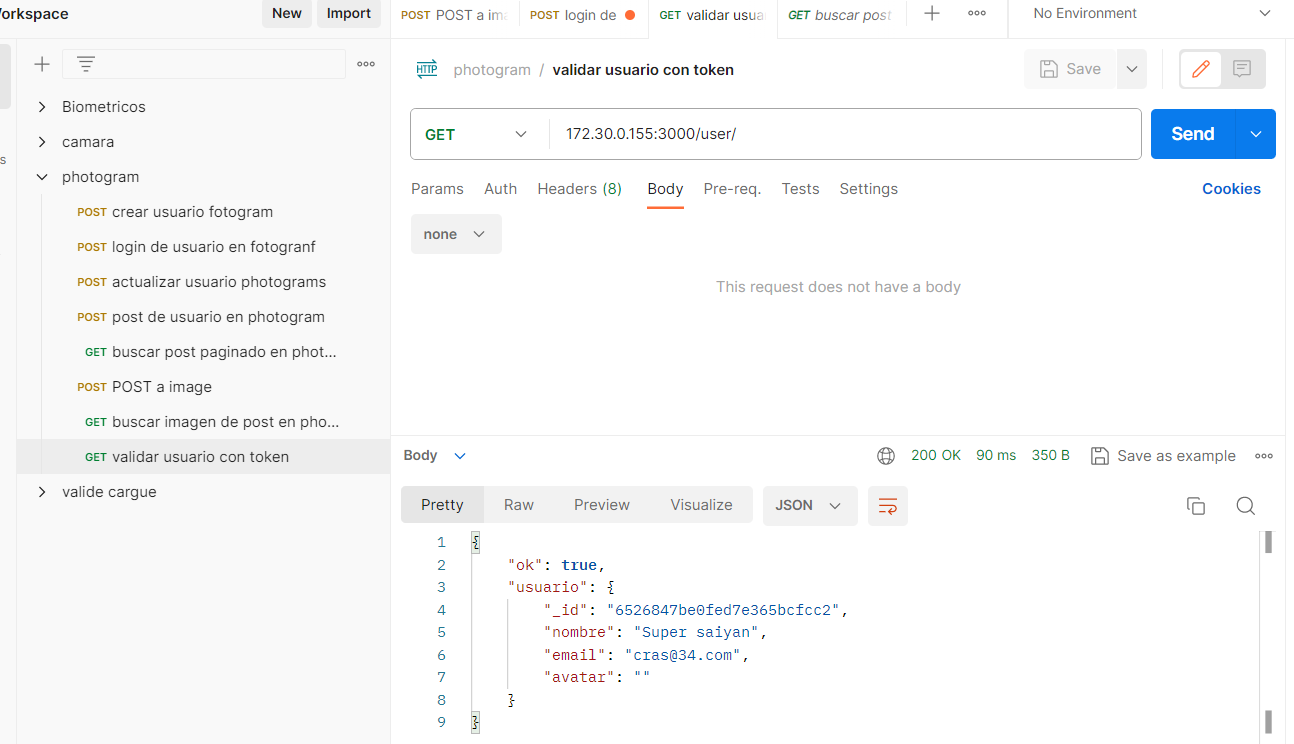
                ok: true,

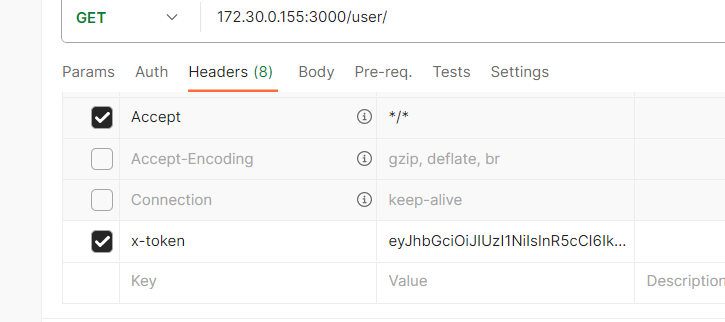
                usuario

            });

        })

Luego la pruebo en posman





En el index debemos de hacer una modificación con el cors para que las peticiones o consumo de la apis se reflejen desde cualquier fronter

import mongoose from "mongoose";

import Server from "./class/server";

import UserRoutes from "./rutas/usuarios";

import postRoutes from "./rutas/post";

import bodyParser from "body-parser";

import fileUpload from "express-fileupload";

import cors from "cors";

//A1 constante del server para usar la extancia de express servidor

const server = new Server;

//===============================fin A1======================

//A4 conectar DB esto lo debo de poner aparte

mongoose.connect('mongodb://127.0.0.1:27017/fotosgram')

.then(()=> {

    console.log('Base de datos online');

}).catch((err)=>{

 if(err) throw err;

});

//middleware with bodyparser pasa estos parametros primero estos middleware deben de ir de primero ya que cuando pase la ruta user deben de existir los datos para el registro

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

server.app.use(bodyParser.json());

//end middleware

//===================== A2 primer uso constante levantar el servidor======================

server.start(()=>{

    console.log(`El servidor esta corriendo en el puerto ${server.port}`);

})

//=====================fin A2 ======================

// FileUpload

server.app.use( fileUpload());

// fin FileUpload

// user rutas

server.app.use('/user', UserRoutes);

// fin rutas

// post rutas

server.app.use('/posts', postRoutes);

// fin post rutas

// user Cors

server.app.use(cors({ origin: true, credentials: true}));

//=====================fin route ======================

Nota si esta api se instala en otro servidor toca agregar el puerto que esta utilizando en el firewall