CATEGORIAS Y PRODUCTOS NODEJS HERRERA

CRUD Y Rutas de categorías

- El endpoint será http://localhost:8080/api/categorías
- Creo el router y los controladores
- categoria.routes.js

```
import {Router} from 'express'
import categoria from '../controllers/categoria.controller.js'


const router = Router()

router.get('/', categoria.getCategorias) //falta implementar el controlador

export default router
```

- categoria.controller.js

```
const getCategorias = (req,res)=>{
    res.json({
        msg: "categorias ok"
    })
}

export default{
    getCategorias
}
```

- En el server añado el path, importo el router y lo uso

```
import express from 'express'
import cors from 'cors'
import userRouter from '../routes/user.routes.js'
import dbConnection from '../database/config.js'
import authRouter from '../routes/auth.router.js'
import categoriaRouter from '../routes/categoria.routes.js'

export class Server {

    constructor(){
        this.app = express()
        this.port = process.env.PORT
        this.usuariosPath = '/api/usuarios'
        this.authPath= '/api/auth'
```

```
        //path categorias
        this.categoriasPath = '/api/categorias'

        //conexion a la DB
        this.conectarDB()

        //Middlewares ( en el constructor van a ejecutarse al levantar el servidor
)
        this.middlewares()

       //Rutas
        this.routes()
    }

    async conectarDB(){
        await dbConnection()
    }

    middlewares(){
        this.app.use(express.static('public')) //Esto servirá lo que haya en la
carpeta public en '/'
        this.app.use(express.urlencoded({extended: false})) //parseo el body
        this.app.use(express.json())
        this.app.use(cors())
    }

    routes(){

        this.app.use(this.usuariosPath, userRouter)
        this.app.use(this.authPath, authRouter)
        //uso el path con el router de categorias
        this.app.use(this.categoriasPath, categoriaRouter )
    }

    listen(){
        this.app.listen(this.port, ()=>{
            console.log(`Server corriendo en puerto ${this.port}`)
        })
    }
}
```

- En el categoria.routes voy a usar también el check y el validarCampos
- Faltan el resto de rutas y controladores (POST, PUT, DELETE) y otro GET para solo una categoría y no todas
- Los creo
- categoria.routes.js

```
import {Router} from 'express'
import categoria from '../controllers/categoria.controller.js'
import {check} from 'express-validator'
import { validarCampos } from '../middlewares/validar-campos.js'
```

```
const router = Router()

router.get('/', categoria.getCategorias)
router.get('/:id', categoria.getCategoria)
router.post('/', categoria.addCategorias)
router.put('/:id', categoria.updateCategorias)
router.delete('/:id', categoria.deleteCategorias)


export default router
```

- categoria.controller.js

```
const getCategorias = (req,res)=>{
    res.json({
        msg: "getCategorias ok"
    })
}

const getCategoria = (req,res)=>{
    res.json({
        msg: "getCategoria ok"
    })
}

const addCategorias = (req,res)=>{
    res.json({
        msg: "addCategorias ok"
    })
}

const updateCategorias = (req,res)=>{
    res.json({
        msg: "updateCategorias ok"
    })
}

const deleteCategorias = (req,res)=>{
    res.json({
        msg: "deleteCategorias ok"
    })
}

export default{
    getCategorias,
    addCategorias,
    updateCategorias,
```

```
    deleteCategorias,
    getCategoria

}
```

Modelo Categoria

- Creo el modelo
- Hago referencia a la colección Usuario

```
import {Schema, model} from 'mongoose'

const categoriaSchema = Schema({
    nombre:{
        type: String,
        required: true,
        unique: true
    },
    estado:{
        type: Boolean,
        default: true,
        required: true
    },
    usuario: {
    type: Schema.Types.ObjectId,
    ref: 'Usuario',
    required: true
    },


})

const Categoria = model('Categoria', categoriaSchema)
export default Categoria
```

Crear una categoría

- Crear categoría debe de ser privado con lo cual debe de tener un token válido
- Entonces deo validar el JWT en la ruta
- **NOTA:** Para extraer el token hacer un login, copiarlo de la consola e introducirlo en headers sin las comillas (y sin espacios!) con el nombre de x-token
- Voy a categorias.routes

```
router.post('/',[validarJWT], categoria.addCategorias)
```

- Para pasar la validación ha de ser un token válido
- Ahora paso a verificar las propiedades. tanto el nombre, como el estado y el usuario son obligatorios

```
router.post('/',[validarJWT,
    check('nombre', "El nombre es obligatorio").not().isEmpty(),
    validarCampos], categoria.addCategorias)
```

- Ahora debo introducir un nombre en el body de la petición para pasar la validación
- Pasemos al controlador
- Paso el nombre de la categoría a myúsculas
- Debo comprobar si existe la categoría
- Debo evitar que se me pase el estado excluyéndolo del body, con lo cual me preparo la data que voy a enviar a la nueva instacia de Categoría
- El usuario lo extraigo del req.usuario._id que viene de la validación del token y así es como lo graba mongo

```
const addCategoria = async (req,res)=>{
    const nombre = req.body.nombre.toUpperCase() //quiero almacenar las categorías
en mayúsculas
    //debo comprobar si existe la categoria
    const categoriaDB = await Categoria.findOne({nombre})

    if(categoriaDB){
        return res.status(400).json({msg: "La categoría ya existe"})
    }

    const data = {
        nombre,
        usuario : req.usuario._id  //_id porque así es como Mongo lo está grabando
( aunque para mostrarlo use id por la modificación que hice)
    }

    const categoria = new Categoria(data)
    await categoria.save()

    res.status(201).json({
        categoria
    })
}
```

CRUD CATEGORIAS

- Listar categorías con paginación

```
const getCategorias = async (req,res)=>{
    const {desde= 0, limite=5} = req.query
```

```
    const categorias = await Categoria.find()
                                .skip(desde)
                                .limit(+limite)
    const total = await Categoria.countDocuments()
    res.json({
        total,
        categorias
    })
}
```

- Puedo optimizarlo usando Promise.all
- Uso el populate para que me muestre la info del usuario, en este caso solo el nombre
- Filtro por el estado en true

```
const getCategorias = async (req,res)=>{
    const {desde= 0, limite=5} = req.query
    const query = { estado: true}

    const [total, categorias] = await Promise.all([
        Categoria.countDocuments(query),
        Categoria.find(query)
            .populate('usuario', 'nombre')
            .skip(desde)
            .limit(+limite)
    ])


    res.json({
        total,
        categorias
    })
}
```

- En el modelo de usuario se hizo un método para quitar el __v, el estado...hago lo mismo con categoria
- categoria.js

```
categoriaSchema.methods.toJSON = function(){
    const {__v, estado, ...categoria} = this.toObject()
    return categoria
}
```

- Listar categoria por id. Uso el populate para mostrar la info del usuario
- Debo verificar que es un id válido, y como voy a hacerlo en otros tres endpoints me creo un custom validator
- categoria.controller.js

```
const getCategoria = async (req,res)=>{
    const {id} = req.params

    const categoria = await Categoria.findById(id).populate('usuario', 'nombre')

    res.json({
        categoria
    })
}
```

- Ahora debo validar el id porque si no va a reventar mi app
- categoria.routes

```
router.get('/:id',[
    check('id', 'No es un id de mongo válido').isMongoId(), validarCampos
],categoria.getCategoria)
```

- Si le paso algo parecido a un MongoId me devuelve null, no estoy validando que exista la categoria
- Puedo hacerlo en el controlador, pero como voy a repetirlo en dos endpoints más me creo un custom middleware
- Creo el método en db-validators.js

```
export const existeCategoria = async(id)=>{
    const categoria = await Categoria.findById(id)

    if(!categoria) throw new Error(`El id ${id} no existe`)
}
```

- Lo añado a la ruta
- Uso bail() para que no me de el error de Cast. Bail sirve para que deje de ejecutar validaciones si una de las validaciones ha fallado
- Así puedes validar primero que el ID sea válido y na vez hecho esto consultar si existe ese ID

```
router.get('/:id',[
    check('id', 'No es un id de mongo
válido').isMongoId().bail().custom(existeCategoria), validarCampos
],categoria.getCategoria)
```

- Actualizar categoría. No debe de existir la categoría nueva que creo al actualizar (TODO:añadir)
- Uso las mismas validaciones en la ruta. Voy a por el controlador primero
- Debo evitar que se pueda cambiar en la req usuario y estado

```

```

```
const updateCategoria = async (req,res)=>{
    const {id} = req.params

    const {estado, usuario, ...data}= req.body

    data.nombre = data.nombre.toUpperCase() //me aseguro de guardar el nombre en
mayúsculas

    data.usuario = req.usuario._id //me aseguro de que el usuario sea el que viene
en req.usuario, el dueño del token

    const categoria= await Categoria.findByIdAndUpdate(id, data, {new: true})

    res.json({
        categoria
    })
}
```

- En la ruta hago las validaciones pertinentes

```
router.put('/:id',[
    check('nombre', "El nombre es obligatorio").not().isEmpty(),
    check('id', 'No es un id de mongo
válido').isMongoId().bail().custom(existeCategoria), validarJWT, validarCampos
], categoria.updateCategoria)
```

- Borrar categoría. Un borrado que será pasar el estado a false. tengo que verificar el id, etc
- La eliminación tiene la condición de ser admin role
- categoria.controller.js

```
const deleteCategoria = async (req,res)=>{

    const {id} = req.params

    await Categoria.findByIdAndUpdate(id, {estado: false})

    res.json({
        msg: "categoría borrada!"
    })
}
```

- uso el middleware esAdminRole en las rutas para validar que sea admin. Antes valido el JWT con el middleware validarJWT
- validarJWT

```
import jwt from 'jsonwebtoken'
import Usuario from '../models/usuario.js'

const validarJWT = async (req, res, next) =>{
    const token = req.header('x-token')

    if(!token)
     return res.status(401).json({msg: "No hay token en la petición"})

    try {
        //verifico el jwt y extraigo el uid
        const {uid} = jwt.verify(token, process.env.SECRET_KEY)

        //creo una propiedad nueva dentro de la request para pasar el uid
        const usuario = await Usuario.findById(uid)

        //verificar que venga usuario por el uid
        if(!usuario) return res.status(401).json({msg:"Token no válido - usuario
no existe en BD"})

        //verificar si el usuario tiene estado en true
        if(!usuario.estado) return res.status(401).json({msg:"Token no válido -
usuario con estado false"})

       req.usuario = usuario
        next()

    } catch (error) {
        console.log(error)
        return res.status(401).json({msg:"Token no válido"})
    }


}

export default validarJWT
```

- esAdminRole

```
export const esAdminRole = (req, res, next) =>{

    if(!req.usuario) return res.status(500).json({msg: "Se quiere validar el role
sin obtener el token"})

    const {rol, nombre} = req.usuario

    if(rol !== 'ADMIN_ROLE') return res.status(401).json({msg: `${nombre} no es
administrador`})
    next()
}
```

- categoria.routes.js

```
router.delete('/:id',[ validarJWT, esAdminRole,
    check('id', 'No es un id de mongo
válido').isMongoId().bail().custom(existeCategoria), validarCampos
], categoria.deleteCategoria)
```

Modelo de Productos y Rutas

- Creo el el path en el server, uso el router
- server

```
import express from 'express'
import cors from 'cors'
import userRouter from '../routes/user.routes.js'
import dbConnection from '../database/config.js'
import authRouter from '../routes/auth.router.js'
import categoriaRouter from '../routes/categoria.routes.js'
import productosRouter from '../routes/producto.router.js'

export class Server {

    constructor(){
        this.app = express()
        this.port = process.env.PORT
        this.usuariosPath = '/api/usuarios'
        this.authPath= '/api/auth'
        this.categoriasPath = '/api/categorias'
        this.productosPath = '/api/productos' //PATH PRODUCTOS!!

        //conexion a la DB
        this.conectarDB()

        //Middlewares ( en el constructor van a ejecutarse al levantar el servidor
)
        this.middlewares()

       //Rutas
        this.routes()
    }

    async conectarDB(){
        await dbConnection()
    }

    middlewares(){
        this.app.use(express.static('public')) //Esto servirá lo que haya en la
carpeta public en '/'
```

```
        this.app.use(express.urlencoded({extended: false})) //parseo el body
        this.app.use(express.json())
        this.app.use(cors())
    }

    routes(){

        this.app.use(this.usuariosPath, userRouter)
        this.app.use(this.authPath, authRouter)
        this.app.use(this.categoriasPath, categoriaRouter )
        this.app.use(this.productosPath, productosRouter) //PATH PRODUCTOS
ROUTER!!
    }

    listen(){
        this.app.listen(this.port, ()=>{
            console.log(`Server corriendo en puerto ${this.port}`)
        })
    }
}
```

- Creo el esqueleto del router y controladores
- producto.router

```
import {Router} from 'express'
import productoController from '../controllers/producto.controller.js'

const app = Router()

app.get('/', productoController.getProductos)
app.get('/:id', productoController.getProducto)
app.post('/', productoController.addProducto)
app.put('/:id', productoController.updateProducto)
app.delete('/:id', productoController.deleteProducto)

export default app
```

-producto.controller

```
const getProductos = (req,res)=>{

    res.json({
        msg: "ok getProductos!"
    })
}

const getProducto = (req,res)=>{

    res.json({
```

```
        msg: "ok getProducto!"
    })
}

const addProducto = (req,res)=>{

    res.json({
        msg: "ok addProductos!"
    })
}

const updateProducto = (req,res)=>{

    res.json({
        msg: "ok updateProductos!"
    })
}

const deleteProducto = (req,res)=>{

    res.json({
        msg: "ok deleteProductos!"
    })
}

export default{
    getProductos,
    getProducto,
    addProducto,
    updateProducto,
    deleteProducto
}
```

- Creo el modelo de producto
- producto.js

```
import {Schema, model} from 'mongoose'

const productoSchema = Schema({

    nombre: {
        type: String,
        required: true,
        unique: true
    },
    estado:{
        type: Boolean,
        default: true,
        required: true
    },
    usuario:{
```

```
        type: Schema.Types.ObjectId,
        ref: 'Usuario',
        required: true
    },
    precio:{
        type: Number,
        default: 0,
    },
    categoria:{
        type: Schema.Types.ObjectId,
        ref: 'Categoria',
        required: true
    },
    descripcion: {type:String},
    disponible: {type: Boolean, default: true}
})

productoSchema.methods.toJSON = function(){
    const {__v, estado, ...producto} = this.toObject()
    return producto
}

const Producto = model('Producto', productoSchema)

export default Producto
```

- Ahora toca implementar la lógica en lso controladores
- Esta bien empezar por la creación de Producto así luego puedo listar
- Los controladores vana a aser muy similares a los de categorías
- producto.router.js

```
import {Router} from 'express'
import productoController from '../controllers/producto.controller.js'
import {validarCampos} from '../middlewares/validar-campos.js'
import validarJWT from '../middlewares/validar-jwt.js'
import { check } from 'express-validator'
import { existeCategoria, existeProducto, userExistsById } from '../helpers/db-
validators.js'
import { esAdminRole } from '../middlewares/validar-roles.js'

const app = Router()

app.get('/', productoController.getProductos)
app.get('/:id',[
    check('id', 'No es un id válido').isMongoId().bail().custom(existeProducto),
    validarCampos
], productoController.getProducto)

app.post('/', [
    validarJWT,
    check('nombre', 'El nombre es obligatorio').not().isEmpty(),
```

```
    check('categoria', "No existe la
categoría").isMongoId().bail().custom(existeCategoria),
    validarCampos
] ,productoController.addProducto)

app.put('/:id', [
    validarJWT,
    check('id', 'No es un id válido').isMongoId().bail().custom(existeProducto),
    validarCampos
], productoController.updateProducto)

app.delete('/:id',[
    validarJWT, esAdminRole,
    check('id', 'No es un id válido').isMongoId().bail().custom(existeProducto),
    validarCampos
],
 productoController.deleteProducto)

export default app
```

- producto.controller.js

```
import Producto from '../models/producto.js'

const getProductos = async (req,res)=>{

    const {desde=0, limite=5} = req.query

    const [total, productos] = await Promise.all([
        await Producto.countDocuments({estado: true}),
        await Producto.find({estado: true})
            .populate('usuario', 'nombre')
            .populate('categoria', 'nombre')
            .skip(desde)
            .limit(limite)
    ])

    res.json({
        total,
        productos
    })
}

const getProducto = async (req,res)=>{

    const {id} = req.params

    const producto = await Producto.findById(id)
                                    .populate('usuario', 'nombre')
                                    .populate('categoria', 'nombre')

    if(!producto) throw new Error(`El producto con id ${id} no existe`)
```

```
    res.json({
        producto
    })
}

const addProducto =async (req,res)=>{

    const {estado, usuario, ...body} = req.body
    const nombre = body.nombre.toUpperCase()
    const productoDB = await Producto.findOne({nombre})

    if(productoDB){
        throw new Error(`El producto con nombre ${productoDB.nombre} ya existe`)
    }



    const data = {
        ...body,
        nombre,
        usuario: req.usuario._id
    }

    const producto = new Producto(data)
    await producto.save()

    return res.status(200).json({
        producto
    })
}

const updateProducto =  async (req,res)=>{
    const {id} = req.params

    const {estado, usuario, ...data} = req.body

    if(data.nombre){
        data.nombre = data.nombre.toUpperCase()

    }

    data.usuario = req.usuario._id


    const producto = await Producto.findByIdAndUpdate(id, data, {new: true} )

    res.json({
        producto
    })
}

const deleteProducto = async (req,res)=>{
```

```
    const id = req.params.id
    const productoBorrado = await Producto.findByIdAndUpdate(id, {estado: false},
{new: true} )

    res.json({
        productoBorrado
    })

}

export default{
    getProductos,
    getProducto,
    addProducto,
    updateProducto,
    deleteProducto
}
```

Ruta para realizar búsquedas

- Creo el archivo de rutas buscar.routes.js y el controlador buscar.controller.js

```
//routes
import {Router} from 'express'
import buscarController from '../controllers/buscar.controller.js'

 const router = Router()

 router.get('/:coleccion/:termino', buscarController.buscar)

 export default router
```

- buscar.controller.js

```
const buscar = (req,res)=>{
       res.json({
        msg: "buscar ok"
    })
}

export default{
    buscar
}
```

- Añado el path y el router al server

```
import express from 'express'
import cors from 'cors'
import userRouter from '../routes/user.routes.js'
import dbConnection from '../database/config.js'
import authRouter from '../routes/auth.router.js'
import categoriaRouter from '../routes/categoria.routes.js'
import productosRouter from '../routes/producto.router.js'
import buscarRouter from '../routes/buscar.routes.js'
export class Server {

    constructor(){
        this.app = express()
        this.port = process.env.PORT
        this.usuariosPath = '/api/usuarios'
        this.authPath= '/api/auth'
        this.categoriasPath = '/api/categorias'
        this.productosPath = '/api/productos'
        this.buscarPath = "/api/buscar"

        //conexion a la DB
        this.conectarDB()

        //Middlewares ( en el constructor van a ejecutarse al levantar el servidor
)
        this.middlewares()

       //Rutas
        this.routes()
    }

    async conectarDB(){
        await dbConnection()
    }

    middlewares(){
        this.app.use(express.static('public')) //Esto servirá lo que haya en la
carpeta public en '/'
        this.app.use(express.urlencoded({extended: false})) //parseo el body
        this.app.use(express.json())
        this.app.use(cors())
    }

    routes(){

        this.app.use(this.usuariosPath, userRouter)
        this.app.use(this.authPath, authRouter)
        this.app.use(this.categoriasPath, categoriaRouter )
        this.app.use(this.productosPath, productosRouter)
        this.app.use(this.buscarPath, buscarRouter)
    }

    listen(){
        this.app.listen(this.port, ()=>{
```

```
            console.log(`Server corriendo en puerto ${this.port}`)
        })
    }
}
```

- Normalmente las búsquedas son get y los argumentos se pasan por el url

---

Busquedas en base de datos

- Hago un array con las colecciones permitidas y lo uso para validar

```
const buscar = (req,res)=>{

    const {coleccion, termino} = req.params

    if(!coleccionesPermitidas.includes(coleccion)) return
res.status(400).json({msg:"La colección no está en la DB"})

    res.json({
        msg: "buscar ok"
    })
}
```

- Hago un switch dónde coloco las tres colecciones (roles lo voy a obviar, lo voy a usar para manejar un error)

```
import Usuario from "../models/usuario.js"
import Categoria from "../models/categoria.js"
import Producto from "../models/producto.js"
import mongoose from "mongoose"

const coleccionesPermitidas = [
    'usuarios',
    'categorias',
    'productos',
    'roles'
]

const buscarUsuarios = async(termino="", res)=>{
    //compruebo que sea un id válido
    const esMongoId= mongoose.isValidObjectId(termino) //TRUE

    if(esMongoId){
        const usuario = await Usuario.findById(termino)
        res.json({
            usuario
        })
    }
```

```
}

const buscar = (req,res)=>{

    //extraigo los parametros de la url
    const {coleccion, termino} = req.params

    //filtro si están een el enum
    if(!coleccionesPermitidas.includes(coleccion)) return
res.status(400).json({msg:"La colección no está en la DB"})


    switch (coleccion) {
        case 'usuarios':
            buscarUsuarios(termino, res)
            break;
        case 'categorias':

            break;
        case 'productos':

            break;
        default:
            res.status(500).json({
                msg:"Se me olvidó hacer esta búsqueda"
            })
            break;
    }


}

export default{
    buscar
}
```

- En la url debo colocar la colección y un id válido de un usuario

> http://localhost:8080/api/buscar/usuarios/6448fa3c5819512255909a86

- Si mando un id válido pero que no existe recibo un null
- Lo manejo con un ternario

```
const buscarUsuarios = async(termino="", res)=>{

    const esMongoId= mongoose.isValidObjectId(termino) //TRUE

    if(esMongoId){
        const usuario = await Usuario.findById(termino)
        return res.json({
            results : usuario ? [usuario]: []
        })
```

```
    }
}
```

- Las demás búsquedas van a ser iguales pero quiero implementar también las búsquedas por nombre

---

Buscar por otros argumentos

- Quiero poder buscar por nombre y por correo
- Es key sensitive y no quiero que sea tan escrito como para tener que escribir el nombre completo en la url
- Para ello uso una expresión regular
- Puedo buscar por nombre o por correo
- Debe de tener el estado en true

```
import Usuario from "../models/usuario.js"
import Categoria from "../models/categoria.js"
import Producto from "../models/producto.js"
import mongoose from "mongoose"

const coleccionesPermitidas = [
    'usuarios',
    'categorias',
    'productos',
    'roles'
]

const buscarUsuarios = async(termino="", res)=>{

    const esMongoId= mongoose.isValidObjectId(termino) //TRUE

    if(esMongoId){
        const usuario = await Usuario.findById(termino)
        return res.json({
            results : usuario ? [usuario]: []
        })
    }

    const regex = new RegExp(termino, 'i') //expresión regular, i de insensitive a
minúsculas y mayúsculas

    const usuarios = await Usuario.find({
        //$or es una propiedad de mongo
        $or: [
            {nombre: regex},
            {correo: regex}
        ],

        $and: [{estado:true}]
    })
    res.json({
```

```
            results: usuarios
        })

    }

    const buscar = (req,res)=>{

        const {coleccion, termino} = req.params

        if(!coleccionesPermitidas.includes(coleccion)) return
    res.status(400).json({msg:"La colección no está en la DB"})


        switch (coleccion) {
            case 'usuarios':
                buscarUsuarios(termino, res)
                break;
            case 'categorias':

                break;
            case 'productos':

                break;
            default:
                res.status(500).json({
                    msg:"Se me olvidó hacer esta búsqueda"
                })
                break;
        }


    }

    export default{
        buscar
    }
```

- Podría usar Usuario.count en lugar del find en el caso de que quisiera contar cuantas respuestas hay
- Hacer las demás búsquedas es practicamente los mismo

Buscar en otras colecciones

```
import Usuario from "../models/usuario.js"
import Categoria from "../models/categoria.js"
import Producto from "../models/producto.js"
import mongoose from "mongoose"

const coleccionesPermitidas = [
    'usuarios',
    'categorias',
```

```
    'productos',
    'roles'
]

const buscarUsuarios = async(termino="", res)=>{

    const esMongoId= mongoose.isValidObjectId(termino) //TRUE

    if(esMongoId){
        const usuario = await Usuario.findById(termino)
        return res.json({
            results : usuario ? [usuario]: []
        })
    }

    const regex = new RegExp(termino, 'i') //expresión regular, i de insensitive a
minúsculas y mayúsculas

    const usuarios = await Usuario.find({
        //$or es una propiedad de mongo
        $or: [
            {nombre: regex},
            {correo: regex}
        ],

        $and: [{estado:true}]
    })
    res.json({
        results: usuarios
    })

}

const buscar = (req,res)=>{

    const {coleccion, termino} = req.params

    if(!coleccionesPermitidas.includes(coleccion)) return
res.status(400).json({msg:"La colección no está en la DB"})


    switch (coleccion) {
        case 'usuarios':
            buscarUsuarios(termino, res)
            break;
        case 'categorias':
            buscarCategorias(termino, res)
            break;
        case 'productos':
            buscarProductos(termino, res)
            break;
        default:
            res.status(500).json({
                msg:"Se me olvidó hacer esta búsqueda"
```

```
            })
            break;
    }
}

const buscarCategorias = async(termino="", res)=>{
    const esMongoId= mongoose.isValidObjectId(termino) //TRUE

    if(esMongoId){
        const categoria = await Categoria.findById(termino)
        return res.json({
            results : categoria ? [categoria] : []
        })
    }

    const regex = new RegExp(termino, 'i') //expresión regular, i de insensitive a
minúsculas y mayúsculas

    const categorias = await Categoria.find({nombre: regex, estado: true})
    res.json({
        results: categorias
    })

}

const buscarProductos = async(termino="", res)=>{
    const esMongoId= mongoose.isValidObjectId(termino) //TRUE

    if(esMongoId){
        const producto = await Producto.findById(termino)
        return res.json({
            results : producto ? [producto]: []
        })
    }

    const regex = new RegExp(termino, 'i') //expresión regular, i de insensitive a
minúsculas y mayúsculas

    const productos = await Producto.find({nombre: regex, estado:true })
    res.json({
        results: productos
    })

}


export default{
    buscar
}
```