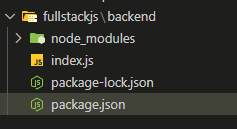
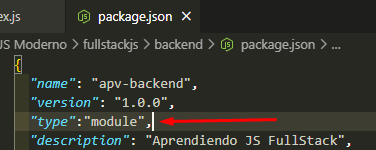
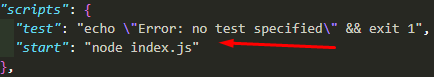
1. Crear la carpeta fullstackjs
2. Crear la carpeta backend
3. Iniciar proyecto de node npm init
4. Crear el index.js que será el punto de partida de Express, básicamente es la configuración para utilizar express como en el proyecto de 54-agencia de viajes donde el index controla la estructura
5. Descargar Express con npm i express
6. Configurar el package.json, agregar “type”: “module”



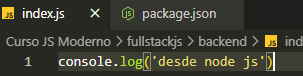


Para verificar que todo funciona hasta aquí, en scripts del archivo package.json agregamos *"start"*: "node index.js"

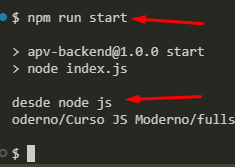
Le decimos que con node corra el index.js



EL index.js por ahora solo contiene un mensaje



Ejecutamos con npm run start, muestra el mensaje y se corta la comunicación, prodiamos tener solo node para estar corriendo cada vez que hagamos cambios y volviéndolo a ejecutar, para evitar esto y sea una conexión persistente requerimos una herramienta como nodemon.

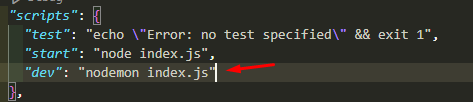


Nota: uno puede correr solo el node y desde ahí correr la aplicación, el tema es que node no sabe de los demás cambios de los archivos, se tendría que detener si es que no se corto el servicio de node y volverlo a correr con npm run start, esto es muy tedioso.

**Nodemon** es una herramienta de utilidad que ayuda a desarrollar aplicaciones basadas en Node.js al reiniciar automáticamente la aplicación del servidor cuando se detectan cambios en los archivos del proyecto. Es particularmente útil durante el desarrollo, ya que elimina la necesidad de reiniciar manualmente el servidor cada vez que se realizan cambios en el código.

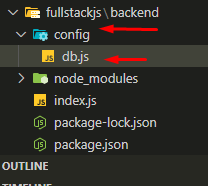
1. Instalamos nodemon como dependencia de desarrollo por que en producción no es necesario. npm i -D nodemon
2. Configuramos el package.json para desarrollo

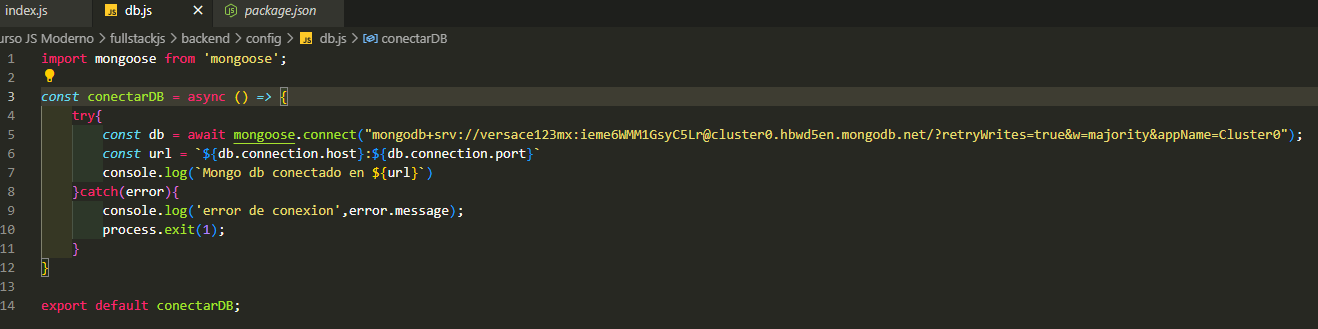
Se agredo dev:”nodemon index.js ”



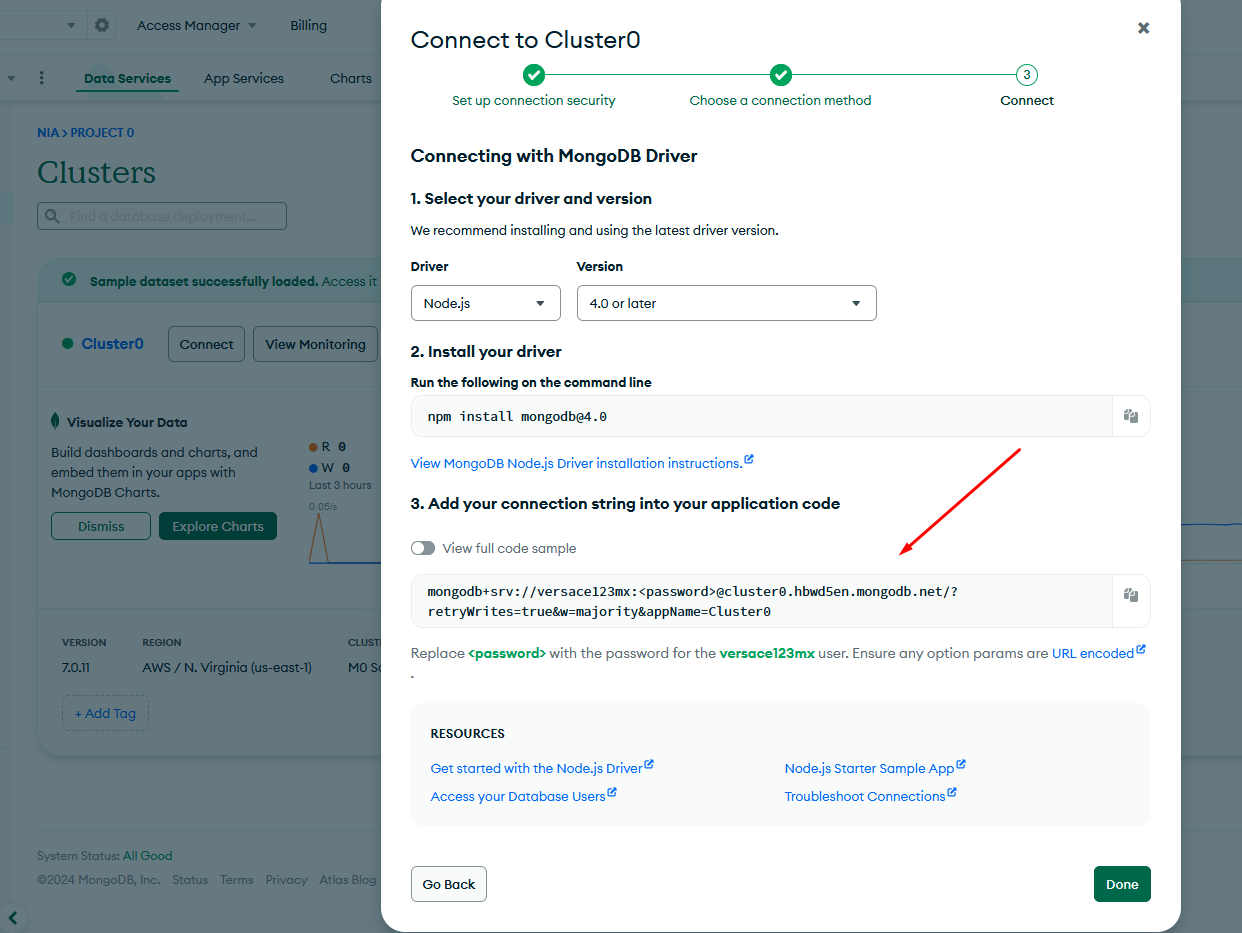
Ahora ejecutamos npm run dev y ya estará escuchando por todos nuestros cambios. Esto es solo para desarrollo ya que en producción el servidor de node ejecuta start: node index.js y ahí se queda.

1. Instalamos el ORM npm i [mongoose](mailto:mongoose@6.1.8)
2. Ahora hay que hacer la configuración a la base de datos
3. Creamos una carpeta config, con un archivo db.js donde ira la configuración

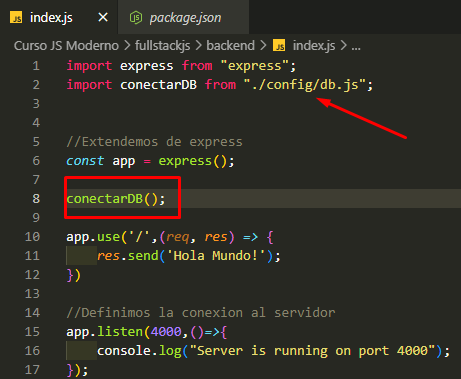




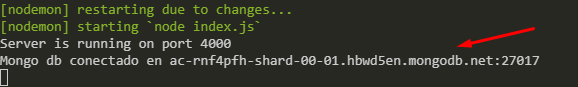
La cadena de coneccion se saca de la pagina de mongo.



En el archivo de configuración de la base de datos llamamos a conectarDB()



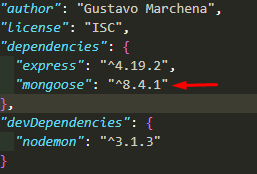
Coneccion correcta



Versión funcional, pero voy a subir de versiones a ver si no se crash

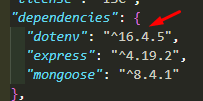


Solo se actualizo mongoose a su versión 8.4.1 sin crash

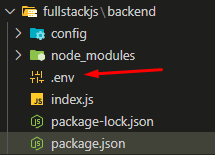


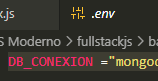
Generar una variable de entorno para evitar pasarle los parámetros de conexión directos

1. Descargamos la dependencia dotenv npm i dotenv



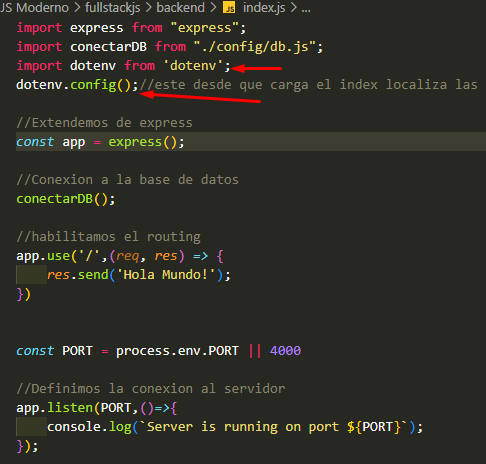
1. Creamos un archivo llamado .env



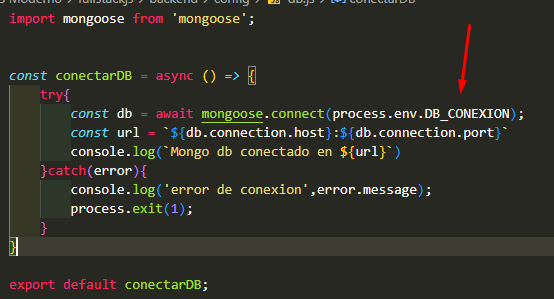


1. Pasamos la configuración al archivo de conexión db.js, nota dotenv.config cuando se llama se genera de forma global, por lo tanto se puede pasar en este mismo archivo o en otro, en nuestro caso lo movimos a la configuración global del index.js





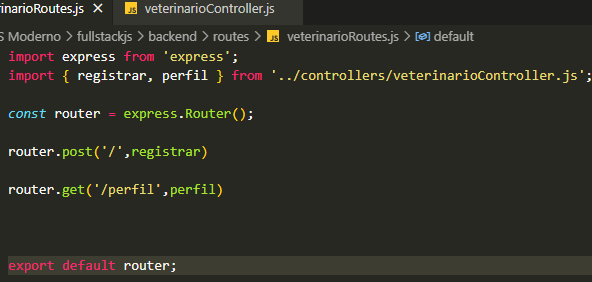
Y en la conexión a la db se puede llamar sin problema, por que es global



Creando el routing



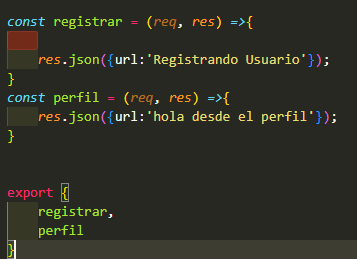
Codigo de las rutas, para no tener todo el código en el router se conecta con el controlador para las rutas y el controlador lleve la logica



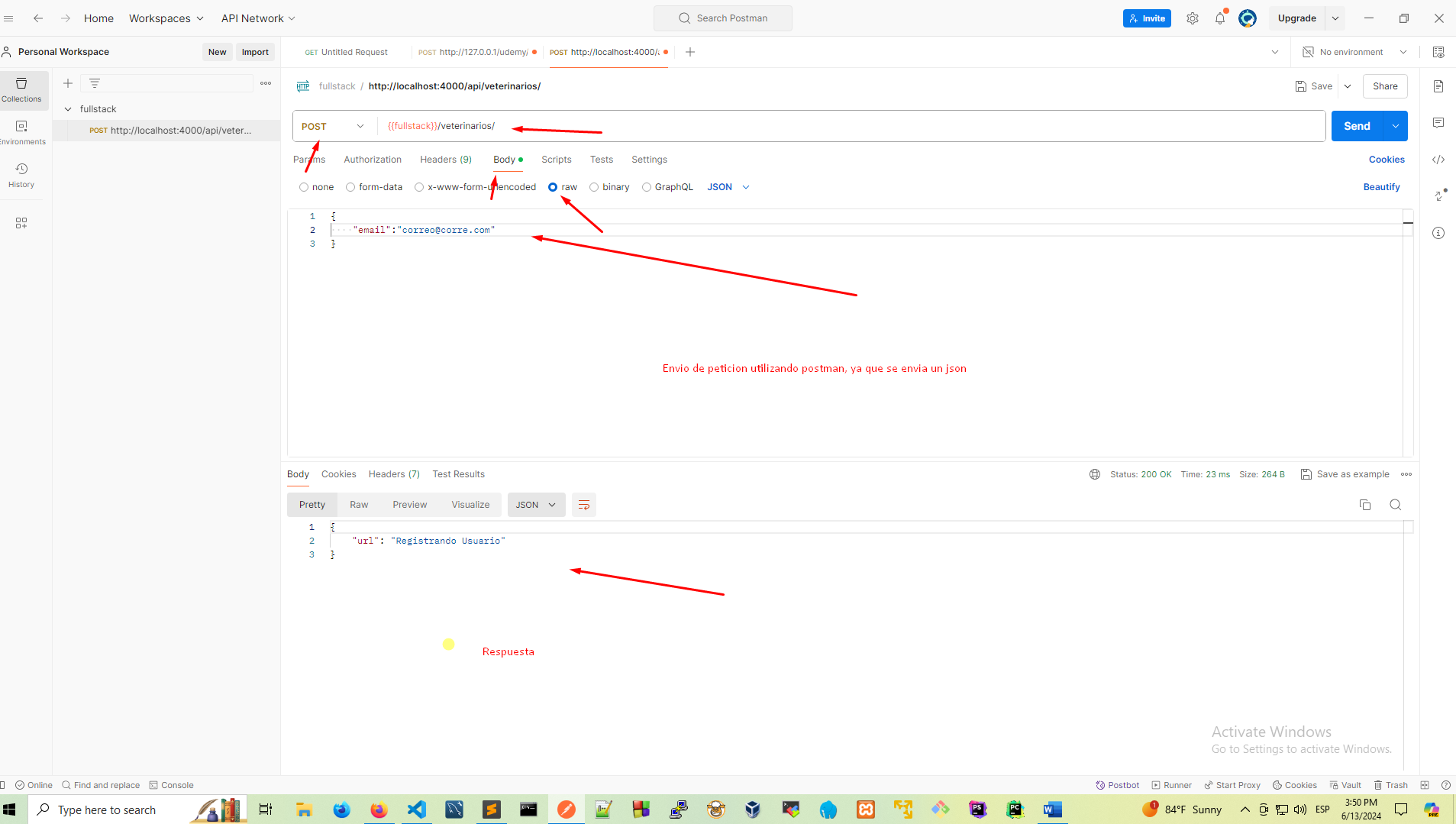
Generando controlador de rutas



Codigo del controlador de las rutas, res.json es que la respuesta será de tipo json

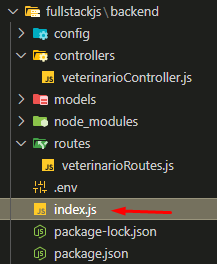


Envio de datos por postman via json

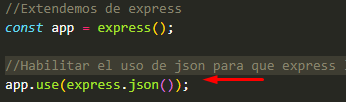


Para leer los datos que le mandamos via json desde postman se requiere habilitar una función que entiende express que los datos que le mandamos son json

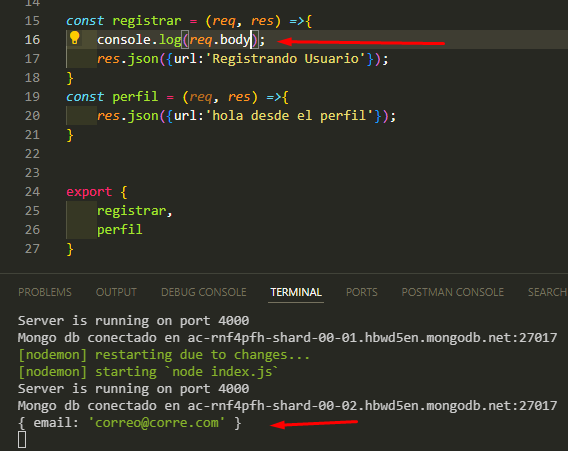
En el archivo index.js de configuración global se coloca lo



app.use(express.json)

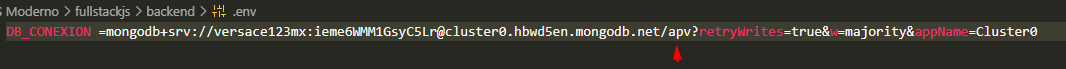


Colocamos un console log a req.body y obtenemos el resultado desde postman

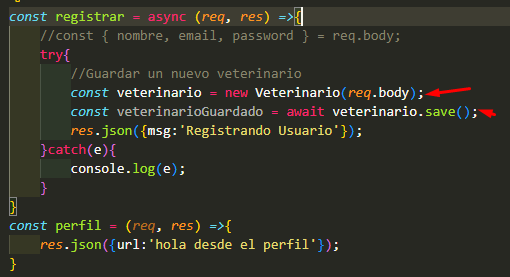


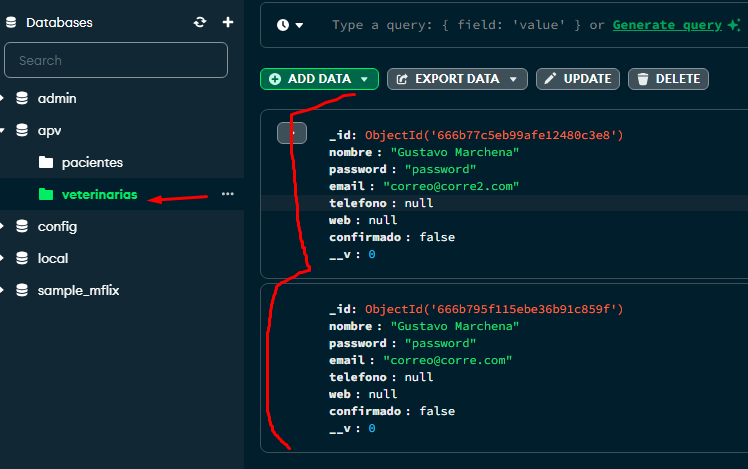
Para insertar los datos en Mongo

1. Verificar a donde apunta nuestra cadena de conexión, si no trea el valor entre / y ? ahí le pones el nombre de nuestra coleccion



1. Con las líneas de veterinario insertamos los datos y veterinarioGuardado

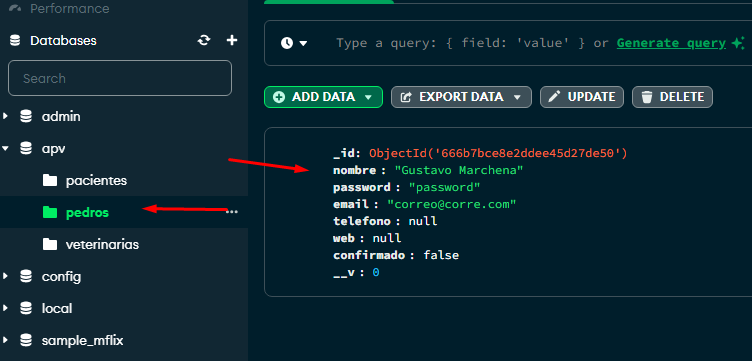


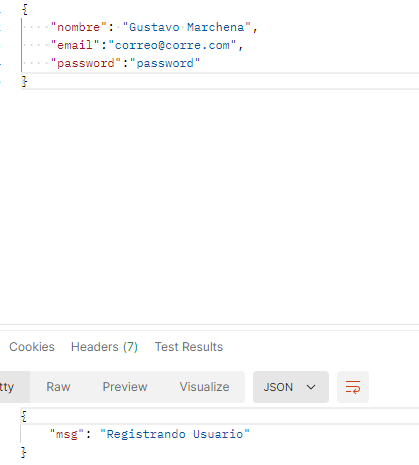


Nota: si al modelo de veterinaria le cambio el nombre este cuando haga la petición desde postma este creara esa colección

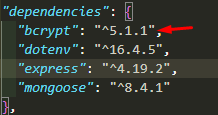




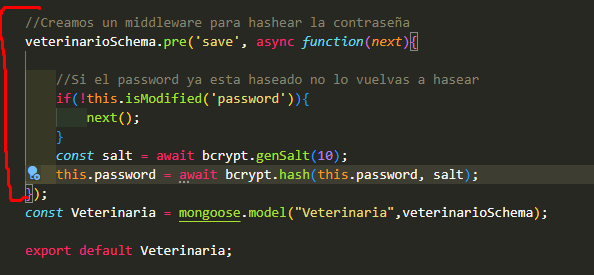




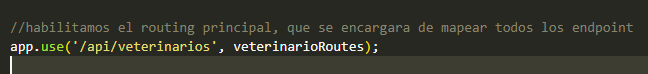
Librería para hasear password



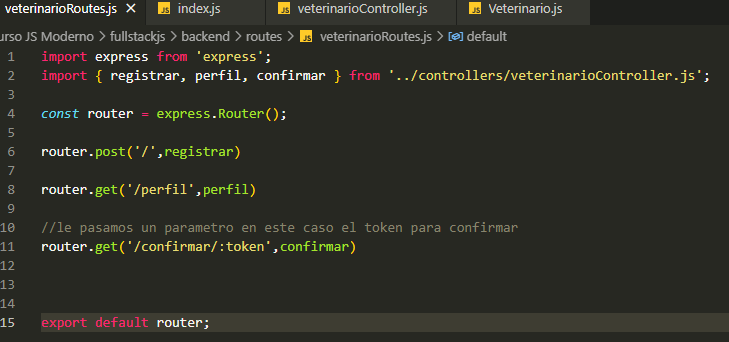
Se usa antes de guardar el registro



En el index.js creamos el routing del api



El archivo de routing

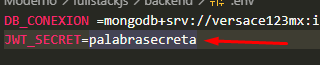


La lógica en el controlador

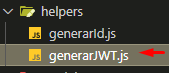


En las aplicaciones tradicionales cuando te logueas se genera una sesión con la cual podemos saber si puedes o no entrar a una pagina y mantener cierta información del usuario, en este caso no tenemos una sesión por que esta parte son apis y no generan sesiones para eso utilizamos un jwt jason web token

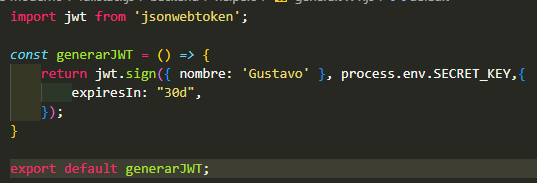
1. Instalamos npm i jsonwebtoken
2. Generamos una llave secreta en el archivo env



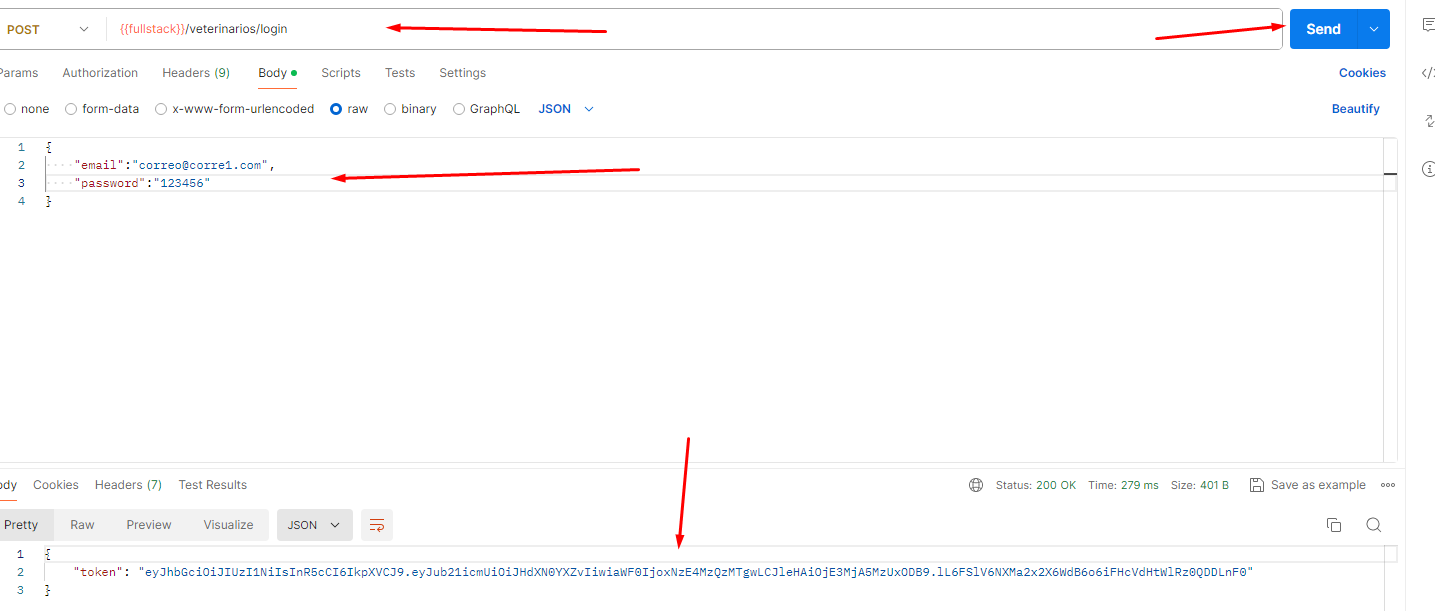
1. Creamos la función que genera el token



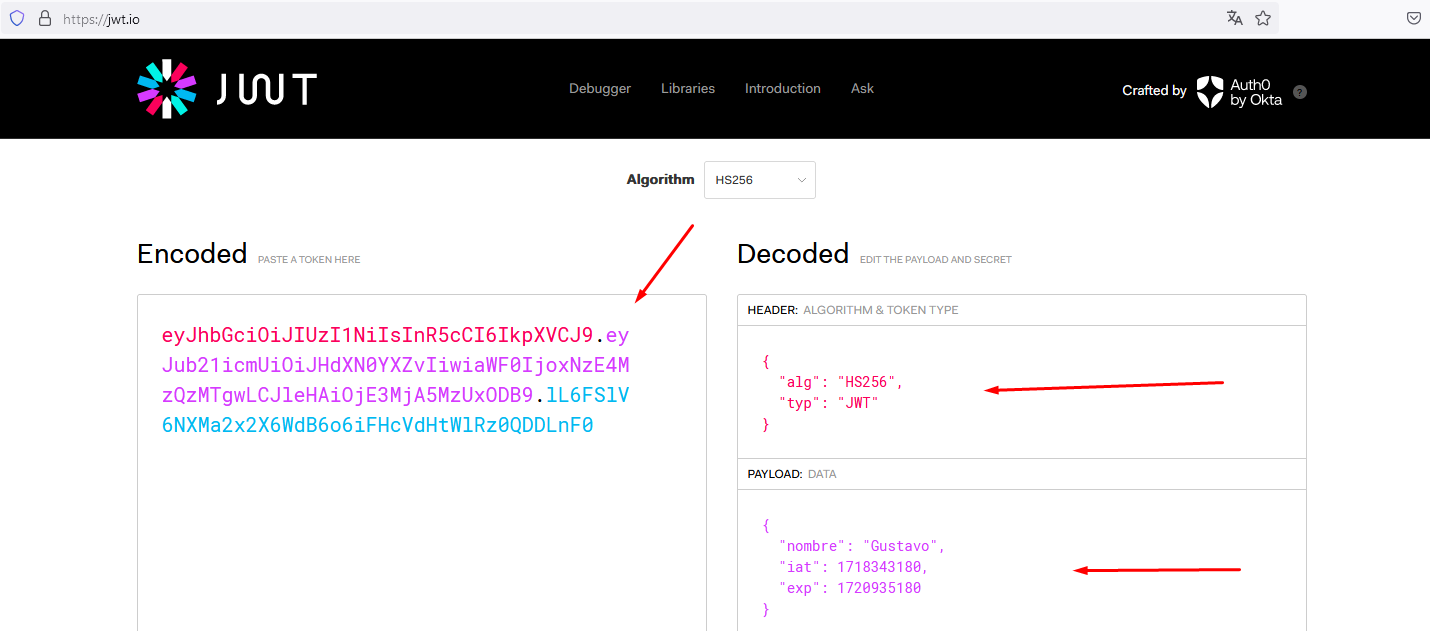
Muestra actual de como se genera, ese ‘Gustavo se qui ahí va el nombre que se obtiene de la consulta’, pero mas adelante se muestra como. Por ahora son solo pruebas para ver como se genera el JWT y lo que devuelve. Aquí falto colocar la imagen de donde se importa y se llama a la función generarJWT

ç

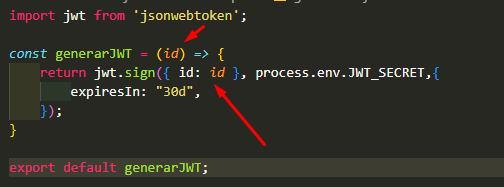
Lo que devuelve el JWT en una petición valida



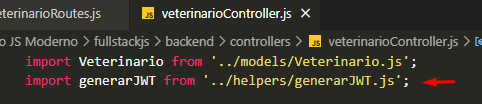
Ahora checamos en la pagina de JWT lo que nos manda ese has



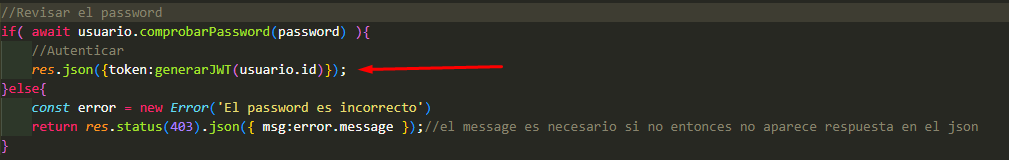
En este caso juan solo guarda el id del usuario que intenta loguearse



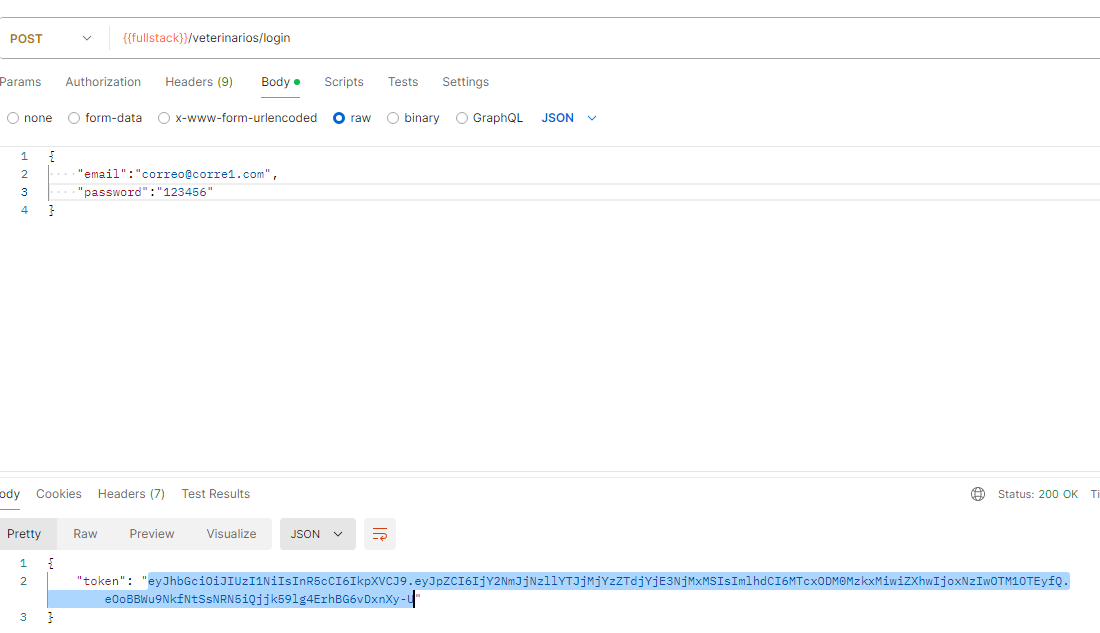
Importamos nuestro helper

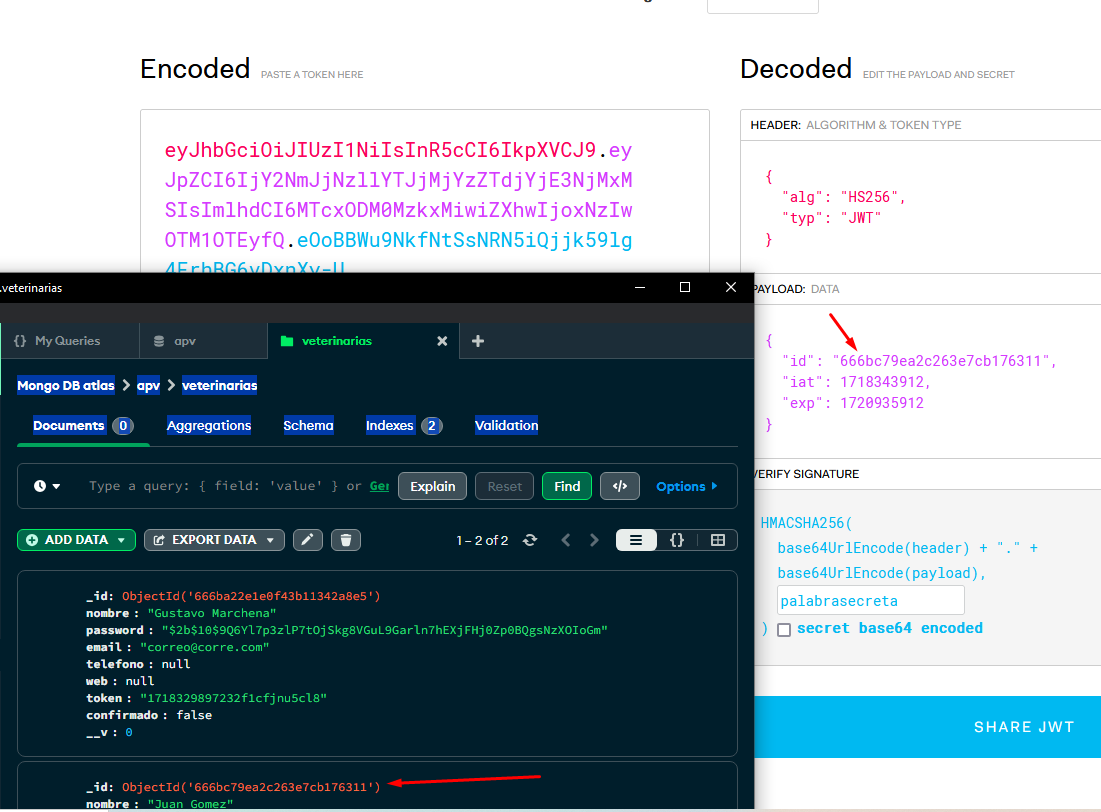


Llamamos a la función de generarJWT



Respuesta de postman

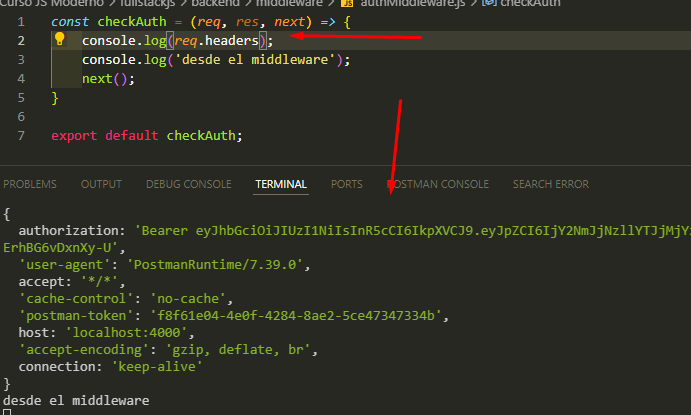




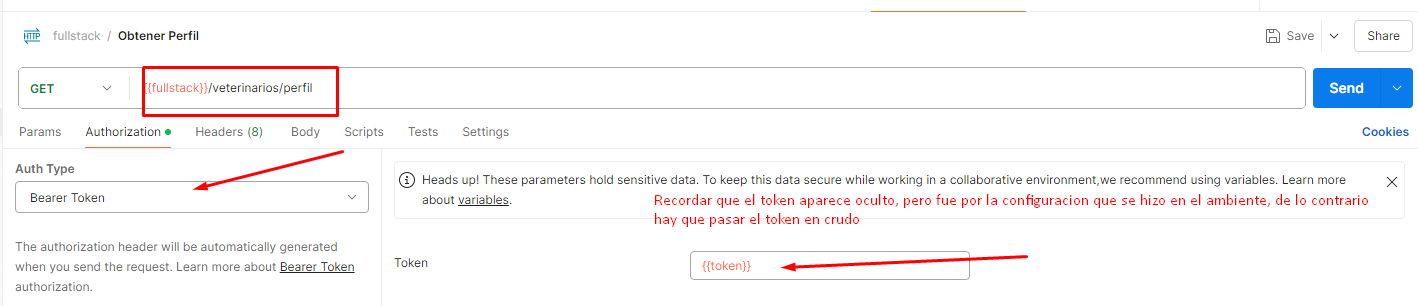
Creando Rutas Protegidas

Se genera un middleware para proteger rutas, las rutas para registrar usuario, confirmar la cuenta, y autenticación son url publicas, pero las otras rutas que ya hacen acciones sobre el flujo del sistema ya tiene que ser rutas protegidas que validan el usuario mediante el token.

Recibiendo los header que se mandaron via postman



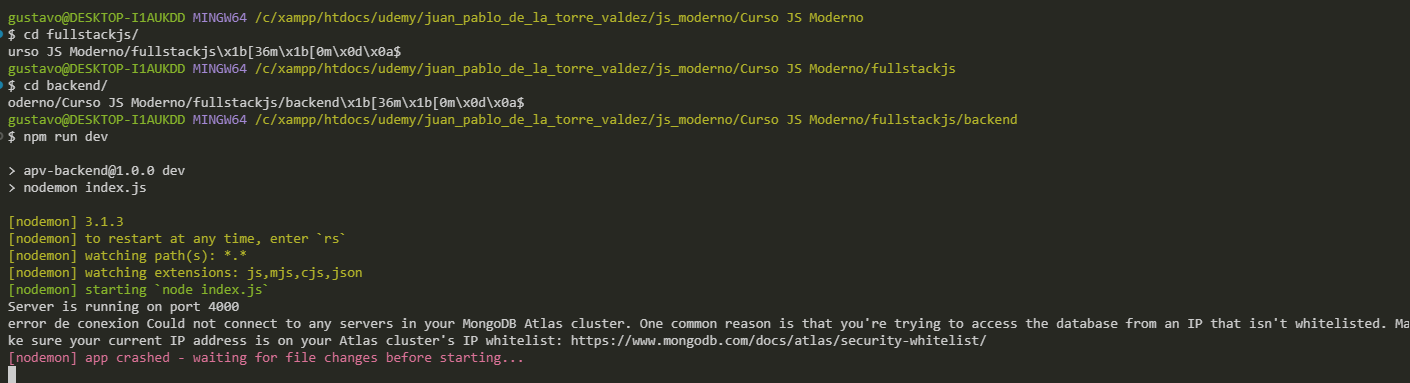
La petición



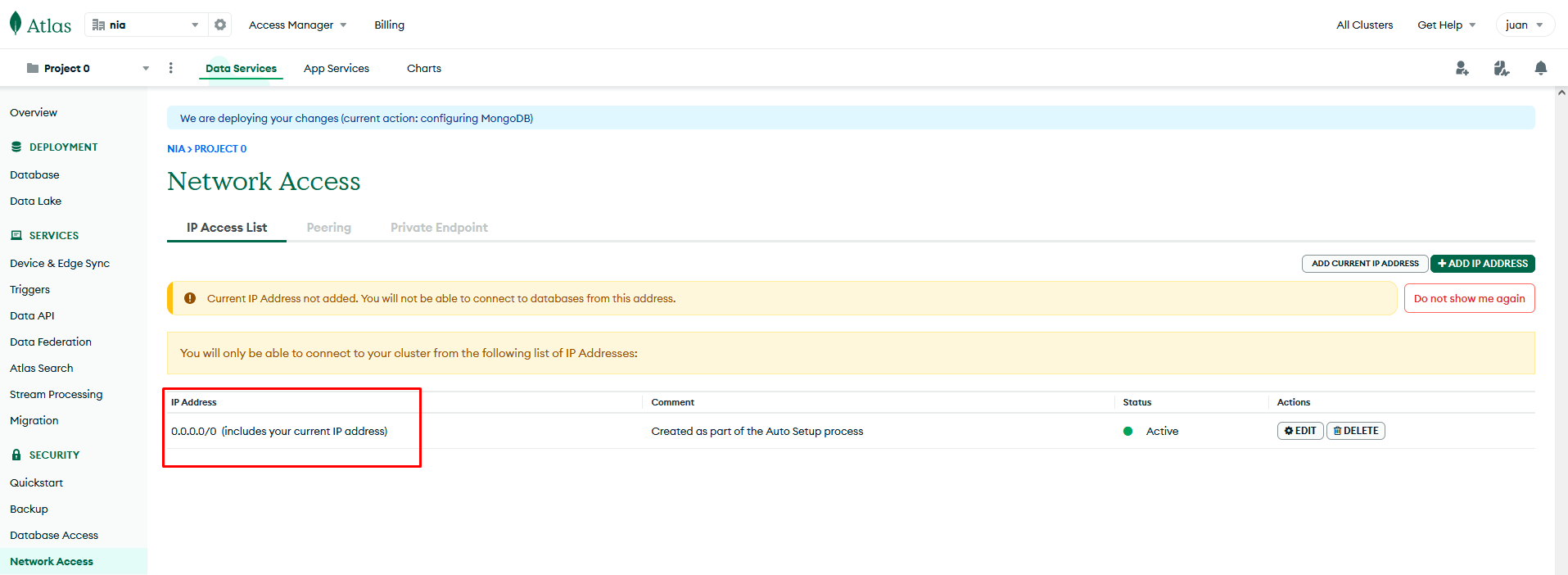
El paquete de jwt permite crear token pero también permite comprobar el token

Nota: la maquina se apago y la ip cambio ahora ya no me deja entrar a mongo

Para ello hay que configurar la ip en mongo y decirle que se conecte desde cualquier ip

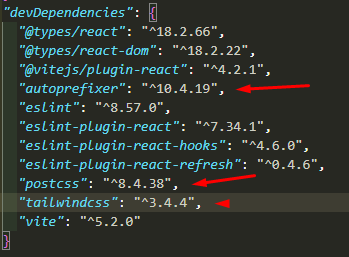


Se configuro para que desde cualquier ip se pueda acceder

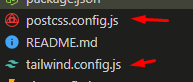


Front

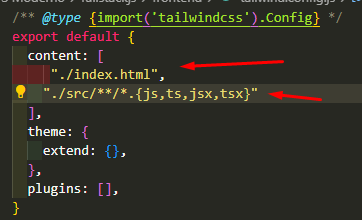
1. Inciamos un proyecto con vite npm init vite@latest
2. Instalamos lo que solicita vite npm install
3. Instalamos dependencias npm i -D tailwindcss postcss autoprefixer



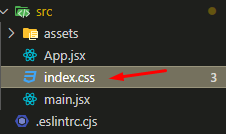
1. Creamos el archivo de configuración de tailwind npx tailwindcss init -p
2. Configuramos el archivo que se creo de tailwind.config.js

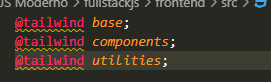


Archivo tailwind.js

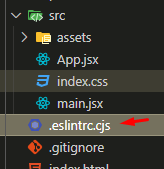


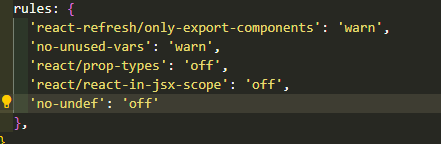
Configuracion del archivo index.css para que entienda tailwind





Configuracion de eslintrc para que no me marque error





Comenta Juan pablo que para trabajar con diferentes paginas en React se tiene que instalar React Router Dom

npm i react-router-dom

BrowserRouter

Todo tiene que estar rodeado por esa

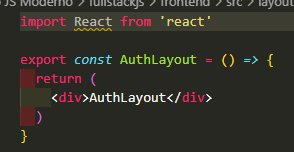
Routes

Permite agrupar difertentes rutas

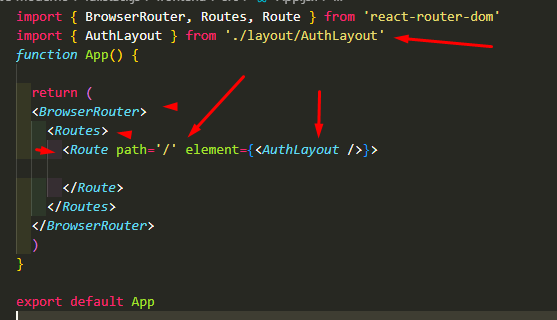
Route

Permite una ruta en especifico

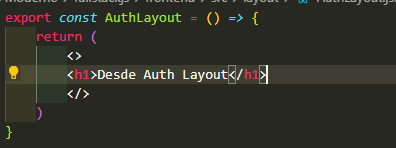
Rafc ayuda a crear un componente mas rápido, ya que nos genera la estructura



Como funciona el router, importamos un layout (componente) que acabamos de crear, como se comento anteriormente todo debe de ir rodeado por BrowserRouter, Routes que agrupa diferentes rutas y Route para una ruta especifica, luego en Route se le pasa una ruta, y con el element le dice que cuando cargue esa ruta muestre lo que hay en AuthLayout (Componente)

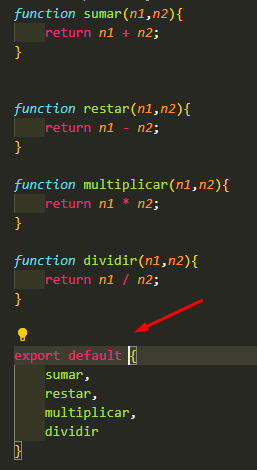


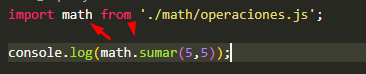
Lo que contiene Auth Layout hasta el momento

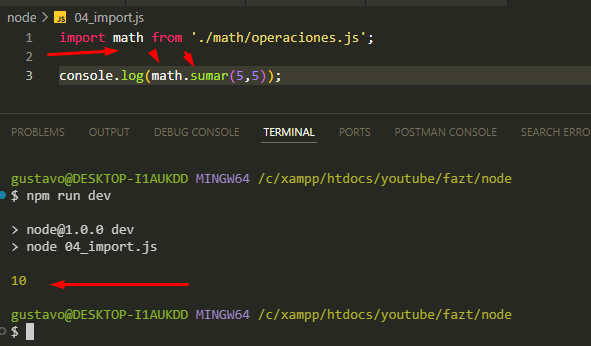


Nota, en el export default tu le puede poner otro nombre a lo que importas, pero no funciona con export

Ejemplo estamos exportando por default sumar, restar, multiplicar, dividir. Y en la importancion le estoy poniendo math, para acceder a su elementos seria math.sumar

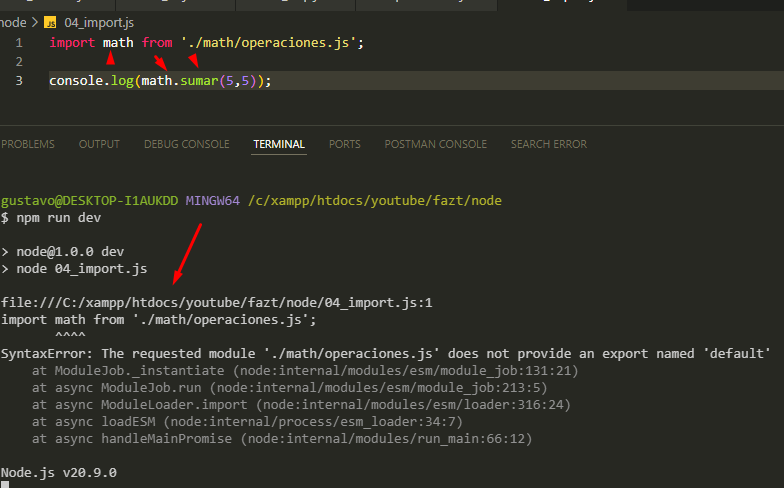






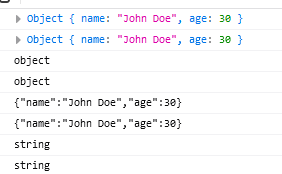
Pero si le hago un export y que no sea por default no le puedo poner math, lo que entiendo que ese mat pasa a ser un alias cuando es un export default



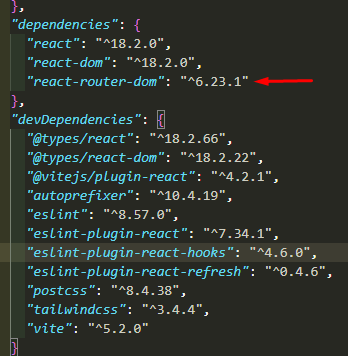


Nota Json:





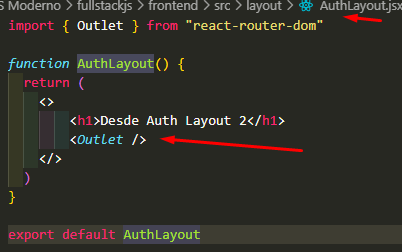
Para la parte del Front creamos un proyecto nuevo con vite, se instalo tailwind, se configuro tailwind como ya se ha configurado en proyectos pasados y se instalo react router dom, para el manejos de diferentes paginas con el router



En la parte del front se han creado las carpetas layout que es la pagina maestra

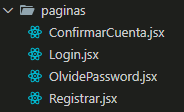


Con el comando rfce nos crea una estructura de componente y nos crea algo como la función y el export, el Outlet nos ayuda a intectar otras paginas en esta, ósea layout trae un diseño predefinido, y para que las otras paginas hagna useo de este requerimos el layaout



Se creo otra carpeta para las paginas llamada paginas donde se almacenan los distintas paginas

Se se utilizaran para este componente y tener todo separado y organizado



El código de Login hasta el momento seria, <></> son Fragments y esto van vacíos para que solo aparezcan las etiquetas h1, o se podría usar con div y luego el h1, pero siempre deben de ir rodeados de un Fragments o un div si no marca error, a qui la cuestión es como se mostrar el html de salida con div:

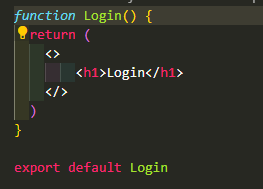
<div>

<h1>Login</h1>

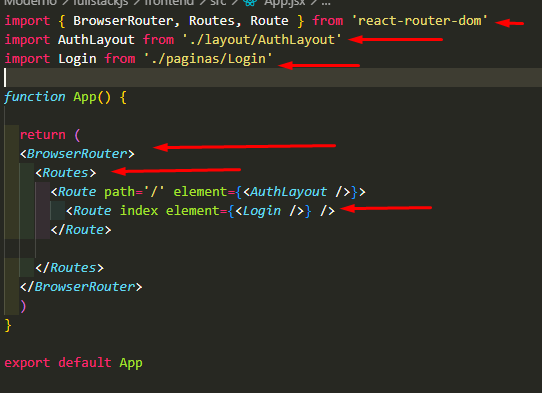
</div>

Con Fragments

La salida solo seria el <h1>Login</h1>



El componente App.jsx es el principal para ello hay que llamar al layout y a sus paginas



Todo tiene que ir dentro de BrowserRouter

Routes agrupa diferentes rutas

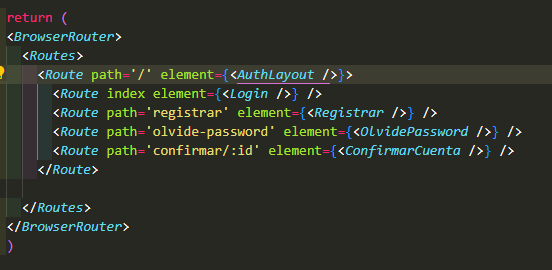
Route es la ruta en especifico

Dentro de un Routes podemos tener mas rutas

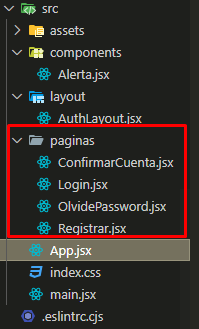
Y dentro de las rutas también podemos poner mas

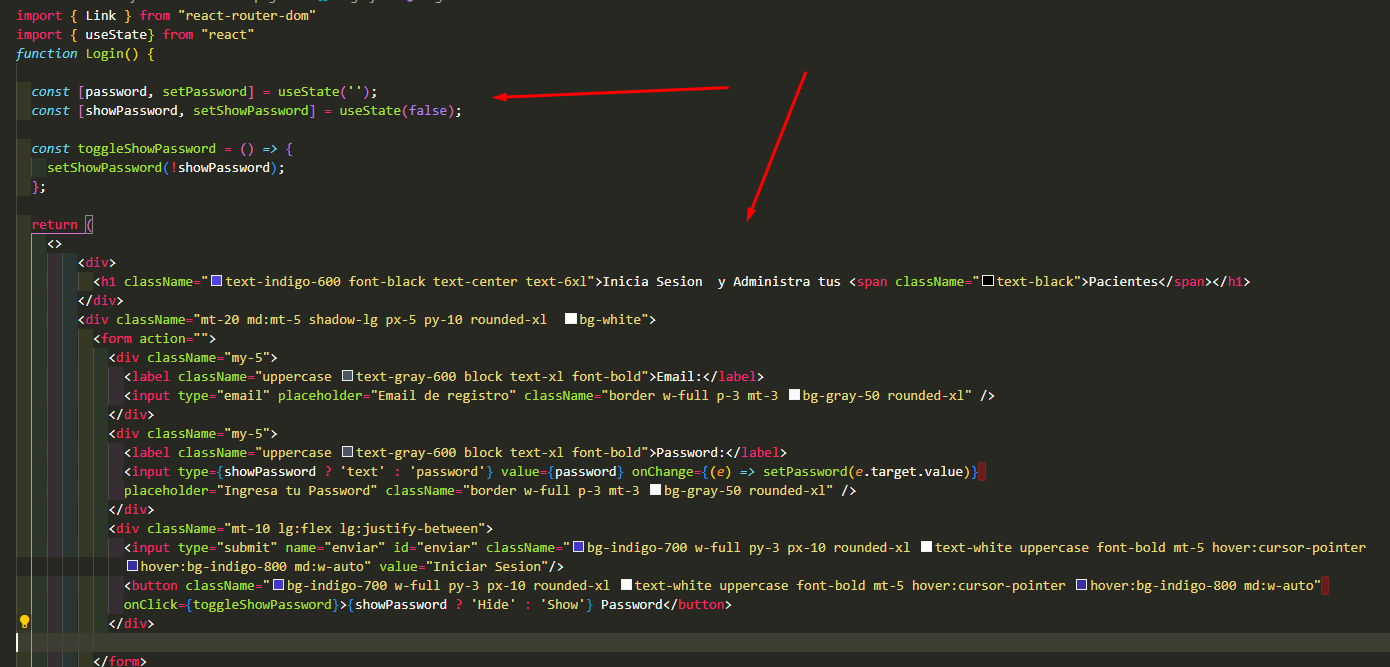
En este caso Route path=’/’ dice que es la ruta principal y cargara por default AuthLayout que es el componente principal de nuestro citio

Dentro hay otro route que contiene las otras rutas ‘paginas que pueden estar asociadas al layout principal’

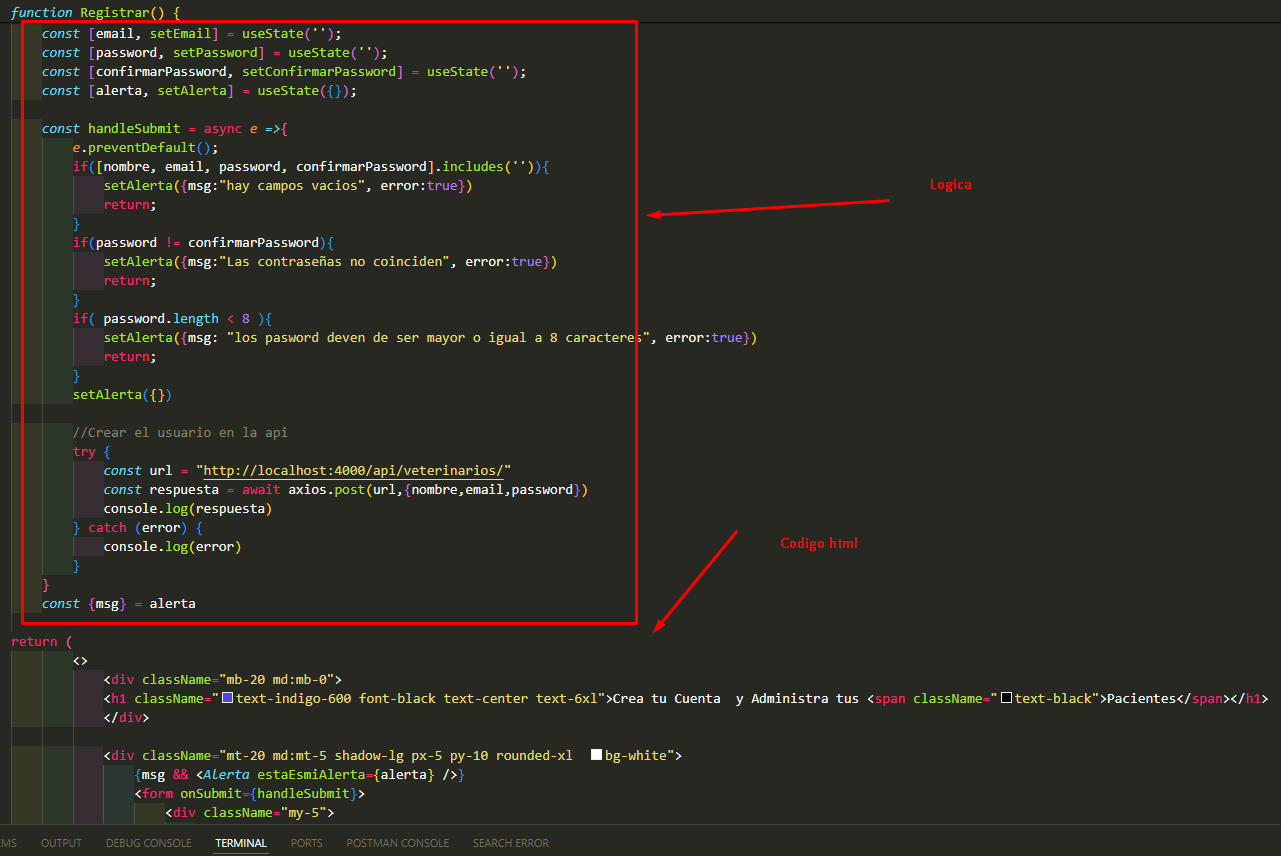


Ejemplo de login

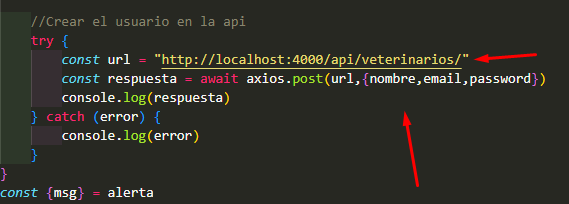




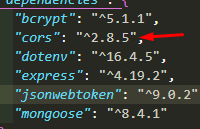
Ejemplo de Crear cuenta ‘’Registrarse’’



Utilizando axios para realizar las peticiones, podríamos utilizar fetch pero aquí utilizamos la librería axios., cuando se le pasen parámetros al endpoint van después de los parámetros, eso lo muestro cuando genermos el token

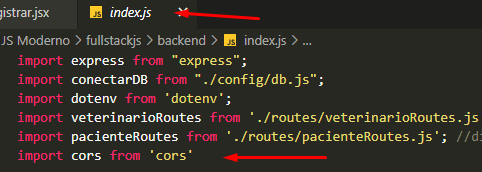


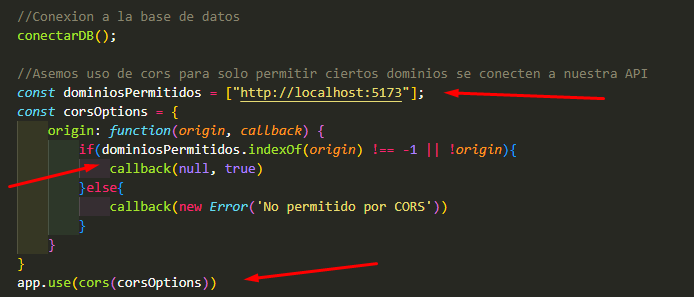
Para temas de cors utilizamos la librería cors en el backend



Configuramos el cors en el backend index.js que es desde donde se ejecuta nuestra aplicación

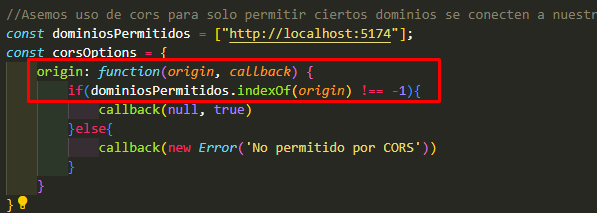
Importamos la librería





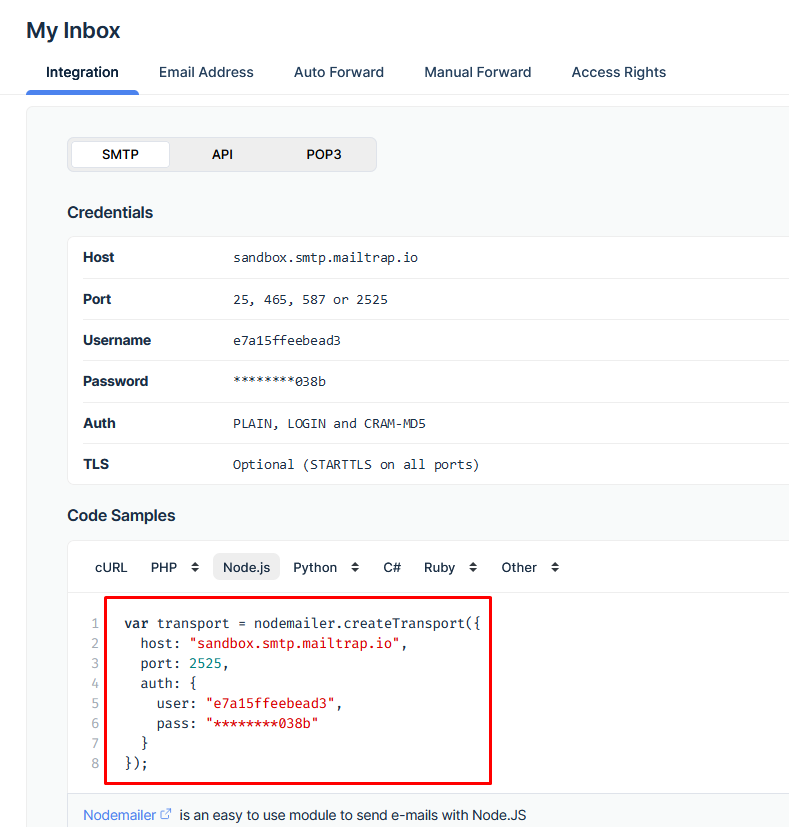
En la parte de políticas de cors solo debemos acetar peticiones de http://localhost:5173

Para este caso, pero en el código que yo utilizo también acepta otros orígenes con la parte de !origin dice o que sea diferente al origen, para evitar eso y solo se acepten peticiones desde el la ip del front el código correcto es. Quitan || !origin para que quede de la siguiente forma.



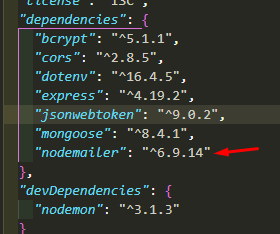
Madar Mail para la confirmación con la librería nodemailer y creamos una cuenta provisional en mailtrap para generar correos electrónicos.

Tomamos la configuración de smt que nos entre mailtrap



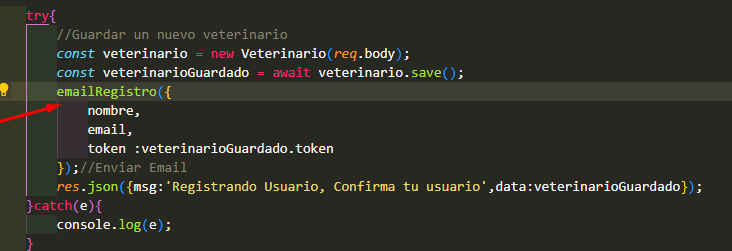
Instalamos nodemailer para el envio de correos





Configurando Mailer

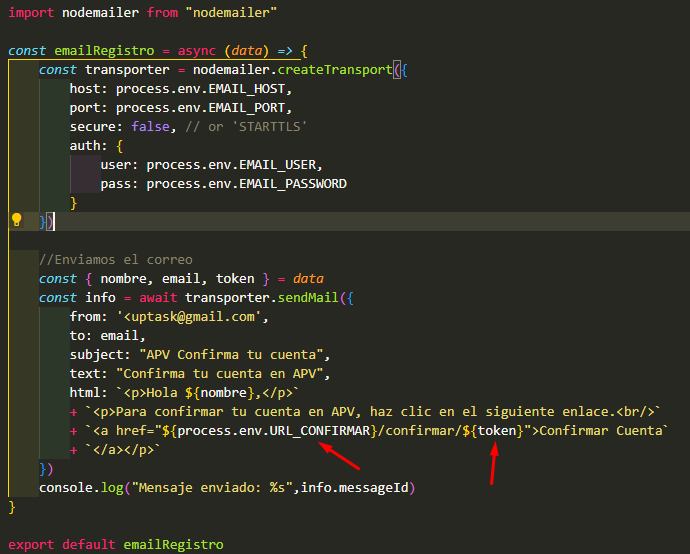
Mandamos los parámetros en la función



Importamos el método que ara el envio



Codigo del método que envia el email, utilizando variables de entorno para que sea configurable



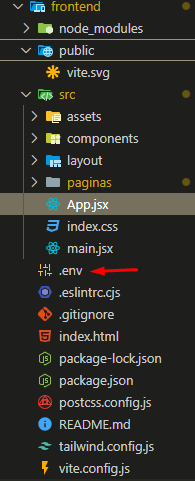
La url a la que se envia no será la del api será la del from para confirmar desde ahí y luego ya se conecte con la api

Ip del servidor del front



Configurando variables de entorno en vit

Creamos en .env



Declaramos la variables, siempre tiene que llevar VITE\_ por delante si no, entonces no las reconoce

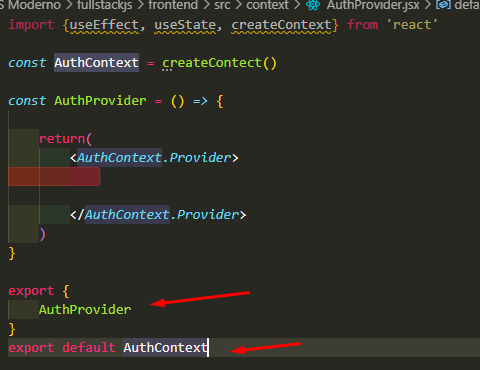


Como se usa, recuerda reiniciar cada vez que se creen variables de entorno



Creando un provider





Importamos



Como usarlo, esto se hace para que el código de este context este disponible en todos los hijos esto será como la sesión que se debe de mantener en todo el flujo de la palicacion, para eso lo estamos asiendo



Ahora no solo basta encerrarlo también hay que pasarle un prop, para decirle todos sus hijos tendrán siponible el codigo

