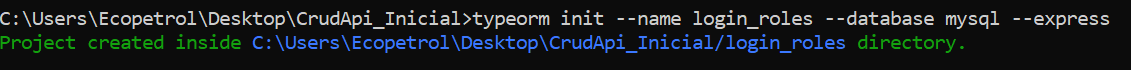
PARTE 1

1. Instalar de manera global

npm i typeorm -g

1. Nos ubicamos en la raíz del proyecto y ejecutamos Ejecuta por consola (mysql)

typeorm init --name NOMBRE\_PROYECTO --database MOTOR\_BASEDATOS --express

Como se ve acontinuacion:

1. Accedemos a la carpeta login\_roles 🡪 code .
2. Como resultado nos genera la carpeta src

En el documento package.json eliminar 🡪  "typescript": "3.3.3333"

"body-parser": "^1.18.1"

1. Instalar las dependecnias del front como lo son cors con el siguiente comando en la raíz del proyecto:

npm i cors jsonwebtoken bcryptjs class-validator

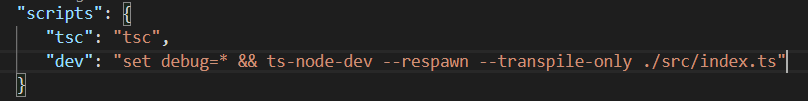
1. Luego insitalar en la misma raíz:

npm i -D ts-node-dev typescript @types/bcryptjs @types/cors @types/jsonwebtoken @types/express @types/node

1. El documento tsconfig.json cambiar 1 linea y agregar otra como se ve:
2. Eliminar la carpeta node\_modules e insertar el siguiente comando:

npm i

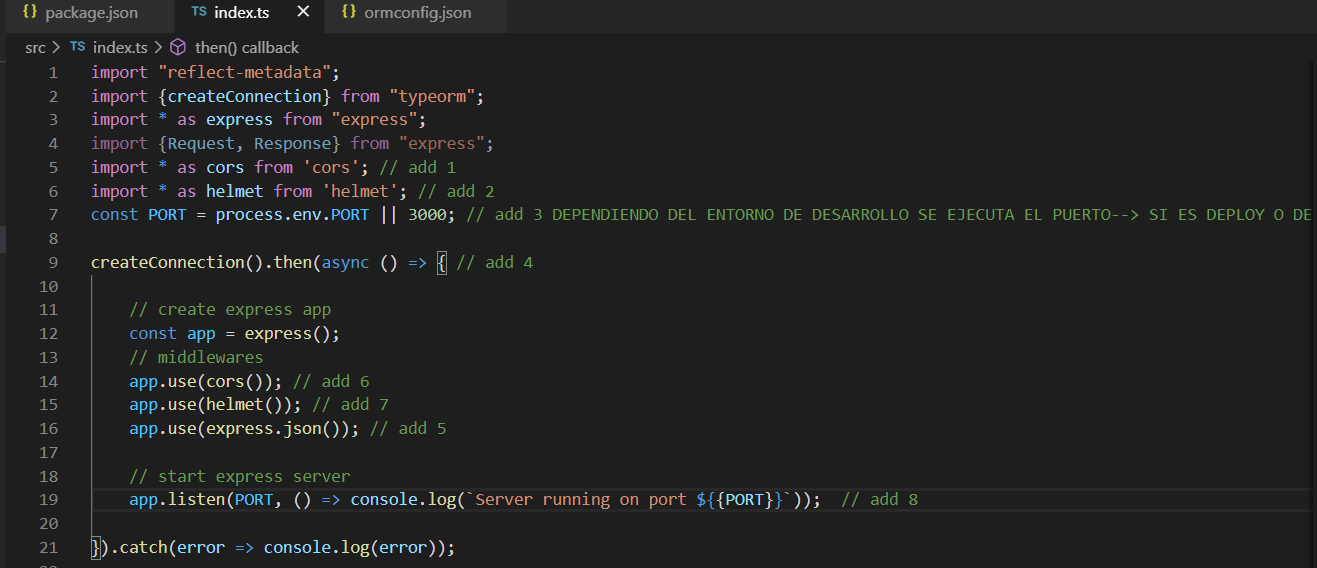
1. En el archivo package.json modificarlo y dejarlo asi:



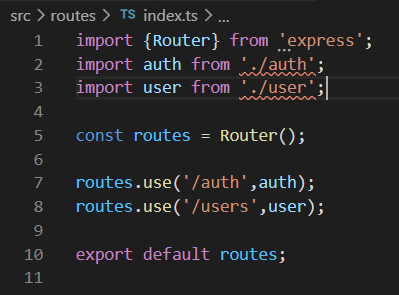
1. Se configura el documento ormconfig.json🡪es donde se realiza la conexión base datos
2. Se ejecuta el siguiente comando🡪 npm run dev

LIMPIEZA DE CODIGO

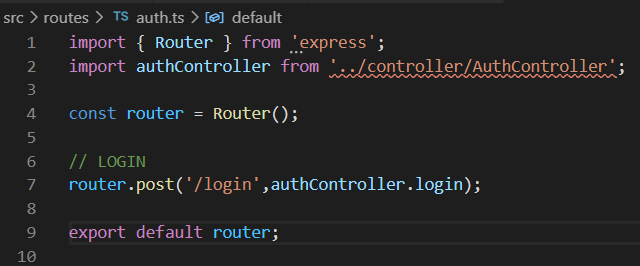
1. Se eliminar el documento routes.ts
2. Se instala en la raíz del proyecto🡪 npm i helmet
3. El documento index.ts quedaría algo asi: se agregan todos los add



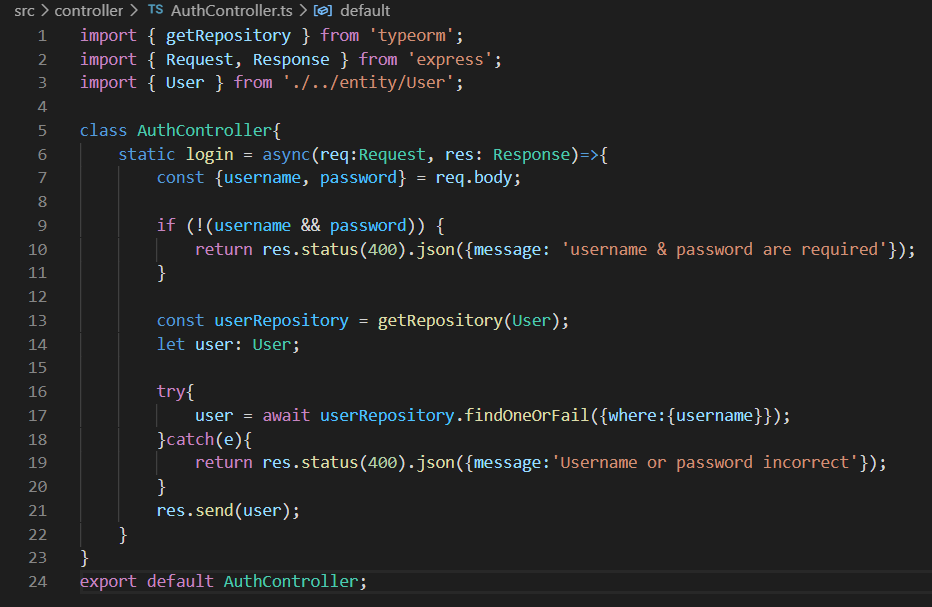
1. Se crea una carpeta routes 🡪 con 3 doc(index.ts, auth.ts, user.ts) y Controller/AuthController.ts
2. En el doc index.ts🡪 colocar lo siguiente



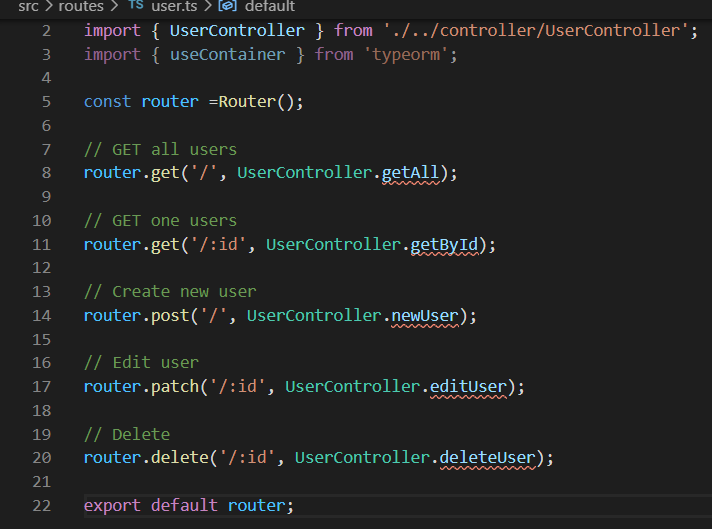
1. En el doc auth.ts 🡪 colocar lo siguiente



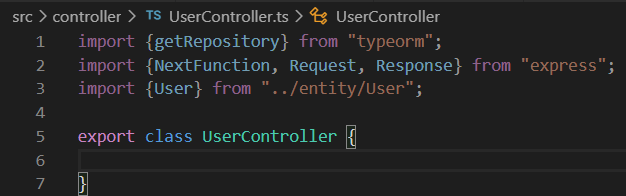
1. En el doc controller/AuthController.ts



1. El doc routes/user.ts 🡪 se coloca lo siguiente eso si sale error por que aun no se han creado los métodos



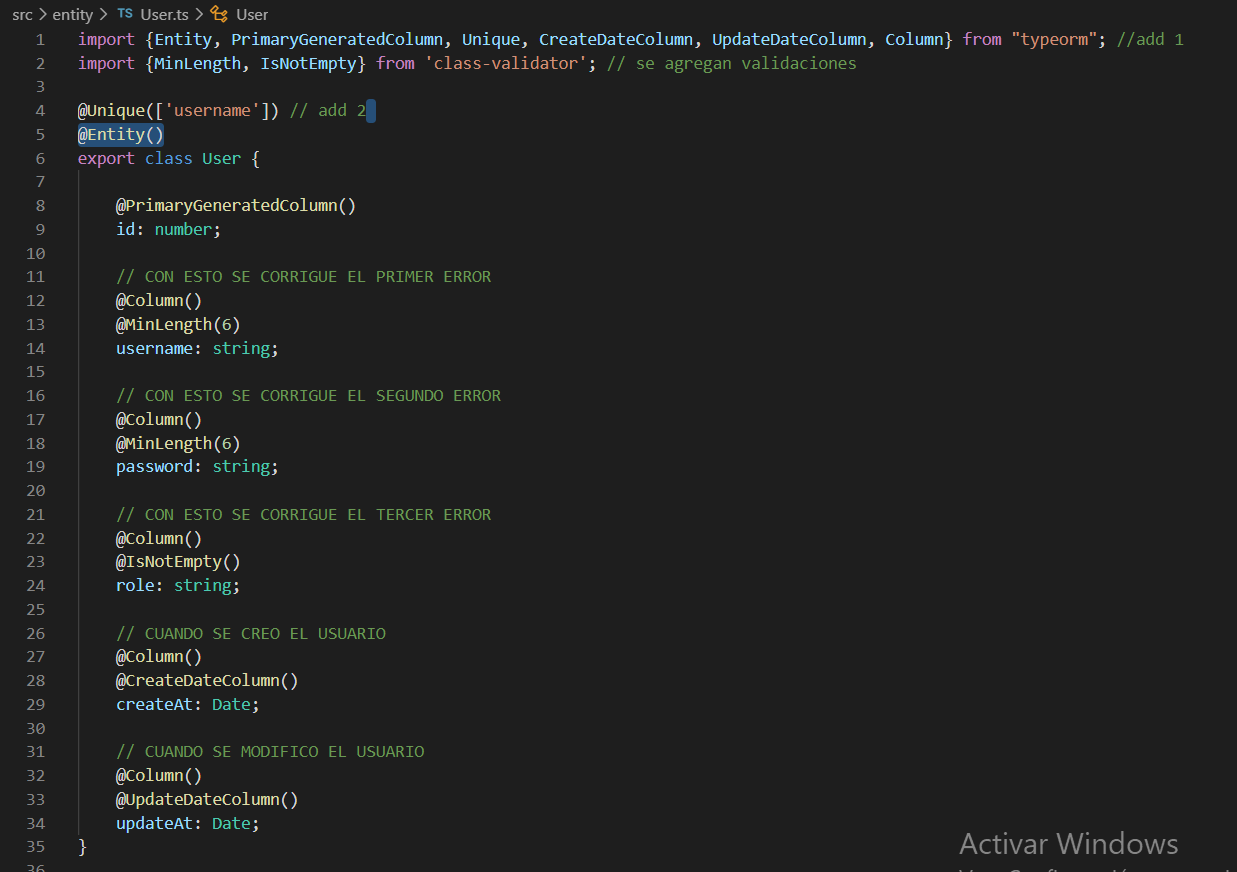
1. En el doc controller/UserController.ts🡪 eliminar el contenido que quede asi, es para agregar los métodos del pantallazo anterior



1. Se crean los 5 metodos dentro de la clase User Controller

|  |
| --- |
|  |
|  |
|  |
| User |
|  |
| SE FINALIZA CON ESA LINEA |

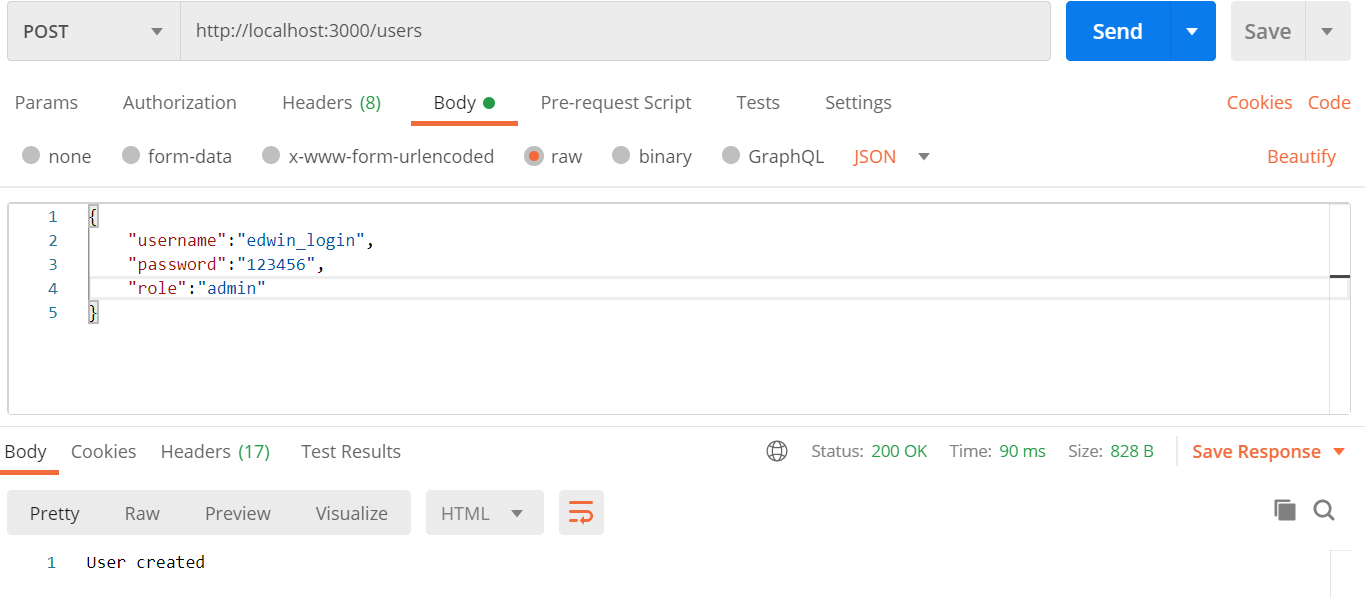
1. Ahora corregir el error anterior se abre el doc entity/user.ts



1. Se agregan estas dos líneas en el doc src/index.ts



1. PARA COMPROBAR SI ESTA BIEN CON EL SIG COMANDO: npm run dev
2. Eliminar los usuarios de la BD 🡪 tabla user
3. Ejecutar el comando punto 25 con POSTMAN 🡪 SRC/ROUTES/USER.TS
4. Para el método POST se coloca algo asi: y enviardo 🡪 SEND



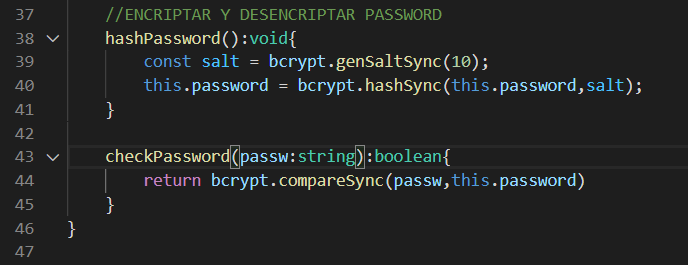
1. Ya teniendo eso, ya con el motodo GET se podrá ver el usuario que se creo, con el método PATCH se podrá editar
2. Despues de verificar los 5 metodos que funcionen se procede a realizar una prueba de un login en este casi se hace con <http://localhost:3000/auth/login>. Como se ve a continuación, recordar que tiene que estar guardado en la base de datos

|  |
| --- |
| Cuando se ingresa un login diferente |
|  |
| Cuando se ingresa un login Correctamente |
| SE PUEDE VER EL ESTATUS 200 |

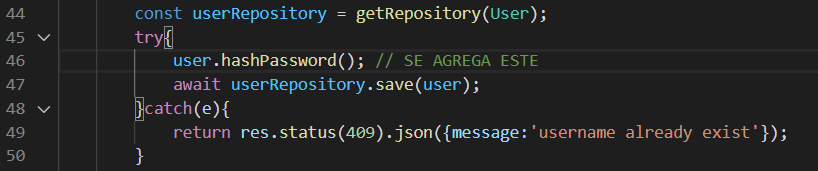
1. Para encriptar(hashSync) la contraseña, y desencriptar(comparateSync). Se abre src/entity/user.ts

Se importa la dependencia del enriptador de password

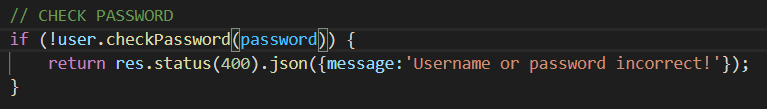




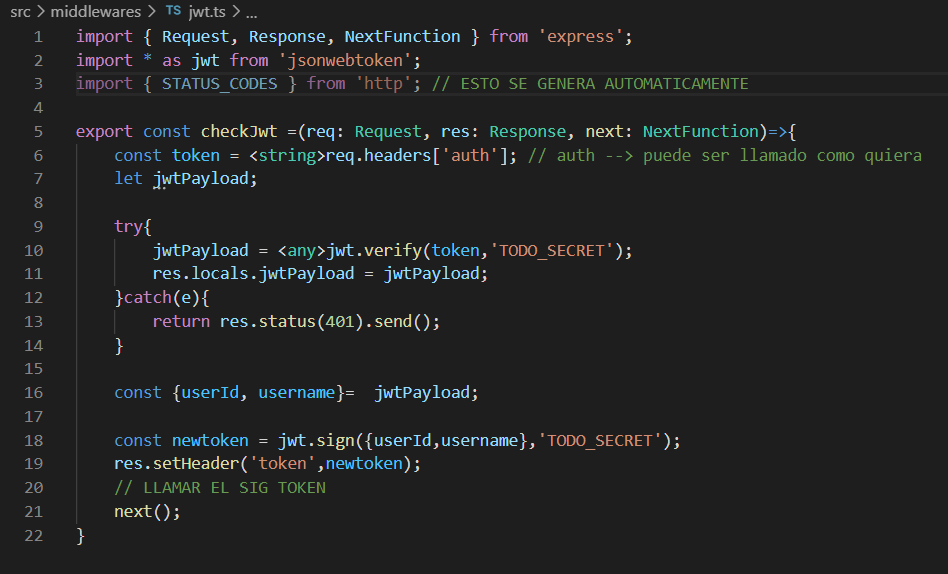
1. El doc controller/usercontroller.ts en el Metodo newUser Se agrega el motodo que se creo en el punto 30

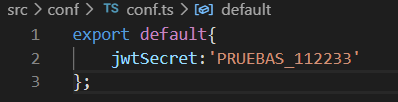


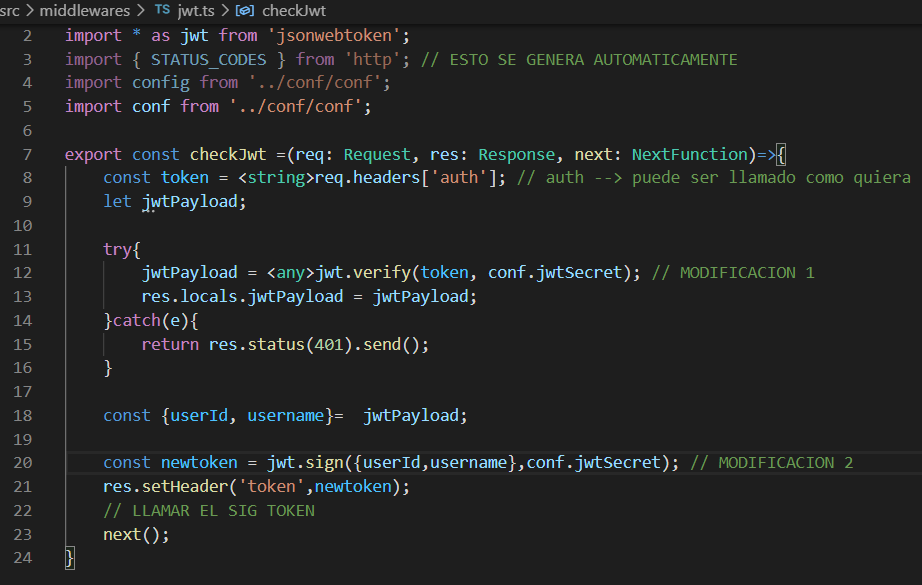
1. Se prueba con el POSTMAN para verificar si funciona encryptacion
2. Se configura para que reciban password encriptados, nos ubicamos en controller/AuthController.ts. después de la condicional. Se introduce el segundo método del punto 30 dentro de la misma clase:



1. Se comprueba nuevamente en el postman método POST 🡪 LOGIN
2. Por lo general cuando se logea muestra los datos de la cuenta para evitar esto se realiza lo sig: en Src 🡪 se crea una carpeta middlewares 🡪 y un doc jwt.ts



1. Se crea una carpeta dentro de SRC 🡪 conf y dentro un doc conf.ts
2. Nos devolvemos al documento jwt.ts. agregamos el import del documento punto 36 Linea 5 y 12, 20 como se ve acontinuacion



1. Para poder logearse dependiendo el rol se hace lo sig: En el doc routes/user.ts. Indicarle a la url permisos respectivos si son admin o un invitado, para ello primero confi de la sig manera:

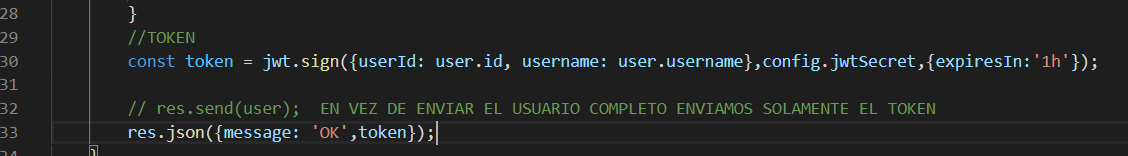
router.get('/', [checkJwt], UserController.getAll);

le agregamos la parte [checkJwt], para que reconozca primero los permisos, luego que traiga los usuarios

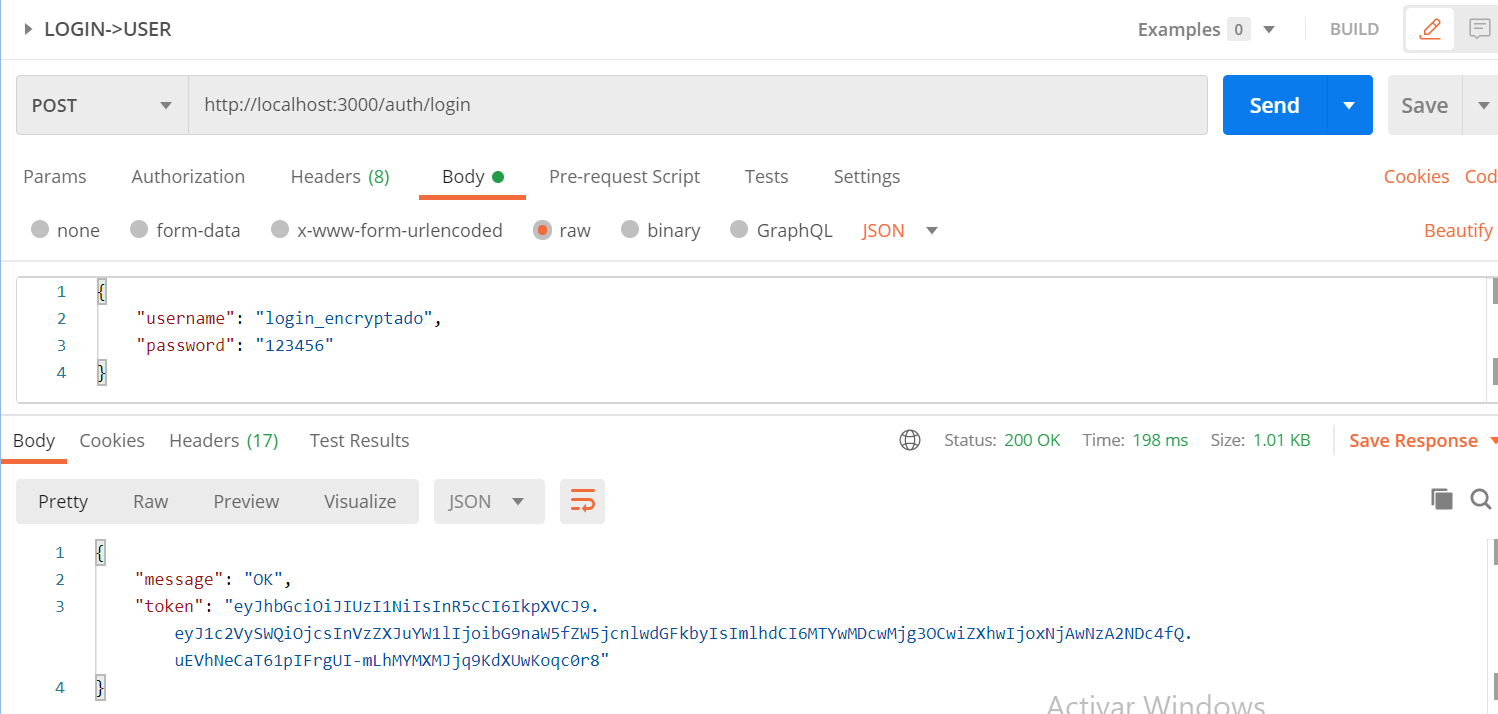
1. Para evitar que extraiga toda la información del usuario en el front hacemos los siguientes cambios en el doc Controller/AuthController.ts



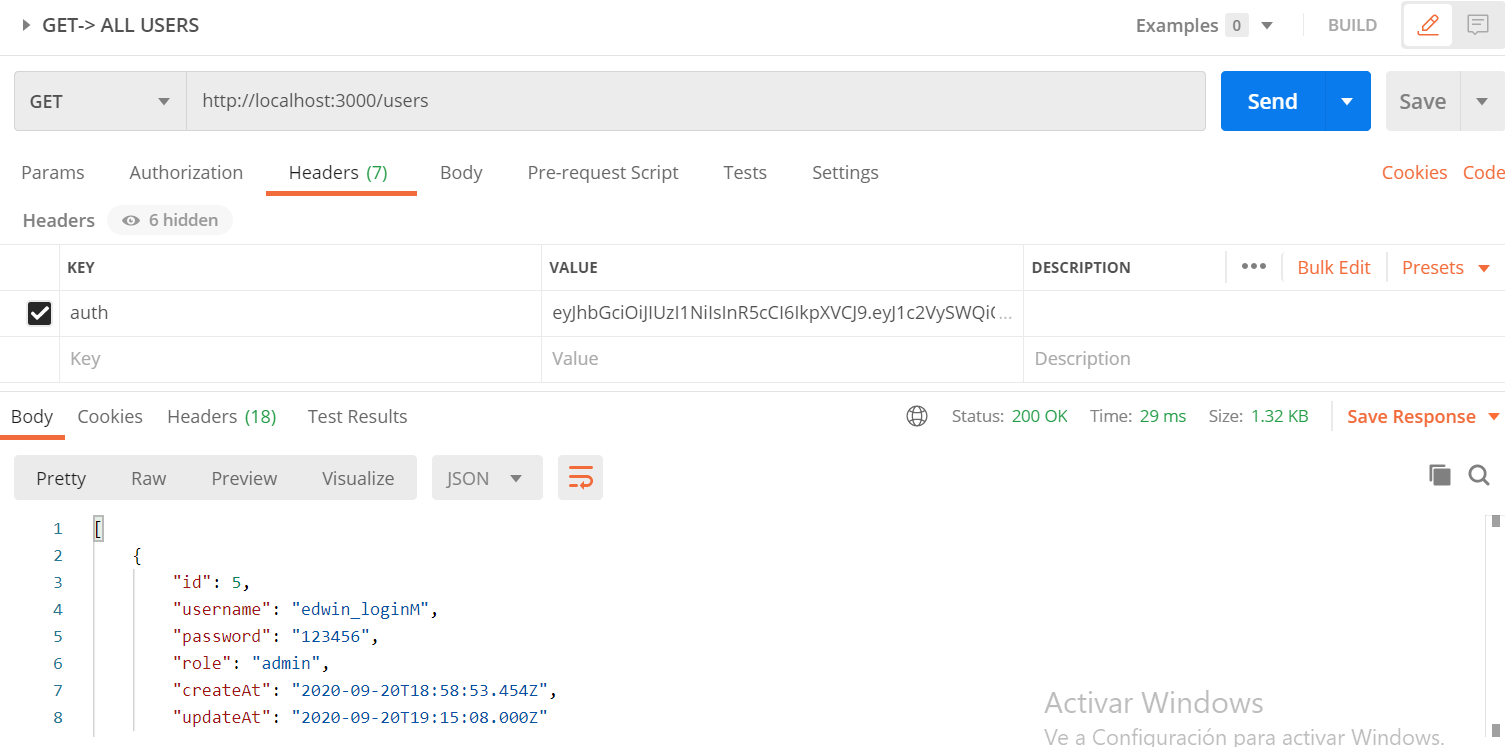
Si se genera otro import de config eliminarlo

Y después de la condicional IF colocar el siguiente codigo

1. En el POSTMAN en el LOGIN enviamos nuevamente la misma solicitud de logeo, y se nos genera lo siguiente



1. Se copia el token generado en el método GET ALL pegarlo en la pestaña de headers como se ve acontinuacion:



Se pega el token aca

Se agrega esto

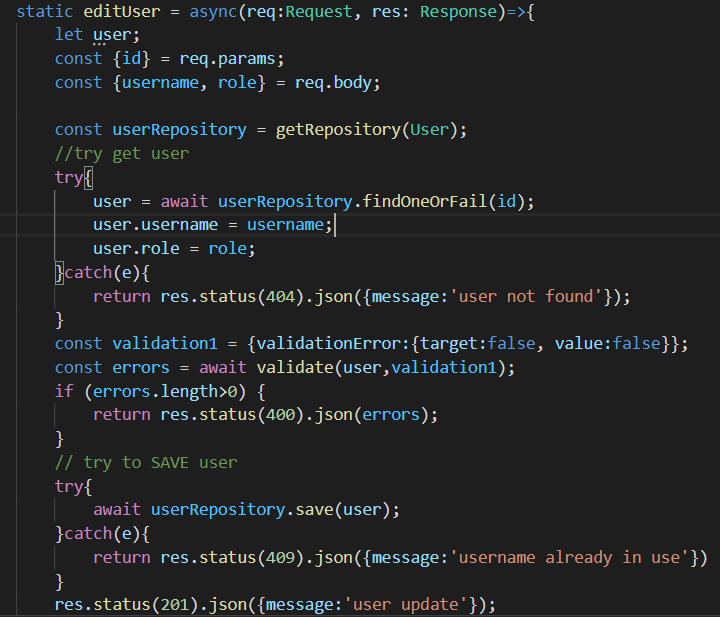
Ahora en vez de salir “No Autorizado” muestra los usuarios registrados

RECORDAR ESTO SE HIZO PARA EL METODO GET 🡪ALL

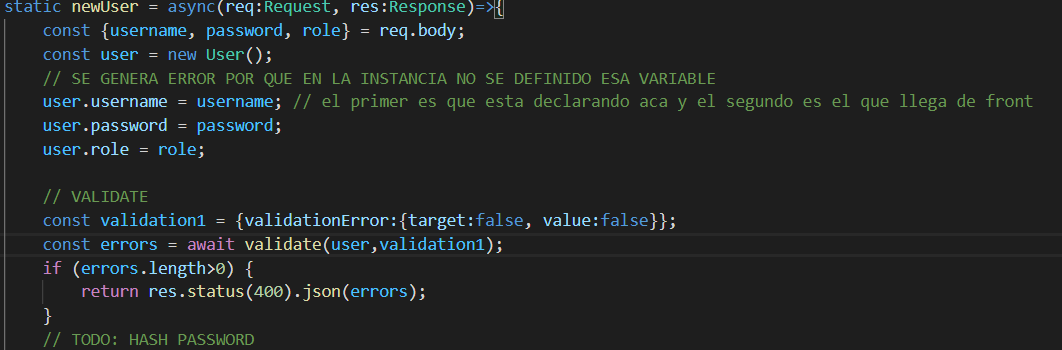
1. Ahora para realizarlo para un id en particular con el METODO POR ID,DELETE, Y LOS DEMAS METODOS, se realiza con agregar lo sig:



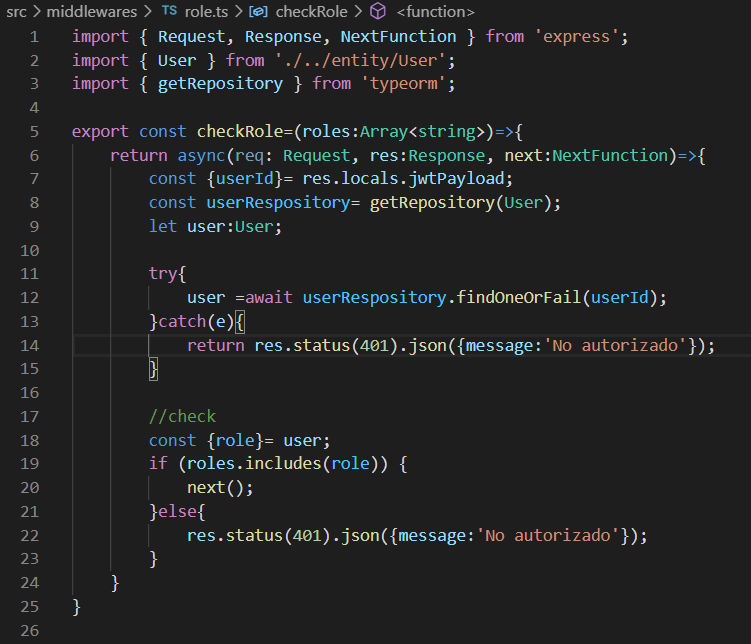
1. Hay un problema de que cuando vamos a editar un registro pero si lo dejamos en blanco, nos mostrara la información del usuario, para evitar esto se realiza lo siguiente

En el doc Controller/UserController🡪 en el método editUser, donde se agrega el parámetro validation1

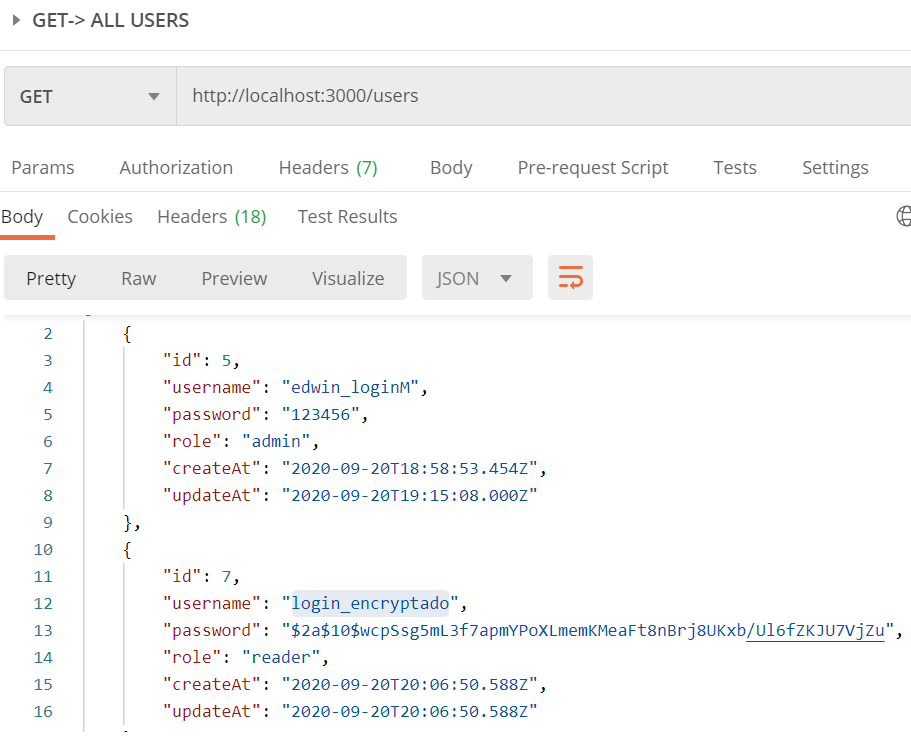
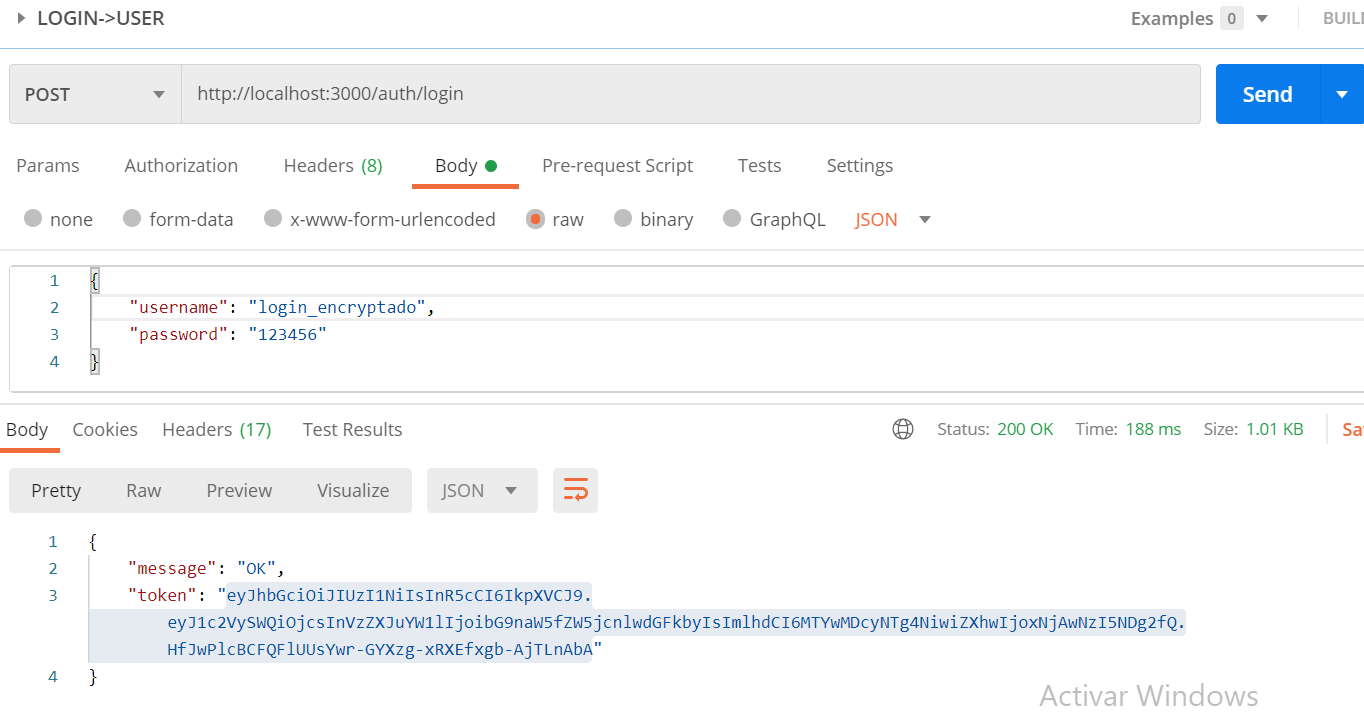
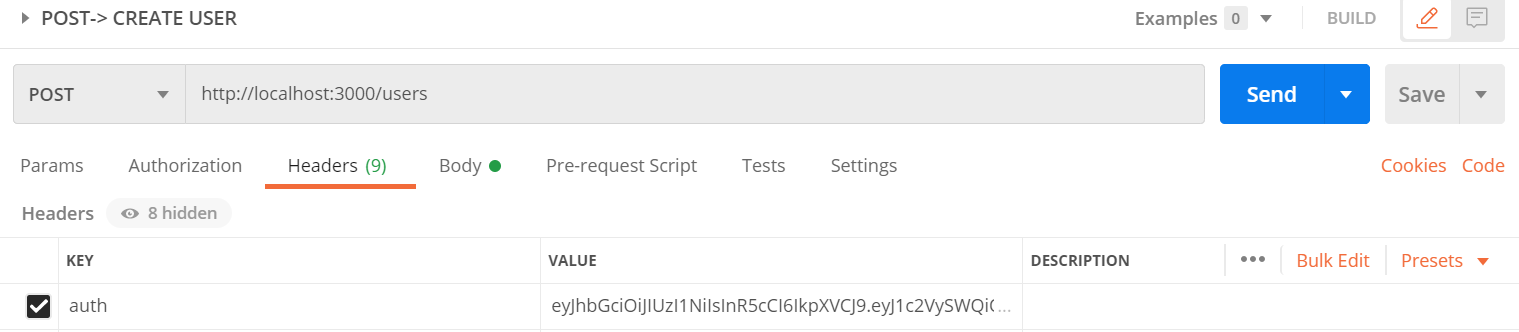
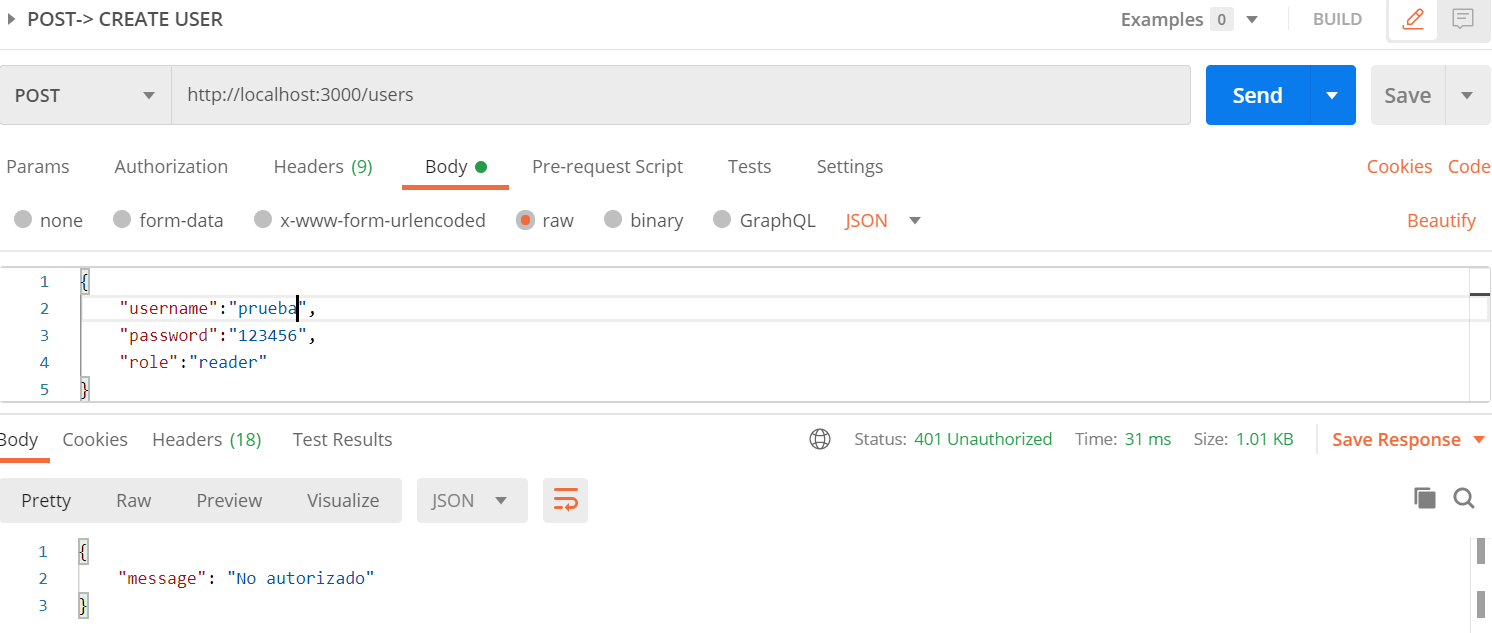
Ya que tanto el target como el value, son los datos que nos mostraban la contraseña

1. Se realiza el mismo proceso para el método newUser 
2. Como se indico en el paso 38, ahora si se configurara los permisos de admin o invitado ya que se ha configurado el token, para ello se crea un doc la carpeta middlewares 🡪 con el nombre role.ts

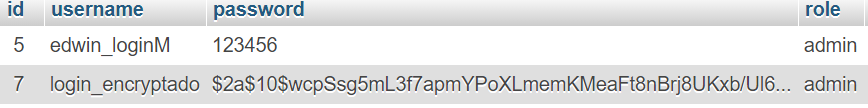
Con este código indicara y comparara si esta autorizado a realizar CRUD llamándolo en el punto que sigue(punto 46)

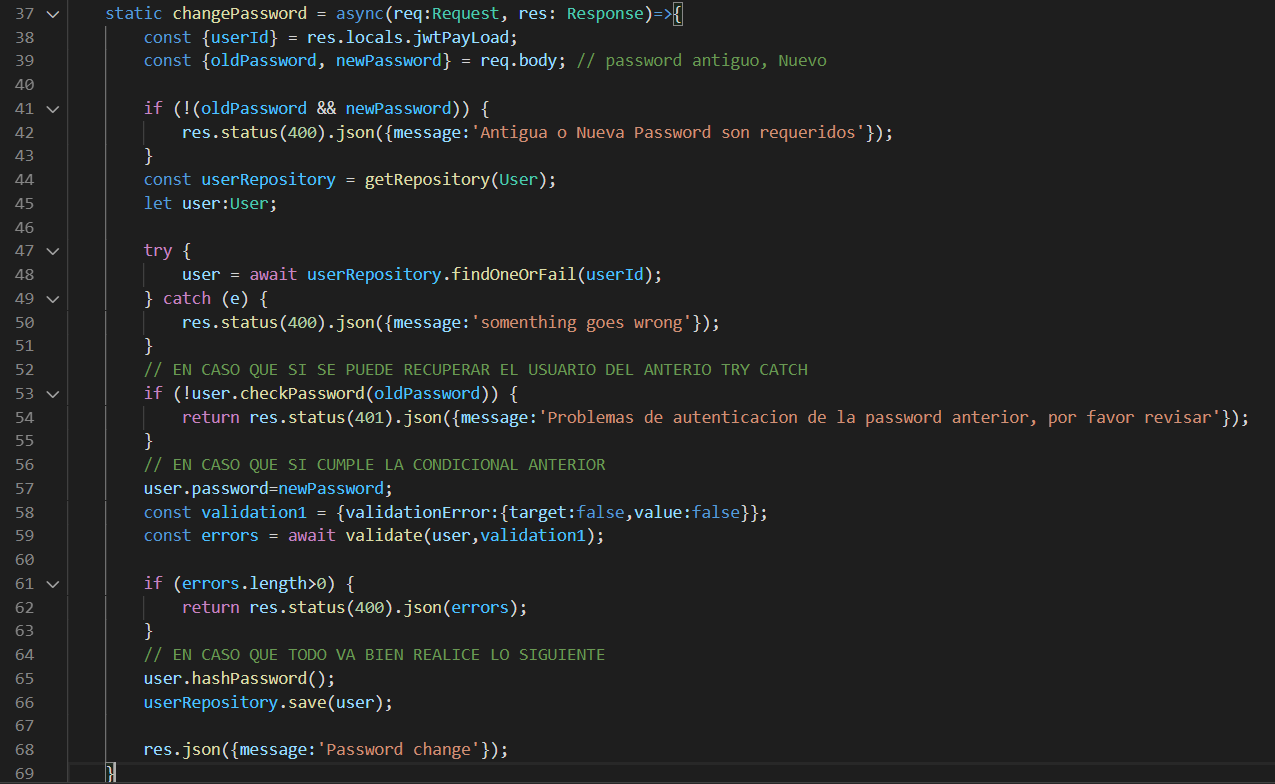
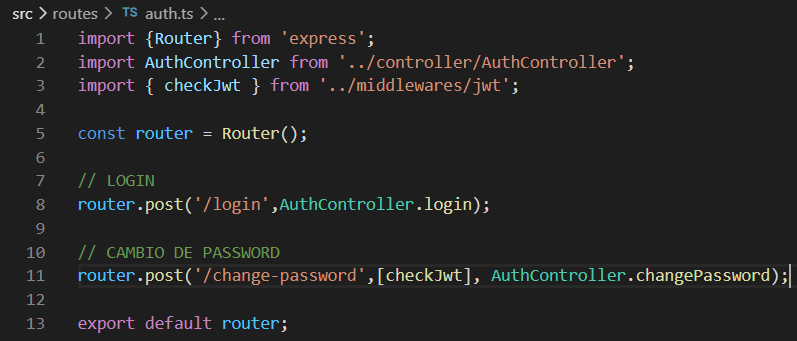
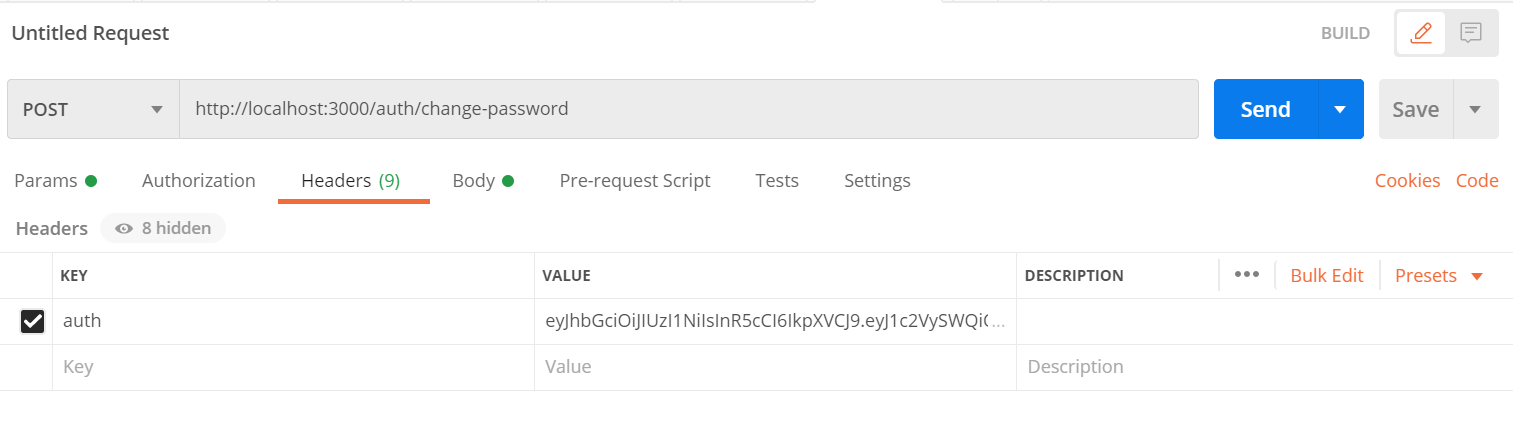


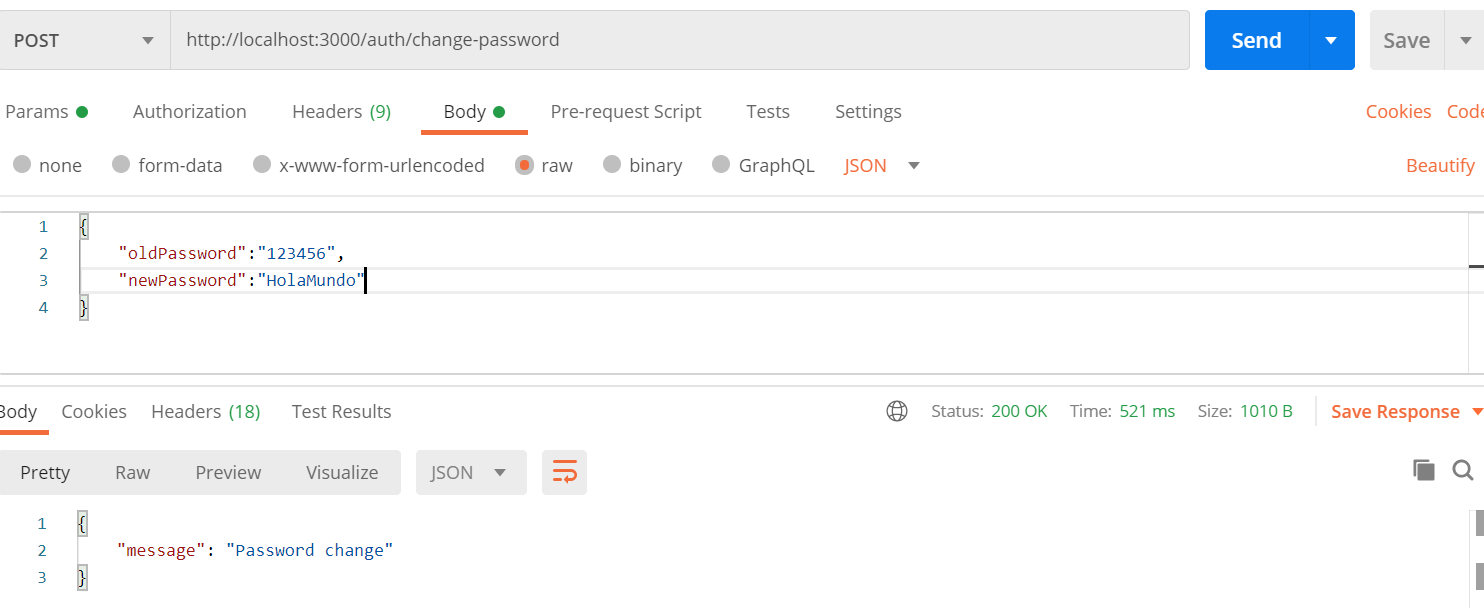
1. Para indicar en que método indicarle que tiene que tener permisos es en routes/user.tss
2. Para poder verificar si quedo bien se realiza de la siguiente manera.

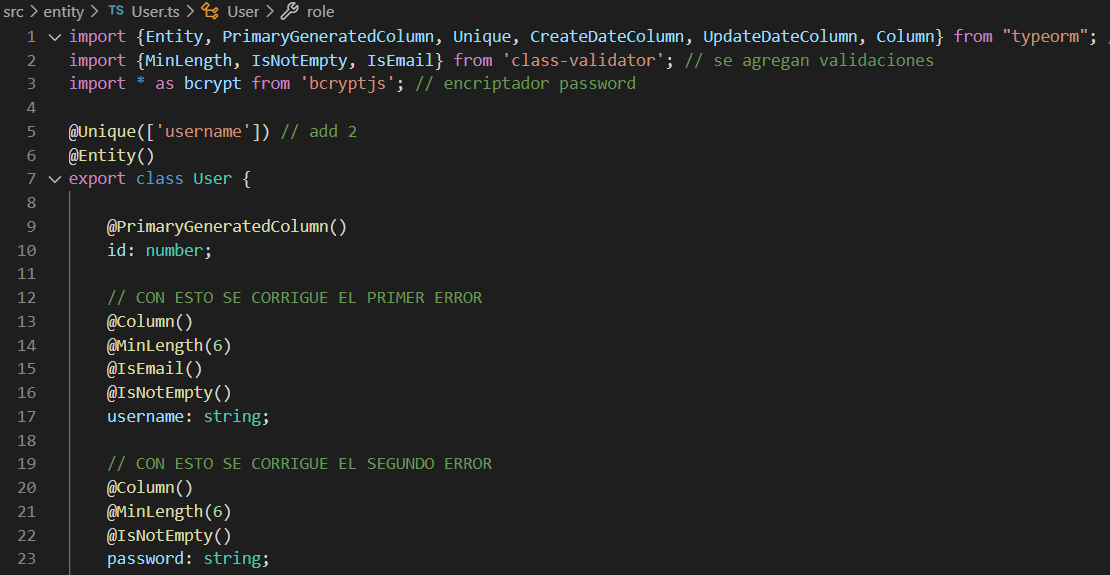
* Primero confirmar que usuarios tengo para ello revisamos en POSTMAN en el método GET ALL 
* En este caso verificamos con el rol de READER entonces utilizaremos el username🡪login\_encryptado y su password(esta encriptado)
* Nos vamos al método LOGIN e insertamos tanto el username como el password
* Copiamos el token generado y nos dirigimos al método CREAR USUARIO, en la pestaña de Headers en el key dejamos auth y en value pegamos el tocken
* Hay mismo en la pestaña de Body. Intentamos crear un nuevo registro (hay no importa que rol crearemos) y enviamos solicitud y observemos que el resultado no esta autorizado

1. Para continuar con las pruebas en el localhost cambiamos ese usuario a admin



1. Y realizamos todos los paso del punto 47. Donde efectivamente si deja crear el usuario
2. Una vez verificado se procede a realizar en los demás métodos como se ve a continuación:
3. Esto es opcional, dejar que los usuarios cambien de password, para ello en el archivo controller/AuthController.ts. se crea un nuevo metodo
4. Por ultimo para dar la autorización final en el doc routes/auth.ts
5. Ahora probarlo en el POSTMAN, copiamos el token del METODO LOGIN y creamos un nuevo método ya que la Url es distinta

En la pestaña Body colocamos tanto el password actual y la nueva contraseña

1. Opcional: para indicarle que el usuario sea email se realiza de la sig manera en el doc entity/user.ts 

<https://www.youtube.com/watch?v=8si7IFlj0l4> 1h 38m

<https://www.youtube.com/watch?v=uMFlyZfdFXo> angular

<https://logisticatransporteycarga.com/>

<https://packen.co/home>

<https://carryexpress.com.co/trabajaconnosotros/>

https://ventascontraentrega.com/