Autenticacion

<https://docs.nestjs.com/techniques/authentication>

En principio no vamos tomar la información de los usuarios desde la BD. En vez de esto lo vamos a hacer desde el mismo código, como un arreglo de objetos usuario que contendrán la información de los diferentes usuarios userid, username y password.

Entonces vamos a implementar un modulo users y dentro del modulo además del archivo module.ts generaremos un servicio que contendrá la clase UsersService en el archivo users.service.ts.

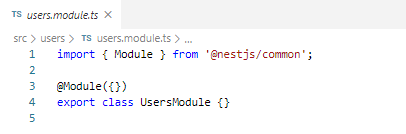
Dentro del Servicio vamos a definir un atributo users que será el arreglo de usuarios que inicializaremos directamente en el constructor de la clase con los diferentes usuarios. La clase tendrá un método findOne() que recibe el nombre del usuario que intenta loguearse y si lo encuentra en el arreglo de usuarios, retorna la información del usuario.

1. Creamos un modulo usuario usando los comandos de nest

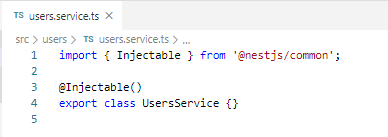
nest generate module users

nest generate service users

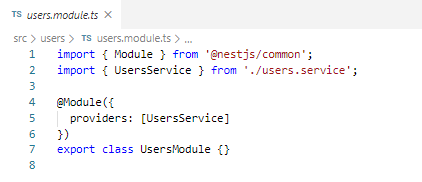
el primero creara la carpeta \users para dicho modulo y el archivo user.module.ts.



El siguiente creara el archivo user.service.ts



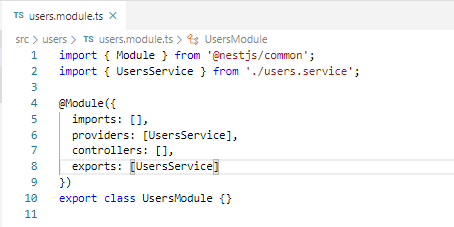
y colocara lo necesario en el user.module.ts



dentro del users.service.ts vamos a definir la clase UsersService y en su constructor vamos a agregar unos usuarios

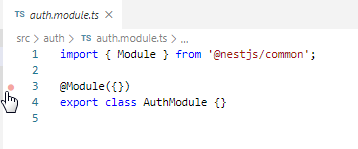


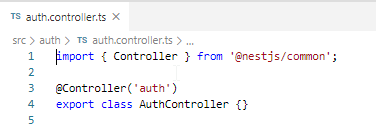
mientras que en el users.module.ts vamos a agregar el arreglo exports[] al que asignamos UsersService. Imports y Controllers los agrego pero no llevan nada. Y en providers agregamos también a UsersService

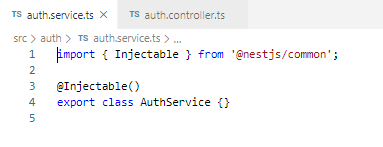


1. Ahora creamos un modulo de autenticación que llamaremos auth de la misma manera que como lo hicimos con el módulo users, con los comandos de Nest.js para generar el módulo auth.module.ts, otro para generar el controlador auth.controller.ts y otro para generar el servicio auth.service.ts

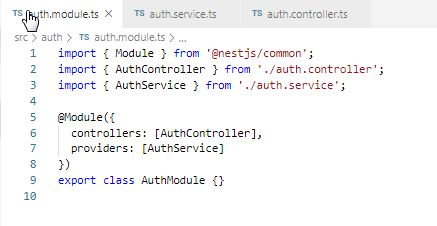
(<https://www.youtube.com/watch?v=d5PrYsHMvNs>)







a esta altura auth.module.ts ya involucra al controlador y al servicio



Se van a necesitar incorporar más cosas dentro del auth.module.ts todas aquellas cosas que necesitemos inyectar dentro del servicio, como veremos mas adelante, por ejemplo las diferentes estrategias deberan incorporarse en el arreglo de providers dentro de este archivo. Tambien módulos como el de users, el de PassportModule y el JwtModule se incorporaran al arreglo Imports en el auth.module.ts. Pero primero va a ser necesario incorporar algunas dependencias.

Para llevar a cabo la autenticación va a ser necesario agregar unas dependencias, una serie de librerias para implementar el modulo de autenticación, librerías que nos van a ayudar

Arranquemos con Passportjs y JWT

**Passport** es el middleware de autenticación para Node.js. Extremadamente flexible y modular, Passport se puede colocar discretamente en cualquier aplicación web basada en Express. Consiste de un conjunto completo de estrategias . Del conjuntpo de estrategias vamos con dos de ellas Passport-local y Passport-jwt (<http://www.passportjs.org/packages/>)

Es la biblioteca de autenticación de node.js más popular, conocida por la comunidad y utilizada con éxito en muchas aplicaciones de producción. Es sencillo integrar esta biblioteca con una aplicación **Nest** usando el módulo @nestjs/passport.

Passport tiene un rico ecosistema de **estrategias** que implementan varios mecanismos de autenticación.

**Passport-local** para autenticar con un nombre de usuario y contraseña. Este módulo permite autenticarse usando un nombre de usuario y contraseña en aplicaciones Node.js. Al conectarse a Passport, la autenticación local puede integrarse fácil y discretamente en cualquier aplicación o marco que admita el middleware de estilo Connect , incluido Express.

**Passport-jwt** para autenticarse con un token web JSON .

Este módulo le permite autenticar puntos finales utilizando un token web JSON. Está destinado a usarse para asegurar puntos finales RESTful sin sesiones.

Por otro lado habrá que instalar **JsonWebToken** para realizar request de forma segura entre dos partes. Esta nos permite decodificar, verificar y generar JWT (<https://jwt.io/>)

Tanto para Passport como para JsonWebToken necesitaremos instalar además las librerías propias de de Nestjs es decir **@nestjs/passport** y **@nestjs/jwt**

Si vamos a hilar un poco más fino guardando las pwd encriptadas de los usuario podemos necesitar importar bcrypjs que es una librería para encriptar contraseñas

npm i passport @nestjs/passport @nestjs/jwt

npm i jsonwebtoken

npm i passport-jwt passport-local

npm i bcrypt

para desarrollo (-D) vamos a instalar algunas dependencias que nos van a ayudar con el autocomplete de los tipos de las librerías anteriores, durante la codificación,

npm i -D @types/passport @types/passport-jwt @types/passport-local

npm i -D @types/jsonwebtoken

npm i -D @types/bcrypt

dentro de \auth vamos a crear las diferentes estrategias.

Una estrategia no es más que una forma en la que nuestro “Passport” va a validar nuestra identificación para autenticarnos en el uso de la aplicación. Puede haber varias estrategias, quizá mañana se nos ocurra crear una autenticación a través de Google, Twitter o Instagram, solo será necesario crear una nueva estrategia y no reescribir todo el código.

# Estrategias

Creamos una estrategia para JWT.

Para esto vamos a crear una clase **JwtStrategy** que vamos a exportar lo hacemos en un archivo **jwt.startegy.ts**. esta clase va a extender a **PassportStrategy()** por lo cual va a ser necesario importar esta función del modulo **@nest/passport**. Esta función recibe como parámetro una **Strategy** que a su vez debemos importar del modulo **passport-jwt**.

Como estamos definiendo una clase vamos a definir el **constructor** para la misma y como estamos heredando vamos a tener que llamar al **super()**. Al super hay que pasarle un argumento que será un objeto que tendrá la siguiente forma

{

jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),

secretOrKey: jwtConstants,

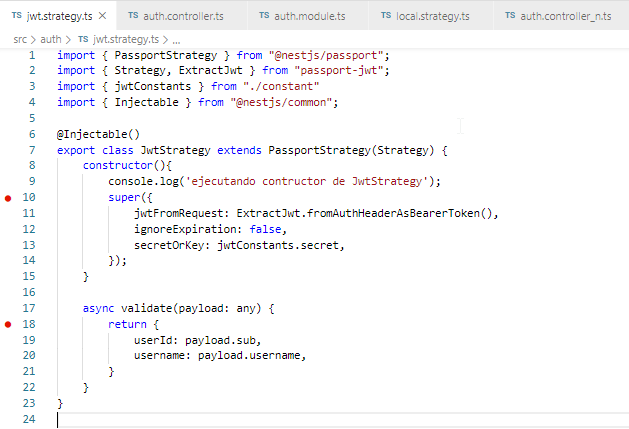
ignoreExpiration: false,

}

* un atributo que se llame **jwtFromRequest** (de donde va a venir nuestro token) y al que se le especifica que lo extraiga del header utilizando el método **ExtractJwt.fromAuthHeaderAsBearerToken()**, método que se importa del modulo passport-jwt
* otro atributo **secretOrKey**: que es la semilla que se utiliza para generar el token en nuestro caso la importaremos pór ahora de un archivo auth\constant.ts
* otro atributo ignoreExpiration, chequea automáticamente por la expiración del token, si esta seteado en true no valida la expiración

nuestra estrategia lo único que va a tener es un método que se va a llamar **validate()** que se va a encargar de validar que ese usuario exista (si no existe no estará autorizado) Creamos un método asíncrono que va a recibir nuestro payload con los datos del userId,

username (en nuestro caso los usuario no están en la BD sino que están implementados en un archivo user.service.ts en el modulo users)



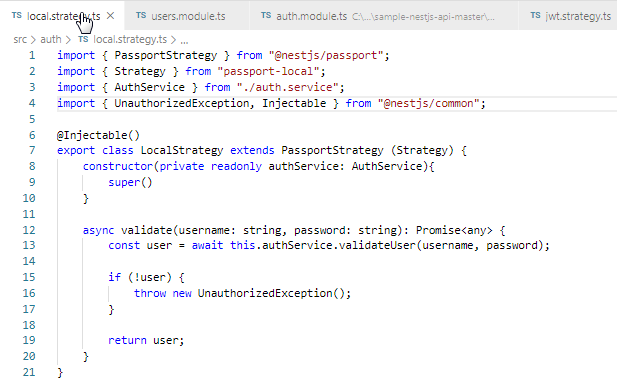
### Creamos una estrategia Local

Para esto vamos a crear una clase **LocalStrategy** que vamos a exportar lo hacemos en un archivo **local.startegy.ts**. esta clase va a extender a **PassportStrategy()** por lo cual va a ser necesario importar esta función del modulo **@nest/passport**. Esta función recibe como parámetro una **Strategy** que a su vez debemos importar esta vez del modulo **passport-local**.

Como estamos definiendo una clase vamos a definir el **constructor** para la misma y como estamos heredando vamos a tener que llamar al **super()**. Al super hay que pasarle un argumento en este caso sera un objeto de la clase AuthService

private readonly authService: AuthService

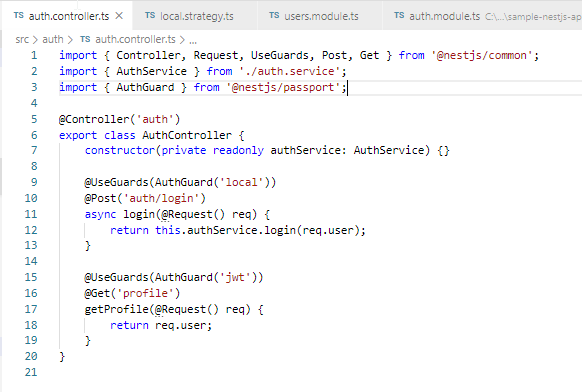
como en el caso de la estategia anterior, esta también contendrá un método **validate()**, en este caso el método llamara al método **validateUser()** del objeto inyectado **authService** (es decir del servicio del modulo de autorización). Si el usario existe retornará el usario, en otro caso lanzará una excepción de indicando la NO autorización para el ingreso.



# auth.controller.ts y auth.service.ts

En el controlador auth.controller.ts vamos a definir la clase **AuthController** y en su constructor inyectamos un objeto del tipo **AuthService**, a través de este último es desde el cual se invocará su método login().

El controlador define el endpoint /auth y posee un método http POST que efectuara el login del usuario al llamar al método login() de AuthService y un método http GET. Requiere una serie de decoradores



El servicio auth.service.ts va a tener dos simples responsabilidades

* hacer el login del usuario (validar usuario)
* registrar al usuario (y en el caso que lo requiera asignarle un rol al mismo)

.Se crea la clase AuthService esta clase posee dos atributos que se inyectan en el constructor de la clase

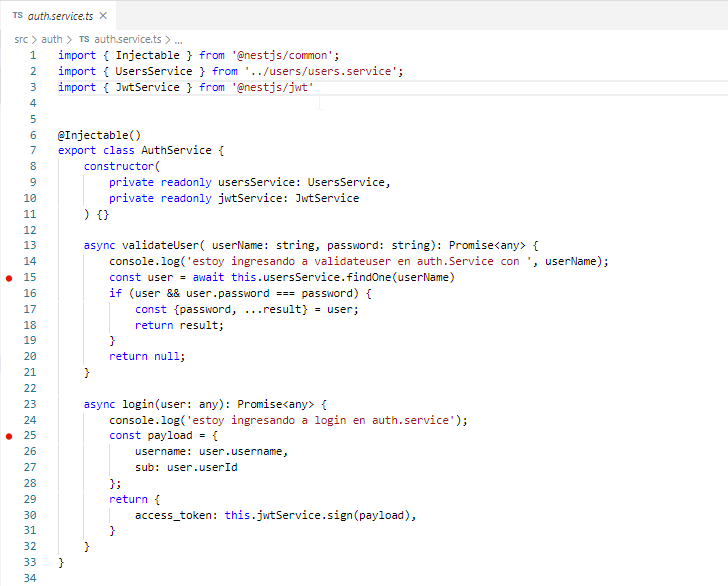
* un objeto **userService** de tipo **UsersService** el que nos va a servir para buscar el usuario que se loguea
* un objeto **jwtService** de tipo **JwtService** para generar un token y manejar la autorización al acceso a las rutas

que se inicializan en su constructor.

Y posee dos métodos

* **validateUser**(username, pwd) que valida que el usuario que se loguea está dentro de los usuarios autorizados
* **login**(user) con el que le otorga un token para acceder a las diferentes rutas

tener en cuenta que en este caso no buscamos los usuarios en la BD para efectivizar el login, si así lo hiciésemos debiéramos de utilizar un repository (ORM) y en ese caso en el contructor en ves de imntectar el UsersModule como lo hacemos en este caso estaríamos inyectando un repository



El auth.module.ts como se mencionó requiere de algunas configuraciones “cordiales” para que nuestras estrategias y autenticación cobren sentido.

Importar las estrategias JwtStrategy y LocalStrategy del os archivos jwt.staregy.ts y local.strategy.ts respectivamente (en el tutorial de autenticación indicaban también incorporarlas al arraglo exports).

Importar el PassportModule de @netsjs/Passport (en el tutorial que vi indican también exportarlo dentro del arreglo exports).

Configurar algunos de los servicios que se utilizan, esto se hace en el arreglo imports.

* Alli agrego el **UsersModule** (en este caso porque los usuarios los cargue directamente en el código en el módulo users, en el caso que lo hubiese hecho desde la BD usando un repository, en este lugar debiera estar configurando el repository).
* Configuro el **PassportModule**. En este caso solo eso, pero puedo usar su método register() e indicarle con un argumento object que posee la propiedad defaultstrategy: la estrategia que voy a utilizar por defecto por ejemplo jwt

PassportModule.register( {defaultStrategy: ‘jwt’} )

* Configuro el **JwtModule**. Uso la funcion register() para indicarla a través de un objeto que posee dos propiedades
  + una propiedad **secret**: a la que le doy el valor de la semilla guardada en el objeto jwtConstants en el archivo ./constants.ts (que también debo importar al modulo)
  + otra propiedad signOptions: a la que le doy un objeto {espiresIn: ‘60s’}

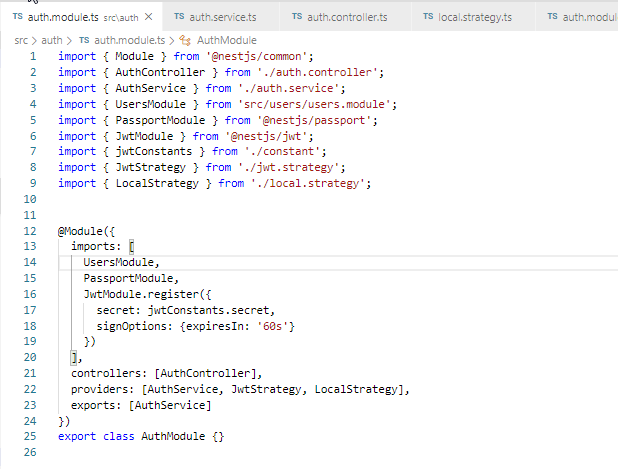
JwtModule.register({

secret: jwtConstants.secret,

       signOptions: {expiresIn: '60s'}

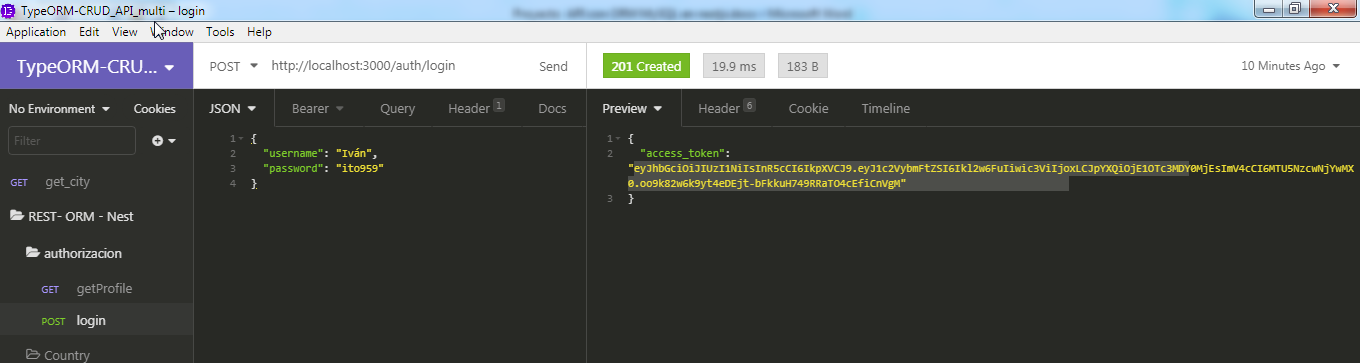
})

Para ver la configuración de estos modulos habrá que referirse a la documentación ya que exsten varias formas de hacerlo de acuerdo a lo que uno pretenda



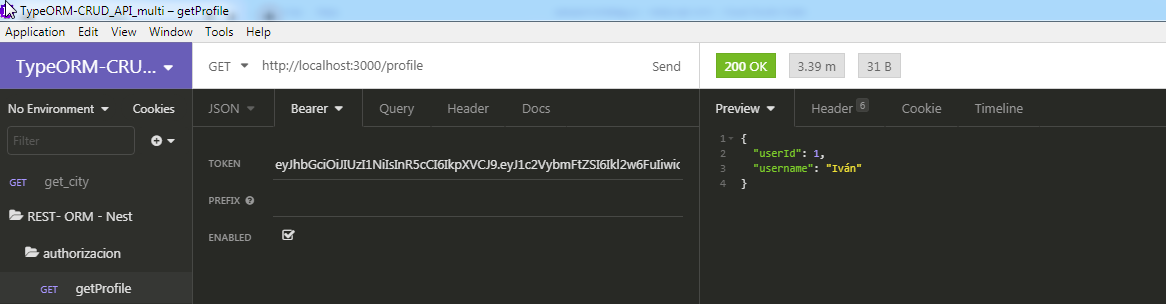
La pruebas de la autenticacion se llevan a cabo desde Postman o insomnia.

Definimos un POST para la ruta <http://localhost:3000/auth/login> y en JSON pasamos u objeto json con el usario y la password. Al enviar la solicitud nos devolverá un jwt sobre la solapa Preview.

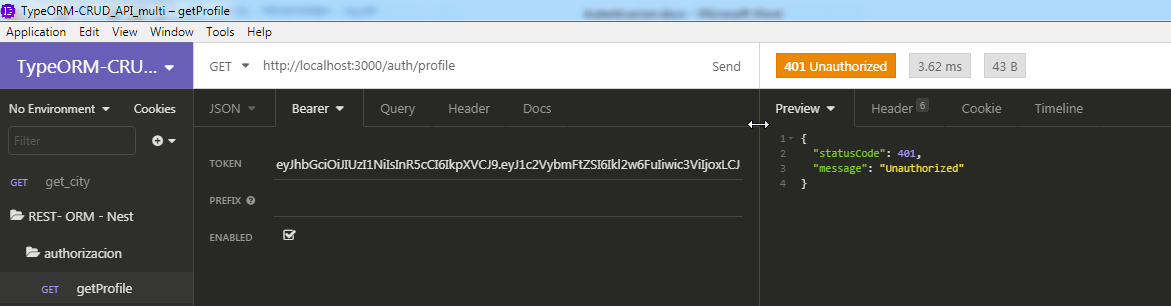


Definimos un GET sobre la ruta <http://localhost:3000/auth/profile> y en la solapa Autorizacion seleccionamos Bearer y pegamos el token que nos respondió el POST, en teoría si aplicacmos el GET dentro de los 60 segs que se especificaron para debiera de retornar algo como

result -> {"userId":1,"username":"Iván"}



o en caso de superar el timeout un error 401 unauthorizated



https://www.youtube.com/watch?v=d5PrYsHMvNs&t=34s

https://www.youtube.com/watch?v=obszPxRF67Q

https://www.youtube.com/watch?v=i-C6zCOvQBI

https://www.youtube.com/watch?v=UahtyzXp\_LA

https://www.youtube.com/watch?v=0sMYa2yQOpY&t=11s

https://www.youtube.com/watch?v=HflUR73LoO0

<https://www.youtube.com/watch?v=gEpAlbT40hE>