**Documentación Adicional**

**Gestión de proyecto y Dependencia**

Se creó el proyecto en visual studio code utilizando npm para las dependencia. Se describen los comandos necesarios de la instalación y las librerías.

-npm install -g @nestjs/cli

- nest new usuarios-api

-cd ./usuarios-api

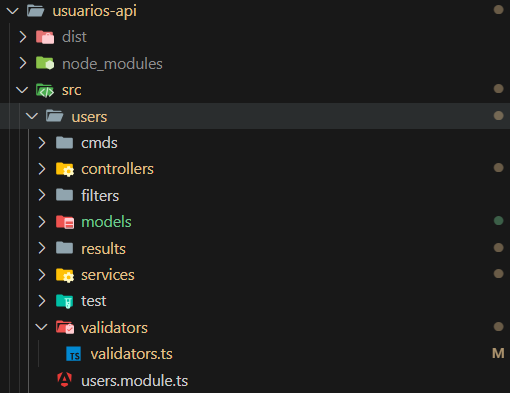
- npm install @nestjs/mongoose mongoose

-npm i install

-npm run start

- docker run -p 3000:3000 usuarios-api

**Estructura del repositorio**



Partimos del directorio raíz usuarios-api, donde se creó la carpeta users donde contendrá todo lo necesario para la implementación de la API ERST.

Cmds: contiene en diferentes archivos las propiedades que se necesitan ser pasadas por parámetro tanto en el controllers y services.

Filters: contiene en diferentes archivos las propiedades que se necesitan ser pasadas por parámetro tanto en el controllers y services, que vendrían a través del body.

Controllers: contiene todas las rutas asociadas a cada servicio.

Models: contiene los esquemas necesarios que se puedan crear con las propiedades que se necesiten.

Result: contiene lo que espera devolver, cuando un método de service llama al mismo.

Test: contiene pruebas unitarias tanto como para el controlador como para el servicio.

Validators: contiene validaciones que se ejecutan en los métodos.

**Uso de los métodos con postman**

Endpoint create:

Se envía una solicitud POST a la URL http://localhost:3000/users con los datos del nuevo usuario en el cuerpo de la solicitud (body).

El servidor recibe estos datos, crea un nuevo objeto de usuario, le asigna un ID único, lo almacena en un array en memoria y te devuelve el usuario recién creado como respuesta

-Método: Post

-URL: <http://localhost:3000/users>

-Body(JSON): {

    "name": "jorge ",

    "email": "jorge@gmail.com",

    "age": 14

}

Resultado: {

"id": 1,

    "name": "jorge ",

    "email": "jorge@gmail.com",

    "age": 14

}

Endpoint findALL:

Se en una solicitud GET a la URL http://localhost:3000/users.El servidor busca todos los usuarios almacenados en la memoria y te devuelve un array con todos ellos.

-Método: GET

-URL: <http://localhost:3000/users>

Resultado: [{

"id": 1,

    "name": "jorge ",

    "email": "jorge@gmail.com",

    "age": 14

}]

Endpoint findOne:

Se envía una solicitud GET a la URL http://localhost:3000/users/{id} donde {id} es el identificador único del usuario.El servidor busca el usuario en la memoria que coincide con el ID proporcionado y te lo devuelve como respuesta.

-Método: GET

-URL: <http://localhost:3000/users/1>

Resultado: {

"id": 1,

    "name": "jorge ",

    "email": "jorge@gmail.com",

    "age": 14

}

Endpoint Update:

Se envía una solicitud PUT a la URL http://localhost:3000/users/{id}, donde {id} es el identificador del usuario a actualizar. En el body de la solicitud, los nuevos datos del usuario.

El servidor busca el usuario correspondiente, actualiza sus datos y devuelve el usuario actualizado como respuesta, si es que lo encontró

Método: PUT

-URL: <http://localhost:3000/users/1>

-Body(JSON):

{

    "name": "jorge ",

    "email": "jorge@gmail.com",

    "age": 14

}

Resultado: {

"id": 1,

    "name": "jorge ",

    "email": "jorge@gmail.com",

    "age": 14

}

Endpoint Delete:

Se envía una solicitud DELETE a la URL http://localhost:3000/users/{id}, donde {id} a eliminar.El servidor busca el usuario en la memoria, lo elimina .

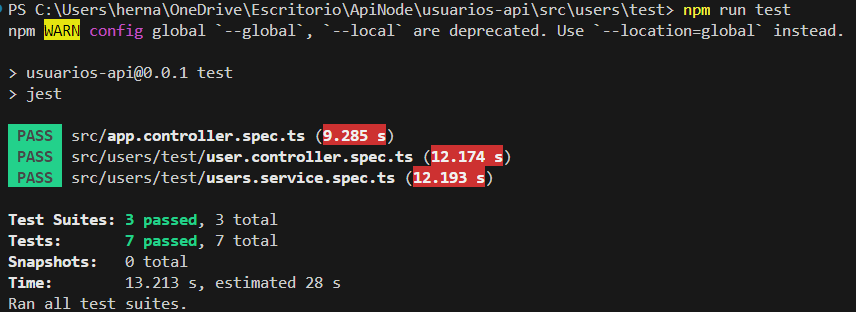
Método: PUT

-URL: <http://localhost:3000/users/1>

**Pruebas Unitarias**

Se realizaron algunas pruebas unitarias para lógica negocio asociado al método créate.En el código se pueden encontrar en la siguiente ruta usuarios-api/src/users/test. Se las puede ejecutar con el comando npm run test.

Se adjuntan los resultados de las pruebas.



**Observaciones**

No se pudo generar el Docker file solicitado por cuestiones del sistema operativo de uso(Windows), por inconvenientes con WSL, no se pudo correr docker y por lo tanto no generar la imagen y completar con lo pedido. De todas maneras dejo los comando que hubiera utilizado y los archivos de docker

* docker build -t usuarios-api:latest .
* docker run -p 3000:3000 usuarios-api