

Universidad Tecnológica de Xicotepec de Juárez

Propuesta de mecanismo de seguridad

Integrantes del Equipo:

David García Olivares

Adolfo Meléndez Rodríguez

Sandra Garrido Garrido

Soledad Reyes Martínez

Claribel Trejo Ocampo

Para crear nuestra API, utilizamos nodejs, este es un framework que está diseñado para crear aplicaciones escalables, permitiendo, establecer y gestionar múltiples conexiones al mismo tiempo.

Node.js fue creado por los desarrolladores originales de JavaScript. Lo transformaron de algo que solo podía ejecutarse en el navegador en algo que se podría ejecutar en los ordenadores como si de aplicaciones independientes se tratara. Gracias a Node.js se puede ir un paso más allá en la programación con JavaScript no solo creando sitios web interactivos, sino teniendo la capacidad de hacer cosas que otros lenguajes de secuencia de comandos como Python.

Malas y Buenas practicas al desarrollar la API

Al desarrollar la Api, tratamos de seguir todas las buenas practicas que encontrábamos, esto para que nuestro código quedara lo más limpio posible y fuera fácil de entender, y es que un buen programador siempre debe seguir buenas practicas, a continuación, presentaremos algunas de las buenas y malas practicas que tuvimos dentro del código.

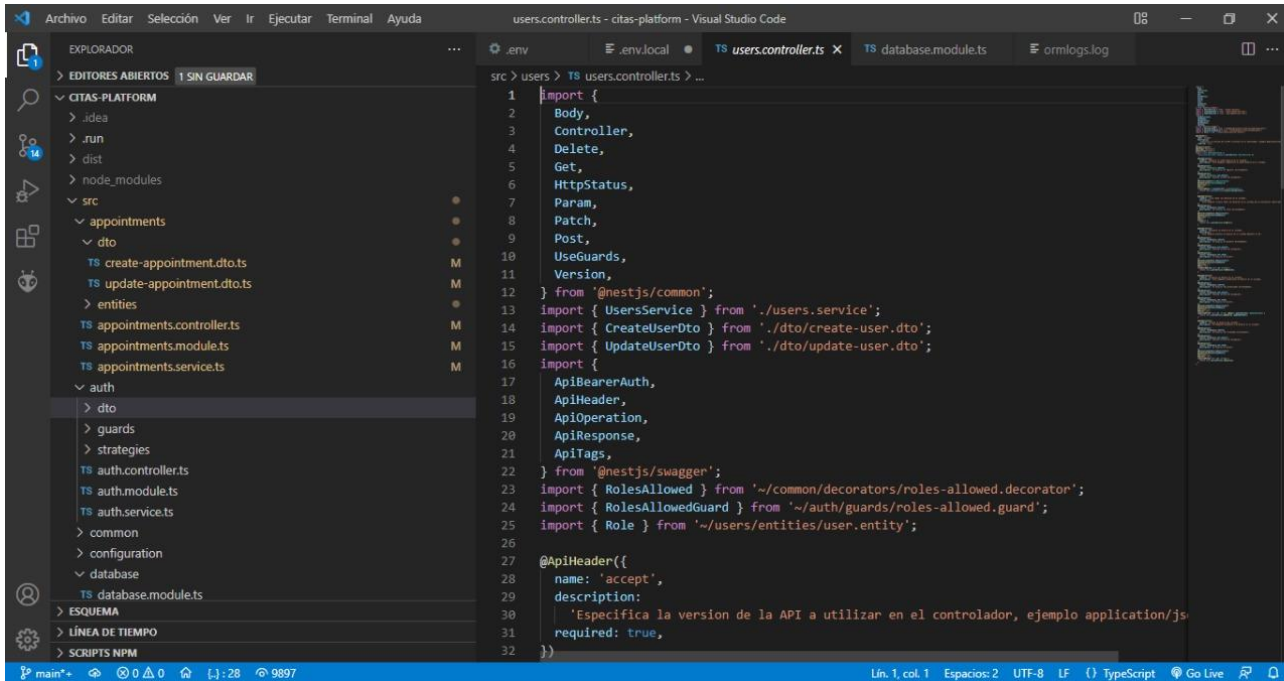
Una mala práctica es no colocar espacios en tu código o dar saltos , ya que todo te queda en una sola línea asiendo de esto un código confuso

```
if (optional > 0)
  throw new ConflictException(`Ya existe una cita registrada para ${createAppointmentDto.procedure} a la hora indicada`);
```

Para poder solucionarlo, se deben de dar unos saltos de línea a tu código, de esta manera lograras tener tu código más limpio y mucho más entendible

```
if (optional > 0)
  throw new ConflictException(
    `Ya existe una cita registrada para ${createAppointmentDto.procedure} a la hora indicada`,
  );
```

Una buena práctica y que siempre se debería utilizar es la separación de módulos, a esto se le llama programación modular y es que así, si tienes algún fallo dentro de la Api, será mucho más fácil saber dónde está tu error que si estuviera todo junto en una misma carpeta



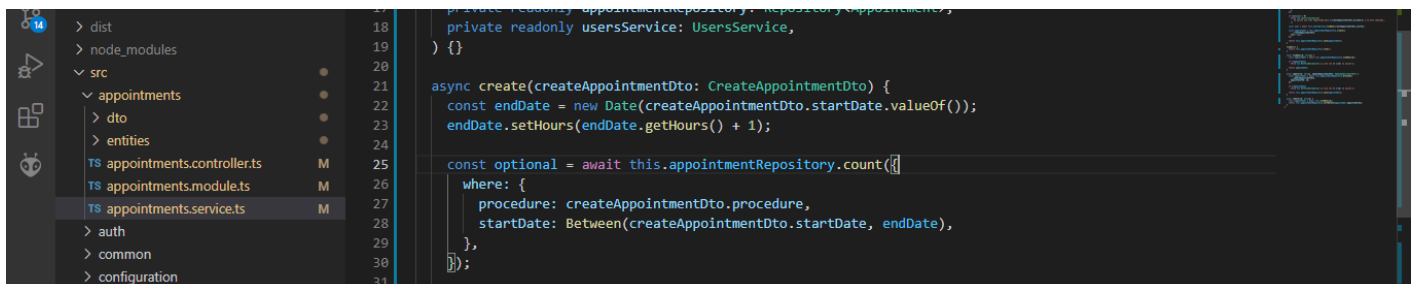
```
1 import {  
2   Body,  
3   Controller,  
4   Delete,  
5   Get,  
6   HttpStatus,  
7   Param,  
8   Patch,  
9   Post,  
10  UseGuards,  
11  Version,  
12 } from '@nestjs/common';  
13 import { UsersService } from '../users.service';  
14 import { CreateUserDto } from '../dto/create-user.dto';  
15 import { UpdateUserDto } from '../dto/update-user.dto';  
16 import {  
17   ApiBearerAuth,  
18   ApiHeader,  
19   ApiOperation,  
20   ApiResponse,  
21   ApiTags,  
22 } from '@nestjs/swagger';  
23 import { RolesAllowed } from '../common/decorators/roles-allowed.decorator';  
24 import { RolesAllowedGuard } from '../auth/guards/roles-allowed.guard';  
25 import { Role } from '../users/entities/user.entity';  
26  
27 @ApiHeader({  
28   name: 'accept',  
29   description:  
30     'Especifica la version de la API a utilizar en el controlador, ejemplo application/js',  
31   required: true,  
32 })
```

Otra buena práctica es cuando se trata de hacer que el código sea lo más corto posible, ya que, si ahorramos línea de código, nuestro programa correrá con mayor facilidad que si se le meten más líneas de código



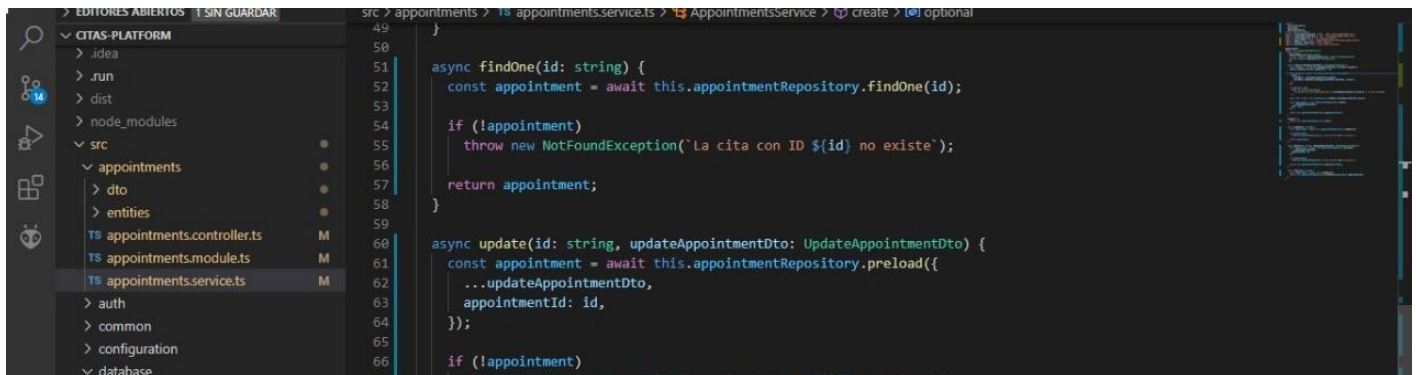
```
48 const user = this.userRepository.create(createUserDto);  
49 return this.userRepository.save(user);  
50 }  
51  
52 findAll() {  
53   return this.userRepository.find();  
54 }  
55  
56 async findOne(id: string) {  
57   const user = await this.userRepository.findOne(id);  
58  
59   if (!user)  
60     throw new NotFoundException('The user with id ${id} doesn't exists');  
61  
62   return user;
```

Otra buena práctica es realizar métodos de tipo asíncrono, la asincronidad nos va a permitir ejecutar nuestros procesos en varios hilos de ejecución o lo que se conoce como multithreading, esto nos permitirá que sea ejecutado de distinta manera a como fue escrito



```
18 private readonly appointmentRepository: Repository<Appointment>;  
19 private readonly usersService: UsersService;  
20  
21  
22  
23  
24  
25 async create(createAppointmentDto: CreateAppointmentDto) {  
26   const endDate = new Date(createAppointmentDto.startDate.valueOf());  
27   endDate.setHours(endDate.getHours() + 1);  
28  
29   const optional = await this.appointmentRepository.count({  
30     where: {  
31       procedure: createAppointmentDto.procedure,  
32       startDate: Between(createAppointmentDto.startDate, endDate),  
33     }  
34   });  
35 }
```

Una buena práctica más crear métodos lo más cortos posibles, que sirven exactamente para lo mismo, pero que al mismo tiempo optimiza nuestro programa ya que de esta manera se ahorraran líneas de código



```
src > appointments > ts appointments.service.ts > AppointmentService > create > optional
49 }
50
51 async findOne(id: string) {
52   const appointment = await this.appointmentRepository.findOne(id);
53
54   if (!appointment)
55     throw new NotFoundException(`La cita con ID ${id} no existe`);
56   return appointment;
57 }
58
59
60 async update(id: string, updateAppointmentDto: UpdateAppointmentDto) {
61   const appointment = await this.appointmentRepository.preload({
62     ...updateAppointmentDto,
63     appointmentId: id,
64   });
65
66   if (!appointment)
```

Otra buena práctica, es heredar métodos lo que esto ara es optimizar el código, y además reutilizaremos código que ya tenemos, optimizando el código, además utilizamos el sofDelete el cual no elimina por completo el dato, solo lo oculta, ya que de esta manera si quieres llevar un archivo histórico podrás sacarlo pues no se eliminaran por completo los registros, solo se pondrá en una columna como oculto.



```
66
67   if (!appointment)
68     throw new NotFoundException(`La cita con ID ${id} no existe`);
69   const appointment: Appointment = await this.appointmentRepository.findOne(id);
70   return this.appointmentRepository.save(appointment);
71 }
72
73 async remove(id: string) {
74   const appointment = await this.findOne(id);
75   return this.appointmentRepository.softDelete(appointment.appointmentId);
76 }
77 }
```