NEST 03

Pipes

- Hay que implementar validaciones. En el controller que el id sea un número y en el servicio que el id exista y sea válido
- NEST tiene por defecto los pipes: transforman la data recibida en requests para asegurar un tipo, valor o instancia de un objeto
    - Transformar de un string a un numero, por ejemplo
- Hay varios integrados en NEST por defecto:
    - ValidationPipe -> validaciones
    - ParseBoolPipe -> de string a boolean
    - ParseFloatPipe -> de string a float
    - ParseIntPipe --> de string a entero
    - ParseArrayPipe--> de string a array
    - ParseUUIDPipe--> de string a UUID
- Los pipes transforman la data
- ParseIntPipe me va a servir para pasar de un string a un entero
- En el controlador, como segundo parámetro de @Params, le paso el pipe

```
import { Controller, Get, Param, ParseIntPipe } from '@nestjs/common';
import { CarsService } from './cars.service';

@Controller('cars')
export class CarsController {

   constructor(
    private readonly carsService: CarsService
   ){}

    @Get()
    getAllCars(){
        return this.carsService.findAll()
    }
    @Get(':id')
    getCarById( @Param('id', ParseIntPipe) id: number ){
        return this.carsService.findOneById(id)
    }
}
```

- Ahora si le paso un id no válido como 3ps, ya me devuelve un 400 Bad Request de la petición
- Pero si le mando un id válido pero que no existe, como el 4, me sigue mandando un status 200 aunque con el body vacío

Exception Filters

- Manejan los errores de código en mensajes de respuesta http. Usualmente NEST ya incluye todos los casos de uso comunes, pero se pueden expandir
- Algunos más comunes son:
    - BadrequestException
    - UnauthorizedException
    - NotFoundException
    - ForbiddenException
    - RequestTimeoutException
    - GoneException
    - PayloadTooLargeException
    - InternalServerErrorException
- Hay más...
- Para manejar si no existe el car por id usaré notFoundException

```
import { Injectable, NotFoundException } from '@nestjs/common';

@Injectable()
export class CarsService {
    private cars = [
        {
            id: 1,
            brand: 'Toyota',
            model: 'Corola'
        },
        {
            id: 2,
            brand: 'Honda',
            model: 'Civic'
        },
        {
            id: 3,
            brand: 'Jeep',
            model: 'Cherokee'
        }
    ]

    findAll(){
        return this.cars
    }
    findOneById(id: number){
        const car = this.cars.find( car => car.id === id)
        if( !car){
            throw new NotFoundException(`Car with id ${id} not found`)
        }
        return car
    }
}
```

- Esto manda un 404 cuando es un id válido que no existe

POST PATCH Y DELETE

- El post se utiliza generalmente para crear un recurso
- Para extraer el body de la petición se usa el decorador @Body()
- Pondré el body de tipo any ya que no se que viene en él

```
import { Body, Controller, Get, Param, ParseIntPipe, Post } from '@nestjs/common';
import { CarsService } from './cars.service';

@Controller('cars')
export class CarsController {

   constructor(
    private readonly carsService: CarsService
   ){}

    @Get()
    getAllCars(){
        return this.carsService.findAll()
    }
    @Get(':id')
    getCarById( @Param('id', ParseIntPipe) id: number ){
        return this.carsService.findOneById(id)
    }
    @Post()
    createCar( @Body() body: any){
        return{
            body
        }
    }
}
```

- Hay que hacer validaciones. Si no viene nada en el brand o en el model del body, no es válido.
- Tampoco sería válido un número, tiene que ser un String
- Parseo con el Pipe

```
import { Body, Controller, Delete, Get, Param, ParseIntPipe, Patch, Post } from
'@nestjs/common';
import { CarsService } from './cars.service';

@Controller('cars')
export class CarsController {

   constructor(
```

```
    private readonly carsService: CarsService
  ){}

  @Get()
  getAllCars(){
      return this.carsService.findAll()
  }
  @Get(':id')
  getCarById( @Param('id', ParseIntPipe) id: number ){
      return this.carsService.findOneById(id)
  }
  @Post()
  createCar( @Body() body: any){
      return{
          body
      }
  }
  @Patch(':id')
  updateCar(
      @Param('id', ParseIntPipe) id: number,
      @Body() body:any){
      return body
  }
  @Delete(':id')
  deleteCar(@Param('id', ParseIntPipe) id: number){
      return {
          msg: "Delete correcto"
      }
  }
}
```

- Si envío el body vacío en la petición Patch me sigue devolviendo un status 200 pero vacío, eso hay que corregirlo
- Con esto todavía no esta implementado el patch put y delete, pero están listos para ello. En la próxima sección