NEST 05

- Voy a trabajar con las marcas para que tengan un id propio y un nombre

NEST Cli Resource - Brands

- La idea es ahora tener el endpoint de brands con todo el CRUD

> http://localhost:3000/brands

Para ello en NEST existe el comando:

> nest g res

- Selecciono RESTAPI y que SI haga los entry points
- Crea el módulo, los servicios, todo
- Crea también una carpeta de entyties.
  - Es como la representación de una tabla, una abstracción de como insertar en la base de datos.
  - Es similar a una interface que me dice cómo van a lucir mis brands
- El dto update-brands es una extensión de create-brand.
  - El PartialType(CreateBrandDto) hace que todas las propiedades del dto del que lo estoy expandiendo sean opcionales

```
import { PartialType } from '@nestjs/mapped-types';
import { CreateBrandDto } from './create-brand.dto';

export class UpdateBrandDto extends PartialType(CreateBrandDto) {}
```

- Falta implementar la lógica de los servicios y los dtos

CRUD Completo - Brands

- Empiezo con entities. Cómo quiero que mi información quede grabada en la base de datos
- Pongo como opcionales la fecha de creación y la fecha de update

```
export class Brand {

    id: string;
    name: string;

    createdAt?: number;
    updatedAt?: number
}
```

- No se le pone de nombre BrandEntity porque así se llamaría luego la DB y no quiero eso
- Voy a los servicios (brand.service)
- Creo una propiedad privada de tipo Brand (de la entity), será un arreglo
- Le agrego un brand por defecto
- Cambio el id de number a string porque así lo maneja uuid(hago la importación de la versión 4)

```
import { Injectable } from '@nestjs/common';
import { IsUUID } from 'class-validator';
import { CreateBrandDto } from './dto/create-brand.dto';
import { UpdateBrandDto } from './dto/update-brand.dto';
import { Brand } from './entities/brand.entity';
import {v4 as uuid} from 'uuid'


@Injectable()
export class BrandsService {

  private brands: Brand[]=[
    {
      id: uuid(),
      name: 'Toyota',
      createdAt: new Date().getTime()

    }
  ]

  create(createBrandDto: CreateBrandDto) {
    return 'This action adds a new brand';
  }

  findAll() {
    return this.brands
  }

  findOne(id: string) {
    const brand = this.brands.find(brand=> brand.id === id)
    if(!brand) throw new NotFoundException(`Brand with id ${id} not found`)

    return brand
  }

  update(id: number, updateBrandDto: UpdateBrandDto) {
    return `This action updates a #${id} brand`;
  }

  remove(id: number) {
    return `This action removes a #${id} brand`;
  }
}
```

- Voy al dto create-brand. Cómo espero que luzca la información que me manden en el post.
- Solo espero el nombre
- Le añado los decoradores

```
import { IsString, MinLength } from "class-validator";

export class CreateBrandDto {

    @IsString()
    @MinLength(1)
    name: string
}
```

- Voy al método create del brands.services
- Pongo brand de tipo Brand, pero contra una base de datos será una nueva instancia de Brand

```
  create(createBrandDto: CreateBrandDto) {
    const brand: Brand ={
      id: uuid(),
      name: createBrandDto.name.toLocaleLowerCase(),
      createdAt: new Date().getTime()
    }
    this.brands.push( brand)
    return brand
  }
```

- Voy al update. Para ello voy al dto update-brand. La única parte que cambia es el name
- Renombro la clase (le quito el PartialType, se usará más adelante) y copio lo mismo que hay en create-brand.dto

```
export class UpdateBrandDto{
    @IsString()
    @MinLength(1)
    name: string
}
```

- Como no estoy trabajando con una base de datos, el update es el mismo trabajo que con cars

```
 update(id: string, updateBrandDto: UpdateBrandDto) {
    let brandDB = this.findOne(id)
    this.brands = this.brands.map(brand=>{
      if(brand.id===id){
        brandDB.updatedAt= new Date().getTime()
        brandDB={
          ...brandDB, ...updateBrandDto
```

```
        }
        return brandDB
      }
      return brand
    })
  }
```

- Ahora falta del Delete. Recuerda: el id es una string, hay que cambiarlo en el controller también

```
  remove(id: string) {
    this.brands = this.brands.filter( brand=> brand.id !== id)
    return
  }
```

- El controller luce así (con el PIpeUUIDPipe incluido)

```
import { Controller, Get, Post, Body, Patch, Param, Delete, ParseUUIDPipe } from
'@nestjs/common';
import { BrandsService } from './brands.service';
import { CreateBrandDto } from './dto/create-brand.dto';
import { UpdateBrandDto } from './dto/update-brand.dto';

@Controller('brands')
export class BrandsController {
  constructor(private readonly brandsService: BrandsService) {}

  @Post()
  create(@Body() createBrandDto: CreateBrandDto) {
    return this.brandsService.create(createBrandDto);
  }

  @Get()
  findAll() {
    return this.brandsService.findAll();
  }

  @Get(':id')
  findOne(@Param('id', ParseUUIDPipe) id: string) {
    return this.brandsService.findOne(id);
  }

  @Patch(':id')
  update(@Param('id', ParseUUIDPipe) id: string, @Body() updateBrandDto:
UpdateBrandDto) {
    return this.brandsService.update(id, updateBrandDto);
  }

  @Delete(':id')
  remove(@Param('id', ParseUUIDPipe) id: string) {
```

```
    return this.brandsService.remove(id);
  }
}
```