03 NEST CLI CRUD

- El comando **resource** me ayuda a crear automáticamente los servicios, controladores, dtos, etc
- En esta sección vamos a aprender la comunicación entre modulos y a crear el módulo de Seed para llenar de coches la DB. Data precargada.
- Los servicios son Singletons. Se crea una instancia y esa instancia es compartida por los demás controladores o servicios mediante inyección de dependencias
- Ahora voy a trabajar con las marcas de los coches, las cuales van a tener su id y el nombre
- Para ello tengo que volver a crear otro modulo, servicio, controlador...pero se hará mediante la linea de comandos
- No lo voy a manejar con una interfaz si no con una entity
- Usaré dtos para la creación y la actualización

Nest CLI Resource - Brands

- En el endpoint /brands voy a tener disponibles todas las marcas con las que voy a trabajar
- Le voy a añadir la fecha de creación y actualización, además del id
- Usaré el CLI para crear el módulo, servicio, controlador, etc (un CRUD completo, se necesita conexión)

> nest g resource brands //g de generar, puedo añadir --no-spec para que no genere los tests

- Selecciono REST API, Generate CRUD entry points? yes
- Automaticamente **añade el módulo** BrandsModule en los imports de *app.module*
- La entity que me ha generado es como una interface. Es una simple clase. Es la representación de una tabla
  - Ahí crearé el nombre, la fecha de creación, etc
- En el package.json puedo ver que instaló **@nestjs/mapped-types**, el resto es igual
- Si tengo problemas de errores puedo deshabilitar "*eslint-config-prettier*" y "*eslint-plugin-prettier*"
- Si analizo el controlador tengo hecha la inyección de dependencias, los dtos en su sitio, etc
- El update-brand.dto hereda de CreateBrandDto con **PartialType**
  - PartialType proviene de *@nestjs/mapped-types*
  - Hace que todas las propiedades sean **opcionales**

CRUD completo de Brands

- Empiezo por la **entity**. Cómo quiero que la información quede grabada en la base de datos
- Las entities no tienen la extension Entity (BrandEntity) porque si no la tabla se llamaría así
- brand.entity.ts

```
export class Brand {
    id: string
    name: string

    createdAt: number
```

```
    updatedAt?: number
}
```

- Creo el arreglo de brands (lo que sería la data) de tipo Brand[]
- En *findOne* uso .find con el id para encontrar el coche. Hago la validación y mando la excepción si no lo encuentra. Retorno brand
- En *findAll* solo tengo que devolver el arreglo con this.brands
- En el método create tengo que ver primero cómo quiero que luzca el dto

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

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

- Creo en el método el objeto brand, utilizo el .push y lo retorno
- En *update* tengo que definir el dto
- **NOTA**: cómo solo tengo una propiedad, de momento no voy a usar PartialTypes

```
//import { PartialType } from '@nestjs/mapped-types';
import { CreateBrandDto } from './create-brand.dto';
import { IsString, MinLength } from 'class-validator';

//export class UpdateBrandDto extends PartialType(CreateBrandDto) {}
export class UpdateBrandDto{
    @IsString()
    @MinLength(1)
    name: string
}
```

- La lógica de actualización es la misma que con los coches
- Para el *delete* uso .filter

```
import { Injectable, NotFoundException } from '@nestjs/common';
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: "Volvo",
      createdAt: new Date().getTime() //debo añadirle getTime para que no choque
con la validación tipo number
    }

  ]

  create(createBrandDto: CreateBrandDto) {
    //lo uso como una interfaz. En la vida real, con una DB, va a ser usado como
una instancia
    const brand: Brand = {
      id: uuid(),
      name: createBrandDto.name.toLowerCase(), //uso toLowerCase porque los quiero
almacenar así
      createdAt: new Date().getTime()
    }

    this.brands.push(brand)

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

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

- En el controlador me aseguro de que el id sea un string y le paso el pipe de UUID

```
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);
  }
}
```

Crear servicio SEED para cargar datos

- Vamos a generar un **SEED** (semilla)
- Se usa para pre-cargar la data
- Uso el CLI

```
nest g resource seed --no-spec
```

- No voy a usar dtos ni entities. No uso los métodos del CRUD, solo necesito el **GET**, lo llamo *runSeed*
- En el servicio borro todos los métodos, dejo un solo método para el GET, lo llamo *populateDB*

- Cars es una propiedad privada en el servicio de cars. Para cargar la data voy a tener que exponerla con un método
- Lo mismo con las brands
- En seed creo una carpeta llamada data. Podría ser un json pero voy a usar TypeScript porque quiero una estructura específica de mi data
- Creo el archivo cars.seed.ts
- En este caso la interfaz Car no necesito que esté importada en el módulo, pero hay cosas que si necesitan estar importadas
- Si son interfaces o clases que no tienen dependencias o ninguna inyección, se pueden importar directamente
- cars.seed.ts

```
import { Car } from "src/cars/interfaces/car.interface";
import { v4 as uuid } from "uuid";


export const CARS_SEED: Car[] = [
    {
        id: uuid(),
        brand: "Toyota",
        model: "Corolla"

    },
    {
        id: uuid(),
        brand: "Suzuki",
        model: "Vitara"

    },
    {
        id: uuid(),
        brand: "Opel",
        model: "Astra"

    }
]
```

- Contra la DB lo que haría es una función que inserte la data (se hará después)
- Hago brands.seed.ts (la relación con los coches se hará cuando se trabaje con una DB real)

```
import { Brand } from "src/brands/entities/brand.entity";
import { v4 as uuid } from "uuid";


export const BRANDS_SEED: Brand[] = [
    {
        id: uuid(),
        name: "Toyota",
        createdAt: new Date().getTime()
```

```
    },
    {
        id: uuid(),
        name: "Suzuki",
        createdAt: new Date().getTime()

    },
    {
        id: uuid(),
        name: "Opel",
        createdAt: new Date().getTime()

    }
]
```

- El método *populateDB* necesita trabajar mediante inyección de dependencias con los otros servicios
- En este caso como trabajo con arreglos podría retornar los arreglos y ya está, pero no es el objetivo de la lección
- Los servicios, como trabajan a través de inyección de dependencias con los controladores, **si debo declararlos en el módulo**
- Además tengo que **poder acceder a la propiedad privada cars y a la propiedad privada brands para cargar la data**
- Creo el método **fillCarsWithSeed** en el cars.service

```
import { BadRequestException, Injectable, NotFoundException } from
'@nestjs/common';
import { Car } from './interfaces/car.interface';
import {v4 as uuid} from 'uuid';
import { CreateCarDto } from './dtos/create-car.dto';
import { UpdateCarDto } from './dtos/update-car.dto';

@Injectable()
export class CarsService {

    private cars: Car[] =[]

    findAll(){
        return this.cars
    }

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

    createCar(createCarDto: CreateCarDto ){

        const car: Car = {
            id: uuid(),
```

```
            ...createCarDto
        }

        this.cars.push(car)

        return car
    }

    updateCar(id: string, updateCarDto: UpdateCarDto){
      let carDB = this.findOneById(id)

      if(updateCarDto.id && updateCarDto !== id){
        throw new BadRequestException('Car id is not valid inside body')
      }

      this.cars = this.cars.map(car=>{
            if(car.id === id){
                carDB={
                    ...carDB,
                    ...updateCarDto,
                    id
                }
                return carDB
            }
            return car
      })

      return carDB
    }

    deleteCar(id: string){
        const car = this.findOneById(id)

        this.cars = this.cars.filter(car => car.id !== id)
    }

    //metodo para el seed

    fillCarsWithSeedData(cars: Car[]){
        this.cars = cars
    }
}
```

- Hago exactamente lo mismo para brands.service

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

  create(createBrandDto: CreateBrandDto) {

    const brand: Brand = {
      id: uuid(),
      name: createBrandDto.name.toLowerCase(),
      createdAt: new Date().getTime()
    }

    this.brands.push(brand)

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

    return brandDB
  }

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

  //método SEED
  fillBrandsWithSeedData(brands: Brand[]){
    this.brands = brands
}
}
```

- Ahora tengo que llamar el *cars.service* y el *brands.service* desde mi **seed.service**

Inyectar servicios en otros servicios

- Para resolver la dependencia de **SeedService** *CarsService* y poder inyectarlo en el constructor tiene que ser parte del **SeedModule**
- Debo exportarlo de *CarsService* e importarlo en *SeedService*
- cars.module

```
import { Module } from '@nestjs/common';
import { CarsController } from './cars.controller';
import { CarsService } from './cars.service';

@Module({
  controllers: [CarsController],
  providers: [CarsService],
  exports:[CarsService] //exporto el servicio
})
export class CarsModule {}
```

- En *imports* importo los **módulos**. En este caso importo el *CarsModule*
- seed.module

```
import { Module } from '@nestjs/common';
import { SeedService } from './seed.service';
import { SeedController } from './seed.controller';
import { CarsService } from 'src/cars/cars.service';
import { CarsModule } from 'src/cars/cars.module';

@Module({
  controllers: [SeedController],
  providers: [SeedService],
  imports:[CarsModule]
})
export class SeedModule {}
```

- Ahora puedo inyectar el servicio y llamar al método pasándole *CARS_SEED*
- seed.service

```
import { Injectable } from '@nestjs/common';
import { CarsService } from 'src/cars/cars.service';
import { CARS_SEED } from './data/cars.seed';


@Injectable()
```

```
export class SeedService {

  constructor(private readonly carsService: CarsService){}

  populateDB() {
    this.carsService.fillCarsWithSeedData(CARS_SEED)

  }
}
```

- Hago lo mismo con brands
- Lo correcto sería borrar todo lo que hay en readme (dejo el logo de Nest) y escribo

```
<p align="center">
  <a href="http://nestjs.com/" target="blank"><img
src="https://nestjs.com/img/logo-small.svg" width="200" alt="Nest Logo" /></a>
</p>

# Car Dealship

Populate DB
```
http://localhost:3000/seed
```
```

- Para ver el archivo abrir README.md y *ctrl+shift+P* Markdown: abrir vista previa en el lateral