01 NEST Primeros Pasos

- Instalar NEST CLI. Ejecutar como administrador

```
npm i -g @nestjs/cli
```

- Crear un proyecto

```
nest new project-name
```

- Para correr la app

```
npm run start
```

- Para correr la app y que escuche los cambios (genera la carpeta dist)

```
npm run start:dev
```

- Por defecto es el puerto 3000

Explicación de src

- Borro todos los archivos, dejo solo el main y el app.module limpio
- Este es el módulo principal. Va a tener referencia a otros módulos, servicios, etc
- app.module.ts

```
import { Module } from '@nestjs/common';


@Module({
  imports: [],
  controllers: [],
  providers: [],
  exports: []
})
export class AppModule {}
```

- Los módulos acoplan y desacoplan un conjunto de funcionalidad específica por dominio
- El **main** tiene una función asíncrona que es *bootstrap* (puedo llamarlo como quiera, main por ejemplo)

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';

async function bootstrap() {
  const app = await NestFactory.create(AppModule); // esto crea la app
  await app.listen(3000); //escucha en este puerto
```

```
}
bootstrap();
```

Controladores

- La diferencia entre las clases de los servicios, controladores, son los decoradores
- Controlan rutas. Son los encargados de escuchar la solicitud y emitir una respuesta
- Para generar un controlador

> nest g co path/nombre

- Para mostrar la ayuda *nest --help*
- Creo un módulo llamado cars en */car-dealership*

> nest g mo cars

- Crea la carpeta cars. La clase cars aparece con el decorador **@Module({})**
- Aparece en el array de imports de app.module (el módulo principal)
- Creo el controlador en */car-dealership* con *nest g co car-dealership*
- Crea una clase **CarsController** con el decorador **@Controller('cars')**
- El controlador lo ha añadido en el modulo de cars
    - Si encuentra un módulo con el nombre cars lo coloca ahí, si no lo hará en el módulo más cercano
- Creo un método GET en el controlador
- Le añado el decorador **@GET()**

```
import { Controller, Get } from '@nestjs/common';

@Controller('cars')
export class CarsController {

    @Get()
    getAllCars(){
        return ['Toyota', 'Suzuki', 'Honda']
    }
}
```

- Si apunto a http://localhost:3000/cars con un método GET me retorna el arreglo

Obtener un coche por ID

- Hago un pequeño cambio, guardo el array en una variable privada y uso el this

```
import { Controller, Get } from '@nestjs/common';

@Controller('cars')
```

```
export class CarsController {

    private cars = ['Toyota', 'Suzuki', 'Honda']

    @Get()
    getAllCars(){
        return this.cars
    }
}
```

- Quiero crear el método para buscar por id. Usaré el parámetro de la url para declararlo como posición en el arreglo
- Tengo que decirselo a Nest. Para obtener parámetros / segmentos uso **@Param('id)**
- Para obtener el body de la petición es **@Body()** y **@Query()** para los query. El response es **Res()** (hay que importarlo de Express)
- Debo añadirle el tipo al id

```
    @Get(':id')   //Puedo colocar /:id pero no es necesario el slash
    getCarById( @Param('id') id: string){
        return this.cars[+id]
    }´
```

- Por ahora puedo parsear el id que viene como string (al venir de la url) con +id, pero **Nest tiene los pipes para parsear la data**

Servicios

- Todos los servicios son providers
- No todos los providers son servicios.
- **Los Providers son clases que se pueden inyectar**
- Para generar un Servicio

> nest g s cars --no-spec //El --no-spec es para que no cree el archivo de test

- No es más que una clase llamada **CarsService** con el decorador **@Injectable()**
- El servicio aparece en el array de providers del módulo *CarsModule*
- Voy a mockear la db en el servicio

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

@Injectable()
export class CarsService {

    //al ser private solo se va a poder consumir dentro del servicio
    private cars =[
        {
            id:1,
```

```
            brand: 'Toyota',
            model: 'Corola'
        },
        {
            id:2,
            brand: 'Suzuki',
            model: 'Vitara'
        },
        {
            id:3,
            brand: 'Honda',
            model: 'Civic'
        }
    ]
}
```

- Ahora necesito hacer uso de la **inyección de dependencias** para usar el servicio en el controlador

---

Inyección de dependencias

- Declaro en el constructor el servicio *private* porque no lo voy a usar fuera de este controlador, y *readonly* para que no cambie accidentalmente algo a lo que apunte
- El arreglo de cars no aparece en el autocompletado de *carsService*. porque es **privado**. Debo crear un método para ello

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

@Controller('cars')
export class CarsController {

    constructor(private readonly carsService: CarsService){}

    @Get()
    getAllCars(){
        return this.carsService. //no aparece el autocompletado porque no tengo
nada público, necesito crear un método
    }

    @Get(':id')
    getCarById(@Param('id') id: string){

    }
}
```

- Añado el método *findAll()*

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

@Injectable()
export class CarsService {

    private cars =[
        {
            id:1,
            brand: 'Toyota',
            model: 'Corola'
        },
        {
            id:2,
            brand: 'Suzuki',
            model: 'Vitara'
        },
        {
            id:3,
            brand: 'Honda',
            model: 'Civic'
        }
    ]

    findAll(){
        return this.cars
    }
}
```

- Ahora si dispongo del arreglo de cars en el servicio con el método *findAll*

```
import { Controller, Get, Param } 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') id: string){

    }
}
```

- Creo también el método *findOneById* en el servicio

```
   findOneById(id: number){
        return this.cars[id]  //esta puede ser una manera. Puedo usar el .find
también
    }
```

- Lo uso en el controlador

```
import { Controller, Get, Param } 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') id: string){
            return this.carsService.findOneById(+id)
    }
}
```

Pipes

- Hay que implementar una validación del argumento que le paso como id
- Si pasara algo que no es un numero como 3a me devolvería un *NaN*. Debo manejar este tipo de errores
- Los pipes transforman la data recibida en requests, para asegurar un tipo, valor o instancia de un objeto.
- Pipes integrados por defecto
    - ValidationPipe - más orientado a las validaciones y también hace ciertas transformaciones
    - ParseIntPipe - transforma de string a número
    - ParseBoolPipe - transforma de string a boolean
    - ParseArrayPipe - transforma de string a un arreglo
    - ParseFloatPIpe - transforma de string a un float
    - ParseUUIDPipe - transforma de string a UUID
- Uso ParseIntPipe para la verificación de que sea un entero

```
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')             //uso el Pipe    ahora si puedo tipar a number el id
    getCarById(@Param('id', ParseIntPipe ) id: number){
            return this.carsService.findOneById(id)
    }
}
```

- Si ahora le paso algo que no sea un número en la url salta el error "*Validation failed (numeric string is expected)*"
- Si yo lanzo un error dentro del controlador con *throw new Error*, el servidor responde con *500 Internal Server Error* y en consola me aparece el error
- La *Exception Zone* incluye cualquier zona menos los middlewares y los *Exception Filters*
- Cualquier error (no controlado) que sea lanzado en la *Exception Zone* será lanzado automáticamente por Nest
- **Falta una cosa**: si yo pongo un id válido (un número) pero que no existe en la DB me manda un *status 200* cómo que todo lo hizo correcto. Esto no debería ser así

Exception Filter

- Maneja los errores de código en mensajes de respuest http. Nest incluye los casos de uso comunes pero se pueden expandir
    - BadRequestException: se estaba esperando un número y recibí un string, por ejemplo
    - NotFoundException: 404, no se encontró lo solicitado
    - UnauthorizedException: no tiene autorización
    - ForbiddenException
    - RequestTimeoutException
    - GoneException
    - PayloadTooLargeException
    - InternalServerErrorException: 500
- Estos solo son los más comunes, hay muchos más (mirar la documentación de Nest)
- Ahora quiero mandar un error si el coche no existe
- cars.service.js

```
findOneById(id: number){
        const car = this.cars.find(car => car.id === id)
        if(!car) throw new NotFoundException() //si el id no existe devuelve Not
Found
        return car
    }
```

- Puedo escribir dentro del paréntesis el mensaje que quiero mostrar

```
    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
    }
```

Post, Patch y Delete

- Creo el método Post en el controlador
- Uso **@Body** para obtener el body de la petición. Lo nombro body y lo tipo *any* (de momento)
- Debo hacer la validación de que me envien un brand y un model en el body y de que sean strings
- Por ahora creo los tres endpoints

```
    @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 id
    }
```

- **NOTA:** En el Patch, si no pongo nada en el body me regresa un *status 200* igualmente!
- La implementación de los métodos en la siguiente sección