

NEST 02

Desactivar Prettier

- Opcional

```
npm remove prettier
```

Obtener un coche por ID

- Muevo el arreglo a una variable
- Ahora cars es una propiedad de la clase
- Lo retorno con el this en la petición Get

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

@Controller('cars')
export class CarsController {

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

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

- Ahora quiero llamar desde el endpoint a uno de los coches por su índice a través de la url

```
http://localhost:3000/cars/1
```

- Para ello me creo otro método
- Usaré el decorador @Param('id') para relacionar el id de la url con el id que quiero usar como índice
- Por defecto cualquier parámetro que venga de la url se va a recibir como un string

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

@Controller('cars')
export class CarsController {

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

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

```
@Get('/:id')
getCarById( @Param('id') id){

    return this.cars[id]
}
}
```

- Si en el endpoint pongo un índice que no existe en el array(el 4, por ejemplo) me sigue devolviendo un status 200 pero vacío
- Esto debe manejarse para responder con un 404(Not Found) y un mensaje

Servicios

- Hasta el momento la información de los cars está en el controlador, lo que no tiene mucho sentido.
- Si los necesito en otra parte de la aplicación no puedo acceder
- Idealmente sería una base de datos
- Los **servicios** alojan la lógica de negocio de tal manera que sea reutilizable mediante inyección de dependencias.
 - Por ejemplo: PeliculaService, encargado de grabar, actualizar o eliminar películas
- **Todos los servicios son providers. No todos los providers son servicios**
- Los providers son clases que se pueden inyectar
- Basicamente es un provider porque lleva el decorador @Injectable
- Para crear el servicio

```
nest g s cars --no-spec
```

- El no-spec es para que no cree el archivo de testing
- El servicio que crea no es más que la clase CarsService con el decorador @Injectable()
 - Esto significa que se puede inyectar En el cars.modulo, en providers, se ha importado automáticamente

- RECUERDA: TODOS LOS SERVICIOS SON PROVIDERS

- Corto y pego los cars en el servicio, lo formateo

```
import { Injectable } 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'
      }
    ]
  }
}

```

- Al ser private sólo puedo consumir mis cars dentro del servicio
- Ahora debo inyectar la dependencia en el constructor (inyección de dependencias)

Inyección de dependencias

- Inyección de dependencias es definir una propiedad en el constructor de la forma corta de typescript, que va a tener un tipo de dato en específico
- en este caso CarsService
- private para que se use solo dentro de este controlador
- readonly para que no modifique nada de a lo que apunta
- es una propiedad de tipo CarsService
- Esto crea una instancia del CarsController. En el caso de ya haber una instancia re-utiliza esa instancia.
- Todavía no puedo acceder a los cars, porque no hay nada público en el servicio CarsService

```

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.????--> Falta el método de CarsService
  }
  @Get('/:id')
  getCarById( @Param('id') id){
    return this.carsService.????
  }
}

```

- Lo que voy a hacer es crear un par de métodos para extraer los cars del servicio en el controlador
- Lo llamaré findAll

```
import { Injectable } 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
  }
}
```

- Muteo por ahora el segundo método con el id para hacer la prueba

```
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){
  //   return this
  // }
}
```

- Hago el método para extraer el coche por el id

```
import { Injectable } 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)
    return car
  }
}
```

- Parseo el id a entero en el controller cuando llamo al método de carsService

```
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 ){
    return this.carsService.findOneById(parseInt(id))
  }
}
```

```
}  
}
```

- También sirve añadiéndole un + al id, (+id)
- Si mando un numero que no existe me sigue dando status 200 , o con una letra devuelve NaN
- Entonces tengo que validar en el controlador que el id sea un número y en el servicio que el id exista