NEST Duro De Roer - APUNTES INICIALES

- Para generar un nuevo módulo, creo una nueva carpeta llamada modules
- Dentro de la carpeta genero el módulo

> nest g mo names

- Lo incluyo en el imports de app.module
- Para generar un controlador, no hace falta que esté en la misma carpeta names
- Pero si **lo llamo igual que el módulo**

> nest g co names

- En controllers de names.module aparece automáticamente NamesController
- Se acostumbra a poner en el decorador controller *ap/v1* **@Controller('api/v1/names')**
- Se usa **api/v1** porque si la app crece y se añade un nuevo controlador, poder seguir dando soporte al antiguo
- Para crear un servicio, estoy en la carpeta modules

> nest g s names

- Aparece automáticamente en providers de names.module NamesService
- En el servicio aparece el decorador **@Injectable()**
- Significa que puedo inyectar esta clase en el constructor (en este caso del controlador)

```
constructor( private readonly namesService: NamesService)
```

- Para este ejemplo no se usará db, por lo que creo un array de nombres

```
@Injectable()
export class NamesService{

    private _names: string[]

    constructor(){
        this._names = []
    }
}
```

- En el controlador, para obtener la data del body uso @Body
- Aquí usaría un dto en lugar del objeto data, donde tipo el name como un string

```
@Post()
createName(@Body() data:{name: string}){
```

```
    return this.namesService.createname(data.name)
}
```

- En el servicio le paso el name
- Uso el push porque no tenemos uns db y es solo un array
- El return true es solo como mensaje de confirmación de que se ha ejecutado el código

```
createName(name: string){
    this._names.push(name)
    console.log("Names: ", this._names)

    return true
}
```

- Tengo que validar si existe el nombre!
- Uso lowerCase para igualar, trim para eliminar los espacios. Solo elimina espacios delante y detrás, no enmedio
- En caso de que no exista que haga la inserción

```
createName(name: string){

    const nameFound = this._names.find(n => n.toLowerCase().trim() ===
name.toLowerCase().trim())

    if(!nameFound){
       this._names.push(name)
        console.log("Names: ", this._names)
        return true
    }else{
        return false
    }
}
```

- Para el GET es sencillo
- controller

```
@Get()
getNames(){
    return this.namesService.getNames()
}
```

- service

```
getNames(){
    return this._names
}
```

- El decorador @Query me sirve para filtrar elementos a través de la url
- Pongamos que quiero recoger el valor de start

> http://localhost:3000/api/v1/products?start=fer&end=ando

- Cuando sólo va a ser un parámetro puedo especificarlo en @Query()

```
@Get()
getNames(@Query('start') start: string){
    return this.namesService.getNames()
}
```

- Si no le paso ningún query tendrá el valor de undefined
- Cuando van a ser muchos lo puedo dejar **vacío** y tiparlo como query:any
- En el servicio lo marco cómo opcional, porque puede llegar o no
- La lógica es que si viene start lo devuelvo filtrado, si no lo devuelvo tal cual
- filter devuelve un array de algo segun una condición. Uso startsWith para comprobar si empieza por esa letra

```
getNames(start?: string){

    if(!start){
        return this._names
    }else{
        return this._names.filter(n =>
n.toLowerCase().trim().startsWith(start.toLowerCase()))
    }
}
```

- Para el update uso @Put, @Param para capturar el nombre a actualizar y el nuevo nombre
- controller

```
@Put('/:name/:newName')
updateName(@Param('name') name: string, @Param('newName') newName: string){
    return this.namesService.updateName(name, newName)
}
```

- Uso el mismo método find en el servicio para verificar si existe el nombre
- El newName no tiene que existir
- En lugar de find uso findIndex. Si devuelve -1 es que no existe

- service

```
updateName(name: string, newName: string){
    const indexNameFound = this._names.findIndex(n => n.toLowerCase().trim() ===
name.toLowerCase().trim())
    const indexNewNameFound = this._names.findIndex(n => n.toLowerCase().trim()
=== name.toLowerCase().trim())

    if(indexNameFound != -1 && newNameFound == -1){
        this._name[indexNameFound] = newname
        return true
    }else{
        return false
    }
}
```

- Para el delete tengo que extraer del parámetro el nombre y confirmar que el nombre exista
- controller

```
@Delete('/:name')
deleteName(@Param('name') name: string){
    return this.namesService.deleteName(name)
}
```

- Declaro una variable con el número de elementos del array antes de borrar
- Uso el filter para devolver un array con todos menos con el name que he extraido con @Param
- Si deletedBefore y deletedAfter son diferentes es que se ha borrado el elemento
- service

```
deleteName(name: string){
    const deletedBefore = this._names.length

    this._names = this._names.filter(n=> n.toLowerCase().trim() !==
name.toLowerCase().trim())

    const deletedAfter = this._names.length

    return deletedBefore != deletedAfter
}
```

- Limpiando todos los nombres
- Será con otro delete pero le voy a poner un nombre

```
@Delete('clear')
clearNames(){
```

```
    return this.namesService.clearNames()
}
```

- service

```
clearNames(){
    this._names = []
}
```

- El problema aquí es que está detectando el clear como un nombre por el otro delete
- Cuanto más **específico sea, debo colocarlo más arriba en el controlador**
- **Es importante el orden**
- Es decir, si le pongo mancuso, Nest dice: "ok, clear no es" y lo trata como un nombre
- Pero para eso el delete de clear **debe estar antes** del delete por nombre