Back End

Carrera Programador full-stack

ERROR HANDLING

Controllers Best Practices

Hasta aquí hemos visto que con los decoradores **@Res** y **@HttpStatus** más el *enum* **HttpCode** podemos componer respuestas acorde al resultado de una petición. Pero el código que hemos generado no es del todo correcto. Bueno, sí lo es porque funciona, pero no cumple un estándar de la industria:

*Los **controladores** deberían tener la **menor** cantidad de **código** posible y delegar la **lógica de negocios** a los **servicios***

```
@Get(':id')
async getTrackById(@Res() response, @Param('id') id: number): Promise<any> {
  const responseFromService = await this.trackService.getTrackById(id);
  if (Object.keys(responseFromService).length) {
    return response.status(HttpStatus.OK).json(responseFromService);
  } else {
    return response
      .status(HttpStatus.NOT_FOUND)
      .json({ error: 'track no existe' });
  }
}
```

Exception Filters en Nest JS

Acabamos de ver que el controlador maneja la lógica para dar una u otra respuesta, dependiendo de si el track buscado está o no en la base de datos.

Vamos a delegar esa lógica al servicio **getTrackById()** y así lograremos:

1. **Mayor abstracción en el controlador**
2. **Mejor encapsulamiento en el servicio**

Podríamos hacerlo “a mano”, tal como hicimos en el controlador, pero resulta que Nest JS nos ofrece una cantidad de **excepciones útiles para devolver errores** pre configurados (**BadRequestException**, **NotFoundException**, **ForbiddenException**, etc.)

Nosotros podemos levantar estas excepciones en cualquier clase por la que pase el flujo de ejecución en una solicitud.

Los **servicios** son un lugar adecuado para **levantar excepciones**.

Exceptiones en Nest JS

1. HttpException: Esta es una excepción base que se usa para crear excepciones personalizadas relacionadas con HTTP. Puedes extender esta clase para crear excepciones específicas para diferentes códigos de estado HTTP.

2. NotFoundException: Una excepción que se utiliza cuando no se encuentra un recurso. Puede estar relacionada con el código de estado HTTP 404 (Not Found).

4. ForbiddenException: Se utiliza cuando el servidor entiende la solicitud, pero el usuario no tiene los permisos necesarios. Corresponde al código de estado HTTP 403 (Forbidden).

5. BadRequestException: Se utiliza para errores de solicitud incorrecta. Esto podría incluir validación de datos fallida u otros problemas relacionados con la solicitud. Corresponde al código de estado HTTP 400 (Bad Request).

Exception Filters en Nest JS

Como ya hemos visto, las excepciones requieren usar bloques try ... Catch, de modo tal que los errores no “rompan” la ejecución del programa.

Los **filtros de excepción** (o *exception filters*) consisten en una capa de procesamiento automático de las excepciones no tratadas que se producen en las aplicaciones.

Con los **exception filters** de Nest JS, cualquier excepción que no haya sido tratada en tu propio código se capturará por los filtros de excepciones y devolverá el mensaje apropiado a los clientes como respuesta.

Lanzar excepción desde un servicio

Veamos cómo levantar una excepción en un servicio, mejorando el código del método **getTrackById()**.

Ese método está preparado para devolver el objeto producto de un identificador dado.

Pero primero, adecuamos nuestro controlador, quitando de allí el código de comprobación y definición de respuestas y el decorador @Res (*ese trabajo lo hará Nest por nosotros de ahora en adelante*):

```
@Get(':id')
getTrackById(@Param('id') id: number): Promise<any> {
  return this.trackService.getTrackById(id);
}
```

Lanzar excepción desde un servicio

Con el *controller "limpio", podemos concentrarnos en levantar una excepción, de ser necesario, en* el método **getTrackById()**.

La lógica es la siguiente:

1. **Si se pasa un id válido, se retorna el recurso.**
2. **Si el track no existe, lanzamos una excepción.**

```
async getTrackById(id: number): Promise<Track | undefined> {
  const res = await fetch(BASE_URL + id);

  try {
    const parsed = await res.json();
    if (Object.keys(parsed).length) return parsed;
  } catch (err) {
    console.log(err)
    throw new NotFoundException('no se encontro el id')
  }
}
```

Lanzar excepción desde un servicio

Independientemente de que el recurso exista, la respuesta al cliente será apropiada, según los estándares HTTP y los principios API REST