

Back End

**Carrera
Programador
full-stack**

HTTP Response Codes

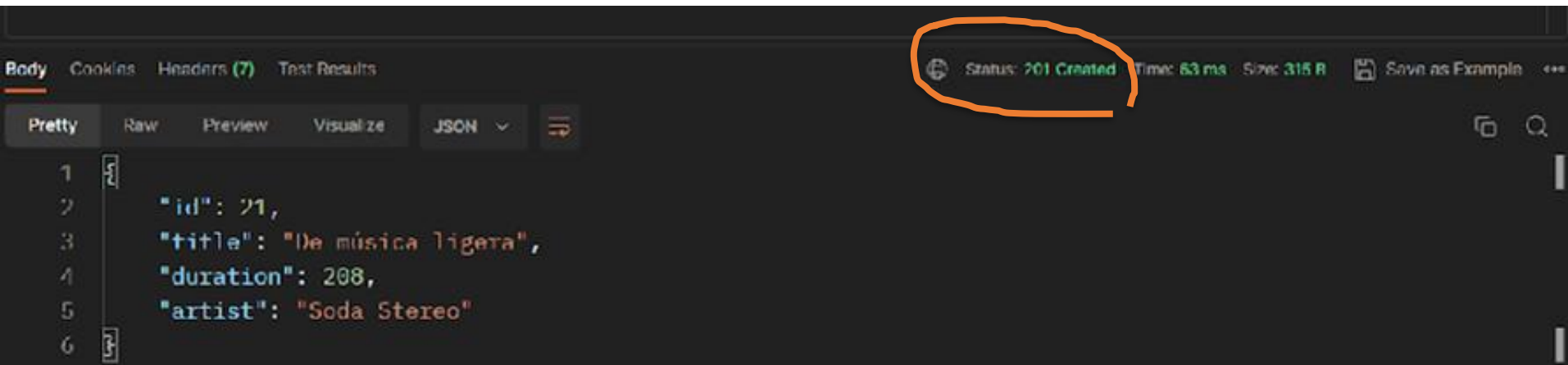
HTTP Status Code

Vamos a avanzar en la programación de nuestros controladores aprendiendo a administrar los *status* de las respuestas.

El "***status code***" nos sirve para indicar a los clientes de nuestras API's cómo ha ido la gestión de sus solicitudes, de modo que puedan representar convenientemente los **mensajes de feedback** en el lado del frontend.

El modelo de desarrollo por API REST tiene **estados de respuesta y mensajes estándar** dependiendo del tipo de solicitud y el resultado.

NestJS nos ofrece por defecto status adecuados según el patrón REST, como por ejemplo el **201** cuando un recurso se ha podido crear después de una solicitud POST.



HTTP Status Code

HttpStatus.CODE

OK = 200 : Respuesta exitosa. El recurso solicitado se ha encontrado y se devuelve en el cuerpo de la respuesta.

CREATED = 201 : La solicitud ha tenido éxito y se ha creado un nuevo recurso como resultado.

NO_CONTENT = 204 : La solicitud se ha procesado correctamente, pero no hay contenido para enviar en la respuesta.

BAD_REQUEST = 400 : La solicitud es incorrecta o mal formada.

UNAUTHORIZED = 401 : El cliente no está autenticado y se requiere autenticación para acceder al recurso.

FORBIDDEN = 403 : El servidor entiende la solicitud, pero el usuario no tiene los permisos necesarios.

NOT_FOUND = 404 : El recurso solicitado no se encontró en el servidor.

UNPROCESSABLE_ENTITY = 422 : La solicitud es válida pero el servidor no puede procesar los datos, por ejemplo, debido a validaciones fallidas.

INTERNAL_SERVER_ERROR = 500 : Se produce un error interno en el servidor al procesar la solicitud.

En el ejemplo anterior vimos el estado “**201 created**” para responder a una solicitud **POST**.

También tenemos un “**200 OK**” como respuesta a una petición **PUT** que se resuelve satisfactoriamente. En este caso, también se incluye en la *response* el detalle del recurso modificado.

The screenshot displays a REST client interface with a PUT request to `http://localhost:3000/tracks/20`. The request body is a JSON object: `{ "title": "Anti-Hero", "duration": 201, "artist": "Taylor Swift" }`. The response is a `200 OK` status with a JSON body: `{ "id": 20, "title": "Anti-Hero", "duration": 201, "artist": "Taylor Swift" }`. The status bar indicates the response time is 110 ms and the size is 303 B.

```
PUT http://localhost:3000/tracks/20
```

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "title": "Anti-Hero",
3   "duration": 201,
4   "artist": "Taylor Swift"
5 }
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 110 ms Size: 303 B Save as Example

Pretty Raw Preview Visualize JSON

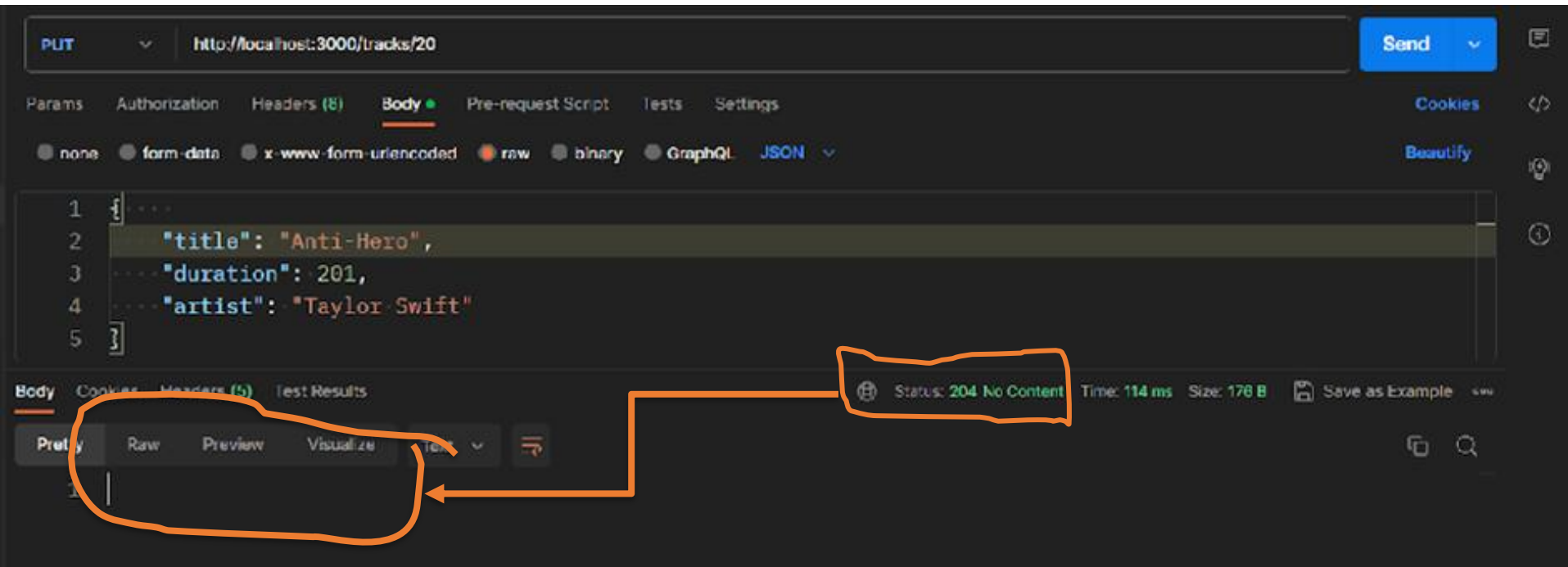
```
1 {
2   "id": 20,
3   "title": "Anti-Hero",
4   "duration": 201,
5   "artist": "Taylor Swift"
6 }
```

No obstante, es bastante común modificar un recurso y no devolverlo en la respuesta. Para eso se utiliza el estado “**204 no content**”.

Vamos a ver a continuación que administrar los códigos de estado es muy sencillo con el decorador **@HttpCode()**. Solo tenemos que importarlo en el controlador y agregarlo en el verbo que deseamos intervenir. Vamos a hacerlo con el manejador **PUT**, en el archivo `track.controller.ts`:

```
@Put('/:id')
@HttpCode(204)
updateTrackById(@Param('id') id: number, @Body() body):
Promise<void> {
    return this.trackService.updateTrackById(id, body);
}
```

Eso es todo lo que se necesita. Ahora, si hacemos una modificación de recurso ya no veremos el recurso modificado en la respuesta, de acuerdo al estándar del *response code 204*:



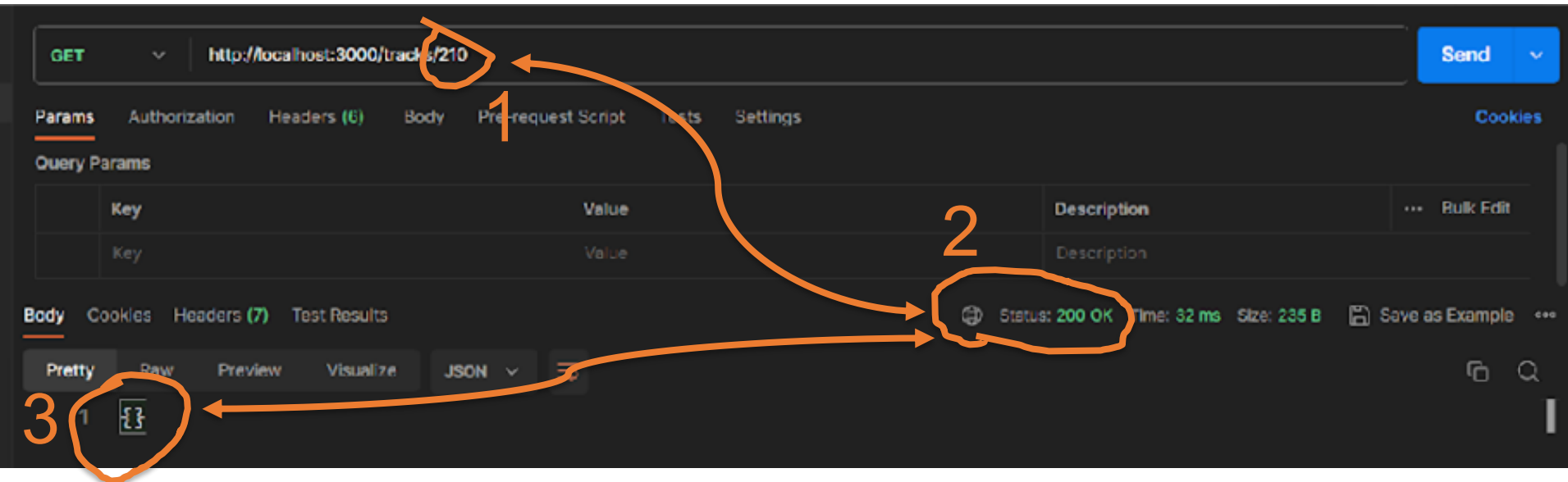
Opcional:

Modificar el servicio **PUT**

Ya ha quedado claro que el método `updateTrackById()` retorna el resultado de la petición aunque no estemos haciendo nada con ese valor de retorno. Si desea puede editar el método:

```
async updateTrackById(id: number, body: Track): Promise<void> {
  const isTrack = await this.getTrackById(id);
  if (!Object.keys(isTrack).length) return; //early return
  const updatedTrack = {...body, id };
  await fetch(BASE_URL + id, {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(updatedTrack),
  });
}
```

Códigos de respuesta condicionales



1. Se envía una **request** con el **id** de un **track** **inexistente**
2. El controlador procesa la petición y la envía al servicio
3. `getTrackById()` busca el track con **id 210** y no lo encuentra
4. El controlador devuelve un **objeto vacío** y el código de respuesta **200 OK**

¿OK? ¡Cómo que OK!



Tenemos que validar este procedimiento en el *controller* y devolver el ***status code*** que indique cómo se resolvió la petición.

En caso que el recurso que se intenta acceder no exista tendremos que dar un código de respuesta de recurso inexistente. Sin embargo, si el identificador sí estaba presente en el sistema, el código de respuesta debería ser otro.

Vamos a ver una posible solución que nos ayudará a salir del paso.

Decorador @Res

Este decorador permite recibir un objeto ***response*** que podemos usar para definir la respuesta de la solicitud de una manera más detallada, incluyendo explícitamente un **código de status determinado**.

Para facilitar la legibilidad del código NestJS proporciona también una lista de **estados con nombre** en un *enum* llamado "**HttpStatus**". Una vez importado, gracias a TypeScript, podemos ver todas las alternativas que nos ofrece en el propio editor.

Así quedaría nuestro controlador:

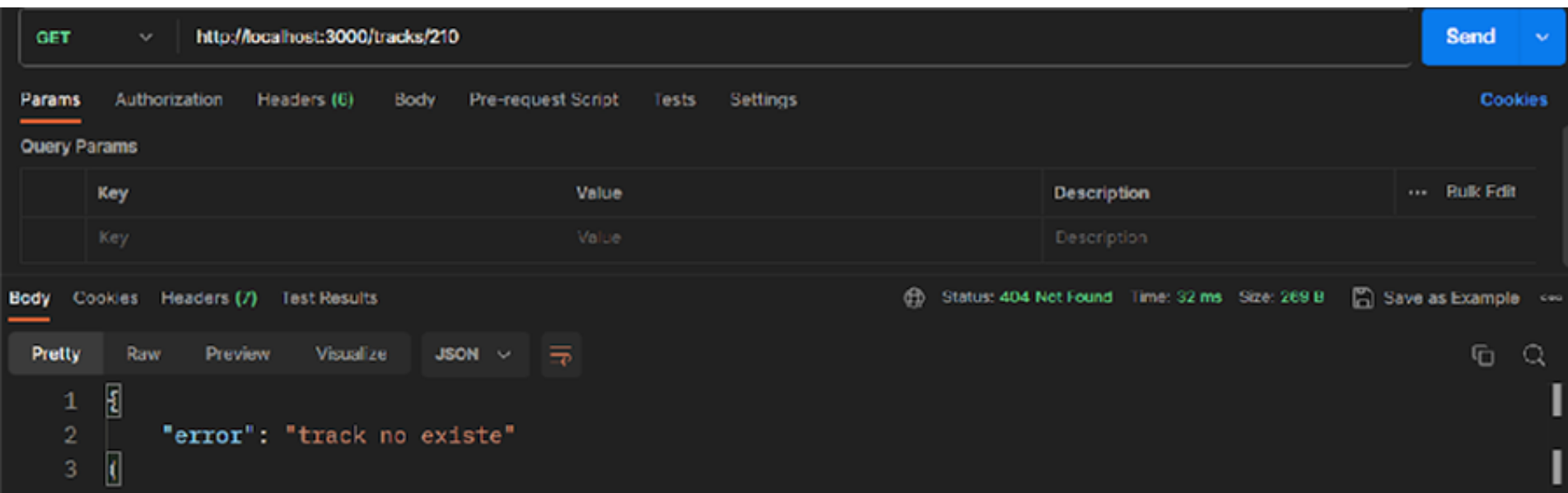
```
@Get(':id')
async getTrackById(@Res() response, @Param('id') id: number): Promise<Track | undefined> {
  const responseFormService = await this.trackService.getTrackById(id);

  if (responseFormService && Object.keys(responseFormService).length) {
    return response.status(HttpStatus.OK).json(responseFormService);
  } else {
    return response.status(HttpStatus.NOT_FOUND).json({ error: 'no se encontró' });
  }
}
```

Y vamos a tener que hacer algunos cambios en getByID y updateByID por que no temenos validaciones y falla :

```
async getTrackById(id: number): Promise<Track | undefined> {  
  const res = await fetch(BASE_URL + id);  
  console.log('Estado:', res.status);  
  
  if (!res.ok) {  
    return undefined;  
  }  
  
  try {  
    return await res.json();  
  } catch (err) {  
    console.error('Error ', err);  
    return undefined;  
  }  
}
```

```
async updateTrackById(id: number, body: Track): Promise<Track | undefined> {  
  const isTrack: Track | undefined = await this.getTrackById(id);  
  
  if (!isTrack || !Object.keys(isTrack).length) {  
    console.warn(`Track con id ${id} no encontrado`);  
    return;  
  }  
  
  const updateTrack = { ...body, id };  
  console.log('Pista actualizada', updateTrack.title);  
  
  const res = await fetch(BASE_URL + id, {  
    method: 'PUT',  
    headers: {  
      'Content-Type': 'application/json',  
    },  
    body: JSON.stringify(updateTrack),  
  });  
  
  if (!res.ok) {  
    console.error(`Error al actualizar: ${res.status}`);  
    return;  
  }  
}
```



The screenshot shows a web browser's developer tools interface. The top bar indicates a GET request to `http://localhost:3000/tracks/210`. The 'Params' tab is active, showing a table with columns 'Key', 'Value', and 'Description'. The 'Body' tab is also visible, showing the response status: 'Status: 404 Not Found', 'Time: 32 ms', and 'Size: 269 B'. The response body is displayed in JSON format, showing an error message: `"error": "track no existe"`.

Key	Value	Description
Key	Value	Description

Body: Cookies Headers (7) Test Results

Status: 404 Not Found Time: 32 ms Size: 269 B Save as Example

```
{  
  "error": "track no existe"  
}
```



¡SALUD!

A partir de aquí tienen las herramientas necesarias para verificar qué controladores necesitan devolver **status codes** de forma condicional, de acuerdo a si las peticiones **GET** retornan el recurso buscado o si otras peticiones terminan con éxito o no.