

Back End

Carrera Programador full-stack

ALTAS

Petición HTTP de tipo POST

POST

Para la creación de un recurso se utiliza el método **POST**.

Vamos a programar en nuestra API que si le enviamos **POST** a **/tracks** se cree una nueva pista.

La característica más relevante de **POST** es que nos enviarán datos mediante el cuerpo de la solicitud. Esos datos definirán el elemento del recurso que se desea insertar.

Crear una ruta con Post en NestJS es tan sencillo como usar el decorador **@Post**, pero además tenemos que aprender a recibir los datos que se envían en la solicitud y poder operar mediante ellos para componer la respuesta.

Pensemos...

Qué nos falta?

- Enviar datos desde Postman / ThunderClient
- Recibir el pedido en el Controller. (*Más adelante veremos que antes de procesar los datos que llegan al endpoint hay que validarlos/sanitizarlos.*)
- Pasarlo al Service y que se procese

¿Cómo enviamos los datos?

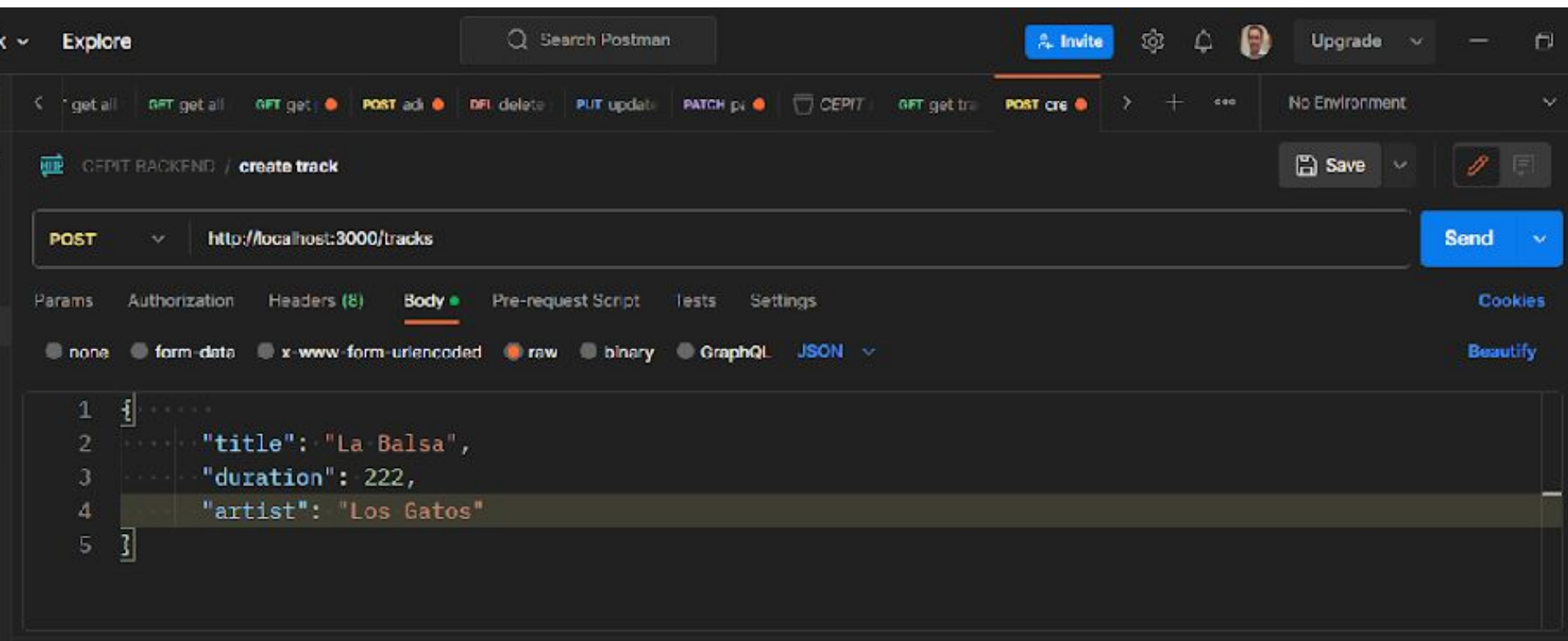
Para trabajar con APIs REST, los datos en general se envían en formato JSON

Para la estructura del dato, usamos la misma de salida

```
{  
  "id": 1,  
  "title": "Bohemian Rhapsody",  
  "duration": 420,  
  "artist": "Queen"  
}
```

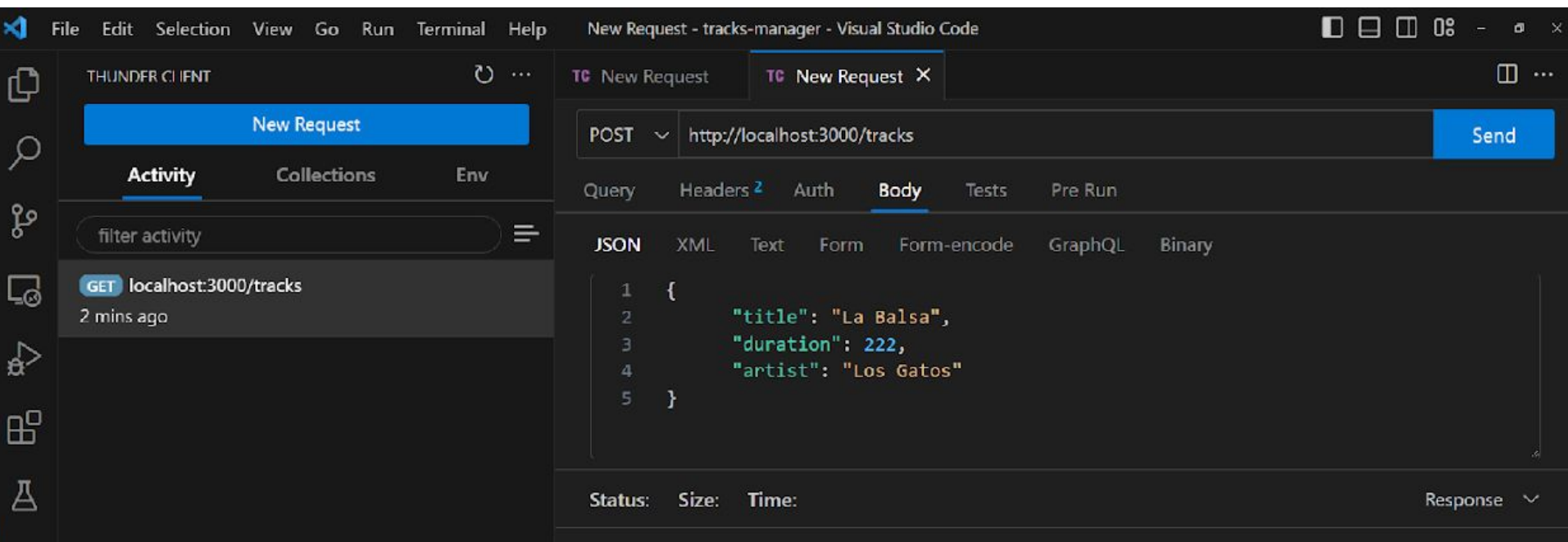
Enviando desde POSTMAN

Hacemos un POST y escribimos nuestro JSON en el **body** de la solicitud.



Enviando desde ThunderClient

Hacemos un POST y escribimos nuestro JSON en el **body** de la solicitud.



¿Cómo recibimos los datos?

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

```
export class TrackController {  
  constructor(private trackService: TrackService) {}
```

```
  ...
```

```
  @Post()
```

```
  create(@Body() body): Promise<Track> {
```

```
    return this.trackService.addTrack(body);
```

```
  }
```

```
  ...
```

```
}
```



No podemos permitir que el usuario ingrese el id de un recurso, eso debe gestionarlo la base de datos para mantener la integridad referencial.

FIND OUT LAST ID AND ADD ONE

Podríamos usar una dependencia para generar un id, pero nos gusta practicar así que vamos a construir un helper method que al último id le sume uno y nos devuelva ese número. Hagamos algo así en nuestra clase `TrackService`:

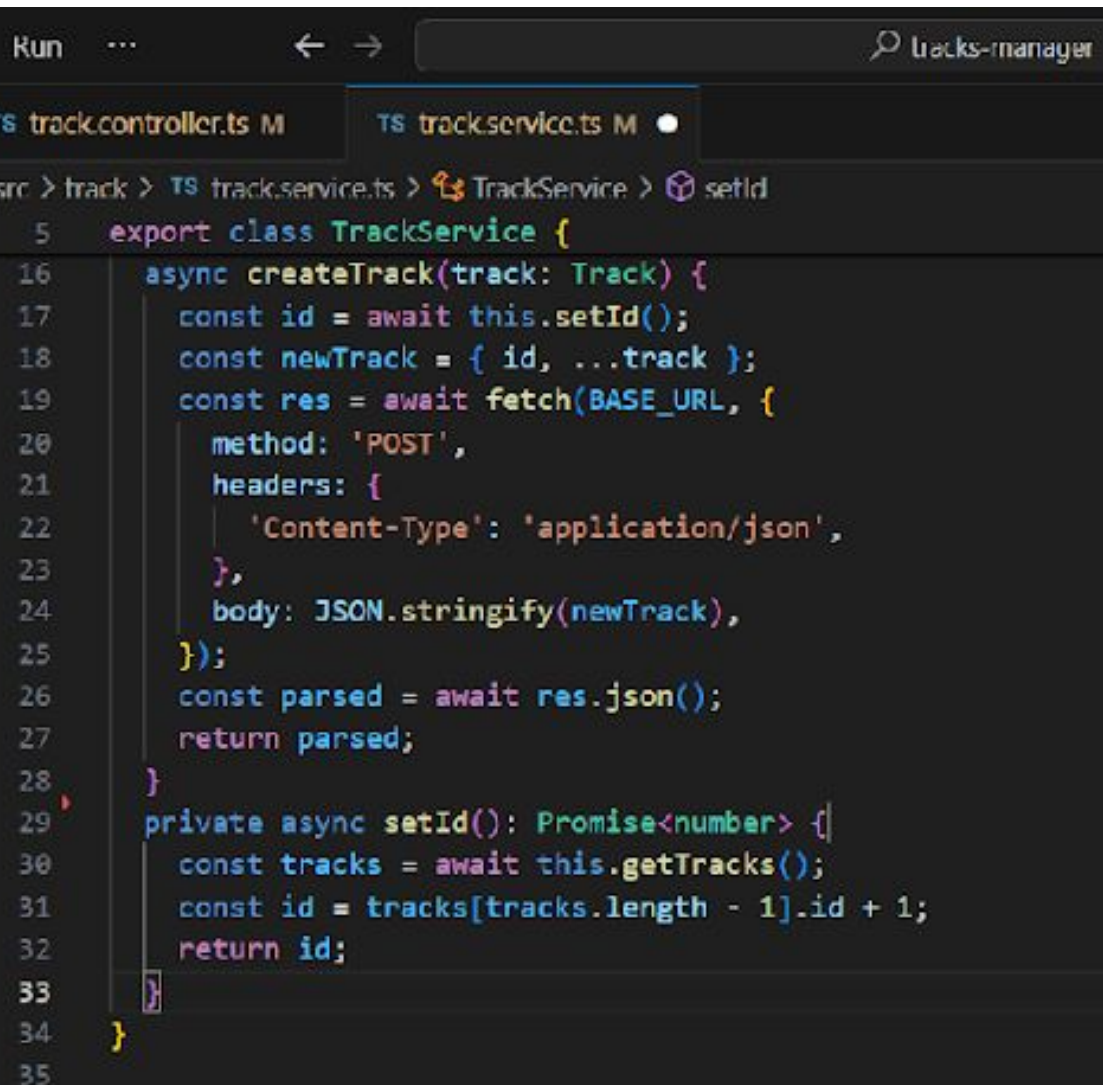
```
private async setId(): Promise<number> {  
  const tracks = await this.getTracks();  
  const id = tracks[tracks.length-1].id + 1;  
  return id;  
}
```

TS track.controller.ts M X

src > track > TS track.controller.ts > ...

```
1  import { Get, Post, Controller, Param, Body } from '@nestjs/common';
2  import { TrackService } from '../track.service';
3  import { Track } from '../track.interface';
4
5  @Controller('tracks')
6  export class TrackController {
7      constructor(private readonly trackService: TrackService) {}
8
9      @Get()
10     getTracks(): Promise<Track[]> {
11         return this.trackService.getTracks();
12     }
13
14     @Get('/:id')
15     getTrackById(@Param('id') id: number): Promise<Track> {
16         return this.trackService.getTrackById(id);
17     }
18
19     @Post()
20     createTrack(@Body() body): Promise<any> {
21         return this.trackService.createTrack(body);
22     }
23 }
```

Solo falta crear el método `createTrack()` en nuestra clase `TrackService`.

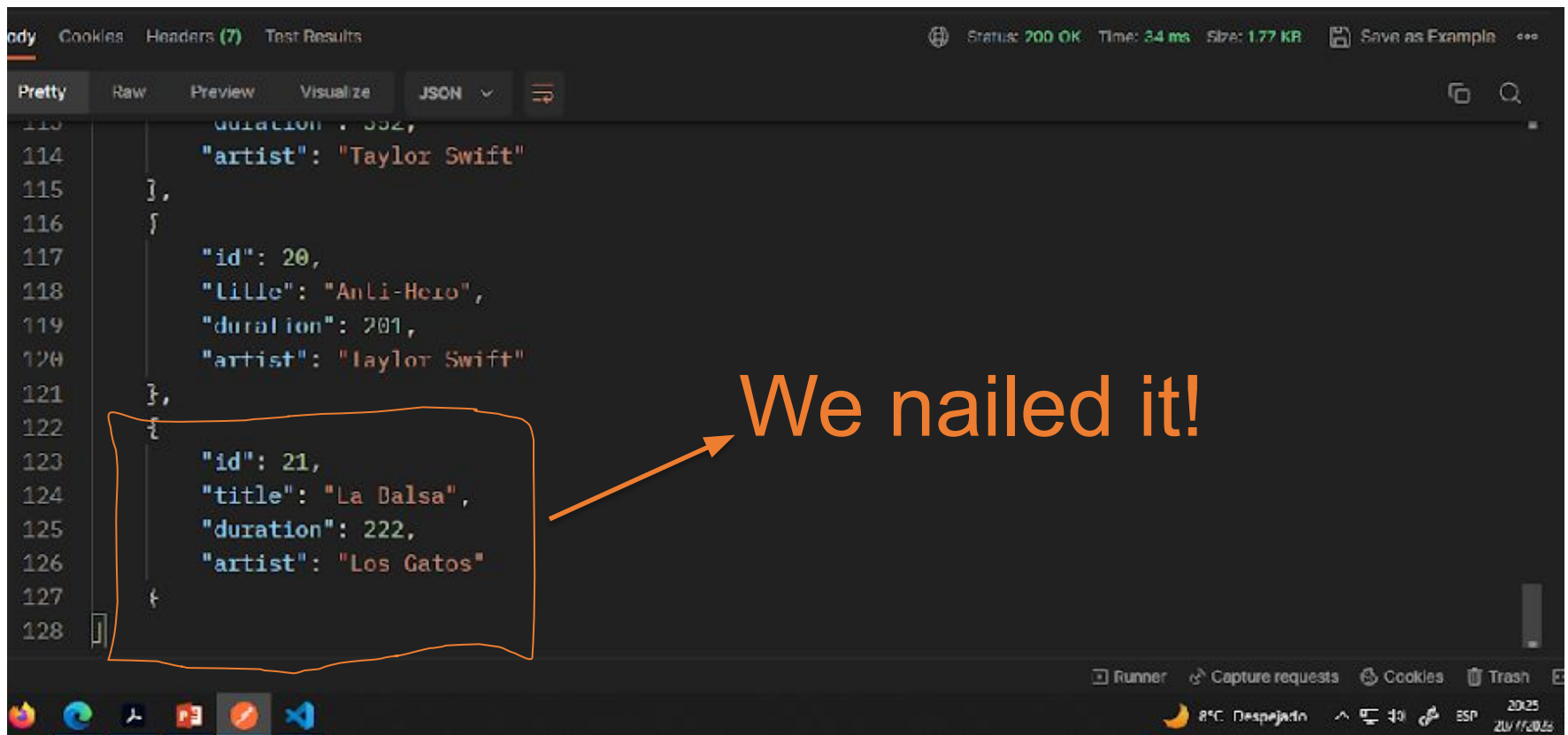


```
Run ... <=> tracks-manager
track.controller.ts M TS track.service.ts M
src > track > TS track.service.ts > TrackService > setId
5 export class TrackService {
16   async createTrack(track: Track) {
17     const id = await this.setId();
18     const newTrack = { id, ...track };
19     const res = await fetch(BASE_URL, {
20       method: 'POST',
21       headers: {
22         'Content-Type': 'application/json',
23       },
24       body: JSON.stringify(newTrack),
25     });
26     const parsed = await res.json();
27     return parsed;
28   }
29   private async setId(): Promise<number> {
30     const tracks = await this.getTracks();
31     const id = tracks[tracks.length - 1].id + 1;
32     return id;
33   }
34 }
35
```

Debe notar que el método recibe el **body** de la *request* pasado desde el controlador, pero le **falta el id** pues dijimos que no lo crea el usuario. Para poder tiparlo como un objeto de tipo **Track** necesitamos modificar la **interfaz** y declararlo como **opcional la propiedad id**. Si no lo hiciéramos tendríamos un problema con el control de tipos de TypeScript.

Si por alguna razón no desea declarar como opcional la prop “id”, debería construir una nueva interfaz que tenga solamente las props que el usuario nos envía en el *body* de la *request*.

Luego de ejecutar el **POST** podemos hacer un **GET ALL** para ver si el recurso fue creado.



```
body  Cookies  Headers (7)  Test Results  Status: 200 OK  Time: 34 ms  Size: 1.77 KB  Save as Example  ...  
Pretty  Raw  Preview  Visualize  JSON  ...  
113     "duration": 332,  
114     "artist": "Taylor Swift"  
115   },  
116   },  
117   "id": 20,  
118   "title": "Anti-Hero",  
119   "duration": 201,  
120   "artist": "Taylor Swift"  
121 },  
122 {  
123   "id": 21,  
124   "title": "La Balsa",  
125   "duration": 222,  
126   "artist": "Los Gatos"  
127 },  
128 }
```

We nailed it!

