

Back End

**Carrera
Programador
full-stack**

MVC

¿Qué es MVC?

- **Model View Controller**
- Patrón de *diseño*
- Utilizado en ampliamente en la industria



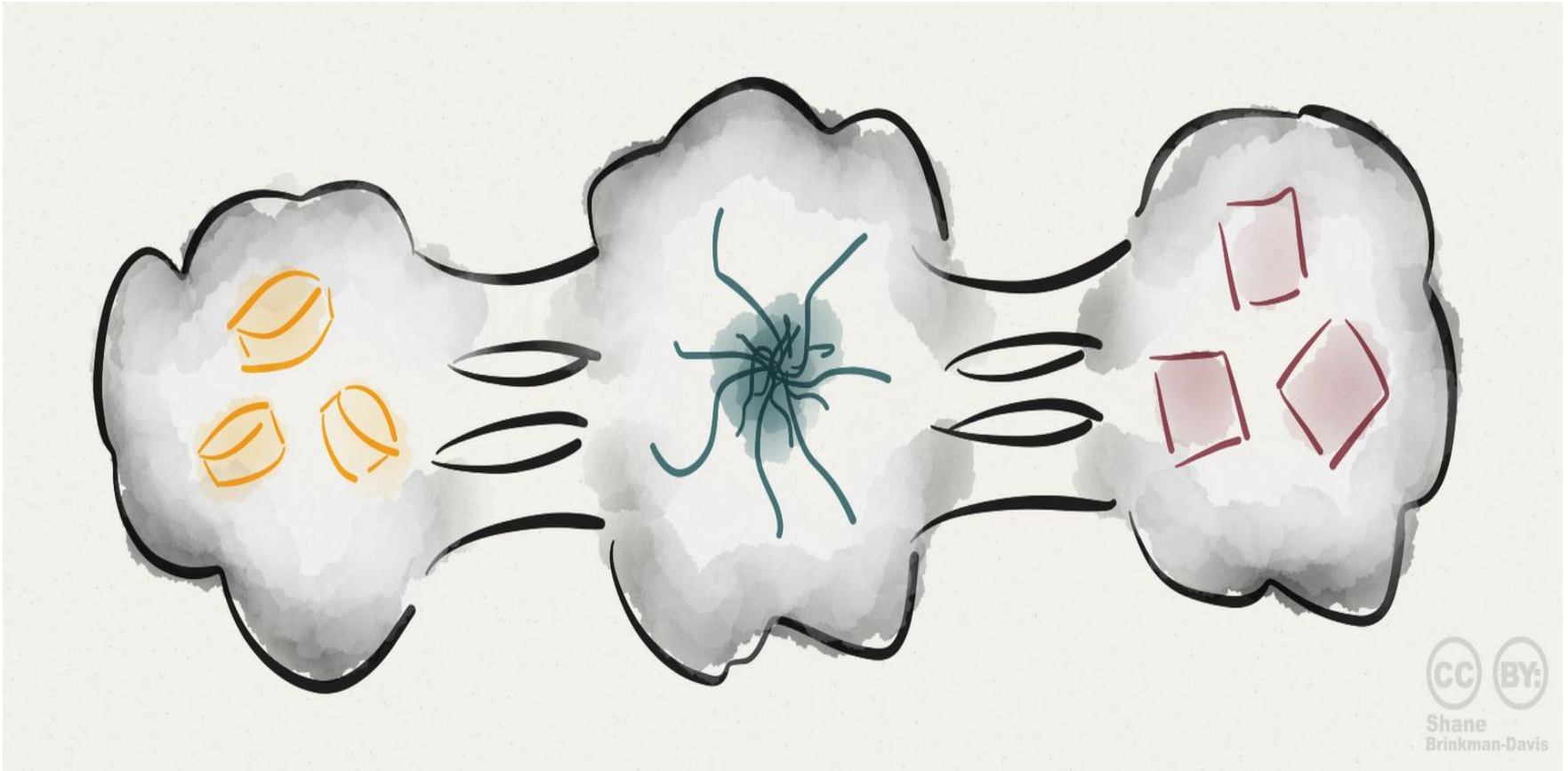
¿Qué es MVC?

- Separar responsabilidades
 - **Modelo:** Acceso a datos
 - **Vista:** Interfaz de usuario (Front End)
 - **Controlador:** Intermediario, responde a eventos

Tengo una aplicación



Separo en partes



Vista - Responsabilidades

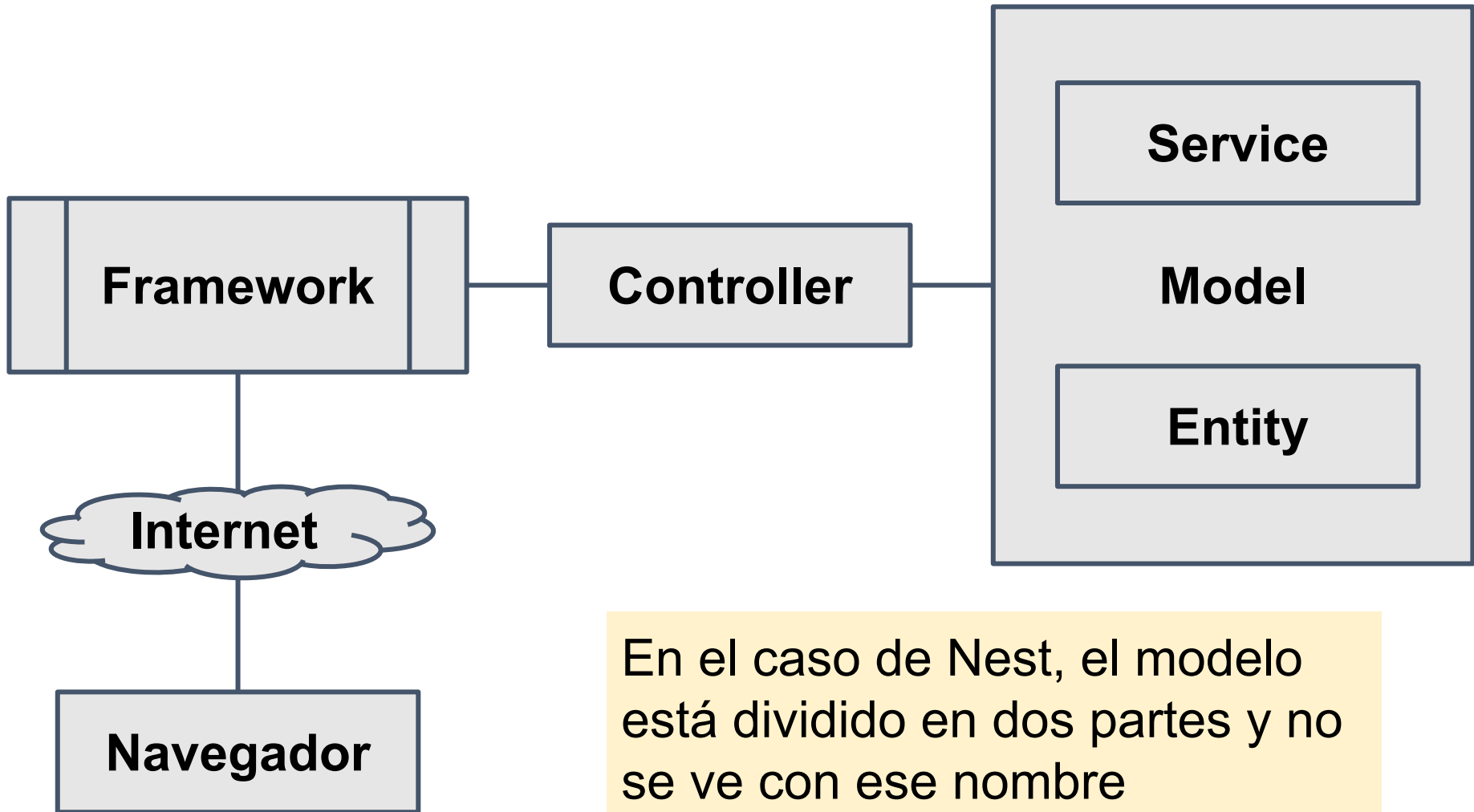
- Se encarga de la presentación los datos
- Ej: Comunicar información al usuario
 - Imágenes, fuentes, estilos, etc. (Front end)
- En el caso de una API (devuelven JSON) generalmente es código estándar que no hacemos nosotros
- O sea, si generaramos HTML va aca



Modelo - Responsabilidades

- Proteger y persistir los datos del usuario
- Asegurar la integridad y consistencia de datos
- Proveer métodos para
 - Consultar Datos
 - Modificar / Borrar Datos
- En este componente manejamos el guardado de datos y todo lo relacionado a lógica específica





En el caso de Nest, el modelo está dividido en dos partes y no se ve con ese nombre

Service

Un servicio en NestJS es responsable de almacenar y devolver datos. En general, son usados por los controladores, aunque un servicio puede llamar a otro servicio.

Es donde está implementada la lógica de negocio. En el módulo de POO se correspondería con la clase gestora.

La lógica de negocio NUNCA debe ir en el controller.

Controller

Se encarga de recibir los pedidos al backend.

Controla el ruteo, y maneja los detalles de la comunicación en sí, delegando tareas a otras partes más específicas. Por ejemplo, puede delegar controles de tipos de parámetros y de lógica de negocio.

En una aplicación hay varios controladores, y cada uno tiene varias rutas relacionadas entre sí.



Parámetros en la URL

Por ejemplo, si tenemos un endpoint como

/mostrar/2

Que devuelve "ud ingresó un 2 como parámetro"
(y sirve para /mostrar/lo-que-sea)

En el controller podemos poner

@Get('/:id')

También es posible tener más de un parámetro:

@Get('/:arg1/fijo/:arg2/:arg3/')

Inyección de dependencias

Las dependencias son servicios o objetos que una clase necesita para lograr sus objetivos. Dependency injection (DI) es una solución genérica (o patrón de diseño) a nivel de código que permite aumentar la modularidad del código.

Con DI, la clase (ej, `TrackController`) pregunta por sus dependencias al exterior (al framework) en lugar de crearlas ella misma.

Inyección de dependencias

Esto permite que en diferentes lugares el framework le pase diferentes subclases. La clase que las usa no necesita saber que tipo le pasaron específicamente.

Esto permite hacer aplicaciones más flexibles, eficientes, robustas, testeables y mantenibles.

Este patrón permite que los objetos creen relaciones entre ellos, pero que el “cableado” (“wiring up”) de las instancias sea manejado por el framework.

Inyección en NestJS

Providers: Es el nombre que se le da a las cosas inyectables. Lo hemos visto en la clase anterior:

```
import { Injectable } from '@nestjs/common';  
@Injectable()  
export class TrackService {  
  ...  
}
```

Usar una dependencia inyectada

Para usar dependencias inyectadas, declaro que las necesito (“las pido”) en el constructor.

Esto se llama “constructor-based injection”.

```
@Controller('pistas')//REEMPLAZAR CADA “pista por track”
export class PistaController {
    constructor(private pistaService: PistaService) {}

    @Get()
    public getPista(): string {
        return this.pistaService.getPista()
    }
    ...
}
```

Back End

**Carrera
Programador
full-stack**

Programar un back-end

Desarrollo Full Stack

Al programar algo full stack al surgir un error es difícil encontrar dónde viene.

Por eso es importante programar cada parte por separado, front y back end.

Es útil tener herramientas para probar cada una por separado:

- Mocks: permiten usar el front end sin el back end real
- Postman: permite hacer pedidos al a API sin usar el front end



Programar una API

- Como vamos a programar la API necesitamos algo para consumirla (llamarla/invocarla)
- Si programamos el Javascript y hay un error, no sabemos si el error está en el JS o en el Back
- Vamos a usar una herramienta para invocar al servicio web





Postman

- Permite construir y gestionar peticiones a servicios REST (POST, GET, Etc.).
- Definir la petición que a realizar.

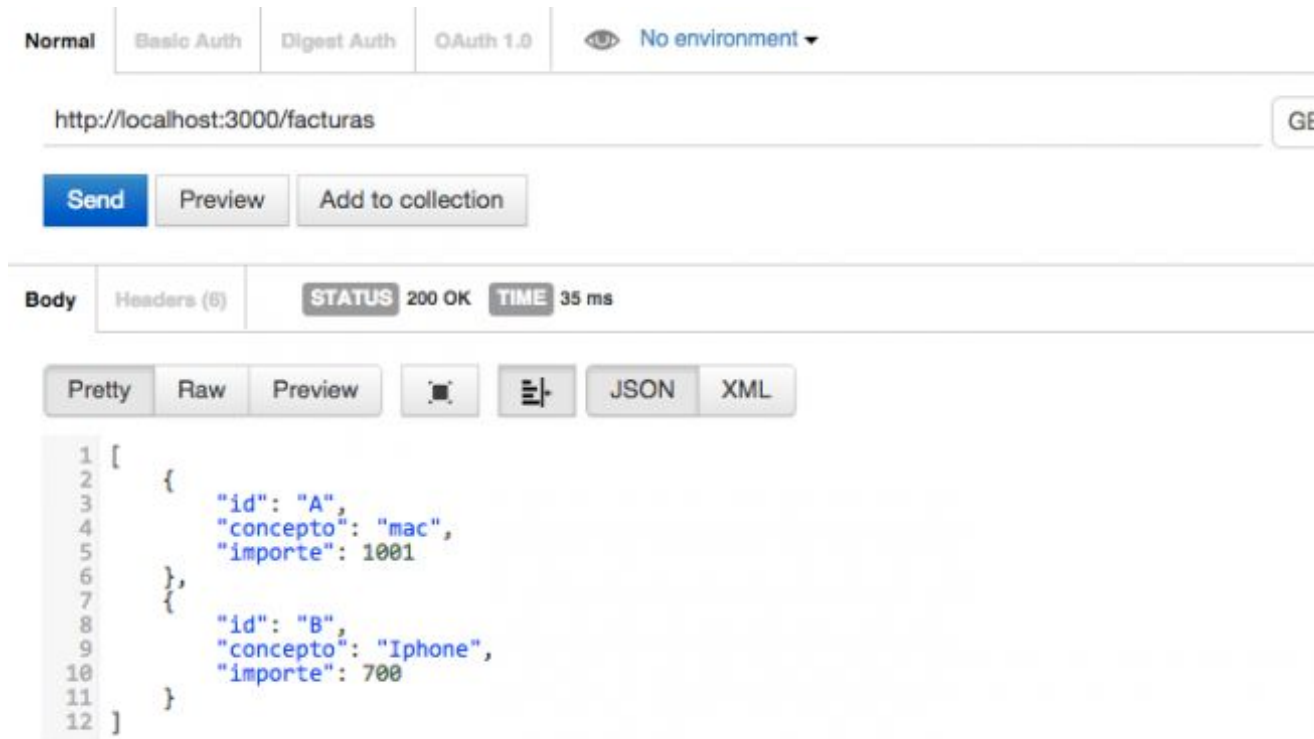


- Le damos enviar y la petición será lanzada contra nuestro servidor. Espera la respuesta (XML/JSON/Texto).
- <https://www.postman.com/downloads/>



Postman

- Captura las respuestas y muestra el resultado de una forma clara y ordenada.





Postman

- Puede realizar peticiones de tipo GET o configurar a medida peticiones de tipo POST
- Mostrará el resultado igual que en el caso anterior

The screenshot displays the Postman web interface for configuring a POST request. At the top, there are tabs for authentication: 'Normal' (selected), 'Basic Auth', 'Digest Auth', and 'OAuth 1.0'. To the right of these tabs is a dropdown menu showing 'No environment'. On the far right, there are icons for a monitor, a star, and a counter showing '0'.

The main area shows the URL 'http://localhost:3000/facturas?id=4&concepto=mac&importe=100' in the address bar. To the right of the URL is a dropdown menu set to 'POST'. Further right are two buttons: 'URL params' and 'Headers (0)'.

Below the URL bar is a table for URL parameters:

id	4	✕
concepto	mac	✕
importe	100	✕
URL Parameter Key	Value	✕
URL Parameter Key	Value	

Below the table are three tabs for the request body: 'form-data' (selected), 'x-www-form-urlencoded', and 'raw'. Under the 'form-data' tab, there is a table for key-value pairs:

Key	Value	Type
		Text

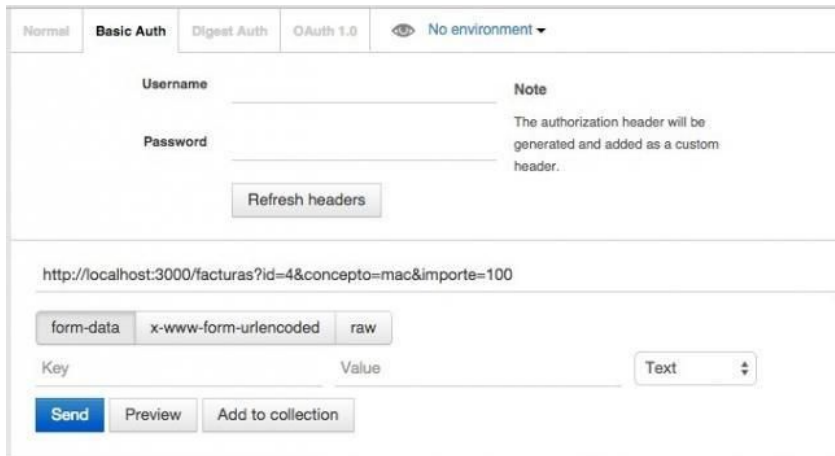
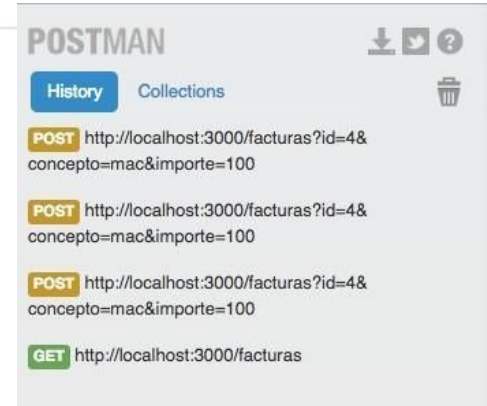
At the bottom left, there are three buttons: 'Send' (blue), 'Preview', and 'Add to collection'. At the bottom right, there is a red 'Reset' button.



Postman



- Hay disponible una barra de historial.



- Permite gestionar funcionalidades como la autenticación.

Back End

**Carrera
Programador
full-stack**

Consumir endpoints

Resultado GET Postman

localhost:3000/pistas

GET localhost:3000/pistas

Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 27 ms Size: 983 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "identificador": 1,
4     "titulo": "titulo 1",
5     "duracion": 207,
6     "interprete": "interprete 1"
7   },
8   {
9     "identificador": 2,
10    "titulo": "titulo 2",
11    "duracion": 198,
12    "interprete": "interprete 3"
13  },
14  {
15    "identificador": 3,
16    "titulo": "titulo 3",
17    "duracion": 85,
18    "interprete": "interprete 3"
19  }
```

Cookies Capture requests Bootcamp Runner Trash

Resultado GET ThunderClient

GET

Query Headers Auth Body Tests Pre Run New

Query Parameters

<input type="checkbox"/>	parameter	value
--------------------------	-----------	-------

Status: 200 OK Size: 745 Bytes Time: 33 ms

Response Headers ⁶ Cookies Results Docs

```
1  [  
2    {  
3      "identificador": 1,  
4      "titulo": "titulo 1",  
5      "duracion": 233,  
6      "interprete": "interprete 1"  
7    },  
8    {  
9      "identificador": 2,  
10     "titulo": "titulo 2",  
11     "duracion": 279,  
12     "interprete": "interprete 1"  
13   },  
14   {  
15     "identificador": 3,  
16     "titulo": "titulo 3",  
17     "duracion": 151,  
18     "interprete": "interprete 3"  
19   },  
20   {  
21     "identificador": 4,  
22     "titulo": "titulo 4",  
23     "duracion": 237,  
24     "interprete": "interprete 2"  
25   },  
26   {  
27     "identificador": 5,  
28     "titulo": "titulo 5",  
29     "duracion": 245,  
30     "interprete": "interprete 2"  
31   },  
32 ]
```

Back End

**Carrera
Programador
full-stack**

GET By id / URL Params

GET Individual

La API tiene que permitir traer una sola pista:

/tracks/:id

① README.md

TS track.controller.ts M

TS track.service.ts M X

src > track > TS track.service.ts > TrackService > getTrackById

```
1  import { Injectable } from '@nestjs/common';
2  import { Track } from '../track.interface';
3  const BASE_URL = 'http://localhost:3030/tracks/';
4  @Injectable()
5  export class TrackService {
6    async getTracks(): Promise<Track[]> {
7      const res = await fetch(BASE_URL);
8      const parsed = await res.json();
9      return parsed;
10   }
11   async getTrackById(id: number): Promise<Track> {
12     const res = await fetch(BASE_URL + id);
13     const parsed = await res.json();
14     return parsed;
15   }
16 }
```

TS track.controller.ts M X

src > track > TS track.controller.ts > TrackController > getTrackById

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


→ Home Workspaces ▾ API Network ▾ Explore Search Postman

POST add new PUT update GET get track DELETE delete track GET get all tracks GET get all previous GET get previous POST add new DELETE delete previous

CEPIT BACKEND / **get track by id**

GET ▼ http://localhost:3000/tracks/5

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

	Key	Value
	Key	Value

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON ▼ ≡

```
1 {  
2   "id": 5,  
3   "title": "I Can See You (Taylor's Version) (From The Vault)",  
4   "duration": 273,  
5   "artist": "Taylor Swift"  
6 }
```