Back End

Carrera Programador full-stack

MVC

¿Qué es MVC?

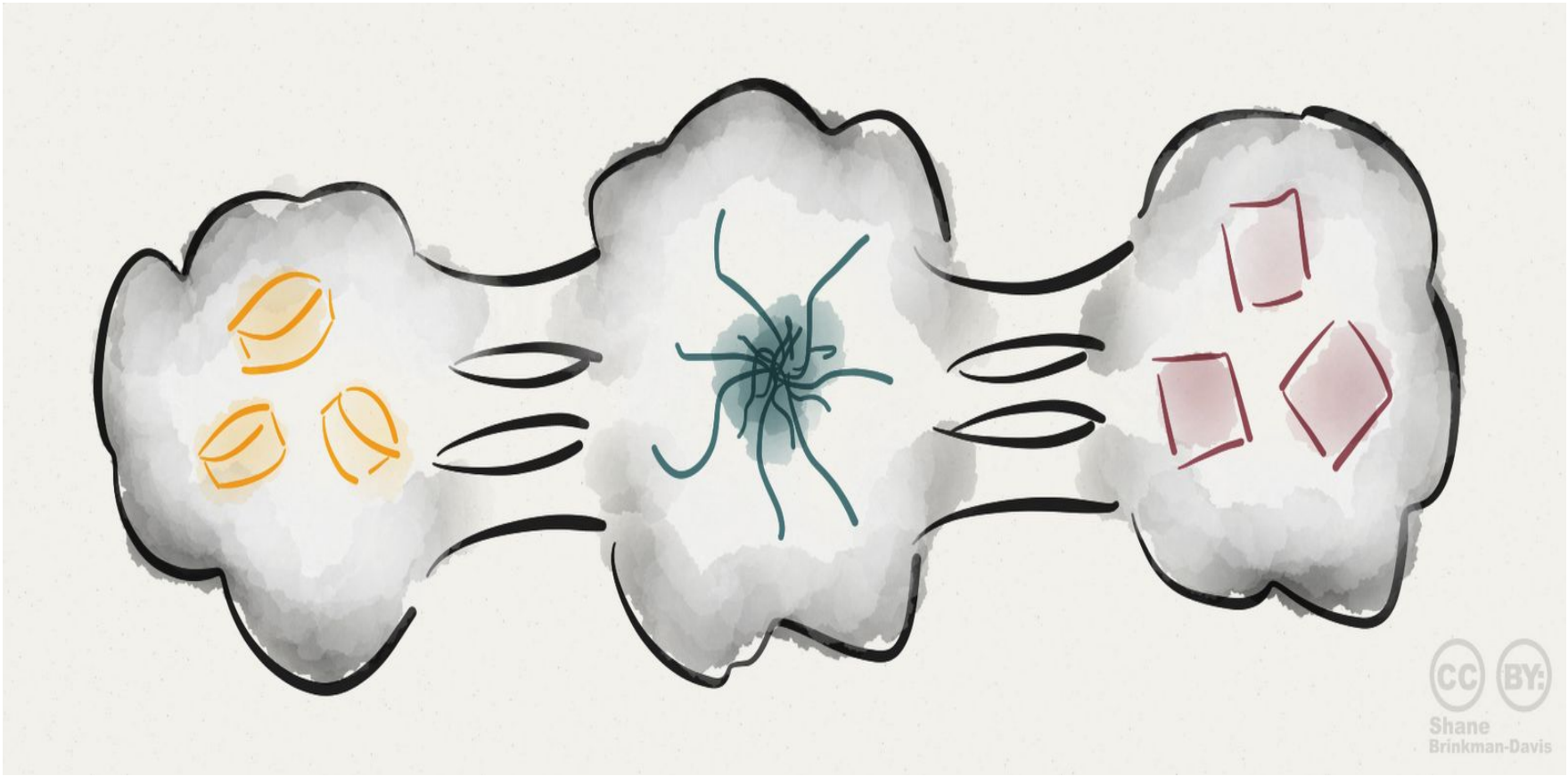
- **M**odel **V**iew **C**ontroller
- Patrón de *diseño*
- Utilizado en ampliamente en la industria

¿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

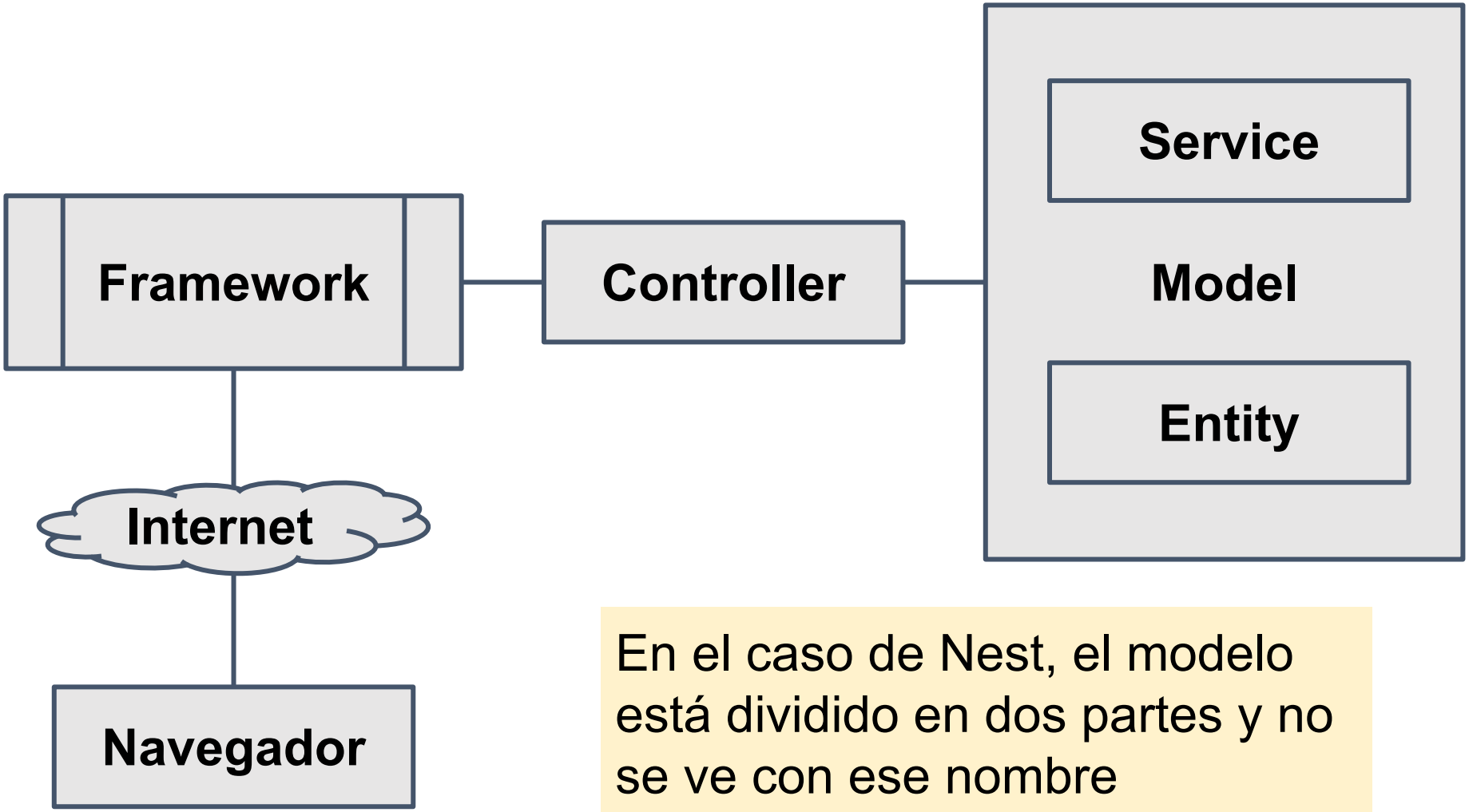


Vista - Responsabilidades

- 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

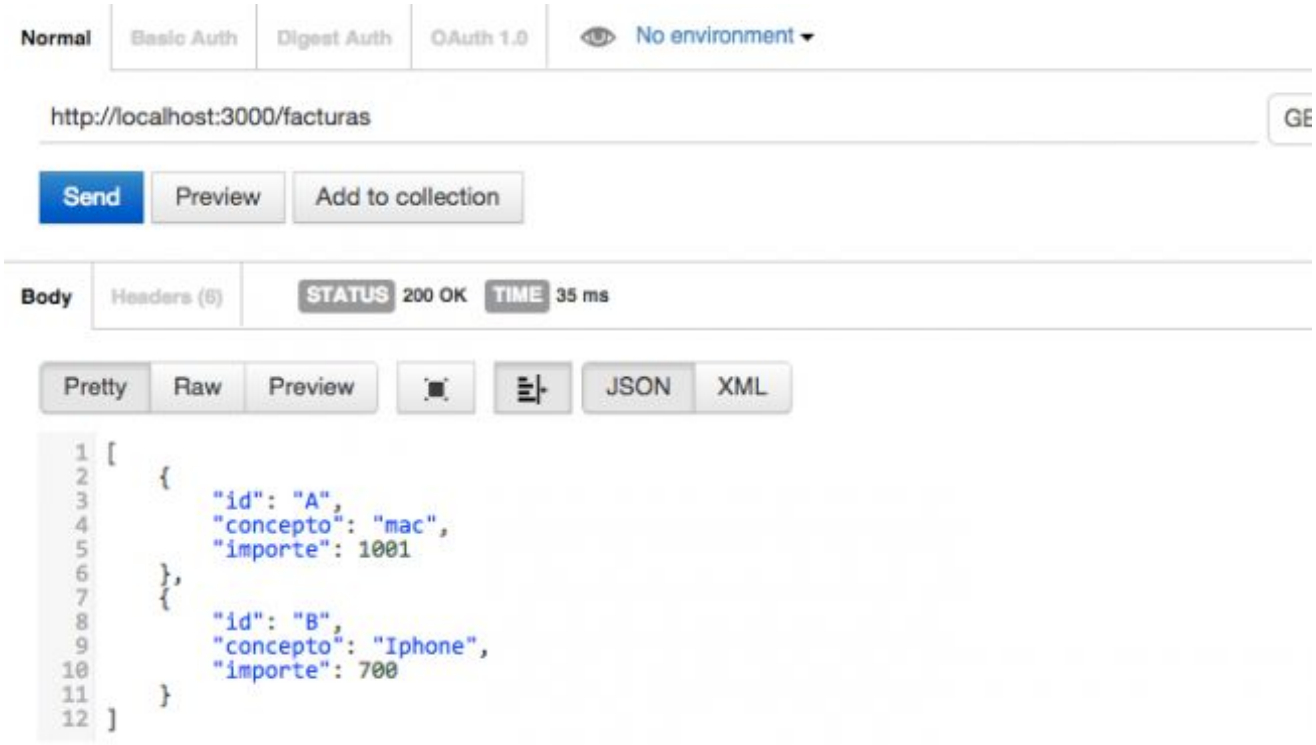
Postman

- 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/

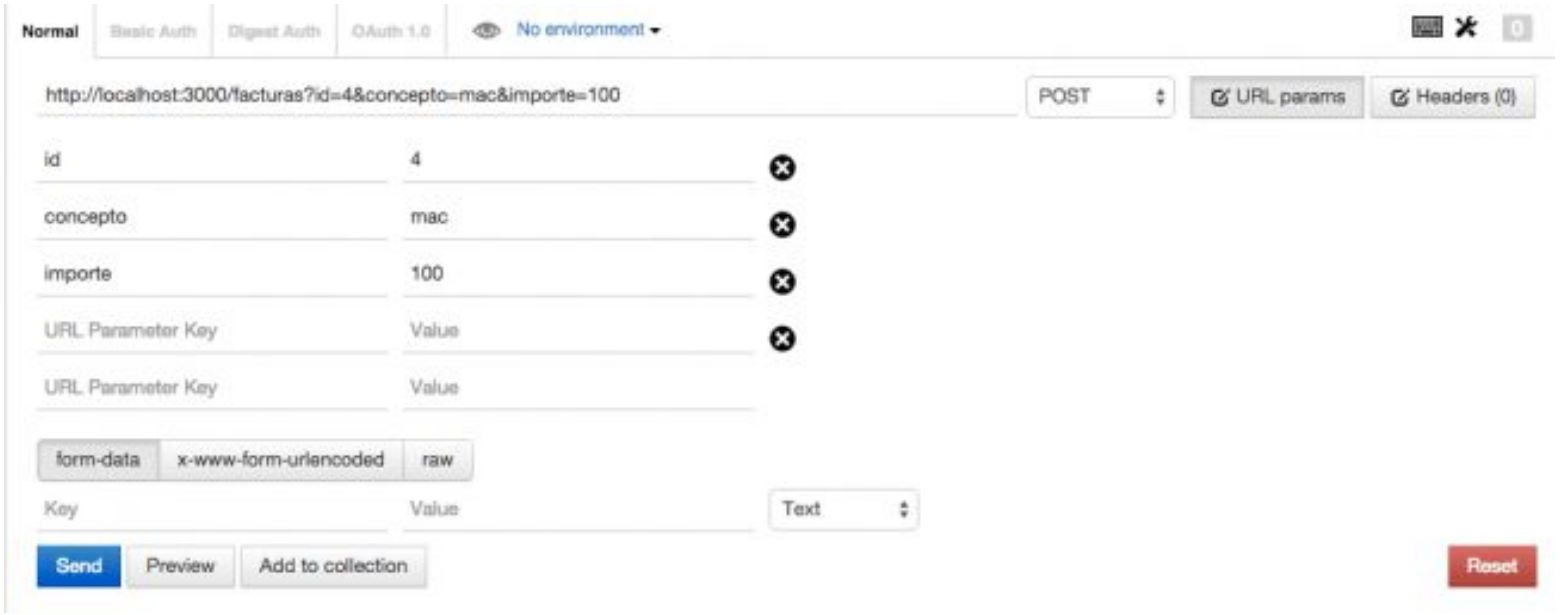
Postman

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

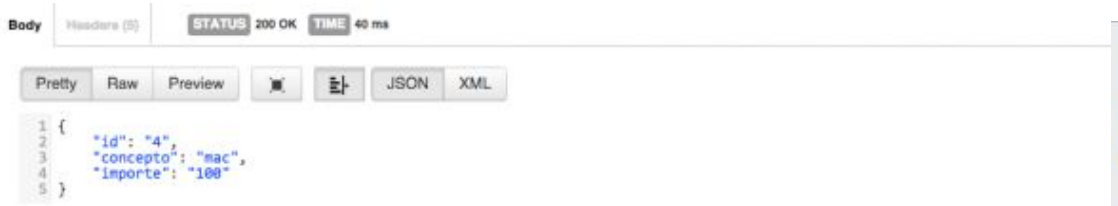


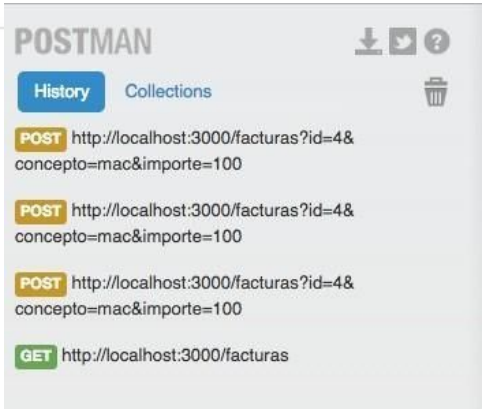
Postman

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

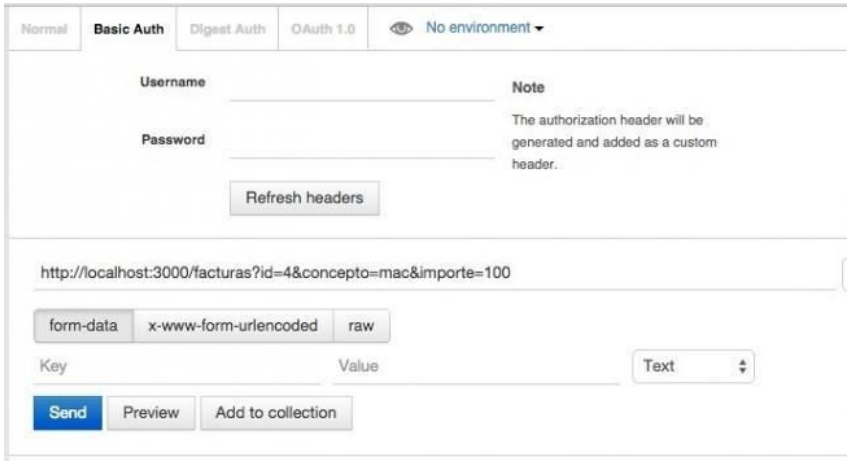


Postman





- Hay disponible una barra de historial.



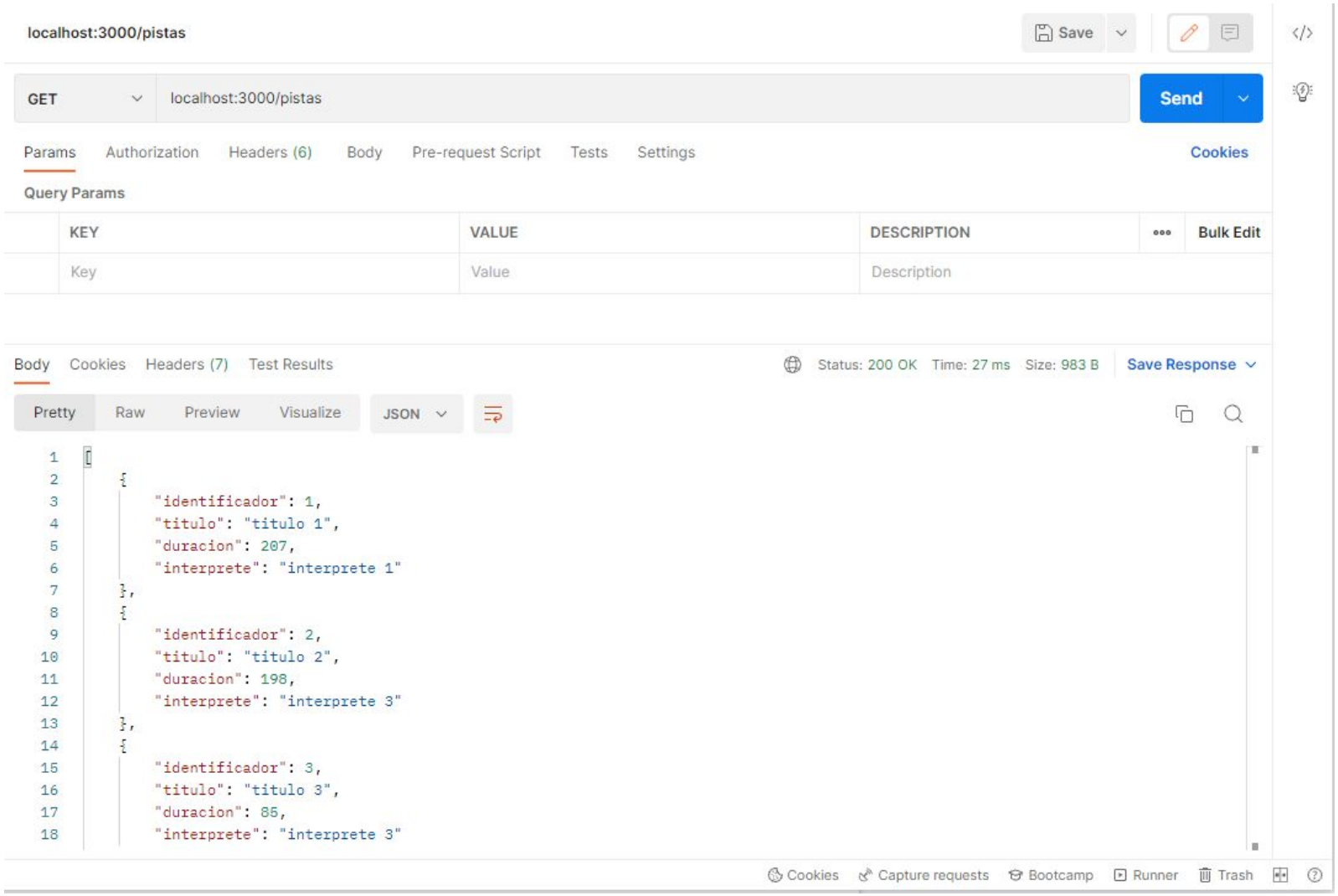
- Permite gestionar funcionalidades como la autenticación.

Back End

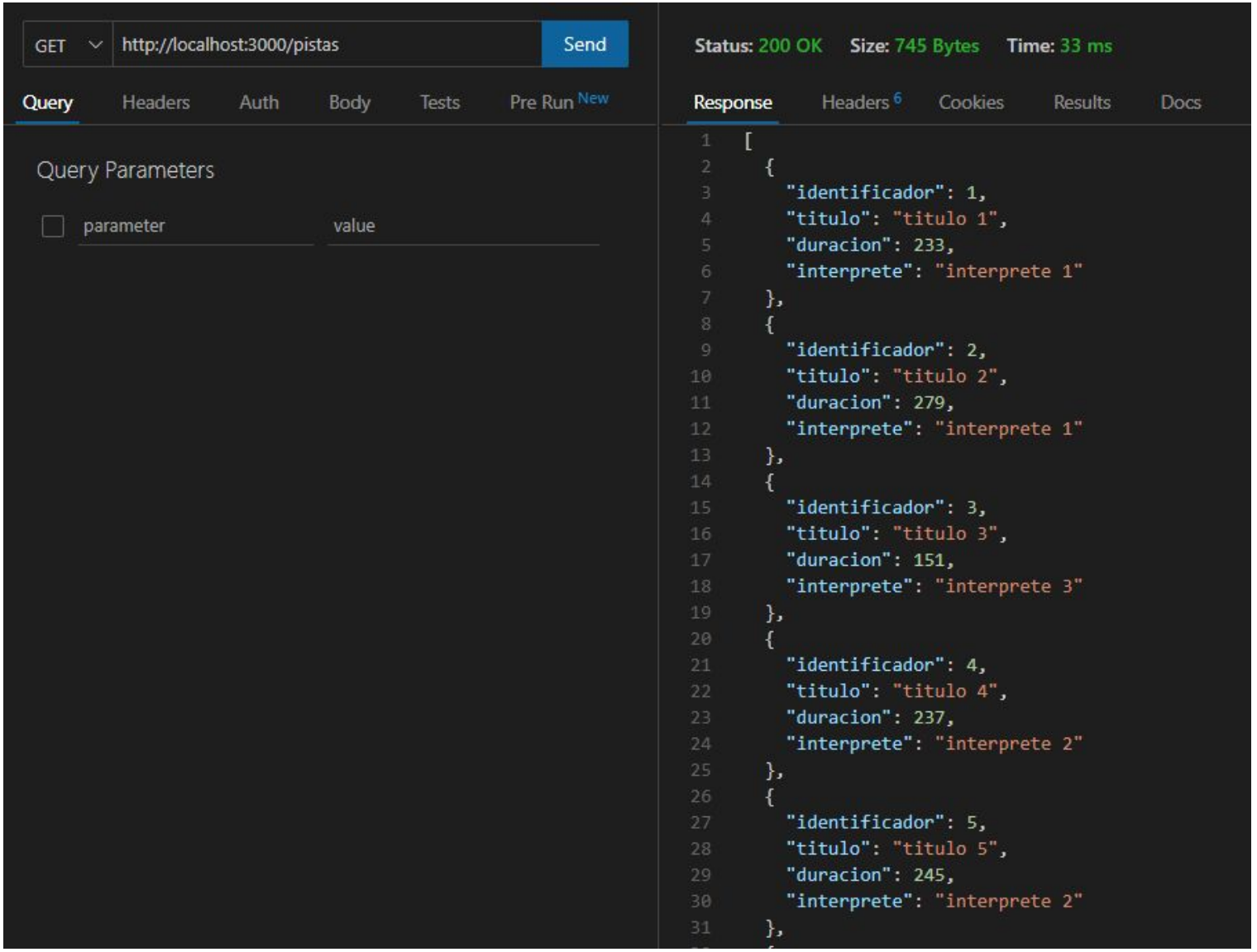
Carrera Programador full-stack

Consumir endpoints

Resultado GET Postman



Resultado GET ThunderClient



Back End

Carrera Programador full-stack

GET By id / URL Params

GET Individual

La API tiene que permitir traer una sola pista:

/tracks/:id

README.md | track.controller.ts M | track.service.ts M

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

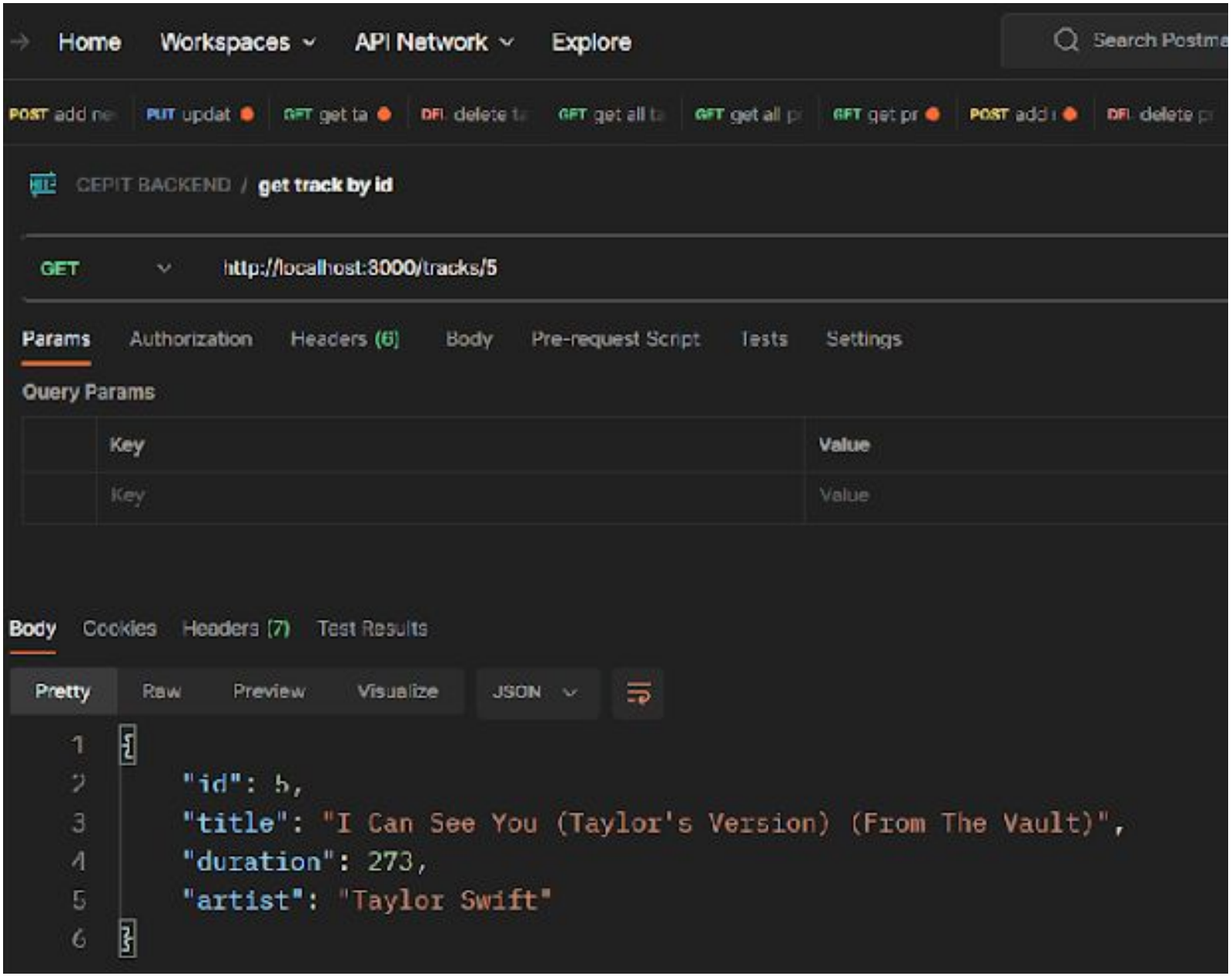
```ts
import { Injectable } from '@nestjs/common';
import { Track } from './track.interface';
const BASE_URL = 'http://localhost:3030/tracks/';
@Injectable()
export class TrackService {
  async getTracks(): Promise<Track[]> {
    const res = await fetch(BASE_URL);
    const parsed = await res.json();
    return parsed;
  }
  async getTrackById(id: number): Promise<Track> {
    const res = await fetch(BASE_URL + id);
    const parsed = await res.json();
    return parsed;
  }
}
```

track.controller.ts M

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

```ts
import { Get, Controller, Param } from '@nestjs/common';
import { TrackService } from './track.service';
import { Track } from './track.interface';

@Controller()
export class TrackController {
  constructor(private readonly trackService: TrackService) {}
  @Get('/tracks')
  getTracks(): Promise<Track[]> {
    return this.trackService.getTracks();
  }
  @Get('/tracks/:id')
  getTrackById(@Param('id') id: number): Promise<Track> {
    return this.trackService.getTrackById(id);
  }
}
```


Home
Workspaces
API Network
Explore
Search Postma
POST add ne
PUT updat
GET get ta
DEL delete t
GET get all t
GET get all p
GET get pr
POST add
DEL delete p
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 }