

Curso básico de **Nestjs**



Jonnathan Vallejo



Objetivos de la clase

- ***Identificar las características de Nestjs***
- ***Analizar la estructura de un proyecto de Nestjs***



Visión General

1º

Primeros **pasos**



Lenguaje

TypeScript por defecto, pero
también permite **vanilla**
JavaScript. La documentación
esta en TypeScript.

Prerrequisitos

Node.js (versión ≥ 12 , excepto
para v13)

Configuración



```
$ npm i -g @nestjs/cli  
$ nest new project-name
```

Modo estricto

Typescript modo estricto:

Añadir **--strict**

Estructura



Node HTTP framework

express

fastify 



```
$ npm run start
```

```
$ npm run start:dev
```

2°

Controladores



Definición

Los **Controladores** son los encargados de escuchar la solicitud y emitir una respuesta.



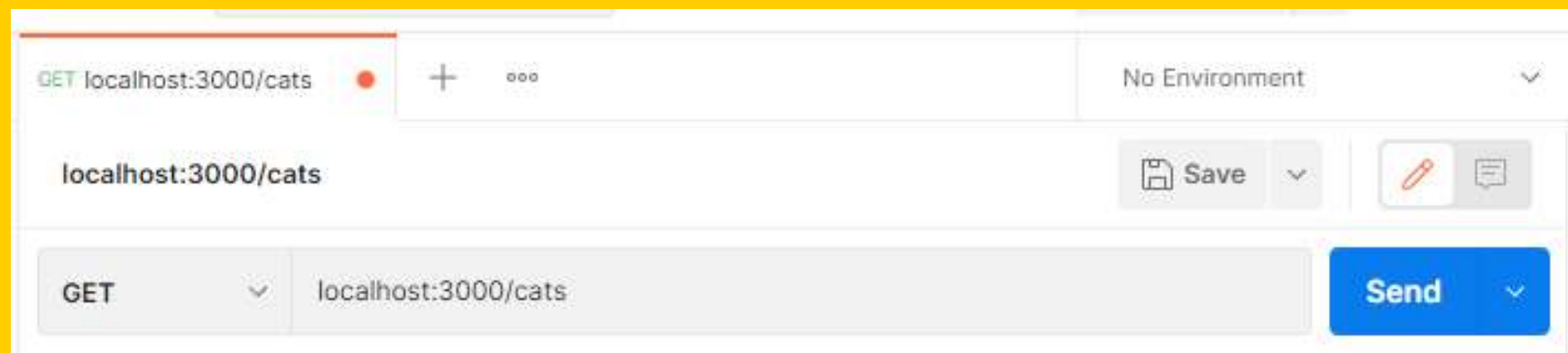
```
$ nest g controller cats
```



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

@Controller('cats')
export class CatsController {
  @Get()
  findAll(): string {
    return 'This action returns all cats';
  }
}
```



Body ▾



200 OK

17 ms

255 B

Save Response ▾

Pretty

Raw

Preview

Visualize

1 `this action return all cats`

200 OK

Standard response for successful HTTP requests. The actual response will depend on the request method used. In a GET request, the response will contain an entity corresponding to the requested resource. In a POST request the response will contain an entity describing or containing the result of the action.



Códigos de estatus **HTTP**

1. Informational responses (100 - 199)
2. Successful responses (200 - 299)
3. Redirection messages (300 - 399)
4. Client error responses (400 - 499)
5. Server error responses (500 - 599)

Códigos de estatus HTTP



200



400



500



GET localhost:3000/dogs

+

...

No Environment

localhost:3000/dogs

Save

▼

GET

▼

localhost:3000/dogs

Send

▼

Params

Auth

Headers (6)

Body

Pre-req.

Tests

Settings

Cookies

Body ▾



404 Not Found

9 ms

309 B

Save Response ▾

Pretty

Raw

Preview

Visualize

404 Not Found

The requested resource could not be found but may be available again in the future. Subsequent requests by the client are permissible.



1

2

3

4

5

"statusCode": 404,

"message": "Cannot GET /dogs",

"error": "Not Found"

Métodos HTTP

GET



POST



PATCH



DELETE





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

@Controller('cats')
export class CatsController {
  @Post()
  create(): string {
    return 'This action adds a new cat';
  }

  @Get()
  findAll(): string {
    return 'This action returns all cats';
  }
}
```

POST localhost:3000/cats

New Session

+

...

No Environment

localhost:3000/cats

Save

POST

localhost:3000/cats

Send

Route wildcard

También se admiten rutas basadas en patrones. Por ejemplo, el **asterisco** se utiliza como **comodín**



```
@Get('ab*cd')  
findAll() {  
    return 'This route uses a wildcard';  
}
```

Status Code

Código de estatus de respuesta
personalizado.




```
@Post()  
@HttpCode(204)  
create() {  
    return 'This action adds a new cat';  
}
```

Route parameters

Para definir rutas con
parámetros, podemos agregar
tokens para capturar el **valor
dinámico**



```
@Get('/:id')
findOne(@Param() params): string {
  console.log(params.id);
  return `This action returns a #${params.id} cat`;
}
```

```
@Get('/:id')
findOne(@Param('id') id: string): string {
    return `This action returns a #${id} cat`;
}
```

```
▼ GET http://localhost:3000/cats
  ► Network
  ▼ Request Headers
    User-Agent: "PostmanRuntime/7.29.2"
    Accept: "*/*"
    Postman-Token: "950fd7a5-685b-4cb4-b431-589c246793a8"
    Host: "localhost:3000"
    Accept-Encoding: "gzip, deflate, br"
    Connection: "keep-alive"
  Request Body
  ▼ Response Headers
    X-Powered-By: "Express"
    Content-Type: "text/html; charset=utf-8"
    Content-Length: "27"
    ETag: "W/"1b-J5ZQLJTTeSA1VsvEQAkJSokaycw""
    Date: "Sat, 05 Nov 2022 15:17:05 GMT"
    Connection: "keep-alive"
    Keep-Alive: "timeout=5"
  ▼ Response Body ➤
    this action return all cats
```

HTTP request line

Request

Headers

Body

Response

Headers

Body

Request payloads

Necesitamos determinar el esquema DTO (objeto de transferencia de datos). Un DTO es un objeto que define cómo se enviarán los datos a través de la red.



dto

```
// create-cat.dto.ts
export class CreateCatDto {
  name: string;
  age: number;
  breed: string;
}
```



```
@Post()  
async create(@Body() createCatDto: CreateCatDto) {  
    return 'This action adds a new cat';  
}
```




Ejercicio en clase

Indicación

Crear un método que actualice la información de un gato en específico

```
@Put update() updateCatDTO
```

```
`This action updates a #${id} cat`
```




```
@Put('/:id')
  update(@Param('id') id: string, @Body() updateCatDto: UpdateCatDto) {
    return `This action updates a #${id} cat`;
  }
```


Indicación

Crear un método que elimine la información de un gato en específico

```
@Delete remove()
```

```
`This action removes a #${id} cat`
```



```
@Delete('/:id')
remove(@Param('id') id: string) {
    return `This action removes a #${id} cat`;
}
```

3°

Providers



Definición

Son un concepto fundamental. Se puede inyectar como una dependencia. Se encarga de lógica compleja.

Servicios

Alojan la **lógica del negocio**
con el fin de reutilizarlo por
medio de **inyección de**
dependencias



```
nest g service cats
```



interfaces/cat.interface.ts

```
export interface Cat {  
  name: string;  
  age: number;  
  breed: string;  
}
```



cats.service.ts

```
import { Injectable } from '@nestjs/common';
import { Cat } from '../interfaces/cat.interface';

@Injectable()
export class CatsService {
  private readonly cats: Cat[] = [];

  create(cat: Cat) {
    this.cats.push(cat);
  }

  findAll(): Cat[] {
    return this.cats;
  }
}
```




```
import { Controller, Get, Post, Body } from '@nestjs/common';
import { CreateCatDto } from '../dto/create-cat.dto';
import { CatsService } from '../cats.service';
import { Cat } from '../interfaces/cat.interface';

@Controller('cats')
export class CatsController {
  constructor(private catsService: CatsService) {}

  @Post()
  async create(@Body() createCatDto: CreateCatDto) {
    this.catsService.create(createCatDto);
  }

  @Get()
  async findAll(): Promise<Cat[]> {
    return this.catsService.findAll();
  }
}
```

Registro de proveedores

Ahora que hemos definido un **proveedor**,
y tenemos un **consumidor** de ese
servicio, necesitamos **registrar** el
servicio con Nest para que pueda
realizar la inyección.



TS app.module.ts

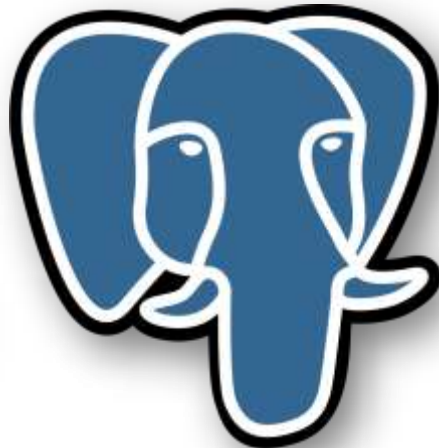
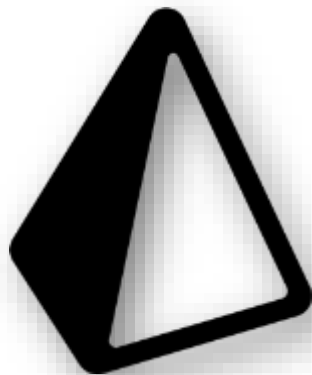
```
import { Module } from '@nestjs/common';
import { CatsController } from '../cats/cats.controller';
import { CatsService } from '../cats/cats.service';

@Module({
  controllers: [CatsController],
  providers: [CatsService],
})
export class AppModule {}
```



Introducción

Tecnologías





Generar proyecto





```
nest new hunting
```





**Crear instancia
PostgreSQL**



Railway





New Project

Deploy your app to production effortlessly

Provision PostgreSQL



Provision PostgreSQL





⚠ This is a temporary project ([what's this?](#)) and will be deleted in 24 hours. Claim it to make it yours.



Postgres



2 minutes ago via Docker Image



pgdata



Postgres

Deployments

Data

Variables

Metrics

Settings



Looking to connect this service to another? Add a [Variable Reference](#)

DATABASE_PRIVATE_URL



DATABASE_URL

postgres://postgres:waiWkAYiVZPmKxXacJIFY



PGDATA



PGDATABASE

PGHOST

PGPASSWORD



Configuración de Prisma



```
yarn add prisma --dev
```





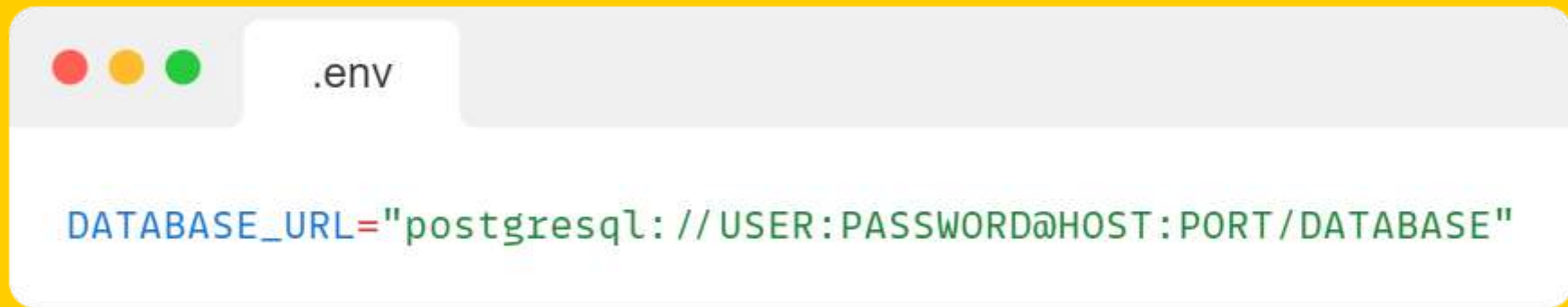
```
yarn add @prisma/client
```





```
npx prisma init
```





```
.env  
  
DATABASE_URL="postgresql://USER:PASSWORD@HOST:PORT/DATABASE"
```

Please don't commit

snappify.io



Prisma schema

Es el `lenguaje` que usa Prisma
para definir su esquema de base
de datos



prisma/schema.prisma

```
generator client {  
  provider = "prisma-client-js"  
}  
  
datasource db {  
  provider = "postgresql"  
  url      = env("DATABASE_URL")  
}
```

Generator

Indica que desea generar
Prisma Client, un generador
de **type-safe query** para su
base de datos

Data source

Especifica la conexión de base de datos. La configuración significa que su proveedor de base de datos es **PostgreSQL**

Data model

Define el modelo de base de datos. Cada **modelo equivale a una tabla** en la base de datos



prisma/schema.prisma

```
1  model Creature {
2    id          Int      @id @default(autoincrement())
3    title       String   @unique
4    description  String?
5    lastSee     String
6    countLastSee Int      @default(1)
7    extinct     Boolean  @default(false)
8    createdAt   DateTime @default(now())
9    updatedAt   DateTime @updatedAt
10 }
```



```
npx prisma migrate dev --name "init"
```



Save the migration

Prisma Migrate tomará una instantánea de su esquema y descubrirá los comandos SQL necesarios para llevar a cabo la migración.

prisma/migrations

Execute the migration

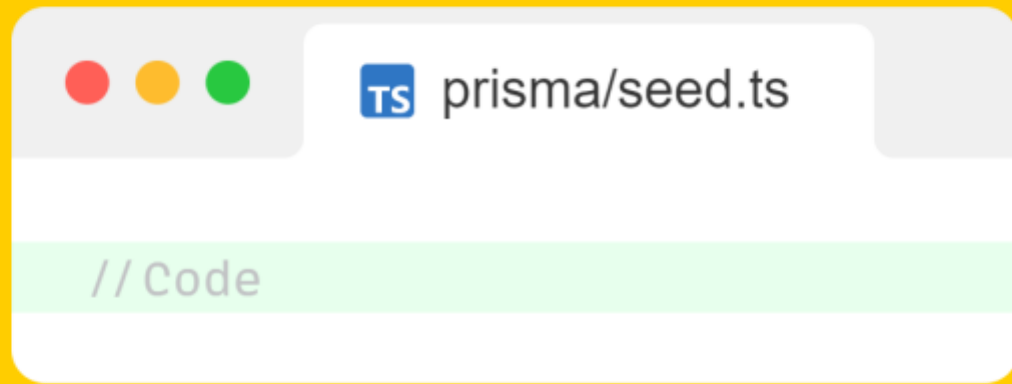
Prisma Migrate ejecutará el SQL
en el archivo de migración para
crear las tablas subyacentes en
su base de datos

Generate Prisma Client

Generará Prisma Client basado
en su último esquema.

Prisma Seed

La inicialización le permite
volver a **crear consistentemente**
los mismos datos en su base de
datos.





package.json

```
// ...  
  "scripts": {  
    // ...  
  },  
  "dependencies": {  
    // ...  
  },  
  "devDependencies": {  
    // ...  
  },  
  "jest": {  
    // ...  
  },  
  "prisma": {  
    "seed": "ts-node prisma/seed.ts"  
  }  
}
```



```
npx prisma db seed
```



```
npx nest generate module prisma
```



```
npx nest generate service prisma
```




src/prisma/prisma.service.ts

```
1 import { INestApplication, Injectable } from '@nestjs/common';
2 import { PrismaClient } from '@prisma/client';
3
4 @Injectable()
5 export class PrismaService extends PrismaClient {}
```



TS src/prisma/prisma.module.ts

```
import { Module } from '@nestjs/common';
import { PrismaService } from '../prisma.service';

@Module({
  providers: [PrismaService],
  exports: [PrismaService],
})
export class PrismaModule {}
```



Configuración de Swagger

Swagger

Documentación para la API, se
visualiza los **enpoints**
correspondientes



```
npm install --save @nestjs/swagger swagger-ui-express
```



src/main.ts

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { SwaggerModule, DocumentBuilder } from '@nestjs/swagger';

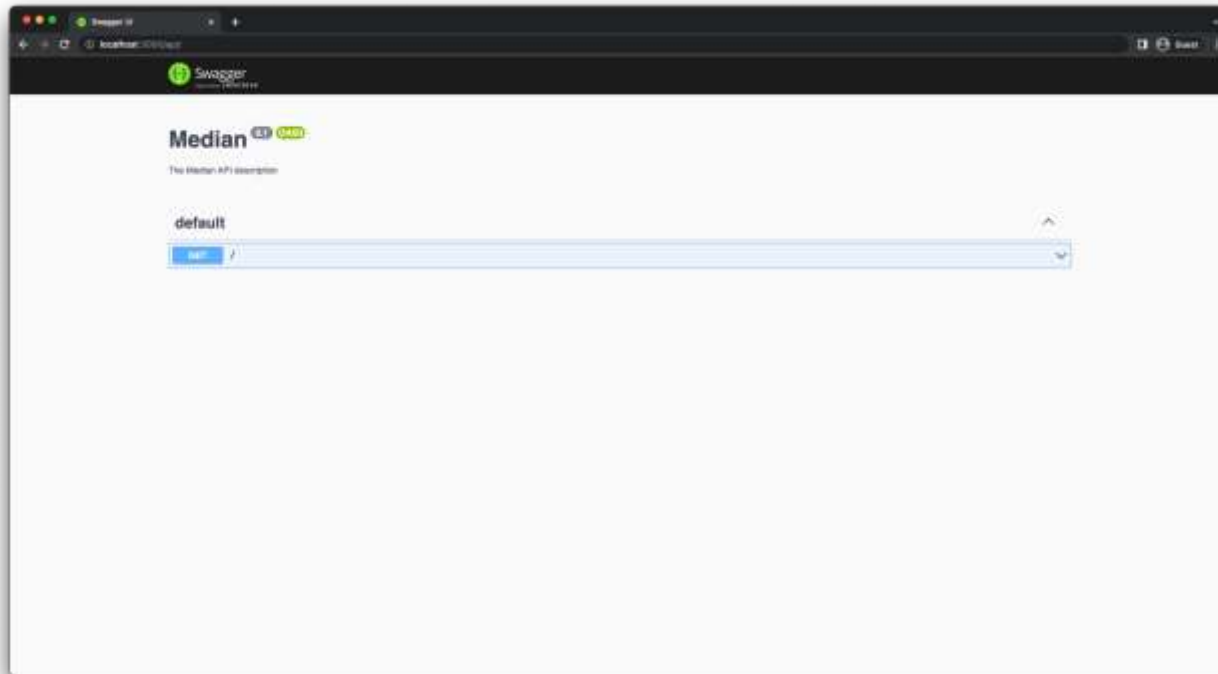
async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  const config = new DocumentBuilder()
    .setTitle('Median')
    .setDescription('The Median API description')
    .setVersion('0.1')
    .build();

  const document = SwaggerModule.createDocument(app, config);
  SwaggerModule.setup('api', app, document);

  await app.listen(3000);
}
bootstrap();
```

http://localhost:3000/api





Implementar operaciones CRUD



```
npx nest generate resource
```



Command Line Interface

What name would you like to use for this resource (plural, e.g., "users")? **creatures**

What transport layer do you use? **REST API**

Would you like to generate CRUD entry points?

Yes

Median 0.1 OAS3

The Median API description

default ^

GET	/	▼
POST	/creatures	▼
GET	/creatures	▼
GET	/creatures/{id}	▼
PATCH	/creatures/{id}	▼
DELETE	/creatures/{id}	▼

Schemas ^

CreateCreatureDto >

UpdateCreatureDto >

Añadir PrimaClient

Para dar acceso a PrimaClient
dentro de Creature module



src/creatures/creatures.module.ts

```
import { Module } from '@nestjs/common';
import { CreaturesService } from './creatures.service';
import { CreaturesController } from './creatures.controller';
import { PrismaModule } from '../prisma/prisma.module';

@Module({
  controllers: [CreaturesController],
  providers: [CreaturesService],
  imports: [PrismaModule],
})
export class CreaturesModule {}
```



src/creatures/creatures.service.ts

```
import { Injectable } from '@nestjs/common';
import { CreateCreatureDto } from '../dto/create-creature.dto';
import { UpdateCreatureDto } from '../dto/update-creature.dto';
import { PrismaService } from 'src/prisma/prisma.service';

@Injectable()
export class CreaturesService {
  constructor(private prisma: PrismaService) {}
  create(createCreatureDto: CreateCreatureDto) {
    return 'This action adds a new creature';
  }
}
```

Definir Endpoint

GET /creatures

Con el fin de obtener solo las
criaturas que no estén extintas
había que definir GET



ts src/creatures/creatures.service.ts

```
1 @Injectable()
2 export class CreaturesService {
3   constructor(private prisma: PrismaService) {}
4   create(createCreatureDto: CreateCreatureDto) {
5     return 'This action adds a new creature';
6   }
7
8   findAll() {
9     return 'This action returns all creatures';
10    return this.prisma.creature.findMany({ where: { extinct: false} });
11  }
```




Ejercicio en clase

Indicación

GET /creatures/extinct

Definir Endpoint para criaturas
extintas

Con el fin de obtener solo las
criaturas extintas hay que crear un
nuevo endpoint GET



ts src/creatures/creatures.service.ts

```
1 @Injectable()
2 export class CreaturesService {
3   constructor(private prisma: PrismaService) {}
4
5   create(createCreatureDto: CreateCreatureDto) {
6     return 'This action adds a new creature';
7   }
8
9   findExtinct() {
10    return this.prisma.creature.findMany({ where: { extinct: true } });
11  }
```



src/creatures/creatures.controller.ts

```
1 @Controller('creatures')
2 export class CreaturesController {
3     constructor(private readonly creaturesService: CreaturesService) {}
4
5     @Post()
6     create(@Body() createCreatureDto: CreateCreatureDto) {
7         return this.creaturesService.create(createCreatureDto);
8     }
9     @Get('extinct')
10    findExtinct() {
11        return this.creaturesService.findExtinct();
12    }
```

Definir Endpoint

GET /creatures/:id

Parámetro dinámico para obtener
una criatura determinada



src/creatures/creatures.controller.ts

```
1  @Get('/:id')
2  findOne(@Param('id') id: string) {
3      return this.creaturesService.findOne(+id);
4  }
```



TS src/creatures/creatures.service.ts

```
1  findOne(id: number) {  
2    return `This action returns a #${id} article`;  
3    return this.prisma.creature.findUnique({ where: { id } });  
4  }
```

Definir Endpoint

POST /creatures

Agregar nuevas criaturas



src/creatures/creatures.controller.ts

```
1  @Post()  
2  create(@Body() createCreatureDto: CreateCreatureDto) {  
3      return this.creaturesService.create(createCreatureDto);  
4  }
```



src/creatures/dto/create-creature.dto.ts

```
1 import { ApiProperty } from '@nestjs/swagger';
2
3 export class CreateCreatureDto {
4   @ApiProperty()
5   name: string;
6
7   @ApiProperty({ required: false })
8   description?: string;
9
10  @ApiProperty()
11  lastSee: string;
12
13  @ApiProperty()
14  countLastSee: number;
15
16  @ApiProperty({ required: false, default: false })
17  extinct?: boolean = false;
18 }
```



TS src/creatures/creature.service.ts

```
1 @Injectable()
2 export class CreaturesService {
3   constructor(private prisma: PrismaService) {}
4
5   create(createCreatureDto: CreateCreatureDto) {
6     return 'This action adds a new creature';
7     return this.prisma.creature.create({ data: createCreatureDto });
8   }
```

Definir Endpoint

PATCH */creatures/:id*

Actualizar criaturas por
medio de su identificador

```
1  @Patch('/:id')
2    update(
3      @Param('id') id: string,
4      @Body() updateCreatureDto: UpdateCreatureDto,
5    ) {
6      return this.creaturesService.update(+id, updateCreatureDto);
7    }
```

src/creatures/creature.service.ts

```
1  update(id: number, updateCreatureDto: UpdateCreatureDto) {  
2    return `This action updates a #${id} creature`;  
3    return this.prisma.creature.update({  
4      where: { id },  
5      data: updateCreatureDto,  
6    });  
7  }
```

Definir Endpoint

DELETE */creatures/:id*

Actualizar criaturas por
medio de su identificador



src/creatures/creatures.controller.ts

```
1  @Delete('/:id')
2  remove(@Param('id') id: string) {
3      return this.creaturesService.remove(+id);
4  }
```




src/creatures/creatures.service.ts

```
1  remove(id: number) {  
2    return `This action removes a #${id} creature`;  
3    return this.prisma.article.delete({ where: { id } });  
4  }
```

Agrupar Endpoints

Para agrupar los endpoints dentro de Swagger se añade el decorador `@ApiTags`



TS src/creatures/creatures.controller.ts

```
1  @Controller('creatures')
2  @ApiTags('creatures')
3  export class CreaturesController {
4    constructor(private readonly creaturesService: CreaturesService) {}
```