05 NEST SEED PAGINACIÓN

- En esta sección, además de crear el resource del SEED y hacer la paginación, crearemos la documentación

---

Crear módulo de SEED

- El SEED me ayudará a que si viene otro desarrollador pueda hacer las pruebas pertinentes con data en la DB
- También es útil por si empleo algo de código destructivo, poder reestablecer la DB
- El SEED es conveniente en desarrollo

```
nest g res seed --no-spec
```

- Digo de hacer un REST API con los entry points (y)
- Borro dtos ya que no los voy a usar (podría quererlos para que el SEED se cree con ciertos argumentos)
- Borro los dtos del controller, dejo solo el método GET. Lo renombro a executeSEED
- Lo mismo con el service
- Borro la entity

NOTA: Axios está dando problemas con Nest (cannot read properties of undefined). Ser recomienda instalar la versión 0.27.2 hasta que se libere una versión superior sin este problema

```
npm i axios@0.27.2
```

Realizar petición http desde Nest

- Lo que quiero es apuntar a este endpoint de pokeAPI y traerme 500 pokemons e insertarlos en la db

```
https://pokeapi.co/api/v2/pokemon?limit=500
```

- Borro todos los registros de la DB
- En la data me devuelve un nombre, una url (campo que no tengo en la db) y no tengo el número
- Solo dispongo del **fetch en versiones de node 18** o superior
- Quiero tener una forma sencilla de cambiar de fetch a axios u otra librería de peticiones http y no tener que refactorizar de arriba a abajo
- Primero se va a escribir el código tal cómo sale y luego aplicaremos el **patrón adaptador** para esto
- Creo una instancia de axios, para que sea claramente visible que estoy usando axios y no sea una dependencia oculta
- No está inyectada pero es una dependencia de mi servicio
- Puedo desestructurar la data de la respuesta. En results tengo el array de pokemons
- seed.service

```
import { Injectable } from '@nestjs/common';
import axios, { AxiosInstance } from 'axios';


@Injectable()
export class SeedService {

  private readonly axios: AxiosInstance = axios

  async executeSEED() {
    const {data} = await this.axios.get("https://pokeapi.co/api/v2/pokemon?
limit=500")

    return data;  //en data.results tengo el array de pokemons
  }
}
```

- Apunto al GET de http://localhost:3000/api/v2/seed y obtengo la data
- Pero como decía antes solo tengo el nombre y la url (que no tengo especificada en mi db)
- Quiero tener el tipado de dato de esta respuesta
- Copio la respuesta de data
- Creo la carpeta interfaces en seed/interfaces/poke-response.interface.ts
- Necesito la extension Paste JSON as Code en VSCode
- Ctrl+Shift+P , Paste JSON as code, selecciono Typescript, me pide que nombre la interfaz del nivel superior, la llamo PokeResponse, y con el resultado de la respuesta copiado en el portapapeles (Ctrl+C) le doy a Enter
- Automáticamente me saca las interfaces

```
export interface PokeResponse {
    count:    number;
    next:     string;
    previous: null;
    results:  Result[];
}

export interface Result {
    name: string;
    url:  string;
}
```

- Uso la interfaz y tipo el get cómo un genérico de respuesta tipo PokeResponse

```
import { Injectable } from '@nestjs/common';
import axios, { AxiosInstance } from 'axios';
import { PokeResponse } from './interfaces/poke-response.interface';
```

```
@Injectable()
export class SeedService {

  private readonly axios: AxiosInstance = axios

  async executeSEED() {

   const {data}= await this.axios.get<PokeResponse>
("https://pokeapi.co/api/v2/pokemon?limit=500")

    return data;
  }
}
```

- Ahora si le añado un punto a la data tengo el **autocompletado**.
- Si pongo data.results[0]. tengo el autocompletado de name y url
- Voy a manipular la data para poder insertarla en la db
- El name es fácil, pero el número está dentro de la url de cada pokemon, después de /api/v2/pokemon/número_del_pokemon/lkjj
- Desestructro el name y el url en el forEach
- Uso el **split** para separar por /. Puedo hacer un console.log de la url para ver **en qué posición** queda el número del pokemon
- El número está en el penúltimo lugar, uso **.length** -2 (-1 sería el último)
- Cómo debe ser de tipo número, uso el **+** para parsearlo a número
- seed.service

```
import { Injectable } from '@nestjs/common';
import axios, { AxiosInstance } from 'axios';
import { PokeResponse } from './interfaces/poke-response.interface';


@Injectable()
export class SeedService {

  private readonly axios: AxiosInstance = axios

  async executeSEED() {

   const {data}= await this.axios.get<PokeResponse>
("https://pokeapi.co/api/v2/pokemon?limit=10")

   data.results.forEach(({name, url})=>{
      const segments = url.split('/')
      const no: number = +segments[segments.length -2] //el número está en la
penúltima posición de la url
      console.log({name, no})
   })

    return data;
```

```
  }
}
```

- Vamos a crear un **Provider** para poder reemplazar facilmente axios, por request, o cualquier otra librería
- Luego implementaremos este patrón adaptador. Por ahora lo manejo sencillo

---

Insertar Pokemons por lote

- Para insertar el nombre y el número usaré la inyección del modelo de Pokemon y el metodo create
- Inyecto el modelo en el constructor del servicio **SeedService**

```
constructor(
    @InjectModel(Pokemon.name)
    private readonly pokemonModel: Model<Pokemon>){}
```

- PokemonModule debe de estar disponible en SeedModule
- Debo exportarlo de PokemonModule e importarlo en SeedModule
- Con exportar el MongooseModel es suficiente, no necesito el forFeature y el codigo interno porque ya lo exporta como tal
- pokemon.module

```
import { Module } from '@nestjs/common';
import { PokemonService } from './pokemon.service';
import { PokemonController } from './pokemon.controller';
import { MongooseModule } from '@nestjs/mongoose';
import { Pokemon, PokemonSchema } from './entities/pokemon.entity';


@Module({
  imports:[
    MongooseModule.forFeature([
      {
      name: Pokemon.name,
      schema: PokemonSchema
      }
    ])
  ],
  controllers: [PokemonController],
  providers: [PokemonService],
  exports:[MongooseModule] //exporto MongooseModule
})
export class PokemonModule {}
```

- Importo el PokemonModule (que está exportando el MongooseModel) en el seed.module

```
import { Module } from '@nestjs/common';
import { SeedService } from './seed.service';
import { SeedController } from './seed.controller';
import { PokemonModule } from 'src/pokemon/pokemon.module';

@Module({
  imports:[PokemonModule],
  controllers: [SeedController],
  providers: [SeedService]
})
export class SeedModule {}
```

- Debo usar el async en el foreach para poder usar el await para la inserción

```
import { Injectable } from '@nestjs/common';
import axios, { AxiosInstance } from 'axios';
import { PokeResponse } from './interfaces/poke-response.interface';
import { Model } from 'mongoose';
import { Pokemon } from 'src/pokemon/entities/pokemon.entity';
import { InjectModel } from '@nestjs/mongoose';


@Injectable()
export class SeedService {

  constructor(
    @InjectModel(Pokemon.name)
    private readonly pokemonModel: Model<Pokemon>){}

  private readonly axios: AxiosInstance = axios

  async executeSEED() {

   const {data}= await this.axios.get<PokeResponse>
("https://pokeapi.co/api/v2/pokemon?limit=10")

   data.results.forEach(async ({name, url})=>{
      const segments = url.split('/')
      const no: number = +segments[segments.length -2]

      const pokemon = await this.pokemonModel.create({name, no})
   })

    return 'Seed Executed';
  }
}
```

- Si ahora hago un GET a http://localhost:3000/api/v2/seed tengo mis 10 pokemons en la DB

- El problema es que si ahora tuviera que hacer 1000 inserciones demoraría mucho tiempo y consumiría recursos
- Se pueden hacer todas las inserciones de manera consecutiva de varias maneras más eficientes
- En la próxima lección

Insertar Múltiples registros simultáneamente

- Voy a expresar de dos maneras diferentes la inserción en la tabla
- Para evitar errores primero borro todo lo que haya en la db
- Manera 1

```
async executeSEED() {

    //borro lo que haya en la db
    await this.pokemonModel.deleteMany()

   const {data}= await this.axios.get<PokeResponse>
("https://pokeapi.co/api/v2/pokemon?limit=10")

   const insertPromisesArray = []

   data.results.forEach(async ({name, url})=>{
      const segments = url.split('/')
      const no: number = +segments[segments.length -2]

      insertPromisesArray.push(
        this.pokemonModel.create({name, no})
      )
   })

   //una vez tengo todas las promesas en el array
      await Promise.all( insertPromisesArray)

    return 'Seed Executed';
  }
```

- Manera 2. Con insertMany. Recomendada

```
import { Injectable } from '@nestjs/common';
import axios, { AxiosInstance } from 'axios';
import { PokeResponse } from './interfaces/poke-response.interface';
import { Model } from 'mongoose';
import { Pokemon } from 'src/pokemon/entities/pokemon.entity';
import { InjectModel } from '@nestjs/mongoose';


@Injectable()
export class SeedService {
```

```
  constructor(
    @InjectModel(Pokemon.name)
    private readonly pokemonModel: Model<Pokemon>){}

  private readonly axios: AxiosInstance = axios

  async executeSEED() {

    await this.pokemonModel.deleteMany({})

   const {data}= await this.axios.get<PokeResponse>
("https://pokeapi.co/api/v2/pokemon?limit=500")

   const pokemonToInsert: {name: string, no:number}[] = []

   data.results.forEach(async ({name, url})=>{
      const segments = url.split('/')
      const no: number = +segments[segments.length -2]

    pokemonToInsert.push({name, no})
   })

   await this.pokemonModel.insertMany(pokemonToInsert)

    return 'Seed Executed';
  }
}
```

- Amplio el README con el comando para usar el SEED
- README

```
<p align="center">
  <a href="http://nestjs.com/" target="blank"><img
src="https://nestjs.com/img/logo-small.svg" width="200" alt="Nest Logo" /></a>
</p>

# Ejecutar en desarrollo

1. Clonar el repositorio
2. Ejecutar
```
npm i
```

3. Tener el Nest CLI instalado
```
npm i -g @nestjs/cli
```
4. Levantar la base de datos
```
```
```

```
docker-compose up -d
```

5. Reconstruir la base de datos con la semilla

```
http://localhost:3000/api/v2/seed
```

Stack Usado

- MongoDB
- Nest

Crear un custom Provider (patrón adaptador)

- Queremos que el cambio sea lo más indoloro posible
- Voy a crear un adaptador que va **a envolver axios**, para que en lugar de tener código de terceros incrustado en mi app, tenga el mio
- Quiero sacar esa instancia de axios y crearme **mi propia implementación de una clase**
- Lo creo dentro de *common*
- Va a ser un provider (porque va a poder inyectarse)
- Los **providers** tienen que estar **definidos en el módulo**
- En common creo la carpeta interface y otra llamada adapters
- En interfaces voy a crear http-adapter.interface.ts
- La clase que implemente esta interfaz va a tener el método get que devuelve una promesa de tipo genérico
- El método get está esperando o puede recibir un genérico. **Que la respuesta es de este tipo de dato**

```
export interface HttpAdapter{
    get<T>(url: string): Promise<T>
}
```

- En adapters creo axios.adapter.ts
- Esta clase va aenvolver mi código, para que si tengo que cambiar la implementación sólo tenga que cambiar la clase
- Implemento la interfaz. La clase debe tener el método get
- Creo la instancia de axios
- Meto en un try y un catch el get y desestructuro la data. La retorno
- Lanzo un error en el catch
- Debo añadirle el operador **@Injectable** para poder inyectarlo

```
import axios, { AxiosInstance } from "axios";
import { HttpAdapter } from "../interfaces/http-adapter.interface";
import {Injectable} from '@nestjs/common'
```

```
@Injectable()
export class AxiosAdapter implements HttpAdapter{

    private axios: AxiosInstance = axios

    async get<T>(url: string): Promise<T> {
        try {
            const {data}=  await this.axios.get<T>(url)
            return data

        } catch (error) {
            throw new Error('This is an error - Check Logs')
        }
    }
}
```

- Los providers están a nivel de módulo. Para que sea visible por otros módulos **tengo que exportarlo**
- Lo hago dentro del decorador **@Module({})**

```
import { Module } from '@nestjs/common';
import { AxiosAdapter } from './adapters/axios.adapter';

@Module({
    providers:[AxiosAdapter],
    exports:[AxiosAdapter]
})
export class CommonModule {}
```

- Importo el CommonModule en el módulo de seed
- seed.module

```
import { Module } from '@nestjs/common';
import { SeedService } from './seed.service';
import { SeedController } from './seed.controller';
import { PokemonModule } from 'src/pokemon/pokemon.module';
import { CommonModule } from 'src/common/common.module';

@Module({
  imports:[PokemonModule, CommonModule],
  controllers: [SeedController],
  providers: [SeedService]
})
export class SeedModule {}
```

- Ya puedo usar este AxiosAdapter!
- Lo inyecto en el servicio de seed
- No hace falta que desestructure la data porque ya lo he hecho en el adaptador

- Si quiero la respuesta entera de axios lo guardo en una variable en lugar de desestruturarla

```
import { Injectable } from '@nestjs/common';

import { PokeResponse } from './interfaces/poke-response.interface';
import { Model } from 'mongoose';
import { Pokemon } from 'src/pokemon/entities/pokemon.entity';
import { InjectModel } from '@nestjs/mongoose';
import { AxiosAdapter } from 'src/common/adapters/axios.adapter';


@Injectable()
export class SeedService {

  constructor(
    @InjectModel(Pokemon.name)
    private readonly pokemonModel: Model<Pokemon>,
    //inyecto el adaptador
    private readonly http: AxiosAdapter
    ){}



  async executeSEED() {

   await this.pokemonModel.deleteMany({})

   //guardo la data del método get
   const data = await this.http.get<PokeResponse>
("https://pokeapi.co/api/v2/pokemon?limit=500")

   const pokemonToInsert: {name: string, no:number}[] = []

   data.results.forEach(async ({name, url})=>{
      const segments = url.split('/')
      const no: number = +segments[segments.length -2]

    pokemonToInsert.push({name, no})
   })

   await this.pokemonModel.insertMany(pokemonToInsert)

    return 'Seed Executed';
  }
}
```

- Si ahora quisiera usar otro adaptador, debería crearlo, hacer que implemente la interfaz, hacer el export correspondiente e inyectarlo

---

Paginación Pokemons

- Mediante query parameters puedo indicarle cuántos pokemons quiero por página
- Implementaremos el offset (los siguientes x) y el limit (x pokemons)
- Para que me traiga 2 pokemons de los siguientes 20

> https://pokeapi.co/api/v2/pokemon?offset=20&limit=2

- Si quiero solo 5 pokemons de los siguientes 5 lo haría asi
- pokemon.service

```
async findAll() {
    return await this.pokemonModel.find()
    .limit(5)
    .skip(5) ;
  }
```

- Entonces lo que necesito es extraer los query parameters de la url
- Vamos a necesitar un nuevo dto, para implementar algunas reglas como que tiene que ser un número, tiene que ser positivo...
- Obtengo los query parameters mediante el decorador **@Query()**
- pokemon.controller

```
  @Get()
  findAll(@Query() paginationDto: PaginationDto) {
    return this.pokemonService.findAll(paginationDto);  //le paso el Dto al
servicio para trabajar con la paginación
  }
```

- Creo el dto. Tiene más sentido crearlo en la carpeta **common** ya que es un dto muy genérico y puedo querer usarlo en otros lugares

```
import { IsOptional, IsPositive, Min, IsNumber } from "class-validator"

export class PaginationDto{

    @IsPositive()
    @IsOptional()
    @IsNumber()
    @Min(1)
    limit?: number  //le añado ? para que TypeScript lo considere opcional

    @IsPositive()
    @IsOptional()
    @IsNumber()
    offset?: number
}
```

- Ahora falta parsear el query parameter que me llega cómo un string a número y acabar la paginación

---

Transform Dtos

- Los query parameters, el body... todo va **como string**
- Puedo hacer la transformación **de manera global en el main**
- Dentro del ValidationPipe, le pongo el **transform** en true. También añado **enableImplicitConversion**

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { ValidationPipe } from '@nestjs/common';

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

  app.setGlobalPrefix('api/v2')

  app.useGlobalPipes(
    new ValidationPipe({
      whitelist: true,
      forbidNonWhitelisted: true,

      //añado transform y enableImplicitConversion
      transform: true,
      transformOptions:{
        enableImplicitConversion: true
      }
    })
  )

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

- Desarrollo la lógica en el servicio
- Desestructuro **limit y offset** que vienen en el dto y les asigno **un valor por defecto** por si no vienen
- Los agrego a los métodos

```
async findAll(paginationDto: PaginationDto) {

  const {limit=10, offset=0}= paginationDto

  return await this.pokemonModel.find()
  .limit(limit)
  .skip(offset) ;
}
```

- Para ordenarlos alfabéticamente puedo usar el método **sort**
- Le digo que ordene la columna no de manera ascendente
- Puedo hacer el select de las columnas y restarle el __v para que no lo muestre

```
async findAll(paginationDto: PaginationDto) {

    const {limit=10, offset=0}= paginationDto

    return await this.pokemonModel.find()
    .limit(limit)
    .skip(offset)
    .sort({
      no:1 //le digo que ordene la columna numero de manera ascendente
    })
    .select('__v') ;
  }
```