NEST 09

Crear módulo SEED

- Uso el codigo del CLI de NEST para generar el módulo SEED

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

- En este caso el seed no va a recibir ningún dto (a no ser que quiera que el seed reciba ciertos argumentos para que solo cree tal cosa)
- Borro tambien el entity, no voy a almacenar el seed en la base de datos
- En el controller dejo solo el Get

```
import { Controller, Get } from '@nestjs/common';
import { SeedService } from './seed.service';


@Controller('seed')
export class SeedController {
  constructor(private readonly seedService: SeedService) {}

  @Get()
  findAll() {
    return this.seedService.findAll();
  }

}
```

- En .services borro todo y creo un nuevo método que se llame executeSeed (lo coloco en el controller)

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


@Injectable()
export class SeedService {

  executeSeed(){
    return 'Seed executed'
  }
}
```

- Coloco el método en el controller

```
import { Controller, Get } from '@nestjs/common';
import { SeedService } from './seed.service';
```

```
@Controller('seed')
export class SeedController {
  constructor(private readonly seedService: SeedService) {}

  @Get()
  executeSeed() {
    return this.seedService.executeSeed();
  }

}
```

Petición http

- Voy a cargar 650 pokemon, pero mi base de datos no tiene url en su schema (lo ves en la pagina de pokeapi de cada pokemon cuando haces un llamado)
- Necesito el nombre
- Estando en otro módulo, tengo que ver como accedo al modelo de Pokemon
- Hago la petición en el servicio. Para ello usaré axios. Luego se verá que es un patrón adaptador
- Es una dependencia de mi servicio. Necesito generar una nueva instancia, no es inyectado, es una dependencia

```
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=650')

    return data
  }
}
```

- Si ahora hago la petición get en POSTMAN o ThunderClient al endpoint:

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

- Me regresa los 650 pokemon, pero solo me interesa el result de la data. Pongo en el return data.results
- Yo lo que quiero es hacer este llamado una vez y que se almacene en mi DB
- Copio la respuesta en el clipboard y creo una interface para añadirsela a la petición de axios
- Creo una carpeta llamada interfaces en /seed con el archivo poke-response.interface.ts

- Abro el archivo y uso la extensión paste JSON as code, ctrl+shift+P, INTRO, le añado el nombre PokeResponse
- Queda así:

```
// Generated by https://quicktype.io

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

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

- Me va a servir como estructura de datos para hacer un match con la respuesta
- Lo usaré con el get como genérico

```
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=650')

    return data.results
  }
}
```

- Ahora si le añado un punto al data en el return (data.) tengo la ayuda de count, next, previous y result
- Si le pongo data.results[0]. puedo acceder al name y la url del primer pokemon
- Ahora me interesa extraer el id que está al final de la url del pokemon
- Uso desestructuración en el forEach para extraerlos

```
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=650')

    data.results.forEach(({name, url})=>{

      console.log({name, url})
    })

    return data
  }
}
```

- Uso el split para dividir por el slash cada segmento

```
    data.results.forEach(({name, url})=>{

      const segments = url.split('/')
      console.log(segments)
    })
```

- Ahora se que la penúltima posición es el id como puedo observar en la respuesta de la ptición Get

```
[ 'https:', '', 'pokeapi.co', 'api', 'v2', 'pokemon', '639', '' ]
```

- Pero tiene que ser un número y no un string. Lo convierto con +

```
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=650')
```

```
    data.results.forEach(({name, url})=>{

      const segments = url.split('/')
      const no: number = +segments[segments.length - 2]
      console.log({name, no})
    })

    return data
  }
}
```

- Ahora ya tengo todo lo que necesito para hacer la inserción en la base de datos
- Más tarde se usará un patrón adaptador para la petición, para evitar dependencias de terceros

Insertar Pokemons por lote

- Se hace la inyección del modelo con @InjectModel y la entity, y llamo al model de mongoose con el genérico Pokemon

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


@Injectable()
export class SeedService {

  private readonly axios: AxiosInstance = axios

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


  ){}


  async executeSeed(){
    const {data} = await this.axios.get<PokeResponse>
('https://pokeapi.co/api/v2/pokemon?limit=650')

    data.results.forEach(({name, url})=>{

      const segments = url.split('/')
      const no: number = +segments[segments.length - 2]
```

```
      console.log({name, no})
    })

    return data
  }
}
```

- Esto me devuelve un error, porque estoy llamando el modelo de Pokemon fuera del módulo (en el seed.module)
- Exporto el módulo de Mongoose

```
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({
  controllers: [PokemonController],
  providers: [PokemonService],
  imports:[

    MongooseModule.forFeature([
      {
        name: Pokemon.name,
        schema: PokemonSchema
      }
    ])
  ],
  exports: [MongooseModule]
})
export class PokemonModule {}
```

- Importo el PokemonModule en 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({
  controllers: [SeedController],
  providers: [SeedService],
  imports:[PokemonModule]
})
export class SeedModule {}
```

- Ahora ya puedo hacer la inserción en la DB

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


@Injectable()
export class SeedService {

  private readonly axios: AxiosInstance = axios

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


  ){}


  async executeSeed(){
    const {data} = await this.axios.get<PokeResponse>
('https://pokeapi.co/api/v2/pokemon?limit=650')

    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'
  }
}
```

- Ahora puedo hacer el llamado Get desde POSTMAN o ThunderClient y los inserta en la db

Insertar multiples resgistros simultáneamente

- Lo que quiero hacer es evitar hacer un await a cada una de las inserciones
- Para ello uso el Promise.all
- Vacío la DB antes de hacer la inserción

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


@Injectable()
export class SeedService {

  private readonly axios: AxiosInstance = axios

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


  ){}


  async executeSeed(){

   await this.pokemonModel.deleteMany({})

    const {data} = await this.axios.get<PokeResponse>
('https://pokeapi.co/api/v2/pokemon?limit=650')


    const insertPromiseArray = []

    data.results.forEach(({name, url})=>{

      const segments = url.split('/')
      const no: number = +segments[segments.length - 2]

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

      await Promise.all( insertPromiseArray)

    return 'Seed executed'
  }
}
```

- Para insertarlo de una vez es más sencillo todavía con insertMany():

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

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


@Injectable()
export class SeedService {

  private readonly axios: AxiosInstance = axios

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


  ){}


  async executeSeed(){

   await this.pokemonModel.deleteMany()

    const {data} = await this.axios.get<PokeResponse>
('https://pokeapi.co/api/v2/pokemon?limit=650')


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


    data.results.forEach(({name, url})=>{

      const segments = url.split('/')
      const no: number = +segments[segments.length - 2]

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

      pokemonToInsert.push({name, no})

    })

    await  this.pokemonModel.insertMany(pokemonToInsert)

    return 'Seed executed'
  }
}
```

Crear un custom provider -opcional

- Con esta lección se busca poder cambiar axios por otra instancia, sea volvera fetch u otro paquete sin que sea un dolor de cabeza

- Voy a crear un adaptador que va a envolver axios para que en lugar de tener el código de terceros incrustado en mi app pueda tener mi propia implementación de una clase
- Lo crearé dentro de common, ya que puede ser que lo necesite para hacer peticiones en algún otro lugar
- Es un customProvider porque va a poder inyectarse
- Los providers tienen que estar definidos en el módulo
- Creo una carpeta llamada interfaces (en common) y otra carpeta llamada adapters
- En /interfaces creo un archivo http-adapter.interfaces.ts
- Voy a crear la definición de lo que necesito de una clase adaptadora tenga que implementar para que la pueda utilizar
- Toda clase que implemente httpAdapter va a tener que tener el get
- No quiero que la promesa sea de tipo any así que lo pongo de tipo genérico

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

- En adapters (esta vez se hará manualmente, se podría hacer con el CLI) creo axios.adapter.ts
- La idea es que este AxiosAdapter sea un envoltorio con mi código el cual va a yudarme que si por alguna razón debo cambiar axios solo tenga que cambiar esta clase
- Esta clase implementa la interface de http-adapter
- Debo añadir el get para que la clase de AxiosAdapter cumpla y satisfaga la interfaz

```
import { httpAdapter} from '../interfaces/http-adapter.interface'

export class AxiosAdapter implements httpAdapter {
    get<T>( url: string): Promise<T>{

    }
}
```

- Importo axios y axiosInstance para hacer la instancia de axios
- Meto la petición get de tipo genérico en un try y un catch
- Retorno la data
- Para poder inyectarlo necesito ponerle el decorador @Injectable

```
import { httpAdapter} from '../interfaces/http-adapter.interface'
import axios from 'axios'
import { AxiosInstance } from 'axios'
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(url)

            return data
        } catch (error) {
            throw new Error("This is an error. Check logs")
        }
    }
}
```

- Los providers estan a nivel de módulo. Si lo necesito en otros módulos debo exportarlo e importarlo
- En el common.module

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

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

- En el seed importo el CommonModule

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

- Ahora solo hace falta inyectarlo en el constructor de SeedService
- Cambio axios por http y la data ya no la desestructuro porque ya hice la desestructuración en el adaptador

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

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


@Injectable()
export class SeedService {

  private readonly axios: AxiosInstance = axios

  constructor(

    @InjectModel(Pokemon.name)
    private readonly pokemonModel: Model<Pokemon>,

    private readonly http: AxiosAdapter,


  ){}


  async executeSeed(){

   await this.pokemonModel.deleteMany()

    const data = await this.http.get<PokeResponse>
('https://pokeapi.co/api/v2/pokemon?limit=650') // cambio axios por http


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


    data.results.forEach(({name, url})=>{

      const segments = url.split('/')
      const no: number = +segments[segments.length - 2]

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

      pokemonToInsert.push({name, no})

    })

    await  this.pokemonModel.insertMany(pokemonToInsert)

    return 'Seed executed'
  }
}
```

Paginación de Pokemons

- Quiero implementar el offset y el limit que son query parameters (offset para empezar desde y limit: numero de pokemons)
- Todo empieza en el controller findAll()
- Voy al service e implemento la lógica

```
findAll() {
  return this.pokemonModel.find();
}
```

- Hago la petición get con POSTMAN o similares

> http://localhost:3000/api/v2/pokemon

- Ahí tengo todos los pokemons. Yo lo que quiero es que pueda especificar los argumentos y si no que me traiga solo 20 (no los 650)
- Si yo le concateno limit y skip me hacen la función de limit y offset de los query parameters
- .service:

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

- Así me salto 5 y me entrega solo 5 resultados
- Vpy a necesitar un dto para los query parameters y poder validarlos
- Mediante el decorador @Query puedo obtener los query parameters en el findAll()
- pokemon.controller

```
@Get()
findAll( @Query() queryParameters) {
  return this.pokemonService.findAll();
}
```

- Si le coloco un console.log a los queryParameters veré en consola los querys que yo le añada en la URL
- Hay que validar los parámetros. Que sea un numero, que sea positivo, etc. Para ello usaré un dto

```
@Get()
findAll( @Query() paginationDto: PaginationDto) {
  return this.pokemonService.findAll();
}
```

- Todavía no existe el PaginationDto, lo creo

- Al ser un dto que puede ser que reutilice porque es algo genérico lo colocaré en el modulo common
- Creo la carpeta /dto en common, con el archivo pagination.dto.ts
- El dto es una clase con los tipados y decoradores para hacer las validaciones respectivas de la data que entra
- Le añado el ? para que typescript sepa que es opcional

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

export class PaginationDto{
    @IsOptional()
    @IsPositive()
    @IsNumber()
    @Min(1)
    limit?: number;

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

- Importo el paginationDto en el controller para que no de error
- Ahora me da error porque los query parameters siempre llegan como string y los necesito como número

Transfrom Dtos

- En los GlobalPipes en el main, yo puedo transformar los Dtos en el tipo de dato que yo espero
- Añado el transform a true y el transform options

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

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

  app.setGlobalPrefix('api/v2')

  app.useGlobalPipes(
    new ValidationPipe({
      whitelist: true,
      forbidNonWhitelisted: true,
      transform: true,
      transformOptions:{
        enableImplicitConversion: true
      }
    })
```

```
  );

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

- En el findAll de .controller voy a mandar el dto

```
  @Get()
  findAll( @Query() paginationDto: PaginationDto) {
    return this.pokemonService.findAll(paginationDto);
  }
```

- En el .service:

```
 findAll(paginationDto: PaginationDto) {

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

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

- Puedo ordenarlos con el sort de manera ascendente
- Si no quiero que se vea el __v puedo restarlo con el select

```
  findAll(paginationDto: PaginationDto) {

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

    return this.pokemonModel.find()
      .skip(offset)
      .limit(limit)
      .sort({
        no:1
      })
      .select('-__v')
  }
```